Chapter 11

Intrusion Detection and Network-Based Attacks

The official version of this book is available at
https://www.springer.com/gp/book/9783030336486

ISBN: 978-3-030-33648-6 (hardcopy), 978-3-030-33649-3 (eBook)

version: 25 Sept 2019

Chapter 11

Intrusion Detection and Network-Based Attacks

This second of two chapters on network security complements Chapter 10's treatment of firewalls and tunnels. Here we discuss intrusion detection and various tools for network monitoring (packet sniffing) and vulnerability assessment, followed by denial of service and other network-based attacks that exploit standard TCP/IP network or Ethernet protocols. We consider TCP session hijacking, and two categories of address resolution attacks—DNS-based attacks, which facilitate pharming, and attacks involving Address Resolution Protocol (ARP) spoofing. Such network-based attacks are carried out regularly in practice. The best defense to stop many of them is encryption of communication sessions; building a true appreciation for this is alone strong motivation for learning at least the high-level technical details of these attacks. In addition, understanding the underlying principles that enable attacks is important to avoid repeating design errors in future networks and emerging Internet of Things (IoT) protocols, as experience tells us that variations of these same attacks are almost certain to reappear.

11.1 Intrusion detection: introduction

Firewalls provide coarse perimeter shields designed to block the bulk of malicious traffic. Intrusion detection systems and related defenses—both host-based and network-based—provide eyes to identify what gets through. Dedicated human resources are required to manage and monitor these systems, and to explore alarms raised. Early research was driven by the desire to automate human analysis of audit trails, to detect unauthorized use of government systems, and to detect behavior non-conformant with IT security policies.

BASIC TERMS. An *intrusion* or *incident* is an event on a host or network that violates security policy, or is an imminent threat to put a system in an unauthorized state. *Intrusion detection* is the process of monitoring and analyzing system events, to identify and report such intrusions. An *intrusion detection system* (IDS) automates the process, and includes monitoring events, logging related data, analysis, and means to report events requiring

human attention. An intrusion may involve an unauthorized or rogue user (*intruder*), process, program, command, action, data at rest (in storage) or in flight (as a network packet). Not all intrusions are deliberate attacks; consider a connection error by an external party.

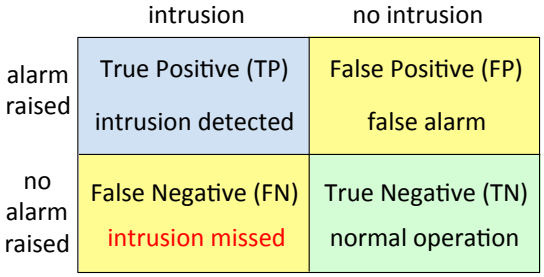
DETECTION VS. PREVENTION. An IDS *detects* intrusions and other adverse events, either in progress, or after the fact. The basis for an IDS is a monitoring system that collects evidence facilitating detection and supporting *forensic analysis*.[1] In practice, sorting out what has actually happened often requires ad hoc analysis by human experts, and exploration may be open-ended for new attacks; such systems are not pragmatic for typical users. An *intrusion prevention system* (IPS), beyond passive monitoring, includes active responses, e.g., stopping in-progress violations or altering network configurations. An IPS augmenting a firewall[2] may alter packets, strip out malware, or send TCP resets to terminate connections; an in-host IPS may terminate processes. An IPS may be configured to operate passively as an IDS. The two acronyms are often interchanged, but a true IPS requires automated real-time responses, and can mitigate a subset of known attacks.

ARCHITECTURAL TYPES. An IDS involves a means to collect *events* from an event source, and components for event analysis, reporting (e.g., logging to a console or by email/text messages), and response (in an IPS). Two complementary IDS categories, based on where *sensors* collect event streams, are *network-based* IDSs (NIDSs) and *host-based* IDSs (HIDSs). NIDS events are derived from packets obtained at a strategic vantage point, e.g., a network gateway, or a LAN (local area network) switch; Section 11.3 discusses packet sniffing. HIDS events may be derived from kernel-generated operations and audit records, application logs (noting userid), filesystem changes (file integrity checks, file permissions, file accesses), and system call monitoring (exercise, page 315); plus specific to the host, network accesses, incoming/outgoing packet contents, and status changes in network interfaces (ports open, services running). Resource use patterns (CPU time, disk space) may reveal suspicious processes. Independent HIDS tools protect only a single host, and detect intruders only thereon. HIDS data must be pooled (e.g., centrally) to provide views beyond a single host. A NIDS provides network-wide views.

EVENT OUTCOMES. On processing an event, an IDS analysis engine may or may not raise an alarm, and the event may or may not be an intrusion. This gives us four cases (Fig. 11.1). Low error rates (the two falses) are desired. High false positive rates, a common problem with anomaly-based systems (Section 11.2), severely limit usability of an IDS; false positives distract human analysts. High false negative rates are a security failure, and thus dangerous (missed intrusions may lead to unknown damage). From a classification view, the *intrusion detection problem* is to determine whether an event is from a distribution of events of intruder behavior, or from a legitimate user distribution. Some IDS approaches offer a tradeoff between false positives and negatives similar to that for biometric authentication, where the task is to classify as intruder (impersonator) or legitimate user. (Recall Chapter 3's two-hump graph of overlapping distributions; the related tradeoff here is shown in Fig. 11.2b on page 314, with a rough analogy that FPR

[1] This follows principle P14 (EVIDENCE-PRODUCTION).

[2] While IPS and firewall functionality are commonly combined, we keep them separate pedagogically.



False positive rate	$FPR = \frac{FP}{(FP+TN)}$
True negative rate	$TNR = 1 - FPR$
False negative rate	$FNR = 1 - TPR$
True positive rate	$TPR = \frac{TP}{(TP+FN)}$
Alarm precision	$AP = \frac{TP}{(TP+FP)}$

Figure 11.1: IDS event outcomes (left) and metrics (right). FP and FN (yellow) are the classification errors. TPR is also called the *detection rate*.

and FNR here map to, respectively, False Reject Rate and False Accept Rate in biometric authentication.)

Example *(Error rates and base rates).* Consider the following situation.[3] There is a disease X, and a test that screens for it. Given 100 non-diseased people, the test on average flags one subject as diseased—so one false positive, and FPR $= 1/(1+99) = 0.01 = 1\%$. Thus TNR $= 1 -$ FPR $= 99\%$. And also, given 100 diseased people, the test on average finds 98 subjects diseased—so two false negatives, and FNR $= 2/(98+2) = 0.02 = 2\%$ or equivalently, TPR $= 98/(98+2) = 0.98 = 98\%$. (Such a test might be marketed as "98% accurate" or "99% accurate", but doing so without explaining the metric used will confuse experts and non-experts alike; we will return to this point.) Now suppose also the incidence of our disease X across the population is 1 in 100,000; in a random set of 100,000 people, we then expect 1 to be diseased. If the screening test is applied to this set, we can expect (from the 1% FPR) to find 1% of 99,999, or 1000 false positives. In all likelihood, the one actually diseased person will also test positive (due to the 98% TPR).

Of course, what the doctors see as an outcome is 1001 people flagged as "may have disease X", so 1000 out of 1001 are false alarms. We might now feel misled by the earlier metrics alone, and by any suggestion the test was 98% or 99% "accurate" (alas, math confuses some, language confuses others). This motivates a further metric, *alarm precision* (AP), the ratio of correctly raised alarms to total alarms (true positives to total positives):

$$\text{AP} = \text{TP}/(\text{TP}+\text{FP}) = 1/(1+1000) = 1/1001 \approx 0.1\%.$$

Positioning this in reverse as *alarm imprecision*, AIP=FP/(TP+FP) $= 1000/(1+1000) \approx 0.999$. We now see that 99.9% (!) of alarms raised are false alarms.[4] A high ratio (high AIP) and a high absolute number of false alarms are both problems for an IDS (below).

Explanation of above example. What fools us in this example is overlooking the low *base rate* of incidence of the disease across the population. For an IDS, this corresponds to the ratio of intrusion events to total events. We move to an IDS setting for our explanation: "diseased" becomes an intrusion event, and "positive test result" is now an IDS alarm raised. Let's revisit the above example algebraically, using approximations that apply to that example—where the number of false positives vastly exceeds the number

[3] To use the Fig. 11.1 ratios, alarms are positive disease tests, and intrusions are incidents of disease.

[4] FPR measures false positives over all events involving no illness (intrusions), while AIP $= 1000/1001$ measures false alarms over all events that involve positive tests (alarms). To avoid confusing these, it may help to note that in Fig. 11.1, AP is a sum across row 1, while FPR and FNR are each sums within a column.

of true positives, i.e., TP $\ll$ FP (a difficult situation for an IDS). Assume a set of n events. It consists of n_I intrusion and n_N non-intrusion events (n is known, but not n_I, n_N). Now:

$$n = n_I + n_N, \qquad \text{TP} = \text{TPR} \cdot n_I, \qquad \text{FP} = \text{FPR} \cdot n_N$$

The last two equations are just the definitions. We expect a (useful) IDS to detect a high proportion of intrusions, and so to a crude approximation, TPR ≈ 1 implying TP $\approx n_I$. A tiny base rate of intrusions (as also expected in a fielded IDS) means $n_I \ll n_N$ and $n \approx n_N$. From TP $\ll$ FP we get TP + FP $\approx$ FP. Now substituting in these approximations,

$$\text{AP} = \text{TP}/(\text{TP}+\text{FP}) \;\approx\; \text{TP}/\text{FP} \;\approx\; n_I/(\text{FPR} \cdot n_N) \;\approx\; (n_I/n)/\text{FPR}$$

This approximation for AP now captures the parameters that dominated the computed value of AP in our example: the base rate n_I/n of incidence, and FPR. As a summary: *alarm precision is governed by both the base rate of incidence and the false positive rate*.

IDS IMPLICATION OF BASE RATE OF INTRUSIONS. Our example illustrates what psychologists call the *base rate fallacy*: people tend to ignore the base rate of incidence of events when intuitively solving problems involving conditional probabilities. Most computer systems have a very low base rate of intrusions.[5] Given a huge number n_N of non-intrusion events, even a very low (non-zero) false positive rate can yield a large number of false positives since FP = FPR $\cdot\, n_N$. It is challenging to keep both FPR and FNR acceptably low—decreasing one of these error rates increases the other in some IDS approaches. For an enterprise IDS, exploring false positives not only steals experts' time better spent on sorting out true positives, but any significant level of alarm imprecision, even 1 in 2 alarms being a false positive, risks complacency and training staff to ignore alarms altogether. And if there are 100 alarms per day, whether true or false positives, the problem may become lack of investigative resources. The tolerance for false positives is also extremely low in end-user systems—consider anti-virus software, a type of IPS.

Exercise (Classification semantics). (a) From Fig. 11.1, describe in words semantically what is captured by the false positive (FPR) and false negative rate (FNR) metrics.
(b) Consider the notation: I (intrusion), $\neg I$ (no intrusion), $\mathcal{A}$ (alarm), $\neg\mathcal{A}$ (no alarm). Using this, define FPR, FNR, TPR and TNR each as expressions of the form: prob$(X|Y)$, meaning "the probability of X given (i.e., in the case of) Y", where Y is I or $\neg I$.
(c) When running an IDS, the main observable is an alarm being raised. The probabilities of interest then are (with higher values more desirable): prob$(I|\mathcal{A})$ (the *Bayesian detection rate*), and prob$(\neg I|\neg\mathcal{A})$. Describe in words what these expressions mean.
(d) By Venn diagram, show the four sets of events (I and $\mathcal{A}$), (I and $\neg\mathcal{A}$), ($\neg I$ and $\mathcal{A}$), ($\neg I$ and $\neg\mathcal{A}$), in the case of far more false positives than true positives (hint: [10, p. 10]).

11.2 Intrusion detection: methodological approaches

In practice, IDSs use a combination of methods spanning different approaches, and individual methods may overlap in approach categories themselves. We distinguish three philosophical approaches to intrusion detection: signature-based, specification-based and

[5] Note there is a crucial difference from disease rates: for computer intrusions we lack historical statistics for the base rates, and widespread instrumentation for reliable measurement is beyond present capabilities.

IDS approach	Alarm when...	Pros, cons, notes
signature-based (expert defines malicious patterns)	events match known-bad patterns	signatures built from known attacks; fast, accurate (fewer false positives); detects only already-known attacks
specification-based (expert defines allowed actions)	events deviate from per-application specifications of legitimate actions	manually developed spec of allowed; can detect new attacks; no alarm on newly seen allowed event; specs are protocol or program-specific
anomaly-based (learning-based profile of normal)	events deviate from profiles of normal	need training period to build profiles; can detect new attacks; false alarms (abnormal may be benign); accuracy depends on features profiled

Table 11.1: IDS methodologies. Signature-based approaches use expert-built patterns (manual blacklists). Specification approaches use expert-built specs (manual whitelists). Anomaly approaches define "normal" behavior from training data (empirical whitelists).

anomaly-based approaches (Table 11.1). Figure 11.2 depicts relationships between these.

1) SIGNATURE-BASED. These approaches examine events for predefined *attack signatures*—pattern descriptors of known-bad behavior. Matches trigger alarms. Simple patterns such as raw byte sequences, and condition lists (e.g., fields to match in packets), may be combined in rules, and are similar to simple anti-virus and packet-filter techniques. Advantages are speed and accuracy. More advanced patterns involve regular expressions. Signature generation and update is a continuous task, relies on known intrusions (attacks), and reflects only these. (Many IPSs are configured to receive automated vendor signature updates.) Variants called *behavior-based* attack signature approaches generalize pattern descriptors beyond attack-instance implementation details by looking for attack side effects or outcomes that provide indirect evidence of attacks.

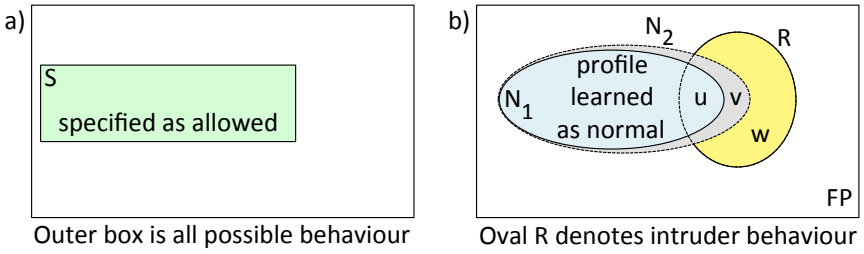


Figure 11.2: Visual model of IDS approaches. a) Specification-based; activity outside the shaded region raises an alarm. b) Anomaly-based and signature-based. Anomaly-based methods alert on detecting activity outside N_1; this may be a true positive if in v or w, but a false positive if in FP. To reduce false positives, parameters or thresholds may be tuned to recognize a larger area (N_2) as normal, but this increases false negatives, as intruder activity in v will no longer trigger an alarm. For signature-based approaches, attack signatures, from a subset of w, are created from known attacks.

2) SPECIFICATION-BASED. For each individual protocol or application selected, an expert creates a *specification* defining allowed protocol or application behaviors (modeled as benign), and a tool to verify conformance; non-conforming events raise alarms. The approach can detect previously unseen attacks, without false alarms on unusual but legitimate behaviors—thus offering a main advantage of anomaly-based approaches, without the main disadvantage. The specifications and related tools are protocol-specific and time-consuming to build, but universal (apply for all users). If a specification's characterization of allowed behavior is too lenient, the false negative rate grows (missing attacks).

3) ANOMALY-BASED. Profiles of normal (expected) activity are built over a *training period*, on a per-network or per-user basis, based on observed usage or from audit trails or logs. In operation, analysis of the event stream (current system behavior) aims to identify deviations from profiles. *Profiles* are based on selected features (characteristics). Simple features may be single discrete operation counts over fixed time periods, e.g., numbers of failed login attempts, accesses to a given file, or total files deleted. Exceeding a threshold for a single feature, or a collection of features, may be deemed anomalous. Other feature examples are sets of files accessed, file accesses within a time window, sets of commands used, CPU resources consumed, and session duration. For profiles composed of extensive feature sets, statistical tests can determine whether deviations are statistically significant at specified confidence levels (vs. being due to chance variation); thresholds and confidence levels are tunable parameters. Machine learning is commonly used.

‡**ANOMALY-BASED APPROACHES: PROS, CONS.** As a strength, previously unseen attacks may be detected. As a limitation, authorized users may behave abnormally, leading to high false positive rates. We note three other challenges:

- *Feature selection*. Selecting features is difficult; efficacy depends highly on the features of system behavior chosen for profiles (features implicitly define models).
- *Intruder-free training*. It is important, but difficult, to ensure that malicious activity is absent during training (lest it be "baselined" into profiles).
- *Session creep*. If profiles are designed to self-adjust over time to accommodate evolution in legitimate behavior, intruders might be able to exploit this by slowly embedding their own malicious behavior into the baseline.

‡**Exercise** (Sequences of system calls). One host-based IDS technique involves using sequences of Unix system calls to define normal (they approximate program control flow); anomalous sequences raise alerts. Summarize the technical details (hint: [34, 41]).

Exercise (Snort: signatured-based NIDS). Snort is a widely used open-source NIDS tool. Simple *Snort rules* identify header fields (e.g., port, address, flags) and byte-patterns (attack signatures) to match packet content. a) Summarize the main goals, architecture and features of Snort (hint: [72]). b) Explain a Snort rule, using a non-trivial example.

Exercise (Bro: NIDS/monitor). Bro (now Zeek) is open-source, and has separate protocol analysis and *policy script* components, leaving the IDS approach open. It supports signature-based detection. (A tool, `snort2bro`, built in 2003, automatically converts Snort rules to Bro signatures. Maintenance of this conversion script was discontinued in 2008, as it no longer supported rules leveraging newer Snort tools.) Data stored by Bro can

support anomaly detection. Protocol-specific analyzers facilitate application-level analysis that incorporates state or context. Summarize the main goals, architecture and features of Bro, including how its design goal of separating mechanism from policy supports all IDS approaches (hint: [64, 78]).

11.3 Sniffers, reconnaissance scanners, vulnerability scanners

We now consider a collection of tools that have both white-hat and black-hat uses.

PACKET SNIFFING. For NIDS, various tools allow packet capture and file storage/retrieval. Tools must process packets at line-speed. For an IPS, real-time analysis is also necessary; for an IDS, rapid processing remains important, but it need not be inline. Independent of intrusion detection *per se*, such packet collection and processing tools are of interest for (passive) *network monitoring* to allow insight into activities on a network, traffic patterns and usage. Logging traffic-related details may support network management, and in the case of incidents, later forensic investigations, loss evaluation, and recovery.[6] Sniffing tools and collection methods are also used by attackers, and this is a primary motivation for traffic encryption (e.g., via SSH, TLS, and IPsec-based VPNs).

HUBS AND SWITCHES. Ethernet is the dominant communication protocol for a wired LAN (local area network). LAN hosts are connected and exchange information through a hub or a switch. A *hub* broadcasts (Fig. 11.3): it relays all packets (Ethernet frames) received from one LAN host, over the physical interfaces to each other LAN host. On a hubbed LAN, by putting its network interface card (NIC) in *promiscuous mode*, a LAN host can passively collect all frames passing the interface of its NIC—those addressed to its own MAC address, and all others. In contrast, for a *switch*, information received from one host will be sent only over the physical interface connecting to the host corresponding to the destination MAC address in the Ethernet frame.[7] Switched LANs are more common, and putting a NIC card in promiscuous mode does not give visibility to all LAN traffic, but this security advantage can be compromised by ARP spoofing (Section 11.5).

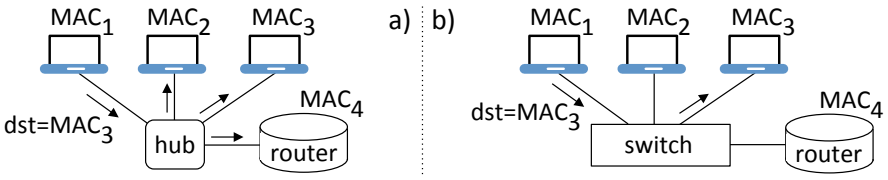


Figure 11.3: LAN hub vs. LAN switch. A hub broadcasts. A switch isolates.

‡**MONITORING SUPPORT: NETWORK TAPS AND SPANS.** Passive monitoring of network data is supported by hardware enabling access to packets. We note two common means. A *SPAN port* (switched port analyzer) on a switch, also called a *port mirror*, is a designated port configured to duplicate traffic from other ports; enterprise-class switches

[6] Thus independent of an IDS, network monitoring supports principle P14 (EVIDENCE-PRODUCTION).

[7] A switch learns the mapping of MAC address to physical interface by passively monitoring frames; mappings can also be manually configured.

commonly support configuration of SPAN ports. A *tap* (test access port) is a dedicated device to facilitate passive monitoring; e.g., a four-port Ethernet tap might use two ports to connect between a router and firewall, and the other two to access traffic for monitoring. SPANs and taps are *inline* in the sense of being in the path packets follow, but do not have general processors as needed for intrusion prevention functionality. For that, dedicated inline devices (*filtering bridges*) are used. These run packet collection tools and store, analyze and potentially alter traffic. Note that in order to *prevent* intrusions, an IPS device must be inline; for detection only, an IDS device may monitor a passive tap or SPAN.

VULNERABILITY ASSESSMENT TOOLS. *Vulnerability assessment* tools may be viewed as a subset of intrusion detection tools—but rather than defending, you now seek weaknesses in your own hosts, largely in three categories: known-vulnerable services, typical configuration errors, and weak default configurations still in use. Results may call for software updates, configuration changes, and changing default passwords.[8] Both host-based tools and network-based tools are used, the latter falling into three categories:

1. reconnaissance tools (below);
2. vulnerability assessment tools (*vulnerability scanners*); and
3. *penetration testing* tools[9] (pre-authorized) or *exploitation toolkits* (used by black-hats).

Authorized parties use these tools for self-evaluation, to provide awareness of network-accessible vulnerabilities, and of services offered to check compliance with security policy. Vulnerability scanners produce comprehensive reports about the systems assessed. In contrast, penetration/exploitation frameworks aim to tangibly exploit live systems, including installation of a payload; they can test potential vulnerabilities flagged by vulnerability assessment tools. In self-assessment, benign payloads are used, albeit sufficient to prove that a flagged vulnerability is not a false positive—false positives need no repair, while true positives do, and distinguishing the two is a major legitimate use of penetration tests. On the other hand, an attacker using an exploitation toolkit seeks actual compromise, e.g., through a single exploit providing a desired level of access (e.g., root).

LIMITATIONS, CAUTIONS. Vulnerability assessments give status at a fixed point in time, and with respect (only) to exploits known to the tools. The dual white/black-hat use of penetration/exploitation frameworks creates some uneasiness. Such tools improve the speed and accuracy of live-testing for both authorized and unauthorized parties. Scans executed as *credentialed* (via an authorized user or administrative account) allow significantly greater detail. Exploit modules are commonly available via Metasploit (page 320). Is it ethical to release exploit modules? The consensus is that attackers already have and use these, so legitimate parties should as well, so as not to fall even farther behind. A *responsible disclosure* approach recommends first providing details to product vendors, to allow patches to be made available; however, this process is complicated by the requirement of vendor cooperation and timely response, as well as ethical questions including whether vulnerable users should be notified even before patches are available.

[8] *Proactive password crackers*, which use a system's password hash file to guess its own users' passwords (asking users to change those so guessed), were early instances of vulnerability assessment tools.

[9] General context on penetration testing is also given in Chapter 1.

In all cases, use of vulnerability assessment and exploitation tools on hosts or networks other than your own, without prior written permission, risks legal and potential criminal liability.

PORT SCANNING AND OS FINGERPRINTING. *Network-based reconnaissance* is a common precursor to attack. Sending *probes* (e.g., TCP connection requests) to various addresses identifies hosts and ports running services. A port can be *open* (daemon waiting), *closed* (no service offered), or *blocked* (denied by a perimeter access control device). *Port scanning* is any means to identify open ports on a target host or network. An IPS that detects port scanning may coordinate with perimeter defenses to block (blacklist) source addresses identified as scanners. Scanning one's own machines allows an inventory of hosts, and a cross-check that services offered are known and authorized by policy. A common feature in network scanners is *remote OS fingerprinting*: identifying a remote machine's OS, and its version, with high confidence. Uses include, e.g., for defenders, informing about needed software updates; for penetration testers and attackers, it allows selection of OS-dependent exploits of identified services at known IP addresses.

Exercise (Scan detection). Describe two specific methods to detect simple port scanning (hint: [44, Sect. 2]).

Example *(OS fingerprinting).* OS fingerprinting tools can be passive (e.g., p0f) or active (e.g., Xprobe2, Nmap). Methods are called *stack querying* when they rely on network stack implementation details. Active methods may, e.g., send TCP connection requests with non-standard header flag combinations; responses distinguish specific OS releases due to idiosyncrasies in vendor software. p0f, which originates no new traffic itself, inspects both TCP headers and application-level HTTP messages; it is useful on systems that block Nmap probes. Xprobe2 makes use of ICMP datagram responses.

Example *(Reconnaissance: Nmap).* Dual-use tools are those used by both white-hats and black-hats. For example, Nmap (Network mapper) is an open-source network scanner with a point-and-click graphical interface, Zenmap. Among other features, it supports:

- finding IP addresses of live hosts within a target network;
- OS classification of each live host (OS fingerprinting, above);
- identifying open ports on each live host (port scanning);
- *version detection* (for open ports, identifying the service listening, and version); and
- *network mapping* (building a network topology—hosts and how they are connected).

Version detection may be as simple as completing a TCP handshake and looking at a service banner if presented (indicating service type/version); if no banner is offered, this information may be possible to deduce by sending suitable probes. For self-assessment, the above features allow an inventory of enterprise services (implying related exposures), useful in carrying out security audits. While this may provide awareness about vulnerabilities, actually testing for (or actively exploiting) vulnerabilities is commonly done using dedicated penetration testing or exploitation tools designed to do so more efficiently.

Example *(Vulnerability scanner: Nessus).* Nessus is a widely used remote vulnerability scanner—again dual use, and in this case proprietary (free for non-commercial use). It has discovery capabilities, but the focus is vulnerability assessment. Its modular

architecture supports programs (plugins) that can test for individual vulnerabilities; a vast library of such plugins exists for CVEs (Common Vulnerability Exposures). Configuring Nessus to run a scan includes specifying targets (IP addresses or network ranges), port ranges and types of port scans (similar to Nmap), and which plugins to run. A summary report is provided. Some plugins, e.g., denial of service tests, may crash a target (in safe mode, such tests are bypassed). While this may be the goal of an attacker, such modules also allow testing a system prior to launching a service or releasing a software product. Tools like Nessus have capabilities similar to an *auto-rooter* (Section 7.3).

‡**Exercise** (Password guessing: Nessus). Some Nessus plugin modules test services for weak passwords via password-guessing attacks. If these services are on a live system, and the system locks out accounts after n incorrect guesses (e.g., $n = 5$ or 10) within a short period of time, how will running these modules affect users of those accounts?

‡**Example** *(Packet capture utilities).* Two popular general-purpose tools for packet capture and processing are the `tcpdump` utility,[10] and somewhat similar but with a graphical front-end and deeper protocol analysis, the open-source Wireshark.[11] Both rely on a standard packet capture library, e.g., `libpcap` on Unix, implementing the `pcap` interface. `libpcap` in turn relies on a BSD packet filter (BPF) implementation supporting user-specified filtering criteria (e.g., ports of interest or ICMP message types), allowing unwanted packets to be efficiently dropped in the kernel process itself. Functionally, `tcpdump` reads packets from a network interface card in promiscuous mode (page 316). Packets can be written to file, for later processing by `tcpdump` or third-party tools supporting `pcap` file formats. Security-focused network traffic analyzers that use `libpcap` directly (rather than through `tcpdump`), like Snort and Bro, augment packet capture and processing with their own specialized monitoring and intrusion detection functionality.

‡ATTACKING THE SNIFFER. On some systems, configuring packet capture tools requires running as root; some require maintaining root privileges. In any case, packet sniffers themselves present attractive new attack surface: even if all listening ports are closed, the tool receives packets and is thus itself subject to exploitation (e.g., by buffer overflow flaws). Software security is thus especially critical for packet capture tools.

Exercise (network statistics). The `netstat` command line utility, available in major OSs, provides information on current TCP and UDP listening ports, in-use connections, and other statistics. a) Read the manual page on your local OS (from a Unix command line: `man netstat`), experiment to confirm, and list the command syntax to get the PID (process ID) associated with a given connection. b) Use `netstat` with suitable options to get information on all open UDP and TCP ports on your current host; provide a summary.

‡**Exercise** (COPS and SATAN). Summarize the architecture and functionality of each of the following vulnerability scanners. a) COPS (Computerized Oracle and Password System), a scanner released in 1990 for Unix systems (hint: [30]). b) SATAN (Security Analysis Tool for Auditing Networks), a scanner released in 1993 for networked computers (hint: [31]). Discuss also why the release of SATAN was controversial.

[10] `tcpdump` as ported to Windows is `WinDump`.

[11] Wireshark was formerly Ethereal, with command-line version `TEthereal`.

‡**Exercise** (Metasploit: vulnerability exploitation framework). The well-known open-source Metasploit framework, a toolkit and application providing command line, console, and browser point-and-click interfaces, is used by systems administrators for penetration testing, and by black-hats for (unauthorized) exploitation of network services. a) Explain how its modular structure accommodates different attack vectors (exploits) independent of attack payloads. b) Summarize procedural use (the steps required by a user). c) Outline the main types of payloads it offers, including an explanation of Meterpreter and its unique advantages as a particularly stealthy payload. (Hint: [46], [76].)

‡**Exercise** (Netcat: network utility and attack tool). Netcat (`nc`) is a white-hat/black-hat tool that supports vulnerability scanning, port scanning, file transfer, and installing backdoors among other things. Its power is eye-opening to non-experts. In less than two pages, summarize its defensive and offensive functionality (hint: [76, Ch. 8]).

‡**Exercise** (Complementary technologies). Summarize how the following technologies are complementary to IDSs: a) firewalls and screening routers (hint: [75, p. 8-6]); and b) anti-virus/anti-malware defenses (hint: [75, p. 8-5]).

‡**Exercise** (Beyond NIDS, HIDS). Two further IDS categories are *wireless-based IDS*, and *network behavior and analysis systems* (NBAs). Summarize how these differ from NIDS and HIDS, and the main threats each addresses (hint: [75]).

11.4 Denial of service attacks

Many security defenses and designs in use are motivated by attacks that exploit specific details of network communications protocols. For this reason and general awareness, Sections 11.4–11.6 provide a basic overview of selected *network-based attacks*.

DoS. *Denial of service* (DoS) attacks are those that deny legitimate users the availability of resources and services, by intentional acts that severely degrade performance or cause outright failure. We highlight two classes of DoS attacks:

I. One class exploits latent implementation flaws (software vulnerabilities).

II. A second exhausts resources (bandwidth, CPU, main memory, disk), by *flooding* to overwhelm by traffic volume, or by consuming fixed resources (SYN floods below), or requesting resource-intensive operations (e.g., generation of asymmetric key pairs).

A flooding attack may be possible even by a single attack machine, limited only by its CPU speed and link capacity, continually sending packets to a target. Such high packet-rate attacks may exploit link speed asymmetries, i.e., use hosts with high-bandwidth connections to attack targets with lower-bandwidth connections.

DoS motives. DoS motives vary (cf. adversary model, Section 1.4), including:

1) direct financial gain via extortion, as per ransomware;
2) commercial competitive gain, by disrupting competitor sales or reputation;
3) ideological or social activism;
4) information warfare, typically by nation states;
5) hacker experimentation, e.g., to boost ego or for peer recognition; and

6) vengeance, towards those associated with the resources under attack.

DDoS. A *distributed denial of service* (DDoS) flooding attack is one that uses large numbers of devices across a wide array of addresses (e.g., using a botnet). See Fig. 11.4.

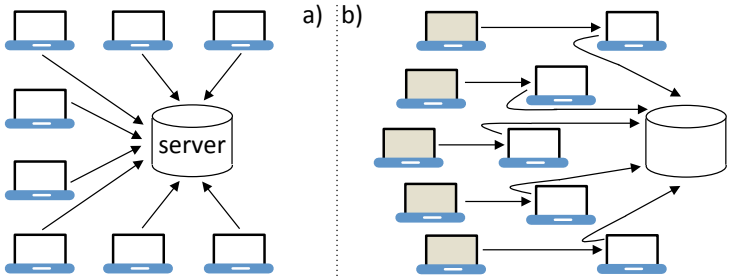


Figure 11.4: DDoS. a) The individual hosts (zombies) flooding the server are controlled by a botnet master directly, or by a large number of "handler" devices, which themselves take directions from the master. b) The shaded hosts (zombies) send packets spoofing the source address of a common (end) victim, such that the responses flood that victim.

Local vs. remote DoS. A DoS attack on a local host might involve simply triggering a buffer overflow in a kernel function, or replicating malware that consumes memory and CPU (cf. rabbits, Chapter 7). Our discussion here focuses instead on (remote) network-related DoS, requiring no *a priori* access to, or account on, a local host.

Example *(DoS by poison packets).* A variety of attacks have used *malformed packets* to trigger implementation errors that terminate a process or crash the operating system itself. For example, the *Ping of Death* is a ping (ICMP echo request) sent as packet fragments whose total length exceeds the 65,535-byte maximum IP packet size. Packet reassembly crashed numerous circa-1996 TCP/IP stack implementations by overflowing allocated storage. A second example, *Teardrop*, sent a packet in fragments with fragment offset fields set such that reassembly resulted in overlapping pieces—crashing TCP/IP reassembly code in some implementations, exhausting resources in others. A third example, *LAND*, sends a SYN packet with source address and port duplicating the destination values, crashing some implementations that send responses to themselves repeatedly. Note that any Internet host can send any of these packets. Such attacks, while high-impact, have clear fixes—simply repairing errors underlying the vulnerabilities (e.g., with standard length and logic checks). Filtering based on source address also helps (Fig.11.6 on page 324).

False source IP addresses. A common tactic in DoS attacks is to send packets with false (often random) source IP addresses. The IP protocol does nothing to stop this. Such *IP address spoofing* can give a superficial appearance that packets are arriving from many places, prevents trivial traceback of the packets, and defeats simple blocking based on source address. A false address means that the true source will not get an IP response, but the attacker does not care. Responses go to the spoofed addresses as *backscatter*.

Example *(SYN flooding: resource exhaustion).* One of the earliest and best known DoS attacks, *SYN flooding*, provides insightful lessons on the ease of abusing open proto-

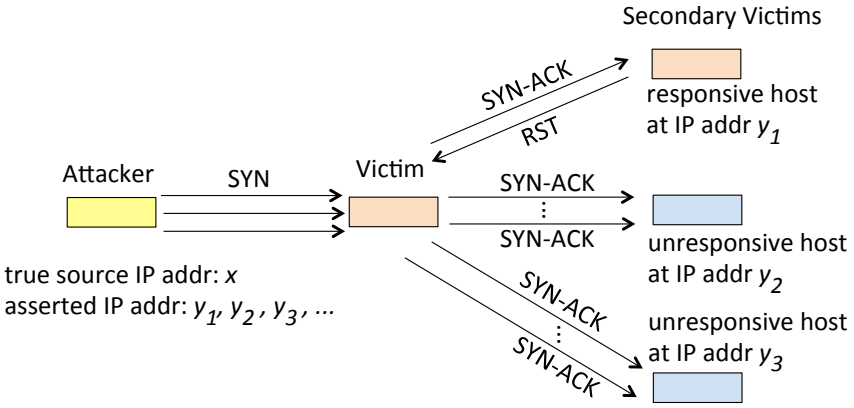


Figure 11.5: SYN flooding with spoofed IP address.

cols, here basic TCP/IP connection set-up.[12] By protocol, on receipt of a TCP SYN packet, the destination sends a SYN-ACK, considers the connection "half-open" (SYN_RECEIVED), and maintains state (e.g., socket and sequence number details) while awaiting the third handshake message. The memory used to maintain this state is typically statically pre-allocated (to avoid dynamic allocation within kernel interrupts), which limits the number of half-open connections. On reaching the limit, new connections are refused until state is freed by, e.g., time-out expiry of a pending connection, or an RST (reset) sent by a host in response to an unexpected SYN-ACK. A SYN flooding attack continually sends SYN packets (first messages), consuming the resource pool for half-open connections—degrading service to legitimate users, whose connection requests compete.

‡**COMMENTS: SYN FLOODING.** SYN flooding as just described neither brute-force floods a network link (to exhaust bandwidth), nor floods an end-host CPU by pure volume of packets. Instead, it exhausts pre-allocated resources with a relatively modest number of connection requests. The original attacks used false IP source addresses but rather than random, they were known unresponsive (e.g., unallocated) addresses; see Figure 11.5. In this case, the victim host periodically resends SYN-ACKs until a time-out period expires, consuming additional CPU and bandwidth; in contrast, on receiving unexpected SYN-ACKs, a responsive host will send an RST (reset), resulting in closing the half-open connection, freeing state earlier. A responsive host's RST is not a total loss for the attacker—sending one SYN packet results in a minor *amplification*, with three resource-consuming packets in total, including the SYN-ACK and RST. This results in a different (less elegant) attack, not so much on the resources allocated to handle half-open connections, but overwhelming network bandwidth and victim CPU by volume of packets.

In SYN flooding by large numbers of compromised machines (bots), access to networking stack software or to *raw sockets* may be used to arrange false source addresses. If true source addresses of bots are used, without altering native network stack implementation, native responses to SYN-ACKs complete TCP connections, resulting in DoS

[12]This section assumes familiarity with basic concepts from networking, per Section 10.6.

by volume flooding (vs. half-open starvation). Aside: while true source addresses allow bot identification, removing malware from bots itself raises pragmatic difficulties—due to scale, and inability of individual defenders to contact thousands of device administrators. Flooding via a botnet also complicates blocking-based defenses.

‡**Exercise** (In-host SYN flood mitigation). Explain how these mechanisms mitigate SYN flooding, and any problems: a) *SYN cookies*; b) *SYN cache*. (Hint: [29, 51, 38].)

‡**Exercise** (Reluctant allocation). SYN flooding attacks exploit end-host TCP protocol implementations that do not follow principle P20 (RELUCTANT-ALLOCATION), instead willingly allocating resources before sanity-checking the legitimacy of the connection request. Cryptographic protocols such as Diffie-Hellman (DH) key agreement are also subject to DoS attacks. a) Summarize three categories of DoS issues for IPsec-related DH implementations. b) Discuss how one DH variant, the *Photuris protocol*, follows P20 to address at least one of these concerns. (Hint: [2, 45].)

UDP AND ICMP FLOODS. Brute-force packet transmission simply overwhelms hosts' bandwidth and CPU. Sending a large number of ping (ICMP echo request) packets to a target, each triggering an ICMP echo reply, is one type of *ICMP flood*. A similar *UDP flood* may bombard UDP packets at random ports on a target—most ports, closed, will result in ICMP "destination unreachable" responses (consuming further bandwidth). Such attacks use protocols often allowed by firewalls in the past, or that are essential for network operations; if an administrator blocks ICMP "echo request" packets outright (foregoing useful functionality), or beyond a set threshold, attackers may instead use ICMP "destination unreachable" packets.

Example *(Smurf flood).* A second type of ICMP flood using ping (echo request) packets and false IP addresses employs broadcast addresses to gain an *amplification* factor. As background, consider a 32-bit IPv4 address as an n-bit prefix defining a network ID, and m-bit suffix identifying a host within it ($n+m=32$); all-zeros and all-ones suffixes are special, all-ones denoting a *broadcast address*. A packet sent to a broadcast address by a local-network host goes to all hosts on that network; and likewise if from a host outside the local network, if routers are suitably configured. *Smurf attacks* from outside target networks send ICMP pings to the broadcast address of target networks (accomplices). On reaching an accomplice network, the ping solicits from each host therein an ICMP echo reply, consuming both that network's bandwidth and the path back to a spoofed source address, the true victim; the unwitting accomplices are secondary victims. Similar attacks can be launched from a compromised host within a local network, on the IP broadcast address of that network. Note that this attack may use any packet (service) evoking general responses, and allowed through firewalls/gateways; ICMP ping is but one choice.

SMURF MITIGATION. One mitigation for externally originated Smurf attacks is for (all) routers to drop packets with broadcast address destinations (like Martian packets, below). Another mitigation is ingress/egress filtering (below). Attacks from within a local network itself may be mitigated by configuring host OSs to ignore ICMP packets arriving for IP broadcast addresses; local hosts will no longer be accomplices.

AMPLIFICATION. In SYN flooding, ICMP flooding (Smurf ping), and UDP and TCP exploits noted below, DoS attacks are aided by *amplification*—this occurs in any protocol

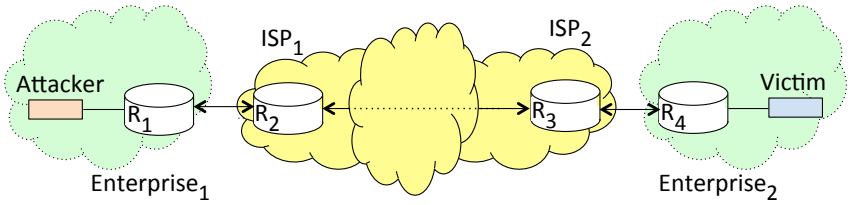


Figure 11.6: Ingress and egress filtering. An attacker may use a spoofed source IP address in traffic sent to a victim. ISP_1 does ingress filtering at R_2 for traffic entering from $Enterprise_1$. $Enterprise_1$ does egress filtering at R_1 for traffic leaving to ISP_1. For firewall rules to implement ingress and egress filtering, see Table 10.1 in Section 10.1.

where originating one message results in more than one response (from one or multiple hosts), or in a response larger than the original packet, or both. In open network protocols, sending a packet requires no authentication, but consumes bandwidth (a shared resource) and host resources of all recipients. This violates in part principles P4 (COMPLETE-MEDIATION), P6 (LEAST-PRIVILEGE), and P20 (RELUCTANT-ALLOCATION).

Exercise (UDP amplification). CERT Advisory CA-1996-01 recommended that aside from using firewall or gateway filtering of related ports, hosts disable unused UDP services,[13] especially two testing/debugging services—the Echo (UDP port 7) and Character Generator (UDP port 19) protocols. (a) Explain what these services do, and the concern, especially with packets whose source and destination sockets connect the services to each other. (b) What is the general risk, if a service generates more output than it receives?

‡**Exercise** (ICMP-based attacks). Outline several ways ICMP can be abused to attack TCP connections; mitigations that comply with ICMP specifications; and challenges in validating the authenticity of ICMP messages. (Hint: [36, 38]; Linux and some Unix OSs include validation checks on sequence numbers found within ICMP payloads.)

INGRESS FILTERING. Ingress filters process packets entering, and egress filters process packets leaving, a network (Fig. 11.6). They mitigate IP source address spoofing, and thus DoS attacks that employ it (TCP SYN, UDP, and ICMP flooding). Service providers use *ingress filtering* on a router interface receiving input packets from a customer network; the filter allows only packets with source addresses within ranges expected or known to be legitimate from that customer network, based on knowledge of legitimate address assignment. Packets with *Martian addresses* (e.g., an invalid source address due to being reserved or a host loopback) are also dropped. An enterprise may likewise do *egress filtering* on packets leaving its network, based on knowledge of legitimate addresses of its internal hosts, to avoid assisting hosts serving as attack agents.[14]

‡**Exercise** (TCP amplification). a) Explain how TCP-based protocols can be abused for amplification attacks despite TCP's three-way handshake. b) Give three reasons why NTP is most vulnerable among these. c) Summarize technical mitigations to NTP amplification attacks per advisories of MITRE (2013) and US-CERT (2014). (Hint: [49].)

[13] Disabling unused services follows principle P1 (SIMPLICITY-AND-NECESSITY).

[14] Ingress/egress filtering supports principle P5 (ISOLATED-COMPARTMENTS).

Example *(DDoS toolkits).* DDoS toolkits emerged in the late 1990s. The *Tribal Flood Network* (TFN), and successor *TFN2K*, allow a selection of UDP flood, ICMP flood, ICMP broadcast (Smurf type), and SYN flood attacks. Target addresses are inputs. Attack client binaries are installed on compromised hosts (bots). TFN-based *Stacheldraht* added encrypted communications between control components, and update functionality.

‡**Exercise** (DDoS: trinoo). DDoS incidents in 1999 used *trinoo* tools. a) Detail how *trinoo* compromised hosts to become slaves called daemons. b) Summarize its master-daemon command and control structure, pre-dating those in later botnets. c) Summarize the technical details of the DoS vectors used by the daemons (hint: [27]).

Exercise (Mirai botnet 2016). The *Mirai* (DDoS) botnet exploited embedded processors and Internet of Things (IoT) devices, e.g., home routers and IP cameras. (a) Summarize its technical details. (b) Discuss the implications for IoT security. (Hint: [48, 5].)

SUMMARY COMMENTS: ATTACKS. DoS attacks are, by definition, easy to notice, but full solutions appear unlikely. Flooding-type attacks are as much a social as a technical problem, and cannot be prevented outright—public services are open to the entire public, including attackers. Defenses are complicated by IP address spoofing, the existence of services (protocols) that can be exploited for amplification, and the availability of botnets for DDoS. DoS artifacts include poison packets and resource exhaustion (slow or fast) on end-hosts, network bandwidth exhaustion, and related attacks on networking devices.

SUMMARY COMMENTS: DEFENSES. Default on-host DoS defenses should include disabling unused services, OS rate-limiting of ICMP responses, and updating software to address poison packets. Good security hygiene decreases the chances that end-hosts become part of a (DoS) botnet, but flooding defenses are largely in the hands of network operators, e.g., blocking non-essential services at gateways, and dropping packets from blacklisted sources and by ingress/egress filtering. Coarse filtering at firewalls is an interim survival tactic when new attacks arise and better alternatives are not yet in place. Proxy firewalls may have the capacity to filter out or alter malformed packets, albeit requiring protocol-level knowledge. Beyond this, flooding attacks are addressed by shared hardware redundancy of ISPs and infrastructure providers—e.g., sites hosted by CDNs (content delivery networks) benefit from spare capacity in resources, and major enterprises invest (with cost) in links with excess capacity, server farms, and load balancing. Sharing of defensive resources is driven by the reality that attackers (leveraging botnets) can harness greater resources than individual defenders. A challenge for future networks is the design of communications protocols and services immune to amplification attacks.

11.5 Address resolution attacks (DNS, ARP)

Here we discuss traffic-hijacking attacks involving DNS (pharming) and ARP (ARP spoofing). Both exploit failures to provide REQUEST-RESPONSE-INTEGRITY (principle P19).

DNS. The Domain Name System (DNS) maps hostnames (Section 9.1) to IP addresses. The full mapping is a distributed database, held entirely by no individual host. Each organization is the authoritative information source for IP addresses of their own

hosts, and in response to DNS protocol queries on UDP port 53, provides this information through server programs. Client applications resolve hostnames using a local (OS-provided) *DNS resolver*, which returns corresponding IP addresses. To get the answer, the resolver in turn contacts one or more *DNS servers*, which contact further sources in a hierarchical query structure, finally asking the authoritative source if required. At various points (Fig. 11.7), query answers may be cached for quicker future responses; by protocol, a cached entry is deleted after a time-to-live (TTL) value specified in its DNS response.

Example *(DNS resolution).* The hostname `www.tgtserver.com` is resolved to an IP address as follows. In Fig. 11.7, assume that all caches are empty, and that the client is configured to use DNS services of its ISP (Internet Service Provider). The application calls (1) the local DNS resolver, which in turn makes a query (2) to the ISP's local DNS server. That server queries (3) the ISP's regional DNS server, S_2. So far, these have all been *recursive queries*, meaning the service queried is expected to return a (final) answer, itself making any further queries as necessary. At this point, S_2 begins a sequence of *interactive queries*, descending down the DNS global hierarchy of Fig. 11.7c) until at some level a server fully resolves the query. The first query (4) is to one of 13 global DNS root servers.[15] The root server R_1 responds with the address of a server (say, T_1) that can handle `.com` queries. S_2 sends a request (5) to T_1, which responds with the address of a server (say, A_1) than can handle `.tgtserver.com` queries. S_2 finally sends a query (6) to A_1. A_1 can return the desired (complete) answer, i.e., the IP address of `www.tgtserver.com`, because A_1's DNS server is administered by the organization that registered the domain `tgtserver.com`, and controls its subdomains (and the IP addresses mapped to the corresponding hostnames) including `www`. The response from A_1 to S_2 is relayed by S_2 to S_1, which returns it to the local DNS resolver L. Each of L, S_1 and S_2 now caches this <hostname, IP address> pair, to expedite repeat queries in the near future.

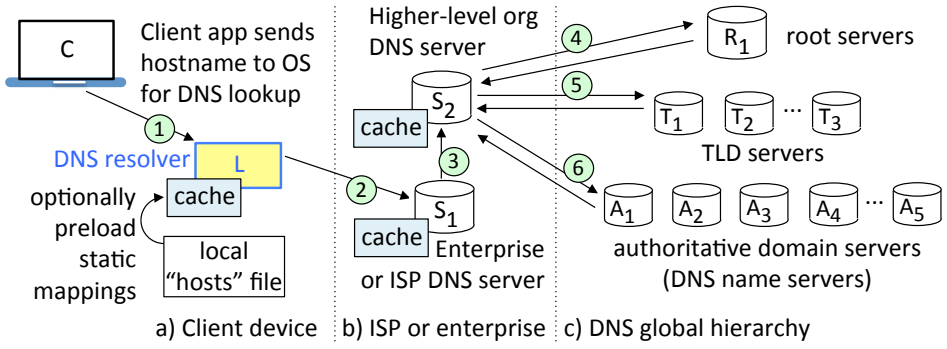


Figure 11.7: DNS name resolution and query hierarchy (simplified).

PHARMING AND DNS RESOLUTION. A *pharming* attack is any means that falsifies the mapping between domain name and IP address. Recall *phishing* (Section 9.8) involves tricking a user to end up on a malicious (often a look-alike of an authentic) site, via some means (*lure*), e.g., a link in an email or web search result. Pharming achieves this with

[15] Root servers are load-balanced clusters; one or more root IP addresses is known to the querying server.

no lure, by forging address resolution. In this case, e.g., a user manually typing a correct domain name into a browser URL bar can still end up at an incorrect IP address, thus retrieving data from a false site. Among other issues facilitating attacks, basic DNS queries and replies are currently void of cryptographic protection (i.e., are unauthenticated).

Example *(DNS resolution attacks).* Figure 11.7 hints at a wide attack surface exposed by the basic DNS resolution process. A few well-known attack vectors are as follows:

1. Local files. On both Unix and Windows systems, a local "hosts" file often statically defines IP addresses for specified hostnames (configuration determines when or if this file is used before external DNS services). This hosts file, and the DNS client cache (DNS resolver cache), are subject to tampering by malware.
2. Tampering at intermediate DNS servers. DNS caches at any other servers (e.g., S_1, S_2) are likewise subject to tampering by malware, and by inside attackers (perhaps involving collusion or bribery). Even authoritative name servers are subject to malicious tampering by insiders, albeit more widely visible.
3. Network-based response alteration. Middle-person attacks on any untrusted network en route can alter (valid) DNS responses before reaching the original requestor.
4. Malicious DNS server settings. Clients are configured to use a specific external DNS server (Fig. 11.7b). Its IP address, visible by a DNS settings dialogue, is subject to being changed to a malicious DNS server. The risk is especially high when using untrusted networks (e.g., in Internet cafés, airports, hotels), as guest IP addresses are commonly allocated using DHCP (Dynamic Host Configuration Protocol); this often results in client devices using DNS servers assigned by the DHCP server provided by the access point, whether wireless or wired.

A further major network-based attack vector involves DNS spoofing (next).

‡**Exercise** (DNS poisoning). *DNS spoofing* is unauthorized origination of (false) DNS responses. a) Explain how a sub-type of this, *DNS cache poisoning* attacks, work in general, including the role of 16-bit ID fields in DNS protocol messages. b) Explain how the Kaminsky technique dramatically increased attack effectiveness. c) Explain how randomized 16-bit UDP source ports are of defensive use. d) Explain how mixing upper and lower case spelling of queried hostnames increases attack difficulty. (Hint: [23].)

‡**Exercise** (DNS attacks). Grouping DNS attacks by architectural domain exploited, describe at least one attack for each of five domains: local services; ISP or enterprise services; global DNS services; authoritative DNS services; domain registrars. (Hint: [63].)

PHARMING DEFENSES. As DNS is a core infrastructure, many security issues related to DNS resolution are beyond the control of regular users. Avoiding use of untrusted networks (e.g., guest Wi-Fi service) is easy advice to give, but not generally pragmatic. A long-term solution, Domain Name System Security Extensions (*DNSSEC*), offers digitally signed responses to DNS queries, but its deployment has been slow, due to the complexity of universal deployment of a supporting public-key infrastructure.

ARP. On a local area network (LAN), Ethernet frames are delivered by MAC address. The *Address Resolution Protocol* (ARP) is used to map IP addresses to MAC addresses. A host aiming to learn a MAC address corresponding to a target IP address sends out a

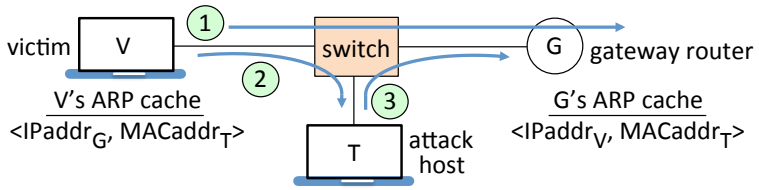


Figure 11.8: ARP spoofing. Intended flow (1), actual flows (2), (3). *T* poisons *V*'s ARP cache. As a result, traffic sent via *G* over the LAN, intended (1) for a destination beyond *G*, is instead sent (2) by *V* to the physical interface of *T*. By also poisoning *G*'s ARP cache, *T* can arrange that incoming traffic to *V* via *G* is sent by *G* to *T*. Thus *T* has a LAN middle-person attack between *V* and *G*. Note: the switch itself is not poisoned.

LAN broadcast message indicating the IP address; the protocol specifies that any host having a network interface assigned that IP address reply with the pair <IP address, MAC address>. Each LAN host then keeps a local table (*ARP cache*) of such responses (mapping OSI layer 3 to layer 2 addresses), as an efficiency to reduce future ARP requests.

ARP SPOOFING. An attacking host can send false ARP replies, asserting its own MAC address as that of the device located at a same-LAN target (victim) IP address. This is *ARP spoofing*, and results in false entries in ARP caches, i.e., *poisoned ARP caches*. It is possible because: 1) ARP replies are not authenticated (any LAN host can reply), and 2) hosts commonly accept replies even in the absence of requests—existing entries are overwritten. In this way, the physical interface to the attack host (*T* in Fig. 11.8) can receive Ethernet frames intended for other LAN hosts. This allows *T* to monitor traffic (before possibly altering and forwarding it), even on a switched LAN.

ARP SPOOFING DEFENSES. ARP spoofing is stopped by static, read-only per-device ARP tables mapping IP address to MAC address; setting and updating these manually requires extra effort. Various tools (beyond our scope) may detect and prevent ARP spoofing, for example, by cross-checking ARP responses. A preferred long-term solution is a reliable form of authentication in an upgraded Address Resolution Protocol.

‡**Exercise** (Port stealing, MAC flooding). Two further attacks that exploit failures to provide REQUEST-RESPONSE-INTEGRITY (P19) involve data link (layer 2) manipulations of network switch *MAC tables*. These tables, unlike ARP tables, map MAC addresses to the physical interfaces (switch ports) to which individual LAN devices are wired. Explain each attack: a) *port stealing*; and b) *MAC flooding*. (Hint: [76]. These attacks can be stopped by manually configuring switch ports with specific MAC addresses, again with extra management effort, and beyond the capability of end-users.)

Exercise (Comparing attacks). Explain how DNS resolution attacks and ARP spoofing are analogous, by using technical details of how each (a) maps identifiers from one network layer to another; and (b) can turn an off-path attacker into an on-path attacker.

‡**Exercise** (Beyond passive sniffers: dsniff, Ettercap). Beyond passive packet capture, broader tools provide active packet manipulation capabilities, e.g., supporting middle-person attacks, ARP spoofing, and denial of service attacks. dsniff is an Ethernet sniffing toolset whose authorized uses include penetration testing and security auditing. Ettercap is

positioned as the premier network-attack middle-person tool. Summarize the functionality of: a) dsniff (including its arpspoof component), and b) Ettercap. (Hint: [76], [19].)

11.6 ‡TCP session hijacking

We now discuss TCP session hijacking, and end with a few final network-based attacks.

TCP SESSION HIJACKING: INTRODUCTION. An *on-path* attacker can easily manipulate unencrypted TCP sessions, and *TCP session hijacking* (below) is conceptually straightforward. The main barrier to overcome is a sequence number mechanism designed for synchronization of TCP/IP packets; trickier implementation details are disposed of by widely available, point-and-click toolkits. This is a rather serious affair, as it requires no host access or privileges; e.g., for a plaintext `telnet` session, this allows injecting commands that will be executed under the authority of the process for which the victim session is running. In one variation, one legitimate end-host loses its ability to participate in the session, its TCP sequence numbering out of synch with the other end. In a second, the attacker plays middle-person, relaying possibly altered packets between legitimate hosts, repairing header sequence numbers to valid values on the fly. These attacks are possible when packets are sent plaintext, and are *not* stopped by start-of-session authentication (even if based on one-time passwords or cryptography). This strongly motivates use of encryption (e.g., via SSH, TLS, IPsec); not only are sniffed packets then unintelligible, but injected packets will decrypt to meaningless bytes at legitimate end-hosts.

HIJACKING AND MIDDLE-PERSONS. TCP session hijacking is distinct from:

1) HTTP session hijacking via cookie theft (Chapter 9);
2) middle-person attacks on Diffie-Hellman key agreement (Chapter 4), which may use a DNS-based means to redirect packets to an intruder-controlled host; and
3) ARP spoofing, which may itself be used as a tool within TCP session hijacking.

Middle-person attacks involving TLS may rely on DNS-based misdirection or resolution exploits, combined with use of a fraudulent or misleading web site certificate.

TCP SEQUENCE NUMBERS (BACKGROUND). TCP header fields (Section 10.6) *sequence number* and *acknowledgement number* help manage the stream of bytes delivered; every data byte transferred is numbered, in separate sequences for each direction. During the three-way handshake, each end chooses a new *initial sequence number* (ISN), associating $\text{ISN}+1$ with its first data byte transmitted. Each end populates the *ack number* field with the next sequence number they expect to receive. The SYN flag in the first message $\texttt{seq}(\text{ISN}_a)$ requests the other end to synchronize to the new ISN value sent by the client. The SYN-ACK segment returned by the server includes its ISN as $\texttt{seq}(\text{ISN}_b)$ and $\texttt{ack}(\text{ISN}_a+1)$ to acknowledge the client's sequence number. All TCP segments in the connection now set the ACK flag (Fig. 11.9), indicating a valid field value in *ack number*. The client's handshake completion is $\texttt{ack}(\text{ISN}_b+1)$. In the now-established TCP connection, each sender populates every segment's *seq number* field with the sequence number that the segment's first data byte corresponds to; and populates *ack number* with the byte number of the next byte missing in its own receive buffer (confirming that every

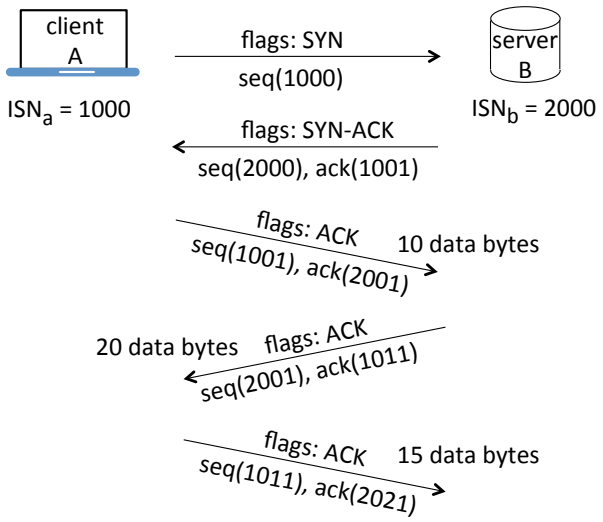


Figure 11.9: TCP three-way handshake and sequence numbering. As shown, the third handshake message's ACK may be delayed and piggy-backed onto a data transfer rather than in an empty segment. The SYN flag counts for one position in the number sequence.

byte preceding this was successfully received). If a gap results due to a segment being corrupted or received out of order, the same ACK value may be resent several times (the missing segment is eventually received, or triggers retransmission). Sequence numbers are particularly relevant due to their role in TCP session hijacking, and RST attacks.

TCP SESSION HIJACKING: CONTEXT. As noted, sequence-numbering fields in TCP headers synchronize byte streams. If SND.NXT$_a$ numbers the next byte A will send (next sequence number to be used), and RCV.NXT$_b$ numbers B's last acknowledgement, then in quiet periods: SND.NXT$_a$ =RCV.NXT$_b$ and SND.NXT$_b$ =RCV.NXT$_a$. Designed for accounting (not security), this byte numbering allows proper handling and placement in receive buffers (and any retransmission) of TCP segments lost or received out of order, and filling of any temporary buffer data holes. If sequence numbers are not within a valid range,[16] a TCP segment (packet) may be dropped; precise conditions depend on the TCP specification and details such as the receive window size (RCV.WND), and how many received segments remain unacknowledged. The attacker aims to craft a packet with valid sequencing numbers, relative to the receiver's TCP state machine.

TCP HIJACKING: OUTLINE. Consider a TCP connection between hosts A and B, with attacker T somewhere *on-path* (on the path the packets travel). Using any sniffer, T can read packet contents including A's IP address. Using any packet creation tool, T will send packets to B, falsely asserting A's source address—this is not prevented by TCP (Fig. 11.10, label 1). Responses will not be addressed to T, but this doesn't matter; they are visible by on-path sniffing. For the A–B connection, T sniffs socket details and the session's current sequencing numbers, using these to craft and inject packets whose TCP

[16] As standard checks, an incoming value SEG.SEQ should be in [RCV.NXT, RCV.NXT+RCV.WND], and an incoming SEG.ACK should never be for bytes not yet sent, so SEG.ACK$\leq$SND.NXT is required.

segments have sequencing numbers valid from B's view of the A–B protocol state. B now receives TCP segments from T, processing them as if from A (e.g., data, commands, programs). To complete the attack, T removes A from being a nuisance (described next).

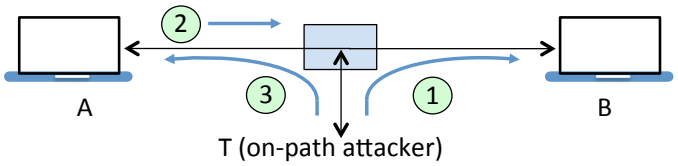


Figure 11.10: TCP session hijacking. T need not be on the LAN of either A or B.

HIJACKING SIDE EFFECT 1: DESYNCHRONIZATION. This attack has side effects and races. Ongoing packets sent by A to B (Fig. 11.10, label 2), while T is also sending packets to B, may render T's byte numbering invalid. For this reason, T may try to improve his odds, e.g., flooding A with packets to slow A's ability to send to B, or sending poison packets to A (Fig. 11.10, label 3). Another option is to actively desynchronize A–B byte numbering. T does so by tampering with the connection set-up: when T sees the SYN–ACK from B to A (second message in the three-way handshake), T sends B an RST segment immediately followed by a SYN with the same socket details but new ISN_T. This causes B to end the first connection to A, then open a new one with new ISN_{B^*} in the SYN–ACK. Observing this, T responds (spoofing A's address) with an ACK. Meanwhile, A believes it has established a TCP connection with B, but is now using sequence numbering different than B. This desynchronized state prevents data exchange between A and B. If desired, T can sniff these packets, modify any contents, and adjust the byte-number fields to render relayed packets acceptable—acting as a middle-person.

‡HIJACKING SIDE EFFECT 2: ACK STORMS. As T injects packets acceptable to B, the byte numbers (and corresponding receive windows) between T (as A) and B advance. As A is out of synch, any packets exchanged between A and B are dropped. In this case, TCP mandates an ACK segment (without data); it will trigger a response, as follows. Each host will repeatedly (and hopelessly) inform the other of byte numbers they expect to next send, and receive. Thus B sends an ACK segment to A asserting: $\texttt{SND.NXT}_b$, $\texttt{RCV.NXT}_b$. A reciprocates, asserting: $\texttt{SND.NXT}_a$, $\texttt{RCV.NXT}_a$. And so on, in an ACK loop until a packet is lost, or the TCP connection is reset. Aside from this *assertion* ACK *storm*, any packets from A to B containing data bytes that go unacknowledged will be resent, raising another ACK storm; T can preempt such follow-up storms by crafting false ACKs.

‡TAMING ACK STORMS. To reduce ACK storms, T may send to A denial of service packets (as noted earlier), or an RST segment. This turns a potential middle-person attack into a straight takeover. (After all, T may feel that A is now disturbing T's otherwise pleasant session with B.) An alternate, cleaner method to avoid ACK storms is ARP spoofing to redirect packets intended for A and B to T;[17] this prevents A and B from receiving packets that trigger ACK responses due to, in the receiver's view, invalid sequencing numbers. Such storms may also be tamed by (legitimate) TCP implementations that automatically

[17] Ettercap and other widely available tools use this method.

rate-limit duplicate or repeated ACKs, e.g., to a few per second.

BLIND TCP RESET ATTACKS. Methods related to those for hijacking (above) have been used for *off-path* ("blind") TCP DoS attacks; here sequence numbers must be guessed rather than observed (sniffed). A first attack aims for connection teardown by sending an RST with valid sequence number. In older TCP implementations, any in-window sequence number is valid (new implementations require exactly matching `SND.NXT`). A second attack, sending an unexpected SYN segment (e.g., for an established connection), aims to trigger an RST. In older TCP implementations, any in-window sequence number triggers an RST (in later implementations, such a segment calls for an assertion ACK and dropped segment, with an RST only if the response is a valid RST; if the SYN was spoofed, the peer discards the ACK as a duplicate ACK). Such resets are of special concern to, e.g., long-lived router connections where connection teardown results in long disruptions due to recomputations needed to rebuild routing tables.

MITIGATING TCP-BASED ATTACKS. To mitigate off-path attacks (immediately above), RFC 5961 provides an update to the TCP specification as noted in Section 11.7. More generally, the main defense against TCP hijacking attacks is, as noted at the beginning of this section, to encrypt network communications using, e.g., SSH, TLS or IPsec.

11.7 ‡End notes and further reading

For background on basic concepts in networking, see Section 10.6, and Chapter 10 end notes for foundational Internet specifications (IP, ICMP, TCP, UDP).

Anderson's 1980 report [3] proposed the detection of *abnormal* computer usage by analyzing audit data. Distinct from legitimate users misusing authorized access (*misfeasance*), he distinguished three types of intruder: *masquerador* (user of stolen credentials), *clandestine user* (evading audits, e.g., as root user modifying audit trail records), and *external penetrator*. The report laid the groundwork for (host-based) *anomaly-based* intrusion detection; suggested use of *sum-of-squares* and *standard deviation* to measure variation of parameters from recorded averages; and recommended creating *security audit trails* independent of accounting and finance needs. Denning's anomaly-based generic IDS model [26] appeared after publication of work on IDES with Neumann [25] that detailed statistical (profile-based) anomaly detection. IDES used both this and *rule-based anomaly detection* methods—see Lunt [52]. Bejtlich [13, App. B] summarizes early network monitoring and NIDS papers. As NIDSs became fashionable, focus shifted from Anderson's goal of finding intruders already inside, to trying to stop attacks in progress.

For IDS background and historical context, see Bace [11] and Debar [24]. Snort [72] is used not only for IDS, but also for packet sniffing and logging. Bro, "an Orwellian reminder that monitoring comes hand in hand with the potential for privacy violations" [64], has a network monitor origin, as does Ranum's Network Flight Recorder (NFR) [71], which predates Snort by two years. The signature-based Suricata NIDS [83] was released in 2010 as a multi-threaded competitor to Snort's single-threaded engine. Signature-based IDS is sometimes categorized under *misuse detection*. For a type of *stateful protocol anal-*

ysis (cf. NIST [75]), Bro signatures can use context (including state) to reduce false positives, as noted by Sommer [77, Sect. 3.5], [78]; the latter also discusses converting Snort rules to Bro signatures and compares the systems. Such *contextualized signatures* differ from using Bro in pure specification-based approaches, which have a stronger whitelist basis—see Ko [47] and Uppuluri [80]. Ptacek [68] explains how IDS evasion is possible by maliciously fragmenting packets and related means causing ambiguities in packet reassembly. Handley [39] outlines *traffic normalization* to address this (related literature refers to *protocol scrubbers*), in a broad-sense example of principle P15 (DATA-TYPE-VERIFICATION).

Rather than industry-driven IDSs, Bejtlich argues for network security based on strong *monitoring* tools in books focused on inbound [13] and outbound traffic [14], calling the latter *extrusion detection*. For IDS in practice, see also Northcutt [61, 60]. For the BSD Packet Filter (BPF) widely used in packet capture tools, see McCanne [53]. Safford [74] introduces Drawbridge, a *filtering bridge*, and describes the TAMU security package, an early monitoring and intruder defense system. Decoy targets called *honeypots* (hosts with no legitimate users) allow extraction of knowledge from attackers and malware capture for signature generation; see Provos [67] for this and Honeyd, and Cheswick [21] including for use of a *chroot jail*. Bellovin's early Internet-monitoring papers [15, 16] were illuminating. The Unix `finger` command (RFC 1288), heavily used in early Internet days to obtain information about users on remote hosts, is deprecated (commonly unsupported), for security reasons. On the *base rate fallacy*, see Axelsson [10] for IDS implications (and IDS base rates), and Beauchemin [12] as it relates to generating probable prime numbers.

Software flaws including buffer overflows (for which Chapter 6 noted static analysis) can be found by *fuzzing* (*fuzz testing*), which may offer information on offending inputs as a bonus. Miller's seminal fuzzing studies [56, 55] explored software responses to random input, respectively for Unix commands and MacOS applications. For *software fault injection* (a sub-class of fuzzing), see Voas [81] and also Anley [4, Ch. 16-17]. For fuzzing and broader penetration testing (cf. McGraw [54, Ch. 6]) see Harper [40], as well as for Metasploit and responsible disclosure. Regarding curious Metasploit usage patterns over the first two days after release of new exploit modules, see Ramirez-Silva [70]. For vulnerability assessment, scanning and exploitation tools, and defenses, see Skoudis [76]. SATAN [31] popularized the now well-accepted practice "hack yourself before the bad guy does"; Bace [11] explains how earlier self-assessments were *credentialed*, i.e., used on-host tools such as COPS [30] run on authorized user accounts. In a later book [32], the SATAN authors explore *computer forensics*. OS detection via TCP/IP stackprinting in Nmap is from Fyodor [35]. p0f is documented by Zalewski [85]. Xprobe2 originates from Ofir [62]. For detection of port scanning based on a small number of probes, see Jung [44], and Staniford [79] for detecting stealthy port scans. For use of *exposure maps* to enumerate sockets responsive to external connection attempts, see Whyte [84]. For Internet-wide scanning using ZMap, see Durumeric [28].

Abliz [1] offers a comprehensive survey on DoS. Paxson [65] explores DDoS attacks by which packets sent to a large number of *reflectors* (any IP host that responds to packets sent) target a specific true victim as the false source IP address, with responses flooding

that victim. See Rossow [73] for a study of UDP-based network protocols vulnerable to amplification attacks, and countermeasures. Moore [59] measures global DoS activity using *backscatter analysis*. Jung [43] discusses relationships between DoS, flash crowds, and CDNs. For *ingress filtering*, see RFC 2827 [33]; and for additional TCP SYN flooding mitigations, RFC 4987 [29]. SYN flooding, popularized by daemon9 [22], was known to Bellovin [18]. DoS-related CERT Advisories (CA) and Incident Notes (IN) include suggested mitigations: CA-1996-01 (UDP flood), CA-1996-21 (TCP SYN flood), CA-1996-26 (*Ping of Death*), CA-1997-28 (*Teardrop*, *LAND*), CA-1998-01 (*Smurf*) and IN-99-07 (*Trinoo*, *TFN*). DNS is standardized in two RFCs by Mockapetris [57, 58]; for a threat analysis, see RFC 3833 [9], and Bellovin [17]. DNSSEC, a collection of new resource records and protocol modifications for DNS to provide data origin authentication and integrity to query-response pairs, is specified in three primary RFCs [6, 7, 8]; for implementation notes, see RFC 6840 [82].

TCP/IP suite vulnerabilities and mitigations are discussed in Gont's security roadmap [38], as a companion to RFCs; see also Bellovin's annotated lookback [18], including for routing-based attacks allowing on-path hijacking. TCP *off-path* (or *blind*) attacks aiming to disrupt (e.g., by resets) or inject data into/hijack connections, typically require knowing or guessing socket details plus an acceptable SEQ and/or ACK number; mitigations include using unpredictable TCP ISNs [37], *randomization of ephemeral ports* [50], and ([69], cf. [38]) narrowing the range of acceptable SEQ/ACK numbers plus additional *challenge* ACK*s*. These build on Morris' 1985 blind TCP connection-spoofing attack [37, Appendix A.1]. Joncheray [42] explained the details of TCP-based session hijacking. For TCP session hijacking by an on-path attacker that is not on the LAN of either end-host, ARP spoofing can be used on intermediate routers; see Skoudis [76, p. 488]. For ARP spoofing, see Bruschi [20]. ARP is defined in RFC 826 [66]. Regarding Ettercap, its authors note in an interview with Biancuzzi [19]: "We were studying for a university exam on networking, and we noticed that network security was more fun than differential equations."

References

[1] M. Abliz. Internet Denial of Service Attacks and Defense Mechanisms, Mar. 2011. University of Pittsburgh Technical Report TR-11-178, pp.1–50.

[2] W. Aiello, S. M. Bellovin, M. Blaze, J. Ioannidis, O. Reingold, R. Canetti, and A. D. Keromytis. Efficient, DoS-resistant, secure key exchange for Internet protocols. In *ACM Comp. & Comm. Security (CCS)*, pages 48–58, 2002. Journal version in *ACM TISSEC* (2004).

[3] J. P. Anderson Co. Computer Security Threat Monitoring and Surveillance, Feb 1980. Revised Apr 15 1980. James P. Anderson Co., Box 42, Fort Washington, PA, 19034 USA.

[4] C. Anley, J. Heasman, F. Lindner, and G. Richarte. *The Shellcoder's Handbook: Discovering and Exploiting Security Holes (2nd edition)*. Wiley, 2007.

[5] Antonakakis, M. and 18 others. Understanding the Mirai Botnet. In *USENIX Security*, 2017.

[6] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. RFC 4033: DNS Security Introduction and Requirements, Mar. 2005. Proposed Standard. Obsoletes RFC 2535 (which obsoleted 2065, Jan 1997); updated by RFC 6014, 6840.

[7] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. RFC 4034: Resource Records for the DNS Security Extensions, Mar. 2005. Proposed Standard. Updated by RFC 4470, 6014, 6840, 6944.

[8] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. RFC 4035: Protocol Modifications for the DNS Security Extensions, Mar. 2005. Proposed Standard. Updated by RFC 4470, 6014, 6840, 8198.

[9] D. Atkins and R. Austein. RFC 3833: Threat Analysis of the Domain Name System (DNS), Aug. 2004. Informational.

[10] S. Axelsson. The base-rate fallacy and its implications for the difficulty of intrusion detection. In *ACM Comp. & Comm. Security (CCS)*, pages 1–7, 1999. Journal version: ACM TISSEC 2000.

[11] R. G. Bace. *Intrusion Detection*. Macmillan, 2000.

[12] P. Beauchemin, G. Brassard, C. Crépeau, C. Goutier, and C. Pomerance. The generation of random numbers that are probably prime. *Journal of Cryptology*, 1(1):53–64, 1988.

[13] R. Bejtlich. *The Tao of Network Security Monitoring: Beyond Intrusion Detection*. Addison-Wesley, 2004.

[14] R. Bejtlich. *Extrusion Detection: Security Monitoring for Internal Intrusions*. Addison-Wesley, 2005.

[15] S. M. Bellovin. There be dragons. In *Proc. Summer USENIX Technical Conf.*, 1992.

[16] S. M. Bellovin. Packets found on an Internet. *Computer Communication Review*, 23(3):26–31, 1993.

[17] S. M. Bellovin. Using the domain name system for system break-ins. In *USENIX Security*, 1995.

[18] S. M. Bellovin. A look back at "Security problems in the TCP/IP protocol suite". In *Annual Computer Security Applications Conf. (ACSAC)*, pages 229–249, 2004. Embeds commentary into 1989 original "Security problems in the TCP/IP protocol suite", *Comp. Commn Review* 19(2):32–48, Apr 1989.

[19] F. Biancuzzi. The men behind ettercapNG. On linux.com, 9 Nov 2004, `https://www.linux.com/news/men-behind-ettercapng`; see also `https://www.ettercap-project.org/`.

[20] D. Bruschi, A. Ornaghi, and E. Rosti. S-ARP: A secure address resolution protocol. In *Annual Computer Security Applications Conf. (ACSAC)*, pages 66–74, 2003.

[21] B. Cheswick. An evening with Berferd in which a cracker is lured, endured, and studied. In *Proc. Winter USENIX Technical Conf.*, 1992.

[22] daemon9, route, and infinity. Project Neptune. In *Phrack Magazine*. 1 Sept 1996, vol.7 no.48, file 13 of 18 (with Linux source), `http://www.phrack.org`.

[23] D. Dagon, M. Antonakakis, P. Vixie, T. Jinmei, and W. Lee. Increased DNS forgery resistance through 0x20-bit encoding: SecURItY viA LeET QueRieS. In *ACM Comp. & Comm. Security (CCS)*, 2008.

[24] H. Debar, M. Dacier, and A. Wespi. A revised taxonomy for intrusion-detection systems. *Annales des Télécommunications*, 55(7-8):361–378, 2000.

[25] D. Denning and P. G. Neumann. Requirements and Model for IDES–A Real-Time Intrusion-Detection Expert System, Aug. 1985. SRI Project 6169-10, Menlo Park, CA, USA.

[26] D. E. Denning. An intrusion-detection model. In *IEEE Symp. Security and Privacy*, pages 118–133, 1986. Journal version: IEEE Trans. Software Eng. 1987.

[27] D. Dittrich. The DoS Project's 'trinoo' distributed denial of service attack tool. 21 Oct 1999, University of Washington, `https://staff.washington.edu/dittrich/misc/ddos/`.

[28] Z. Durumeric, E. Wustrow, and J. A. Halderman. Zmap: Fast internet-wide scanning and its security applications. In *USENIX Security*, pages 605–620, 2013.

[29] W. Eddy. RFC 4987: TCP SYN Flooding Attacks and Common Mitigations, Aug. 2007. Informational.

[30] D. Farmer and E. H. Spafford. The COPS security checker system. In *Proc. Summer USENIX Technical Conf.*, pages 165–170, 1990.

[31] D. Farmer and W. Venema. Improving the security of your site by breaking into it. White paper, available online along with tool, 1993. `http://www.porcupine.org/satan/admin-guide-to-cracking.html`.

[32] D. Farmer and W. Venema. *Forensic Discovery*. Addison-Wesley, 2005.

[33] P. Ferguson and D. Senie. RFC 2827: Network Ingress Filtering—Defeating Denial of Service Attacks that employ IP Source Address Spoofing, May 2000. Best Current Practice (BCP 38). Updated by RFC 3704: Ingress Filtering for Multihomed Networks, Mar 2004.

[34] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A sense of self for Unix processes. In *IEEE Symp. Security and Privacy*, pages 120–128, 1996.

[35] Fyodor. Remote OS detection via TCP/IP Stack FingerPrinting. In *Phrack Magazine*. 25 Dec 1998, vol.8 no.54, article 9 of 12, `http://www.phrack.org`. Nmap details: `https://nmap.org/book/`.

[36] F. Gont. RFC 5927: ICMP Attacks Against TCP, July 2010. Informational.

[37] F. Gont and S. Bellovin. RFC 6528: Defending Against Sequence Number Attacks, Feb. 2012. Proposed Standard. Obsoletes RFC 1948. Updates RFC 793.

[38] Gont, Fernando (on behalf of CPNI). Security Assessment of the Transmission Control Protocol (TCP). CPNI Technical Note 3/2009, Centre for the Protection of National Infrastructure (CPNI), U.K.

[39] M. Handley, V. Paxson, and C. Kreibich. Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics. In *USENIX Security*, 2001.

[40] A. Harper, S. Harris, J. Ness, C. Eagle, G. Lenkey, and T. Williams. *Gray Hat Hacking: The Ethical Hacker's Handbook (3rd edition)*. McGraw-Hill, 2011.

[41] S. A. Hofmeyr, S. Forrest, and A. Somayaji. Intrusion detection using sequences of system calls. *Journal of Computer Security*, 6(3):151–180, 1998.

[42] L. Joncheray. A simple active attack against TCP. In *USENIX Security*, 1995.

[43] J. Jung, B. Krishnamurthy, and M. Rabinovich. Flash crowds and denial of service attacks: characterization and implications for CDNs and web sites. In *WWW—Int'l Conf. on World Wide Web*, 2002.

[44] J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan. Fast portscan detection using sequential hypothesis testing. In *IEEE Symp. Security and Privacy*, pages 211–225, 2004.

[45] C. Kaufman, R. J. Perlman, and B. Sommerfeld. DoS protection for UDP-based protocols. In *ACM Comp. & Comm. Security (CCS)*, pages 2–7, 2003.

[46] D. Kennedy, J. O'Gorman, D. Kearns, and M. Aharoni. *Metasploit: The Penetration Tester's Guide*. No Starch Press, 2011. See also `https://www.metasploit.com` (The Metasploit Project).

[47] C. Ko, M. Ruschitzka, and K. N. Levitt. Execution monitoring of security-critical programs in distributed systems: A specification-based approach. In *IEEE Symp. Security and Privacy*, 1997.

[48] C. Kolias, G. Kambourakis, A. Stavrou, and J. M. Voas. DDoS in the IoT: Mirai and Other Botnets. *IEEE Computer*, 50(7):80–84, 2017.

[49] M. Kührer, T. Hupperich, C. Rossow, and T. Holz. Exit from hell? Reducing the impact of amplification DDoS attacks. In *USENIX Security*, pages 111–125, 2014.

[50] M. Larsen and F. Gont. RFC 6056: Recommendations for Transport-Protocol Port Randomization, Jan. 2011. Best Current Practice (BCP 156).

[51] J. Lemon. Resisting SYN flood DoS attacks with a SYN cache. In *USENIX BSDCon*, 2002.

[52] T. F. Lunt, A. Tamaru, F. Gilham, R. Jagannathan, P. G. Neumann, and C. Jalali. IDES: A progress report. In *Annual Computer Security Applications Conf. (ACSAC)*, pages 273–285, 1990. For details of the IDES anomaly-based statistical subsystem, see Javitz and Valdes, Oakland 1991.

[53] S. McCanne and V. Jacobson. The BSD packet filter: A new architecture for user-level packet capture. In *Proc. Winter USENIX Technical Conf.*, pages 259–270, 1993.

[54] G. McGraw. *Software Security: Building Security In*. Addison-Wesley, 2006. Includes extensive annotated bibliography.

[55] B. P. Miller, G. Cooksey, and F. Moore. An empirical study of the robustness of MacOS applications using random testing. *ACM Operating Sys. Review*, 41(1):78–86, 2007.

[56] B. P. Miller, L. Fredriksen, and B. So. An empirical study of the reliability of UNIX utilities. *Commun. ACM*, 33(12):32–44, 1990. Revisited in Tech. Report CS-TR-95-1268 (Apr 1995), Univ. of Wisconsin.

[57] P. Mockapetris. RFC 1034: Domain Names—Concepts and Facilities, Nov. 1987. Internet Standard. Obsoletes RFC 882, 883, 973.

[58] P. Mockapetris. RFC 1035: Domain Names—Implementation and Specification, Nov. 1987. Internet Standard. Obsoletes RFC 882, 883, 973.

[59] D. Moore, C. Shannon, D. J. Brown, G. M. Voelker, and S. Savage. Inferring Internet denial-of-service activity. *ACM Trans. Comput. Syst.*, 24(2):115–139, May 2006. Earlier: USENIX Security 2001.

[60] S. Northcutt, M. Cooper, M. Fearnow, and K. Frederick. *Intrusion Signatures and Analysis*. New Riders Publishing, 2001.

[61] S. Northcutt, J. Novak, and D. McLachlan. *Network Intrusion Detection: An Analyst's Handbook (2nd edition)*. New Riders Publishing, 2000.

[62] A. Ofir and F. Yarochkin. ICMP based remote OS TCP/IP stack fingerprinting techniques. In *Phrack Magazine*. 11 Aug 2001, vol.11 no.57, file 7 of 12, `http://www.phrack.org`.

[63] G. Ollmann. The Pharming Guide: Understanding and Preventing DNS-Related Attacks by Phishers. Whitepaper, available online, July 2005.

[64] V. Paxson. Bro: A system for detecting network intruders in real-time. *Computer Networks*, 31(23-24):2435–2463, 1999. Earlier version in: 1998 USENIX Security Symp.

[65] V. Paxson. An analysis of using reflectors for distributed denial-of-service attacks. *Computer Communication Review*, 31(3):38–47, 2001. See also: Steve Gibson, Distributed Reflection Denial of Service, 22 Feb 2002, online.

[66] D. C. Plummer. RFC 826: An Ethernet Address Resolution Protocol, Nov. 1982. Internet Standard.

[67] N. Provos and T. Holz. *Virtual Honeypots: From Botnet Tracking to Intrusion Detection*. Addison-Wesley, 2007.

[68] T. H. Ptacek and T. N. Newsham. Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection. January 1998, available online.

[69] A. Ramaiah, R. Stewart, and M. Dalal. RFC 5961: Improving TCP's Robustness to Blind In-Window Attacks, Aug. 2010. Proposed Standard.

[70] E. Ramirez-Silva and M. Dacier. Empirical study of the impact of Metasploit-related attacks in 4 years of attack traces. In *Asian Computing Sci. Conf. (ASIAN)*, pages 198–211, 2007. Springer LNCS 4846.

[71] M. J. Ranum, K. Landfield, M. T. Stolarchuk, M. Sienkiewicz, A. Lambeth, and E. Wall. Implementing a generalized tool for network monitoring. In *Large Installation Sys. Admin. Conf. (LISA)*, 1997.

[72] M. Roesch. Snort: Lightweight intrusion detection for networks. In *Large Installation Sys. Admin. Conf. (LISA)*, pages 229–238, 1999. For official documentation see `https://www.snort.org`.

[73] C. Rossow. Amplification hell: Revisiting network protocols for DDoS abuse. In *Netw. Dist. Sys. Security (NDSS)*, 2014.

[74] D. Safford, D. L. Schales, and D. K. Hess. The TAMU security package: An ongoing response to internet intruders in an academic environment. In *USENIX Security*, 1993.

[75] K. Scarfone and P. Mell. Guide to Intrusion Detection and Prevention Systems (IDPS). NIST Special Publication 800–94, National Inst. Standards and Tech., USA, Feb. 2007.

[76] E. Skoudis and T. Liston. *Counter Hack Reloaded: A Step-by-Step Guide to Computer Attacks and Effective Defenses (2nd edition)*. Prentice Hall, 2006 (first edition: 2001).

[77] R. Sommer. Bro: An open source network intrusion detection system. In *17th DFN Workshop on Communication Networks*, pages 273–288, 2003.

[78] R. Sommer and V. Paxson. Enhancing byte-level network intrusion detection signatures with context. In *ACM Comp. & Comm. Security (CCS)*, pages 262–271, 2003. (Compares Bro to Snort).

[79] S. Staniford, J. A. Hoagland, and J. M. McAlerney. Practical automated detection of stealthy portscans. *Journal of Computer Security*, 10(1/2):105–136, 2002.

[80] P. Uppuluri and R. Sekar. Experiences with specification-based intrusion detection. In *Reseach in Attacks, Intrusions, Defenses (RAID)*, pages 172–189, 2001.

[81] J. Voas and G. McGraw. *Software Fault Injection: Inoculating Programs Against Errors*. Wiley, 1998.

[82] S. Weiler and D. Blacka. RFC 6840: Clarifications and Implementation Notes for DNS Security (DNSSEC), Feb. 2013. Proposed Standard.

[83] J. White, T. Fitzsimmons, J. Licata, and J. Matthews. Quantitative analysis of intrusion detection systems: Snort and Suricata. In *Proc. SPIE 8757, Cyber Sensing 2013*, pages 275–289. Apr 30, 2013.

[84] D. Whyte, P. C. van Oorschot, and E. Kranakis. Tracking darkports for network defense. In *Annual Computer Security Applications Conf. (ACSAC)*, pages 161–171, 2007. Earlier version: USENIX HotSec 2006 (Exposure maps: Removing reliance on attribution during scan detecion).

[85] M. Zalewski. p0f v3: passive fingerprinter. README file, 2012. `http://lcamtuf.coredump.cx/p0f3/README`.