Attacking NTP's Authenticated Broadcast Mode

Aanchal Malhotra
Boston University
aanchal4@bu.edu

Sharon Goldberg
Boston University
goldbe@cs.bu.edu

ABSTRACT

We identify two attacks on the Network Time Protocol (NTP)'s cryptographically-authenticated broadcast mode. First, we present a replay attack that allows an on-path attacker to indefinitely stick a broadcast client to a specific time. Second, we present a denial-of-service (DoS) attack that allows an off-path attacker to prevent a broadcast client from ever updating its system clock; to do this, the attacker sends the client a single malformed broadcast packet per query interval. Our DoS attack also applies to all other NTP modes that are 'ephemeral' or 'preemptable' (including manycast, pool, etc). We then use network measurements to give evidence that NTP's broadcast and other ephemeral/preemptable modes are being used in the wild. We conclude by discussing why NTP's current implementation of symmetric-key cryptographic authentication does not provide security in broadcast mode, and make some recommendations to improve the current state of affairs.

1. INTRODUCTION

The Network Time Protocol (NTP) [10], one of the Internet's oldest protocols, is used to set time on Internet clocks. Time places a crucial and often-ignored role in the security and correctness of computing applications, and especially in cryptographic protocols. As we discussed in [8], an attacker that manipulates time using NTP can seriously undermine the security of key Internet protocols and applications, including TLS certificates [6,11,14], DNSSEC, routing security with the RPKI, authentication with Kerberos, caching, and bitcoin [3]. NTP operates in several modes including (1) client/server, (2) symmetric active/passive, and (3) broadcast/multicast. Our earlier work [8] considered attacks on NTP's client/server mode. In this companion paper, we consider the security of NTP's broadcast mode.

We use network measurements to find that NTP's broadcast mode, which is intended for an environment with a few servers and potentially a large client population, is used by thousands of NTP clients in the wild (Section 5). Next, we show that while symmetric-key cryptographic authentication of NTP broadcast traffic is recommended by the NTP specification [10] and required by the open-source NTP reference implementation *ntpd*, it does not provide sufficient protection against attacks on broadcast mode. We consider both (1) on-path attacks, where the attacker occupies a privileged position on the path between NTP client and one of its servers, and (2) off-path attacks, where the attacker can be anywhere on the network and does not observe the traffic between client and any of its servers. We present an on-path replay attack on authenticated broadcast mode (CVE-2015-7973) that causes the NTP client to get stuck at a particular time (Section 3), and a new off-path denial-of-service attack on authenticated broadcast mode (CVE-2015-7979) that also applies to all of NTP's "preemptable" and "ephemeral" modes of operation (Section 4). We conclude by discussing the inherent challenges of cryptographically authenticating NTP's broadcast mode, and provide several recommendations for the way forward (Section 6).

2. NTP'S BROADCAST MODE

NTP clients and servers are not configured to operate in broadcast mode by default on most operating systems. However, there is a configuration option that allows for this mode of operation.

Broadcast servers. An NTP broadcast server can be preconfigured to periodically send '*persistent*' broadcast-association server packets (*NTP mode 5 packets*) to the clients on the broadcast network. By *persistent*, we mean the server mobilizes the broadcast association upon initialization, and never demobilizes the association [10]. Figure 1 presents a sample NTP mode 5 broadcast packet.

Broadcast clients. An NTP client can be preconfigured to accept NTP mode 5 packets.[1] When a broadcast client receives its first NTP mode 5 packet, the client must first calculate the propagation delay by exchanging a volley of client/server mode packets with the broadcast server—where the client sends the server an NTP mode 3 query and the server responds with an NTP mode 4 response.[2] After this, the client reverts to broadcast client mode, and creates an *ephemeral* association with the server upon receipt of further mode 5 broadcast packets. An *ephemeral* association is mobilized upon arrival of a packet and exists until error or timeout [10].

[1]The configuration option *broadcastclient* or *multicastclient [address]* allows an ntpd client to receive and process mode 5 broadcast packets. Note that a client configured to accept multicast messages from a particular address also accepts broadcast messages from ANY address.

[2]The server and client also run the Autokey security protocol, if they are configured to do so. Autokey [5] is public-key authentication method for NTP, but NTP clients do not request Autokey associations by default [1], and many public NTP servers do not support Autokey (*e.g.,* servers in `pool.ntp.org`). In fact, a lead developer of the ntpd client wrote in 2015 [16]: "Nobody should be using autokey. Or from the other direction, if you are using autokey you should stop using it." We therefore do not consider Autokey any further here.

Authenticating an association. How does an NTP client validate incoming packets before establishing an association with a server? Most NTP traffic (especially client/server-mode traffic) is not cryptographically authenticated[3] However, even in the absence of cryptographic authentication, NTP clients running in client/server mode or symmetric active/passive mode use a *nonce* to validate a server's response. The nonce is a field in the NTP packet, called the *origin timestamp*; see Figure 1. Upon receipt of an NTP response packet, the client checks if the 64-bit *transmit timestamp* field in the most recent query packet it sent the server, matches the 64-bit *origin timestamp* field in the incoming response packet. This is called *TEST2* in the NTP specifications. This non-cryptographic authentication is based on the premise that the nonce has enough entropy such that an *off-path* attacker, who can not see the NTP packets in transit, cannot guess the nonce. Indeed, as we argued in [8], we can safely assume that this nonce has about 32 bits of entropy, and so it is difficult to forge from off path.

Authenticating broadcast. In contrast to the client/server mode, where the client actively sends a query to the server to get the response, the broadcast client operates in *listen-only* mode. Thus, because the client does not send the server any queries for broadcast packets, and the *origin timestamp* field in the broadcast server packet is always set to *null*. So now *TEST2* that defends NTP's client-server mode from off-path attacks does not apply. Moreover, the most recent NTP reference implementation (ntpd v4.2.8p4) does NOT use UDP source port randomization [7], and so an off-path attacker can easily forge an unauthenticated mode 5 packet.

Also, an NTP client preconfigured to run in broadcast client mode will accept and process packets from ANY server that sends it broadcast packets; it is NOT configured to listen only to one particular broadcast server. So any off-path attacker can easily send broadcast messages and the client will accept them. RFC5905 [10, pg 57] says:

> Filtering can be employed to limit the access of NTP clients to known or trusted NTP broadcast servers. Such filtering will prevent malicious traffic from reaching the NTP clients.

To fill this gap and the lack of nonce check, RFC 5905 [10] strongly suggests the use of cryptography to authenticate broadcast packets. Indeed, NTP's current reference implementation (ntpd v4.2.8p4) requires *symmetric-key* cryptography, by default, for clients that wish to listen to broadcast mode packets. NTP's symmetric cryptographic authentication appends an MD5 hash keyed with symmetric key k of the NTP packet contents m as MD5($k||m$) [11, pg 264] to the NTP packet in Figure 1; authenticated NTP packets also have a 32-bit *key ID* which is used to identify the symmetric key that was used to authenticate the message.

In this paper, however, we present two attacks that show that NTP's symmetric key cryptography does not provide sufficient protection for broadcast mode.

[3]As we discussed in [8], NTP's symmetric-key cryptography is not commonly because the symmetric keys must be preconfigured manually ; this can be quite cumbersome for public servers that must accept queries from arbitrary clients. (NIST, for example, distributes symmetric keys for its public servers via US mail or facsimile [2].) Moreover, NTP's public-key cryptography (Autokey) is not recommended for use in the wild, see footnote 2.

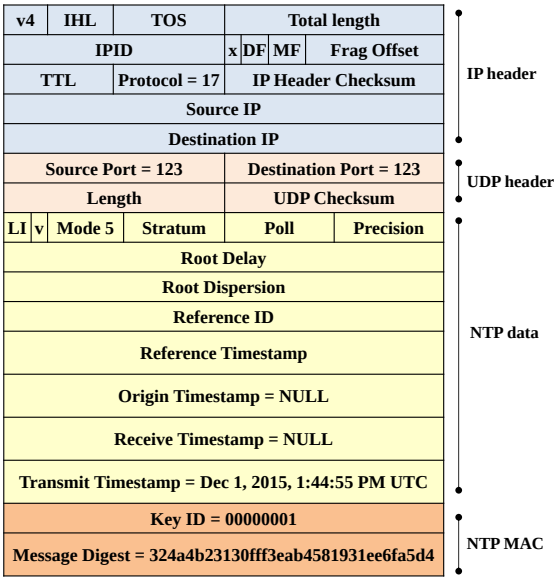

Figure 1: Mode 5 NTP Broadcast Packet.

3. TIMESHIFTING ATTACKS

Should broadcast be robust to replay attacks? According to RFC 5905 [10], NTP's "on-wire protocol ... resists replay of a server response packet." This is supposed to be accomplished through what is called *TEST1* in the protocol specification: Upon receipt of an NTP response packet, the NTP client matches the transmit timestamp in the current packet to that of the last response packet it received; if the timestamp matches, it marks the packet as duplicate and drops it. However, we now show that because broadcast mode does not impose TEST2, then TEST1 cannot provide sufficient protection against replay attacks, even when NTP packets are cryptographically authenticated.

a) Deja Vu: Our on-path time-sticking attack (CVE-2015-7973). Consider a man-in-the-middle (MiTM) attack, where the attacker is positioned between the server and the victim client, and can intercept and replay a packet and prevent onward transmission of the original packet, but does not possess the symmetric key that authenticates broadcast messages. We show that the protocol does not resist the following replay attack. The MiTM collects and records a contiguous sequence of server broadcast packets. (The attacker requires a sufficient number of these packets for the client to update its clock; this is because NTP requires a client to obtain between eight to hundreds of messages from a server before the client's *clock discipline algorithms* synchronizes it to the server [10, Sec. 10-12].) He then replays this sequence of packets, over and over, to the victim client; the victim accepts the same time over and over, and thus gets stuck at a particular time. Notice that these are the authenticated packets from the broadcast server, and so they pass the authentication check on the client. Moreover, by replaying a *sequence* of packets, rather than just one packet, the attacker ensures that the replayed packets pass *TEST1*.

b) Our off-path time-shifting attack. If the attacker is one of the clients on the broadcast network, or on an adjacent network that also gets the broadcast packets from the

same broadcast server, it then shares the same symmetric key with the server as the victim client. In this case, the attacker possesses the same key as the server and therefore can simply forge authenticated NTP mode 5 packets and send them to the victim client. The attacker can then send the victim back/forward in time as discussed in [8], or can make him stick to a particular time.

Why do these attacks work? These attacks highlight the following weaknesses in NTP; a) The protocol specifies and defaults to the use of symmetric key cryptography for broadcast authentication, where all nodes share the same key and one/some of them could be malicious or may be compromised, b) an NTP client is unable to recognize that it is stuck in a particular time for long periods of time, and c) in the absence of *TEST2*, *TEST1* doesn't actually prevent replay in general—it just prevents replay of the most recent packet. Our replay attack passes *TEST1* because the client only matches the current transmit timestamp with that of the last packet.

Experiments: As a proof-of-concept, we set up an ntpd v4.2.8p3 broadcast client and server using the configuration options *broadcastclient* and *broadcast IP_address_range*. Another machine on the same network behaves as MiTM and collects 12 *mode 5* packets and stores them. The MiTM then drops the original *mode 5* packets to the victim and replays his previously collected set of *mode 5* packets. The victim accepts the time after getting sufficient samples required for a server to pass the clock discipline algorithms, gets into the 'STEP' mode[4]and clears the state variables for this association. We continue this experiment for ≈ 4 hours and observe that the victim's system clock is stuck at the same time.

Implications of the attack. A MiTM can use a replay attack to make the victim client get stuck at a particular time value forever. Moreover, a compromised machine on the same or adjacent subnet can or forge authenticated *mode 5* packets and shift time forward or backward on the victim client. Shift time forward/backward has severe implications on security guarantees provided by various core Internet protocols, such as DNSSec, BGP, TLS, and authentication services that use Kerberos; see [8] for discussion.

4. DENIAL OF SERVICE ATTACKS

We now present an off-path denial-of-service attack that generically succeeds on any preemptable or ephemeral association that is cryptographically authenticated, including authenticated broadcast mode.

Preemptable and empheremal associations. NTP's broadcast clients use an *ephemeral association* to listen to NTP mode 5 from a broadcast server; as discussed in Section 2, this association is automatically demobilized upon error or timeout.

NTP also supports *preemptable* associations [9], which are similar to ephemeral associations. Preemptable associations are mobilized if the ntpd client has the keyword "preempt" to the line in its configuration file that establishes a association with a particular server. Alternatively, the ntpd client may be preconfigured with the *manycastclient* or *pool* [*pool_address*] options; in this case, the client establishes a preemptable association upon receipt of a server discovery packet. Preemptable associations are also demobilized upon error or timeout.

Our off-path DoS attack (CVE-2015-7979). An off-path attacker can easily cause an error by sending *mode 5* with bad cryptographic authentication (*e.g.,* wrong key, mismatched key, incorrect message digest, *etc.*). The attacker sends one such error-causing packet for every legitimate response the client receives from the server, so that the client immediately tears down its association with the server. This way, the client never collects enough good NTP response to allow its clock discipline algorithms to update its local clock, resulting in a denial-of-service attack on the client.

Experiment. As a proof-of-concept, we set up ntpd v4.2.8p3 broadcast client and server using the configuration options *broadcastclient* and *broadcast IP_address_range* respectively. Once the client is synchronized with the broadcast server, another machine which behaves as an off-path attacker sends badly-authenticated *mode 5* packet to the client. The client immediately tears down the association with the server and clears all the state variables. Next, the client receives the legitimate packet from the broadcast server and again mobilizes the association. The attacker again sends the bad *mode 5* packet and the client again tears down the association. The attacker keeps repeating this and the client never obtains enough consistent time samples from the server to allow it to update its system clock.

Implications a) An off-path attacker can deny NTP service to the broadcast client even when it uses cryptographic authentication. b) If the client is preconfigured to a bad timekeeper or one of the servers' that the client is configured to is controlled/compromised by the attacker, then using this DoS attack, the client can pin the client to bad server that is controlled by him. The attacker can then send the client back/forward in time which has implications as mentioned in Section 3.

5. MEASUREMENT RESULTS

We use NTP's *peers* command to check for the presence of broadcast and other ephemeral and preemptable modes in the wild. As shown in Figure 2, NTP's *peers* command returns a list of all associations used by an NTP client; associations with a broadcast server are marked with a *b* or *B*, client/server associations are marked with a *u*, *etc.*. '*' is used to indicate the association that the client last took time from, and '+' indicates an association that is a candidate for synchronization. While this command provides a variety of useful information for network measurement, it's also a great tool for adversarial network reconnaissance and DDoS amplification attacks [4]; for this reason, network operators commonly disable remote *peers* queries, or configure firewalls or other middleboxes to drop them. Moreover, while we conjecture broadcast mode is most common when both the clients and the broadcast server are behind a NAT, we are unable to scan clients behind a NAT. Therefore, it's important to remember that our measurement results can only provide a lower bound on the number of broadcast/ephemeral/preemptable associations in the wild.

Our scan. Thus, we scanned the entire IPv4 address space using the *peers* command on 10-11 November 2015,

[4]An NTP client enters 'STEP' mode whenever it needs to shift its clock by more than 125ms but less than ≈ 16 min; our replay attack shifts the client back in time by more than 125ms, causing the client to enter STEP mode.

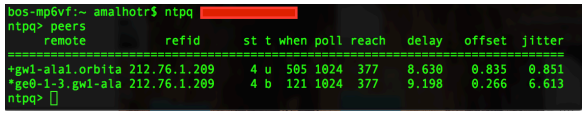

Figure 2: Sample response to *peers* query.

and obtained responses from 4,443,118 IPv4 addresses. On 16-21 November 2015 we rescanned only the 4.4M responding IP addresses with NTP's *peers* command, as well the *rv* command (which reveals useful information about the NTP client, including its version, build date, and the operating system it runs on), and the *as* command (which has useful information about each association used by the client). For this second scan we obtained responses from 3,716,362 IPv4 addresses; we consider only these addresses here.

Results. Of the 3.7M responding IPs, we found that 18,020 (0.4%) IPs have at least one broadcast association, and 1,767 IPs use multicast associations. (Recall that clients configured for multicast will also accept broadcast associations.) Moreover, we see 9,806 IPs that use broadcast associations exclusively, of which 7,556 IPs were synchronized to a broadcast server, while the remaining 2,250 were unsynchronized and thus likely malfunctioning.

As an aside, we were also surprised to find many symmetric associations in the wild; 190,724 (5.1%) of the responding IPs had at least one symmetric association. Overall, we found 2,848,238 IPs that have at least one client/server associations, 77 IPs use multicast exclusively, 67383 IPs use symmetric associations exclusively, and 9806 use broadcast exclusively. This is a total of 2,925,504 (78.7%) IPs; for the rest of the IPs, their association status is "-" which may mean they are initializing, or using a local clock (*e.g.,* via GPS) rather than taking time from the Internet using NTP. Thus, while broadcast is not an especially popular mode of operation for NTP, we do find thousands of clients in the wild that rely upon it for time synchronization.

Who are these broadcast clients? Of the 18K IPs that have at least one broadcast association, 16,552 of them also responded to NTP's *rv* query, and thus reveal information about their operating systems, ntpd version, and compile date. Most of these broadcast clients are running on unix (10,671 IPs) or 'cisco' devices (5,135 IPs). Also, out of 16.5K IPs that responded to the *rv* query, only 326 replied with the detailed ntpd version and compilation details (the rest merely say "ntpd version 4"). Of these, the majority (212 IPs or 65%) of these have been compiled between 2012 and 2015 inclusive. The most popular ntpd version that we found is 4.2.6p5@1.2349 (23%) which was released in December 2011, closely followed by 4.2.8@1.3265 (20%) which was released in December 2014, while 4.1.1c-rc1@1.836 (released in 2001) and 4.2.4p5-a (released in 2008) are 9% each. The bottom line is that we do find evidence of recently maintained NTP implementations that use broadcast mode in the wild.

6. RECOMMENDATIONS

We have several recommendations to mitigate our attacks.

1) Ephemeral and pre-emptable associations considered harmful. Our denial-of-service attack from Section 4 points to a serious problem with the notion of ephemeral and preemptable associations; namely, that an off-path attacker can easily forge a packet that can tear down an association. Even though an ephemeral association can easily be reestablished, the attacker can quickly tear it down again before the client has the chance to update its clock. For this reason, we suggest that NTP does NOT tear down ephemeral associations upon receipt of a malformed packet; instead, the malformed packet should just be dropped, while the association remains in place.

2) Prevent replay in broadcast mode. As others have pointed out [12, 15], NTP's broadcast mode should contain a robust mechanism for preventing replay attacks. TEST1 is insufficient, since it only checks if the most recent packet has been replayed. One solution is to require authenticated mode 5 NTP packets to include an incrementing counter (*e.g.,* in the extension field). Another idea is to use the transmit timestamp field in the mode 5 response packet (Figure 1) as an incrementing counter; to do this, the broadcast client would need to ensure that the transmit time is monotonically increasing. Alternatively, the MAC computed on the broadcast packet could become a hash chain; that is, the MAC on packet p_i should be computed over the contents of packet p_i concatenated with the MAC on packet p_{i-1}.

3) Monitor to detect time-sticking. In the absence of replay protection, monitoring could be used as a "band-aid" solution against time-sticking attacks. That is, NTP clients could monitor their own clocks to see if they are stuck at the same timestamp for a considerable amount of time; if so, they could log an error or alert to warn an admin.

4) Only the broadcast server should be able to sign broadcast packets. As others have pointed out [12, 15], the broadcast servers' symmetric key should NOT be distributed to all its client; as we noted in Section 3, this allows clients to trivially forge packets from the server. Instead, the only entity that should be able to sign broadcast (mode 5) packets is the broadcast server itself.

7. CONCLUSION

Our analysis highlights the difficulty of designing robust cryptographic authentication for NTP's broadcast mode. One might be tempted to dismiss broadcast mode altogether, by arguing that broadcast mode is a legacy from the past that is largely unused today. Our measurements, however, indicate that this is not the case; while broadcast mode is not especially popular, we do find thousands of NTP clients in the wild that have broadcast associations. Thus, we believe that the community should take another careful look at authentication for NTP's broadcast mode.

One approach, taken by a new Internet draft for the "Network Time Security protocol (NTS)", achieves recommendations (2) and (4) above through a modified version of TESLA [13]. TESLA uses public-key cryptography to ensure that a server is the only entity that can authenticate broadcast messages, but that the authenticators themselves can be computed and validated using fast symmetric cryptography. To do this, however, TESLA requires loose time synchronization between the broadcast server and its client. Thus, using TESLA in the context of NTP creates a circular dependency on time. NTS suggests avoiding this circular dependency by using an authenticated unicast association to achieve the loose synchronization between the server and

each of its clients. This, however, once again requires pairwise associations between server and each client, and may defeat the purpose of using the broadcast mode in the first place. Finding a cryptographic solution that can authenticate NTP's broadcast mode, without a unicast association or a circular dependency on time, remains an interesting open problem.

Acknowledgements

We thank Matt Van Gundy, Jonathan Gardner, Matthew Street, Stephen Gray and Hai Li from Cisco's ASIG team for useful discussions and assistance with the responsible disclosure of these results, Jared Mauch and the openNTP-project for performing the initial scan of the IPv4 address space with NTP's *peers* command, and David Mills for suggesting we look into the security of broadcast mode. This research was supported, in part, by NSF awards 1347525, 1414119 and 1012910 and a gift from Cisco.

8. REFERENCES

[1] Autokey Configuration for NTP stable releases. The NTP Public Services Project: `http://support.ntp.org/bin/view/Support/ConfiguringAutokey` (Accessed: July 2015).

[2] The NIST authenticated ntp service. `http://www.nist.gov/pml/div688/grp40/auth-ntp.cfm` (Accessed: July 2015), 2010.

[3] corbixgwelt. Timejacking & bitcoin: The global time agreement puzzle (culubas blog), 2011. `http://culubas.blogspot.com/2011/05/timejacking-bitcoin_802.html` (Accessed Aug 2015).

[4] J. Czyz, M. Kallitsis, M. Gharaibeh, C. Papadopoulos, M. Bailey, and M. Karir. Taming the 800 pound gorilla: The rise and decline of NTP DDoS attacks. In *Proceedings of the 2014 Internet Measurement Conference*, pages 435–448. ACM, 2014.

[5] B. Haberman and D. Mills. *RFC 5906: Network Time Protocol Version 4: Autokey Specification*. Internet Engineering Task Force (IETF), 2010. `https://tools.ietf.org/html/rfc5906`.

[6] J. Klein. Becoming a time lord - implications of attacking time sources. Shmoocon Firetalks 2013: `https://youtu.be/XogpQ-iA6Lw`, 2013.

[7] M. Larsen and F. Gont. *RFC 6056: Recommendations for Transport-Protocol Port Randomization*. Internet Engineering Task Force (IETF), 2011. `https://tools.ietf.org/html/rfc6056`.

[8] A. Malhotra, I. E. Cohen, E. Brakke, and S. Goldberg. Attacking the network time protocol. *NDSS'16*, February 2016.

[9] D. Mills. Association management. `https://www.eecis.udel.edu/~mills/ntp/html/assoc.html`.

[10] D. Mills, J. Martin, J. Burbank, and W. Kasch. *RFC 5905: Network Time Protocol Version 4: Protocol and Algorithms Specification*. Internet Engineering Task Force (IETF), 2010. `http://tools.ietf.org/html/rfc5905`.

[11] D. L. Mills. *Computer Network Time Synchronization*. CRC Press, 2nd edition, 2011.

[12] T. Mizrahi. *RFC 7384 (Informational): Security Requirements of Time Protocols in Packet Switched Networks*. Internet Engineering Task Force (IETF), 2012. `http://tools.ietf.org/html/rfc7384`.

[13] A. Perrig, R. Canetti, J. D. Tygar, and D. Song. The tesla broadcast authentication protocol. *RSA CryptoBytes*, 5, 2005.

[14] J. Selvi. Breaking SSL using time synchronisation attacks. *DEFCON'23*, 2015.

[15] D. Sibold, S. Roettger, and K. Teichel. *draft-ietf-ntp-network-time-security-10: Network Time Security*. Internet Engineering Task Force (IETF), 2015. `https://tools.ietf.org/html/draft-ietf-ntp-network-time-security-10`.

[16] H. Stenn. Antw: Re: Proposed REFID changes. NTP Working Group Mailing List `http://lists.ntp.org/pipermail/ntpwg/2015-July/002291.html`, July 2015.