A Devil of a Time: How Vulnerable is NTP to Malicious Timeservers?

Yarin Perry, Neta Rozen-Schiff and Michael Schapira
The Hebrew University of Jerusalem
yarin.perry@mail.huji.ac.il, neta.rozenschiff@mail.huji.ac.il, schapiram@huji.ac.il

***Abstract*—The Network Time Protocol (NTP) synchronizes time across computer systems over the Internet and plays a crucial role in guaranteeing the correctness and security of many Internet applications. Unfortunately, NTP is vulnerable to so called time shifting attacks. This has motivated proposals and standardization efforts for authenticating NTP communications and for securing NTP *clients*. We observe, however, that, even with such solutions in place, NTP remains highly exposed to attacks by malicious *timeservers*. We explore the implications for time computation of two attack strategies: (1) compromising *existing* NTP timeservers, and (2) injecting *new* timeservers into the NTP timeserver pool. We first show that by gaining control over fairly few existing timeservers, an *opportunistic* attacker can shift time at state-level or even continent-level scale. We then demonstrate that injecting new timeservers with disproportionate influence into the NTP timeserver pool is alarmingly simple, and can be leveraged for launching both large-scale *opportunistic* attacks, and strategic, *targeted* attacks. We discuss a promising approach for mitigating such attacks.**

I. Introduction

The Network Time Protocol (NTP) synchronizes computer systems across the Internet. Internet services and applications ranging from financial services to security mechanisms (TLS certificates, Kerberos, DNS and BGP security, BitCoin, and more [7], [21]–[23], [27], [43]), crucially rely on NTP for both correctness and security. Unfortunately, as highlighted by recent studies [24], NTP, designed over three decades ago, is vulnerable to many forms of attacks. Particularly disconcerting are *time-shifting attacks*, in which an attacker shifts the local time at an NTP client forward/backwards [6], [24]

Proposals for securing NTP have thus far focused on two complementary directions: (1) authenticating client-server communications through encryption [8], [9], and (2) altering the manner in which local time is computed at *the client* to minimize the impact of timeserver-provided erroneous time reports [6]. However, both approaches to NTP security are very limited in their ability to protect NTP against malicious *timeservers*. Clearly, when the timeserver with which an NTP client communicates is under the direct control of an attacker, encrypting client-server communications offers no protection. In addition, the security of client-side algorithms for syncing with timeservers is crucially dependent on the assumption that the set of timeservers to which a client can potentially synchronize does not contain "too many" timeservers that are under the attacker's (direct or indirect) control. As our results below demonstrate, this assumption can easily be violated in practice.

Our focus, in contrast to the above discussed directions, is on investigating and addressing NTP's vulnerability to strategic attacks by malicious *timeservers*. The natural starting point for our investigation is the NTP Pool Project [20], which centralizes access to thousands of timeservers across many countries and organizational domains and is used, by default, by most, if not all, major open source OS distributions (including the major Linux distributions), router vendors, home automation systems, security cameras, household appliances, and more [5], [18]. As evidenced by the many millions of systems that rely on the NTP server pool for time synchronization, the NTP Pool Project successfully facilitates accurate time synchronization at scale. However, as our results shall demonstrate, the pool's mechanisms for assigning timeservers to clients are vulnerable to hazardous attacks.

We consider two strategies for an attacker: compromising *existing* NTP timeservers and injecting *new* timeservers into the NTP server pool.

Control over fairly few existing NTP timeservers can impact time at many clients. An NTP client that uses the NTP server pool is periodically assigned timeservers to sync to by the pool. Recent proposals [6] call for significantly increasing the number of timeservers the client can potentially sync with to avoid being overly dependent on the input of any single timeserver. We observe, however, that even so, an attacker in control of fairly few servers in the pool can inflict significant harm. As explained next, the root cause for this vulnerability is that the pool's mechanism for assigning timeservers to clients is oblivious to inter-server dependencies. Local time at most NTP timeservers is derived from interaction with other (lower strata [38]) timeservers. This implies that an attacker in control of a low-stratum NTP timeserver can potentially influence time at a client indirectly by manipulating time at *other* (higher-stratum) timeservers. We show, through extensive empirical analyses, how this can be leveraged by an attacker for shifting time at country/state-scale, or even continent-scale, adversely impacting the performance or security of various applications. In particular, an attacker in control of merely 10s of timeservers in Europe or the US (out of thousands of timeservers in Europe and many hundreds in the US) can shift time forward/backwards by *hours* at many clients across the entire continent/country, impacting various applications of interest. We observe, however, that more effective, and

Network and Distributed Systems Security (NDSS) Symposium 2021
21-25 February 2021, Virtual
ISBN 1-891562-66-5
https://dx.doi.org/10.14722/ndss.2021.24302
www.ndss-symposium.org

also simpler to launch, attacks are feasible by injecting new timeservers into the NTP server pool.

Influencing time computation at clients via the injection of new timeservers is effective and simple. Entering a new timeserver into the NTP server pool is remarkably easy. Moreover, she/he entering the new timeserver, and operating that timeserver, is trusted to provide truthful information about the timeserver (e.g., its stratum [46]). We show how, through the proper configuration of parameters, an attacker that enters a timeserver into the pool can increase the number of clients its timeserver is assigned to by the pool by three orders of magnitude (compared to the default timeserver configuration). As our empirical analyses establish, this translates to hundreds of thousands of clients per hour trying to sync with that timeserver. We show how this state of affairs can be leveraged by an attacker for launching large-scale opportunistic attacks (as with taking over existing NTP timeservers) but also for launching strategic and stealthy attacks that target a *specific* NTP client or group of clients. In addition, this attack strategy is not limited in terms of the extent to which time can be shifted and can be employed to shift time at clients by days, weeks, months, and beyond, impacting a much broader range of applications.

Towards mitigating attacks by malicious timeservers. The manner in which the NTP server pool balances load across timeservers when assigning timeservers to NTP clients reflects differences across servers in terms of compute power and network capacity, and in terms of the volume of NTP queries the timeservers' owners consent to support. Thus, *any* solution for attacks on NTP by malicious timeservers must not only preserve the time accuracy and precision of today's NTP time computation but also avoid overloading NTP timeservers. We observe that promising recent proposals for "crowdsourcing" NTP queries across many servers [6], namely, the Chronos NTP client, if not applied with care, will not only fail to provide meaningful security benefits but also, if widely deployed, increase the load on a large fraction of NTP timeservers in the pool by $200x - 300x$. We propose backwards compatible changes to the NTP Pool Project's server-assignment scheme for mitigating attacks by malicious servers. We explain how Chronos can be coupled with our server assignment scheme in manner that yields significant security benefits without overloading timeservers or impacting today's NTP time accuracy and precision. Our solution is compatible with the ongoing efforts to standardize Chronos at the Internet Engineering Task Force (IETF).

Organization. We provide necessary background on NTP and the NTP server pool in Section II. We then provide a high-level overview of the two considered attack strategies in Section III. We present our empirical analyses of the two attack strategies in Sections IV and V. We discuss how better security can be attained without compromising today's NTP time accuracy and precision, or overloading timeservers, in Section VI, present related work in Section VII, and conclude in Section VIII.

Ethics statement. Some of our results rely on experiments with servers injected by us into the real NTP server pool. These involved either measurements without manipulation or targeted our own NTP clients. When not queried by our own clients, our servers, which synced with popular stratum 1 servers in their regions, and passed the NTP pool's monitor tests, provided truthful responses.

II. Background: NTP and the NTP Pool Project

A. NTP Overview

We present below a high-level overview of NTP's client-server architecture, focusing on the elements needed for the exposition of our results. We refer the reader to the Appendix and to [31], [32], [38] for a detailed exposition of NTP.

NTP clients. An NTP client periodically queries a set of timeservers. The client exchanges messages with these timeservers to learn the current clock readings at the timeservers and to estimate the network delay with respect to each timeserver. Based on the estimated delay and the reported clock readings of a time server, the client computes the *offset* with respect to that timeserver, i.e., the estimated difference in time between the client's local clock and the timeserver's local clock. We henceforth refer to the local time at a timeserver queried by an NTP client, as estimated by that client from the server's offset, as the timeserver-provided "*time sample*". To update its local time, the client feeds the time samples obtained from the timeservers into an algorithm that discards outliers and computes, from the "surviving" time samples, a time to update the local clock to. Specifically, Marzullo's algorithm [28]–[30] is applied in standard NTPv4 clients to find a majority of timeservers with accurate clocks ("truechimers" [1], [32]). See Appendix A for a more thorough exposition of the NTPv4 client.

NTP timeservers. NTP timeservers are hierarchically ordered according to *strata*. Stratum 0 devices are expected to be highly accurate (e.g., atomic clocks, or clocks directly connected to GPS antennas), and are not reachable via a network connection. Stratum 1 timeservers are timeservers that use a Stratum 0 device as a reference clock and are accessible via the Internet. Stratum 2 timeservers are timeservers with Internet connections that sync to Stratum 1 timeservers, and so on.

The NTP Pool Project. The NTP Pool Project centralizes access to thousands of volunteer-provided NTP timeservers across different countries and organizational domain. Prior to the creation of the Project's NTP timeserver pool, it was customary to *manually* configure into an NTP client a handful of timeservers with which that client can synchronize. Naturally, this entailed the risk of overloading these few servers, as well as high vulnerability to faulty or compromised timeservers. Reliance on a small set of timeservers is still common for users of products by various vendors (Apple, Microsoft, Alibaba, etc.), which synchronize with timeservers offered by these vendors by default [18], [41]. The NTP Pool Project was started following the abuse of a small number of public timeservers and is used, by default, by most, if not all, open source OS distributions (including all major Linux distributions), router vendors, home automation systems, security cameras, household appliances, and more [5], [18].

The pool utilizes DNS to assign timeservers to NTP clients based on client geolocation and also balances load across

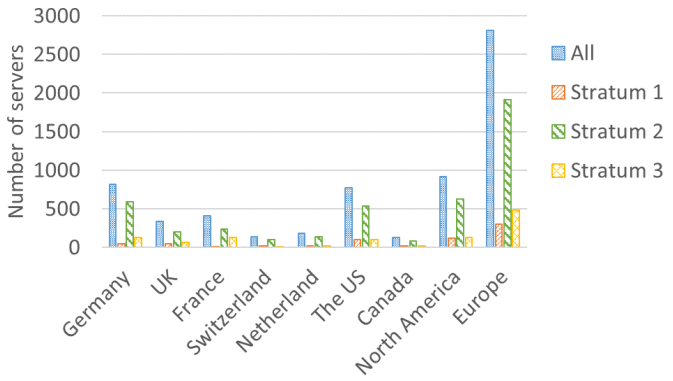


Figure 1: The number of timeservers in several countries in Europe and North America.

Zone	Pool size
Africa	46
Antarctica	0
Asia	307
Europe	2810
North America	915
Oceania	116
South America	56

Table I: The number of timeservers per zone

these servers. The server pool is divided into *zones* based on continent (e.g., europe.ntp.pool.org and asia.pool.ntp.org) and country (e.g., us.pool.ntp.org and cn.pool.ntp.org). Table I presents the number of servers in *continent-level* zones [12]. As can be seen in the table, the timeservers in Europe and North America constitute roughly 90% of all timeservers in the pool. In addition, as shown in Fig. 1, which specifies the number of servers in different countries, there is high diversity in the number of servers across countries (for instance, 30% of the timeservers in Europe are in Germany, while France and the Netherlands host about 15% and 7% of timeservers in Europe, respectively). Moreover, only around 10% of timeservers in the pool are stratum 1 servers.

New timeservers are entered into the pool via an interface that enables specifying servers' IP addresses and other configuration parameters. The best recommended practice for NTP *timeservers* is to sync with a set of $4-7$ *manually-configured* timeservers [13], which typically does not significantly change over time.

The pool monitors servers' health by periodically (every roughly 12 minutes [20]) querying servers for time reports from a pool-controlled *monitor server* and removing servers from the pool if these are unresponsive, or if their time reports deviate from the monitor's local time by "too much".

B. Time-Shifting Attacks

As highlighted by recent studies [24], [26], NTP clients are highly vulnerable to *time-shifting attacks*, in which an attacker shifts the local time at the client forward/backwards.

Recall that the local time at a client is determined based on the clock readings received from the timeservers the client interacts with and the delay with respect to these timeservers, as estimated by the client. By reporting false clock readings in NTP messages, or affecting the experienced delay between the client and the timeservers, an attacker can induce wrong decisions at the NTP client. In particular, if the attacker has sufficient presence in the set of timeservers with which an NTP client communicates, it can stealthily shift time at the client forward/backwards by repeatedly pushing the client further away from the actual time when queried by the client. For instance, in ntpd v4.2.8p15 (released in June 2020), every 5 minutes, a malicious server can shift time at a client by 16 minutes, and so shifting the client's local time by x (seconds/minutes/hours/days/months/years) requires roughly $\frac{x}{3}$ time.

To accomplish this, the attacker must have sufficient presence in the set of timeservers the client syncs with. Such attacks can be launched, e.g., by a man-in-the-middle attacker capable of intercepting and tampering with NTP messages between the client and (a significant subset of the) timeservers, or by an attacker in direct control of (a subset of) the NTP timeservers themselves.

To attack...	Change Time by...	To attack...	change time by...
TLS Certs	years	Routing (RPKI)	days
HSTS (see [44])	a year	Bitcoin (see [4])	hours
DNSSEC	months	API authentication	minutes
DNS Caches	days	Kerberos	minutes

Table II: Impact of timeshifting attacks on different applications (taken from [24])

As shown in [24], time shifting attacks on NTP can serve as building blocks for compromising many applications/services of interest. Table II, taken from [24], presents several such applications and the extent to which time at an NTP client should be shifted needed to harm them.

C. NTP Security

To combat time-shifting (and other) attacks, NTP practitioners and researchers have investigated two main approaches:

Authenticating NTP communications. While NTP supports cryptographic authentication [7], [36], in practice NTP traffic is very rarely authenticated for various reasons [11], [34], [40]. More importantly (1) even if NTP traffic is encrypted, an

attacker capable of delaying/dropping traffic can still influence time at the NTP client, and (2) encryption clearly does not defend against an attacker in control of the NTP timeservers themselves.

Client-side solutions: the Chronos NTP client. The recently introduced Chronos NTP client [6], which is currently being promoted at the IETF [42], reflects a different approach to NTP security. Chronos distributes time queries across a large number of NTP timeservers and employs a theory-informed approximate agreement algorithm to discard outlying responses and to update the local clock. Specifically, in Chronos, a set of servers consisting of hundreds of timeservers is assigned to a client and the IP addresses of these timeservers are stored at the client. The client periodically queries a small subset (say $10 - 15$) of these servers, chosen uniformly at random. By removing from consideration the lowest and highest time samples gathered from the queried servers, and setting the local time to be the average of the surviving time samples, Chronos provably attains high time accuracy so long as the attacker cannot influence time at "too many" of the servers assigned to the client. We describe Chronos in more detail in Appendix B.

Intuitively (and as formalized and proven in [6]), by relying on many timeservers for synchronization and employing a secure methodology for computing local time from server-reported time samples, Chronos sets a higher bar for the attacker, forcing it to compromise a large fraction of the timeservers to successfully shift time at the client. However, as our results below shall demonstrate, even if relying on many timeservers for synchronization, unless these servers are carefully chosen, the attacker can gain (direct or indirect) control of a large fraction of these servers, nullifying the security benefits of such client-side solutions. We discuss our scheme for secure assignment of timeservers to clients in Section VI.

III. Two Attack Strategies

We next present a high-level overview of the two attack strategies considered: (1) taking control of existing timeservers and (2) injecting new timeservers into the pool. We present empirical analyses of the described attacks in Sections III-A and III-B, respectively.

A. Attack I: Utilizing Existing Timeservers

Recall (see Section II) that NTP timeservers in different strata synchronize with timeservers in lower strata. This implies that an attacker in control of an NTP timeserver might potentially be able not only to influence time at a client directly by misreporting that timeserver's clock readings, but also to influence time at the client *indirectly* by shifting time at *other* timeservers (in higher strata) the client queries. We next explain how such attacks can be launched.

Goals of the attack. We consider an *opportunistic* attacker whose goal is to shift time at many clients in a certain geographical region R (country or even continent) so as to harm the performance or security of a certain Internet application/service. The term "opportunistic" here is used to indicate that the attacker does not target specific clients, but rather means to wreak havoc at scale. (We will later explain how targeted attacks can be facilitated by injecting new servers into the pool.)

Threat model. The attacker is in control of a subset A of the pool's timeservers in region R. This encompasses a variety of scenarios, including the following: (1) the attacker is an organization that legitimately hosts NTP timeservers with which other servers synchronize, (2) the attacker is capable of compromising a subset of the servers (e.g., by exploiting software vulnerabilities), and (3) the attacker can attract traffic from NTP clients destined for subset A via off-path attacks on DNS or BGP (like DNS cache poisoning or IP prefix hijacking), taking advantage of the fact that NTP communications are not authenticated to masquerade as the timeservers in A.

The attack. By leveraging its control over timeserver subset A, the attacker can, from some point in time onwards, respond with inaccurate times to all queries issued to its servers by higher strata timeservers, for the purpose of shifting time at these servers and so, indirectly, at all clients in the region that sync with them. We next discuss some important specifics.

- **The attacker-controlled servers must be highly influential.** Clearly, if the attacker has direct control over all timeservers in region R (that is, the set A consists of all timeservers in R), or even over a large fraction of these, it can succeed in shifting time at many clients across that region. Our objective, however, is to demonstrate that gaining control of a fairly small subset of timeservers in the region is sufficient. However, this requires the attacker-controlled timeservers to be highly influential in the sense that "many" higher strata servers crucially rely on the attacker-controlled servers for time synchronization. As we shall show in Section IV, small yet influential subsets of timeservers indeed exist in many regions. In particular, control of merely 10s of timeservers in Europe or the US (out of thousands, and many hundreds, respectively) is sufficient for shifting time at continent/country scale. Moreover, the methodology employed in our empirical analyses can be leveraged by the attacker to identify such subsets of timeservers.
- **How to shift time at higher strata servers?** Higher-strata servers that synchronize with attacker-controlled servers do so by executing NTP's client-side protocol for synchronizing with timeservers (see Section II). Hence, timeshifting attacks such as those discussed in Section II-B can be executed by the attacker-controlled servers to influence time at other servers.
- **The pool's monitor limits the harm that can be inflicted by attacker.** Recall that the pool's monitor periodically queries timeservers in the pool and, if the gap between the responses of some timeserver and its local time exceeds a certain threshold, the monitor will remove that server from the pool. Timeservers under the attacker's direct control can evade being spotted by the monitor simply by responding with correct times when queried by the monitor, while continuing to report false times when queried by others. However, this is not so for higher-strata timeservers whose local times the attacker indirectly influences;

these will respond with erroneous times when query by all, including the monitor. Thus, after some time (around 12 minutes or less [20], attacker-influenced timeservers will be removed from the pool. However, our empirical analyses show that many clients that synchronize with some server will continue to do so for 10s of minutes, and even hours, after that server's removal from the pool. This enables the attacker to indirectly shift time at these clients by continuing to shift time at a timeserver *after its removal*. While this is sufficient for attacking *some* applications/mechanisms of interest (e.g., BitCoin, API authentication, and Kerberos), other applications (e.g., TLS certs, HSTS, DNS caches, RPKI) [24], which require shifting time at clients by days, weeks, or more, seem impossible to attack using this strategy.

Ascertaining the feasibility of the attack. To demonstrate the feasibility of this attack for the current implementations of NTP client [2] and server software [20], as well as the current pool monitor [20] implementation, we tested the attack on local installations of NTP servers, the pool, and the monitor. We set up four VMs, emulating an attacker-controlled server S_A, an (honest) higher-stratum server S_H that syncs with it, the pool P, and the pool monitor M. We then executed an attack by S_A on S_H in which S_A incrementally increases time at S_H by 16 minutes every 5 minutes, using the attack technique presented in [24]. When queried by M, S_A was configured to respond with its actual local time (so as to evade being detected by the monitor). We verified that this resulted, as expected, in S_A succeeding to shift time at S_H, and in S_H being removed from the pool P by M (and S_A not being removed). We repeated the same experiment, only this time both S_A and S_H were registered into the actual NTP server pool, to verify that the behaviors of the actual pool and monitor (in terms of server removal) are as expected.

B. Attack II: Injecting New Timeservers

Entering a new timeserver into the timeserver pool is easy; the attacker need only register as a timeserver and provide an IP address and an e-mail address. The legitimacy of a registered timeserver depends only on its availability and time accuracy, which are monitored by the pool and are required to be above very easy to pass thresholds [14], [20]. Indeed, in our experiments, we were able to successfully register 10s of new timeservers into the pool at various regions. In addition, she/he entering the timeserver into the pool, and operating that timeserver, is trusted to provide truthful information regarding the server stratum, and so an attacker can always specify the server stratum for its injected timeservers as 1. Somewhat surprisingly, however, as we shall show in Section V, we find that this type of "lie" does not really aid the attacker. We identify, however, a much more effective strategy for the attacker: manipulating the *netspeed* parameter.

The netspeed parameter. The specified "netspeed" of a timeserver in the NTP pool is correlated with the volume of NTP clients directed to the timeserver by the pool, with higher netspeed yielding higher probability that the timeserver be assigned to a client by the pool. As shall be discussed in Section V, by setting the netspeed for its registered server to be the maximum possible value, the attacker can increase the chances that its injected server is assigned to any client by orders of magnitude. This can enable the attacker to attain critical mass in some regions with relatively few injected servers.

Goals of the attack. We consider both opportunistic attackers, who aim to shift time at many clients, as in Section IV, and strategic attackers that wish to shift time at specific clients.

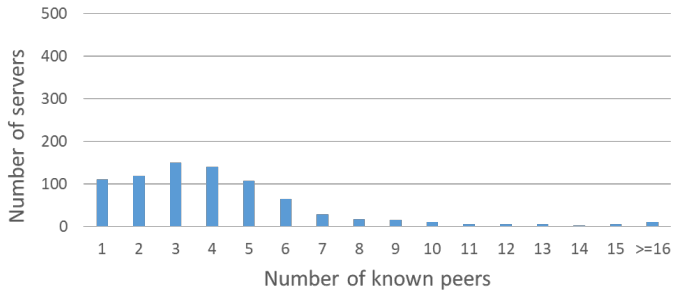
Threat model. As in Section III-A, the attacker is in direct control of a subset A of the pool's timeservers in region R, which were all injected into the pool by the attacker. Note that, unlike the attack in Section V, the attacker does not attempt to shift time at other *timeservers*. Instead, the attacker's mission is to shift local time at clients by directly interacting with these clients.

The attack. The attacker first creates new timeservers and registers these in the pool. Since the pool geolocates timeservers to assign them to specific zones, to impact time at a client located in a certain zone, the attacker's injected timeservers must be in that region. This is simple to do by leveraging public clouds. In our experiments, for instance, Amazon AWS was used to set up servers at multiple locations and register these into the pool. When registering a timeserver, the attacker configures the netspeed to the maximum value. Importantly, there is no limit on the number of timeservers the attacker can inject into the pool and so the attacker can arbitrarily increase the probability that servers are assigned to a client in a certain region by simply adding more servers in that region to the pool. To determine how many servers should be added to the pool, the attacker can employ our methodology in Section V. Since the attacker is in direct control of the timeservers, it can always report accurate times to the monitor (to prevent the removal of its timeservers from the pool). In addition, it can report accurate times to some clients and inaccurate times to others, facilitating targeted attacks against specific clients. To stealthily shift time at a client, the attacker can employ the attack technique from [6], [24] to repeatedly increase/decrease time by 16 minutes. Observe that since the monitor cannot detect misdeeds by the attacker's servers, the attack can persist for as long as needed. Thus, unlike the attack in Section III-A, this class of attacks enables arbitrary time shifts and so impacts many more applications/mechanisms [24].

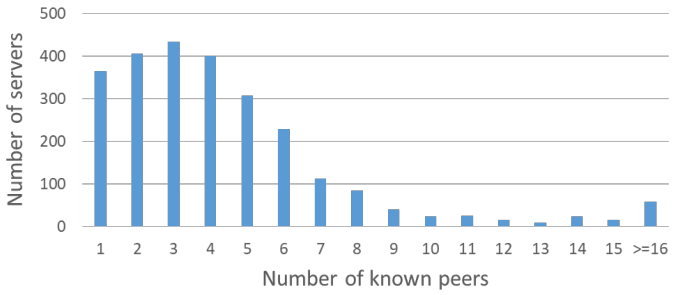
Ascertaining the feasibility of the attack. We report in Section V on our experimentation with injecting new servers into the actual server pool and interacting with real NTP clients to which our injected timeservers have been assigned by the pool.

IV. Empirical Analysis of Attacks Exploiting Inter-Server Dependencies

We empirically quantify to what extent an attacker that gains control of existing NTP timeservers can influence time in its region. However, as discussed in Section III, the answer to this question is dependent on two important factors: (1) the extent to which the attacker-controlled timeservers can impact time at *other* timeservers in the region, and (2) the implications of attacker-influenced timeservers being detected and removed from the pool. We next address each of these in turn. We will show that an attacker already in control of fairly few timeservers, or capable of gaining control of fairly few

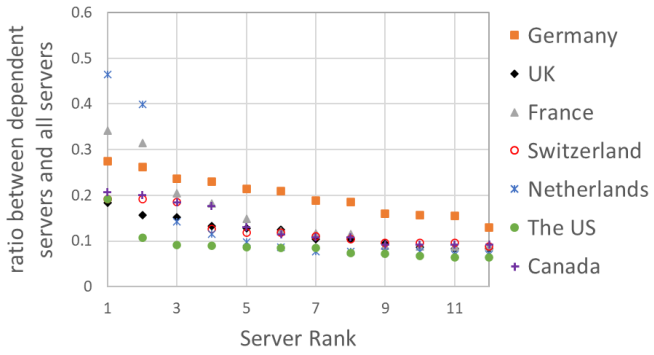


(a) North America

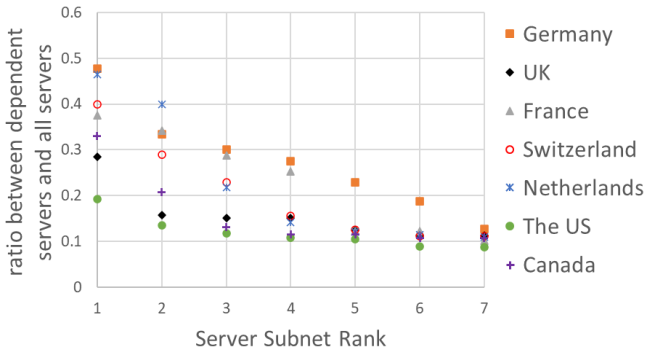


(b) Europe

Figure 2: Distribution of the number of system peers revealed in North America and Europe



(a) The most popular NTP timeservers to sync with, in decreasing order



(b) The /24 subnets containing popular NTP timeservers, in decreasing order

Figure 3: Inter-server dependencies in Germany, the UK, France, Switzerland, the Netherlands, the US, and Canada.

timeservers, can still impact time across its region. We will then show that despite timeserver removals by the monitor, the attacker can still succeed in shifting time by hours at many clients in its region.

A. *Control over Fairly Few Timeservers is Enough*

Quantifying the extent to which a set of timeservers can impact time at other timeservers involves inferring the dependencies between timeservers, that is, which timeservers sync to which other timeservers. This, however, is not straightforward, as explained next.

An NTP timeserver provides, upon request, the identity of a *single* timeserver to which it syncs, called the "system peer". Recall (see Section II-A) that an NTP client periodically queries a set of timeservers, casts outliers, and derives, from the "surviving" timeservers, a time to update its local clock to. The surviving server whose time sample is "closest" to the computed time (in terms of "root distance" [31] and other parameters) is designated the "*system peer*" [31]. The system peer of an NTP timeserver can be inferred by sending a query to that server and examining the refid field in the server's response packet. For stratum ≥ 2 servers, if the system peer is IPv4, the refid field contains its IPv4 address. If the system peer is IPv6, the refid field contains the first four octets of the MD5 hash of the IPv6 address. The IPv6 address of an IPv6 system peer can then be determined by finding a match of the refid value to the hash of a known IPv6 server.

However, the system peer is typically only one of *several* timeservers the NTP client syncs with and that influence the local time at the client. Hence, merely learning the identity of the system peer is not sufficient as the identities of *other* timeservers that impact time at the client will remain hidden.

Building the timeserver dependency graph. The best practice for NTP timeservers is to sync with a set of $4-7$ *manually-configured* timeservers [13], which is not expected to significantly change over time (see Section II-A). Our aim is to infer these sets, thus generating NTP's *server-dependency graph*. To this end, we applied the following methodology. We first compiled the lists of timeservers in the NTP server pool. This was accomplished by launching an NTP client and repeatedly querying the NTP server pool for timeservers in different geographical regions spanning the globe, once every two minutes for over a week. E.g., the response for a DNS query for the domains europe.pool.ntp.org and north-america.pool.ntp.org will specify several timeservers, selected

at random, in Europe and in North America, respectively. By accumulating the responses, and generating the union of the timeservers, we were able to learn the IP addresses of around 99% of all timeservers in the NTP timeserver pool worldwide (the number of the timeservers in the pool is specified in [19]), and also associate with each timeserver the country in which it resides. We focus henceforth on NTP timeservers in Europe and North America, which, put together, constitute roughly 90% of all timeservers worldwide. We choose to restrict our attention to these two continents since only in these can country-level pool zones that consist of hundreds of servers be found. (When less timeservers are available an attacker in control of 10s of timeservers can trivially impact time at many clients.) For the same reason, we focus primarily on countries with more than 300 servers: Germany, the UK, France, and the US.

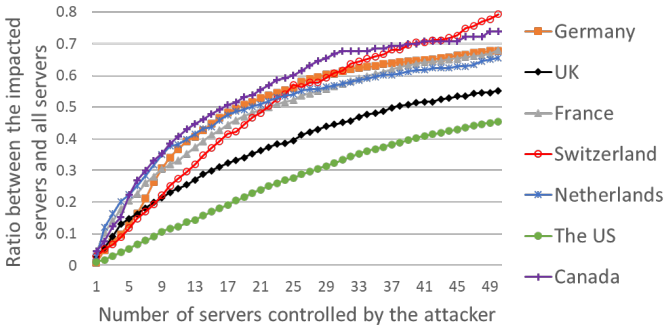


Figure 4: The fraction of timeservers in the UK, Germany, France, Switzerland, the Netherlands, the US and Canada (the y axis) for which x of the most popular NTP timeservers constitute more than 50% of the $Peer$ set.

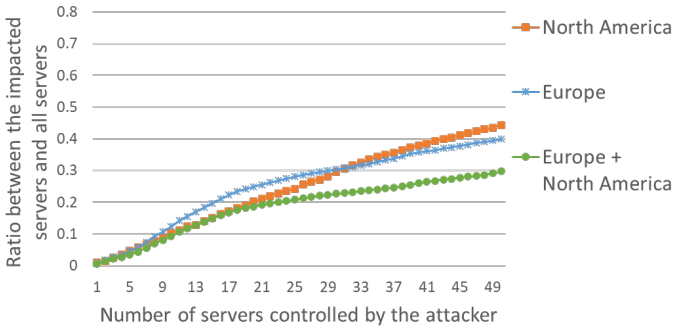


Figure 5: The fraction of timeservers in Europe and North America (the y axis) for which x of the most popular NTP timeservers constitute more than 50% of the $Peer$ set.

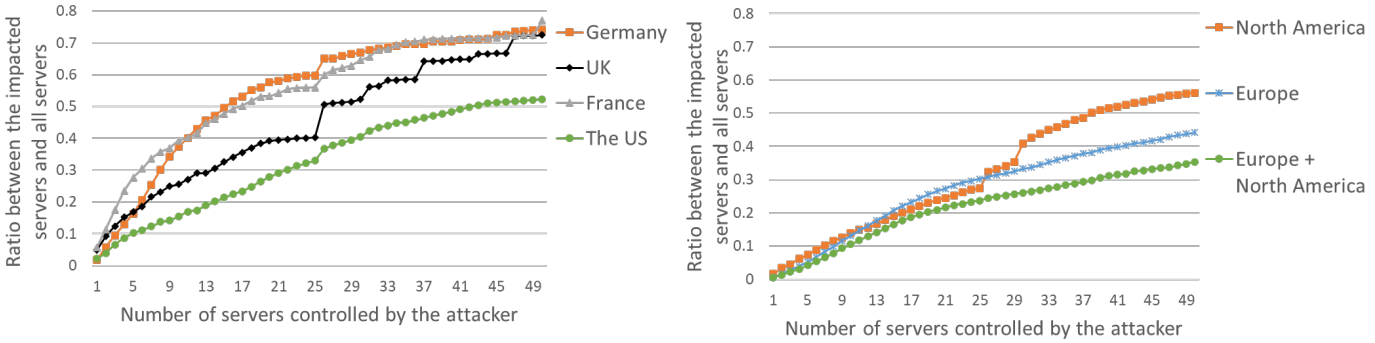
After replicating the list of timeservers in the NTP timeserver pool, we queried each of the timeservers for its system peer 8 times per hour for several months (from July to September 2019 and in July 2020). We denote the union of all systems peers reported by each timeserver i by $Peers(i)$. As discussed above, $Peers(i)$ might be a strict subset of the actual set of timeservers that timeserver i can sync with and so there might be (and likely are) even more dependencies between timeservers than those revealed by our empirical investigation. Fig. 2 plots the distribution of the size of the generated $Peer$ sets for timeservers in North America and Europe. The distributions for specific countries such as Germany, the UK, France, Canada, and the US, exhibit the same trends. Since the exact size of the set of manually-configured timeservers a certain timeserver i syncs with is unknown to us, we cannot tell what fraction of this set is covered by $Peer(i)$. In particular, $Peer(i)$ might be smaller than the recommended size of $4-7$ timeservers either because timeserver i has configured a lower number of timeservers than recommended or because some of the timeservers in its configured set were not chosen by it as the system peer in the course of our measurements.

We present below results for various countries in both Europe and North America: Germany, the UK, France, Switzerland, the Netherlands, the US, and Canada. Even though the pool's zones for these countries contain many NTP timeservers (e.g., 818 and 337 NTP timeservers in Germany and the UK, respectively), our results indicate that a fairly small-sized set of timeservers can impact time at many of the other timeservers. Fig. 3(a) plots for different timeservers in Germany, the UK, France, Switzerland, the Netherlands, the US, and Canada (the x axis), in decreasing order of popularity, the fraction of all timeservers in the country that have these timeservers in their system peer sets. Observe, for instance, that one timeserver in Germany (at $x = 1$) is in the intersection of the $Peer$ sets of 27% of the timeservers in Germany.

We observe also that the IP addresses of highly popular timeservers are sometimes in the same $/24$ subnet and, in fact, are sometimes even consecutive addresses. This implies that a single organization might be in control of a several popular NTP timeservers. Indeed, a closer inspection of our results reveals that both the National Metrology Institute of Germany (PTB) and the University of Erlangen-Nuremberg (FAU) in Germany control 3 and 4 of the most popular timeservers in Germany, respectively. This has important implications for security: gaining control of an IP prefix, e.g., via BGP hijacking [10], can enable even an off-path attacker to become the destination of NTP queries with respect to multiple popular timeservers. Fig 3(b) plots for different $/24$ IP subnets in Germany, the UK, France, Switzerland, the Netherlands, the US, and Canada (the x axis), ordered from most popular downwards, the fraction of the timeservers in the country whose $Peer$ sets contains at least a single timeserver in the subnet. Observe that for the $/24$ subnet in Germany that corresponds to $x = 1$, at least 47% of the timeservers in Germany have at least a single timeserver in their $Peer$ sets within this subnet.

Control of fairly few timeservers is sufficient for influencing time at many other timeservers. Using the timeserver-dependency graph, we quantify the fraction of the timeservers in a country/continent whose local time can be influenced by a fairly small number of timeservers.

We first present our results for timeservers in Germany, UK, France, Switzerland, the Netherlands, the US, and Canada in Fig. 4. A point (x, y) in the figure means that x of the most popular NTP timeservers in the country constitute more than 50% of the $Peer$ set for y% of the timeservers in the country. Thus, for instance, 20 of the most popular timeservers in Germany constitute the majority of the timeservers in the $Peer$ sets of 52% of the timeservers in Germany. We point



(a) Countries

(b) Continents

Figure 6: The fraction of server weight (the y axis) for which x of the most popular NTP servers constitute more than 50% of the $Peer$ set.

out that no NTP client is protected from an attacker in control of more than 50% of the timeservers in its $Peer$ set [6]. In fact, our results *underestimate* the influence of popular NTP timeservers as, in practice, control of less than 50% of a client's $Peer$ set might sometimes be sufficient for shifting time at a client (see discussion in Section IV.D in [6]).

Fig. 5 presents our results for Europe and North America. Observe that controlling as few as 50 timeservers in Europe is sufficient for dominating the $Peer$ sets of roughly 40% of the $2,810$ timeservers in the continent. Similarly, controlling 44 timeservers in North America suffices for dominating the $Peer$ sets of 40% of the 915 timeservers in North America.

So far, we have quantified the influence of a set of servers in terms of the *fraction of servers* whose Peer sets are dominated. Recall, however, that different timeservers are weighed differently by the server pool (according to their configured *netspeeds*, see Section III-B). Fig. 6 thus quantifies the influence of a set of timeservers in terms of the aggregate *weight* of impacted timeservers. Specifically, a point (x,y) in the figure indicates that x of the most popular NTP servers in the country constitute more than 50% of the Peer set for timeservers whose aggregate weight is $y\%$ of the total weight across all servers in the country). Observe that quantifying influence by weight yields the same trends as before.

B. The Implications of Timeserver Removals

As discussed in Section III-A, timeservers whose local times are influenced by the attacker-controlled timeservers will eventually be removed from the pool by the pool's monitor after responding to its queries with times that deviate from its local time by "too much" (three seconds in the current implementation). While these servers will no longer be assigned to clients by the pool (until re-admitted to the pool), as we show next, a large fraction of clients that already sync with a removed server will continue to do so for 10s of minutes, and even several hours, after that server is removed from the pool. We conclude that this is due to these clients querying the pool for new servers at these time granularities, and so sticking with the timeservers assigned to them for extended periods of time. Thus, the attacker can continue shifting time at timeservers even after their removal, indirectly shifting time

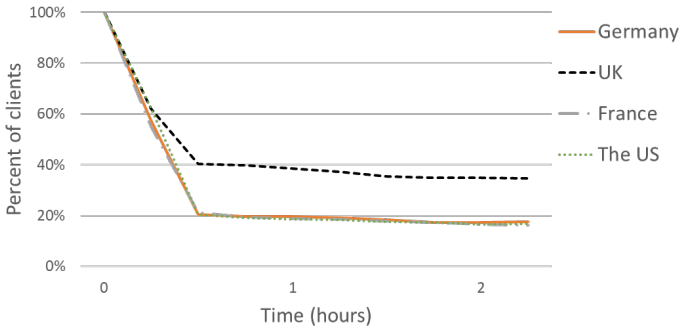


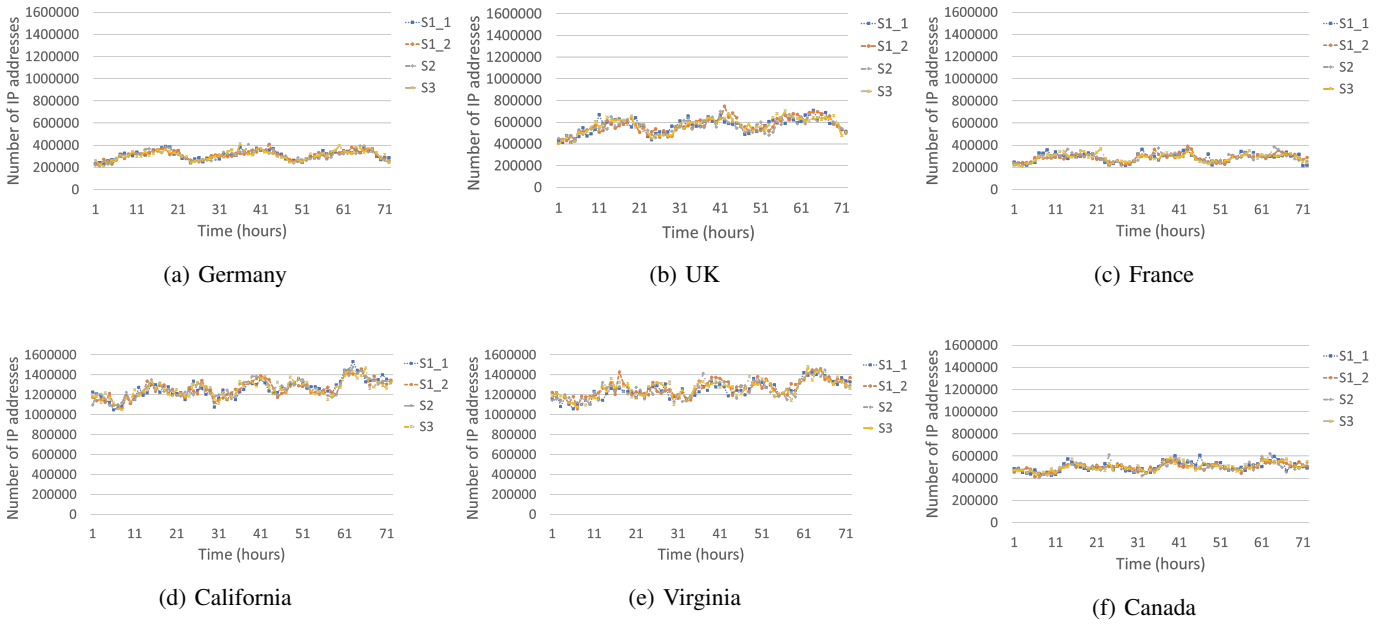
Figure 7: The number of clients that sync with a removed server.

at many clients by *hours* before the critical mass of clients abandons the removed server.

To quantify the fraction of clients that continues syncing with an NTP server after its removal from the pool, we launched (in July 2020), NTP timeservers at various Amazon AWS regions and registered these into the NTP timeserver pool (we discuss timeserver registration in more detail in Section V). The netspeed for all registered timeservers configured to be 1000x the default value, so as to attract sufficient traffic for the results to be meaningful. Once a timeserver reached a steady number of queries from distinct IP addresses per hour, we removed that server from the NTP pool and continued tracking the number of distinct IP addresses from which that server received NTP queries over time. We found that, as presented in Fig. 7, a large fraction of the clients that synchronized with our timeservers (around 20% or more in all regions) continued to do so for hours after these timeservers have been removed from the pool. In addition, the majority of the other clients continued syncing with our timeservers for 10s of minutes after their removal.

V. Empirical Analysis of the Implications of Injecting New Timeservers

We next quantify the effects of attacks that are based on injecting new timeservers into the timeserver pool, as discussed



(a) Germany (b) UK (c) France

(d) California (e) Virginia (f) Canada

Figure 8: Number of queries to our servers from distinct IP addresses per hour to our timeservers in Europe and North America. $S1\_1$ and $S1\_2$ are two stratum 1 servers, whereas $S2$ and $S3$ are stratum 2 and stratum 3 servers, respectively.

in Section III-B. Registering a new timeserver into the timeserver pool entails providing an IP address and an e-mail. After passing simple tests intended to establish the availability and time accuracy of the server, the registered server is added to the pool [14], [20]. The stratum of a timeserver is periodically reported to the pool's monitor whenever queried by the monitor for the current time. While, intuitively, claiming to be a low-stratum timeserver (e.g., a stratum 1 server), even when this is not so, might seem beneficial, our results actually show that this type of lie does not really aid the attacker. We identify, however, a much more effective strategy for the attacker: manipulating the *netspeed* parameter.

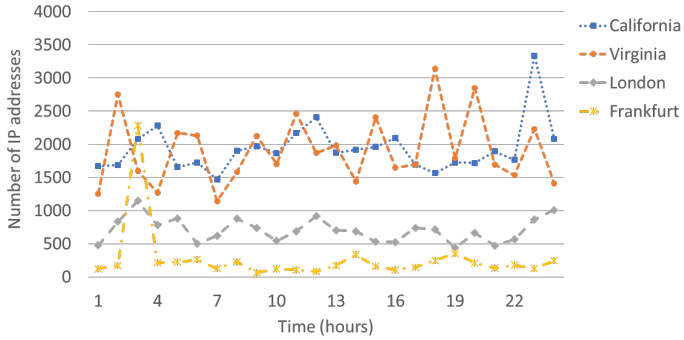


Figure 9: Number of queries from distinct IP addresses per hour to our timeservers in the UK, Germany and the US with default netspeed values.

Lying about your stratum is not helpful. To quantify the benefits to the attacker of lying about the stratum of a registered timeserver, we launched 4 NTP timeservers in the same Amazon AWS region and registered these into the NTP timeserver pool. We reported the strata of these timeservers to

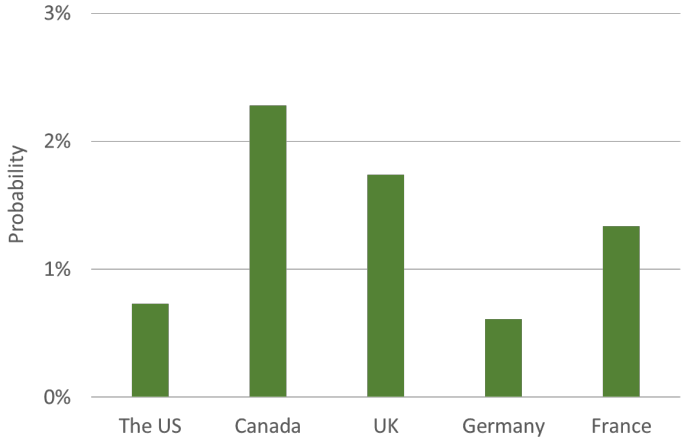


Figure 10: The probability that our injected server is assigned to a client in its region, averaged over the 4 injected servers in each region.

be 1 for two timeservers, 2 for one timeserver, and 3 for one timeserver. The netspeed for all timeservers (to be discussed below) was configured to be 1000x the default value, so as to attract sufficient traffic for the results to be meaningful. We repeated this experiment for different choices of Amazon AWS regions to show that the revealed trends do not vary based on the geographical locations of the timeservers. We measured, for each timeserver, the number of different IP addresses from which it received NTP queries within the same hour, across 72 consecutive hours. Our results, shown in Fig. 8, establish that all timeservers were contacted from roughly the same number of distinct IP addresses regardless of their strata. This is to be expected since the NTP client's algorithm for selecting between pool-assigned timeservers only examines the stratum

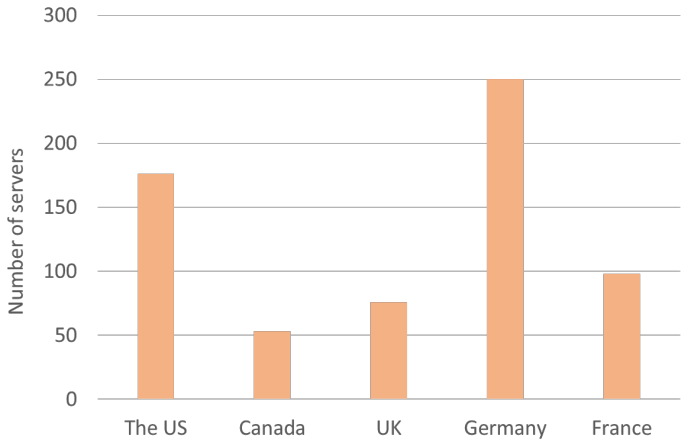


Figure 11: Required number of injected servers to constitute over 50% of the total weight of all timeservers in the region.

to discard timeservers with "unreasonable" strata (higher than 16 [45], [46]). The difference in numbers across regions can be explained by the differences in the total number of timeservers per client across regions.

The netspeed parameter. The specified "netspeed" of a timeserver in the NTP pool is correlated with the volume of NTP clients directed to the timeserver by the timeserver pool, with higher netspeed yielding higher probability that the timeserver be assigned to a client by the pool. Specifically, suppose that the timeserver pool for a certain region, say, the US, consists of n timeservers, and each timeserver i's specified netspeed is w_i. When contacted by an NTP client in the US, the NTP pool will select r timeservers to assign to the client as follows: the first timeserver to assign to the client is selected by sampling a single timeserver from the n timeservers according to the probability distribution in which each timeserver j's probability of being selected equals its proportional weight $\frac{w_j}{\sum_i w_i}$; the next timeserver is then similarly selected from the remaining $n-1$ timeservers according to the proportional weights after the removal of the first timeserver, and so on until r timeservers are selected. See [16]. Our experiments with different values of netspeed validate the above.

Configuring the netspeed to be high is highly effective. As evident from the above discussion, by configuring the netspeed parameter to be high, a timeserver can drastically increase the number of clients directed to it by the timeserver pool. To illustrate this point, we contrast the results in Fig. 8 with the results of the same experiments with the netspeed set to be the default value. Fig. 9 plots the averaged results for each region across 24 hours of measurements. The difference in the number of requests from clients between the two figures demonstrates the huge benefits for the attacker from configuring its netspeed to be high.

We next explore the probability that a timeserver introduced by us into the NTP timeserver pool be assigned to a client in its region. To this end, we launched an NTP timeserver on Amazon AWS in different regions, as in the above experiments, set its netspeed to 1000x the default value (i.e., to the maximum permissible value), and queried the NTP pool every 2 minutes over the course of a week from a client in the same region. We deduce from the fraction of responses in which our timeserver appears the probability that it be assigned by the pool to a client in its region. Fig. 10 summarizes our results. As shown in the figure, due to their high netspeed values, the probability that each of our timeservers is assigned to a client in its region is much higher than the probability had all timeservers been uniformly sampled.

Based on these computed probabilities, we quantify the number of timeservers an attacker needs to inject at a certain region so that the aggregated weight of its injected timeservers be more than half the total weight of timeservers in that region. This would imply that (in expectation) the set of servers assigned to a client by the pool is dominated by the attacker. The results are shown in Fig 11. Note the diversity across different regions, which reflects the different netspeed distributions in different regions. Observe also that by injecting 10s of (properly configured) timeservers in regions such as Canada and UK, an attacker can guarantee that (in expectation) most of the timeservers assigned to a client by the pool be the attacker's.

VI. Towards Better Security

After highlighting NTP's high vulnerability to malicious servers, we now turn our attention to identifying possible ways forward. To this end, we first outline the requirements from any security solution and explain why previously considered approaches fall short of achieving these. We then discuss an alternative, more secure, methodology, which we view as a promising first step. Our proposed approach is based on coupling recent proposals for security-enhanced NTP clients, namely, the Chronos NTP client [6], with a more secure scheme for assigning timeservers to clients, and is compatible with current efforts for standardizing Chronos by the IETF.

A. Requirements from Any Solution

Any solution for NTP's alarming vulnerability to malicious servers must satisfy three basic requirements: (1) preserve the time accuracy and precision of today's NTP, (2) respect today's load distribution over timeservers, and (3) enhance NTP's security against malicious servers.

Preserve NTP's time accuracy and precision. Time accuracy of an NTP client refers to the proximity of the local time at a client to the Coordinated Universal Time (UTC). Time precision of a client refers to the consistency of clock readings over time. For instance, a clock whose local time jitters uniformly at random around the UTC might be accurate (if realized times are sufficiently close to the UTC) but not precise. In contrast, a clock whose time progresses linearly, might be precise, but not accurate (for instance, if the clock's local time progresses twice as fast as the actual time; increasing local time by $2\delta_t$ in every δ_t time interval). Both accuracy and precision are important for the correct and secure operation of different applications. Naturally, a solution for NTP's security problems will ideally not come at the expense of interfering with NTP's primary objective—providing accurate and precise times.

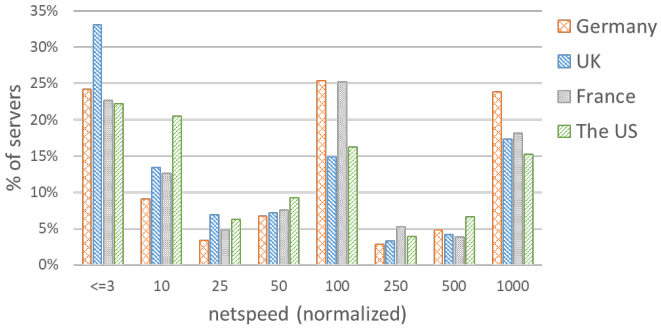


Figure 12: Distribution of timeservers' netspeeds.

Respect today's load distribution across timeservers. The netspeed parameter, discussed in Section V, is used by the NTP timeserver pool to load balance across timeservers, yielding load distributions in which some timeservers' probability of being assigned to clients (and so the expected number of clients that sync with these servers) is higher by three orders of magnitude than others. This accounts for differences in hardware/capacity across timeservers, and also reflects timeserver contributors' willingness to support different volumes of NTP queries. Deviating from the current load distribution by "too much" could lead to NTP servers being overloaded, resulting in inaccurate times (due to local burden) or even server crashes.

Improve security. Naturally, any solution should address the attack vectors presented in the previous sections.

B. A Failed Simple Server-Assignment Scheme

The Chronos NTP client [6] improves over NTPv4's algorithm for computing local time by applying a provably secure approximate-agreement algorithm (see Appendix B) to (randomly sampled) clock readings from a large timeserver set. In [6], a simple heuristic for generating such a server set for a Chronos client is presented. Specifically, the Chronos client repeatedly issues DNS queries to the NTP pool for servers to sync with, aggregating the IP addresses of servers received in response until the number of distinct addresses exceeds a desired threshold (say, several 100s of timeservers).

As we argue below, however, this heuristic for selecting servers to sync with for Chronos clients leaves such clients unprotected from the attack vectors discussed earlier and will increase the load on a non-negligible fraction of NTP servers by orders of magnitude. In addition, Chronos' synchronization process, which is optimized for accuracy, can potentially lead to suboptimal precision. We next discuss why this is. We then (in Section VI-C) explain how Chronos' client-side synchronization process can be coupled with an appropriate server assignment scheme in a manner that meets our three goals for NTP security (as described in Section VI-A).)

Security vulnerabilities. Chronos' security guarantees are conditioned on the attacker not controlling "too big" a fraction of the server set the client can sync with. The simple heuristic for assigning servers to Chronos clients proposed in [6], while yielding a large server set for a client, is oblivious to the dependencies between these servers and to their reputation. As our results in Sections IV and V show, simply relying on a large number of servers that the client can potentially sync with is not enough. First, inter-dependencies between timeservers could imply that even an attacker in direct control of a fairly small fraction of the timeservers can impact time at a huge fraction of the servers. Second, by injecting sufficiently many servers into the pool, the attacker can easily dominate the set of timeservers available to the client.

Suboptimal time precision. Even if *all* timeservers with which the Chronos client communicates are honest, the periodic transition between different subsets of servers will inevitably cause local time jitters due to variations in local time readings across different servers, as well as different network latencies between the client and the different servers queried. (This should be contrasted with today's NTPv4-clients, which stick with the timeservers with which they synchronize for long stretches of time to avoid bad time precision).

Overloading timeservers. We show that, if deployed at scale, the above discussed simple heuristic for assigning servers to clients, runs into the risk of inducing significant deviations from today's load distribution, in which a large fraction of the timeservers experiences over $200x$ increase in load. Intuitively, this is because today's practice of assigning timeservers to NTP clients with probability that is proportional to their netspeed values (see Section III-B) is at odds with Chronos' uniform distribution of load across large sets of servers, irrespective of their netspeed values (which is needed for establishing its security guarantees [6]). Consequently, if the sets of servers with which Chronos clients synchronize, and the frequency of synchronization, are not chosen with care, low-netspeed servers will experience a huge surge in load.

Estimating what the exact implications of high rise in load on a substantial fraction of the server pool is hard as we do not have visibility into NTP servers' hardware and external (non-NTP-related) load. That said, we point out that: (1) Beyond the risk of NTP servers crashing, drastic rise in load can also harm time accuracy due to the local computational burden at servers. As the volume of NTP traffic keeps rising over time, these risks might be further aggravated. (2) The netspeed parameter enables those volunteering timeservers to the NTP pool to control the (relative) number of NTP queries they will support, thus keeping the entry bar low for new volunteers and preventing the over-centralization of the system. Eliminating this control might prove detrimental in this respect.

To establish that using the simple heuristic proposed in [6] to assign timeserver sets to Chronos clients is problematic, we first empirically infer today's distribution of netspeed values across timeservers. To this end, we repeatedly issued DNS queries to the NTP timeserver pool (every two minutes over the course of two weeks) to obtain timeservers in Europe (via europe.pool.ntp.org) and in the US (via us.pool.ntp.org). The ratio between the number of responses in which a timeserver appears and the total number of responses approximates the timeserver's proportional weight. The distributions of netspeed values of timeservers in Europe and the US are presented in Fig. 12. Netspeed value of 1 in our figures stands for the default netspeed value whereas 1000 represents 1000x the default value (the maximum permitted). As can be seen in these

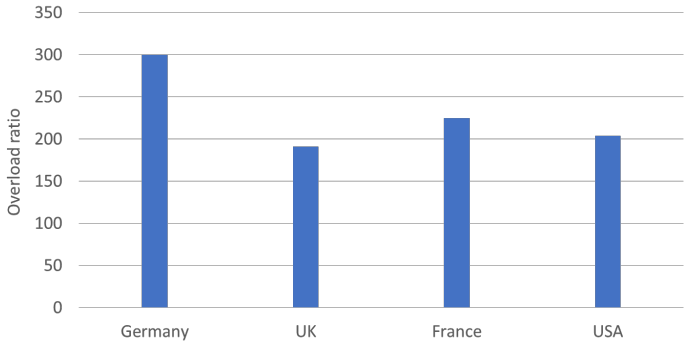


Figure 13: The expected overload experienced by timeservers with (normalized) netspeed 1 under uniform sampling.

figures, a large fraction of all NTP timeservers have netspeeds ≤ 3 (almost 25% in Europe and 20% in the US) and over 50% of the timeservers in both regions have netspeeds no higher than 50.

Chronos' uniform sampling of servers to sync with effectively equalizes load across the timeserver set assigned to a Chronos client. We next quantify the effects of wide adoption of this scheme on load balancing if timeserver sets are assigned using the simple heuristic considered in [6]. Fig. 13 presents the multiplicative factor by which the load on a timeserver with default netspeed in various regions (in which hundreds of timeservers are available) would increase had timeservers been sampled uniformly at random from all timeservers in the region. Observe that in all considered regions, the timeservers whose netspeed is 1, which constitute a large fraction of all timeservers in the region, will suffer an increase of $200-300$x. Observe also that this is not easily addressed by adding more timeservers to a certain region (that is, to the appropriate NTP pool zone), as the current number of servers in that region would have to be increased by two orders of magnitude for today's low-netspeed servers to not experience an increase in load.

C. Overview of Our Approach

Our objective, as discussed in Section VI-A, is to enhance NTP's security against malicious servers while not adversely impacting its time accuracy and precision, nor the distribution of load across timeservers. Another important design goal is preserving the ability of the pool to continue scaling through the continuous and easy-to-do addition of new timeservers at diverse geographical locations.

Our solution is simple: each NTP client should simultaneously run two parallel synchronization processes. The first synchronization process is precisely that used by today's NTPv4 clients to sync with pool-assigned servers in their region. This "primary" synchronization process is used, by default, to determine the client's local time. The second synchronization process is run in "watchdog mode"; the client applies Chronos' provably secure approximate agreement algorithm [6] to a large timeserver-set consisting of pool-provided stratum 1 timeservers, called "*Ananke*". So long as the watchdog's time does not deviate from the primary time by "too much", the primary sychronization process continues to update the local time. If, however, the results of these two time calculations are too far apart, which is indicative of an attack, the watchdog takes over and updates the local time (until such a time when the two computed times are close again).

We next discuss a few important details concerning this scheme.

Why Ananke? Which timeservers should Ananke contain? Since the local time at stratum 1 timeservers in the pool is not dependent (by definition) on other NTP timeservers in the pool, syncing with the timeservers in Ananke circumvents the inter-server-dependency-induced security vulnerabilities discussed in Section IV. That is, an attacker in control of a subset of Ananke cannot impact time at any of the other servers in Ananke. This, however, does not prevent timeserver-injection attacks of the form discussed in Section V.

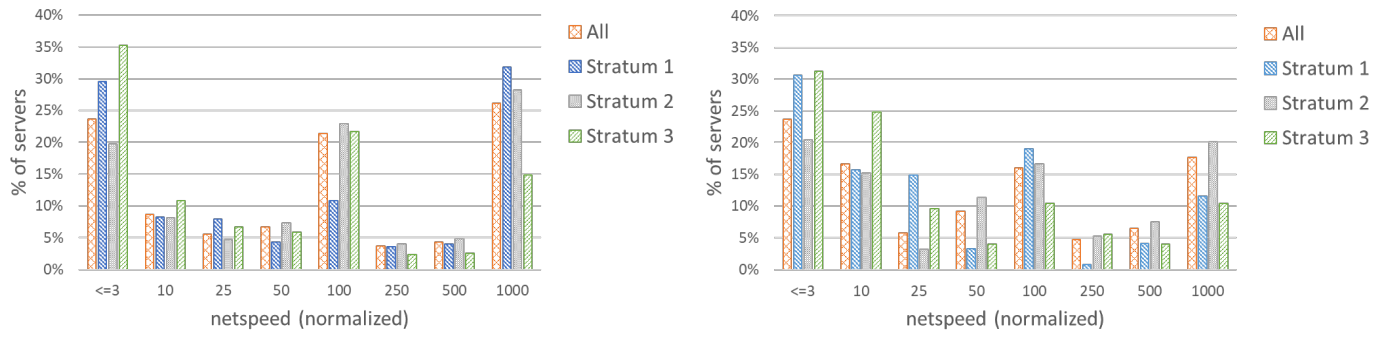
Ananke should not automatically consist of all stratum 1 timeservers in the pool. We argue that the addition of timeservers into Ananke should be performed much more cautiously than the addition of timeservers to other pool zones (which can remain unaltered), and should involve some manual auditing process (e.g., incorporating timeservers demonstrated to belong to reputable organizations). This will significantly raise the bar for an attacker while not impacting the pool's ability to continue expanding (as non-Ananke timeservers can still be easily registered to the pool). In addition, servers in Ananke should have acceptable compute power and capacity.

As a first step, we envision this manual auditing as being done by the NTP pool project itself to offer a "secure mode" of operation to interested parties. In the long run, global authorities (such as IANA) could take responsibility for this (as with other roots-of-trust for core Internet protocols).

How to avoid overloading the timeservers in Ananke? Observe that today's load distribution over timeservers not in Ananke is trivially preserved since these are only relevant for the primary synchronization processes (which are exactly as in today's NTPv4 time synchronization). To avoid overloading the timeservers in Ananke, syncing with these timeservers in our scheme occurs much less frequently than in the primary synchronization process. We will discuss how this is achieved while attaining good security guarantees in Section VI-F.

Today's NTP time accuracy and precision are preserved. Observe that the time accuracy and precision of today's NTP clients are trivially preserved by our scheme (since the primary processes are identical to NTPv4 synchronization).

Remark: Simply applying Chronos to the NTP pool's stratum 1 servers is not good enough. Our solution should be contrasted with simply applying Chronos to all stratum 1 servers (without imposing two synchronization processes that operate at different time scales and are applied to different sets of timeservers). Such a scheme, while simpler, suffers from two significant drawbacks: (1) Our empirical results show that the distribution of netspeeds within a region is fairly constant across strata in both the North America and Europe, as shown in Fig. 14. In addition, stratum 1 servers constitute around 11% and 13% of all servers in Europe and the North America, respectively. Thus, if Chronos clients were



(a) Europe

(b) North America

Figure 14: The distribution of netspeed across strata in Europe and North America

all to use stratum 1 servers, the load on a large fraction of such servers would increase by three orders of magnitude (due to them now carrying the burden previously placed on higher-stata servers and equalizing the load on all stratum 1 servers regardless of netspeed); (2) As discussed above (see Section VI-B), while this simple scheme might achieve time accuracy, Chronos' periodic transition between different subsets of servers will inevitably cause local time jitters, leading to suboptimal precision.

D. Realizing Our Approach

Algorithm 1 client-side synchronization pseudocode

```
1: Global count = 0
2: Global Ananke_time = 0
3: procedure GetTime(count)
4:     NTPv4_time = GetNTPv4Time
5:     if count == 0 then
6:         Ananke_time = GetChronosTime(Ananke)
7:     if |NTPv4_time − Ananke_time| > ω + count · (Θ + 1) · Δt then
8:         new_time = Ananke_time + count · Δt
9:     else
10:        new_time = NTPv4_time
11:    count = (count + 1)%F
12:    return new_time
```

ω	an upper bound on the distance from the UTC of the local time at any NTP server not injected by the attacker.
Θ	an upper bound on the drift of the client's local clock across time [ms/sec]
Δt	The estimated time interval from the last time GetNTPv4Time was executed [sec].
F	The number of times GetNTPv4Time is called for each time GetChronosTime is called.

Table III: Notation Table

A new Ananke pool zone. Ananke can be realized as a new NTP pool zone (see Section II-A), which a client can issue two types of queries to: (1) requesting the list of *all* timeservers in Ananke, and, (2) requesting a subset of the timeservers in Ananke of predetermined size ($m = 12$ in our security analysis) chosen uniformly at random from *all* servers in Ananke (and not only those in the client's region). We envision the timeservers in Ananke as being geographically dispersed across geographical, political, and organizational boundaries. Ananke can be bootstrapped using a manually chosen subset of the stratum 1 timeservers currently in the pool (which today contains around 400 timeservers [41]).

Building on recent advances in client-side synchronization. The pseudocode for the client-side synchronization procedure employed by our scheme appears in Alg. 1. As explained in Section VI-C, the client runs two parallel synchronization processes. One exactly identical to NTPv4's and another applying Chronos' approximate-agreement scheme to the servers in Ananke. The second synchronization process happens less frequently than the first; once every F NTPv4 updates (where the variable F captures the frequency ratio between the two processes). Local time at the client (the new_time parameter) is the time computed by NTPv4 ($NTPv4\_time$) by default, unless the gap between the two computed time values exceeds a certain threshold, which takes into account both the reasonable distance of an honest NTP timeserver from the UTC (ω) and the clock's natural *drift* since the servers in Ananke were last queried. When this occurs, Chronos' computed time is used to update the client's clock. Table III presents the notation used in the pseudocode.

We point out that the required client-side changes are highly compatible with those currently being promoted at the IETF [42]. Specifically, to benefit from Chronos' [6] improved security while preserving NTP's time accuracy and precision, Chronos' approximate-agreement-based time synchronization is intended to operate in the background (as a watchdog) while traditional NTPv4 is used to update the local time by default. Our scheme prescribes the set of servers with which the Chronos watchdog process interacts (Ananke), and the frequency of this interaction, to contend with attacks (by malicious servers) and load-related considerations, which were previously not considered by NTP security schemes.

E. Assigning Values to the Parameters F and ω.

The choice of values for the parameters in the pseudocode of Alg. 1 has important implications. Specifically, the choice

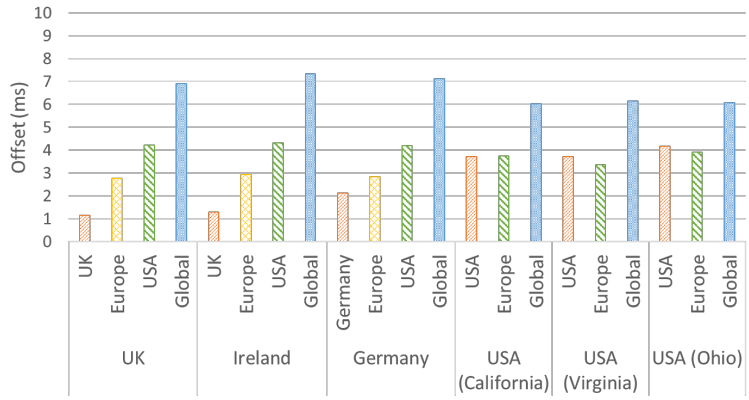


Figure 15: The average offsets measured from different locations with respect to timeservers at different levels of proximity. Different possible locations for NTP timeservers the client syncs with appear above.

of F determines how frequently timeservers in Ananke will be queried, and so the load on these servers. Another crucial choice is determining the gap between NTPv4's computed time and Chronos' computed time (for Ananke) required for Chronos' computed time to be used. The higher the gap is, the more time can be shifted at the client before the watchdog mode takes over; the lower the gap, the higher the risk that the watchdog takes action without cause, harming the client's time accuracy and precision.

Implications of F for load on timeservers in Ananke. Recall that stratum 1 servers constitute around 10% of timeservers in the NTP server pool, amounting to around 400 servers. We envision the set of Ananke servers as eventually consisting of hundreds of servers with acceptable compute power and capacity constraints (e.g., probably not include those that currently have very low netspeed values configured). Even so, if our scheme sees wide adoption, all NTP clients, which currently sync with thousands of servers, will *also* be required to periodically contact the timeservers in Ananke. To avoid overloading the servers in Ananke, clients should query these much less frequently than in the parallel NTPv4 synchronization process. In our security analysis (Section VI-F), $F = 10$. F could also set to higher values (resulting in lower loads on Ananke timeservers) without significantly weakening the security guarantees.

Implications of ω for time accuracy and precision. In our scheme, a client c in a certain region might sync with a stratum 1 server x in a *distant* region as part of the Chronos synchronization process. The time at x, as computed by c, also depends on the network latency between the two (see Appendix A for an explanation). Hence, ω, which serves as an upper bound on the distance between the local time at an honest timeserver and the UTC, should be set to be high enough to account to inaccurate time estimations due to long and variable network latency. We show below, however, that under normal conditions (i.e., when not under attack), these two times are not far.

The time offset between an NTP client and an NTP server is a value computed by the client that reflects the difference between the local times at the client and at the server (see Appendix A for a formal definition). Fig. 15 presents the average time offsets measured at an NTP client by querying timeservers at different levels of proximity to the client: (1) timeservers in the client's country, (2) timeservers in Europe excluding the timeservers in the client's country if located in Europe, (3) timeservers in the US, and (4) timeservers across the world excluding the timeservers in the US and Europe. As can be seen from the results, the difference between the average time offsets with respect to servers in the client's region and to faraway servers are merely several milliseconds apart. $\omega = 25$ms has been shown in [6] to be a good choice when the client syncs only with servers in its own region. Our offset analysis indicates that when syncing with servers from other regions, setting ω to be sufficiently higher to also account for inaccuracies of several milliseconds on average is needed. We use the very conservative choice of $\omega = 50$ms in our security analysis.

F. Security Analysis

The security of our scheme is immediately derived from Chronos' security guarantees [6]. As explained in [6], the expected time needed for the attacker to shift a Chronos client by T seconds from the UTC can be approximated by

$$\frac{I}{\left(P_{\frac{2}{3}m,m}\right)^{\frac{T}{E}}}, \tag{1}$$

where I is the length of the time interval between two consecutive time updates, E is the maximum time-shift permitted in each time update, m is the size of the subset of servers queried in each update, and $P_{\frac{2}{3}m,m}$ is the probability that at least two thirds of the sampled subset of servers are controlled by the attacker. In the case of our Chronos synchronization process (the watchdog), $I = F \cdot \Delta t$, $E = \omega + F \cdot \Theta \cdot \Delta t$.

Recall that the time computed by the Chronos synchronization process is only used to update the clock if the gap between this time and that computed by the NTPv4 synchronization process exceeds a certain threshold, which is upper bounded by E (as defined above). Thus, the time computed by our client can be at most further away from the UTC than the Chronos synchronization process by an additive factor of E.

This implies that the probability of an attacker to shift time at a client using our scheme by $T + E$ is as in Equation 1.

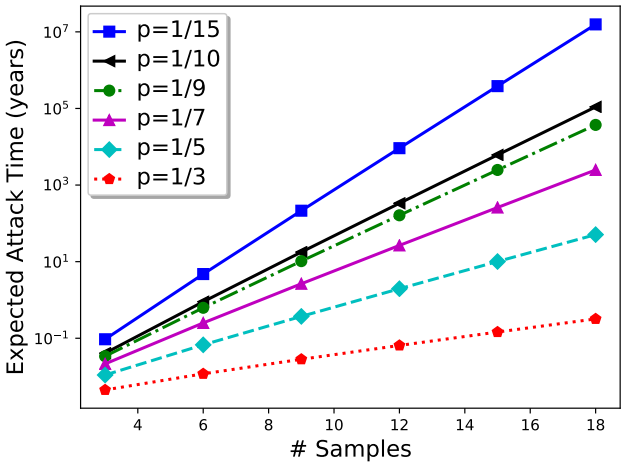


Figure 16: The (expected) time required for the attacker to succeed in shifting a client's time by 1.1s.

To illustrate the security guarantees provided by our scheme, we present the implications for security of specific choices of values for the different parameters.

Theorem 6.1: When $|Ananke| = 200$, $\Theta = 50ms/hour$ (as in [6]), $\Delta t = 1$ hour (as in [6]), $\omega = 50ms$, $m = 12$, $F = 10$, and the attacker controls $\frac{1}{7}$ of the timeservers in Ananke, 26 years in expectation are needed for the attacker to shift the client's local time by 1.1sec or more from the UTC.

Thus, even if the timeservers in Ananke are queried $10x$ less frequently than the "default" timeservers assigned by the NTP server pool, shifting time by over 1.1s requires 26 years in expectation for an attacker with significant presence in Ananke (which should not be trivial to accomplish, as Ananke should consist of hundreds of geographically diverse and manually audited stratum 1 servers). Consequently, even a fairly powerful attacker is effectively incapable of shifting time sufficiently to harm many applications of interest (see Table II).

Fig. 16 presents the expected time (in years) required for the attacker to shift a client's clock by 1.1sec from the UTC for different choices of m (the number of servers in Ananke sampled by the client in each update), and for different fractions of timeservers in Ananke controlled by the attacker. The assignment of values to all other relevant parameters is as in Theorem 6.1. Observe that the result in Theorem 6.1 corresponds to the value on the y-axis (26 years) for $x = 12$ (the number of timeservers in Ananke queried) of the curve representing the scenario that the attacker controls $\frac{1}{7}$ of the timeservers in Ananke.

We note that even if the frequency of queries to Ananke is further reduced (by, setting, e.g., $F = 100$), the security bounds yielded by our theoretical analysis would still be meaningful.

G. Another Proposed Defense: Deploying "Secret Monitors"

Our attacks leveraging injection of new timeservers into the NTP pool build on the attacker's ability to avoid being detected by the NTP pool's monitor. An attacker-controlled timeserver can evade the monitor by providing accurate times to the monitor when queried while selectively reporting erroneous times to others. (Recall that the attack that utilizes existing NTP servers takes into account that the attacker-controlled server will eventually be detected by the monitor and removed from the NTP server pool, as discussed in Section III-A).

Currently, the NTP pool employs a *single* monitor server, whose IP can be easily inferred; when a new timeserver is registered to the pool, the first NTP queries to that server are made by the monitor. A natural defensive measure, then, is to extend the pool's monitoring infrastructure to contain many monitors and attempting to keep the identities of these servers hidden. We believe that this will indeed raise the bar for an attacker and limit its ability to inflict harm. We point out, however, that preserving the anonymity of "secret monitors" might prove hard against strategic attackers, as such attackers can periodically misreport times and keep track of the IPs of the timeservers that queried them before their server scores were decreased by the NTP pool.

VII. Related Work

NTP, one of the Internet's oldest protocols, is still widely used throughout the world [5], [24], [27], [35]. However, NTP suffers from many security vulnerabilities. Already in 1985, in the context of the development of the Kerberos security model, NTP's inadequacy for achieving secure time synchronization was pointed out [33].

The NTP pool was created in 2003 to provide better reliability and scalability [41]. The current NTP pool is divided into zones, (e.g., europe.pool.ntp.org, us.pool.ntp.org, and de.pool.ntp.org [15], [17]). Thousands of timeservers in different zones were analyzed in [41]. Similarly to our measurements, [41] uses the "system peer" attribute to create a dependency graph. In addition, [41] reports experience with entering new timeservers into the pool. Our analysis differs from that in [41] in that: (a) [41] primarily targets non-security-related questions, with security-related discussions limited to sparsely populated regions of the pool (where control of a timeserver trivially grants the attacker immense power); (b) Our measurements are at somewhat finer granularity (e.g., servers are queried more often in our experiments).

Recently, there have been several studies describing errors, misconfigurations and attacks against NTP [5], [24]–[26], [43]. These studies demonstrate, for example, the ability of off-path attackers to launch denial-of-service (DoS) attacks and also to shift the local time at the client by exploiting weaknesses in NTP's implementation (e.g., via spoofed Kiss-o'-Death packets) [24], [25]. Recently introduced patches to NTP's implementation eliminate/mitigate some of these vulnerabilities.

Many efforts to secure NTP focus on authentication and encryption [3], [7]–[9], [11], [34], [36], [39], [40]. The Chronos NTP client [6] reflects an orthogonal, client-side approach that leverages approximate-agreement algorithms for secure time synchronization. We believe that this constitutes a promising

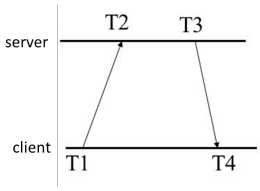


Figure 17: Time-offset computation (taken from [6])

approach for protecting NTP clients. However, our results indicate that to attain meaningful security guarantees and to avoid unacceptable load distributions on NTP timeservers, this approach should be coupled with an appropriate scheme for assigning timeservers to clients.

VIII. Conclusion

NTP is crucial for the correct and secure operation of many Internet services. We showed that NTP is highly vulnerable to attacks by malicious servers. We examined two such types of attacks: (1) attacks where the server is in control, or gains control, of existing timeservers in the NTP server pool, and (2) attacks where the attacker introduces new timeservers into the server pool. We also presented an agenda for enhancing NTP's security against malicious timeservers. Our proposed scheme balances different goals, namely, preserving today's NTP time accuracy and precision, improving security, and not overloading timeservers.

Acknowledgements

We thank Samuel Jero and the anonymous reviewers of this publication for many valuable comments. We also thank Danny Dolev, Tal Mizrahi, and the members of the IETF's NTP Working Group for helpful discussions. This research was partly funded by an ERC Starting Grant and by the Israel National Cyber Directorate (INCD).

References

[1] Ntp version 4.2.8p9 code, November 2016.

[2] Current versions of ntp. http://support.ntp.org/bin/view/Main/SoftwareDownloads, 2019.

[3] Andreeva, O., Gordeychik, S., Gritsai, G., Kochetova, O., Potseluevskaya, E., Sidorov, S. I., and Timorin, A. A. Industrial control systems vulnerabilities statistics. Tech. rep., Kaspersky lab, 2016.

[4] Boverman, A. Timejacking & bitcoin. Culubas blog, May 2011. http://culubas.blogspot.com/2011/05/timejacking-bitcoin_802.html.

[5] Czyz, J., Kallitsis, M., Gharaibeh, M., Papadopoulos, C., Bailey, M., and Karir, M. Taming the 800 pound gorilla: The rise and decline of ntp ddos attacks. In *Proceedings of the 2014 Conference on Internet Measurement Conference* (New York, NY, USA, 2014), IMC '14, ACM, pp. 435–448.

[6] Deutsch, O., Rozen-Schiff, N., Dolev, D., and Schapira, M. Preventing (network) time travel with chronos. Proceedings of the 25th Network and Distributed Systems Security Symposium (NDSS).

[7] Dowling, B., Stebila, D., and Zaverucha, G. Authenticated network time synchronization. In *25th USENIX Security Symposium (USENIX Security 16)* (Austin, TX, 2016), USENIX Association, pp. 823–840.

[8] Dowling, B., Stebila, D., and Zaverucha, G. Authenticated network time synchronization. In *Proceedings of the 25th USENIX Conference on Security Symposium* (Berkeley, CA, USA, 2016), SEC'16, USENIX Association, pp. 823–840.

[9] Franke, D. F., Sibold, D., Teichel, K., Dansarie, M., and Sundblad, R. Network Time Security for the Network Time Protocol. Internet-Draft draft-ietf-ntp-using-nts-for-ntp-20, Internet Engineering Task Force, July 2019. Work in Progress.

[10] Gilad, Y., Hlavacek, T., Herzberg, A., Schapira, M., and Shulman, H. Perfect is the enemy of good: Setting realistic goals for bgp security. In *Proceedings of the 17th ACM Workshop on Hot Topics in Networks* (New York, NY, USA, 2018), HotNets '18, ACM, pp. 57–63.

[11] Haberman, B., and Mills, D. Rfc 5906: Network time protocol version 4: Autokey specification. internet engineering task force (ietf), 2010.

[12] Hansen, A. B. Global — pool.ntp.org. https://www.pool.ntp.org/zone/@, 2018.

[13] Hansen, A. B. How do i join pool.ntp.org? https://www.ntppool.org/en/join.html, 2018.

[14] Hansen, A. B. How do i use pool.ntp.org? https://www.ntppool.org/en/use.html, 2018.

[15] Hansen, A. B. Europe — europe.pool.ntp.org, 2019.

[16] Hansen, A. B. netspeed - definition and explanation. https://github.com/abh/ntppool/blob/master/docs/manage/tpl/manage/servers_help.html, 2019.

[17] Hansen, A. B. North america — north-america.pool.ntp.org, 2019.

[18] Hansen, A. B. The ntp pool for vendors. https://www.ntppool.org/en/vendors.html, 2019.

[19] Hansen, A. B. Ntp pool project - introduction. https://www.ntppool.org/en/, 2019.

[20] Hansen, A. B., and Gallego, X. R. ntppool github project. https://github.com/abh/ntppool/blob/master/lib/NTPPool, July 2020.

[21] Hoch, D. Integrating sun kerberos and microsoft active directory kerberos, 2005.

[22] Hodges, J., and Jackson, C. Http strict transport security (hsts), November 2012.

[23] Ltd, N. M. Time traceability for the finance sector. Tech. rep., NPL Management Ltd, United Kingdom, March 2016.

[24] Malhotra, A., Cohen, I. E., Brakke, E., and Goldberg, S. Attacking the network time protocol. *IACR Cryptology ePrint Archive 2015* (2015), 1020.

[25] Malhotra, A., and Goldberg, S. Attacking ntp's authenticated broadcast mode. *SIGCOMM Comput. Commun. Rev. 46*, 2 (May 2016), 12–17.

[26] Malhotra, A., Gundy, M. V., Varia, M., Kennedy, H., Gardner, J., and Goldberg, S. The security of ntp's datagram protocol. In *Financial Cryptography and Data Security - 21st International Conference, FC 2017, Sliema, Malta, April 3-7, 2017, Revised Selected Papers* (2017), pp. 405–423.

[27] Malhotra, A., Toorop, W., Overeinder, B., Dolmans, R., and Goldberg, S. The impact of time on dns security. Cryptology ePrint Archive, Report 2019/788, 2019. https://eprint.iacr.org/2019/788.

[28] Marzullo, K., and Owicki, S. Maintaining the time in a distributed system. In *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing* (New York, NY, USA, 1983), PODC '83, ACM, pp. 295–305.

[29] Marzullo, K. A. Maintaining the time in a distributed system. Tech. rep., Xerox, 1984.

[30] Marzullo, K. A. *Maintaining the Time in a Distributed System: An Example of a Loosely-coupled Distributed Service (Synchronization, Fault-tolerance, Debugging)*. PhD thesis, Stanford, CA, USA, 1984. AAI8506272.

[31] Mills, D., Martin, J., Burbank, J., and Kasch, W. Rfc 5905: Network time protocol version 4: Protocol and algorithms specification. internet engineering task force (ietf), 2010.

[32] Mills, D. L. How ntp works. https://www.eecis.udel.edu/~mills/ntp/html/warp.html, March 2014.

[33] Mills, D. L., Mamakos, L., and Petry, M. Network Time Protocol (NTP). RFC 958, sep 1985.

[34] Mizrahi, T. Rfc 7384 (informational):security requirements of time protocols in packet switched networks, October 2014.

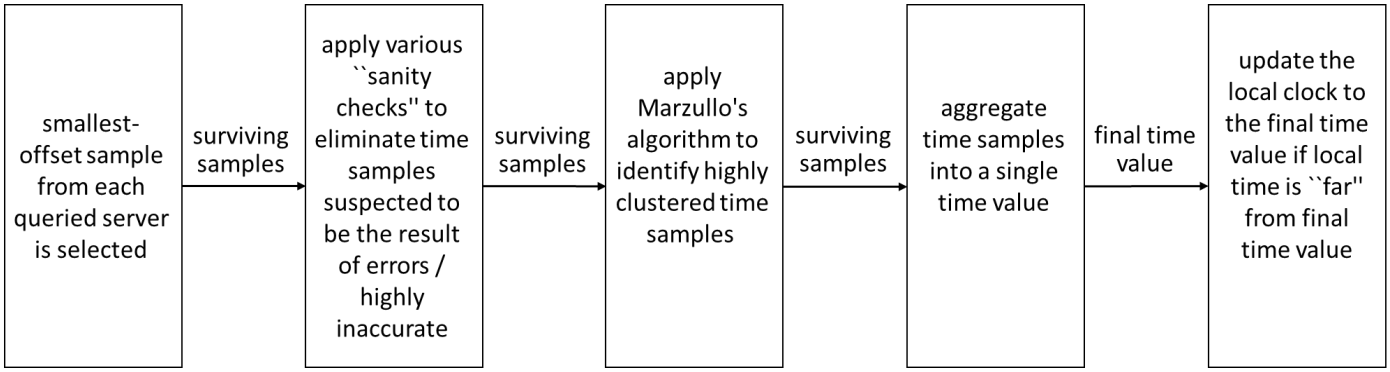


Figure 18: NTPv4's time-update computation (taken from [6])

[35] MURTA, C. D., TORRES JR., P. R., AND MOHAPATRA, P. Qrpp1-4: Characterizing quality of time and topology in a time synchronization network. In *IEEE Globecom 2006* (Nov 2006), pp. 1–5.

[36] NIST. The NIST authenticated ntp service. http://www.nist.gov/pml/div688/grp40/auth-ntp.cfm, 2010.

[37] NOVICK, A. N., AND LOMBARDI, M. A. Practical limitations of ntp time transfer. In *2015 Joint Conference of the IEEE International Frequency Control Symposium the European Frequency and Time Forum* (April 2015), pp. 570–574.

[38] NTP-NETWORK-FOUNDATION. How does it work?, 2019.

[39] REILLY, D., STENN, H., AND SIBOLD, D. Network Time Protocol Best Current Practices. RFC 8633, July 2019.

[40] ROTTGER, S. Analysis of the ntp autokey procedures. master's thesis, technische universitt braunschweig, 2012.

[41] RYTILAHTI, T., TATANG, D., KÖPPER, J., AND HOLZ, T. Masters of time: An overview of the ntp ecosystem. In *2018 IEEE European Symposium on Security and Privacy (EuroS P)* (April 2018), pp. 122–136.

[42] SCHIFF, N. R., DOLEV, D., MIZRAHI, T., AND SCHAPIRA, M. A Secure Selection and Filtering Mechanism for the Network Time Protocol Version 4. Internet-Draft draft-ietf-ntp-chronos-01, Internet Engineering Task Force, Sept. 2020. Work in Progress.

[43] SELVI, J. Bypassing http strict transport security. In *Black Hat Europe* (2014).

[44] SELVI, J. Bypassing http strict transport security. Black Hat Europe, 2014. https://www.blackhat.com/docs/eu-14/materials/eu-14-Selvi-Bypassing-HTTP-Strict-Transport-Security-wp.pdf.

[45] STENN, H. ntp header file. https://github.com/ntp-project/ntp/blob/master-no-authorname/include/ntp.h, 2016.

[46] STENN, H. ntp protocol code. https://github.com/ntp-project/ntp/blob/master-no-authorname/ntpd/ntp_proto.c, 2016.

APPENDIX

This appendix contains additional details about the NTPv4 and Chronos clients.

A. The NTPv4 Client

The time synchronization process between NTPv4 clients and NTP timeservers consists of two steps: (1) the *poll process*, in which the client exchanges messages with timeservers to collect servers' time samples, and (2) discarding outliers and computing the new local time from the remaining time samples. We next elaborate on each of these two steps [1], [31], [32].

The poll process. An NTPv4 client periodically queries a set of NTP timeservers to learn the clock readings at these servers. Through interaction with each server, the client obtains 4 distinct timestamps per query: (1) T_1, the local time at the client when the query is sent, (2) T_2, the local time at the server when the query is received, (3) T_3, the local time at the server when the response is sent, and (4) T_4, the local time at the client when the response is received.

Algorithm 2 Pseudocode for Chronos' Time Sampling Scheme [6]

```
counter := 0
while counter < K do
    S := sample(m)          ▷ gather time samples from m randomly chosen servers
    T := bi-sided-trim(S,d) ▷ trim d lowest and highest values
    if (max(T) − min(T) <= 2ω) and (|avg(T) − t_C| < ERR + 2ω then
        return avg(T)
    counter++
                            ▷ panic mode
S := sample(n)
T := bi-sided-trim(S, n/3)  ▷ trim bottom and top thirds
return avg(T)
```

These timestamps are then used to compute the *offset* $\theta = \frac{1}{2}((T_2 - T_1) + (T_3 - T_4))$ [24], [37], which is intended to capture the difference between the local times at the client and at the server. See Fig. 17 (taken from [6]) for an illustration. The client queries each server several times to obtain several offsets associated with that server.

Computing the local time at the client. After computing these offsets, the client applies a 5-step algorithm to compute a new time to update its local clock to, as described in Fig. 18 (taken from [6]). For each timeserver, the offset associated with the lowest network delay measured with respect to that server is identified. Marzullo's algorithm [28]–[30] is applied to these offsets to identify a "majority clique of truechimers" [1], [32], i.e., a large cluster of servers with accurate clocks. This set of time samples can be further pruned, with the aim of improving accuracy, by removing all but some predetermined number of time samples that are within the smallest distance of each other. Lastly, a weighted average of the offsets of the remaining time samples is computed. If this value is "far" from the current

local time (and so the current local time is viewed as "stale"), the local time is updated to the computed time value.

B. The Chronos NTP Client

The Chronos [6] NTP client is a security-enhanced NTP client. Chronos' time synchronization process applies a provably secure approximate-agreement algorithm to a large set of timeservers. Specifically, a Chronos NTP client periodically obtain clock readings from m (say, $10 - 15$ servers) out of a large fixed set of servers S (ideally, containing 100s of servers). Then, the offsets with respect to these servers are ordered from lowest to highest and the bottom d and top d offsets according to this order are removed from consideration (choosing $d = \frac{m}{3}$) is shown in [6] to yield good security guarantees). If the surviving time samples are "not far" from each other and are (on average) close to the client's local clock, the local time is updated to be the average of these time samples. Otherwise, a new server set of size m to sync with is re-sampled from S. In the event of reaching k consecutive re-samplings, the Chronos client enters "panic mode" and queries *all* timeservers in S, again eliminating the top and bottom outliers and averaging over the rest, to determine its new local time. Chronos' pseudocode is presented in Algorithm 2.