Active Detection and Prevention of Sophisticated ARP-Poisoning Man-in-the-Middle Attacks on Switched Ethernet LANs

K. Kalajdzic[1,2] and A. Patel[2,3]

[1] Center for Computing Education - CCED, Alipasina 2/1, 71000 Sarajevo, Bosnia and Herzegovina
[2] Department of Computer Science, Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia, 43600 UKM Bangi, Selangor, Malaysia
[3] Faculty of Computing Information Systems and Mathematics, Kingston University, Penrhyn Road, Kingston upon Thames KT1 2EE, United Kingdom
e-mail: kenan@unix.ba, whinchat2010@gmail.com

Abstract

In this paper we describe two novel methods for active detection and prevention of ARP-poisoning-based Man-in-the-Middle (MitM) attacks on switched Ethernet LANs. As a stateless and inherently insecure protocol, ARP has been used as a relatively simple means to launch Denial-of-Service (DoS) and MitM attacks on local networks and multiple solutions have been proposed to detect and prevent these types of attacks. MitM attacks are particularly dangerous, because they allow an attacker to monitor network traffic and break the integrity of data being sent over the network. We introduce backward compatible techniques to prevent ARP poisoning and deal with sophisticated stealth MitM programs.

Keywords

ARP, ARP Poisoning, Man-in-the-Middle Attacks, Intrusion Prevention, LAN

1. Introduction

Address Resolution Protocol (ARP) (Plummer, 1982) is an essential component of communication in an Ethernet LAN environment. It provides a mechanism to translate logical network addresses into physical (MAC) addresses which are required for the exchange of packets on a local network.

ARP is a stateless protocol designed without security in mind, which makes it an ideal means for launching DoS and MitM attacks on a LAN. By sending spoofed MAC addresses in ARP reply packets, a malicious host can poison the ARP cache of other hosts on the local network and thereby easily redirect network traffic.

To mitigate the danger of ARP-based attacks on local networks, multiple techniques have been proposed to detect and prevent attacks by malicious hosts. Detection of ARP poisoning is usually performed by specialized network tools, such as `arpwatch` (LBNL Network Research Group, n.d.), or Intrusion Detection Systems.

In (Carnut & Gondim, 2003) and (Trabelsi & Shuaib, 2007) the authors propose delegating the detection to specialized detection or test stations.

For prevention of ARP-based attacks, a simple solution consists of using static ARP entries in the ARP cache. This solution, however, doesn't scale well especially in heterogeneous networks with dynamic IP addressing. Other solutions include use of cryptography for authenticating ARP replies (Bruschi *et al.*, 2003), (Goyal & Tripathy, 2005), (Lootah *et al.*, 2007), artificial intelligence (Trabelsi & El-Hajj, 2007), or hardware support for dynamic ARP inspection (Cisco Systems, 2009).

We have developed two methods for detection and prevention of ARP-poisoning-based MitM attacks. For simplicity and convenience, we call these Method 1 and Method 2, respectively. Our motivation was to find ways to cope with increasingly sophisticated MitM attack tools, while still maintaining backward compatibility with existing ARP implementations. We avoided the use of specialized computers as helpers in the attack detection process, in contrast with several of the aforementioned methods which require the use of such computers (e.g. a test station or a CA server).

Method 1, described in Section 2, uses certain techniques proposed in (Trabelsi & Shuaib, 2007), but brings several improvements in the approach to detection. Instead of relying on a test host to detect potential attacks, each host performs detection by itself. This eliminates the need for a test host, which is a single point of failure, and makes it possible to extend Method 1 to perform distributed and coordinated detection with multiple hosts. Moreover, with Method 1 detection is triggered by a reception of one or more ARP replies and targets only the hosts who send these replies, instead of scanning the whole network in the search of potential attackers.

Method 2, introduced in Section 3, addresses limitations of Method 1 in dealing with sophisticated MitM attack tools. It relies on a novel technique for detection of MitM attacks on switched Ethernet LANs through modification of the switch CAM table in a way which makes the detection transparent to the MitM host.

2. Method 1 – Reverse ARP poisoning with active IP probing

Method 1 consists of the following two steps:

1. **Reverse ARP poisoning** – A host implementing reverse ARP poisoning sends an ARP reply as a response to every ARP reply it receives from other hosts. The purpose of this reverse ARP reply is to poison the ARP cache of attacking hosts.
2. **Active IP probing** – Active IP probing is then used to differentiate between legitimate hosts and MitM hosts. This step consists of sending a single IP packet to the host from which the initial ARP reply was received and analyzing the response. For simplicity, in this document we use probe packets containing simple ICMP echo requests, even though it may generally be more reliable to use TCP or UDP instead of ICMP.

The best way to illustrate the workings of Method 1 is to see it in action. For this purpose, we use two common scenarios.

In the first scenario, we analyze the packet exchange in the case of a legitimate host sending an ARP reply. The second scenario will then show how Method 1behaves when a MitM host attempts to carry out an ARP poisoning attack.

Figure 1 is used as a reference for both scenarios. We assume that all three hosts, `HostA`, `HostB` and `HostX`, are on the same Ethernet LAN. Furthermore, `HostA` and `HostB` are legitimate hosts and `HostX` is a MitM attacker. Also, `HostA` uses a regular implementation of ARP, as found in modern operating systems. `HostB`, on the other hand, implements Method 1, and thus handles ARP traffic in a different way, as will be described shortly.

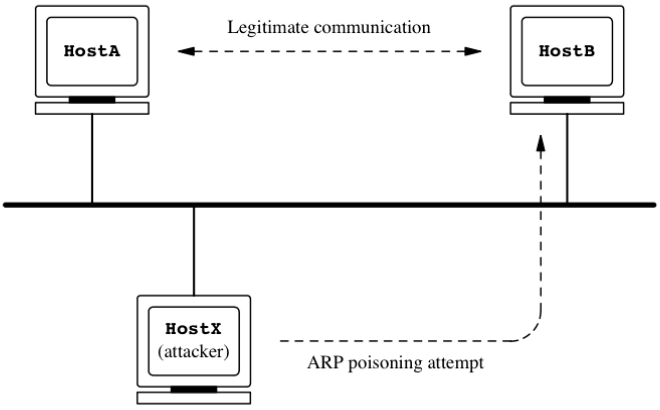

Figure 1: An ARP poisoning attack on a switched LAN

2.1. Scenario 1 – Legitimate ARP reply

In this scenario, `HostA` sends a legitimate ARP reply to `HostB`. We can follow the exchange of packets generated as Method 1is employed:

1. `HostA` sends an ARP reply packet to `HostB`. Since this is a legitimate ARP reply, it contains the mapping between `HostA_IP` and `HostA_MAC`.
2. `HostB` executes the first step of Method 1, and immediately sends an ARP reply back to `HostA` attempting to poison its ARP cache. In this ARP reply `HostB` maps `HostA_IP` to `HostB_MAC`. Since, however, `HostA` is the owner of `HostA_IP`, it simply drops this ARP reply with the invalid mapping.
3. `HostB` then continues to the second step of Method 1 and sends an ICMP echo request packet addressed to `HostA_IP` with `HostA_MAC` as the destination MAC address in the Ethernet frame header.

4. `HostA` receives the ICMP echo request and responds to `HostB` with an ICMP echo reply. For `HostB` this is an indicator that the reverse ARP poisoning attempt was unsuccessful and that the ARP reply sent by `HostA` is a legitimate one.
5. As a result, `HostB` stores the mapping `HostA_IP` ↔ `HostA_MAC` in its ARP cache.

2.2. Scenario 2 – ARP poisoning attempt

In this scenario the attacking host `HostX` attempts to poison the ARP cache of `HostB` in order to impersonate `HostA`. This should allow the attacker to hijack all traffic going from `HostB` to `HostA`. Since `HostB` implements Method 1, the exchange of packets in this case will be as follows:

1. The first packet is an ARP reply sent from `HostX` to `HostB`. This ARP reply contains the mapping between `HostA_IP` and `HostX_MAC`. If `HostB` had a regular implementation of ARP, it would accept this ARP reply and store the incorrect mapping in its ARP cache. From that point on, `HostB` would deliver all network traffic destined to `HostA_IP` to `HostX`'s network interface.
2. Nevertheless, `HostB` handles ARP traffic in compliance with Method 1, so instead of blindly accepting the ARP reply from `HostX`, `HostB` begins the detection procedure by sending a reverse ARP reply to `HostX`. This ARP reply contains the mapping between `HostA_IP` and `HostB_MAC`. Assuming that the attacking host (i.e. `HostX`) uses an unmodified implementation of ARP, the reply sent by `HostB` will poison its ARP cache.
3. `HostB` proceeds with the MitM detection by delivering an ICMP echo request packet, destined to `HostA_IP`, to `HostX`'s network interface (by using `HostX_MAC` as the destination in the Ethernet frame header).
4. `HostX` is acting as a MitM attacker, so it attempts to forward this ICMP echo request packet to `HostA`. However, since `HostX`'s ARP cache has previously been poisoned by `HostB`, `HostX` delivers the probe packet to `HostB`'s MAC address. This effectively means that the same packet sent in the previous step by `HostB` will be returned to it by `HostX`. The detection of a duplicate packet is a clear indicator for `HostB` that reverse ARP poisoning was successful and that `HostX` is a MitM attacker.
5. `HostB` thus drops the initial ARP reply sent by `HostX`. Since at this point an intrusion attempt has been detected, `HostB` can generate a real-time intrusion alert and log the intrusion attempt for the purpose of a future forensic investigation.

3. Method 2 – IP probing with CAM table poisoning

Method 1, described in Section 2, works well for detection of MitM computer systems which rely on the operating system built-in routing and ARP functions. There are, however, much more sophisticated MitM programs, which take full control over packet forwarding. This allows these programs to disguise themselves very well in order to evade detection. One popular program which falls into this category is the well-known *Cain & Abel* (Montoro, n.d.).

Cain & Abel doesn't rely on the ARP and routing functions of the operating system, but instead maintains its own mappings between IP addresses and MAC addresses. The program utilizes these private mappings when forwarding frames between hosts on the network. This makes it insusceptible to reverse ARP poisoning, which is the basis of Method 1.

In order to be able to detect any MitM host, regardless of the way it handles routing of packets between other hosts in the network, we need to influence flow of packets in a way which is beyond control of the MitM host.

In the following paragraphs, we describe one method to achieve this, which we call Method 2 for brevity and simplicity. Figure 2 will serve as a reference for our description of Method 2.

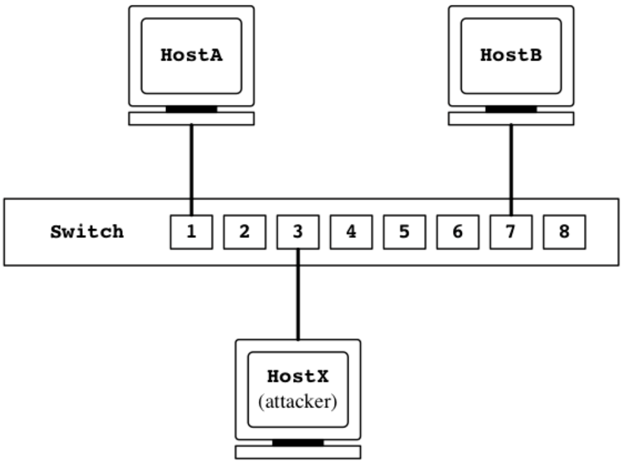

Figure 2: Physical connection of hosts in our LAN

During normal operation of the switch, its CAM (in order to minimize switching latency, Ethernet switches store the mappings between MAC addresses and switch ports in a table inside a special Content-Addressable Memory) table contains the mappings shown in Table 1.

MAC Address	Port
HostA_MAC	1
HostB_MAC	7
HostX_MAC	3

Table 1: Switch CAM table during normal operation

We again assume that HostX wants to redirect traffic between HostA and HostB through the use of ARP poisoning. HostA uses a regular implementation of TCP/IP, including ARP, and HostB employs Method 2. We can now follow the use of Method 2 through the following flow of events:

1. HostX sends an ARP reply to HostB. This ARP reply contains the mapping between HostX_MAC and HostA_IP.
2. Before entering this mapping into its ARP cache, HostB begins executing Method 2, whose first step is broadcasting of an ARP request for HostA_IP.
3. As a result of this ARP request, HostB receives two replies with two different MAC address mappings for HostA_IP: one reply comes from HostA with HostA_MAC and the other is from the attacker with HostX_MAC. Method 2 doesn't require these two replies to reach HostB in any particular order.
4. The reception of two different MAC addresses for a single IP address is a first indicator for HostB that one of them comes from a MitM attacker. Thus, HostB continues with the next step of Method 2, which is sending multiple ICMP echo request packets out its network interface. All these packets carry HostB_IP as the source IP address and HostA_IP as the destination IP address in their IP header. However, their Ethernet frame header may contain one of the following two combinations of MAC addresses:
   (a) HostX_MAC is the destination MAC address and HostA_MAC is the source MAC address,
   (b) HostA_MAC is the destination MAC address and HostX_MAC is the source MAC address.
5. To understand the purpose of using these two MAC address combinations, let us analyze what happens when HostB sends two ICMP echo request packets addressed as in 4a and 4b, respectively:
   (a) The frame, addressed as specified in 4a leaves HostB and enters the switch through port #7. Based on the entries in its CAM table (see Table 1), the switch forwards the frame to HostX through port #3. Meanwhile, since the frame with source MAC address HostA_MAC entered the switch through port #7, the switch updates its CAM table with a new mapping for HostA_MAC so the CAM table now has the contents shown in Table 2. HostX receives the frame, looks up the destination IP address,

and forwards the frame immediately towards `HostA`, specifying `HostA_MAC` as the destination MAC address. Once this frame reaches the switch, two possibilities exist:

i. If the switch CAM table still contains the mapping between `HostA_MAC` and port #7, the switch will forward the frame out through port #7. `HostB` receives its own ICMP echo request packet, which is an indicator that `HostX` attempted to forward this frame to `HostA`. This means that `HostX` is not the real owner of `HostA_IP`, but a MitM attacker.

ii. If, in the meantime, `HostA` sent some network traffic through switch port #1, the original mapping of `HostA_MAC` to port #1 in the CAM table of the switch will have been restored. In this case, the switch forwards the ICMP echo request through port #1 to `HostA`, and `HostA` responds by sending an ICMP echo reply packet back to `HostB`. In this case `HostB` cannot conclude with certainty that `HostX` forwarded the frame to `HostA`.

(b) The frame, addressed as specified in 4b enters the switch through port #7, and switch forwards it through port #1 to `HostA`. Since the source MAC address of this frame is `HostX_MAC`, the switch maps `HostX_MAC` to port #7 in its CAM table. Table 3 shows the new mapping. When `HostA` receives the ICMP echo request packet, it builds a response in form of an ICMP echo reply packet with source IP address `HostA_IP` and destination IP address `HostB_IP`.

i. Assuming that `HostA`'s ARP cache has been previously poisoned by `HostX`, the response packet will be sent in a frame addressed to `HostX_MAC`. If the contents of the CAM table haven't been modified in the meantime (i.e. they are still as shown in Table 3), the switch will deliver this frame through port #7 to `HostB`. If, on the other hand, `HostX` generated some network traffic while `HostA` was preparing the response, the CAM table will have returned to its original state (see Table 1). Thus, the switch will send the response packet from `HostA` to `HostX` through port #3. Because `HostX` is a MitM host, it will forward the response to `HostB`.

ii. If the ARP cache of `HostA` hasn't been modified, it will contain a correct mapping between `HostB_IP` and `HostB_MAC`. Therefore, the ICMP reply packet from `HostA` will be sent to `HostB_MAC` and delivered by the switch through port #7 to `HostB`.

We see that, in either case, using the MAC address combination given in 4b results in an ICMP echo reply packet being sent to `HostB`. In other words, it can not happen that in the given scenario an ICMP echo request packet with source MAC address `HostX_MAC` and destination MAC address `HostA_MAC` gets delivered back to `HostB`.

MAC Address	Port
HostA_MAC	7
HostB_MAC	7
HostX_MAC	3

Table 2: Switch CAM table after HostB sends a frame from HostA MAC to HostX MAC through port #7

MAC Address	Port
HostA_MAC	1
HostB_MAC	7
HostX_MAC	7

Table 3: Switch CAM table after HostB sends a frame from HostX MAC to HostA MAC through port #7

When, on the other hand, the combination of source and destination MAC addresses is specified as in 4a, it is possible for the original ICMP echo request packet to be delivered back to `HostB` (see 5(a)i), though it may also happen that `HostB` receives an ICMP echo reply from `HostA` (see 5(a)ii). The latter case cannot generally be distinguished from the case described in 5b, which uses frames addressed as in 4b.

Therefore, we must ensure that a host implementing Method 2 (in our case, `HostB`) quickly sends multiple ICMP echo request packets with both combinations of source and destination MAC addresses given in 4a and 4b. To identify the MitM host it suffices for `HostB` to receive only one of its own ICMP echo request packets back through its network interface.

Method 2 alters the CAM table of the switch so that some frames destined to HostA are delivered to `HostB` (see Table 2). To restore the original mapping of `HostA_MAC` to port #1 (see Table 1), `HostB` may broadcast an ARP request for `HostA_IP`. This would force `HostA` to send back an ARP reply and thereby help switch *reassociate* its MAC address with port #1.

Notice that, for Method 2 to work, `HostB`'s network card must be put into *promiscuous mode* (when a network card operates in promiscuous mode, it accepts all traffic and passes it to the central processing unit, even if this traffic is not addressed to that particular network card), so it can collect the hijacked frame which `HostX` attempts to forward to `HostA`. Another important assumption is that `HostA` was not subject to a DoS attack, so it was able to respond to our ARP requests.

4. Results

We ran multiple tests on a switched Ethernet LAN to test the effectiveness of Method 1 and Method 2 in detecting ARP-poisoning-based MitM attacks. In all these tests our setup was as depicted by Figure 2. `HostA` and `HostB` were running *Windows XP* and *Linux* respectively, and the operating system of `HostX` changed as required by the tests. Using several common tools, we performed MitM attacks from `HostX`, attempting to poison the ARP cache of `HostA` and `HostB`. The role of `HostB` was to detect these attack attempts by employing Method 1 and Method 2.

4.1. Detecting `Ettercap` and `dsniff` with Method 1

In the first test `HostX` (running *Backtrack Linux*) performed attacks against ARP cache of `HostA` and `HostB` using two mainstream attack tools, `Ettercap` (Ornaghi & Valleri, n.d.) and `arpspoof` with `dsniff` (Song, n.d.).

`HostB` was set up to perform attack detection with Method 1. Since `Ettercap` and `dsniff` rely on the operating system built-in ARP and routing functions, we were able to successfully perform reverse ARP poisoning and detect all the attacks through active IP probing (i.e. Method 1) with 100% accuracy.

4.2. Detecting *Cain & Abel* with Method 1

For the purpose of this test we booted `HostX` into *Windows XP* and launched multiple MitM attacks against `HostA` and `HostB` using *Cain & Abel*. This time, however, `HostB` failed to detect any of our attacks. Knowing that *Cain & Abel* uses its own IP-to-MAC address mappings when forwarding packets, this was expected.

4.3. Detecting *Cain & Abel* with Method 2

As we know from Section 3, when using Method 2 `HostB` poisons the CAM table of the switch in order to capture the frame which `HostX` attempts to forward towards `HostA`. This is not a big problem when `HostA` is idle. If, however, `HostA` is actively communicating, this creates a race between `HostA` and `HostB`. Depending on the rate at which `HostA` sends out packets into the network, it may be more or less difficult for `HostB` to win the race and hijack the packet required for detection of the MitM attack.

To test the effectiveness of Method 2, we set up `HostA` to send many thousands of packets per second into the network and measured the attack detection ratio, whereby

$$\textit{Detection ratio} = \frac{\textit{Number of successful detections}}{\textit{Total number of probes sent}}$$

During these tests, `HostB` was sending either single probe packets or series of 3, 5 or 7 packets per probe. The results of our measurements are summarized in Figure 3.

We notice that the success of detection depends on the number of packets sent in a single probe. The rather low detection ratio of 30% for single-packet probes was doubled by sending three packets in each probe. Further increases in number of packets per probe to five and seven raised the detection ratio to 80% and 90% respectively.

It is also obvious that the detection ratio doesn't depend on the rate at which `HostA` sends packets into the network. If we neglect the variations in the value of the detection ratio, which exist due to a stochastic nature of real-time network communication, we can consider all four curves in Figure 3 as constants.

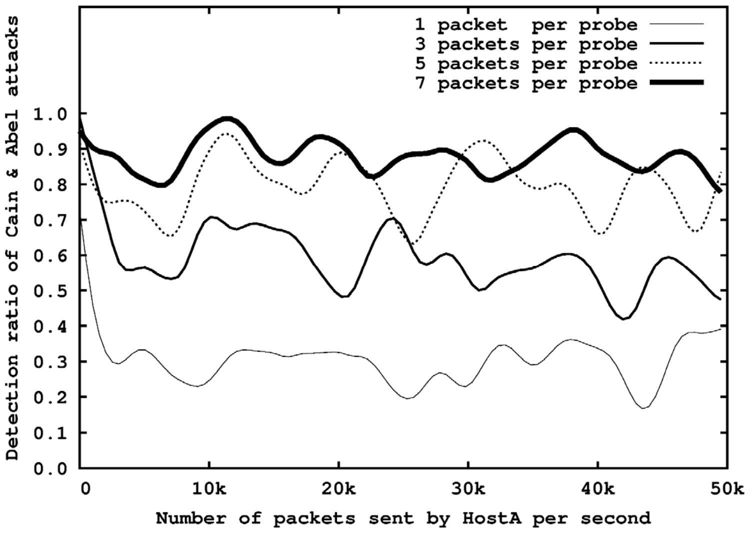

Figure 3: Success in detection of *Cain & Abel* with Method 2

5. Limitations of Method 1 and 2

Both the theoretical discussion and results of our experiments have revealed certain limitations of both proposed methods.

As we know from Sections 2 and 4.2, the biggest limitation of Method 1 is its inability to handle detection of MitM attack tools which use their own IP-to-MAC address mappings for forwarding packets (e.g. *Cain & Abel*). Even though Method 2 solved this problem, other factors exist which may limit its effectiveness.

In the third step of the detection process with Method 2, we assumed that `HostB` receives ARP replies for `HostA_IP` from both `HostA` and `HostX`. While this is generally the case, `HostX` might as well launch a DoS attack against `HostA`, preventing it from successfully delivering its ARP reply to `HostB`. This way only `HostX`'s ARP reply would reach `HostB`, rendering Method 2 useless.

The results of our experiments in Section 4.3 have shown that the effectiveness of Method 2 depends on the number of packets sent in a single probe. Sending too many probe packets, however, may cause disruption in traffic flow towards `HostA`, due to the fact that `HostB` temporarily hijacks all LAN traffic destined to `HostA_MAC`. This problem may be solved by storing the hijacked packets in a queue on `HostB` and delivering them back to `HostA` after the probe.

6. Conclusion

In this paper we have described two novel methods for detection and prevention of ARP-based MitM attacks on switched Ethernet LANs. Both methods work as extensions to the ARP protocol and don't interfere with normal ARP operation. Therefore, both these methods can co-exist on the same LAN with regular ARP implementations and are thus suitable for incremental deployment. We have seen examples of such co-existence in experiments in which one host (`HostB`) used either Method 1 or Method 2, while another host (`HostA`) used default implementation of ARP as provided by the operating system.

Even though both our methods can be used to identify and prevent ARP poisoning attacks, an ultimate solution to the problem of ARP insecurity can only be provided through an improved version of the ARP protocol, which would be backwards compatible and would allow for an incremental implementation. In (Abad and Bonilla, 2007) the authors have given a definition of an ideal solution for prevention of ARP-based attacks, which may be the first step towards reaching this goal.

7. References

Abad, C. & Bonilla, R. (2007), "An analysis on the schemes for detecting and preventing ARP cache poisoning attacks", in *Distributed Computing Systems Workshops, 2007. ICDCSW'07. 27th International Conference on Distributed Computing Systems Workshops*, IEEE, p. 60.

Bruschi, D., Ornaghi, A. & Rosti, E. (2003), "S-ARP: a secure address resolution protocol", in *Proceedings of the 19th Annual Computer Security Applications Conference, 2003. ACSAC'03*.

Carnut, M. & Gondim, J. (2003), "ARP spoofing detection on switched Ethernet networks: A feasibility study", in *Proceedings of the 5th Symposium on Security in Informatics*.

Cisco Systems (2009), "Configuring Dynamic ARP Inspection", in *Catalyst 6500 Series Switch Cisco IOS Software Configuration Guide, Release 12.2(18) SXF and Rebuilds and Earlier Releases*, San Jose, CA, USA: Cisco Systems, 38:1–38:22.

Goyal, V. & Tripathy, R. (2005), "An efficient solution to the ARP cache poisoning problem", in *Information Security and Privacy*, Springer: pp. 40–51.

LBNL Network Research Group. `arpwatch`: the Ethernet monitor program; for keeping track of Ethernet/IP address pairings (online) available at ftp://ftp.ee.lbl.gov/ arpwatch.tar.gz.

Lootah, W., Enck, W. & McDaniel, P. (2007), "TARP: Ticket-based address resolution protocol", *Computer Networks 51(15)*, 4322–4337.

Montoro, M. Cain & Abel (online) available at http://www.oxid.it/cain.html.

Ornaghi, A. & Valleri, M. Ettercap (online) available at http://ettercap. sourceforge.net/.

Plummer, D. (1982), "RFC-826: An Ethernet Address Resolution Protocol", Network Working Group.

Song, D. `dsniff` (online) available at http://monkey.org/~dugsong/dsniff/.

Trabelsi, Z. & El-Hajj, W. (2007), "Preventing ARP Attacks Using a Fuzzy-Based Stateful ARP Cache", in *Communications, 2007. ICC'07. IEEE International Conference on Communications 2007*, IEEE: pp. 1355–1360.

Trabelsi, Z. & Shuaib, K. (2007), "NIS04-4: Man in the Middle Intrusion Detection", in *Global Telecommunications Conference, 2006. GLOBECOM'06.* IEEE: pp. 1–6.