TARP: Ticket-based Address Resolution Protocol

Wesam Lootah, William Enck, Patrick McDaniel
Department of Computer Science and Engineering
The Pennsylvania State University
{lootah, enck, mcdaniel}@cse.psu.edu

June 20, 2005

Abstract

IP networks fundamentally rely on the Address Resolution Protocol (ARP) for proper operation. Unfortunately, vulnerabilities in the ARP protocol enable a raft of IP-based impersonation, man-in-the-middle, or 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 MAC/IP address mapping attestations through existing ARP messages. We detail the TARP protocol and its implementation within the Linux operating system. 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.

1 Introduction

The Address Resolution Protocol (ARP) [31] is the glue that holds together the network and link layers of the IP protocol stack. The primary function of ARP is to map hardware addresses onto host IP 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 [6, 7]. Adversaries can exploit ARP to impersonate hosts, perform man-in-the-middle attacks, or simply DoS victims. 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 protocol stack is technically and 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 [18, 8, 15, 20] 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) protocol. TARP implements security by distributing centrally generated MAC/IP address mapping attestations [35, 5]. 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 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 trade-off. Ticket generation costs grow at the linear inverse of the ticket's lifetime. The ticket lifetime dictates the vulnerability to replay attacks.[1] 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.

[1]We consider an alternate design in Section 4.3 in which we address replay vulnerabilities through the introduction of a revocation service.

Security in resolution services remains an open problem. Whether resolving domain or hostnames [14, 12], claims of address ownership [35, 5], or other network artifacts, 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 inform of 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 considering the vulnerabilities inherent to its current design. Section 3 considers past efforts at securing ARP and other related works. Section 4 details the TARP architecture and its operation within local networks. Section 5 outlines the integration of the TARP client within the Linux kernel and identifies. Section 6 explores the performance of TARP experimentally. Section 7 considers several operational issues associated with the use of TARP. Section 8 concludes.

2 Background

The *Address Resolution Protocol* (ARP) [31] 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 the 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. Those packets destined for internal network are delivered directly. Whether the destination address is local or gateway, 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 address should reply with its local hardware address. This reply, containing both the requested IP address and associated MAC address, is sent via unicast to the querying host. The host caches the association, which expires and is evicted at a later time per some local policy. Once evicted, the host repeats the request, cache, and eventual ejection. While the cache hold time for a response is undefined in the protocol specification, many implementations set the expiration to approximately 20 minutes, with the option of resetting the expiry timer after each use [8].

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 [36].

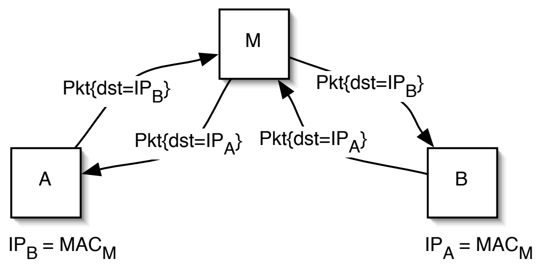

Figure 1: Example of a Man-in-the-Middle attack in progress. Both host A and B believe they are talking directly to each other.

Most IP protocol 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 ICMP `ping` message. The spoofed message contains the targeted IP address, requiring the host to resolve the MAC address to reply. By carefully poisoning the cache and spoofing replies, an adversary can perform both *Denial of Service* (DoS) and *Man in the Middle* (MITM) attacks [13]. Such attacks were known even in 1989 [6], and they still exist today [7].

Cache poisoning can be used to mount various types of DoS attacks. In the most simple 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 Figure 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

Several attempts have been made to address the above security issues through methods external to the ARP protocol. For example, it has been proposed that hosts can statically configure ARP tables [1]. Of course, this would incur a huge administrative overhead and is largely intractable for dynamic environments. Conversely, the port security [9] features available in recent switches restrict 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.

Other solutions attempt to detect misbehavior, rather than prevent it. ARPWatch [20], 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 [37]. 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.

A number of cryptographic protocols have targeted issues in the ARP security. 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, yet untrustworthy hosts, from injecting malicious messages. In yet another approach, Gouda and Huang [15] 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-address-to-hardware-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. Kempf exploits Identity-Based cryptographic techniques in the Address Based Keys (ABK) [18] protocol. IP addresses are used as public keys in ABK. However, contemporary identity-based systems require one or more heavyweight cryptographic operations per signature or validation. Hence, their cost is prohibitive for many resource poor devices.

The most popular ARP security protocol, S-ARP [8], also uses asymmetric cryptography. However, unlike ABK, hosts use self-created public/private key pair 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. S-ARP requires, at minimum, a single signature generation and verification per address resolution. As illustrated in Section 6, this cost can be significant.

None of these 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 major flaw in ARP is the lack of message authentication. For the remainder of this paper, we classify ARP vulnerabilities as falling into one of the two following categories:

- **reply spoofing:** forging an ARP reply to inject a new address association into the victim's cache
- **entry poisoning:** forging an ARP reply to replace an address association in the victim's cache

We address these vulnerabilities through the Ticket-based Address Resolution Protocol (TARP). TARP implements security by distributing centrally generated attestations [35, 5]. These attestations, called *tickets*, authenticate the association between MAC and IP addresses through statements signed by the local Local Ticket Agent (LTA). Each ticket encodes a validity period as an expiration time. Of course, the use of expiration times assume some form of loose clock synchronization between the issuer LTA and the validating clients. Such synchronization is a common requirement for many protocols, and devices for its enforcement are well known [26]. We defer discussion of issues relating to synchronization to future work.

To securely perform address resolution using TARP, a host broadcasts a ARP request. The host with the requested IP address sends a reply, attaching previously obtained ticket. The signature on the ticket proves that the LTA issued it, i.e., the MAC to IP 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 signatures is valid, the address association is accepted; otherwise, it is ignored. With

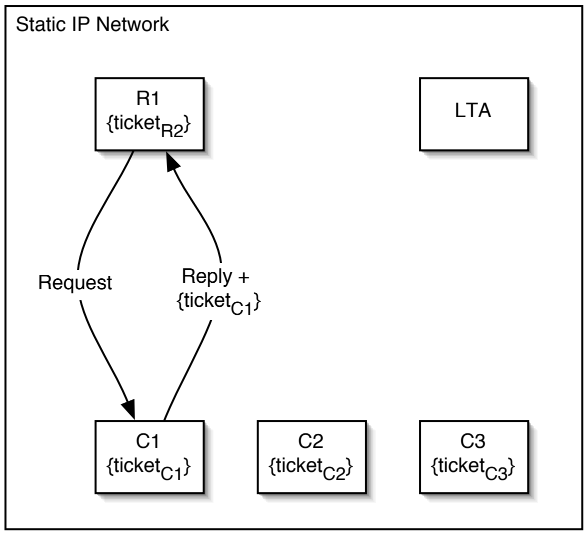

Figure 2: Static IP Address Assignment - hosts receive TARP tickets during initial setup, and include them with each ARP reply.

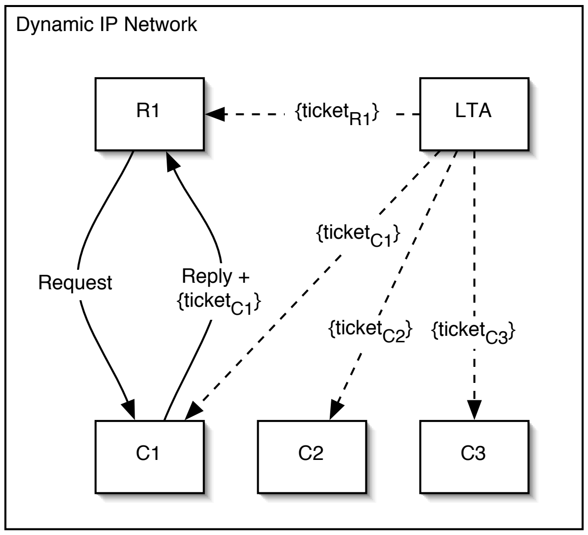

Figure 3: Dynamic IP Address Assignment - hosts receive TARP tickets during the initial DHCP exchange, and include them with each ARP reply.

the introduction of TARP tickets, an adversary cannot successfully forge a TARP reply and, therefore, cannot exploit ARP poisoning attacks.

4.1 The TARP Protocol

The means by which a ticket is created and distributed is dependent on whether the IP address assignments are static or dynamic. Illustrated in Figure 2, whenever a host is added to a static assignment network, it is configured with the network public key, an IP address, and a ticket. 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). Each host receives a lease on an IP address and sends a renewal request upon expiration as shown in Figure 3. 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. 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.

A host requires the LTA's public key in order to verify tickets. Key distribution is most secure if performed out of band. While less secure, this distribution could also be performed through assertion and user acceptance, similar to that in the Secure Shell (SSH) protocol [38]. Unfortunately, this allows an adversary new methods of attack. For this paper, we only consider manual distribution of the LTA's public key.

The operation of ticket resolution proceeds identical in both the static and dynamic cases once tickets have been distributed to each host. TARP message flow is similar to the ARP, with the exception that the ticket is appended to each replies as defined in the preceding section.

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 Figure 4, 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 the new reply from an ARP reply. If it is a TARP reply the magic field is set to 0x789a0102 [2]. Since TARP

[2]The magic field originally appeared in S-ARP, and we use it for a

has only one message type, the *Type* field actually designates the cryptographic algorithm.[3] The *SigLen* indicates the signature length. The remaining fields contain information required to ensure proper operation. The *MAC Addr* and *IP Addr* create the address association. The *Expiration Time* indicates how long the ticket is valid. *Issue Timestamp* indicates when the ticket was generated and is used for ticket revocation as discussed below.

4.3 Revocation

A reality of current networks that IP/MAC address associations can change; dynamic bindings (e.g., DHCP [10]) or changes in network configuration can occur before the 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 [17, 25, 4, 29, 19], limit notification latency, e.g., OCSP [28], or provide frameworks for trading off security guarantees and semantics [23, 4, 16].

Revocation speaks to the central tradeoff of TARP. Because 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 similar to the *short lived certificates* suggested by Ellison et al. in the SPKI/SDSI system [33]. 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.[4] In those rare cases where mappings change, one can revoke through reissuance; all clients would only use the ticket with the latest expiry timestamp. This "latest ticket wins" approach would

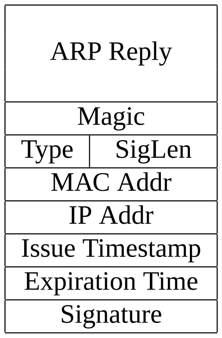

Figure 4: TARP Reply Packet Format - the TARP signature covers all fields from the Magic through the expiration time.

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 [17] to instantaneous online verification of ticket validity [28]. Note that the 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 [27, 24].

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 a 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 employ techniques to reduce the impact of LTA compromise, e.g., key-splitting [22], 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 DoS through ticket flooding. These attacks result from problems out of the scope of this paper.

An active adversary that can block all communication been 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

similar purpose.

[3]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]The expiration time field is a 32-bit value. When set to its maximum value, the ticket will expire in 2038.

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 spoofing.

An adversary can also take advantage of the cost imbalance between generating a TARP reply and processing it. Exploiting this, a DoS attack can be launched by flooding the victim with bogus TARP replies. These bogus replies are trivial to generate, hence allowing the adversary to easily send thousands of TARP replies at a cost magnitudes lower than the resulting validation attempt. By flooding a victim with bogus ICMP requests and corresponding TARP replies, the adversary successfully circumvents even a stateful ARP implementation and consumes the victim's CPU resources. As this attack results from ICMP behavior, mitigation requires adaptation of that protocol and is outside the scope of this paper.

5 Implementation

We have implemented TARP on Linux, version 2.6. The source code is available for download at URL: [21]. 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 makes use of a number of libraries including libpcap [3] for packet capture, libnet [34] for packet injection and openSSL [2] for cryptographic operations. Similar to S-ARP, our implementation has two main components – a loadable kernel module and a userspace daemon. The kernel module provides the proper hooks to disable kernel processing of incoming ARP packets. This allows the userspace daemon, `tarpd`, to capture and respond to incoming ARP packets. By implementing most of the functionality in userspace, we gain better portability and avoid implementing complex cryptography in the Linux kernel.

When loaded, the userspace daemon instructs the kernel module (through the /proc filesystem) to disable kernel processing of incoming ARP packets and waits for ARP packets. 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 white-list entries. 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 this key size is fit-to-purpose, striking a good balance between security and performance.

Our current version of TARP does not include a DHCP server and client, instead it includes an administrative tool to generate the LTA's key pair and generate tickets. A TARP aware implantation of a DHCP is left for future work. Such an implementation is not necessary for the evaluation of TARP's performance as the cost of ticket generation and distribution is amortized across the lifetime of a ticket.

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 2.8GHz Pentium 4 processors and 1GB of RAM, while the laptop contained a 1.0GHz 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 [30] was written for an earlier version of Linux, small updates were required to compile and run it in our environment.

6.1 Macro-benchmark

The macro-benchmark observes the round trip time of the three tested protocols: ARP, S-ARP, and TARP. While direct measurement in the kernel is possible, we chose an indirect route, measuring delay at the application level. For this method, ARP is used as a baseline. The overhead is calculated by taking the difference between ARP and both S-ARP and TARP. Since the overhead is the desired result, the indirect method produces the same results as a direct measurement. Additionally, measuring delays from the application level not only reflects real costs, it provides consistent measurement between the tested protocols.

To observe round trip delay from the application level, we used a custom `ping` program that flushed 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.

Protocol	$\overline{x}$ (μs)	σ (μs)	Median (μs)	$\overline{x}$ Overhead (μs)
ARP	1178.59	259.98	1108	N/A
S-ARP	6579.57	415.99	6535	5401.02
TARP	1276.54	262.47	1206	97.95

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

Protocol	$\overline{x}$ (μs)	σ (μs)	Median (μs)	$\overline{x}$ Overhead (μs)
ARP	1178.59	259.98	1108	N/A
S-ARP	12479.71	571.47	12176	11319.12
TARP	1364.21	253.93	1297	185.62

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

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 proton. This resulted from the largely controlled test environment.

With caching turned on, 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 performance sensitivity design requirement.

Table 2 summarizes the worst case performance measurements. Again, the mean was used to calculate the overhead. When caching is disabled, the delay introduced by S-ARP doubles, resulting in an overhead of 11 ms. 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 is virtually insignificant.

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 generations approaches one, TARP performs similarly to S-ARP. This is where the real power of TARP is introduced.

7 Discussion

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 [11], where the protocol has been extended with additional security parameters. Of

Operation	Average (μs)	σ
Ticket Signature Verification	119.12	2.00
Update of ARP cache	74.07	7.15
Ticket Generation	4535.36	68.33

Table 3: Execution times in microseconds for TARP operations (Average of 100 measurements).

course, such systems need a way to tie authentication into a central system. Many network installations already have such devices deployed. Applicable backends include RADIUS [32], a common authentication database. How and when such authentication is the subject of network policy and available infrastructure, and hence should be dealt with as operational needs dictate.

While TARP successfully and efficiently prevents cache poisoning, it is useless without a plan for incremental deployment. Mixed networks result in two scenarios of interest to incremental deployment: 1) an ARP host (H_c) sends an ARP request to TARP-enabled host (H_t), or 2) a TARP host (H_t) sends a request to ARP host (H_c).

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

Scenario 2 occurs when H_t replies to H_c. H_c receives the TARP request, which again is identical to an ARP request, and replies to H_t. As H_t cannot verify the address association, it ignores the reply. After time elapses, the higher layer protocols times out. If H_t could be made to accept this reply, the lookup would complete successfully.

The only barrier keeping the mixed network from functioning is the verification of an ARP reply. A TARP enabled host cannot simply accept all ARP replies; this invalidates any security gained from the new protocol. In order to allow the above scenarios, 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 server[5] 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 hardcoded 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 ARP entries for known devices that do not support TARP. Example devices include embedded hosts such as routers that require vendor support for protocol updates.

[5]The DHCP server signs the whitelist with the network private key

8 Conclusions

ARP is essential to the proper operation of IP networks. However, the lack of authentication 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 protocols.

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. Extensions including support for dynamic environments are requisite. Finally, we seek further operational experience; a deeper understanding of the costs and limitations of our approach can only be gleaned from field testing. We are currently actively performing such a field test within our parent institution.

9 Acknowledgements

Thanks to Patrick Traynor and Kevin Butler for their support and invaluable comments on the style and content of this paper. Thanks also to Dhananjay Bapat for being so accommodating and allowing us to use his work space and equipment.

Finally special thanks to Ruth Lootah and Megan McDaniel for their help and support throughout the process of this project.

References

[1] Anatomy of an arp poisoning attack. http://www.watchguard.com/infocenter/editorial/135324.asp, accessed June 2005.

[2] The openssl library. http://www.openssl.org/.

[3] The packet capture library. http://www.tcpdump.org/.

[4] C. Adams and R. Zuccherato. A General, Flexible Approach to Certificate Revocation, June 1998. http://www.entrust.com/securityzone/whitepapers.htm.

[5] W. Aiello, J. Ioannidis, and P. McDaniel. Origin Authentication in Interdomain Routing. In *Proceedings of 10th ACM Conference on Computer and Communications Security*, pages 165–178. ACM, October 2003. Washington, DC.

[6] S. M. Bellovin. Security problems in the tcp/ip protocol suite. *Computer Communications Review*, 2(19):32–48, April 1989.

[7] S. M. Bellovin. A look back at ”security problems in the tcp/ip protocol suite”. In *20th Annual Computer Security Application Conference (ACSAC)*, pages 229–249, December 2004.

[8] D. Bruschi, A. Orgnaghi, and E. Rosti. S-arp: a secure address resolution protocol. 2003.

[9] Cisco Systems. *Catalyst 4500 Series Switch Cisco IOS Software Configuration Guide, 12.1(19)EW*. http://www.cisco.com/en/US/products/hw/ switches/ps4324/products_configuration_guide_chapter 09186a008019d0de.html, accessed May 2005.

[10] R. Droms. Dynamic host configuration protocol. RFC 2131, March 1997.

[11] R. Droms and W. Arbaugh. Authentication for dhcp messages. RFC 3118, June 2001. http://www.ietf.org/rfc/rfc3118.txt?number=3118.

[12] D. Eastlake and C. Kaufman. RFC 2065, Domain Name System Security Extensions. *Internet Engineering Task Force*, January 1997.

[13] B. Fleck and J. Dimov. Wireless access points and arp poisoning: Wireless vulnerabilities that expose the wired network. http://downloads.securityfocus.com/ library/arppoison.pdf.

[14] J. Galvin. Public Key Distribution with Secure DNS. In *Proceedings of the 6th USENIX Security Symposium*, pages 161–170, July 1996.

[15] M. Gouda. and C. Huang. A secure address resolution protocol. *Computer Networks*, 41:860–921, January 2003.

[16] C. A. Gunter and T. Jim. Generalized certificate revocation. In *POPL ’00: Proceedings of the 27th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 316–329, New York, NY, USA, 2000. ACM Press.

[17] R. Housley, W. Ford, W. Polk, and D. Solo. RFC 2459, Internet X.509 Public Key Infrastructure Certificate and CRL Profile. *Internet Engineering Task Force*, January 1999.

[18] J. Kempf, C. Gentry, and A. Silverberg. Securing ipv6 neighbor discovery using address based keys (abks). http://www.watersprings.org/pub/id/draft-kempf-abk-nd-00.txt. draft-kempf-ipng-secure-nd-00.txt work in progress.

[19] P. Kocher. On Certificate Revocation and Validation. In R. Hirschfeld, editor, *Financial Cryptography FC ’98*, volume 1465, pages 172–177, Anguilla, British West Indies, February 1998. Springer.

[20] L. B. N. L. (LBNL). Arpwatch: Ethernet monitor program. http://www-nrg.ee.lbl.gov, accessed May 2005.

[21] W. Lootah. Tarp source code. http://siis.cse.psu.edu/tools.html.

[22] M. Malkin, T. D. Wu, and D. Boneh. Experimenting with shared generation of rsa keys. In *Proceedings of Network and Distributed Systems Security 1999*. Internet Society, February 1999. San Diego, CA.

[23] P. McDaniel and S. Jamin. Windowed Certificate Revocation. In *Proceedings of IEEE INFOCOM 2000*, pages 1406–1414. IEEE, March 2000. Tel Aviv, Israel.

[24] P. McDaniel and A. Rubin. A Response to ‘Can We Eliminate Certificate Revocation Lists?’. In *Proceedings of Financial Cryptography 2000*. International Financial Cryptography Association (IFCA), February 2000. Anguilla, British West Indies.

[25] S. Micali. Efficient Certificate Revocation. Technical Report Technical Memo MIT/LCS/TM-542b, Massachusetts Institute of Technology, 1996.

[26] D. L. Mills. RFC 1301, network time protocol (version 3) specification, implementation. *Internet Engineering Task Force*, March 1992.

[27] M. Myers. Revocation: Options and Challenges. In R. Hirschfeld, editor, *Financial Cryptography FC ’98*, volume 1465, pages 165–171, Anguilla, British West Indies, February 1998. Springer.

[28] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. RFC 2560, X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. *Internet Engineering Task Force*, June 1999.

[29] M. Noar and K. Nassim. Certificate Revocation and Certificate Update. In *Proceedings of the 7th USENIX Security Symposium*, pages 217–228, January 1998.

[30] A. Ornaghi. S-arp: a secure address resolution protocol. http://security.dico.unimi.it/research.it.html#sarpd, accessed May 2005.

[31] D. C. Plummer. An ethernet address resolution protocol or converting network protocol addresses to 48.bit ethernet address for tansmission on ethernet hardware. RFC 826, November 1982.

[32] C. Rigney, S. Willens, A. Rubens, and W. Simpson. RFC 2865, remote authentication dial in user service (RADIUS). *Internet Engineering Task Force*, June 2000.

[33] R. Rivest and B. Lampson. SDSI A Simple Distributed Security Infrastructure, October 1996. http://theory.lcs.mit.edu/r̃ivest/sdsi11.html.

[34] M. Schiffman. The libnet packet construction library. http://www.packetfactory.net/libnet/.

[35] K. Seo, C. Lynn, and S. Kent. Public-Key Infrastructure for the Secure Border Gateway Protocol (S-BGP). In *Proceedings of DARPA Information Survivability Conference and Exposition II*. IEEE, June 2001.

[36] D. Song. dsniff: a collection of tools for network auditing and penetration testing. http://www.monkey.org/ dugsong/dsniff, accessed May 2005.

[37] M. V. Tripunitara and P. Dutta. A middle approach to asynchronous and backward compatiable detection and prevention of arp cache poisoning. pages 303–309, 1999.

[38] T. Ylonen. SSH - Secure Login Connections Over the Internet. In *Proceedings of 6th USENIX UNIX Security Symposium*, pages 37–42. USENIX Association, June 1996. San Jose, CA.