Mission Arp-Possible Final Report

How To Make ARP Secure - ARPSec

Members: Varun Sayal, Amit Bapat, Leixiang Wu

Section 1: Introduction

1 Introduction:

ARP (Address Resolution Protocol) is a protocol layer between network and data-link layer. It is often refereed as 2.5 layer protocol. It is designed to allow members of a network to communicate with each other without having to rely on IP routing to reach machines. The ARP protocol is ubiquitous and very useful as it allows communication within the network with very little overhead and setup. A MAC (Media Access Control) address is all that is needed for this type of communication. This specified hardware MAC address is often times burned into the NIC(s) (Network Interface Card) of most machines. This MAC address is specified in a 6 hex couples (i.e. a3:3d:ef:aa:bc:e1). MAC addressing within networks is a fundamentally useful abstraction that allows hosts within a network to communicate via ARP enabled switches. These switches behave in a plug-and-play way and learn the topology of the network (i.e. which MAC address is on which output port) over time. However, the designers of ARP protocol made an assumption everyone in the LAN is trusted. Therefore, they didn't consider security when they design the protocol. Attackers exploit this assumption to achieve ARP spoofing which could lead to attacks, ex: man-in-the-middle, denial of service, and spying. To prevent ARP spoofing, people have came up with a simple way which is to make ARP table static. Although it does make ARP protocol secure, it makes ARP less useful. We propose a new protocol, called ARPSec, that achieves security while preserving all the features of ARP.

1.1 How does ARP work:

ARP table provides a mapping between IP addresses and MAC addresses. This mapping is stored on a machine in the form of a table where the key is some IP address and the value is the MAC address associated with the IP address. This mapping is often referred to as ARP cache or ARP table. This table is extremely helpful to improve the performance because network host consults this table to get the MAC address for a given IP if the table contains the mapping and broadcasting to resolve an IP address whenever you want to send a packet to a network host in the local area network is extremely expensive. It could easily cripple your LAN. Besides the mapping, each entry in the table has a Time-To-Live. After TTL, ARP table will delete the entry. This is helpful for avoiding table to be full.

Initially this mapping (will now be referred to as ARP-Table) is empty when the host first comes on the network. In the case where a host has an empty table or simply no entry for some desired IP address, the host broadcasts to the broadcast MAC address (ff:ff:ff:ff:ff:ff) with a query for the unknown IP. This query is in the form of an ARP message where the op field, specifying the message type, is set to 'who-has'. In a 'who-has' query, the message contains some pdest information; this field specifies the IP we want resolved. Finally, since the sender must have a way to receive a response, the query message also contains a source MAC address (hwsrc field). Since this message is broadcasted, all hosts on the network receive this message, however the ARP protocol states that every host that does not have the matching IP should simply drop the packet. The one host which does have this destination IP will respond to the sender with an ARP message encoded with the 'is-at' op. Also, it learns the MAC address of the sender IP and adds the entry to its ARP table. This preemptive caching is efficient because usually the responder host will eventually send messages to the sender host. In this response message, it sends back its own MAC address within the 'hwsrc' field of the response. It uses the requests src field to then send a unicast message to the sender of the query. When the sender gets this unicast message, it caches the entry in its ARP table to avoid next broadcast. The figure below shows a sample ARP request and response as seen in our final presentation test.

Figure 1: Broadcast ARP query (left), Unicast ARP response (right). Important Fields HIghlighted in White.

1.2 Limitations of ARP:

The limitations of ARP are directly related to its purpose. The purpose of ARP is to allow two hosts on the same network to communicate with one another. There is a fundamental assumption made within this paradigm and that is the fact that these hosts are all assumed to be trusted. Therefore, it doesn't have authentication step. While this assumption may have been well-founded when this protocol was invented (in 1982), it is no longer well founded. There are many cases in which a malicious user may be able to join a network. One such example is public

networks, such as the ones seen in starbucks and some hotels. In this case any malicious user can join the local network along with their potential victims. Another such instance is an otherwise benign user being compromised on the network and coming under the control of a malicious user. In both these cases, all hosts on the network blindly trust the ARP responses of these malicious invaders.

1.3 Potential vulnerabilities of ARP:

ARP is is often considered as 2.5 layer in networks because it is widely used to resolve IP addresses into MAC addresses. When a packet needs to be transferred from one host to another in a local area network, the destination IP must be translated into MAC address since any layer below network layer only understands MAC address. To resolve the mapping, an ARP request is sent from the host. The request is a broadcast message that is sent out on the entire LAN. Every host in the LAN will receive the message and is expected to reply only if the IP matches its host. This is known as ARP reply. To avoid sending a broadcast message every time you want to send a message to another host in the LAN, every hosts caches ARP replies. Next time, when the host wants to send a message, it will consult the cache first. If cache has the mapping, the sender uses the entry in the cache. The problem in here is that hosts cache the ARP reply regardless of who sent it. That is, no verification of the identity of the sender is done at all.

This lack of verification can expose a large attack surface to attackers on the network and allow them to spoof ARP messages that can enable them to gain substantial power over the data flow within the network.

Section 2: The Threat Model

2. Threat Model:

ARP spoofing is the act of sending a false ARP message to a host in order to trick them into associating a false MAC address with a particular IP address. ARP spoofing is useful to malicious hosts because it allows them to poison the ARP cache of a victim host on the network. This poisoning can then lead to even more powerful Man-in-the-Middle attacks. There are two features of the ARP protocol that make this possible. First, when an IP address is not known, the resolver will send a broadcast to everyone on the network with the ‘who-has’ message as specified in section 1.1. This means that even the attacker will receive this message. The attacker obviously has no obligation to send an honest response the sender and may advertise the attacker’s mac address as being associated with some other machines IP address. The second vulnerable feature is that the recipient of an ARP response does no identity verification of the message. This means the attacker can poison the victim’s ARP cache with a false IP address ->

MAC mapping. In the future, if the victim wants to contact some target, it will falsely send data to the attacker rather than it's intended recipient. Figure 2 has an example flow of an ARP cache poisoning attack.

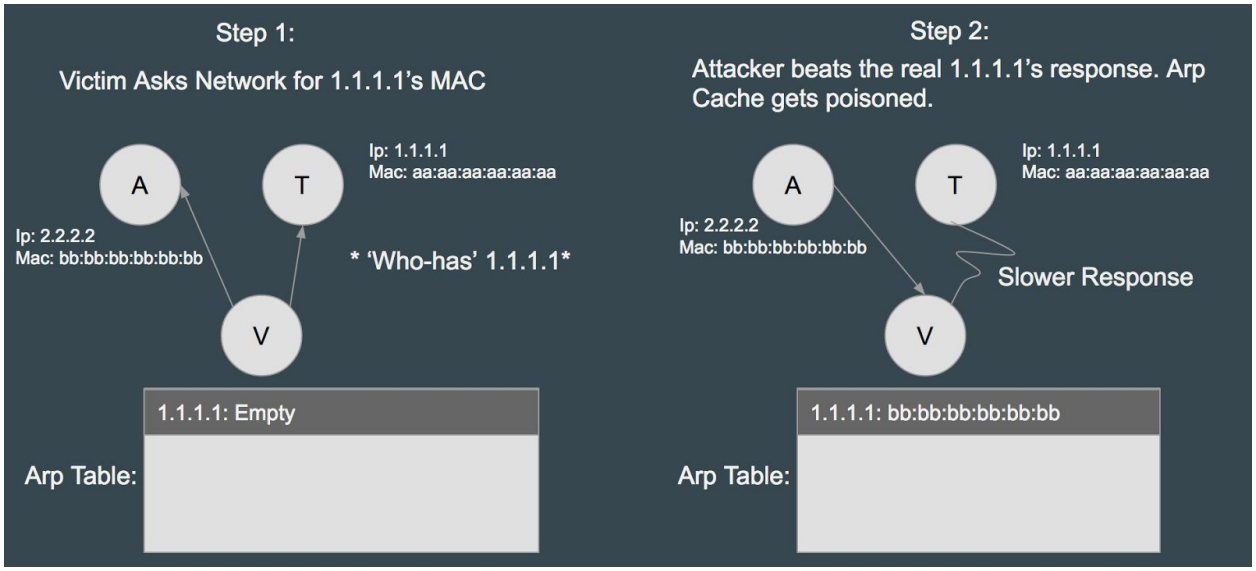

Figure 2: An example ARP-poisoning attack. A=Attacker, T=Target (actual data target), V=Victim (data sender)

As shown above, the attack is simple to execute and there are a myriad of pre-existing tools to do this in an efficient automated manner (Kali Linux comes with a few pre-installed). In the above example, the attacker beats the response of the target but this race is also not necessary. A variant of this attack is to simply spam the recipient with spoofed ARP-responses for some target IP address. In this case, the victim will take this new mapping as a gratuitous update and update the entry anyway. Furthermore, it may seem like that attack cannot happen so long as the target IP address entry continues to stay in the ARP-cache but this is also not the case. An attacker can execute and ARP flood attack where they spam a victim machine with spoofed ARP entries for all ip's in the address space (0.0.0.0 -> 255.255.255.255 or some subset). Because the ARP cache is sure to cache fewer IP's than the whole space, this is sure to overload the ARP cache of the victim and consistently flush the target IP entry from it's cache.

2.1 Attacker's uses of ARP-poisoning:

An ARP-poisoning attack in itself is a powerful attack; however, it is oftentimes the entry attack that leads into the more powerful Man-in-the-Middle attack. In this attack the attacker

simply ARP-poisons two hosts on the network and in the process, effectively places itself between them. Figure 3 shows a particularly dangerous case of Man-in-the-Middle (MITM). In this case the attacker convinces a host that the attacker MAC address corresponds to the IP of the network router. It then also convinces the router that the victim's IP address corresponds to the attacker's MAC address. In this setup the victim and the router will send all data intended for each other to the attacker. It is then the attacker's decision to do with the data whatever they please. The attacker can extract sensitive data if the data is plaintext and can even do so if the data is encrypted. For instance, the attacker can perform an SSL Strip on HTTPS requests by the victim and simply read plaintext data afterwards. The explanation of SSL strip is out of scope for this project but demonstrates the power of a Man-in-the-Middle.

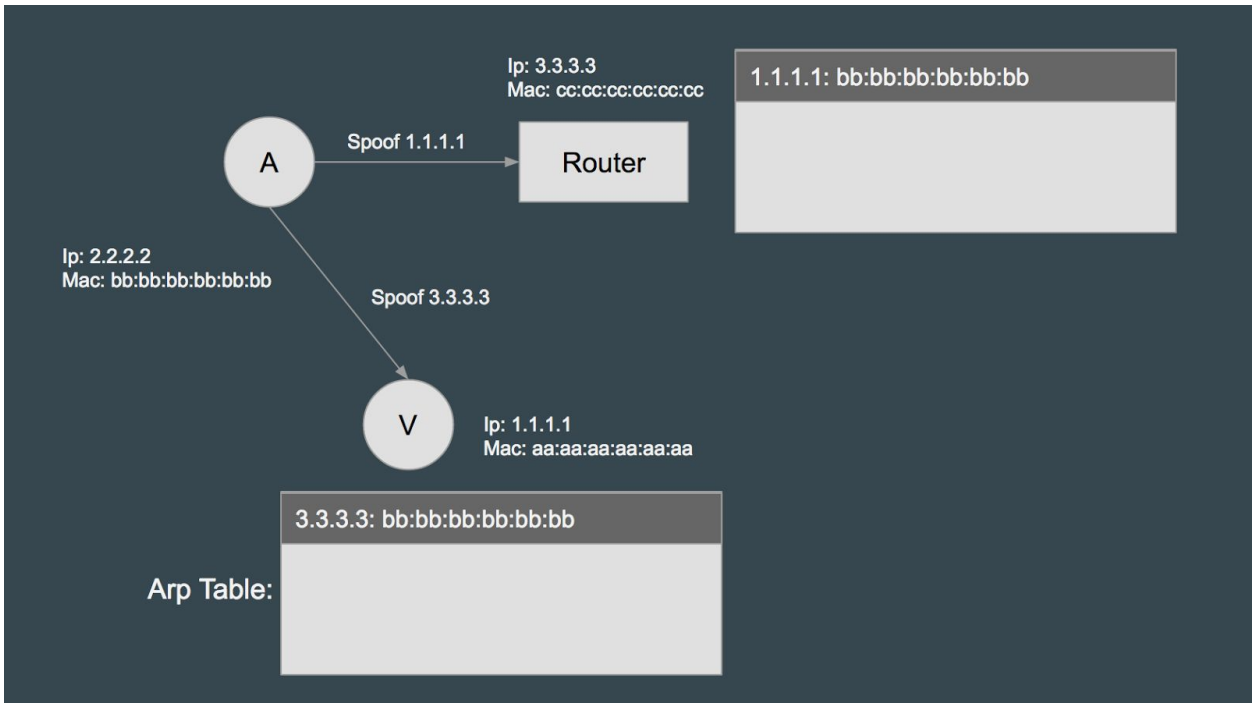

Figure 3: How a Man-in-the-Middle is achieved. A=Attacker, V=Victim

Section 3: Some Considered Approaches

3.1 Approaches of others: #1 - Static ARP tables

The simplest way to prevent ARP spoofing is to use a static and read-only ARP cache. What this means is that ARP table is static and can be only updated by network administrator. Each host in the LAN will ignore all ARP replies by dropping the packets. Therefore, a malicious machine can be only added through network administrator. We know this is unlikely to happen because network administrator could be trusted. Besides, there is no security anymore

if network administrator is compromised. For a network with N machines, network administrator would have to update N-1 machines ARP table if he/she wants a host to join the LAN. In modern network, it is very common to see a network with thousands of machines. As you can see, the work is very cumbersome and tedious. Network administrator already has other things to manage. Adding this to network admin could add more costs to the company. Furthermore, most LAN don't even have a network administrator. In summary, static ARP table does provide security against ARP spoofing, but it results manual management of ARP table which is tedious, hard to manage, doesn't scale, and costs more money.

3.2 Approaches of others: #2 - Automated Approach using Static ARP

In "An Automated approach for Preventing ARP Spoofing Attack using Static ARP Entries" paper, the authors proposed another solution. Their architecture has two parts. They design a protocol on top of static ARP protocol, called client-server protocol. The protocol automatically configures static ARP table when a host joins/leaves LAN or wants to update ARP table. Their approach works without adding overhead to the network admin. Furthermore, it doesn't need any special hardware. Their architecture has two parts: servers and clients. Clients are network hosts, and any host can be a server. The server contains latest ARP table. The protocol has three types of messages. Register message: a unicast message that is sent from the client to the server when the client just joins the LAN. The message contains the client MAC address and IP, and hashed authentication key. Update message: a broadcast notification from the server to all the hosts in the network, telling them that a new host just entered the LAN and what his/her MAC and IP address are. Register Response message: a unicast message that server tells the new client all the trusted ARP entries. The flow works as follows: when a host enters the LAN, it sends register message to the server. The server checks the hashed authentication in the register message. If it is valid, the server adds the new host to the ARP table and ask everyone in the LAN to add this new host to its ARP table. The problem with this approach is that a host can cripple the LAN by just joining and leaving. This will trigger the server to send broadcast every time that a host joins. As we all know, broadcast is expensive and we should avoid it if possible. Besides, verification of signature is all done at the server. This may cause a bottleneck at the server.

3.3 Brief Preamble of Public and Private Key Crypto:

Our approach relies heavily on the digital signatures (DS) provided by asymmetric cryptography suites. Asymmetric crypto is the paradigm where each member in communication has a public and private key. It is assumed that everyone globally knows the public key of everyone else but no one knows the private key of anyone else. Using mathematical proofs

concerning certain one-way functions, it is possible for a party to encrypt a message using the recipient's public key and have the recipient decrypt it using their own private key. In this way, only party A can read messages encrypted with A's public key.

Another feature of these keys is the ability to provide signatures. Signatures are cryptographic hashes that are appended to some sent data. The recipient of the data can then check the signature of the data and affirm the identity of the sender mathematically. We use this feature of cryptography to ensure that ARP messages claiming to be sent by IP address A are in fact sent by the host with that IP address.

Section 4: Our Approach

4.1 Problem Statement:

Our goal is to propose a novel protocol, ARPSec, that adds authentication to regular ARP protocol. The purpose of ARPSec is to successfully prevent ARP spoofing, with a minor performance overhead comparing original ARP protocol. Our approach is different from Automated Approach using Static ARP because our signature verification is done at the each host, rather than the server. Furthermore, each host can't cripple the network if it keeps joining and leaving the LAN because each host will cache the public key and a host will request public key from CA only if the host just joins wants to send a message to another host. Our hypotheses are that our protocol can successfully prevent ARP spoofing with a minor performance overhead.

4.2 Protocol Design:

To prevent ARP spoofing, we will use digital signatures to ensure only valid hosts have entries in the ARP table; this will prevent poisoning. Each host on the network will have a public/private key (RSA 2048 bit) associated with it's IP address. There will be a Certificate Authority (CA) on the network that will need to be set up by a network administrator. The CA will hold the mapping of all IP addresses to their public keys. Furthermore, each host on the network will have knowledge of their public and private keys. For our protocol to work, we rely on 2 assumptions:

1. Every host who joins the network will create a public/private key pair and inform the DHCP server of its public key when it receives and IP. This mapping is then relayed to the CA.
2. Every host who joins the network knows the public key of the certificate authority. This can also be done using Secure DHCP

We believe (1) is a reasonable assumption because many routers already implement a way to assign IPs securely, so the host can send its public key in the DHCP request. Assumption (2) can be accomplished by giving the public key of the CA when the host is assigned an IP.

For implementation purposes, we will run the protocol on application layer. An ARP packet will be sent as normal, but have a signature appended to it in the payload section of UDP. The signature will be verified by the host receiving the response, by validating it with the appropriate public key. The public key is obtained by querying the Certificate Authority and caching the result in memory. Each host will have the public key of the CA, so they can verify responses sent by the CA.

Protocol Steps:

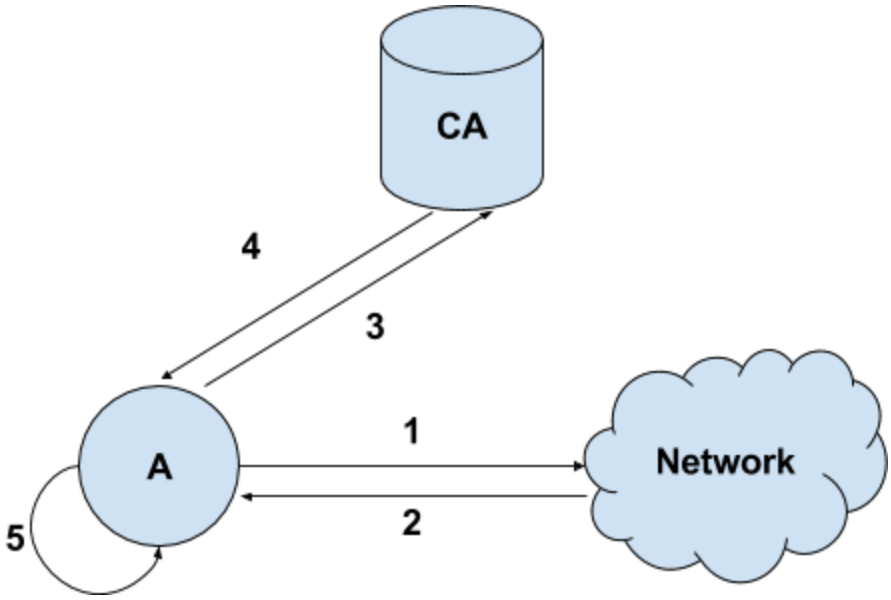

Figure 4: An example network topology of a network.

0. All hosts have a pub/private key and the public key of the CA that can be disseminated via Secure DHCP. The CA has knowledge of this information. This can happen when hosts join the network.
1. Host A joins network, and wants to send data to some node on the network. Host A sends an ARP request for some host's mac address
2. A reply is received by host A from host B in the network. A signature is attached to the ARP packet

3. Host A queries the CA for the public key of Host B
4. The CA responds to A with the public key of Host B. A signature is attached to this message
5. Host A verifies the signature of the message sent by the CA (using the CA's public key). Then it verifies the ARP response using the public key of B to ensure it was sent by B. It then caches the public key of B for later use.

4.3 Protection Against Replay Attacks:

A replay attack occurs when a previously sent packet is replayed and passed as a fresh, valid one. We designed our protocol to prevent replay attacks by making each response unique. When any query is sent (can be ARP or CA query), a random string (nonce) is appended to that message. A valid response message will contain the signed nonce (using the private key) appended to the message. In this way, we can reasonably ensure no two responses will be the same, eliminating the possibility of a replay attack (since the signature validation will fail).

Section 5: Testing/Evaluation

5.1 Evaluation Setup:

We exploit the fact that an ARP reply can be sent to a host to update the host ARP cache table to achieve our demo attack. We used Mininet to create our testing environment, we have four hosts, CA, Victim, Target, Attacker, and S1 (switch). All hosts are directly connected to the switch. We stimulate the ARP layer in the application layer.

5.2 ARP Poisoning in Mininet:

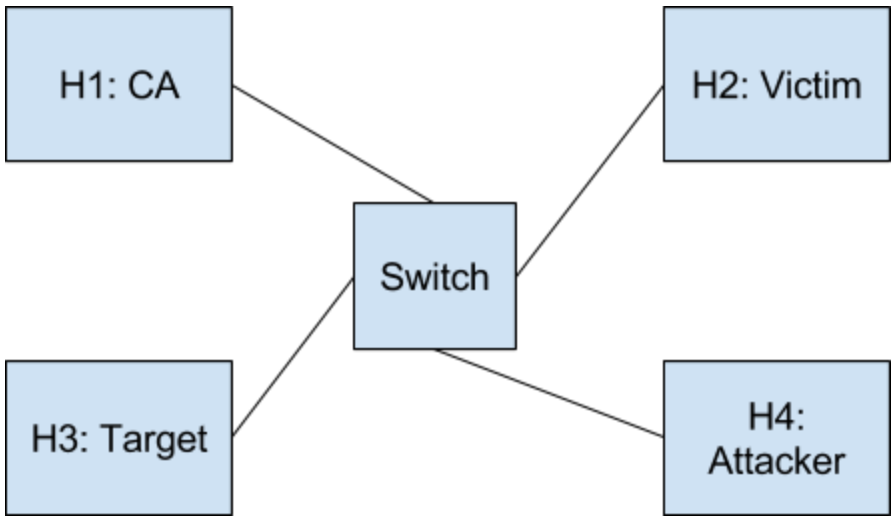

Figure 4: Our mininet network setup

In our setup, the Victim tries to send data to the Target. At first we disable all signature based verification in our protocol and send some data from the victim to the target. Without the attacker's interference, the victim correctly resolves the MAC address of the target and is able to send a Data-Link layer payload to the target correctly. The Target receives this data and prints it to console. We then, with authentication still off, turn on the attacker machine. The attacker sends a spoofed ARP message and poisons the ARP cache of the victim by impersonating the IP of the target. The target then consults it's poisoned ARP cache when it is ready to send data again and reads the MAC address of the attacker. It uses the MAC address thinking that it belongs to the Victim's machine but because it did not verify the received ARP message, it sends the data to the Attacker host instead. The attacker then prints and drops the message.

After this experiment, we turned digital signature verification on. In this case all proceeds as mentioned in the previous case until the Attacker sends the spoofed ARP message. In this case, the Victim consults the CA for the public key of it's intended target and performs a signature verification against the signature provided in the spoofed ARP message. Naturally because the private keys of the target and attacker are different and secret, the verification fails, at this point the ARP message is dropped and the ARP cache is not poisoned. Thus when the victim sends the data, the target receives it correctly and the attacker does not.

5.3 Evaluation Results

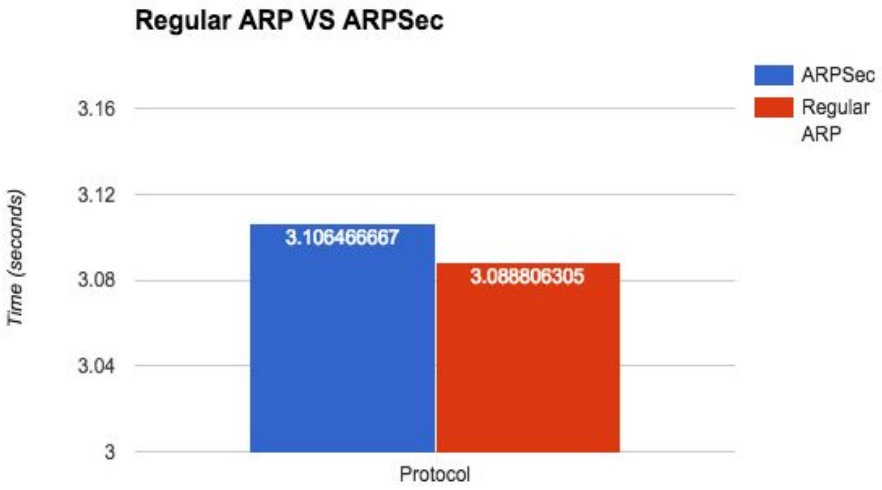

Figure 5: Performance Comparison

We run our application layer ARP protocol without signature verification and our ARP-Sec with verification on three times. The average of time it took for a host to update its

cache table with regular ARP is 3.089 seconds where the average of time for ARP sec is 3.106 seconds. The performance overhead is 17 milliseconds, presumably to verify the signature of a message. We believe there is some internal caching going on to speed up the verification process over multiple calls.

5.4 Conclusion

In conclusion, ARP is an insecure protocol to map IP addresses to Mac addresses. The creators of the protocol did not have security in mind at the time as it was not as much of a factor as it is today. One approach to secure ARP is to have static ARP tables that need to be updated by a network administrator. This is cumbersome and does not scale well for dynamic networks. Another approach was the automated approach using static ARP. In this approach, a centralized server has the updated ARP table, any hosts that joins the network must authenticate with this server. The server then notifies all members of the network about this new host. The downside of this approach is that all the verification of the ARP messages is left completely to the centralized server. This introduces a lot of computation at the central server and may become a bottleneck. Also people can repeatedly join and leave the network to flood the central server with pointless work.

How our protocol digs deeper and deserves that 30%:

Our approach rectifies both of the issues with the other approaches by allowing for dynamic changes of members in the network and de-centralizing the workload for authenticating ARP messages. For the first point, we piggy-back on DHCP to disseminate public/private keys to the hosts and Certificate Authority which will allow for our Certificate Authority to learn the authentication keys of each new host on the network. With regards to the second point we implement a method for each host on the network to do it's own verification thereby reducing the load on the CA. In our implementation the CA is only responsible for delivering public keys of un cached IP's to hosts that need resolve it.

With regards to evaluation we note that the overhead we added with signature verification is close to negligible and is not a problem for commodity hardware of today (using moderately sized keys).

Git: https://github.com/abapat/Mission-ARP-Possible
Presentation:
https://docs.google.com/presentation/d/1gGuEcIyQUMb90qCBGGSWnz5NXg6GHqpZ9HYsHUZ2Atc/edit?usp=sharing