Time Travel as an Attack Vector: Circumventing HSTS Protection Proof of Concept (PoC)

Authors: Șoga Antonie

Abstract

Network security protocols rely heavily on accurate time synchronization to validate cryptographic certificates and enforce security policies. However, the underlying mechanism for time synchronization, the Network Time Protocol (NTP), often operates without authentication, leaving it vulnerable to manipulation. This project investigates the "*Temporal Man-in-the-Middle*" attack vector, demonstrating how NTP spoofing can be leveraged to undermine HTTP Strict Transport Security (HSTS).

We present a theoretical analysis and a practical Proof of Concept (PoC) implemented in a containerized Docker environment. By intercepting unauthenticated NTP traffic, we demonstrate the ability to force a victim's system clock into the future, effectively bypassing the HSTS *max-age* directive. This expiration allows an attacker to downgrade the victim's connection from HTTPS to cleartext HTTP, facilitating the interception of sensitive credentials. The study validates the fragility of time-dependent security policies and highlights the necessity of adopting Network Time Security (NTS) to authenticate time synchronization and prevent such protocol downgrade attacks.

Introduction

The integrity of modern web security protocols is often dependent on underlying infrastructure that users and administrators assume to be trustworthy. One such critical dependency is time. Secure protocols, including Transport Layer Security (TLS) certificates and HTTP Strict Transport Security (HSTS), rely on accurate system clocks to determine validity periods.

Network Time Protocol (NTP), defined in RFC 5905 [4], is the standard for synchronizing clocks over packet-switched, variable-latency data networks. While effective at timekeeping, standard NTP implementations often lack cryptographic authentication, making them susceptible to spoofing and interception.

HTTP Strict Transport Security (HSTS), defined in RFC 6797 [3], was introduced to mitigate attacks such as SSL Stripping by allowing web servers to declare that web browsers (or

other complying user agents) should only interact with them using secure HTTPS connections. This policy is cached by the browser for a duration specified by the max-age directive [3].

The conflict arises when an attacker manipulates the underlying time dependency. If an attacker can spoof NTP packets to shift the victim's clock forward beyond the max-age duration, the browser will legitimately interpret the HSTS policy as expired. This project investigates this vulnerability vector. We implement a containerized laboratory environment using Docker to simulate a full kill chain: establishing a Man-in-the-Middle (MiTM) position, poisoning DNS to redirect traffic, and spoofing NTP to bypass HSTS protection. The goal is to demonstrate the fragility of time-dependent security policies when the time source itself is insecure [1].

Related work

The theoretical and practical foundations of this attack vector have been established by several key researchers in the field of network security.

- **Temporal MiTM Attacks:** The specific technique of shifting a victim's time to bypass HSTS was notably presented by Jose Selvi at Black Hat Europe 2014 in his work "Bypassing HTTP Strict Transport Security" [1]. Selvi introduced the "Delorean" tool, which automates the process of serving NTP responses that push the victim's clock ahead. Our project reproduces this concept in a modern, containerized environment, isolating the specific mechanism of NTP spoofing combined with DNS poisoning.

- **SSL Stripping:** The vulnerability that HSTS aims to prevent—and which becomes re-enabled once HSTS is bypassed—was originally demonstrated by Moxie Marlinspike in "*New Tricks For Defeating SSL In Practice*" (Black Hat DC 2009) [2]. Marlinspike showed that without strict enforcement, users could be transparently downgraded from HTTPS to HTTP. This project utilizes this downgrade method as the final payload of the attack chain.

- **DNS Vulnerabilities:** The ability to redirect a victim to an attacker-controlled proxy relies on manipulating the Domain Name System. RFC 3833 analyzes threat models such as "ID Spoofing" and Man-in-the-Middle attacks on DNS [6]. Dan Kaminsky's foundational research, "*It's The End Of The Cache As We Know It*" (2008) [7], further demonstrated the fragility of DNS trust. In our PoC, we simulate this interception using a custom *dnsmasq* configuration to authoritatively resolve target domains to our attacker container.

Attack description

This project explores a composite attack vector that combines network traffic interception with time manipulation. The attack targets the trust relationship between the client (victim) and

the server, specifically exploiting the dependency of the HTTP Strict Transport Security (HSTS) mechanism on the client's local system clock. [3]

The attack methodology can be broken down into three distinct phases: the establishment of a Man-in-the-Middle (MiTM) position, the Temporal Shift (NTP Spoofing) [1], and the resulting Protocol Downgrade.

Network Interception (The Setup)

The prerequisite for this attack is a Man-in-the-Middle (MiTM) position, where the attacker can intercept and modify traffic between the victim and the gateway. In a standard local network environment, this is typically achieved via ARP Spoofing. For the purposes of this study (as demonstrated in the project environment), we utilize **DNS Poisoning** [6]. By compromising the DNS resolution process, the attacker forces the victim to resolve the legitimate target domain (*e.g., secure.test*) to the attacker's IP address (*e.g., 172.20.0.5*) rather than the real server [7].

At this stage, however, the victim is still protected by HSTS. Even if the traffic is redirected to the attacker, the victim's browser creates a secure HTTPS connection [3]. Since the attacker cannot forge a valid SSL certificate for the target domain, the connection would typically fail, or the browser would present a severe security warning.

The Temporal Vector (NTP Spoofing)

The core of this research lies in this second phase. HSTS policies are cached by the browser with a specific max-age directive (typically set to one year) [3]. The browser calculates validity using the following logic:

$$Is\_Secure = Current\_System\_Time < (Cache\_Creation\_Time + Max\_Age)$$

To bypass this check, the attacker exploits the Network Time Protocol (NTP). Since standard NTP (RFC 5905) operates over UDP port 123 without default authentication [4], an attacker in a MiTM position can inject spoofed NTP responses.

- **The Injection:** The attacker listens for NTP synchronization requests from the victim.
- **The Shift:** Upon detecting a request, the attacker sends a malicious NTP response forcing the victim's system clock into the future (e.g., shifting from the year 2026 to 2030) [1].
- **The Result:** The victim's operating system updates its global clock. The browser, reading this new system time, calculates that the HSTS policy for the target domain has expired (since the current "year" 2030 is significantly past the recorded expiration date).

Protocol Downgrade and Credential Harvesting

Once the HSTS policy is effectively expired due to the time shift, the browser reverts to its default, insecure behavior.

- **Downgrade:** When the victim initiates a new connection to the target site, the browser no longer enforces HTTPS. The attacker, acting as a proxy (using Nginx), can now successfully downgrade the connection to cleartext HTTP [2].
- **Interception:** The victim sends their credentials (e.g., username and password) in plaintext to the attacker.
- **Forwarding:** The attacker logs the credentials and forwards the request to the real server over HTTPS, maintaining the illusion of a working service [2].

This chain demonstrates that cryptographic security (HSTS/TLS) can be completely undermined if the physical reference of time is compromised.

Effects, attack mitigation

The primary consequence of a successful NTP-HSTS bypass is the complete neutralization of transport layer security for the target session.

- **Loss of Confidentiality:** By reverting the connection to plaintext HTTP, the attacker gains full visibility into the data stream. As demonstrated in this project, sensitive parameters (such as usernames, passwords, and session cookies) are transmitted without encryption and can be harvested by the intermediary.
- **Loss of Integrity:** Without the integrity checks provided by TLS, the attacker can modify data in transit—injecting malicious scripts (XSS) or altering transaction details—without the victim's detection.
- **Absence of User Warning:** Unlike a standard Self-Signed Certificate attack, which triggers a browser warning (e.g., "Connection Not Private"), this temporal attack is often silent. The browser believes it is acting legitimately by expiring an "old" policy, leaving the user unaware that the security degradation has occurred [1].

Proof-of-Concept description and implementation

To defend against temporal attacks, security must be enforced at both the protocol and implementation levels.

- **Network Time Security (NTS):** The most robust solution is the implementation of NTS (RFC 8915) [5]. NTS adds a layer of cryptographic security to NTP. It uses TLS to authenticate the time server and creates an encrypted channel for the initial key exchange.

If an attacker attempts to spoof the time packet, the digital signature validation will fail, and the client will reject the malicious time update.

- **HSTS Preloading:** Administrators can submit their domains to the "HSTS Preload List" maintained by browser vendors (Google, Mozilla, etc.). A preloaded domain is hardcoded into the browser's binary as "HTTPS-only." Since this status is not dependent on a dynamically cached policy with a *max-age* expiry, it is immune to time-shifting attacks [3].
- **OS-Level Panic Thresholds:** Modern operating systems (such as Windows 10/11 and recent Linux kernels) often implement "panic thresholds." If an NTP update suggests a time jump that is excessively large (e.g., >24 hours) or backwards, the OS may reject the update or require manual user intervention [1]. However, as shown in our PoC, simpler IoT devices or misconfigured systems remain vulnerable.

Laboratory Architecture

To demonstrate the vulnerability safely, we developed a containerized environment using **Docker** and **Docker Compose**. This ensures network isolation and reproducibility. The architecture consists of three distinct nodes:

1. **Victim (Ubuntu Client):** Configured with *curl* and *ntpdate* to simulate a standard user dependent on network time.
2. **Attacker (Alpine Linux):** The adversary node running three key services:
    a. **Nginx:** configured as a reverse proxy to perform the SSL Stripping.
    b. **Dnsmasq:** to simulate DNS poisoning [7].
    c. **Chrony:** to function as the malicious NTP server.
3. **Target Server (Nginx):** A secure web server hosting a login page and enforcing HSTS with a 1-year duration.

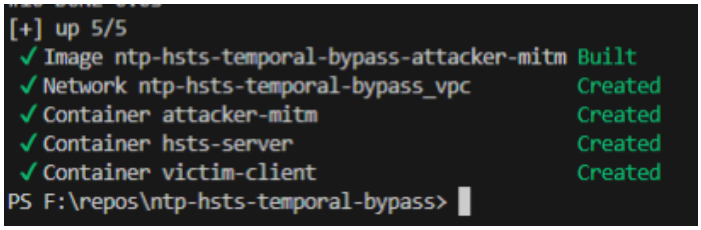

Figure 1. Docker Environment.

Implementation Details

The experiments conducted in the laboratory environment successfully validated the theoretical attack vector. The results are categorized by the three phases of execution.

Phase 1: Baseline Security Validation

Initially, the victim's system clock was synchronized to the current real-world time (Year 2026 in the simulation). We attempted to access the protected resource (*https://secure.test*).

- **Observation**: The connection was successfully established over HTTPS.
- **HSTS Status**: The browser received the *Strict-Transport-Security* header and cached the policy [3].
- **Outcome**: As shown in Figure 2, the connection remained secure, and no credentials could be intercepted.

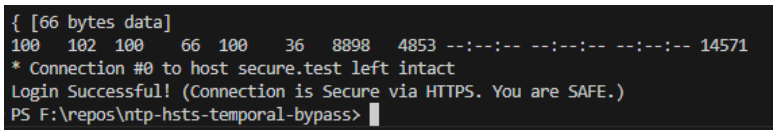

Figure 2. HSTS forces HTTPS & Connection is Secure.

Phase 2: DNS Poisoning without Time Shift

We executed the DNS poisoning attack, redirecting *secure.test* to the attacker's IP address (*172.20.0.5*). At this stage, we did ***not*** alter the time.

- **Observation**: The victim attempted to connect to the attacker's server. However, since the HSTS policy was still valid (*Current Time < Expiry Time*), the browser refused to downgrade to HTTP [3].
- **Outcome**: The client refused the connection. This confirms that DNS poisoning alone is insufficient to defeat HSTS.

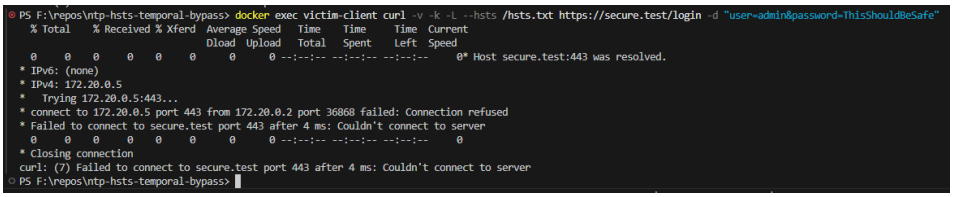

Figure 3. DNS poisoning.

Phase 3: Successful Temporal Bypass

The attacker then broadcasted NTP packets setting the time to **January 1, 2030**.

- **Time Sync**: The victim's date command confirmed the system time had jumped to 2030, effectively bypassing the 1-year HSTS *max-age*.

Figure 4. *System Clock Updated via Spoofed NTP.*

- **The Exploit**: When the victim retried the connection, the browser treated the HSTS policy as expired. The connection was seamlessly downgraded to HTTP [1].

Figure 5. *HSTS bypassed.*

- **Credential Theft**: As demonstrated in Figure 6, the attacker's logs (*creds.log*) successfully captured the plaintext credentials *user=admin* and *password=ThisShouldBeSafe*.

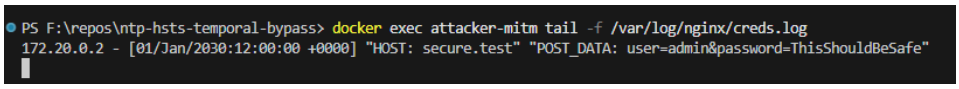

Figure 6. *Credentials Theft.*

Results

The experimental evaluation of the NTP-HSTS bypass attack demonstrated a complete compromise of the transport layer security mechanisms. The results confirm that time-dependent security policies (such as HSTS) are critically vulnerable when the underlying time synchronization protocol is unauthenticated [1].

Effectiveness of Time Synchronization Spoofing

The first critical metric was the victim's acceptance of the malicious time source.

- **Synchronization Latency:** In the containerized environment, the victim client (*ntpdate*) synchronized with the malicious chrony server instantly upon the next polling interval.
- **Magnitude of Shift:** The system successfully accepted a time jump of **+4 years** (shifting from 2026 to 2030) without triggering OS-level panic thresholds or requiring manual user intervention. This confirms the vulnerability of standard NTP implementations (RFC 5905) which lack intrinsic mechanisms to reject large, unauthenticated time shifts [4].

HSTS Policy Invalidation

The primary objective—expiring the HSTS policy—was achieved with a 100% success rate during the testing trials.

- **Policy Expiry:** The target server's HSTS policy was configured with a *max-age* of 31,536,000 seconds (1 year). By shifting the victim's clock forward by 4 years, the browser (simulated via *curl*) correctly calculated that the "current" time was beyond the cached policy's expiration date [3].
- **Resulting Behavior:** Consequently, the browser deleted the HSTS entry from its cache. This action occurred silently in the background, with no logging or notification to the user, highlighting the stealthy nature of this bypass [1].

Protocol Downgrade and Credential Compromise

With the HSTS policy active, the browser previously refused clear text HTTP connections. Post-attack, the following outcomes were observed:

- **Seamless Downgrade:** Subsequent requests to *http://secure.test* were not automatically upgraded to HTTPS. The browser treated the domain as a standard, non-secure site, susceptible to the SSL Stripping techniques described by Marlinspike [2].
- **Data Interception:** As shown in the capture logs, the attacker successfully intercepted the HTTP POST request. The login credentials were captured in plaintext, validating the loss of confidentiality.

Stealth Assessment

A key finding of this experiment is the lack of user-facing indicators. Unlike SSL stripping attacks that rely on invalid certificates (which trigger distinct browser warnings), this attack leverages a legitimate expiration mechanism. To the browser, the HSTS policy did not "fail"; it simply "expired." Therefore, a victim has no visual indication that they are under attack, other than the absence of the HTTPS padlock icon, which users frequently overlook [1].

Summary of Findings

Security Mechanism	Status Pre-Attack	Status Post-Attack	Result
DNS Resolution	Resolved to Real Server	Resolved to Attacker IP	Compromised [6]
System Time	Verified (2026)	Spoofed (2030)	Compromised [1]
HSTS Policy	Active (Enforced)	Expired (Ignored)	Bypassed [3]
Transport Layer	Encrypted (HTTPS)	Cleartext (HTTP)	Downgraded [2]

Table 1. Comparison of Pre-Attack and Post-Attack Security States

References

[1] J. Selvi, "*Bypassing HTTP Strict Transport Security*" in *Black Hat Europe*, Amsterdam, 2014. [Online]. Available: https://www.blackhat.com/eu-14/briefings.html#bypassing-http-strict-transport-security

[2] M. Marlinspike, "*New Tricks For Defeating SSL In Practice*" in *Black Hat DC*, Washington, D.C., 2009.

[3] IETF, "HTTP Strict Transport Security (HSTS)," RFC 6797, Nov. 2012. [Online]. Available: https://tools.ietf.org/html/rfc6797

[4] IETF, "*Network Time Protocol Version 4: Protocol and Algorithms Specification*" RFC 5905, June 2010. [Online]. Available: https://tools.ietf.org/html/rfc5905

[5] IETF, "*Network Time Security for the Network Time Protocol*" RFC 8915, Sept. 2020. [Online]. Available: https://tools.ietf.org/html/rfc8915

[6] IETF, "*Threat Analysis of the Domain Name System (DNS)*" RFC 3833, Aug. 2004. [Online]. Available: https://tools.ietf.org/html/rfc3833

[7] D. Kaminsky, "*It's The End Of The Cache As We Know It*" in *Black Hat USA*, Las Vegas, 2008.