Formally Secure Networking: Securing ARP From the Bottom Up

Jing (Dave) Tian, Kevin Butler, Patrick McDaniel* and Padma Krishnaswamy**

Computer and Information Science Department, University of Oregon *Pennsylvania State University **Federal Communications Commission

The basis for all network communication is the Address Resolution Protocol (ARP), mapping the IP address to a device's Media Access Control (MAC) address. ARP resolution has long been vulnerable to spoofing and enabling other attacks, as shown in Fig 1. Modern public key based solutions cannot tell the real identity and integrity state of the remote. By using the Trusted Platform Module (TPM, a cryptographic hardware chip) and a new formally-defined ARP binding logic system, we introduce a new secure ARP protocol – ARPSEC, which can not only defend from ARP attacks but also attest the remote for real identity and integrity state.

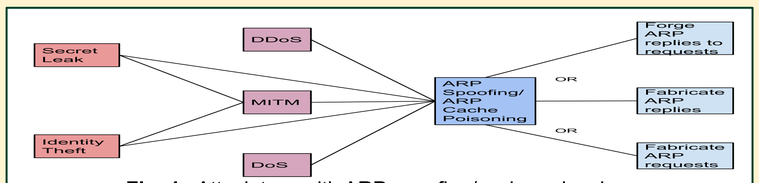


Fig. 1. Attack tree with ARP spoofing/cache poisoning.

ARPSEC

ARPSEC is consisted of a arpsec daemon (arpsecd) in the user space and the kernel changes (arpseck) in the kernel space. The arpsecd is consisted of CPU_read, logic layer (a Prolog query engine) and TPM layer while the arpseck instruments the kernel for ARP processing. With the help of kernel relay and netlink socket, ARPSEC could achieve the high performance of ARP message processing with logic reasoning and TPM hardware attestation.

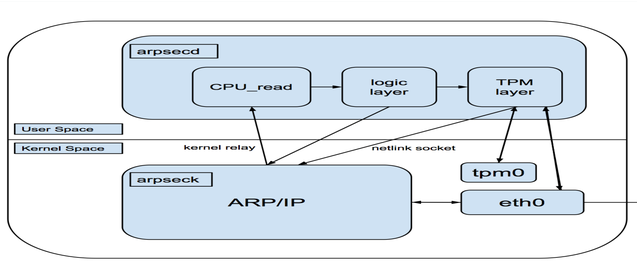


Fig. 2. The architecture of ARPSEC.

Logic and TPM

To formally define the requirement of ARP security, we propose a novel and logic-based ARP binding system. The instance of this system is defined as a 6-element tuple including the time epochs, the network address (IP), the media address (MAC), the name of the system making this assertion about the ARP binding, the timing of system trust validation via system integrity attestation using TPM and the complete binding assertion. Based on these settings, we further define the operational rules to guarantee the trust of the ARP binding. The rule in Fig 3 is saying that any binding, asserted at or prior to the time *l* by a trusted system and no later assertion with the same network address or media address was received, is asserted (trusted).

When the logic layer is not able to validate the ARP response, the TPM layer is in charge of the next.

$$\exists \bar{R}_{i,j,k,x} \in \bar{\mathcal{R}},\ x \le l,,\ s_i \in A_x,$$
$$\nexists \bar{R}_{v,j,p,y} \in \bar{\mathcal{R}}, p \ne k, y > x, s_v \in A_y,$$
$$\nexists \bar{R}_{v,q,k,y} \in \bar{\mathcal{R}}, j \ne q, y > x, s_v \in A_y$$

Fig. 3. A formal rule example

As a software tamper-proof hardware, TPM provides the ability of system measurement and secure storage. The measurement of BIOS, boot loader and OS are saved in the Platform Configuration Registers (PCRs) and could only be read via TPM Quote command. TPM also provides Attestation Identity Key (AIK) for the 3rd party to do identity and integrity state check. To leverage the power of TPM, we design the AT (ARPSEC Attestation) protocol for remote attestation.

First, the challenger (arpsecd) sends out an AT request. The attester then invokes TPM to do the Quote and sends back the signature. Upon receiving the AT reply from the remote, the challenger verifies the PCR digest and the signature. The details of AT protocol is shown in Fig 4.

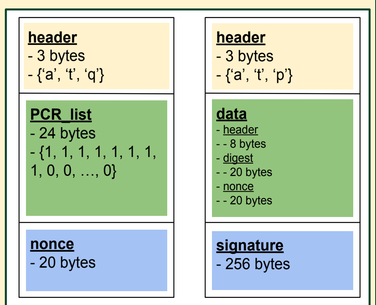
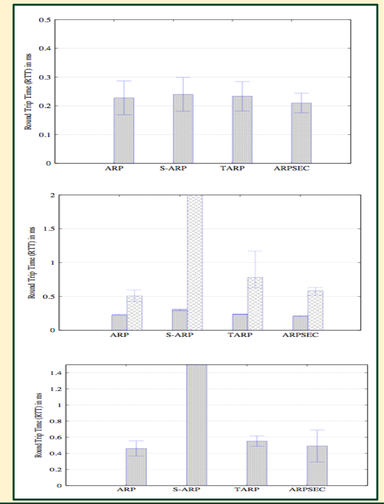


Fig. 4. The AT Request/Reply

Evaluation

To evaluate the overhead of ARPSEC, we compare it with the original ARP implementation in the Linux kernel and two other PKI-based protocols – S-ARP and TARP, using the common ping command and a customized ping (ncping). While Round-Trip-Time (RTT) from ping gives the system overhead from the application view, ncping, clearing the ARP cache before sending each ICMP echo request, reveals the performance in the worst case. All tests are 1000 runs and the timing metric is ms. The left figures, from top to down, show the RTT measurement of different protocols among different cases including ping with caching, ping without caching and ncping. We can see that ARPSEC introduces no overhead in the first case, beats S-ARP and TARP in the second case and shows the smallest overhead in the worst case.



Conclusion

Within the Linux 3.2 kernel environment and evaluation, we show that ARPSEC incurs an overhead ranging from 7% to 15.4% over the standard Linux ARP implementation. More over, this formally-defined protocol based on bottom-up trust provides a first step towards a formally secure, trustworthy networking stack.

OSRIS Oregon Systems Infrastructure Research & Information Security Laboratory

For further information, contact Dave Tian (*daveti@cs.uoregon.edu*).