A New Look at an Old Attack: ARP Spoofing to Create Routing Loops in Ad Hoc Networks

J. David Brown(✉) and Tricia J. Willink

Defence R&D Canada, Ottawa, Canada
{david.brown,tricia.willink}@drdc-rddc.gc.ca

Abstract. This paper examines a new application of the well-known ARP spoofing (or ARP cache poisoning) attack. Traditionally, ARP spoofing has been applied in local area networks to allow an attacker to achieve a man-in-the-middle position against target hosts, or to implement a denial of service by routing messages to non-existent hardware addresses. In this paper, we introduce a variant of ARP spoofing in which a routing loop is created in a target wireless ad hoc network. The routing loop not only results in a denial of service against the targeted hosts, but creates a resource consumption attack, where the targets waste power and occupy the channel, precluding its use by legitimate traffic. We show experimental results of an implementation and provide suggestions as to how to prevent, detect, or mitigate the attack.

Keywords: Denial of service · ARP spoofing · Ad hoc networks
Sensor networks · Routing loops · Resource consumption · DoS defences

1 Introduction

ARP cache poisoning, or ARP spoofing, is a well-known network attack technique against local area networks (LANs) in which an attacker sends spoofed Address Resolution Protocol (ARP) messages to one or more target hosts. ARP spoofing can be performed as the first step in a larger attack, where the end goal of the attacker could be to achieve a man-in-the-middle position between two hosts or to cause a denial of service (DoS) against one or more hosts. The Address Resolution Protocol is vulnerable to spoofing because ARP messages include no authentication (see RFC 826, and updates in RFC 5227, 5494 [1–3]) and thus any host connected to the target network can emit an ARP request or response purporting to come from another host. This technique has been recognized for nearly 20 years, and it remains an area of interest as evidenced by continued activity in the security community, examining techniques to detect it and mitigate it—see for instance [4–7].

While ARP spoofing is usually discussed in the context of wired LANs, it is arguably more damaging—and easier to perform—in wireless ad hoc networks, where hosts are expected to leave and join frequently, the physical communication medium is easily accessible, and there is no central entity for security co-ordination; [14] enunciates the devastating effects of ARP poisoning on ad hoc networks. The "gold standard" of protection against ARP spoofing—namely, hard-coding the MAC and IP address pair

Y. Zhou and T. Kunz (Eds.): AdHocNets 2017, LNICST 223, pp. 47–59, 2018.
https://doi.org/10.1007/978-3-319-74439-1_5

associations in each host—is impractical for many ad hoc network use cases, since it is often the case that the complete set of hosts that participate in an ad hoc network is not known *a priori*. Other existing defences against ARP spoofing rely on making modifications to existing protocols [15]; employing schemes or tools that perform passive monitoring of traffic or internal system parameters [8, 9]; or modifying operating system (OS) configurations. Although these defences are simple and practical, the reality is that they are often not implemented, leaving networks vulnerable [10]. As ad hoc networks become increasingly pervasive—in applications including sensor networks and for devices residing on the "Internet of Things"—it is likely that many of these networks will be developed and deployed without such defences in place since the driver for many industries will be in developing devices of low cost, low complexity, and interoperability. In fact, although it has been pointed out that ARP is not truly suited to ad hoc networks [11], the protocol will no doubt continue to be used in many implementations despite competing suggestions and algorithms.

In this paper, a novel use of the ARP spoofing technique is presented that can create a powerful DoS attack against a target ad hoc network. An attacker injects spoofed ARP packets into the ad hoc network such that a "routing loop" is formed between two or more hosts; as a result, an IP packet directed through any of the affected hosts oscillates "forever" in a loop—or until the packet's time to live (TTL) expires. In this fashion, the attacker exerts relatively little effort (in terms of power resources) but creates a situation where the target network exhausts its own resources and floods the shared wireless channel. The attack is unique to ad hoc networks and does not port directly to the wired case, since in an ad hoc network all hosts on a common subnet can act as routers as well as endpoints, thus presenting the opportunity for creating routing loops among the hosts themselves. This is different from the standard set of ARP spoofing attacks, which generally force hosts to route through the attacker (creating a man-in-the-middle) or direct hosts to route to non-existing addresses (see [14]). While directing a host to a non-existing address results in a link failure (and a DoS against the host), the attack proposed here causes hosts to continue transmitting duplicate copies of packets—effectively depleting battery life and consuming channel resources, thus denying them to other non-targeted hosts.

The remainder of the paper is organized as follows. In Sect. 2, a brief review of the standard ARP spoofing attack is provided followed by a walk-through of a simple example of the new ARP-route-looping attack. Section 3 discusses how the ARP-route-looping attack could be applied to ad hoc networks in general, and identifies required topology pre-conditions that target networks must satisfy in order to allow for a successful effect. Section 4 provides the results of an experiment conducted on an ad hoc network comprised of Android smartphones, showing the effect of ARP-route-looping in a real-world scenario. Finally, Sect. 5 provides suggested defences and mitigations against the attack, with concluding remarks in Sect. 6.

2 The ARP-Route-Looping Attack

This section describes the ARP-route-looping attack by looking at a simple walk-through example. First, a brief description of traditional ARP spoofing is provided.

2.1 Traditional ARP Spoofing

Consider a simple IP network of two hosts, Alice and Bob, where Alice and Bob communicate over a wireless interface. When Alice sends a message to Bob, the message consists of a packet containing Bob's network IP address, denoted here as IP_B. As the packet travels down Alice's protocol stack, Alice's OS adds a hardware (or MAC) address for Bob, denoted here as MAC_B. Alice's OS obtains Bob's MAC from the local ARP cache, which contains a mapping of IPs to MACs for the hosts in the network. If the ARP cache does not contain an entry for Bob, Alice must broadcast an ARP request and wait for Bob's ARP reply (which contains MAC_B). In a traditional ARP spoofing attack, an attacker (Eve) sends spoofed ARP reply messages into the network to mislead Alice and Bob about the mappings of IPs to MACs.

In this scenario, Eve sends ARP spoofing messages to Alice indicating that Bob has hardware address MAC_E: Eve's MAC address. We use the notation $\mathrm{Tx}(E, A, <\mathrm{IP}_B, \mathrm{MAC}_E>)$ to denote that Eve (host h_E) sends a message to Alice (host h_A), where the message consists of an ARP spoof mapping IP_B to MAC_E. Likewise, Eve sends $\mathrm{Tx}(E, B, <\mathrm{IP}_A, \mathrm{MAC}_E>)$, indicating to Bob that Alice has hardware address MAC_E. Thus, Alice unwittingly sends traffic destined for Bob to MAC_E (and Bob sends traffic for Alice to MAC_E). Even in a wireless setting where Alice and Bob can hear all the traffic in the network, they will not process frames addressed to MAC_E (and will only process frames addressed to their own MAC addresses). Thus, once the poisoning is complete, Eve acts as a relay for all traffic between Alice and Bob and can modify, re-route, or drop packets as desired.

2.2 A Simple Example of ARP-Route-Looping

The ARP-route-looping attack is easily explained using a simple example. Consider an ad hoc network of four hosts (A, B, C and D), which we denote as h_A, h_B, h_C, and h_D. In this example, the ad hoc network is a complete graph, meaning that every host is within range of every other host. Thus in the absence any disruptions, all hosts can communicate with one another directly (i.e., without requiring multi-hop routes). The simple network is depicted in Fig. 1, where links are shown as light blue lines.

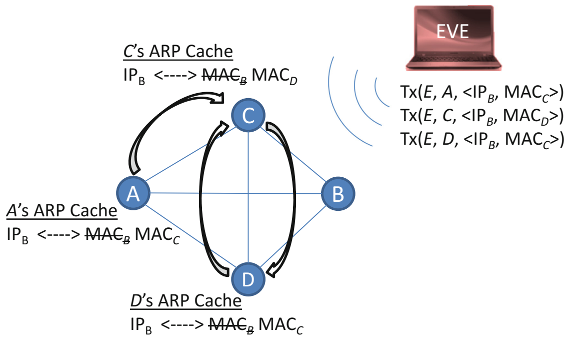


Fig. 1. An example ARP-route-looping attack. (Color figure online)

Suppose an attacker, Eve, wants to disrupt communication between h_A and h_B using ARP-route-looping. Initially, hosts h_A, h_C, and h_D all have direct routes to h_B, along with ARP caches correctly mapping IP_B to MAC_B. First, Eve sends Tx(E, A, <IP_B, MAC_C>) to h_A, poisoning the ARP cache of h_A such that the MAC of h_C is associated with the IP for h_B. Thus, whenever h_A wants to send any unicast messages to h_B, h_A will address the messages with the IP address of h_B, but the MAC address of h_C. Next, Eve sends Tx(E, C, <IP_B, MAC_D>), poisoning the ARP cache of h_C such that all traffic from h_C intended for h_B will be sent to h_D. Finally, Eve sends Tx(E, D, <IP_B, MAC_C>), poisoning the ARP cache of h_D such that all traffic from h_D intended for h_B will be sent to h_C.

Eve's activities are now finished and the conditions for ARP-route-looping have been set. Consider the following steps that inform the flow of a unicast packet that h_A sends to h_B in this scenario:

1. h_A constructs a packet and inserts the IP address for h_B;
2. h_A consults its routing table and determines that it has a direct route to h_B;
3. h_A adds the MAC address for h_B to the packet; h_A consults its ARP cache to determine the MAC address for h_B;
4. h_A inserts the (poisoned) entry MAC_C and sends the packet;
5. h_C receives the packet, examines the IP address and finds that the packet is destined for h_B; since the network is ad hoc, h_C has an IP-forwarding capability and so forwards the packet to h_B (the intended destination according to the IP address);
6. h_C has a direct link to h_B according to its routing table, so it must update the MAC address with the entry for h_B before forwarding the packet; h_C consults its ARP cache for the MAC of h_B and inserts the (poisoned) entry MAC_D, then forwards the packet;
7. h_D receives the packet, examines the IP address and finds that the packet is destined for h_B and so forwards the packet to h_B;
8. h_D consults its ARP cache to determine the MAC address for h_B and inserts (poisoned) entry MAC_C.

At this point, the cycle repeats and returns to step 5, with h_C once again receiving the packet. The packet will continue to cycle between h_C and h_D until the TTL counter for the packet reaches zero. This not only creates a DoS between hosts h_A and h_B, but also creates a situation where h_C and h_D occupy the channel (precluding legitimate usage) and exhaust resources by transmitting duplicate packets in a loop.

When the TTL expires, h_D sends an ICMP time exceeded message to h_A (the source of the initial packet), which indicates that the TTL field in the IP header has reached zero. If desired, Eve can set conditions such that this ICMP message follows a routing loop as well, making it loop between hosts h_C and h_B until its own TTL reaches zero.

3 Generalized ARP-Route-Looping in Ad Hoc Networks

The ARP-route-looping attack was introduced in Sect. 2 for a specific example network. This section describes the technique in general terms and discusses how to determine which hosts to poison and what content to place in the spoofing messages.

3.1 Notation and Assumptions

We begin by introducing additional notation to describe the general ARP-route-looping attack. Consider a network of n hosts, where the network is represented as a graph. If an edge exists between any two hosts, h_i and h_j, they are "neighbours". The neighbour-set of any host h_i is denoted by $\boldsymbol{N}(i)$, and consists of the set of all neighbours of h_i. Note that sets are denoted with bold italicized typeface, and hosts in a set are denoted by their indices for simplicity (e.g., we write $\{A\}$ instead of $\{h_A\}$). In the network from Fig. 1, for instance, $\boldsymbol{N}(C) = \{A, B, D\}$. We denote the relative complement of a set $\boldsymbol{N}(i)$ and a set of hosts $\boldsymbol{H}$ by $\boldsymbol{N}(\mathrm{i})\backslash\boldsymbol{H}$: this is the set of hosts in $\boldsymbol{N}(i)$ excluding the hosts in $\boldsymbol{H}$. So, for instance, in Fig. 1, $\boldsymbol{N}(C)\backslash\{A, B\} = \{D\}$.

We say there is a route or path between two hosts, h_i and h_j, if there exists a sequence of edges in the graph connecting h_i and h_j through some set of vertices. Assuming the network employs a shortest-path routing algorithm, we denote by $r(i, j)$ the first host (after h_i) along the route from h_i to h_j. In cases where there is more than one possible shortest path and $r(i, j)$ could have multiple values, for simplicity we select the host with the lowest index.

We remark that to achieve ARP-route-looping, the attacker must transmit a series of ARP spoofing messages to various hosts in the network. In a geographically diffuse ad hoc network it is possible that a single attacking node would be insufficient to reach all target hosts. For the purposes of this paper, however, we assume all spoofing packets arise from a single attacker, Eve, where it is understood that this may in fact consist of multiple co-ordinated transmitting stations. Furthermore, we assume that Eve has knowledge of the adjacency matrix of the graph representing the network—that is, Eve can compute which hosts are neighbours and can compute the shortest path routes between hosts in the network. Admittedly, in a dynamic network, this knowledge may be challenging to achieve; one possible strategy is for Eve to observe routing control messages and attempt to infer the adjacency matrix from these.

Finally, we note that any routing loop will terminate if the looping packet arrives at either the originator of the packet (since a host will not forward a packet for which it is identified as the source IP address) or the intended destination for the packet. Thus, in creating an ARP-route-loop both of these hosts must be avoided.

3.2 Two-Host Loops

Consider a network of n hosts, where Eve wants to implement ARP-route-looping against messages sent from h_A to h_B. The simplest loop is one that involves only two hosts (neither of which is h_A or h_B). An ARP-route-looping attack can be mounted in this case if the following condition holds:

$$\exists i \in \boldsymbol{N}(A)\backslash\{B\} \text{ such that } \boldsymbol{N}(i)\backslash\{A,B\} \neq \varnothing. \tag{1}$$

Condition (1) says that for two-node looping to be possible there must exist a host, h_i, that is a neighbour of h_A, but is not equal to the destination h_B; furthermore, h_i must have at least one neighbour that is equal to neither h_A nor h_B. When this condition is satisfied, we denote by h_j a node in the (non-empty) set $\boldsymbol{N}(i)\backslash\{A, B\}$. To create an ARP-route-loop, Eve can send out the following three ARP spoofing messages: Tx(E, A, <$IP_{r(A,B)}$, MAC_i>), Tx(E, i, <$IP_{r(i,B)}$, MAC_j>), Tx(E, j, <$IP_{r(j,B)}$, MAC_i>). For hosts h_A, h_i, and h_j, these spoofing messages associate the IP for the next hop on the route to h_B with a poisoned MAC address. Eve's work is done and a routing loop is created between hosts h_i and h_j.

3.3 Loops with More Than Two Hosts

Although a loop with only two hosts is sufficient to perform ARP-route-looping, Eve may be interested in creating a situation where a message from h_A to h_B is looped among more than two hosts. This could have the effect of wasting the resources of more hosts in the network, and in the case of a geographically diffuse network an attacker may wish to occupy the channel over a larger geographic area by involving more hosts in the loop.

Ultimately, to achieve a multi-host loop attack involving k hosts ($k > 2$), Eve needs to identify a path in the graph originating at host h_A that contains a loop of k hosts, where host h_B is not part of the path. Finding loops, or cycles, in graphs is a well-studied problem—see, for instance, [12] and references therein. Although some optimizations to the problem exist in certain cases, for small graphs a brute force search is simple and not onerous to implement. A recursive brute force algorithm that Eve can use to find an appropriate cycle of length k is provided below in Algorithm 1, which is reminiscent of a depth-first search algorithm (e.g., [13]) with minor alterations. Note that *a priori*, Eve does not know whether the network contains a loop of length k and thus would run Algorithm 1 for all values k of interest.

Algorithm 1: *LoopSearch$_k$(i)*

```
1:  push($\underline{\mathbf{S}}$, $i$)
2:  **If** $\underline{\mathbf{S}}$ contains a loop of length $k$ **then**
3:      EXIT → sequence of hosts in $\underline{\mathbf{S}}$ is desired answer
4:  **Else**
5:      $\mathbf{Current} = \mathbf{N}(i) \setminus \{A, B, \underline{\mathbf{S}} \setminus k^{\text{th}}\text{-last})\}$
6:      **If** ($\mathbf{Current} = \emptyset$) or (LoopSearch$_k$ completed for all elements in $\mathbf{Current}$)
7:          pop($\underline{\mathbf{S}}$); return
8:      **Else**
9:          **For** each element $j$ in $\mathbf{Current}$
10:             LoopSearch_k($j$)
11:         **End For**
12:     **End If**
13: **End if**
14: **End**
```

Algorithm 1 includes the concept of a stack, which represents an ordered sequence of hosts—we denote the stack as a vector, $\underline{\mathbf{S}}$ (where we denote vectors using bold and underlining). The operator push($\underline{\mathbf{S}}$, i) means to add host h_i as the last element of vector $\underline{\mathbf{S}}$, and the operator pop($\underline{\mathbf{S}}$) means to remove the last element of vector $\underline{\mathbf{S}}$. When the stack, $\underline{\mathbf{S}}$, is used as part of a set, as in $\mathbf{C} = \{A, B, \underline{\mathbf{S}}\}$, the meaning is that all hosts in the stack are to be included in $\mathbf{C}$. When we write "$\underline{\mathbf{S}} \setminus k^{\text{th}}$-last", this refers to all hosts in $\underline{\mathbf{S}}$ except the k^{th}-last element. To find a loop of length k for a message sent by h_A to h_B, we start with an empty stack (i.e., $\underline{\mathbf{S}}$ = [ ]) and run LoopSearch$_k$(A). Upon completion, either LoopSearch$_k$(A) will terminate with an empty stack (and the network contains no loop of length k) or LoopSearch$_k$(A) will terminate with a non-empty stack, where $\underline{\mathbf{S}}$ contains the sequence of hosts containing a length-k loop for traffic originating at h_A.

At a high level, Algorithm 1 works as follows. When LoopSearch is called for any host h_i, the host is pushed onto the stack. Then we check if the stack in its current form contains a loop; if so, we are done. If not, we recursively run LoopSearch again for all the neighbours of h_i, unless those neighbours are A, B, or other values currently in $\underline{\mathbf{S}}$ except for the k^{th}-last value in $\underline{\mathbf{S}}$ (because we do not want to find any loops in the network except loops of length k). Running LoopSearch in this fashion and excluding A, B, and $\underline{\mathbf{S}}$ is a generalization of condition (1) for the k-loop case. Running LoopSearch on h_A searches for a path containing a cycle, where the path originates at host h_A. As an example, running LoopSearch$_4$(A) on the graph in Fig. 2 provides the output: $\underline{\mathbf{S}}$ = [A, 4, 5, 6, 7, 9, 0, 6], identifying the 4-host loop among h_6, h_7, h_9, and h_0.

Once Eve has determined a sequence of hosts containing a loop of length k, ARP spoof packets can be crafted and sent. Consider that Eve has run Algorithm 1, which returned a vector of hosts, $\underline{\mathbf{S}}$, where $\underline{\mathbf{S}}$ contains m elements. In this case, Eve can craft (m – 1) ARP spoof messages as follows:

$$\text{Tx}(E, \underline{\mathbf{S}}_i, < IP_{r(\underline{\mathbf{S}}_i, B)}, \text{MAC}_{\underline{\mathbf{S}}(i+1)} >), \forall i \in [1, (m-1)], \quad (2)$$

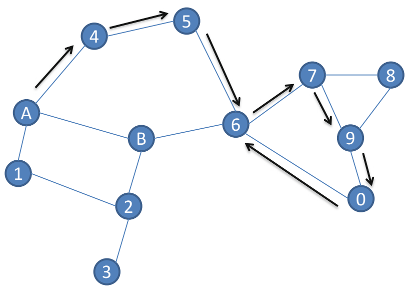


Fig. 2. Finding a loop of length $k = 4$ in a general graph with Algorithm 1.

where $\underline{\boldsymbol{S}}_i$ corresponds to the i^{th} host appearing in vector $\underline{\boldsymbol{S}}$. Thus, to the i^{th} host in $\underline{\boldsymbol{S}}$, Eve sends a spoofed message that maps the IP of the next host *en route* to h_B as corresponding to the MAC of the $(i + 1)^{\text{st}}$ host in $\underline{\boldsymbol{S}}$.

4 Experimental Results

To evaluate the effect of ARP-route-looping on an ad hoc network, we built a test network using Android smartphones (specifically, we used Nexus 5 model smartphones running the Cyanogenmod 13 Android-based operating system). The phones were configured to support ad hoc networking and multi-hop IP forwarding. We constructed the network shown in Fig. 3, where hosts h_A and h_B communicate via a multi-hop route through h_1 and h_2. The hosts in the network ran the optimum link state routing (OLSR) protocol to compute their neighbours and routes. To examine network traffic, we used a laptop running Wireshark in monitor mode as a packet sniffer.

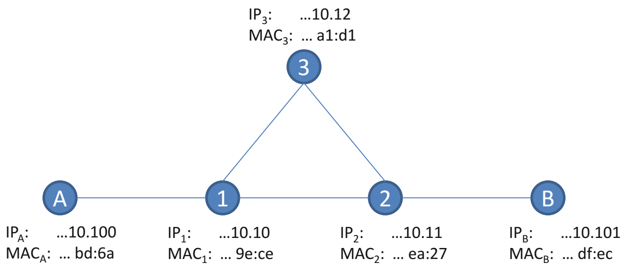
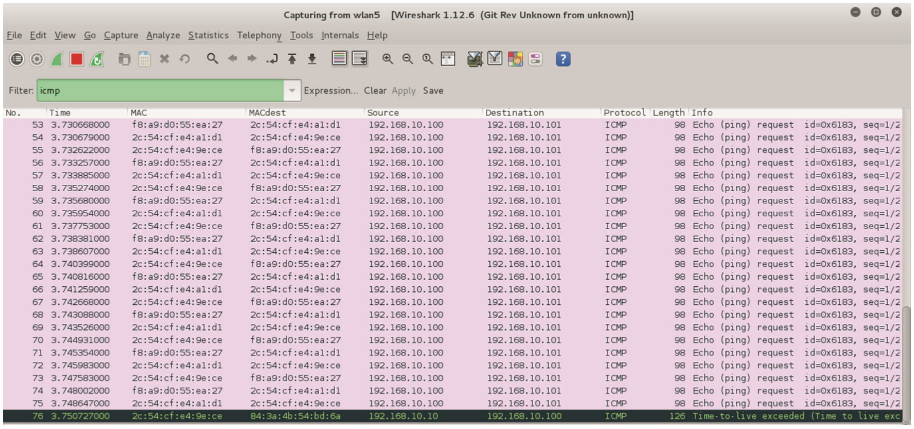


Fig. 3. Experimental network consisting of Android ad hoc hosts; the last two octets of the IPs and MACs are shown for each host.

In this scenario, we examined the ability of ARP-route-looping to disrupt communications from h_A to h_B; to target multiple hosts, we applied Algorithm 1 to find a 3-host loop, yielding $\underline{\boldsymbol{S}} = [A, 1, 2, 3, 1]$ (in this case, of course, we could have found the loop by inspection). To create ARP-route-looping in the network, we sent ARP spoofing messages as follows, based on Eq. (2): Tx(E, A, <IP$_1$, MAC$_1$>), Tx(E, 1, <IP$_2$, MAC$_2$>), Tx(E, 2, <IP$_B$, MAC$_3$>), Tx(E, 3 <IP$_2$, MAC$_1$>).

We expected the spoofing messages to create a routing loop such that all traffic sent from h_A to h_B would cycle around in a loop of hosts h_1, h_2, and h_3 until the TTL of the packet expires. Figure 4 shows a screen capture of our Wireshark packet sniffer when a single ICMP ping is sent from h_A to h_B in the presence of ARP-route-looping. In reading the figure, note that IP_A = 192.168.10.100 and IP_B = 192.168.10.101; for reference, MAC addresses of the hosts in the network are shown in Fig. 3. The single ping from h_A can be seen looping continuously among three hosts (note the MAC addresses in the Wireshark capture). Eventually—after 64 packets have been forwarded around the loop—the TTL of the packet expires and we see the ICMP TTL exceeded message returned to h_A. Not shown in Fig. 4 is the fact that we could have also simultaneously poisoned h_B, h_1, h_2, and h_3 creating a situation where the ICMP TTL exceeded message itself is looped 64 times among h_B, h_2, and h_3.


Capturing from wlan5 [Wireshark 1.12.6 (Git Rev Unknown from unknown)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Expression... Clear Apply Save

Filter: icmp

No.	Time	MAC	MACdest	Source	Destination	Protocol	Length	Info
53	3.730668000	f8:a9:d0:55:ea:27	2c:54:cf:e4:a1:d1	192.168.10.100	192.168.10.101	ICMP	98	Echo (ping) request id=0x6183, seq=1/2
54	3.730679000	2c:54:cf:e4:a1:d1	2c:54:cf:e4:9e:ce	192.168.10.100	192.168.10.101	ICMP	98	Echo (ping) request id=0x6183, seq=1/2
55	3.732622000	2c:54:cf:e4:9e:ce	f8:a9:d0:55:ea:27	192.168.10.100	192.168.10.101	ICMP	98	Echo (ping) request id=0x6183, seq=1/2
56	3.733257000	f8:a9:d0:55:ea:27	2c:54:cf:e4:a1:d1	192.168.10.100	192.168.10.101	ICMP	98	Echo (ping) request id=0x6183, seq=1/2
57	3.733885000	2c:54:cf:e4:a1:d1	2c:54:cf:e4:9e:ce	192.168.10.100	192.168.10.101	ICMP	98	Echo (ping) request id=0x6183, seq=1/2
58	3.735274000	2c:54:cf:e4:9e:ce	f8:a9:d0:55:ea:27	192.168.10.100	192.168.10.101	ICMP	98	Echo (ping) request id=0x6183, seq=1/2
59	3.735680000	f8:a9:d0:55:ea:27	2c:54:cf:e4:a1:d1	192.168.10.100	192.168.10.101	ICMP	98	Echo (ping) request id=0x6183, seq=1/2
60	3.735954000	2c:54:cf:e4:a1:d1	2c:54:cf:e4:9e:ce	192.168.10.100	192.168.10.101	ICMP	98	Echo (ping) request id=0x6183, seq=1/2
61	3.737753000	2c:54:cf:e4:9e:ce	f8:a9:d0:55:ea:27	192.168.10.100	192.168.10.101	ICMP	98	Echo (ping) request id=0x6183, seq=1/2
62	3.738381000	f8:a9:d0:55:ea:27	2c:54:cf:e4:a1:d1	192.168.10.100	192.168.10.101	ICMP	98	Echo (ping) request id=0x6183, seq=1/2
63	3.738607000	2c:54:cf:e4:a1:d1	2c:54:cf:e4:9e:ce	192.168.10.100	192.168.10.101	ICMP	98	Echo (ping) request id=0x6183, seq=1/2
64	3.740399000	2c:54:cf:e4:9e:ce	f8:a9:d0:55:ea:27	192.168.10.100	192.168.10.101	ICMP	98	Echo (ping) request id=0x6183, seq=1/2
65	3.740816000	f8:a9:d0:55:ea:27	2c:54:cf:e4:a1:d1	192.168.10.100	192.168.10.101	ICMP	98	Echo (ping) request id=0x6183, seq=1/2
66	3.741259000	2c:54:cf:e4:a1:d1	2c:54:cf:e4:9e:ce	192.168.10.100	192.168.10.101	ICMP	98	Echo (ping) request id=0x6183, seq=1/2
67	3.742668000	2c:54:cf:e4:9e:ce	f8:a9:d0:55:ea:27	192.168.10.100	192.168.10.101	ICMP	98	Echo (ping) request id=0x6183, seq=1/2
68	3.743088000	f8:a9:d0:55:ea:27	2c:54:cf:e4:a1:d1	192.168.10.100	192.168.10.101	ICMP	98	Echo (ping) request id=0x6183, seq=1/2
69	3.743526000	2c:54:cf:e4:a1:d1	2c:54:cf:e4:9e:ce	192.168.10.100	192.168.10.101	ICMP	98	Echo (ping) request id=0x6183, seq=1/2
70	3.744931000	2c:54:cf:e4:9e:ce	f8:a9:d0:55:ea:27	192.168.10.100	192.168.10.101	ICMP	98	Echo (ping) request id=0x6183, seq=1/2
71	3.745354000	f8:a9:d0:55:ea:27	2c:54:cf:e4:a1:d1	192.168.10.100	192.168.10.101	ICMP	98	Echo (ping) request id=0x6183, seq=1/2
72	3.745983000	2c:54:cf:e4:a1:d1	2c:54:cf:e4:9e:ce	192.168.10.100	192.168.10.101	ICMP	98	Echo (ping) request id=0x6183, seq=1/2
73	3.747583000	2c:54:cf:e4:9e:ce	f8:a9:d0:55:ea:27	192.168.10.100	192.168.10.101	ICMP	98	Echo (ping) request id=0x6183, seq=1/2
74	3.748002000	f8:a9:d0:55:ea:27	2c:54:cf:e4:a1:d1	192.168.10.100	192.168.10.101	ICMP	98	Echo (ping) request id=0x6183, seq=1/2
75	3.748647000	2c:54:cf:e4:a1:d1	2c:54:cf:e4:9e:ce	192.168.10.100	192.168.10.101	ICMP	98	Echo (ping) request id=0x6183, seq=1/2
76	3.750727000	2c:54:cf:e4:9e:ce	84:3a:4b:54:bd:6a	192.168.10.10	192.168.10.100	ICMP	126	Time-to-live exceeded (Time to live exc

Fig. 4. Wireshark display showing the effect of ARP-route-looping on a ping from h_A to h_B

While the looping of a single ping message is an interesting curiosity, the serious potential detrimental effects of ARP-route-looping are shown in Fig. 5. In our test network, we examined a scenario where host h_A streams UDP traffic at a rate of 10 kB/s to h_B. Before ARP spoofing, the traffic is delivered and the load on the network (i.e., the total number of UDP packets transmitted in the network) is seen to be approximately 15 packets per second—the blue curve in Fig. 5. After creating a 3-host ARP-route loop (among h_1, h_2, and h_3), the load on the network increases to an average of approximately 305 packets per second—the red curve in Fig. 5. Clearly, in this case ARP-route-looping has created a situation where hosts h_1, h_2, and h_3 expend considerably more energy and occupy the channel for a considerably greater period of time than if the loop were not present.

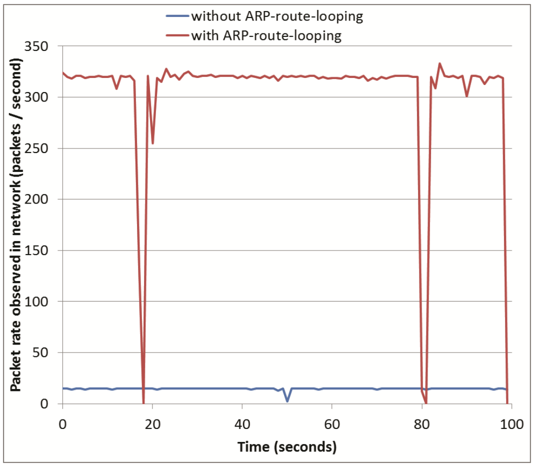


Fig. 5. Packet rate observed for UDP traffic from h_A to h_B with and without ARP-route-looping. (Color figure online)

In general, for an ARP-route-looping attack against host h_A sending to h_B, we expect the packet rate (of affected traffic) to increase by a factor of μ, which we call the traffic multiplication factor. For the UDP example considered here, μ_{UDP} is computed as follows:

$$\mu_{\mathrm{UDP}} = \left(\mathrm{TTL}_{\mathrm{init}} + 1\right) \big/ (\dim(\underline{\boldsymbol{R}}(A, B)) - 1). \tag{3}$$

Here, $\mathrm{TTL}_{\mathrm{init}}$ is the initial time-to-live value for IP packets generated in the network, $\underline{\boldsymbol{R}}(A, B)$ is a vector containing the hosts along the shortest-path route from host h_A to host h_B, and $\dim(\underline{\boldsymbol{R}})$ is the number of elements in vector $\underline{\boldsymbol{R}}$. The numerator contains a "plus one" to account for the additional packet required for the ICMP TTL exceeded message returned to h_A; in the case where ARP-route-looping is also implemented against the return message ICMP TTL exceeded message, then $(\mathrm{TTL}_{\mathrm{init}} + 1)$ is replaced by $(2 \cdot \mathrm{TTL}_{\mathrm{init}})$.

In our test case, $\underline{\boldsymbol{R}}(A, B) = [A, 1, 2, B]$ and $\mathrm{TTL}_{\mathrm{init}} = 64$. Thus we compute $\mu_{\mathrm{UDP,expected}} = (64 + 1)/(4 - 1) = 21.7$. In our UDP example in Fig. 5, we measure the traffic multiplication factor as $\mu_{\mathrm{UDP,observed}} = 305$ fps/15 fps $= 20.3$, which is quite close to our expectation.

Finally, we note that Eq. (3) does not account for any return traffic or acknowledgements in computing the denominator. This is valid for the UDP traffic streams considered in Fig. 5; for an ICMP ping, however, we would expect the denominator to be doubled since each ping request results in a ping reply; thus

$$\mu_{\mathrm{ping}} = \left(\mathrm{TTL}_{\mathrm{init}} + 1\right) \big/ (2 \cdot (\dim(\underline{\boldsymbol{R}}(A, B)) - 1)). \tag{4}$$

For TCP, the effect is more complex since it depends at what stage in the TCP session the ARP-route-loop is established. If an ARP-route-loop is created before h_A and h_B

begin a TCP session, then the session is precluded from starting since the initial SYN from h_A will loop in the network without ever reaching h_B. If the session is already established, then TCP traffic—along with myriad retries—will loop among h_1, h_2, and h_3 without ever being delivered to h_B.

5 ARP-Route-Looping Defences

In this section, we briefly outline a few common defences against ARP spoofing, and introduce new mitigations specific to ARP-route-looping.

5.1 Prevention

Since the ARP protocol includes no authentication, other means are needed to prevent an ARP spoofing attack. Well-known methods are listed below.

- Hard-code fixed ARP tables: For all hosts in the network, permanently fix the IP and MAC address mappings and do not use ARP messages. While effective, this may be impractical depending upon the network deployment.
- Dynamic ARP inspection and DHCP snooping: This is a service available on certain switches and validates ARP messages based on previous DHCP assignments. This is not a realistic solution for an ad hoc network, whose hosts likely do not support the service, nor does the network likely utilize a central switch that could drop ARP messages, and it may not employ DHCP.
- Employ a non-standard protocol that includes authentication, e.g., S-ARP [15].
- Do not use ARP: This is a tautological statement, in that by avoiding ARP it is possible to avoid inherent security problems with ARP. For ad hoc networks, however, this is a legitimate and important option; it has been identified in [11] that other techniques should be explored for address resolution.

5.2 Detection

Existing tools such as ARP Watch [8] and ARP Guard [9] allow network administrators to gather a log of IP-MAC address pairs and perform a forensic analysis to determine if ARP spoofing has taken place. These systems identify when IP-MAC pairs have changed and can flag or alert an administrator when this occurs. For unattended ad hoc networks, such systems would provide a means for after-action analysis, but may not be helpful while the attack is occurring. For the specific case of an ARP-route-looping attack in an ad hoc network, we propose the following detection schemes.

- Record/flag duplicate packets: If a host observes that it is forwarding a duplicate packet (i.e., one it has already forwarded), where everything is unchanged with the exception of having a smaller TTL value, this is a strong indication that the host is part of a routing loop. If this behavior is observed over and over again during a short time span, it is all but confirmed.

- Detect duplicate MAC addresses in the ARP cache: If the ARP cache contains two or more different IP addresses corresponding to the same MAC address, this is a red flag in an ad hoc network and may be suggestive that an attacker is attempting to misdirect traffic along unintended routes.

5.3 Mitigation

As noted above, in an ad hoc network without an administrator actively monitoring network health, simply detecting an ARP spoofing attack is not enough—hosts must be able to take action or have methods in place to mitigate the attack as well. For the specific case of ARP-route-looping, we propose two possible mitigation strategies.

- Reduce the default $\mathrm{TTL}_{\mathrm{init}}$: By Eq. (3), if we reduce the initial TTL for packets generated in the network, we in turn will reduce the traffic multiplication factor, μ, thus dampening the severity of the attack. In an ad hoc network with n hosts, if all traffic is expected to be limited to the local network then it is reasonable to set the initial TTL value to n (or less), since packets should not hop through any host more than once. Note that this solution is only practical, however, if it is not expected that traffic will travel outside the ad hoc network.
- Drop duplicate packets: Further to the detection strategy proposed above, not only could hosts detect duplicate packets (where only the TTL value changes), but hosts could drop duplicates when they are seen. This still will not restore connectivity between h_A and h_B, but it will reduce the severity of the resource depletion and channel occupancy.

6 Conclusion

This paper introduced a new application of ARP spoofing that creates routing loops in an ad hoc network, such that hosts in the network continuously forward packets around the network without the packets ever reaching their intended destination. This so-called ARP-route-looping results in hosts depleting their resources—i.e., a resource consumption attack—and increases channel occupancy in the network. In essence, the hosts are misled into a situation where they deny access to the network and resources to one another by amplifying existing network traffic. This is different from typical ARP spoofing attacks, which are often intended to create a man-in-the-middle situation, a situation where traffic is simply dropped, or an ARP flooding situation where the attacker must expend considerable resources.

In this paper we discussed that in ad hoc networks, where every host acts as an endpoint as well as a router, ARP spoofing remains a serious concern and in fact leaves open another vulnerability: ARP-route-looping. While there are many well-known defences against ARP spoofing attacks, unfortunately these are often not implemented. In addition to these well-known preventative techniques, we proposed additional mitigation strategies specific to ARP-route-looping. We hope that this new application will further emphasize the importance of taking steps to avoid this common, yet avoidable, vulnerability.

References

1. Plummer, D.C.: An Ethernet address resolution protocol. RFC 826, November 1982 (1982). http://tools.ietf.org/html/rfc826
2. Cheshire, S.: IPv4 address conflict detection. RFC 5227, July 2008 (2008). http://tools.ietf.org/html/rfc5227
3. Arkko, J., Pignataro, C.: IANA allocation guidelines for the address resolution protocol (ARP). RFC 5494, April 2009 (2009). http://tools.ietf.org/html/rfc5494
4. Mangut, H.A., Al-Nemrat, A., Benzaid, C., Tawil, A.H.: ARP cache poisoning mitigation and forensics investigation. In: Proceedings of 14th IEEE International Conference on Trust, Security, Privacy in Computing and Communications, Helsinki, Finland (2015)
5. Yang, M., Wang, Y., Ding, H.: Design of WinPcap based ARP spoofing defense system. In: Proceedings of 2014 Fourth International Conference on Instrumentation and Measurement, Computer, Communication and Control, Harbin, China (2014)
6. Jinhua, G., Kejian, X.: ARP spoofing detection algorithm using ICMP protocol. In: Proceedings of 2013 International Conference on Computer Communication and Informatics, Coimbatore, India (2013)
7. Salim, H., Li, Z., Tu, H., Guo, Z.: Preventing ARP spoofing attacks through gratuitous decision packet. In: Proceedings of 11th International Symposium on Distributed Computing and Applications to Business, Engineering and Science, Washington DC, USA (2012)
8. LBL Network Research Group, Information and Computing Sciences Division, at Lawrence Berkeley National Laboratory, ARP Watch. http://www.securityfocus.com/tools/142
9. ISL, ARP-Guard. https://www.arp-guard.com/en/arp-guard/product.html
10. Zdrnja, B.: Malicious JavaScript insertion through ARP poisoning attacks. IEEE Secur. Priv. **7**, 72–74 (2009)
11. Carter, C., Yi, S., Kravets, R.: ARP considered harmful: manycast transactions in ad hoc networks. In: Proceedings of 2003 IEEE Wireless Communications and Networking, New Orleans LA, USA (2003)
12. Birmelé, E., et al.: Optimal listing of cycles and st-paths in undirected graphs. In: Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans LA, USA (2013)
13. Shaffer, C.A.: A Practical Introduction to Data Structures and Algorithm Analysis. Virginia Tech, Blacksburg (2010)
14. Sadhir, G., Hu, Y., Perrig, A.: ARP attacks in wireless ad hoc networks (2003). http://dl.icdst.org/pdfs/files/0d65ca5916c99a18d087bad19f6d1d0d.pdf
15. Bruschi, D., Ornaghi, A., Rosti, E.: S-ARP: a secure address resolution protocol. In: Proceedings of the 19th Annual Computer Security Applications Conference (2003)