Chapter 1

Security Issues in the TCP/IP Suite

Prabhaker Mateti

Department of Computer Science and Engineering
Wright State University
Dayton, Ohio 45435
http://www.cs.wright.edu/~pmateti/

The TCP/IP suite has many design weaknesses so far as security and privacy are concerned. Some of these are protocol design weaknesses per se, whereas the rest are defects in the software that implements the protocols. In this paper, we describe these issues from a practical perspective.

1.1. Introduction

This paper is an overview of security attacks in the core protocols (IP, UDP, and TCP) and infrastructure protocols (ARP, ICMP, DNS). However, we do not address the exploits in various application protocols, but do focus on additional issues such as covert channels. We describe these issues from a practical perspective.

Some of these are protocol design weaknesses per se, whereas the rest are defects in the software that implements the protocols. IP, UDP, TCP and the infrastructure protocols were designed at a time when security concerns were almost non-existing and trust was assumed. While this paper summarizes design weaknesses in the TCP/IP suite from a security point of view, it is important to remember that many implementations have "fixed" these weaknesses, but are not described in RFCs. We assume that the reader is fluent in TCP and IP details.

Protocol weaknesses can be divided into those due to (i) the design of the protocol itself, and (ii) the configuration, deployment and daily operation of the DNS servers. As can be expected, there is a strong interplay between the two.

All major OS have made improvements in their implementations of the protocol stack that mitigate or disable many of the attacks described below. Of course, the attack tools also improve. A number of enhancements for TCP/IP have been made that are not yet in common use. Several of them (e.g., DNSSEC and IPv6) involve heavy use of encryption and require more computing power. As computing power in end-user hosts increases, we expect to see these universally deployed.

Bellovin[1] gives broad coverage of security issues in TCP/IP and reminisces on an earlier version[2] The papers by Arce,[3] and Schneier[4] describe attack trends of recent years.

1.2. Attack Techniques

This section briefly describes the techniques that are in the arsenal of a TCP/IP attacker. Rigorous definitions for these are not (yet) available. Successful attacks are almost always a combination of these basic techniques.

1.2.1. *Sniffing*

Sniffing is eavesdropping on the network. A (packet) sniffer is a wire-tap program. Sniffing is the act by machine S of making copies of a network packet sent by machine A intended to be received by machine B. Such sniffing, strictly speaking, is not a TCP/IP problem, but it is enabled by the near-universal choice of Ethernet, a broadcast media, as the physical and data link layers. Sniffing can be used for monitoring the health of a network as well as capturing the passwords used in telnet, rlogin, and FTP connections. Attackers sniff the data necessary in the exploits described below. Depending on the equipment used in a LAN, a sniffer needs to be run either on the victim machine whose traffic is of interest or on some other host in the same subnet as the victim. In the normal mode, an NIC captures only those frames that match its own MAC address. In the so-called promiscuous mode, an NIC captures all frames that pass by it. The volume of such frames makes it a real challenge for an attacker to either immediately process all such frames fast or clandestinely store them for later processing. An attacker at large on the Internet has other techniques that make it possible to install remotely a sniffer on the victim machine.

Attacks that do not sniff and therefore cannot see the information in the packet flows are called *blind* attacks.

1.2.2. *Buffer Overflow*

A large number of TCP/IP server programs suffer from a class of programming errors known as buffer overflows. Many of these server programs run with the privileges of a super user. Among the many servers that suffer from such bugs are several implementations of FTP servers, the ubiquitous DNS server program called `bind`, the popular mail server called `sendmail`, and the Web server `IIS`, to name a few. An attacker supplies cleverly constructed inputs to such programs causing them to transfer control to executable code she has supplied. A typical code produces a shell that she can interact with from a remote machine with all the privileges of the super user.

1.2.3. *Spoofing*

Spoofing refers to altering (portions of) a packet so that the overall packet remains structurally legitimate (e.g., checksums are valid) but the "info" it contains is fake. Spoofing often accompanies sniffing, but may newly manufacture packets with fake values. Spoofed packets are injected into the network.

1.2.4. *Poisoning*

Many network services are essentially mappings implemented as table lookups. The mappings are dynamic, and update methods are well-defined. Unfortunately, who is qualified to provide the updates, and how messages that provide update information are to be authenticated are ill-defined. An attacker takes advantage of this, and provides fake updates causing the table to be "poisoned."

1.2.5. *Illegal Packets*

Packets containing "unexpected" values in some of the fields are illegal in the sense that a legitimate sender would not have constructed them. Software in the receiver ought to check for such illegal packets, but RFCs were ambiguous and/or legacy software was not cautious. Attackers have written special programs that construct illegal packets and cause the receiving network hosts to crash or hang. The so-called `Ping of Death` attack of 1996 sent an ICMP echo request (ping) packet that was larger than the maximum permissible length ($2^{16} - 1$). TCP segments have a number of flags that have, collectively, a strong influence on how the segment is processed. However, not all the flags can be independently set or reset. For example, SYN+FIN, SYN+FIN+PSH, SYN+ FIN+ RST, and SYN+ FIN+ RST+ PSH are all illegal combinations. Past implementations have accounted only for valid combinations, ignoring the invalid combinations as "will not happen." An IP packet should not have source address and port equaling the destination address and port. The 1997 attack tool called `land` exploited this vulnerability.

1.2.6. *Finger Printing a System*

An attacker wants to remotely scan entire LANs and identify what services run where, etc. and to identify the exact version of an OS running on a targeted victim because operating system exploits will usually only work against a specific operating system or software running. Nuances in the TCP/IP stacks implemented in the various OS, and versions of the same OS, make it possible to remotely probe the victim host and identify the OS. Such probing deliberately constructs illegal packets, and attempts to connect to each port and observes the responses it gets.

The tool called `nmap` (`http://insecure.org/nmap/`) is comprehensive in this regard. Beverly[5] describes yet another classifier. However, Panjwani et al.[6] report an experimental evaluation that over 50% of the attacks were *not* preceded by a scan.

To make such finger printing difficult, a scrubber should be deployed; see Section 1.11.

1.2.7. *Storms*

A storm is the flow of a certain kind packets that is abnormally high. For example, if 50 ARP requests per second is normal, we would consider, say, 500 per second abnormally high. Storms are often created by a few packets generated on a compromised host. The attackers send the packets to intermediary machines (amplifiers or reflectors). The packets are often source spoofed also.

There have been several attacks that generate enormous numbers of packets rendering (portions of) a network ineffective. These amplify the numbers of packets into a "storm." Section 1.8.8 describes an ACK storm.

Clever use of broadcast or multicast addresses helps the storms.

1.2.8. *Denial of Service*

The goal of a denial of service (DoS) attack is to prevent legitimate clients from receiving service(s). Since the service being performed is almost always a constant time operation, it is easy for an external observer process to detect a DoS attack. These attacks are generally transient.

1.2.9. *Distributed Denial of Service*

An attack whose main gaol is DoS, and the attack is conducted by a coordinated set of hosts, it is termed a distributed denial of service (DDoS) attack. The hosts (often called zombies) that participate in such an attack are compromised through other techniques such as buffer overflow. The attacker prepares the zombies ahead of the DDoS attack, and issues a start up command remotely through an installed backdoor.

1.3. ARP Poisoning

ARP (RFC 826, 1982) discovers the (Ethernet) MAC address of a device whose IP address is known. This needs to be done only for outgoing IP packets, because IP datagrams must be (Ethernet) framed with the destination hardware address. The translation is performed with a table look-up. Operating systems maintain this table known as the ARP *cache*. When an entry is not found in the cache, the OS uses ARP to broadcast a query. Reverse ARP (RARP) (RFC 903) allows a host to discover its own IP address by broadcasting the Ethernet address and expecting a server to reply with the IP address.

ARP poisoning is an attack technique that corrupts the ARP cache with wrong Ethernet addresses for some IP addresses. An attacker accomplishes this by sending an ARP response packet that is deliberately constructed with a "wrong" MAC address. The ARP is a stateless protocol. Thus, a machine receiving an ARP response cannot determine if the response is because of a request it sent or not. Also, there is neither a verification of sender identify nor any authentication of the information received. It is simple enough that there are attack tools that can cause the poisoning within microseconds.

ARP poisoning enables the so-called man-in-the-middle attack that can defeat cryptographic communications such as SSH, SSL and IPSec. An attacker on machine M inserts him- or herself between two hosts A and B by (i) poisoning A so that B's IP address is associated with M's MAC address, (ii) poisoning B so that A's address is associated with M's MAC address, and (iii) relaying the packets M receives A from/to B. ARP packets are not routed, and this makes it very rewarding to the attacker if a router can be ARP poisoned.

To defend, we must monitor change in the ARP cache using tools such as `arpwatch`. Unfortunately, it is not possible to block the poisoning other than after-the-fact "immediate" cleanup. Setting static mapping of IP to MAC addresses eliminates the problem but this does not scale to large networks. Note that ARP does not use an IP header and tracking the attackers is difficult. It is often suggested that one should periodically broadcast MAC address of important servers and gateway, which cleans up poisoning of corresponding entries.

Ramachandran and Nandi[7] present an active technique to detect ARP spoofing. They inject an ARP request and TCP SYN packets into the network to probe for inconsistencies. Trabelsi and Shuaib[8] present a similar technique. Goyal et al.[9] present a new technique to make ARP secure and provide protection against ARP cache poisoning. This technique combines digital signatures and one time passwords based on hash chains. Gouda and Huang[10] propose a secure address resolution protocol consisting of a secure server, an invite-accept protocol and a request-reply protocol. S-ARP[11] is another new protocol that uses host public/private key pairs and

ARP messages are digitally signed by the sender.

1.4. ICMP Exploits

Internet Control Message Protocol (RFC 792, 1981; RFC 950, 1985) is a required protocol that manages and controls the IP layer. In general, much of the best effort in delivering IP datagrams is associated with ICMP. The purpose of the ICMP messages is to provide feedback and suggestions about problems. The popular network utilities ping and traceroute use ICMP. The ICMP protocol is a simple protocol with one message per packet. ICMP is in the network layer. But, an ICMP message is encapsulated as an IP datagram. These are treated like any other IP datagrams.

ICMP is also one of the easiest to exploit. The attack tool of 1997, called `smurf` sends ICMP ping messages. There are three machines in smurfing: the attacker, the intermediary router, and the victim. The attacker sends to an intermediary an ICMP echo request packet with the IP broadcast address of the intermediary's network as the destination. The source address is spoofed by the attacker to be that of the intended victim. The intermediary puts it out on that network. Each machine on that network will send an ICMP echo reply packet to the source address. The victim is subjected to network congestion that could potentially make it unusable.

It made possible several other attacks such as route redirection, reconnaissance and scanning.[1] Arkin[12] describes an OS finger printing method based on ICMP. Berg and Dibowitz[13] note that "over-zealous security administrators are breaking the Internet" in how firewalls are configured. They note that firewall administrators set up path MTU discovery, but block ICMP type 3 code 4 packets required for the protocol to work. ICMP also enables covert channels as described in Section 1.10.

1.5. IPv4 Exploits

This section describes what is often simply called "IP spoofing." which is the spoofing of the source address field. Being the carrier protocol without any guarantee regarding payload integrity, there are other spoofing possible but such techniques belong in higher layers.

1.5.1. *IP Address Spoofing*

The IP layer of the typical OS simply trusts that the source address, as it appears in an IP packet is valid. It assumes that the packet it received indeed was sent by the host officially assigned that source address. The IP protocol specifies no method for validating the authenticity of this address. Replacing the true IP address of the sender (or, in rare cases, the destination) with a different address is known as IP spoofing. Because the IP layer of the OS normally adds these IP addresses to a data packet, a spoofer must circumvent the IP layer and talk directly to the raw network device.

IP spoofing is used as a technique aiding an exploit on the target machine. For example, an attacker can silence a host A from sending further packets to B by sending a spoofed packet announcing a window size of zero to A as though it originated from B.

Note that the attacker's machine cannot simply be assigned the IP address of another host T, using `ifconfig` or a similar configuration tool. Other hosts, as well as T, will discover (through ARP, for example) that there are two machines with the same IP address.

1.5.1.1. *Detection of IP spoofing*

We can monitor packets using network-monitoring software. A packet on an external interface that has both its source and destination IP addresses in the local domain is an indication of IP spoofing. Another way to detect IP spoofing is to compare the process accounting logs between systems on your internal network. If the IP spoofing attack has succeeded on one of your systems, you may get a log entry on the victim machine showing a remote access; on the apparent source machine, there will be no corresponding entry for initiating that remote access. Templeton and Levitt[14] describe variety of more advanced methods for detecting spoofed packets.

1.5.1.2. *Prevention of IP Spoofing*

All routers must employ proper IP filtering rules. They should only route packets from sources that could legitimately come from the interface the packet arrives on. Most routers now have options to turn off the ability to spoof IP source addresses by checking the source address of a packet against the routing table to ensure the return path of the packet is through the interface it was received on.

1.5.2. *IP Fragment Attacks*

A well-behaving set of IP fragments is non-overlapping. Malicious fragmentation involves fragments that have illegal fragment offsets. A fragment-offset value gives the index position of this fragment's data in a reassembled packet. For example, the fragments may be so crafted that the receiving host in its attempts to reassemble calculates a negative length for the second fragment. This value is passed to a function (such as `memcpy()`) that copies from/to memory, which takes the a negative number to be an enormous unsigned (positive) number. A pair of carefully crafted but malformed IP packets thus causes a server to "panic" and crash. The 1997 attack tool called `teardrop` exploited this vulnerability.

Note that the RFCs require no intermediate router to reassemble fragmented packets. Obviously the destination must reassemble. Many firewalls do not perform packet reassembly in the interest of efficiency. These only consider the fields of individual fragments. Attackers create artificially fragmented packets to fool such firewalls.

In a so-called tiny fragment attack, two fragments are created where the first one is so small that it does not even include the destination port number. The second fragment contains the remainder of the TCP header, including the port number. A variation of this is to construct the second fragment packet with an offset value less than the length of the data in the first fragment so that upon packet reassembly it overrides several bytes of the first fragment (e.g., if the first fragment was 24 bytes long, the second fragment may claim to have an offset of 20). Upon reassembly, the data in the second fragment overwrites the last 4 bytes of the data from the first fragment. If these were fragments of a TCP segment, the first fragment would contain the TCP destination port number, which is overwritten by the second fragment. Such techniques do not cause a crash or hang of a targeted system but can be used to bypass simple filtering done by some firewalls.

Fragmentation attacks are preventable. Unfortunately, in the IP layer implementations of nearly all OS, there are bugs and naive assumptions in the reassembly code.

1.6. Routing Exploits

There have been many exploits of all well-known routing protocols.[1]

Wang et al.[15] describe a "path-filtering approach to protect the routes to the critical top-level DNS servers." Hsu and Chiueh[16] describe a new router prototype in a centralized TCP architecture based on honeypot ideas. Kent et al.[17,18] present a new secure Border Gateway Protocol called S-BGP. Hu et al.[19] present a symmetric cryptographic mechanism to guard an ASPATH from alteration, and propose a new protocol that is claimed to be several times faster than S-BGP.

1.7. UDP Exploits

UDP (RFC 768, 1980) is a connectionless protocol belonging to the transport layer (OSI layer 4). It is a thin protocol on top of IP, providing high speed but low functionality. UDP does not guarantee the delivery of datagrams. Messages can be delivered out of order, delayed, or even lost. Datagrams may get duplicated without being detected. The UDP protocol is used mostly by application services where squeezing the best performance out of existing IP network is necessary, such as Trivial File Transfer (TFTP), NFS, and DNS. Unfortunately, UDP cannot provide security and privacy of the data flow.

A UDP flood attack sends a large number of UDP packets to random ports. Such ports may be open or closed. If open, an application listening at that port needs to respond. If closed, the network layer, replies with an ICMP Destination Unreachable packet. Thus, the victim host will be forced into sending many ICMP packets and wasting computing cycles. If the flooding is large enough, the host will eventually be unreachable by other clients. The attacker will also IP-spoof the UDP packets, both to hide and to ensure that the ICMP return packets do not reach him.

Surprisingly, using legitimate applications or OS services an attacker can generate a storm of packets. On many systems, the standard services known as `chargen` that listens typically at port 19 and echo that listens typically at port 7 are enabled. Chargen sends an unending stream of characters intended to be used as test data for terminals. The `echo` service just echoes what it receives. It is intended to be used for testing reachability, identifying routing problems, and so on. An attacker sends a UDP packet to the port

19 with the source address spoofed to a broadcast address, and the source port spoofed to 7. The chargen stream is sent to the broadcast address and hence reaching many machines on port 7. Each of these machine will echo back to the victim's port 19. This ping-pong action generates a storm of packets.

An attack called `fraggle` uses packets of UDP echo service in the same fashion as the ICMP echo packets.

To defend, most hosts disable many UDP services such as the chargen and echo mentioned above. Because UDP is better suited for streaming applications, there are suggestions to run UDP over SSL or even create a protocol[20] immediately above UDP.

1.8. TCP Exploits

TCP/IP is vulnerable to variety of attacks[21] ranging from password sniffing to denial of service attacks of several kinds mostly because of design weaknesses.

1.8.1. *TCP Sequence Number Prediction*

TCP exploits are typically based on IP spoofing and sequence number prediction.[22] In establishing a TCP connection, both the server and the client generate an initial sequence number (ISN) from which they will start counting the segments transmitted. Host Y accepts the segments from X only when correct SEQ/ACK numbers are used. The ISN is (should be) generated at random and should be hard to predict.[2] However, some implementations of the TCP/IP protocol make it rather easy to predict this sequence number. The attacker either sniffs the current SEQ+ACK numbers of the connection or can algorithmically predict them.

1.8.2. *Closing Connections*

An established TCP connection is expected to be close by the 4-way handshake that involves sending FIN segments. An attacker can enact this easily. The attacker constructs a spoofed FIN segment and injects it fast. It will have the correct SEQ number, that is computed from segments sniffed from the immediate past, so that it is accepted by the targeted host. This host would believe the (spoofed) sender had no data left. Any segments that may follow from the legitimate sender would be ignored as bogus. The rest of the four-way handshake is also supplied by the attacker.

A similar connection killing attack using RST is also well known. Section 1.8.3 describes a blind reset attack that does not involve sniffing.

1.8.3. *TCP Reset Attack*

The recent (2004) TCP reset attack lead to severe concerns in the routing protocol BGP used in large routers. This is attack does not involve sniffing, but does depend on having done enough reconnaisance regarding the BGP partners and the source port numbers they use.

TCP protocol requires that a receiver close a connection when it receives an RST segment, even if it has a sequence number that is not an exact match of what is expected, as long as the number is within the window size. Thus, the larger the window size the easier it is to guess a sequence number to be placed in a spoofed TCP RST segment. Watson[23] gives a detailed account of this attack and presents experimental evidence that the attack can be successful in a matter of a few seconds.

Arlitt and Williamson[24] present a one-year study of Internet packet traffic from a large campus network, showing that 15-25percent of TCP connections have at least one TCP RST (reset). While this study does not claim that the RST packets are part of attacks, it does imply that detection is therefore more difficult.

1.8.4. *Low-Rate/Shrew TCP Attacks*

The low-rate TCP attack,[25] also known as the shrew attack, exploits the congestion control mechanism, forces other well-behaving TCP flows to back off and enter the retransmission timeout state. The TCP flow of the attack is low rate stream exploiting protocol homogeneity.

Congestion is a condition of significant delay caused by overload of datagrams at one or more routers. A congestion window size `cwnd` that limits the number of outstanding unacknowledged bytes that are allowed at any time is dynamically computed by the sender based on network congestion. The TCP sliding window size is the smaller of rwnd and `cwnd`. TCP improves throughput by avoiding congestion. When a segment loss is detected, TCP assumes that the loss is due to congestion. There are many variations on how `cwnd` should be adjusted.

TCP Slow-Start Exponential Increase algorithm doubles `cwnd` after a round trip time (RTT, typically milliseconds) until a slow-start threshold is reached. TCP Congestion Avoidance Additive Increase takes over when the `cwnd` reaches the slow-start threshold. Then `cwnd` is increased by 1 until retransmissions begin to occur either because an RTO has timed out (typically in seconds), or because three ACKs are receive. The Congestion Detection Multiplicative Decrease then takes effect and the size of the threshold is dropped to half. Depending on the implementation, either (i) `cwnd` is set to one segment, and the slow start phase is entered, or (ii) `cwnd` is set to the threshold, and the congestion avoidance phase is entered. Upon further loss, RTO doubles with each subsequent time out. If a segment is successfully received, TCP reenters the slow start.

Shrew attacks consist of short bursts that repeat with a fixed, frequency. If the total traffic (shrew and regular TCP traffic) during an RTT-timescale burst is sufficient to induce enough segment losses, the TCP flow will enter a timeout and attempt to send a new segment RTO seconds later. If the period of the shrew flow approximates the RTO of the TCP flow, the TCP flow will continually incur loss as it tries to exit the timeout state, fail to exit timeout, and obtain near zero throughput.

TCP congestion control has undergone major improvements in recent years resulting in many TCP variants that are soon to be adopted in actual implementations. E.g., Linux 2.6 has TCP BIC, and TCP CUBIC and Windows Vista has TCP New Reno and Compound TCP.

Vulnerability to low-rate DoS attacks is not an easily fixed TCPdesign flaw. Being a low average rate flow, the shrew attack is difficult to detect. Also, there is a large family of such attack patterns.

We consider a distributed approach to detect and to defend against the low-rate TCP attack [7]. The low-rate TCP attack is essentially a periodic short burst which exploits the homogeneity of the minimum retransmission timeout (RTO) of TCP flows and forces all affected TCP flows to back off and enter the retransmission timeout state. This sort of at- tack is difficult to identify due to a large family of attack pat- terns.

Guirguis et al.[26] expose new variants of these low-rate attacks that could potentially have high attack potency per attack burst. Detection and defense mechanisms capable of mitigating the attack are being developed by several groups.[27,28]

1.8.5. *ACK Tricks*

Savage et al.[29] describe how a (misbehaving) receiver can convince a sender to send at a too high rate and gain an unfairly large portion of the available bandwidth. (i) ACK Division: The receiver sends many ACKs for subsets of each received segment, increasing the congestion window with each ACK. (ii) DupACK spoofing: Sends a large number of duplicate ACKs, even though a segment has not been lost. Each DupACK increases the window. (iii) Optimistic ACKing: Acknowledges segments that have not been received (yet). This causes a faster slow start, but, of course, creates problems if the segment is actually lost.

1.8.6. *Illegal Segments: SYN+FIN*

The TCP specification does not specify clearly certain transitions. As an example, suppose an attacker sends a TCP segment with both the SYN and the FIN bit set. Depending on the TCP implementation, victim host processes the SYN flag first, generates a reply segment with the corresponding ACK flag set, and perform a state-transition to the state SYN-RCVD. Victim host then processes the FIN flag, performs a transition to the state CLOSE-WAIT, and sends the ACK segment back to attacker. The attacking host does not send any other segment to victim. TCP state machine in the victim for this connection is in CLOSE-WAIT state. The victim connection gets stuck in this state until the expiry of the keep-alive timer.

1.8.7. *Simultaneous Connections*

Occasionally, it is possible that hosts XX and YY both wish to establish a connection and both of them simultaneously initiate the handshake. This is called simultaneous connection establishment [RFC 793]. Both hosts XX and YY send out SYN's to each other. When the SYN's are received, each receiver sends out a SYN+ACK. Both hosts XX and YY must detect that the SYN and SYN+ACK actually refer to the same connection. If both hosts XX and YY detect that the SYN+ACK belongs to the SYN that was recently sent, they switch off the connection establishment timer and move directly to the SYN-RECVD state. This flaw could be used to stall a port on a host, using protocols such as FTP where the server initiates a connection to the client. As an example, consider (malicious) host XX which has started an (active) FTP connection to a server YY. XX and YY are connected using the control-port (21 on YY). YY initiates the connection establishment procedure to initiate data transfer with XX. YY sends a SYN to XX, and makes a transition to SYN-SENT state. YY also starts the connection establishment timer. XX receives the SYN, and responds with another SYN. When YY receives the SYN, it assumes that this is a case of a simultaneous open connection. So, it sends out SYN-ACK to XX, switches off the connection establishment timer, and transitions to the state SYN-RCVD. XX receives the SYN-ACK from YY, but does not send a reply. Since YY is expecting a SYN-ACK in the SYN-RCVD state, and there is no timer, YY gets stalled in SYN-RCVD state. XX was able to create a denial-of-service attack.

1.8.8. *Connection Hijacking*

If an attacker on machine Z can sniff the segments between X and Y that have a TCP connection, Z can hijack the connection. Z can send segments to Y spoofing the source address as X, at a time when X was silent. (Z can increase the chances of this by launching a denial of service attack against X.) Y would accept these data and update ACK numbers. X may subsequently continue to send its segments using old SEQ numbers, as it is unaware of the intervention of Z. As a result, subsequent segments from X are discarded by Y. The attacker Z is now effectively impersonating X, using "correct" SEQ/ACK numbers from the perspective of Y. This results in Z hijacking the connection: host X is confused, whereas Y thinks nothing is wrong as Z sends "correctly synchronized" segments to Y. If the hijacked connection was running an interactive shell, Z can execute any arbitrary command that X could. Having accomplished his deed, a clever hijacker would bow out gracefully by monitoring the true X. He would cause the SEQ numbers of X to match the ACK numbers of Y by sending to the true X a segment that it generates of appropriate length, spoofing the sender as Y, using the ACK numbers that X would accept.

Such connection forgery is also possible to do "blind", i.e., without being able to sniff. The attacker guesses that a connection exists, and guesses a valid port and sequence numbers. Note that well-known routers use well-known ports and stay connected for fairly long periods.

ACK storms are generated in the hijack technique described above. A host Y, when it receives packets from X after a hijack has ended, will find the packets of X to be out of order. TCP requires that Y must send an immediate reply with an ACK number that it expects. The same behavior is expected of X. So, X and Y send each other ACK messages that may never end.

1.8.9. *Connection Flooding*

TCP RFC has no limit set on the time to wait after receiving the SYN in the three-way handshake. An attacker initiates many connection requests with spoofed source addresses to the victim machine. The victim machine maintains data related to the connection being attempted in its memory. The SYN+ACK segments that the victim host sends are not replied to. Once the implementation imposed limit of such half-open connections is reached, the victim host will refuse further connection establishment attempts from any host until a partially opened connection in the queue is completed or times out. This effectively removes a host from the network for several seconds, making it useful at least as a stepping tool to other attacks, such as IP spoofing.

Chen describes defenses against TCP SYN flooding attacks under different types of IP spoofing.[30]

"SYN cookies"[31] are particular choices of initial TCP sequence numbers. A server that uses SYN cookies sends back a SYN+ACK, when its SYN queue fills up. It also must reject TCP options such as large windows, and it must use one of the eight MSS values that it can encode. When the server receives an ACK, it checks that the secret function works for a recent value of a 32-bit time counter, and then rebuilds the SYN queue entry from the encoded MSS.

1.9. DNS Exploits

DNS (Domain Name Service; RFC 1034, RFC 1035) is about associating names such as `osis110.cs.wright.edu` with their IP addresses and vice-versa. Thus, one might argue that it is not an essential protocol. However, there is no debating that the mnemonic value it provides to humans and the flexibility it provides for scaling and re-configuring through mapping several IP addresses to same mnemonic name have been essential in the growth of Internet.

1.9.1. *Protocol Refresher*

The DNS name space is a tree hierarchy. A fully qualified domain name is the sequence of labels, separated by a dot, on the path from a node to the root of the tree. The domain name space is maintained as a database distributed over several domain name servers. A server can delegate the maintenance of any sub-domain to another server. A delegated sub-domain in the DNS is called a zone. The parent server keeps track of such delegations. Each name server has authoritative information about one or more zones. It may also have cached, but non-authoritative, data about other parts of the database. A name server marks its responses to queries as authoritative or not.

The database is a collection of *resource records* (RR), each of which contains a domain name and four attributes: (i) The record type identifies what is stored in the data value. (ii) The class attribute of the record is "IN" for Internet. (iii) The time-to-live (TTL) value indicates how long, in seconds, a non-authoritative name server can cache the record. Some record types are: A, WKS, PTR, HINFO, MX, and AAAA. The data of an A record type is an IPv4 address, of an AAAA record is an IPv6 address, of an MX record is the canonical host name of the mail server and its IP address, of a PTR record is a pointer used to map an IP address to a domain name. An HINFO record type gives a description of hardware and operating system used by that host.

The DNS *resolver* is a piece of software. Every host is configured with at least one local name server N if it is to find hosts not listed in the etc/hosts file. Each host maintains a short cache table that maps fully qualified domain names to IP addresses. When a name is not found in either this file or this cache, the host enquires with N using the resolver. Either TCP or UDP can be used for DNS depending on the length of the DNS response, connecting to server port 53. TCP is used for zone transfers, UDP is used for look ups.

The primary function of DNS is to answer a query to translate a fully qualified domain name into its IP address. This is done by retrieving the A record. A reverse look-up (also called an inverse query) is to find the host name given the IP address. This is done by retrieving the PTR record.

The protocol is stateless – all the information needed is contained in a single message. DNS messages are either queries or responses. The messages begin with a header field of 12 bytes beginning with a 16-bit identifier. The DNS response message includes (i) the question section of the query message that caused it, followed by the (ii) answer, (iii) authoritative, and (iv) additional sections.

An *iterative DNS query* to a name server D receives a reply with either the answer or the IP address of the next name server. If the name is in the local zone, the local name server N can respond to a query directly. Otherwise, N queries one of the root servers. The root server gives a referral with a list of name servers for the top-level domain of the query. N now queries a name server on this list and receives a list of name servers for the second-level domain name. The process repeats until N receives the address for the domain name. N returns the address or other DNS data to the querying host, and caches the record for the next TTL (time to live) seconds.

A *recursive DNS query* to D will make D obtain the requested mapping on behalf of the querying host. If D does not have the answer, it forwards the query to the next name server in the chain, and so on until either an answer is found or all servers are queried and hence returns an error code. Because recursive look-ups take longer and need to store many records, it is more efficient to provide a recursive DNS server for LAN users and an iterative server for Internet users.

The mapping of names to addresses is not required to be one-to-one. It is possible to associate a name with multiple IP addresses thus providing load distribution. A single IP address can be associated with multiple names, one being a canonical name and others perhaps more mnemonic thus providing host aliasing.

The name to IP address mapping changes often. Adding a new host, deleting an existing one, or changing the IP address, etc. are accommodated by the protocol wherein a server is given updates by others.

The DNS protocol did not specify the qualifications of the servers that can supply the updates. Nor did it specify that such servers should be authenticated. Apart from checking that source and destination IP addresses and ports matched, the transaction ID (a 16-bit number in the DNS message header) is the sole "authentication".

1.9.2. *Security Threats of the Protocol*

1.9.2.1. *DNS zone transfers*

Questioning the legitimacy of a zone transfer request is left out of the protocol. It is also possible to include a zone transfer gratuitously as part of a response to a legitimate query.

1.9.2.2. *DNS Cache Poisoning*

Cache poisoning happens when a DNS server D updates ("poisons") its cache based on misinformation supplied by Z a host/server in the control of an attacker. This may be about just one domain name or an entire zone if D accepted a zone transfer from Z. All the clients of this server will then receive invalid responses until the TTLs of these entries expire.

1.9.2.3. *DNS Forgery*

The DNS answers that a host receives may have come from an attacker who sniffs a query between the victim resolver and the legitimate name servers and responds to it with misleading data faster than the legitimate name server does. The attacked host may in fact be a DNS server. DNS forgery is also called spoofing.

A remote attacker, who cannot sniff a query, needs to guess the query time, the transaction ID and (perhaps) the query port.

Consider n different DNS clients sending simultaneous queries to a DNS server D to resolve the same domain name dnm. D will, in general, forward the n requests received to other DNS servers, starting from root-servers and trying to get replies for each of the n requests. These requests will be processed independently, and will be assigned different identifiers. An attacker produces this scenario by simultaneously sending n queries to D using same domain name dnm but a different IP source address in each query. While D is now waiting for n replies with different IDs for the resolution of dnm, the attacker sends several replies to D with guessed IDs and source ports. The probability of success is increased by the so-called birthday effect. (The birthday effect is the fact that among 60 randomly chosen people, the probability that there are two born on the same day is 95%.) A determined attacker may also launch DoS attacks to the other DNS servers so as to increase the waiting time of D.

1.9.2.4. *Domain Hijack*

A domain is hijacked when an attacker is able to redirect queries to servers that are under the control of the attacker. This can happen because of (i) cache poisoning, (ii) forgery, or (iii) a domain server has been compromised. DNS hijacking is also known as redirection.

1.9.3. *DNS Infrastructure*

DNS is not as "distributed" as one might think. The database is partitioned and each partition is in the control of a small number (usually just one) of servers. There is no replication and no consensus of replies from multiple servers.

A 2005 survey by the Beehive group[32] found the following: Of the 535000 domains and 164000 name servers scanned, 79%of domain names rely on two or fewer servers. 33%of domains have a single bottleneck link whose failure would result in disappearance of that domain. 20%of DNS servers contain security vulnerabilities that enable attackers to spoof records or block their distribution entirely. An attacker exploiting well-documented vulnerabilities in DNS name servers can hijack more than 30%of the names appearing in the Yahoo and DMOZ.org directories. And certain name-servers, especially in educational institutions, control as much as 10%of the name space.

1.9.3.1. *Server Software*

The server programs that provided this service were, over the years, almost without exception, prone to buffer overflow exploits. The BIND package is a widely deployed implementation of DNS. BIND does considerably more than what the DNS RFCs require. It has extensive tuning of its configuration that can "harden" the service. Unfortunately, this otherwise highly capable software, in many releases has had much vulnerability of its own that allowed remote attackers to launch denial-of-service attacks, hijack domain names, or use vulnerable DNS servers to gain access to other systems.

The server software usually runs with the privileges of the super-user. Thus, a buffer overflow attack compromises the entire machine on which the server software is running.

1.9.3.2. *Denial of Service Attack*

It is easy to flood a DNS server with recursive queries. The CPU and memory usage of the server eventually reaches its maximum, and the DNS Server service becomes unavailable causing many network services to also become unavailable. Such flooding not only can use spoofed IP addresses but can also employ a distributed network of zombie machines.

1.9.3.3. *Reconnaissance*

DNS zone transfers help map the targeted network during the reconnaissance stage of an attack. An attacker commonly begins with this foot printing. Note that ordinary hosts and servers are often named mnemonically indicating their function or location. DNS servers can now be configured to refuse zone transfers.

1.9.4. ***DNSSEC***

The security of the DNS protocol is improved in the DNSSEC (DNS Security Extensions; RFC 4033[2005]), which is yet to be deployed widely.

Using cryptography, DNSSEC provides (i) origin authentication of DNS data, (ii) data integrity, and (iii) authenticated denial of existence. The responses in DNSSEC are digitally signed. Note that DNSSEC does not provide confidentiality of data. Transaction Signatures (TSIG) authenticate communication between DNS servers by signing each transaction to ensure authenticity. Each DNS zone is signed to ensure its integrity and authenticity. There are two additional resource records called RRSIG and DNSKEY. The RRSIG record holds the signature of the RR, signed with the server's private key The DNSKEY record contains the public key of the zone that the receiver will use to verify the signature.

1.9.5. *Best Practices*

The only effective counter measure against DNS poisoning is to populate the `etc/hosts` file with DNS entries of important servers. This file should be updated through a secure procedure.

Note that DNS forgeries can be detected by noticing multiple DNS replies. While this cannot prevent a hijack, it can certainly be used in cleaning the cache.

The DNS server software should run on a highly secured machine and administered by highly trusted individuals. It should generate random transaction IDs, and source ports for each query. It should also rate-limit the queries. It should limit zone transfers to specified IP addresses.

1.9.6. *New Developments*

There are many new developments worth noting. Several groups are adding support for DNSSEC and IPv6.[33–35]

Cheung and Levitt[36] use formal specifications to characterize DNS clients and DNS name servers, and to define a security goal that a name server should only use DNS data that is consistent with data from authoritative name servers. A DNS wrapper examines the DNS query/response messages to detect violations, and cooperates with the corresponding authoritative name servers to diagnose those messages.

Cachin and Samar report on a design and implementation of a secure distributed name service, on the level of a DNS zone, that is able to provide fault tolerance and security even in the presence of a few corrupted name servers.[37] There are also new protocols with the same service functionality.[38]

1.10. Covert Channels

A covert channel is "any communication channel that can be exploited by a process to transfer information in a manner that violates the systems security policy." Covert channels do not modify the TCP/IP stack. The make legitimate use of the protocols. Obviously, covert channels need either specialized client and/or servers to inject and retrieve covert data.

Covert channels are the principle enablers in a distributed denial of service (DDoS) attack that causes a denial of service to legitimate machines.[39] A DDoS attacker covertly distributes (portions of) his attack tools over many machines spread across the Internet, and later triggers these intermediary machines into beginning the attack and remotely coordinates the attack.

Covert channels are possible in nearly all the protocols of the TCP/IP suite. Covert channels can be setup using the ID field of IP packets, IP checksums, TCP initial sequence numbers, or TCP timestamps. E.g., ICMP echo request packets should have an 8-byte header and a 56-byte payload. ICMP echo requests should not be carrying any data. However, such ICMP packets can be significantly larger, carrying covert data in their payloads.

A simple ICMP implementation is `covert-tcp`,[40] and the project Loki (`http://www.phrack.org/leecharch.php?p=49`) tunnels covert data in the data portion of ICMP-ECHO, and ICMP-ECHOREPLY messages. `stegtunnel` (`http://www.synacklabs.net/projects/stegtunnel/`) hides data in the initial SEQ numbers and IP IDs of TCP connections. Unlike covert-tcp, stegtunnel does not simply write raw packets out. It intercepts outbound and inbound traffic, and rewrites them.

Singh et al.[41] present a systematic solution to ICMP tunneling for covert channels. The BS thesis of Llamas[42] is a detailed analysis of covert channels in TCP/IP. Nagatou and Watanabe[43] describe detection of covert channels. Tumoian and Anikeev[44] describe a method of detecting covert channels embedded in the ISNs of TCP/IP. Murdoch and Lewis[45] point out that TCP/IP covert channels embedded into header fields such as the IP identifier, TCP initial sequence number (ISN) or the least significant bit of the TCP time stamp can be detected easily. They "describe reversible transforms that map block cipher output onto TCP ISNs, indistinguishable from those generated by Linux and OpenBSD." Bauer[46] describes new covert channels in HTTP.

1.11. Traffic Scrubbing

Scrubbing refers to forcing the TCP/IP traffic to obey all the rules of the RFCs. Reserved fields can be set to a random value; illegal combinations of flags are checked, and so on. Scrubbing is expected to be done not only at the originating hosts but also on the routers and especially in firewalls.

Scrubbing adds to the computational burden of the hosts. Because of hidden assumptions made by programs beyond the specifications of the RFCs, scrubbing may disrupt interoperability.

Watosn et al.[47,48] describe the design and implementation of a scrubber that supports downstream passive network-based intrusion detection systems and transparent fail-closed active network-based intrusion detection systems.

Smart and et al.[49] describe a TCP/IP stack fingerprint scrubber, a new tool to restrict a remote user's ability to determine the operating system of another host on the network.

1.12. Conclusion

The TCP/IP suite has many design weaknesses so far as security and privacy are concerned, all perhaps because in the era (1970s) when the development took place network attacks were unknown. The flaws present in many implementations exacerbate the problem. A number of these are due to the infamous buffer overflow which is preventable by better programming practices. However, considerable blame belong to the many ambiguous RFCs.

In this paper, we highlighted the technical details behind past attacks, and summarized current research.

Acknowledgements

This paper is titled to reflect the fact that it is strongly influenced by a paper by Bellovin.[1] All the RFCs mentioned are archived at `www.rfc-editor.org`. Vulnerability Notes and Technical Cyber Security Alerts are archived at `http://www.us-cert.gov/`.

References

1. S. M. Bellovin. A look back at "Security Problems in the TCP/IP Protocol Suite". In *ACSAC 2004*, pp. 229–249, Tucson, Arizona (Dec, 2004). IEEE Computer Society. ISBN 0-7695-2252-1.
2. S. M. Bellovin, Security problems in the TCP/IP protocol suite, *Computer Communications Review*. **19:2**, 32–48, (1989). URL `http://www.research.att.com/~smb/papers/ipext.pdf`.

3. I. Arce, Attack trends: More bang for the bug: An account of 2003's attack trends, *IEEE Security & Privacy.* **2**(1), 66–68 (Jan./Feb., 2004). ISSN 1540-7993.
4. B. Schneier, Attack trends: 2004 and 2005, *ACM Queue: Tomorrow's Computing Today.* **3**(5), 52–53 (June, 2005). ISSN 1542-7730.
5. R. Beverly. A robust classifier for passive TCP/IP finger printing. In eds. C. Barakat and I. Pratt, *Passive and Active Network Measurement, 5th International Workshop, PAM 2004, Antibes Juan-les-Pins, France, April 19-20, 2004, Proceedings*, vol. 3015, *Lecture Notes in Computer Science*, pp. 158–167. Springer, (2004). ISBN 3-540-21492-5.
6. S. Panjwani, S. Tan, K. M. Jarrin, and M. Cukier. An experimental evaluation to determine if port scans are precursors to an attack. In *DSN*, pp. 602–611. IEEE Computer Society, (2005). ISBN 0-7695-2282-3. URL `http://www.enre.umd.edu/faculty/cukier/81_cukier_m.pdf`.
7. V. Ramachandran and S. Nandi. Detecting ARP spoofing: An active technique. In eds. S. Jajodia and C. Mazumdar, *Information Systems Security, First International Conference, ICISS 2005*, vol. 3803, *Lecture Notes in Computer Science*, pp. 239–250. Springer, (2005). ISBN 3-540-30706-0.
8. Z. Trabelsi and K. Shuaib. Spoofed ARP packets detection in switched LAN networks. In eds. M. Malek, E. Fernández-Medina, and J. Hernando, *SECRYPT 2006: International Conference on Security and Cryptography*, pp. 40–47, Setbal, Portugal, (2006). INSTICC Press. ISBN 972-8865-63-5.
9. V. Goyal, R. Tripathy, C. Boyd, and J. M. Gonzlez. An efficient solution to the ARP cache poisoning problem. In *ACISP : Australasian conference on information security and privacy*, Lecture Notes in Computer Science, ISSN 0302-9743, pp. 40–51, Brisbane, Australia (July, 2005). Springer Berlin / Heidelberg. Volume 3574, ISBN 978-3-540-26547-4.
10. M. G. Gouda and C.-T. Huang, A secure address resolution protocol, *Computer Networks.* **41**(1), 57–71, (2003).
11. D. Bruschi, A. Ornaghi, and E. Rosti. S-ARP: a secure address resolution protocol. In *ACSAC*, pp. 66–75. IEEE Computer Society, (2003). ISBN 0-7695-2041-3.
12. O. Arkin, A remote active OS fingerprinting tool using ICMP, *;login: the USENIX Association newsletter.* **27**(2), 14–19 (Apr., 2002). ISSN 1044-6397. URL `http://www.usenix.org/publications/login/2002-04/pdfs/arkin.pdf`.
13. R. van den Berg and P. Dibowitz. Over-zealous security administrators are breaking the internet. In *Proceedings of the 16th USENIX System Administration Conference — LISA 2002*, pp. 213–218. USENIX, (2002). URL `http://www.usenix.org/publications/library/proceedings/lisa02/tech/full_papers/vanderberg/van_den_berg.pdf`.
14. S. J. Templeton and K. E. Levitt. Detecting spoofed packets. In *DARPA Information Survivability Conference and Exposition*, pp. 164– 175. IEEE Computer Society, (2003). ISBN 0-7695-1897-4. URL `http://seclab.cs.ucdavis.edu/papers/DetectingSpoofed-DISCEX.pdf`.
15. L. Wang, X. Zhao, D. Pei, R. Bush, D. Massey, and L. Zhang, Protecting

BGP routes to top-level DNS servers, *IEEE Trans. Parallel and Distrib. Systems.* **14**(9), 851–860, (2003).
16. F.-H. Hsu and T. cker Chiueh. CTCP: A transparent centralized TCP/IP architecture for network security. In *20th Annual Computer Security Applications Conference (ACSAC'04)*, pp. 335–344. IEEE Computer Society, (2004). ISBN 0-7695-2252-1.
17. S. Kent, C. Lynn, and K. Seo, Secure border gateway protocol (S-BGP), *IEEE Journal on Selected Areas in Communications.* **18**(4), 582–592 (Apr., 2000).
18. S. Kent, C. Lynn, J. Mikkelson, and K. Seo. Secure border gateway protocol (S-BGP) - real world performance and deployment issues (Feb. 24, 2000). URL `http://citeseer.ist.psu.edu/415133.html;http://www.isoc.org/ndss2000/proceedings/045.pdf`.
19. Y.-C. Hu, A. Perrig, and M. Sirbu. SPV: secure path vector routing for securing BGP. In *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pp. 179–192, (2004). URL `http://doi.acm.org/10.1145/1015467.1015488,http://www.ece.cmu.edu/~adrian/projects/spv.pdf`.
20. A. E. Hassan. Securing multimedia applications using: Secure udp. Technical report, (2000). URL `http://plg.uwaterloo.ca/~aeehassa/home/papers/crypto/secureUDP.htm`.
21. B. Harris and R. Hunt, TCP/IP security threats and attack methods, *Computer Communications.* **22**(10), 885–897, (1999).
22. L. Joncheray. A simple active attack against TCP. In *Proceedings of the 5th Symposium on UNIX Security*, pp. 7–20, Berkeley, CA, USA (June, 1995). USENIX Association. ISBN 1-880446-70-7. URL `http://www.cs.purdue.edu/homes/clay/papers/tcp-ip/Simple_Active_Attack_Against_TCP.ps`.
23. P. A. Watson. Slipping in the window: TCP reset attacks. In *CanSecWest 2004 Conference*, Vancouver, Canada (December, 2004). URL `www.terrorist.net`.
24. M. F. Arlitt and C. L. Williamson, An analysis of TCP reset behaviour on the internet, *SIGCOMM Computer Communication Review.* **35**(1), 37–44, (2005). ISSN 0146-4833.
25. A. Kuzmanovic and E. W. Knightly, Low-rate TCP-targeted denial of service attacks (the shrew vs. the mice and elephants), *Submitted to IEEE/ACM Transactions on Networking, March 2004.* (July 30. 2003). URL `http://www-ece.rice.edu/networks/papers/dos.ps.gz`.
26. M. Guirguis, A. Bestavros, and I. Matta. On the impact of low-rate attacks. Technical Report 2006-002, CS Department, Boston University (Feb. 6, 2006). URL `http://www.cs.bu.edu/techreports/2006-002-low-rate-attack-impact.ps.Z`.
27. H. Sun, J. C. S. Lui, and D. K. Y. Yau. Defending against low-rate TCP attacks: Dynamic detection and protection. In *ICNP*, pp. 196–205. IEEE Computer Society, (2004). ISBN 0-7695-2161-4. URL `http://csdl.computer.org/comp/proceedings/icnp/2004/2161/00/21610196abs.htm`.
28. H. Farhat. Protecting TCP services from denial of service attacks. In *LSAD*

'06: Proceedings of the 2006 SIGCOMM workshop on Large-scale attack defense, pp. 155–160, New York, NY, USA, (2006). ACM Press. ISBN 1-59593-571-1.

29. S. Savage, N. Cardwell, D. Wetherall, and T. Anderson, TCP congestion control with a misbehaving receiver, *Computer Communication Review.* **29** (5) (Oct., 1999). URL `papers/Savage99_TCP_misbehaving_rec.pdf`.
30. W. Chen and D.-Y. Yeung. Defending against TCP SYN flooding attacks under different types of IP spoofing. In *International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies (ICNICONSMCL '06)*, p. 38, Morne, Mauritius (Apr., 2006). IEEE Computer Society. ISBN 0-7695-2552-0.
31. D. J. Bernstein. *SYN Cookies*, (1997). URL `http://cr.yp.to/syncookies.html`.
32. V. Ramasubramanian and E. G. Sirer. Perils of transitive trust in the domain name system. In *Proceedings of Internet Measurement Conference (IMC)*, Berkeley, California (October, 2005).
33. B. Manning. Adventures in DNS. In *Proceedings of the General Track: 2002 USENIX Annual Technical Conference, June 10–15, 2002, Monterey, California, USA*. USENIX, (2002). ISBN 1-880446-00-6. Unpublished invited talk.
34. J. L. S. Damas. A review of IPv6 in DNS. protocol and implementations. In *SAINT Workshops*, pp. 58–61. IEEE Computer Society, (2005). ISBN 0-7695-2263-7.
35. R. Curtmola, A. D. Sorbo, and G. Ateniese. On the performance and analysis of DNS security extensions. In eds. Y. Desmedt, H. Wang, Y. Mu, and Y. Li, *Cryptology and Network Security, 4th International Conference, CANS 2005*, vol. 3810, *Lecture Notes in Computer Science*, pp. 288–303. Springer, (2005). ISBN 3-540-30849-0.
36. S. Cheung and K. N. Levitt. A formal-specification based approach for protecting the domain (July 29, 2002). URL `http://seclab.cs.ucdavis.edu/papers/Cheung_LevittDNS.pdf`.
37. C. Cachin and A. Samar. Secure distributed DNS. In *2004 International Conference on Dependable Systems and Networks (DSN'04)*, pp. 423–432. IEEE Computer Society, (2004). ISBN 0-7695-2052-9.
38. V. Ramasubramanian and E. G. Sirer. The design and implementation of a next generation name service for the internet. In eds. R. Yavatkar, E. W. Zegura, and J. Rexford, *Proceedings of the ACM SIGCOMM 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pp. 331–342. ACM, (2004). ISBN 1-58113-862-8. URL `http://www.cs.cornell.edu/People/egs/papers/codons-sigcomm.pdf`.
39. T. Sohn, T. Noh, and J. Moon. Support vector machine based ICMP covert channel attack detection. In eds. V. Gorodetsky, L. J. Popyack, and V. A. Skormin, *Computer Network Security, Second International Workshop on Mathematical Methods, Models, and Architectures for Computer Network Security, MMM-ACNS 2003, St. Petersburg, Russia, September 21-23, 2003,*

Proceedings, vol. 2776, *Lecture Notes in Computer Science*, pp. 461–464. Springer, (2003). ISBN 3-540-40797-9.

40. C. H. Rowland, Covert channels in the TCP/IP protocol suite, *First Monday.* **2**(5), (1997). URL `http://firstmonday.org/issues/issue2_5/rowland/index.html`.
41. A. Singh, O. Nordström, C. Lu, and A. L. M. dos Santos. Malicious ICMP tunneling: Defense against the vulnerability. In eds. R. Safavi-Naini and J. Seberry, *ACISP: Information Security and Privacy, 8th Australasian Conference, ACISP 2003, Wollongong, Australia, July 9-11, 2003, Proceedings*, vol. 2727, *Lecture Notes in Computer Science*, pp. 226–235. Springer, (2003). ISBN 3-540-40515-1. URL `http://gray-world.net/papers/icmp-paper.ps`.
42. D. Llamas. Covert channel analysis and data hiding in tcp/ip. Master's thesis, Napier University, Edinburgh, Scotland, (2004). URL `http://www.buchananweb.co.uk/PROJECTS/2004/david_llamas.pdf`.
43. N. Nagatou and T. Watanabe. Run-time detection of covert channels. In *ARES*, pp. 577–584. IEEE Computer Society, (2006).
44. E. Tumoian and M. Anikeev. Network based detection of passive covert channels in TCP/IP. In *LCN*, pp. 802–809. IEEE Computer Society, (2005). ISBN 0-7695-2421-4.
45. S. J. Murdoch and S. Lewis. Embedding covert channels into TCP/IP. In eds. M. Barni, J. Herrera-Joancomartí, S. Katzenbeisser, and F. Pérez-González, *Information Hiding, 7th International Workshop*, vol. 3727, *Lecture Notes in Computer Science*, pp. 247–261. Springer, (2005). ISBN 3-540-29039-7.
46. M. Bauer. New covert channels in HTTP. In *Proceedings of the 2003 ACM Workshop on Privacy in the Electronic Society* (Apr. 26, 2003).
47. G. R. Malan, D. Watson, F. Jahanian, and P. Howell. Transport and application protocol scrubbing. In *INFOCOM*, pp. 1381–1390 (Dec. 05, 2000). URL `malan00transport.pdf`.
48. D. Watson, M. Smart, G. R. Malan, and F. Jahanian, Protocol scrubbing: network security through transparent flow modification, *IEEE/ACM Transactions on Networking.* **12**(2), 261–273 (Apr., 2004). ISSN 1063-6692.
49. M. Smart, G. R. Malan, and F. Jahanian. Defeating TCP/IP stack fingerprinting. In ed. USENIX, *Proceedings of the Ninth USENIX Security Symposium*, Denver, Colorado (August, 2000). USENIX. ISBN 1-880446-18-9. URL `http://www.usenix.org/publications/library/proceedings/sec2000/smart.html`.