TARP: Ticket-based address resolution protocol

Wesam Lootah, William Enck *, Patrick McDaniel

Systems and Internet Infrastructure Security Laboratory, Department of Computer Science and Engineering, The Pennsylvania State University, 344 IST Building, University Park, PA 16802, United States

Received 14 May 2006; received in revised form 29 March 2007; accepted 17 May 2007
Available online 15 June 2007

Abstract

IP networks fundamentally rely on the Address Resolution Protocol (ARP) for proper operation. Unfortunately, vulnerabilities in ARP enable a raft of Internet Protocol (IP)-based impersonation, man-in-the-middle, or Denial of Service (DoS) attacks. Proposed countermeasures to these vulnerabilities have yet to simultaneously address backward compatibility and cost requirements. This paper introduces the Ticket-based Address Resolution Protocol (TARP). TARP implements security by distributing centrally issued secure IP/Medium Access Control (MAC) address mapping attestations through existing ARP messages. We detail TARP and its implementation within the Linux operating system. We also detail the integration of TARP with the Dynamic Host Configuration Protocol (DHCP) for dynamic ticket distribution. Our experimental analysis shows that TARP improves the costs of implementing ARP security by as much as two orders of magnitude over existing protocols. We conclude by exploring a range of operational issues associated with deploying and administering ARP security.

© 2007 Elsevier B.V. All rights reserved.

Keywords: Network security; ARP security

1. Introduction

The Address Resolution Protocol (ARP) [46] is the glue that holds together the network and link layers of the Internet Protocol (IP) stack. The primary function of ARP is to map IP addresses onto hardware addresses within a local area network. As such, its correctness is essential to proper functioning of the network. However, like other protocols within IP, ARP is subject to a range of serious and continuing security vulnerabilities [14,15]. Adversaries can exploit ARP to impersonate hosts, perform man-in-the-middle attacks, or simply launch a Denial of Service attack. Moreover, such attacks are trivial to perform, and few countermeasures have been widely deployed. Current network environments present two central design challenges for ARP security. Firstly, the solution must not require ARP be discarded. The deployed base of IP is large and diverse enough that replacing any major component of the IP stack is not only technically challenging but also cost prohibitive. Secondly, the
costs of implementing ARP security must be minimal. Resource constrained devices and already computationally loaded hosts cannot afford to budget large amounts of resources for ARP security. Any solution that would demonstrably change the performance profile of ARP will not be adopted. The primary reason that proposed solutions [16,26,30,33] have not been widely deployed is that they have yet to simultaneously address these two requirements.

In this paper, we introduce the *Ticket-based Address Resolution Protocol* (TARP) [34]. TARP implements security by distributing centrally generated IP/MAC address mapping attestations [7,50]. These attestations, called tickets, are given to clients as they join the network and are subsequently distributed through existing ARP messages. Unlike other popular ARP-based solutions, the costs per resolution are reduced to one public key validation per request/reply pair in the worst case. As such, TARP is a feasible approach for the diverse assortment of existing network capable devices. We provide a detailed description of the protocol design and integration with the Dynamic Host Configuration Protocol (DHCP) [21] for dynamic ticket distribution, as well as its implementation within the Linux operating system. Our experimental analysis shows that TARP retains compatibility while reducing the request costs by as much as two orders of magnitude over existing protocols. We explore a range of crucial operational issues including revocation and incremental deployment and show how TARP can be deployed with limited administrative oversight.

Note that TARP embodies a central design tradeoff. Ticket generation costs grow at the linear inverse of the ticket’s lifetime. The ticket lifetime dictates the vulnerability to replay attacks. Hence, administrators can directly control cost and security through the selection of ticket lifetime. The ability to balance between these competing factors is a central benefit to TARP’s design. We explore the management of this tradeoff throughout and reflect on the necessity of such compromises in the practical use of security technologies.

Security in resolution services remains an open problem. Whether resolving domain or hostnames [9,10,25], or claiming address ownership [7,50], one needs to authenticate the contents and freshness of received data. This work represents a new point in the design space of these services. As such, it can be used to determine the specific costs and advantages of resolution services. In particular, our practical analysis indicates that for certain kinds of resolution, great performance gains can be achieved by slightly relaxing security requirements.

We begin in the next section by providing background on ARP and consider the vulnerabilities inherent in its current design. Section 3 overviews past efforts of securing ARP and other related works. Section 4 details the TARP architecture and its operation within local networks. Section 5 outlines the implementation of TARP within Linux. Section 6 explores the performance of TARP. Section 7 considers several operational issues associated with the use of TARP. Section 8 concludes.

### 2. Background

The *Address Resolution Protocol* (ARP) [46] is used by hosts to map IP addresses onto *Medium Access Control* (MAC) link layer addresses. The resulting address associations are used to direct packet delivery within the physical local network.

Every packet in an IP network must be delivered to some interface in a local network. Those whose destination IP addresses are external to the local network (as determined by the subnet mask) are delivered to the subnet gateway, while packets destined for the internal network are delivered directly. Whether the destination address is local or external, the IP address must be mapped onto a MAC address. ARP resolution performs a distributed lookup via a simple broadcast request followed by a unicast response. The querying host sends the request to the local broadcast address. According to the protocol, only a host assigned to the requested IP address should reply with its local hardware address. This reply, containing both the requested IP address and associated MAC address, is sent directly to the querying host. The host caches the association, which expires and is evicted at a later time as determined by local policy. Once an entry is evicted, the host no longer knows that address association, and the resolution process repeats as required. While the cache hold time for a response is undefined in the protocol specification, many implementations set the expiration to

---

1 We consider an alternate design in a section on revocation in which we address replay vulnerabilities.
approximately 20 min, with the option of resetting the expiry timer after each use [16].

Since hosts implicitly trust the address associations residing in the ARP cache, if an adversary can influence these values, the host can be manipulated into sending packets to the wrong hardware address. The lack of authentication of address association data leaves hosts susceptible to reply spoofing and cache entry poisoning, commonly referred to as cache poisoning. In fact, freely available tools are designed to exploit these vulnerabilities [51].

Most IP stacks are designed to ignore unsolicited ARP replies. However, this does little to prevent cache poisoning. An adversary can coerce a host into requesting a specific address by spoofing an Internet Control Message Protocol (ICMP) ping message. The spoofed message contains the targeted IP address, requiring the host to resolve the MAC address to reply. By spoofing replies and poisoning the cache of network hosts, an adversary can perform both Denial of Service (DoS) and Man in the Middle (MITM) attacks [24]. Although such attacks were identified a decade and a half ago [14], they still exist today [15].

Cache poisoning can be used to mount various types of DoS attacks. In the simplest case, the adversary replaces the MAC address of a particular host with another value. When the victim attempts to communicate with that remote host, all traffic is sent to the wrong MAC address. This effectively denies service to the remote host. If this remote host happens to be the gateway, the host will be unable to communicate with hosts outside of the subnet. Finally, if the adversary knows the IP address of all nodes on the subnet, cache entries can be crafted so that the victim cannot communicate with any remote hosts.

While DoS is a serious concern, cache poisoning resulting in a MITM attack is more dangerous. This attack, as shown in Fig. 1, not only allows the adversary to insert messages into the communication channel, but more importantly, it often goes undetected. Furthermore, cache poisoning used in this manner allows eavesdropping even on a layer-2 switch. In order to launch this attack, the adversary must effectively manipulate the caches at both ends of a conversation. Once both ends believe the adversary is the correct remote destination, manipulating packet streams is trivial.

3. Related Work

3.1. External solutions

Several attempts have been made to address the security issues posited above. Some attempts use methods external to the protocol. These methods do not use cryptography and do not modify the protocol in any way. For example, it has been proposed that hosts be configured with static ARP entries [1]. While this is viable for very small and static networks, it leads to considerable administrative overhead for large dynamic networks.

Port security [17], a security feature available in recent switches, restricts the use of physical ports to configured MAC addresses. This approach only prevents certain kinds of MAC hijacking, but does nothing to prevent MITM attacks. Hence, it is only a partial (and in many ways limited) solution.

Dynamic ARP Inspection (DAI) [19], a security feature available on some recent network switches, is another solution external to the protocol. DAI provides security by preventing invalid or malicious ARP packets from being forwarded on the network. When DAI is enabled on a network switch, the switch inspects ARP packets before forwarding them. ARP packets are compared to a database of valid IP-to-MAC address bindings. If an ARP packet contains a valid binding, it is forwarded; otherwise, it is dropped. Clearly, key to the proper
operation of DAI is the IP-to-MAC address binding database. A system administrator can maintain this database manually, or preferably it can be built using DHCP snooping [18]. DHCP snooping, also a security feature, can be used in concert with DAI. With DHCP snooping, the port on which the network DHCP server is connected is labeled “trusted,” while all other ports are labeled “untrusted.” Only DHCP server packets that originate from a “trusted” port are forwarded, while all other DHCP server packets are dropped. When DHCP snooping is enabled on a switch, it acts as a firewall for DHCP traffic. It also builds and maintains a database of IP-to-MAC address bindings from information in DHCP packets. This database can then be used with DAI to provide security against ARP-based attacks.

Although DAI with DHCP snooping can provide effective security against ARP based attacks, it is not widely available on commodity layer-2 switches. Moreover, the overhead of inspecting packets at layer-2 can be prohibitive. The overhead of deep inspection increases latency and has a negative impact on throughput. It is also important to note that DAI and DHCP snooping are not available for wireless environments, where physical port security is not available due to the shared medium. However, this technique is applicable for wireless networks that create virtual ports, e.g., IEEE 802.1X [4]. Note that 802.1X alone does not solve the ARP security problem; it simply authenticates hosts that attach to the network. In many cases, even authenticated hosts pose a significant threat.

A final class of external solutions attempt to detect misbehavior, rather than prevent it. ARP-Watch [33], a network-level detection device, detects malicious ARP packets by monitoring MAC/IP address pairings occurring on a subnet. Conversely, host-level detection services differ in that each host on the network attempts to detect malicious messages arriving at the local interface [52]. This is achieved by detecting duplicate and/or unsolicited ARP packets. Detection techniques are punitive by definition, and hence are of limited utility in many environments.

### 3.2. Cryptographic solutions

A number of cryptographic protocols have targeted security issues in ARP. In the Secure Link Layer (SLL), all link layer traffic is authenticated and encrypted. While this prevents authorized hosts from injecting malicious messages, it does not prevent authorized, but untrustworthy hosts, from injecting malicious messages. SLL is an example of a solution that provides message authenticity but no proof of IP address ownership.

In another approach, Gouda and Huang [26] propose the Secure Address Resolution Protocol. A secure server in this protocol shares secret keys with each host on a subnet. The server maintains a database of IP-to-MAC address mappings that is updated periodically through communication with each host. All ARP requests and replies occur between a host and the server, and replies are authenticated using the shared pair keys. Note that the server represents a singular point of failure and congestion, which make it a poor match for most networks.

Another ARP security scheme, S-ARP [16], depicted in Fig. 2, uses asymmetric cryptography to provide authenticity of ARP replies. Hosts use self-generated public/private key pairs certified by a local trusted party. Each host registers its public key with the Authoritative Key Distributor (AKD) server. The server’s public key and MAC address are also securely distributed to all hosts during a bootstrapping process. S-ARP requests proceed as normal ARP requests. However, S-ARP replies are signed by the sender’s private key. Upon receiving a reply, the signature is verified using the sender’s public key. If the receiver does not have the sender’s public key, or if the signature cannot be verified by the keys currently in its key ring, the public key of the sender is requested from the AKD. The AKD sends it to the requesting host in a signed message. If the new public key verifies the signature, the reply is accepted and the public key is cached; otherwise, it is rejected. To avoid replay attacks, messages are time-stamped and synchronization messages are exchanged with the AKD.

![Fig. 2. S-ARP address resolution. Alice broadcasts a request for Bob’s MAC address. Bob sends Alice a signed reply. Alice requests Bob’s public key from the AKD. The AKD responds with a signed reply.](image-url)
S-ARP requires, at minimum, a single signature generation and verification per address resolution. As illustrated in Section 6, this cost can be significant.

3.3. Neighbor discovery

Neighbor Discovery (ND) [43] bridges the link and network layer in IPv6. Among other tasks, ND performs address resolution. Similar to an ARP request, an ND host sends a multicast Neighbor Solicitation (NS) to determine the MAC address of a peer. Upon receiving an NS, an ND host replies with a Neighbor Advertisement (NA) asserting the association between its IP and MAC addresses. Unfortunately, the ND protocol specification assumes no malicious hosts on the local network and therefore is susceptible to the same shortcomings as ARP.

Since its inception, there have been a number of suggestions to secure Neighbor Discovery. IPv6 relies on ND to perform a number of tasks in addition to address resolution, e.g., router discovery. We focus on those mechanisms related to securing address resolution in ND.

The initial Neighbor Discover RFC [43] suggested using IPsec for message authentication. IPsec [31] requires IKE [28] to establish keys required to set up security associations between hosts. However, IKE uses IP addresses, therefore, key negotiations cannot occur until each host knows the MAC address of its peer. Hence, IKE does not work for establishing security associates for ND messages. Short of manually configuring security associations on all hosts, ND cannot use IPsec to secure address resolutions.

To address the security problems with ND, Kempf et al. propose using Address Based Keys (ABK) [30], which uses identity-based cryptography. In this scenario, IP addresses are used as public keys. However, contemporary identity-based systems require one or more heavyweight cryptographic operations per signature generation or validation. Hence, their cost is prohibitive for many resource poor devices. Furthermore, ABK is not appropriate for IPv4 due limited address space. If IP addresses are reassigned and the public cryptographic parameters remain unchanged, multiple hosts have knowledge of the private key.

Cryptographically Generated Addresses (CGA) [13] has also been proposed to address the security in Neighbor Discovery. In CGA, a host created public key and network address are combined with additional parameters and hashed to create the interface portion (lower 64 bits) of the IPv6 address. This provides immediate binding between a host’s IP address and its public key, which allows a host to claim ownership of an IP address by demonstrating that it posses the private/public key pair associated with that IP address.

Secure Neighbor Discovery (SEND) [12] was designed to provide security to ND. SEND includes support for CGA. Along with nonce and RSA options, secure address resolution is addressed. When using SEND, the target host includes a CGA parameters option, a nonce option, and an RSA option in the NA message. The RSA option includes a signature that is generated using the host’s private key. The host receiving the reply validates it by first validating the CGA parameters option, which insures that the target address was cryptographically generated using the included public key. Second, the nonce is checked for freshness and finally the signature in the RSA option is verified. If the reply is valid, the host uses the address association to direct messages to the target host.

SEND relies on CGA, which has undergone revision since its proposal. Initially, the public key was hashed and truncated to produce the lower 62 bits of the IPv6 address. As indicated by Arkko et al. [11], this technique is susceptible to brute force attacks, and the current CGA RFC [13] defines a more complex transform. IPv4 addresses are only 32 bits. Even if CGA used the full 32 bits, it would be vulnerable to brute force attacks.

The above mentioned solutions are either partial or limited in scope. None of the solutions simultaneously address both the compatibility and performance requirements of current networks. As we will show in the following section, TARP successfully achieves resilience to cache poisoning and compatibility with ARP, at virtually no cost.

4. A ticket-based approach

The security flaws in ARP stem from the following facts: (1) ARP messages lack guaranteed integrity and authenticity; and (2) ARP messages do not provide proof of IP address ownership. Therefore, adding authentication alone to ARP messages does not solve the problem. Authentication must also be combined with a means to prove address ownership.

We address these requirements through the Ticket-based Address Resolution Protocol (TARP)
TARP implements security by distributing centrally generated attestations \cite{7,50}. These attestations, called tickets, authenticate the association between IP and MAC addresses through statements signed by the Local Ticket Agent (LTA). Each ticket encodes a validity period represented as a start time and an expiration time. Of course, the use of time values assumes some form of loose clock synchronization between the issuing LTA and the validating clients. Such synchronization is a common requirement for many protocols, and devices for its enforcement are well known\cite{40}.

To securely perform address resolution using TARP, a host broadcasts an ARP request. The host with the requested IP address sends a reply, attaching a previously obtained ticket. The signature on the ticket proves that the LTA issued it, i.e., the IP-to-MAC address mapping is valid (or at least was at the time of issuance – see revocation below). The requesting host receives the ticket, validating it with the LTA’s public key. If the signature is valid, the address association is accepted; otherwise, it is ignored. With the introduction of TARP tickets, an adversary cannot successfully forge a TARP reply and, therefore, cannot exploit ARP poisoning attacks.

The remainder of this section discusses the protocol in detail and considers revocation. The following notation is used:

\begin{align*}
H_i & \quad \text{A Generic host} \\
\text{Request}(x) & \quad \text{Request for object } x \\
\text{Reply}(x) & \quad \text{Reply with object } x \\
\text{Ticket}_{H_i} & \quad H_i's\text{ digitally signed ticket} \\
\text{MAC}_i & \quad H_i's\text{ MAC address}
\end{align*}

### 4.1. The ticket-based address resolution protocol

The means by which a ticket is created and distributed is dependent on whether the IP address assignments are static or dynamic. In either case, the address resolution exchange consists of a broadcast request and unicast reply as follows:

\begin{align*}
H_i \to \text{all:} & \quad \text{Request}(\text{MAC}_j) \\
H_j \to H_i: & \quad \text{Reply}(\text{MAC}_j) || \text{Ticket}_{H_i}
\end{align*}

Barring the inclusion of \text{Ticket}_{H_i}, the exchange is identical to ARP. The method in which \textit{H}_j obtains \text{Ticket}_{H_i} differs depending on network requirements.

A network administrator may choose to statically assign IP addresses to hosts. In which case, whenever a host is added to the network, it is configured with the network public key, and network configuration parameters including an IP address, and a ticket, as illustrated in Fig. 3. Because the associations are unlikely to change frequently, it may be acceptable to set long ticket lifetimes. However, there are security, performance, and administrative considerations related to the selection of ticket lifetimes. We consider these issues in depth in Section 4.3 below.

In dynamic IP networks, hosts are assigned IP addresses and configuration parameters by a configuration server using the Dynamic Host Configuration Protocol (DHCP)\cite{21}. Each host receives a lease on an IP address and sends a renewal request upon expiration. At this time, the DHCP server may or may not reassign the host the same IP address.

In a TARP-enabled dynamic IP network, the DHCP server also performs the functionality of an LTA, as shown in Fig. 4. In response to a DHCP request, the server packages a ticket with the configuration information. Accordingly, the ticket expires along with the IP lease. Note that tickets are by definition public; therefore, a secure communication channel is unnecessary. Having the DHCP server play the role of LTA eliminates the need for additional ticket distribution messages, hence maintaining simplicity of protocol design. Additionally, using this method of distribution is logical, as DHCP was designed to distribute configuration parameters. Finally, note that if the LTA becomes

---

2 The LTA is trusted by all network hosts to make claims about IP-to-MAC address mappings.

3 The ticket-approach is also appropriate for optimized ARP implementations that listen to all address resolutions, i.e., any host on the network can validate the ticket.

---

![Fig. 3. Static IP Address Assignment – hosts receive TARP tickets during initial setup, and include them with each ARP reply.](image-url)
temporarily unavailable, the network will not cease to function. TARP resolutions will proceed until tickets expire.

A host requires the LTA’s public key in order to verify tickets. Key distribution is most secure if performed out of band. Another option, although not secure, is Key distribution through assertion and user acceptance, similar to that in the Secure Shell (SSH) protocol [53]. In fact, this technique is used by some 802.1X [4] clients to receive and accept certificates for EAP-TLS [5]. Unfortunately, requiring the user to accept keys allows an adversary new methods of attack. For this paper, we only consider a priori, out of band, manual, distribution of the LTA’s public key, therefore allowing the host to verify responses.

4.2. Ticket format

Maintaining backwards compatibility with ARP is crucial for the adoption of any enhanced address resolution protocol. Compatibility is achieved by integrating the ticket into the ARP reply; no changes need take place to the request. As shown in Fig. 5, the ticket is appended as a variable length payload, with the ticket header modified accordingly.

The Magic field in the ticket header is used to distinguish a TARP reply from an ARP reply. In a TARP reply, the magic field is set to 0x789a0102.4 Since TARP has only one message type, the Type field actually designates the cryptographic algorithm.5 The SigLen field indicates the signature length. The remaining fields contain information required to ensure proper operation. The MAC Addr and IP Addr fields create the address association. The Expiration Time field indicates how long the ticket is valid. Issue Timestamp field indicates when the ticket was generated and is used for ticket revocation as discussed below.

It is possible to imagine tickets generated with very short validity periods. In this case, the overhead of TARP approaches the overhead of S-ARP. In fact, one can think of TARP’s validity period as a generalization of the freshness timestamp and nonce included in S-ARP (see Fig. 2). That is, in an extreme case the LTA performs a similar number of signatures as the AKD in S-ARP. However, this analogy is not exact. S-ARP messages have timestamps on ARP replies and key exchanges with the AKD. This allows the AKD to use longer timestamp intervals asserting timestamps, but it must perform one signature per host per timestamp period, and therefore will not scale as well as TARP.

The ticket validity period is a key element in the design of TARP. It allows the cost of ticket generation to be amortized over the lifetime of a ticket. This concept is not unique to TARP. DNS Security Extensions (DNSSEC) [8–10], uses a signature inception and expiry information in signature resource records (SIG RR). The signature inception and expiry provide a validity period for SIG RRs. It is not surprising that both DNSSEC and TARP make use of validity periods for signed data, as both protocols’ primary objective is to add security to an address resolution protocol while keeping overhead to a minimum.

4 The magic field appeared in S-ARP, and we use it for a similar purpose.
5 As discussed in Section 5, our implementation currently uses 1024-bit RSA, but other key sizes and algorithm may be used as appropriate and desirable.
4.3. Revocation

A reality of current networks is the fact that IP/MAC address associations can change; dynamic bindings (e.g., DHCP) or changes in network configuration can occur before a ticket expires. To be secure, one must provide a revocation mechanism that securely notifies clients which tickets are no longer valid. Historical studies of revocation have sought to limit the cost of notification, e.g., CRLs and other data structures [6,29,32,39,44], limit notification latency, e.g., OCSP [42], or provide frameworks for trading off security guarantees and semantics [6,27,36].

Revocation speaks to the central tradeoff of TARP. Because a revoked ticket may be replayed at any time prior to its expiration, administrators may be tempted to keep the lifetimes short. However, ticket issuance costs grow at the linear inverse of the ticket lifetime. The ability to calibrate the balance between these competing factors through the selection of ticket lifetimes is a central benefit to its design.

The simplest method of handling revocation is to issue certificates that are only valid for a short time. This is similar to the short lived certificates suggested by Ellison et al. in the SPKI/SDSI system [23,48]. Because the tickets are only valid for a short time, the vulnerability to replay is limited and no notification is necessary. Note that a window of vulnerability to replay also exists in S-ARP. The window that is equal to the cache hold time of the ARP reply. Users of TARP can provide similar window by setting the lifetime of the ticket to the ARP cache hold time. However, the burden of the creating the tickets is on the LTA, rather than on the hosts themselves. We experimentally explore the costs of the ticket creation and validation in Section 6.

ARP associations are long lived in networks where IP addresses are assigned manually. For this reason it may be advantageous to create tickets whose lifetimes are essentially infinite for these static associations. In those rare cases where mappings change, one can revoke through re-issuance; all clients would only use the ticket with the latest expiry timestamp. This “latest ticket wins” approach would be vulnerable to active attacks in which the adversary can block delivery of the new ticket. Such attacks represent a powerful adversary within the local area network, and may signal larger and more serious problems. Hence, the risk may be acceptable for many environments.

The most secure solution is to implement a separate revocation service. Such solutions range from the distribution of simple signed certificate revocation lists [29] to instantaneous online verification of ticket validity [42]. Note that simple solutions like CRLs are likely most appropriate, as the costs of the complex ones would eclipse the costs of securing ARP. Hence, we expect that simple, low cost solutions will be used in all networks but those with the highest security requirements. We defer further discussion of the design tradeoffs of revocation services to the relevant literature [37, 41].

An important question is how to recover in the presence of compromise of the LTA. This issue is similar to CA recovery in PKI systems. Unlike many PKI deployments, all TARP clients serviced by an LTA are likely to be under a single administrative domain. Hence, it is reasonable to expect that each client can be manually configured with a new certificate as needed. Larger domains may use techniques to reduce the impact of LTA compromise, e.g., key-splitting [35], issue and revoke LTA keys through local certificate management services, and may use automated management tools for the distribution of LTA signing keys.

4.4. Attacks against TARP

Networks implementing TARP are vulnerable to two types of attacks: active host impersonation, and Denial of Service through ticket flooding. In the following we discuss these attacks and show how they can be mitigated.

4.4.1. Active host impersonation

An active adversary that can block all communication between two hosts can impersonate its victim by spoofing its MAC address and replaying a captured ticket. While this attack is present in the ARP, with TARP, the adversary can only impersonate the victim as long as the ticket is valid. Furthermore, a variant of this attack is present in any solution that uses caching. Fortunately, this attack can be mitigated by using a layer-2 switch with port security, thereby preventing MAC address spoofing.

---

6 The expiration time field is a 32-bit value. When set to its maximum value, the ticket will expire in 2038.
4.4.2. Ticket flooding

Incorporating cryptographic verification into any protocol opens the possibility for a processor-bound DoS attack. By flooding the victim with garbage packets, the adversary causes the victim to devote resources to cryptographic verification of each packet. This attack is made possible by the cost imbalance between generating a garbage request and verifying a request. Generic solutions exist for this DoS attack, and many secure protocol definitions do not address the problem directly. However, we now suggest mitigation strategies for ticket flooding.

A novice approach applies a stateful ARP implementation. In such an implementation, a host ignores ARP replies unless it contains a previously requested IP address. An adversary can circumvent this protection by preceding ARP replies with an ICMP ping request, thereby causing the victim to request a specified IP address.

Another approach, suggested to solve a similar concern in CGA [11], causes the host to defer ticket verification of the TARP reply until the information is needed. In this case, the ARP cache includes a “dirty” flag to indicate the entry requires verification. By foregoing ticket verification, the resulting processor overhead is limited to that present in existing ARP implementations. Note, however, that this technique is not failsafe, as the adversary can interleave ICMP ping requests with TARP replies to invoke verification of dirty cache entries.

A third approach addresses the cost imbalance between generating and verifying garbage tickets. Client puzzles [38] require the sender to solve a small puzzle before the host commits resources for verification. Typically, the host has a trapdoor to allow quick verification of puzzle solutions; verifying puzzle solutions requires significantly less resources than a ticket verification. Client puzzles have been successfully applied to many other protocols, including TLS [20]. Note that an adversary can still flood its victim with client puzzles, but significantly more messages are required to achieve detrimental processor loads.

Both ticket and client puzzle flooding can be mitigated through rate limitation, e.g., incorporated into layer-2 switches. Such switches can limit the rate of ARP packets originating from any or all switch ports, hence alleviating the victim’s processor load. However, while rate limitation reduces processor load, it impedes legitimate requests. There is no way to eliminate all effects of traffic flooding, regardless of the existence of cryptographic operations; ARP alone incurs some overhead. At best, a host can minimize resource commitment until request legitimacy is reasonably verified.

5. Implementation

We have implemented TARP on Linux, version 2.6. The source code is available for download.⁷ Our implementation has two primary goals: to demonstrate that TARP indeed works and is compatible with ARP; and more importantly, to measure the overhead of TARP and compare it to the overhead of both ARP and S-ARP.

Our implementation uses a number of libraries, including libpcap [3] for packet capture, libnet [49] for packet injection, and OpenSSL [2] for cryptographic operations. Similar to S-ARP, our implementation, shown in Fig. 6, has two core components: a loadable kernel module, and a userspace daemon. The kernel module exists to disable kernel processing of incoming ARP packets. Once kernel ARP processing is disabled, the userspace daemon, tarpd, uses existing system libraries to interface with the kernel directly, performing the necessary processing. By implementing the core functionality in userspace, we gain portability and avoid adding complex cryptographic implementations in the Linux kernel.⁸

When loaded, the userspace daemon instructs the kernel module (through the /proc file system) to disable kernel processing of incoming ARP packets and waits for ARP packets to arrive. When an ARP packet arrives, it is processed according to its type. If it is a request, a TARP reply is sent. Otherwise, it is a reply, and the source IP address is compared to whitelist entries (whitelists are described in Section 7.2). If not found, it is treated as a TARP reply and the attached ticket is verified. If the ticket is valid the ARP cache is updated using a netlink socket, and the ticket is cached (in a hash table) to speedup later verifications.

The current implementation of TARP uses RSA with 1024-bit keys. We choose RSA among a number of alternative signature schemes because of its fast signature verification and the availability of highly efficient open source implementations (OpenSSL). We also choose a 1024-bit key size as

⁷ http://siis.cse.psu.edu/tools.html.
⁸ The current CryptoAPI in the Linux 2.6 kernel does not support asymmetric cryptographic operations.
it is fit-to-purpose, striking a good balance between security and performance.

Our current version of TARP includes an administrative tool to generate the LTA’s public/private key pair as well as tickets for manual distribution. We have also implemented an LTA server for dynamic ticket distribution.

During the design phase of the LTA, we have considered two different approaches. In the first approach, a DHCP server performs ticket distribution. This requires modification of the DHCP server code to include LTA functionality. In this case, the DHCP server sends TARP tickets along with DHCP OFFER messages. The ticket is included in the message as a DHCP option. Similarly, the DHCP client must be modified to parse and use the ticket embedded in the DHCP OFFER message. Once a DHCP OFFER is accepted and an ACK message is received from the DHCP server, the DHCP client sends the ticket to the TARP daemon. This ticket distribution approach does not require any specific ticket distribution messages; however, it does require significant modifications to both DHCP server and client code.

A second approach modifies neither the DHCP server nor the client. In this approach, the LTA is separated from the DHCP server. The LTA is triggered by the messages exchange between the DHCP server and client. Once a DHCP OFFER message is sent by the server and accepted by the client, the LTA sends a special TARP message. This is an ARP message with a special opcode (opcode 5), containing the ticket and binding the IP address in the DHCP lease with the MAC address of the client for the lease duration. The client TARP daemon receives the special TARP message and validates the ticket for use in future TARP resolutions.

We choose the latter approach for implementing the LTA. This approach has the advantage of not being dependent on the source or version of the DHCP server running in an environment, making it much more deployable. We believe that an open source TARP aware DHCP server and client are still needed; however, we leave such an implementation for future work.

Our LTA uses the libpcap library to capture DHCP messages (UDP port 67 and 68). The DHCP options in the packet are parsed. Option 53 represents the DHCP message type. It is used to identify DHCP ACK messages. Option 51 represents the lease time in seconds and it is used to set the expiry time of the generated ticket. The ticket IP and MAC address are copied from the fields yiaddr and chaddr from the DHCP message header. We use the packet header provided by libpcap to ensure that the DHCP server generated the packet. This design choice assumes the LTA process is running on the same host as the DHCP server; however, given another secure method for the LTA to learn of completed DHCP transactions, the LTA and DHCP server can reside on different hosts. For example, a dedicated listener could reside on the DHCP server and inform the remote LTA of events. Finally, the LTA uses the same libraries (OpenSSL and libnet) as tarpd to generate and send the tarp ticket. Our implementation of the LTA integrates seamlessly with DHCP without the need for changes to the protocol or its implementation.

---

9 Some DHCP servers allow administrators to configure static user defined options; however, the LTA requires the field to be populated with a different ticket for each host.
6. Performance evaluation

In order to understand the cost incurred by TARP, we performed two types of measurements: the macro-benchmark indicates the cost seen by an application, the micro-benchmark evaluates the delay of the primary operations. For the macro-benchmarks, we compare our protocol to both ARP and S-ARP.

Our test environment consists of two desktop PCs and included a laptop as the AKD in the S-ARP measurements. The desktops were equipped with 2.8 GHz Pentium 4 processors and 1GB of RAM, while the laptop contained a 1.0 GHz Pentium 3 processor and 1GB of RAM. All machines ran version 2.6 of the Linux Kernel and were connected via a Gigabit Ethernet switch. Finally, because S-ARP was written for an earlier version of Linux, small updates were required to compile and run it in our environment.

6.1. Macro-benchmark

TARP provides significant advantage over S-ARP due to the decrease in signatures and message exchanges. The ticket-approach embodies the central design tradeoff. As described in Section 4.2, the LTA performs significantly less signatures than the AKD in S-ARP; however, the genuine performance advantage of TARP lies in the client operations. In TARP, the host performs one signature verification per received ARP reply, and if the ticket was cached, only a memory comparison is required. In S-ARP, every ARP reply must be signed by one endpoint and verified by the other endpoint. Furthermore, if the host receiving the ARP reply has already verified and cached the public key of the sender, it must also perform an exchange with the AKD and verify the response.

We characterize TARP’s improvement over S-ARP by comparing the round trip delay. Measurements were taken from an application to provide a realistic and consistent characterization of the system level costs of both protocols. Finally, we compare the round trip delay to ARP, acting as a baseline for overhead in our test environment.

To observe round trip delay from the application level, we used a custom ping application to flush the system’s ARP cache after each ICMP echo request/reply pair. This ensured each measurement included the overhead of address resolution. We performed five experiments, each consisting of 1000 ICMP echo requests. These experiments measured the round trip delay for ARP, S-ARP, and TARP with and without caching.

Table 1 summarizes the mean, standard deviation, and median of the recorded measurements for the protocols operating with caching turned on (best case scenario). The overhead was calculated from the mean. As shown, we observed small standard deviations for each experiment. This resulted from the largely controlled test environment.

With caching turned on, the S-ARP reply sender must perform a signature operation, and the S-ARP reply receiver must perform a signature verification. On the other hand, the TARP reply sender performs no cryptographic operations, and with caching turned on, neither does the receiver. Table 1 verifies this difference. S-ARP observes an overhead of approximately 5.4 ms. This is 55 times greater than TARP, which observes only a 98 µs overhead. The delay incurred by TARP is essentially unnoticeable, meeting our low overhead design requirement, and makes the protocol appropriate even for devices with limited computational power.

Table 2 summarizes the worst case performance measurements: when neither protocol enables caching. Again, the mean was used to calculate the overhead. With caching disabled, the delay introduced by S-ARP doubles, resulting in an overhead of 11 ms. This is primarily due the network communication with the AKD. On the other hand, TARP is two orders of magnitude faster, incurring only 186 µs of overhead. Hence, when signature verification is required, the delay incurred by TARP remains virtually insignificant.

Table 1
Round-trip delay for ICMP echo requests with caching (1000 requests)

<table>
<thead>
<tr>
<th>Protocol</th>
<th>( \bar{\sigma} ) (µs)</th>
<th>( \sigma ) (µs)</th>
<th>Median (µs)</th>
<th>( \bar{\text{Overhead}} ) (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARP</td>
<td>1178.59</td>
<td>259.98</td>
<td>1108</td>
<td>N/A</td>
</tr>
<tr>
<td>S-ARP</td>
<td>6579.57</td>
<td>415.99</td>
<td>6535</td>
<td>5401.02</td>
</tr>
<tr>
<td>TARP</td>
<td>1276.54</td>
<td>262.47</td>
<td>1206</td>
<td>97.95</td>
</tr>
</tbody>
</table>

Table 2
Round-trip delay for ICMP echo requests without caching (1000 requests)

<table>
<thead>
<tr>
<th>Protocol</th>
<th>( \bar{\sigma} ) (µs)</th>
<th>( \sigma ) (µs)</th>
<th>Median (µs)</th>
<th>( \bar{\text{Overhead}} ) (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARP</td>
<td>1178.59</td>
<td>259.98</td>
<td>1108</td>
<td>N/A</td>
</tr>
<tr>
<td>S-ARP</td>
<td>12479.71</td>
<td>571.47</td>
<td>12176</td>
<td>11319.12</td>
</tr>
<tr>
<td>TARP</td>
<td>1364.21</td>
<td>253.93</td>
<td>1297</td>
<td>185.62</td>
</tr>
</tbody>
</table>
In summary, our results show that TARP out-performs S-ARP by at least an order of magnitude in all experiments, and by as much as two orders of magnitude in some cases. More importantly, the results indicate TARP incurs a virtually insignificant overhead. As discussed in our solution criteria, this is vital to the adoption of a secure replacement for ARP.

6.2. Micro-benchmarks

Operationally breaking down TARP’s overhead provides insight into how the protocol will perform on different types of devices. TARP message flow begins by requesting an address association. Since the request is identical to that of ARP, no overhead is introduced. When the remote host replies, a ticket is simply appended to a reply. While this requires additional system I/O and network traffic, the overhead is negligible. Upon receiving a TARP reply, a host must verify the ticket signature. This stage requires an asymmetric cryptographic operation and should therefore be investigated. As TARP operates in userspace, cache updates result in additional context switches, slowing down operation. Determining this cost foretells the gain resulting from a kernel based implementation. Finally, TARP gains significant performance improvements by amortizing the cost of ticket generation.

Table 3 summarizes the micro-benchmarks. The experimental environment was more controlled than that of the macro-benchmarks, therefore, even with 100 runs, a small standard deviation was achieved. The ticket signature verification consists mainly of a 1024-bit RSA signature verification. This operation is only required when a received ticket does not exist in the cache. The average time of 119 µs corresponds directly to the difference between the two TARP variations measured in the macro-benchmark. The cache update also reflects the values measured in the macro-benchmark. If TARP was implemented in kernelspace, 74 µs would be virtually eliminated, removing essentially all overhead when tickets are cached. Finally, ticket generation requires 4.5 ms. As the ratio of requests to ticket

<table>
<thead>
<tr>
<th>Operation</th>
<th>Average (µs)</th>
<th>σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ticket signature verification</td>
<td>119.12</td>
<td>2.00</td>
</tr>
<tr>
<td>Update of ARP cache</td>
<td>74.07</td>
<td>7.15</td>
</tr>
<tr>
<td>Ticket generation</td>
<td>4535.36</td>
<td>68.33</td>
</tr>
</tbody>
</table>

Table 4

Execution times in microseconds for ticket distribution operations (Average of 100 measurements)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Average (µs)</th>
<th>σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>LTA Ticket Generation</td>
<td>5383.62</td>
<td>984.95</td>
</tr>
<tr>
<td>New Ticket Verification</td>
<td>151.26</td>
<td>33.75</td>
</tr>
</tbody>
</table>

6.3. DHCP integration micro-benchmarks

We also measured the overhead of TARP associated with ticket distribution. Ticket distribution in our experiment was done by an LTA. The LTA process is triggered by the DHCP message exchange between a DHCP server and client. The LTA extracts the mapping between IP and MAC addresses from the DHCP ACK message, which indicates that an offer was accepted by the client and further acknowledged by the server. The LTA generates a ticket and sends it to the client in a special TARP message.

In our experiment we measured the overhead incurred by the LTA for ticket generation as well as the overhead incurred by the client for ticket validation. Table 4 summarizes our results. As shown, the results agree with the micro-benchmark finding in Table 3.

7. Discussion

In this section, we briefly discuss TARP with respect to DHCP, interoperability, and Secure Neighbor Discovery.

7.1. DHCP

As previously indicated, TARP does not include key and ticket distribution messages. Instead of creating a new distribution protocol, DHCP is used. Clients receiving tickets alongside DHCP replies can readily authenticate the DHCP reply by verifying the signature on the ticket. However, this only provides one-way authentication. In some cases, authenticating clients before distributing DHCP leases and tickets may be required in order to restrict network access or avoid attacks such as IP address pool exhaustion.

Methods for securing DHCP exist. Suggestions include Authentication for DHCP [22], where the protocol has been extended with additional security
parameters. Of course, such systems need a way to tie authentication into a central system. Many network installations already have such devices deployed. Applicable back-ends include RADIUS [47], a common authentication database. How and when such authentication is performed is the subject of network policy and influenced by available infrastructure, and hence should be dealt with as operational needs dictate.

7.2. Interoperability

The success of any secure ARP replacement hinges on its ability to interoperate with legacy infrastructure, if only to support some transition period. ARP maps IP addresses to MAC addresses on a local network. Within that network, transition to TARP may be impeded by the speed of manufacturer updates, e.g., gateway appliances and print servers. Hence, TARP supports incremental deployment. When operating in a mixed environment, two scenarios of interest result: (1) an ARP host, $H_a$, sends an ARP request to TARP-enabled host, $H_t$; or (2) a TARP host, $H_t$, sends a request to an ARP host, $H_a$.

In scenario 1, when $H_t$ receives an ARP request, it does not know if $H_a$ runs the original protocol or not, because both request packet forms are identical. $H_t$ proceeds to return a TARP reply. $H_a$ receives this reply and parses it correctly. This occurs, because to an ARP host, the ticket simply appears as network garbage. Hence, $H_a$ can successfully resolve $H_t$’s MAC address and therefore transmit data.

In scenario 2, $H_a$ receives a TARP request, which is identical to an ARP request, and replies to $H_t$ with an ARP reply (no ticket attached). As $H_t$ cannot verify the address association, it ignores the reply. After time elapses, the higher layer protocols on $H_a$ time out. The only barrier keeping the mixed network from functioning is the verification of an ARP reply by a TARP-enabled host. A TARP-enabled host cannot simply accept all ARP replies; this invalidates any security gained from the new protocol. In order to allow address resolution to proceed in scenario 2, TARP supports whitelists. Whitelist entries are one of two types: whitelisted IP ranges, or static ARP mappings.

TARP supports whitelisted IP ranges. This allows a DHCP server¹⁰ to distribute IP addresses from two different pools – ARP hosts, and TARP hosts. Such a configuration may be necessary for a transition to TARP, as it is needed to specify precisely which hosts are participating in the protocol.

These lists can also contain hard-coded MAC and IP address mappings. While currently not implemented, this type of whitelist can be distributed by the network administrator or DHCP server. This allows dynamic configuration of static DHCP entries for known devices that do not support TARP. Example devices include embedded hosts such as routers that require vendor support for protocol updates.

Although TARP is designed to interoperate with ARP to facilitate incremental deployment, hosts running ARP are not in any way protected by TARP. Moreover, TARP cache entries referencing whitelisted hosts are also subject to poisoning for specific addresses. In order to achieve the most from TARP all hosts on the local area network should be migrated to TARP.

7.3. Secure neighbor discovery

As described in Section 3.3, the Secure Neighbor Discovery (SEND) [12] protocol was designed to provide security to ND. SEND adds a number of options to ND that together provide message authentication and proof of address ownership. Address ownership is provided by the Cryptographically Generated Address (CGA) parameters option, while message authentication is provided by the timestamp or nonce option along with the RSA signature option.

SEND uses CGA to add distributed address resolution security to Neighbor Discovery; there is no need for a central authority.¹¹ However, each address resolution includes an RSA signature generation and verification. This overhead is similar to SARP with caching. As shown in Section 6, this approach results in an order of magnitude greater overhead than TARP.

We propose using tickets to secure Neighbor Discovery. Tickets can be included in ND messages as a new option. The ticket option includes a timestamp and validity period. A signature is included as a new RSA signature option. The signature covers the address association and the ticket option, and is generated using the LTA’s private key. No nonce

---

¹⁰ The DHCP server signs the whitelist with the network private key.

¹¹ Recall that CGA cannot protect IPv4 address resolution.
or timestamp ND options are needed. By using the new RSA and ticket options with ND, the cost of signature generation is amortized over the lifetime of the address association.

7.4. Wireless networks

Insecure wireless network deployments have rekindled interest in ARP security. Increasing the barrier of entry, e.g., using IEEE 802.1X [4] with EAP-TLS [5], prevents outsiders from affecting legitimate host ARP caches, but it does not address the vulnerability in ARP. Frequently, network administrators must allow partially trusted hosts to join the network. Therefore the security concern remains.

TARP’s security model assumes an untrusted local network. This model is consistent with both protected and unprotected wireless networks. TARP operates between the link and network layers, therefore it is agnostic to the physical transmission medium and will operate unmodified on wireless networks.

8. Conclusions

ARP is essential to the proper operation of IP networks. However, the lack of authentication and proof of address ownership in ARP leads to a range of serious security vulnerabilities. Previous solutions to ARP have failed to simultaneously address the compatibility and cost requirements of current networks. We have introduced TARP: a Ticket-based Address Resolution Protocol, and detailed its implementation. Built as an extension to ARP, TARP achieves resilience to cache poisoning. We have shown experimentally that TARP reduces cost by as much as two orders of magnitude over existing solutions for security issues in ARP.

ARP vulnerabilities will remain a serious network security problem until a viable alternative is accepted. We have shown TARP to be viable, but much work remains before our implementation can be broadly used. Acceptance by the Internet community and implementation of TARP not only in popular operating systems but also in network devices is needed to make ARP based attacks a thing of the past.

Acknowledgments

We would like to thank Patrick Traynor and Kevin Butler for their editorial comments as well as the SIIS Lab as a whole for their many suggestions in the design of this work.

References


[46] D.C. Plummer, An Ethernet address resolution protocol or converting network protocol addresses to 48-bit Ethernet address for transmission on Ethernet hardware, RFC 826, Nov. 1982.


Wesam Lootah graduated with a Master of Science Student from the Pennsylvania State University’s Department of Computer Science and Engineering in May 2006. He graduated from the Ohio State University in 1997 with a BS in Computer and Information Science. After graduating with his BS, he was employed by the Government of Dubai. The United Arab Emirates and still holds the position of Head of Application Services. He managed one of the largest ERP implementation projects in the Middle East for the Government of Dubai. The project entailed the design of a core model and the rollout of 20 modules across 6 Government entities with over 40,000 employees.
William Enck is a Ph.D. candidate in the department of Computer Science and Engineering at the Pennsylvania State University. He received a B.S. (with highest distinction and honors) and M.S. in Computer Science and Engineering from the Pennsylvania State University in 2004 and 2006, respectively. His current research efforts include operating systems security, network and telecommunications security, and secure hardware architectures.

Patrick McDaniel is an Associate Professor in the Computer Science and Engineering Department at the Pennsylvania State University and co-director of the Systems and Internet Infrastructure Security Laboratory. His research efforts centrally focus on network, telecommunications, and systems security, language-based security, and technical and public policy issues in digital media. He was awarded the National Science Foundation CAREER Award and has chaired several top conferences in security including, among others, the 2007 and 2008 IEEE Symposium on Security and Privacy and the 2005 USENIX Security Symposium. He is also an associate editor of the journals ACM Transactions on Information and System Security, IEEE Transactions on Software Engineering, and ACM Transactions on Internet Technologies. Prior to pursuing his Ph.D. in 1996 at the University of Michigan, he was a software architect and program manager in the telecommunications industry.