Using a Secure Permutational Covert Channel to Detect Local and Wide Area Interposition Attacks

Jaroslaw Paduch
Department of Math.& Computer Science
John Jay College, CUNY
New York, 10016
jpaduch@jjay.cuny.edu

Jamie Levy
Department of Math.& Computer Science
John Jay College, CUNY
New York, 10016
jlevy@jjay.cuny.edu

Bilal Khan
Department of Math.& Computer Science
John Jay College, CUNY
New York, 10016
bkhan@jjay.cuny.edu

ABSTRACT

In this paper, we present new techniques to detect interposition attacks on stream-based connections in local and wide area networks. The approach developed here is general enough to apply *uniformly to all circumstances where the man-in-the-middle attacker achieves interposition by corrupting higher-layer to low-layer address mappings.* Thus, both the problem of local area network interposition through ARP poisoning, and the problem wide area interposition through DNS poisoning are addressed as special cases of our work. Like other solutions that reside between Layers 3 and 4 (e.g. IPSEC), our techniques enjoy the property that they *do not* require redesigning legacy software, as is the case for approaches that reside above Layer 4 (e.g. SSL/TLS). Unlike IPSEC, however, the developed system is tailored only to the *detection* of interposition attacks, and thus circumvents the overhead and complexity introduced in guaranteeing stream confidentiality and integrity. We describe the design of the system, demonstrate its efficacy, and provide a publicly accessible prototype implementation.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Protocols–*security*

General Terms

Security, Protocols

Keywords

interposition, DNS, ARP, covert channels

1. BACKGROUND

This paper addresses the problem of interposition attacks in which interposition relies on corrupting higher-layer to low-layer address mappings. We begin by describing the two most important concrete instances of this phenomenon.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
IWCMC '09 , June 21-24, 2009, Leipzig, GERMANY
Copyright 2009 ACM 978-1-60558-569-7/09/06 ...$5.00.

1.1 ARP Poisoning

Address Resolution Protocol (ARP) enables dynamic definition of the map from IP address to the link layer (MAC) address for network devices. ARP cannot be static since machines may need to change their IP addresses as needed; unfortunately, this aspect of the protocol is readily exploitable. When sending packets on the LAN, the destination MAC is resolved from the destination IP using ARP. The sender issues an ARP *Request* providing the IP, and the intended destination replies with it's MAC, which is then cached at the sender for some finite time. A "man-in-the-middle" attack based on ARP requires (1) periodically flooding the network with requests/replies for the victim's IP and the attackers's MAC address in place of the victim's and send them out on the network every few seconds, and (2) enabling packet forwarding on the attacker's machine in order to avoid a "sink hole" and allow the victim to continue new and existing conversations [1, 2]. There are many tools for packet injection [3, 4] so instrumenting this attack is straightforward. Though *ARP poisoning* is considered "old school", it still dangerous today [5]. Such attacks are increasingly prevalent and highly publicized in the media [6, 7].

1.2 Defenses Against ARP Poisoning

Many solutions are proposed in order to detect such attacks. The most direct approach is to have static ARP tables [8], and this is pursued by programs like anti-arp [9]. However opinions differ as to the merits of this strategy [10] particularly with respect to configuration overhead [8]. Another approach, taken by systems like Arpwatch [1, 11], is to monitor for changes in ARP replies, and email notifications of observed changes. However, such email may be intercepted, missed, or the recipient may be in a position where the damage of ARP Poisoning can be quickly mitigated. DHCP snooping is another solution vector and is frequently implemented as a hardware solution. In this approach, the network device keeps track of the MAC address/port bindings of clients when they request an IP address. All attempts to send false ARP requests are then blocked to prevent clients from becoming poisoned [12]. This approach has its weakness as well, since actual MAC addresses themselves can be changed or spoofed to a valid MAC address. ArpOn is another software solution that mirrors like the aforementioned architecture. ArpOn manages all ARP protocol options, and replaces ARP utilities. Since it keeps its own cache of MAC addresses, ArpOn can ignore conflicting MAC addresses [13].

1.3 DNS Poisoning

DNS poisoning is another example of a serious threat that is based on corrupting higher-layer to low-layer address mappings, and is a high profile threat frequently receiving public attention [14, 15, 16, 17, 18, 19]. One way that DNS poisoning occurs is when the attacker requests the information for a particular domain. If the server does not contain the information in its cache, it will need to look up the information at a higher-level DNS server. However, because there is no restriction on where the answer comes from, an attacker can send false information to another domain. The only technical obstacle is that the attacker must use the same transaction ID (XID) when sending the information as was provided with the request. Although they may not know what that XID is, there is no restriction on the number of attempts. With the latest DNS flaw the attacker's chances of success increase greatly [17, 18, 19, 20]; several tools are available to make DNS poisoning easier [3, 21, 17]. After the last major patching incident for DNS, the protocol was modified to include a random source port [18, 20, 22] together with the XID, so the attacker has a much harder (though still not intractible) task of guessing the XID when sending fake information to the DNS server [18, 20].

1.4 Defenses against DNS Poisoning

As in the case of ARP poisoning, the most direct approach against DNS poisoning is to use static IP addresses for high risk domains in the hosts file of the users computer. However, this ARPANET-throwback is hardly a credible solution for the dynamic modern Internet. Secure Sockets Layer (SSL) and domain certificates are often implemented as a defense against DNS Poisoning. In this approach, when the user connects to an interposed site, they see a warning that the site certificate does not match. In practice two problems arise [18, 20]: (1) most users either do not notice these warnings or just skip through them, and (2) since certificate authorities validate by sending an email, if the user's mail server has been hijacked, the attacker can generate valid certificates for their malicious site. Another approach is attempted by OpenDNS [23], which purposely blocks harmful domains. However, since domains can pop up suddenly, there is a small interval during which a domain is not known as harmful and is not yet blocked by OpenDNS. Also, as we have seen with the latest Kaminsky DNS flaw, OpenDNS will not protect against exploitable flaws in the DNS protocol itself, since OpenDNS is also vulnerable to such flaws until it is patched.

2. SYSTEM DESIGN

We deduce our approach *ab initio*. Interposition attacks frequently rely on deliberate corruption of higher-layer to low-layer address mappings. Concrete examples include poisoning ARP tables to enable local area interposition, and wide area interposition attacks based on DNS poisoning. To detect such man-in-the-middle attacks it suffices for each endpoint to be able to verify that it's beliefs regarding the lower-layer address of the remote peer are valid. The ultimate authority capable of verifying such beliefs is the remote peer itself. However, two issues arise in the design and implementation of such an *Remote Address Verification Protocol* (RAVP):

1. How can each endpoint communicate its beliefs to the remote peer "in band", i.e. without introducing a new higher-layer protocol that would in turn necessitate software modification, and
2. How can one prevent the attacker from tampering with address verification protocol's messages?

The first concern is driven by the desire to develop a comprehensive solution which will retroactively provide interposition detection for legacy TCP/IP applications without requiring any code modifications (as is usually required when one introduces protocols such as SSL/TLS above OSI Layer 4). We address (1) by first implementing a permutational covert channel superimposed on the IP packet stream, and using this covert channel to carry all traffic for the Remote Address Verification Protocol. The second concern is addressed by making sure that all information sent over the covert channel is digitally signed by the sender. This makes it effectively impossible for the intermediary to corrupt the messages sent over the covert channel, and thereby sabotage the operation of the Remote Address Verification Protocol.

Given this deduction ab initio, we summarize our approach concretely as follows: Each party in the connection repeated sends its (digitally signed) beliefs regarding the lower-layer address of the remote endpoint over a permutational covert channel that is superimposed over the overt TCP/IP stream. Upon receiving this covert communication, the receiving endpoint verifies the sender's signature, and then compares its own local (authoritative) lower-layer address what the remote party has sent over the covert channel. Under normal auspicious circumstances, the two lower-layer addresses will agree; but if they remote system's beliefs are determined to disagree with the local authoritative lower-layer address, the system responds according to a system-wide policy (e.g. the event is logged, the connection is closed, or the local human user is alerted to the fact that there is an interposition attack in progress and that the data being received may have been altered by an intermediary).

Figure 1 shows how the system operates normally in the local area. `192.168.1.101` believes `00:B0:D0:86:BB:F2` is the lower-layer MAC address of remote peer `192.168.1.102`, and so communicates this, digitally signed, to the remote host `192.168.1.102` via the covert channel. Likewise, machine `192.168.1.102` believes the lower-layer MAC address of the remote peer `192.168.1.101` to be `00:B0:D0:86:BB:F1`, and so communicates this, digitally signed, to the remote host `192.168.1.101` via the covert channel. The IP stack at each host receives the covert communication, verifies the signature using the sender's public key (retrieved from the Certificate Authority based on the higher-layer IP address of the remote peer). It then verifies that the remote peer's beliefs coincide with the local authoritative MAC address.

In the event of a local area interposition attack due to ARP poisoning, the situation is as shown in Figure 2. Both `192.168.1.101` and `192.168.1.102` believe the MAC of their remote peer is `00:0B:AD:BA:DB:AD` and communicate this, digitally signed, to each other. The IP stack at each host receives the covert communication, verifies the signature using the sender's public key (retrieved from the Certificate Authority based on the higher-layer IP address of the remote peer), and then detects that the remote peer's beliefs *do not* coincide with the local authoritative MAC address. Even if the interposing attacker is aware of the covert channel and the details of its implementation, they are unable to reli-

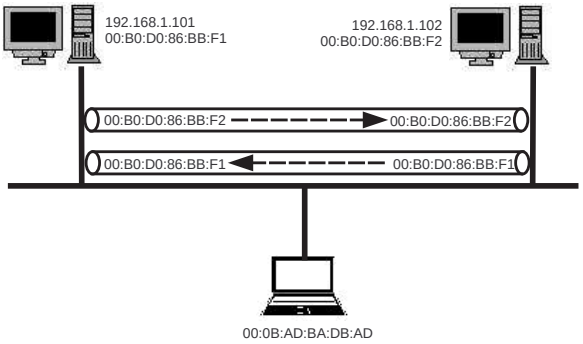

Figure 1: Normal LAN operation.

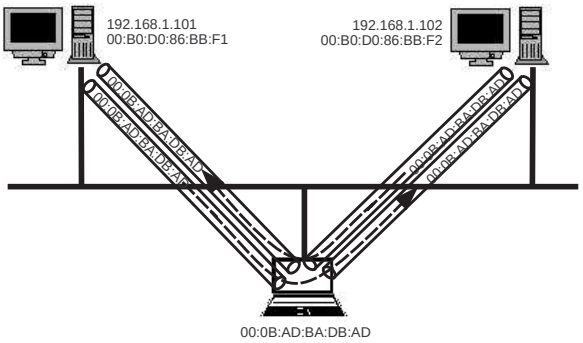

Figure 2: Interposition in the LAN.

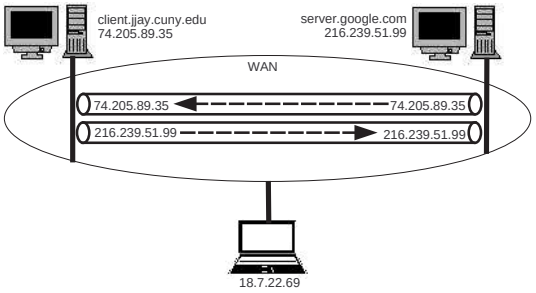

Figure 3: Normal WAN operation.

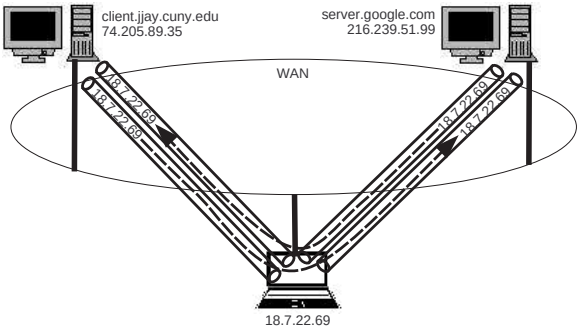

Figure 4: Interposition in the WAN.

ably alter the MAC address that is conveyed over the covert channel, because these messages are signed by the sender; integrity violations would be detected by the recipient and count towards the evidence of an interposition.

Figure 3 shows how the system operates normally in the wide area. Machine `client.jjay.cuny.edu` believes the IP address of the remote peer `server.google.com` to be `216.239.51.99`, and so communicates this to the remote host `server.google.com` via the covert channel. Likewise, machine `server.google.com` believes the IP address of the remote peer `client.jjay.cuny.edu` to be `74.205.89.35`, and so communicates this to `client.jjay.cuny.edu` via the covert channel. The IP stack at each host receives the covert communication, verifies the signature using the sender's public key (retrieved from the Certificate Authority based on the domain name of the remote peer). It then verifies that the remote peer's beliefs coincide with the local authoritative IP address.

In the event of a wide area interposition attack due to DNS poisoning, the situation is as shown in Figure 4. Both machines `client.jjay.cuny.edu` and `server.google.com` believe the IP address of the remote peer is `18.22.7.59` and communicate this, digitally signed, to each other. The IP stack at each host receives the covert communication, verifies the signature using the sender's public key (retrieved from the Certificate Authority based on the domain name of the remote peer), and then detects that the remote peer's beliefs *do not* coincide with the local authoritative IP address. Even if the interposing attacker is aware of the covert channel and the details of its implementation, they are unable to reliably alter the IP address that is conveyed over the covert channel, because these messages are signed by the sender; integrity violations would be detected by the recipient and count towards the evidence of an interposition.

3. CHANNEL DESIGN

In this section we describe the covert channel that we used in our system. The operation of the channel is based on the simple observation that TCP provides a reliable in-order transport layer data stream over the unreliable network layer IP datagram service. Moreover, since TCP is designed to be robust to out of order network layer packet delivery, we can artificially permute IP packets before they leave the source, and read the permutation at the destination. The choice of permutation is used to encode a covert value, and the sequence of chosen permutations, in turn, encodes the covert message. Applying such permutations does not adversely affect TCP's ability to reconstruct the higher layer transport layer data stream. Suppose we want to transmit a covert message C consisting of a sequence of n bits

$$C = C_0 \cdot C_1 \cdots C_i \cdots C_{n-1} \cdot C_n,$$

to machine A. We begin by determining bit C_i of the covert message to A. For each covert bit, we allow two IP packets to destination A to go through. If $C_i = 0$, we ensure that the two packets are in ascending order by IP ID; if $C_i = 1$, we ensure that the two packets are in descending order by IP ID. Thus, we transmit C by taking the next $2n$ IP packets submitted by TCP for A:

$$P_0, P_1, \ldots, P_{2i}, P_{2i+1}, \ldots, P_{2n-2}, P_{2n-1}.$$

The decoding of the covert message can be deduced by considering the sequence of IP IDs and computing the first derivative of successive pairs: a negative first derivative signifies a 1 while a positive first derivative signifies a 0. For example, let us see how letter 'J' (hex `0x4a`, binary `0100 1010`) can be encoded in a suitable permutation of 16 IP packets. Sixteen queued packets

$$P_0 P_1 \cdots P_i \cdots P_{14} P_{15}$$

are sent out (2 at a time) according to the order:

$$P_0 P_1 P_3 P_2 P_4 P_5 P_6 P_7 P_9 P_8 P_{10} P_{11} P_{13} P_{12} P_{14} P_{15}.$$

The question of whether packets $2i$ and $2i+1$ arrive in ascending or descending IP ID order determines the value of covert bit i. Figure 5 shows the character 'J' being transmitted over a sequence of 16 IP packets.

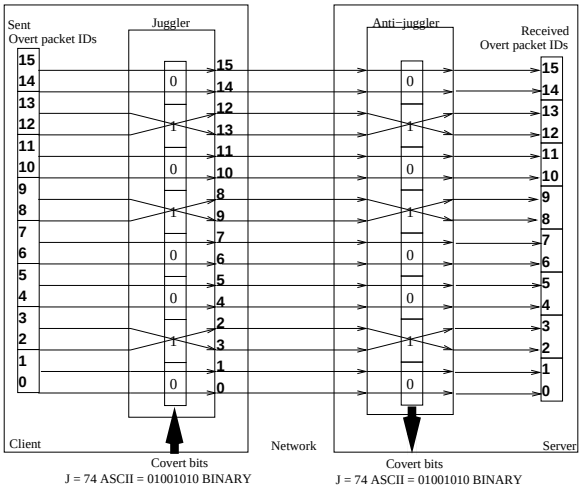

Figure 5: Transmitting 'J' over a sequence of 16 IP packets.

A coarse upper bound on the capacity of such a covert channel can be computed by noting that if the covert channel queues n IP packets and then permutes them before sending them, it can encode $\log_2(n!) = O(n \log n)$ bits of information for every n packets sent in its choice of permutations. In our scheme, however, each bit in the covert message is represented by the placement of *two* IP packets. The covert channel thus exhibits a throughput of 1/2 covert bit per legitimate packet. With IP packets of around 576 bytes, the bit rate of the covert channel is expected to be approximately 1/1152 fraction of the bit rate of the overt channel. In actuality, TCP overhead caused the covert channel to achieve only approximately 1/3039 of the overt channel throughput. We affirmed this analysis in several experiments–for example over an 7.45 Mb/s overt TCP connection, we were able to superimpose a 2.51 Kb/s covert channel having a bit error rate of 7.51%. While this is significantly less efficient compared to the theoretically optimal permutational channel, it provides certain advantages:

- The scheme requires delaying at most one IP packet (when the covert bit that needs to be transmitted is a 1), and does not require delaying any IP packets (when the covert bit that needs to be transmitted is a 0). Amortizing, if the covert message consists of roughly equal 0s and 1s, expected queue depth is 0.5 packets.
- Packet re-ordering produces *localized* corruption in the covert bitstream: If two sequential packets arrive in reversed order (because of network effects), this produces *on average* a one bit error of the covert message received.
- Packet loss produces either *localized* corruption in the covert bitstream, or large scale corruption that is recognizable: If an even number of packets are lost, this causes the corresponding bits of the covert channel to be lost; if an odd number of packets are lost, a framing error exhibits itself as a burst of 0-valued covert bits.

More efficient permutational schemes are certainly possible, though in practice they require queueing more IP packets prior to their transmission, which causes TCP to retransmit the same segment repeatedly, and since RFC 2988 follows Karn's algorithm for taking RTT samples TCP is deprived of an estimate of round trip time (RTT) and interprets the delay as extreme network congestion, backing off the retransmission timer exponentially as per Van Jacobson's algorithm [24]. The net effect is that the throughput (of the legitimate channel) is reduced tremendously, effectively to zero. A more detailed account of real-world tradeoffs in permutational covert channel design was described by the authors [25].

4. RESULTS

PERMEATE-MITM was released as an open source project, and a version of it is publicly available from Sourceforge at `http://permeate.sourceforge.net`. The architecture of PERMEATE-MITM is depicted in Figure 6 below. The covert channel is used to send fixed size (covert) frames separated by an end of frame marker (8 covert zeros followed by 8 covert ones followed by 8 covert zeros). Each frame contains a signed MAC and/or a signed IP address, which is encoded to be free of any occurrence of the end of frame marker. Linux Netfilter forwards all outbound traffic to PERMEATE-MITM's juggler module (via `ip_queue`), which acts a transparent proxy queueing pairs of outbound IP packets and shuffling them based on the covert message to be sent (the signed local belief concerning the remote endpoint's lower layer address). Linux Netfilter also forwards all inbound traffic to PERMEATE-MITM's anti-juggler module (via `ip_queue`). As incoming packets are received, the numeric values of their IP IDs are saved until the end of frame marker is seen on the covert channel. Then, the previous frame's worth of saved IDs is decoded, the signature is verified and decoded to obtain information specifying the remote peer's belief concerning the recipient's lower layer address. This belief is validated against authoritative local information about the recipient's lower layer address. Repeated evidence of disagreement between remote beliefs and local facts results in a diagnosis of interposition and action as per system-wide policies. In our system, "repeated evidence" was taken to be three consecutive corruptions; this threshold criterion is needed to make the diagnosis since the occurrence of (overt) packet reordering/loss results in corruption/loss of covert bits, making it possible for the verification to fail for natural causes–and not because of deliberate interposition. In practice, we found that we were able to detect interposition attacks (executed on the LAN using Ettercap) when three framed signed MAC/IPs had

been sent covertly in both directions between the endpoints. On average, this requires the transmission of approximately 1.12 megabytes of overt traffic between the hosts. If there is significant traffic between the hosts (e.g. 10% utilization on a 10Mb/s LAN) the entire verification occurs fairly quickly (approximately 8.9s). Faster link rates yield proportionately faster detection times; regretably, the technique is not as responsive when traffic rates between the hosts is low.

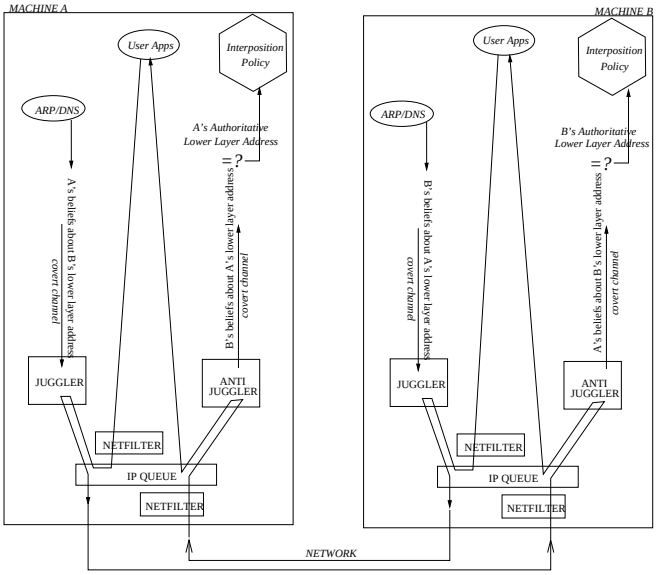

Figure 6: The architecture of PERMEATE-MITM.

5. CONCLUSION AND FUTURE WORK

In this paper, we present new effective techniques to detect interposition attacks on connection streams that apply in any circumstance where the man-in-the-middle attacker achieves interposition by corrupting higher-layer to low-layer address mappings. We demonstrated that the scheme is an effective way to address both ARP and DNS poisoning simultaneously, and does not require redesigning existing software. The proposed system addresses the *detection* of interposition attacks in order to circumvent the full overhead and complexity required for stream confidentiality and integrity. We demonstrated its efficacy, and provide a publicly accessible prototype implementation of the system. Future efforts will involve extending the system to make it more responsive in low traffic situations.

6. REFERENCES

[1] L. Loeb, "On the lookout for dsniff: Part 1." [Online]. Available: http://www.ibm.com/developerworks/library/s-sniff.html

[2] J. Erickson, *Hacking: The Art of Exploitation, No Starch Press, First Edition, 2003.*

[3] P. Biondi, "Scapy." [Online]. Available: http://www.secdev.org/projects/scapy/

[4] M. V. Alberto Ornaghi, "Ettercap." [Online]. Available: http://ettercap.sourceforge.net/

[5] R. Bejtlich, "Old school layer 2 hacking." [Online]. Available: http://taosecurity.blogspot.com/2008/06/old-school-layer-2-hacking.html

[6] H. D. Moore, "Full disclosure: Re: Metasploit - hack ?" [Online]. Available: http://seclists.org/fulldisclosure/2008/Jun/0011.html

[7] ——, "Full disclosure: Re: Metasploit - hack ? (analysis)." [Online]. Available: http://seclists.org/fulldisclosure/2008/Jun/0013.html

[8] D. Song, "Dsniff faq." [Online]. Available: http://monkey.org/~dugsong/dsniff/faq.html#How%20do%20I%20detect%20dsniff%20on%20my%20network

[9] unknown, "anti-arp." [Online]. Available: http://sync-io.net/Sec/anti-arpspoof.aspx

[10] K. Levinson, "comp.security.misc: Re: Defending arp spoofing." [Online]. Available: http://www.derkeiler.com/Newsgroups/comp.os.ms-windows.nt.admin.security/2005-11/0004.html

[11] L. N. R. Group, "Arpwatch." [Online]. Available: http://ee.lbl.gov/

[12] I. Cisco Systems, "Catalyst 4500 series switch cisco ios software configuration guide, 12.1(13)ew." [Online]. Available: http://www.cisco.com/en/US/docs/switches/lan/catalyst4500/12.1/13ew/configuration/guide/pref.html

[13] A. D. Pasquale, "Arpon." [Online]. Available: http://arpon.sourceforge.net/

[14] R. McMillan, "Dns attack writer a victim of his own creation." [Online]. Available: http://www.pcworld.com/businesscenter/article/149126/dns_attack_writer_a_victim_of_his_own_creation.html

[15] R. Naraine, "Websense reports china netcom dns cache poisoning." [Online]. Available: http://blogs.zdnet.com/security/?p=1776

[16] K. Poulsen, "Comcast hijackers say they warned the company first." [Online]. Available: http://blog.wired.com/27bstroke6/2008/05/comcast-hijacke.html

[17] C. Vulnerabilities and Exposures, "Cve-2008-1447." [Online]. Available: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-1447

[18] D. Kaminsky, "Black ops 2008: ItâĂŹs the end of the cache as we know it." [Online]. Available: http::://www.doxpara.com/DMK_BO2K8.ppt

[19] R. Bejtlich, "Thoughts on latest kaminsky dns issue." [Online]. Available: http://taosecurity.blogspot.com/2008/07/thoughts-on-latest-kaminski-dns-issue.html

[20] S. Friedl, "An illustrated guide to the kaminsky dns vulnerability." [Online]. Available: http://www.unixwiz.net/techtips/iguide-kaminsky-dns-vuln.html

[21] H. Moore and I)ruid, "Kaminsky dns cache poisoning exploit for metasploit." [Online]. Available: http://www.caughq.org/exploits/CAU-EX-2008-0002.txt

[22] Microsoft, "Microsoft security bulletin ms08-037." [Online]. Available: http://www.microsoft.com/technet/security/Bulletin/MS08-037.mspx

[23] "Opendns." [Online]. Available: http://www.opendns.com/

[24] V. Jacobson, "Congestion avoidance and control," *Computer Communication Review*, vol. 18, pp. 314–329, Aug. 1988.

[25] J. Levy, J. Paduch, and B. Khan, "Superimposing permutational covert channels onto reliable stream protocols," in *Proceedings of MALWARE 2008, Alexandria VA*, Oct. 2008.