CHAPTER

1

Fundamentals of Secure Proxies

The evolution of the secure proxy is a reflection of the evolution of the web. The proxy began as a gateway that bridged content that was processed and managed by various information systems, and served that content to the open web during the early days of Internet web construction. The term *web proxy server* was given to this general intermediary to reflect its main duty at the time, namely, translating web requests from the Internet to representations that could be understood and fulfilled by different internal systems, and vice versa.

The web has evolved, expanded, and flourished from a content-centric, information-sharing system into an elaborate ecosystem for commerce, an acculturation establishment for Millennials, and a foundation for modern-day cloud computing. The web browser has become the instrument that unlocks all of the wealth the web offers. The fundamental web protocols and technology, such as HTTP, SSL, HTML, XML, Java, and JavaScript, have been amalgamated into a complex conduit, which faces relentless assaults from nefarious forces that try to subvert it for profit. However, private intellectual properties and confidential data hosted in private and protected networks are accessible through a browser over secure connections across the Internet. The web has also been adopted as a system of portals for managing critical infrastructures at municipal, state, and national levels. Consequently, the user and the browser have become attack vectors for breaching corporate as well as national security.

The web proxy has evolved from a content gateway into an essential security gateway that focuses on users, applications, and content. The security proxy

differs from a generic web proxy in that the secure proxy can interpret and intercept more application protocols than just HTTP. Secure proxies, especially when deployed in enterprise environments, serve as both protectors and enablers so that their user community can benefit from the web while minimizing the risk of being victimized by malware delivery networks.

Security Must Protect and Empower Users

The rise of the Internet becoming the foundation of the new era in commerce, culture, communication, education, entertainment, and technology was invasive, with profound impact on our social behaviors. It is now ubiquitous and is an indispensable element of both professional and personal life. At the time of the Internet boom, even long before the advent of mobile computing, the line between work hours and personal time was indistinguishable. With the introduction and rapid adoption of smart phones and tablet computing, there is no longer a distinction between a personal and a work-related computing device. This situation is particularly true for employees who travel a great deal as part of their job functions. For this mobile workforce, a regular laptop computer is typically installed with both personal software and work-related applications. They work wherever and whenever they can while roaming through airports and hotels. The expansion of both the Internet and affordable residential broadband networks has enabled many employees to work from home. Similar to the mobile workforce, the home computer serves as both a personal entertainment and productivity platform and a professional instrument that performs corporate-related job functions. Both computing paradigms raise a dilemma: a well-formed physical perimeter that isolates and guards the enterprise network with traditional IT governance is nonexistent. This lack of separation of personal, private information from corporate intellectual property and data on the same storage device can be a liability for both the employee and the employer.

The Birth of Shadow IT

Business applications are migrating from locally hosted solutions within the enterprise to a cloud-hosted collaborative model. This transition means enterprise users are accessing business-critical applications through their web browser, over the standard web protocols, using a diverse range of computing devices that may not be owned or managed by the enterprise. Consequently, the traditional security practice of the allow-or-deny-all approach is inadequate in managing today's complex web-oriented computing paradigm.

In today's enterprises, users demand the ability to choose from a vast number of applications that they can utilize to maximize their productivity when performing their duties, while at the same time leveraging those same applications

for personal objectives. Because enterprise IT and network access policies tend to be restrictive, many user-chosen applications may not be authorized for use in an enterprise network due to security risks, such as the type of information the application gathers and transmits to entities that are external to the enterprise. The servers that the application communicates with may also be easily compromised by attacks. For example, many organizations prevent users from running Dropbox for file sharing for fear that company-related confidential documents may be leaked as a result of unintentional but careless actions. Another typical restriction is that users are forbidden from running any application that participates in a peer-to-peer (P2P) network. This prohibition is likely the precipitant of the Digital Millennium Copyright Act that was signed into law in the United States in 1998. From an enterprise perspective, any copyright infringing material that is stored and that transits the enterprise network presents serious legal liabilities and ramifications. Application software may be produced by various publishers that range from large commercial vendors to independent software developers. An enterprise may exclude an application from its permissible list based on the publisher and its reputation.

One of the fundamental evolutions that have taken place in the enterprise IT environment is the emergence and growth of *shadow IT*. Employees' desire to circumvent IT restrictions led to the use of shadow IT. In the previous example, if Dropbox were blocked by IT policies, then employees would find alternative mechanisms and tools to share files, thus resulting in shadow IT usage. Consider the following example: sales engineers (SEs) travel constantly, and they need to share files with other SEs, employees, and their customers. E-mail systems implement file size limits such that large files cannot be transferred over e-mail. Because Dropbox has been blocked, these SEs may experiment exhaustively with Box.com, Wuala.com, Google Docs, Google Drive, TeamDrive, SugarSync, OneDrive, CloudMe, or Amazon Cloud Drive until they find a solution that is capable of penetrating the IT security net.

Internet of Things and Connected Consumer Appliances

The *Internet of Things* (IoT) refers to uniquely identifiable embedded devices that are networked, which are reachable and manageable through the Internet infrastructure. These embedded devices have proliferated and matured beyond just smart sensors to more intelligent applications such as smart building and home automation systems. Google's $3.2 billion acquisition of Nest in January 2014, followed by Samsung's acquisition of SmartThings in August 2014, offers a glimpse into market developments that are shaping the future of the IoT. Much of this IoT can now be accessed and controlled through applications on popular mobile devices such as the Apple iPhone and iPad and Google's Android-based gadgets. For example, a homeowner can use the ADT Pulse app on their iPad to activate or deactivate their ADT home alarm system, check motion sensors,

and watch live video feeds from various video cameras that have been installed in their home. The Tesla Model S iPhone app allows a car owner to track their car's location or start and stop electrical charging of the vehicle.

The IoT has met little resistance as it has gradually become engrained into our daily lives, in what appears to be almost a seamless integration, because convenience and ease-of-use have replaced security at center stage. Securing the IoT is a complex problem. Two main aspects of defense include protecting the IoT device and securing the access channel. The access channel includes the communication between the device and its peer (commonly known as *machine-to-machine communications* [M2M]), and the communication between the device and its operator. Because it is embedded, the IoT device has limited computing power and resources, which limits the device's ability to run sophisticated software such as a virus scanner. Such an embedded device is typically powered by either a custom operating system (OS) or a special variant of a known OS. An embedded OS generally lacks security software that is commonly found in a desktop OS, for example, antivirus software. At the time of this writing, the popular Apple iOS has been on the market for over seven years, yet antivirus software for the iPhone and iPad is limited in both variety and functionality; more importantly, such antivirus software is rarely installed by iOS users. Considering the iPhone is by definition an embedded device, the prospect of antivirus and anti-malware software finding its way into the iPhone as a standard application seems impossible, at least for the next few years.

Running an embedded OS implies that software patches that fix security vulnerabilities may not be released at a regular interval, if such a practice exists at all. Even when such a firmware patch mechanism exists, in most cases the patch process relies on the user to be diligent in exercising security practices, and such a demand on the general population is simply unrealistic. Therefore, these factors indicate that IoT devices can become popular attack targets and can be compromised with relative ease. Once such an IoT device is hacked, user information may be retrieved and the device can in fact cause physical harm to its owner; for example, a hacker shutting off a smoke detector during a house fire can cause physical injury or damage. These IoT devices can also be turned into zombies and become part of a large botnet, which can be commandeered into participating in a planned distributed denial-of-service (DDoS) attack against another target.

Other types of consumer electronic appliances, such as the Sony PlayStation 4 (PS4) and Internet-ready HDTVs, are network-capable and face security threats similar to those faced by IoT devices. An Internet-ready HDTV may not allow its owner to browse and surf the web; however, it permits its owner to log in to Facebook and update their Facebook status through the built-in application. The Facebook account information could be stolen if the Internet-ready HDTV is hacked. The Sony PlayStation owner can purchase games at the PlayStation

Store. The PlayStation Network user account information includes the account holder's birthday and contains a stored credit card number. The user credential to log in to the PlayStation Network to play multi-player online games can be stolen by an attacker who has compromised the PS4, thus putting the account holder's privacy at great risk.

Conventional Security Solutions

The *security posture* of an organization refers to the role security plays in the organization's business planning and its business operation. The security posture encompasses the design and implementation of a well-defined security plan. The security plan is comprised of technical solutions including technology in terms of software, hardware, and services that can be implemented at end points and within the network. The security plan also includes non-technical aspects: employee education on the importance of security as an essential element of business operations; a definition of policies on employee conduct and behavior that conforms to corporate security governance; a definition of policies for achieving regulatory compliance; and a definition of procedures and guidelines on responding to security incidents, both internally and externally.

In essence, the security posture refers to how an organization views security: as a business enabler or as a hindrance and an inconvenience to its operational efficiency. An organization's security posture dictates its practices of security and determines the effectiveness of its security implementation. In today's information age, the availability and timely accessibility of information are important keys to an enterprise's success. Enterprises strive to foster innovation by harnessing the wealth of information capital available on the Internet, while at the same time maintaining an energized and engaged workforce.

Security should afford users the freedom to explore and harvest the riches of the Internet, and alleviate the fear of becoming victims of cyber threats. Existing threats change and new ones emerge as the web evolves; therefore, security postures cannot remain static for long and need regular assessment. It is essential to have an in-depth knowledge of available security solutions, and an understanding of the strengths and the weaknesses of each solution in order to perform assessments such as vulnerability testing, penetration testing, and standards-based auditing. Understanding security technologies is the key to implementing the layered defense that is now mandatory in securing users and enterprise networks.

Traditional Firewalls: What Are Their Main Deficiencies?

The firewall, the most commonly known and referenced security device, was once the motif of security-related conversations and continues to be an

essential element of any network security design. The traditional firewall is still the first line of defense. However, the growing body of threats have long surpassed the capabilities of the traditional firewall. The security landscape is now cluttered with acronyms such as unified threat management (UTM), deep packet inspection (DPI), intrusion detection system (IDS), intrusion prevention system (IPS), secure web gateway (SWG), web application firewall (WAF), next-generation firewall (NGFW), application intelligence and control (AIC), and many more. These acronyms create the perception that perhaps the security threats are largely under control, yet in reality, adroit, menacing malware crafters flourish in the shadows, and security battles rage on with growing ferocity and intensity. The various technologies that are behind the acronyms add confusion and inundate the security implementers with overlapping solutions. These overlapping solutions obscure the deficiencies in the core technologies, and this lack of clarity results in the construction and deployment of inadequate defenses.

The deficiencies of the traditional firewall lie in its inability to examine the packet payload, especially when content is encrypted. The traditional firewall examines layer-2 (L2) to layer-4 (L4) packet header information, such as source and destination IP addresses, L4 protocol type, and L4 source and destination port information, as depicted in Figure 1-1. A firewall rule can be written to compare any header field or bits against any specific values and can define instructions for the firewall to apply one or more actions accordingly. For example, a firewall rule can state, "If an incoming packet is a TCP connection initiation frame (i.e., the TCP header contains the SYN flag bit), then transmit a TCP RESET frame back to the sender." Basically, this firewall rule blocks all incoming TCP connection requests.

Here is another example of a firewall policy: "If the source IP address is 10.9.44.108, the protocol is TCP, and the destination port is 6881, then discard the packet." TCP port 6881 is commonly used by the BitTorrent program for P2P traffic. Enterprise firewalls block this port to prevent employees from downloading questionable content and consuming valuable network bandwidth. This firewall policy can be problematic in actual deployment. First, the popularity of BitTorrent has enabled its adoption by various organizations for legitimate use, for example, by communities that distribute open source software releases. In such cases, blocking TCP traffic on port 6881 would preclude users from permissible use of BitTorrent and, in some cases, would interrupt the only distribution channel for a specific open source project. Therefore, the content of a specific BitTorrent session, instead of simply the destination port, should determine whether such a session is permitted. However, a traditional firewall does not have the ability to perform content analysis. Second, BitTorrent uses port 6881 when the port is available; otherwise, port 6882 and subsequent ports are tried until an unused port is found. As such,

port 6881 can be occupied by traffic belonging to an admissible application. The firewall cannot determine which application originated the traffic to port 6881. Consequently, simply blocking port 6881 could disrupt a permissible application from its normal operation. Figure 1-2 illustrates the port-sharing dilemma that confuses a firewall.

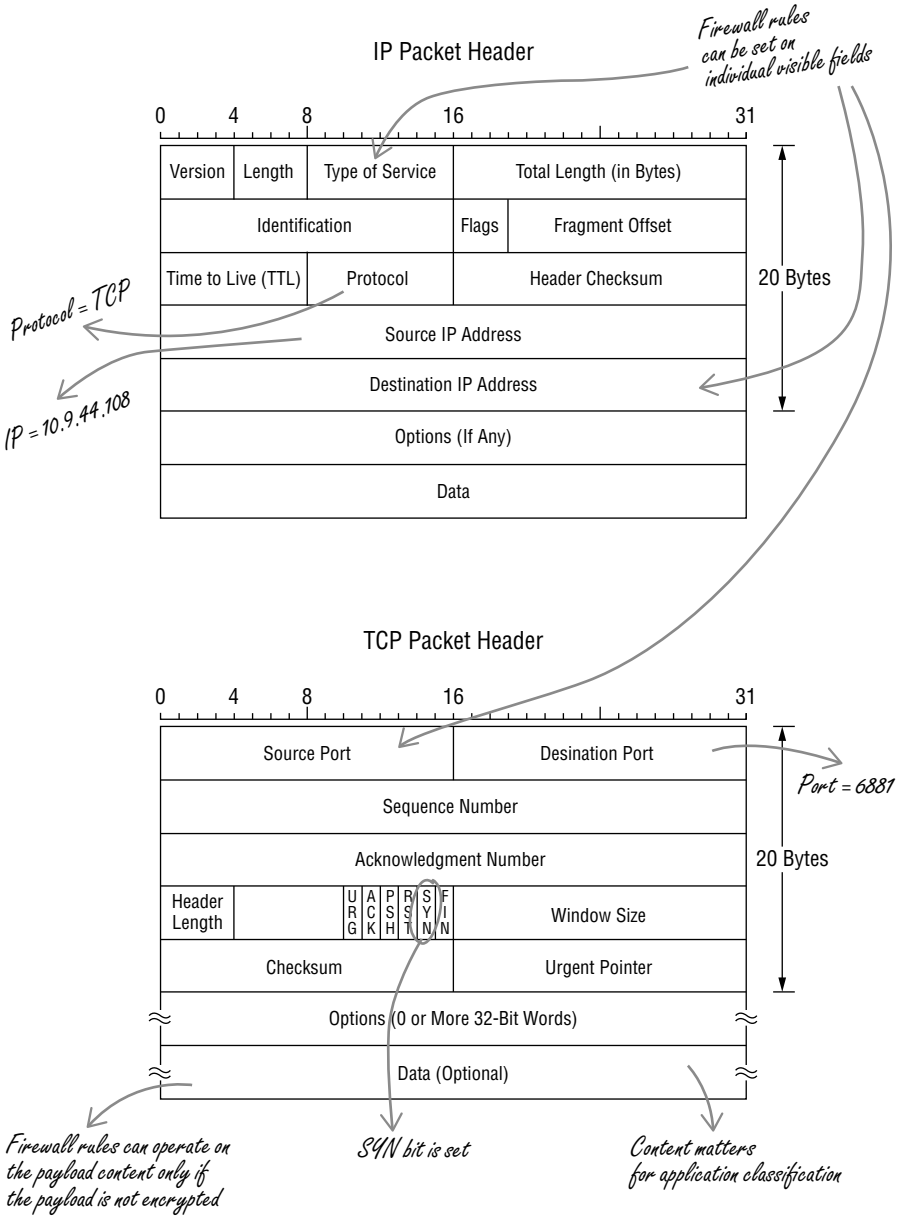

Figure 1-1: TCP/IP Headers for Firewall Processing

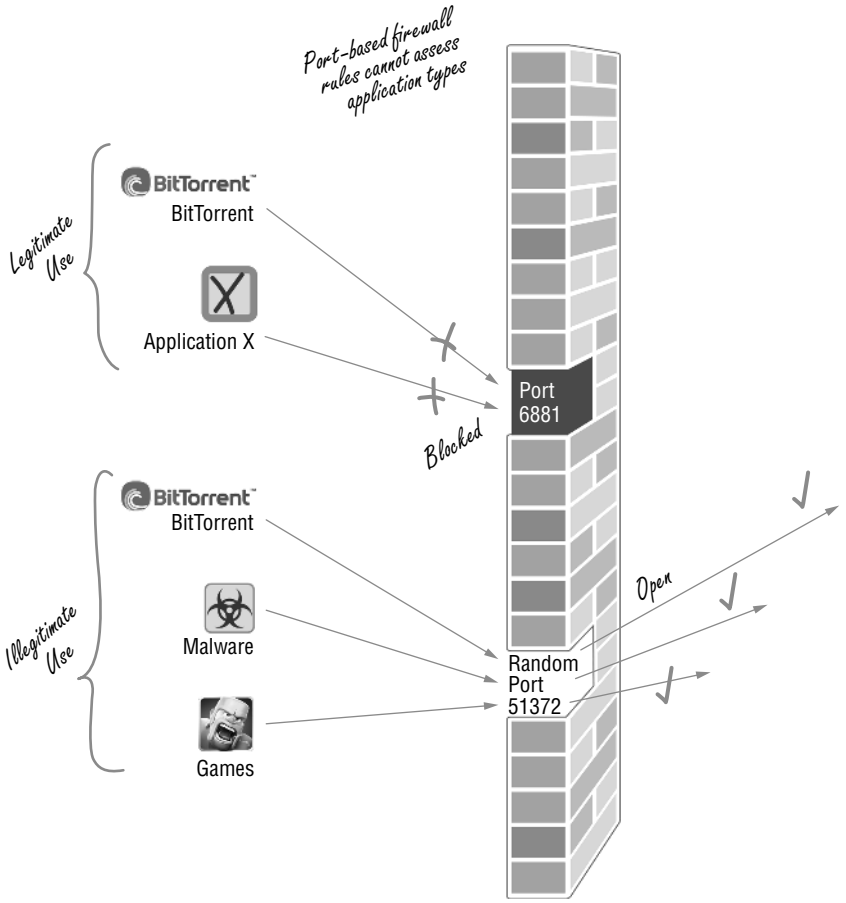

Figure 1-2: Port Overloading

This example depicts a serious deficiency in a firewall, where it cannot block a malicious application that runs over a non-default port. Consider another example where a firewall permits outbound HTTP traffic: traffic destined to TCP port 80 is permissible because otherwise users will not be able to access any websites on the Internet. Malware writers have common knowledge of well-known destination ports that are allowed by firewalls. They create their malware to transmit on these ports to circumvent the firewall because they know the firewall is incapable of distinguishing HTTP traffic from non-HTTP traffic just by examining the packet headers. This example exposes another serious deficiency in the firewall: it cannot block a malicious application that transmits over allowed well-known ports. We can make another observation in Figure 1-2, that malware can perform port hopping to discover "holes" in the firewall. The malware can transmit in the dynamic port range, beginning with a high-value port, and increment the port number by 1 until it successfully receives a response from its intended peer.

Every packet that passes through a firewall will match at least one firewall rule. The firewall understands the connection concept, whether it is a TCP connection or a UDP connection. A common firewall feature is that it keeps stateful information on TCP connections and UDP sessions. This stateful information cache, known as the *state table*, reduces the firewall workload and increases firewall scalability. For example, when the first packet of a TCP connection is seen by the firewall (in this case a TCP SYN packet), the firewall executes its rules against this TCP SYN packet, which results in a firewall action. This resulting action and the TCP connection information (connection 4-tuple and the TCP header) is then stored into the state table. Figure 1-3 shows an example of a firewall state table.

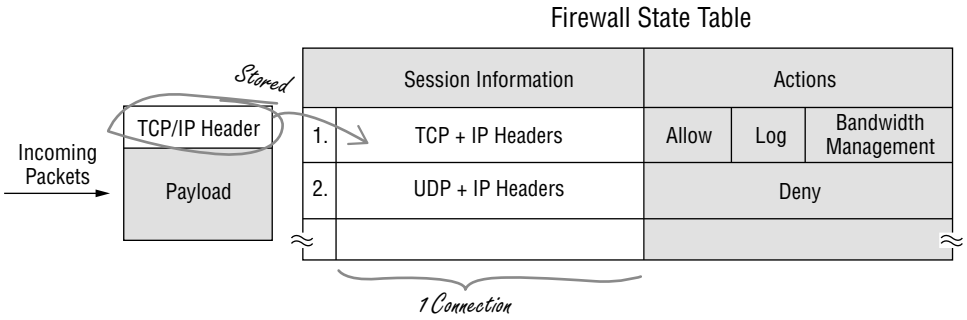

Figure 1-3: Firewall State Table

Instead of running through all of the rules repeatedly on the subsequent packets from that same TCP connection, the firewall can consult the state table directly and obtain the action quickly. This is the reason why the firewall is also known as the *stateful packet inspection* (or simply stateful inspection) firewall. Examples of firewall actions include allowing traffic to pass, denying traffic by silently dropping the packets, denying a TCP connection by generating a TCP RESET protocol packet, and generating connection logs. Each entry in the state table contains minimal information that represents a connection. Packets belonging to connections that are permitted by the firewall transit the firewall unmodified. In practice the firewall updates its state table entries using only information from the packet headers.

Firewall with DPI: A Better Solution?

A new breed of firewalls—let us call them the second-generation firewall (SGFW)—incorporated DPI technology to address the problem of identifying what application generated the traffic in question. With DPI, the packet payload is scanned for specific known patterns, also known as *signatures*, which can

potentially identify applications. We say potentially identify because of the challenges of application identification using static patterns, which is a topic we cover in Chapter 7. These SGFWs enable administrators to specify and enforce policies that are based on application names and types, more than just IP addresses and port numbers. In addition, by integrating user authentication information that provides mapping between a user and a specific IP address, some of these SGFWs extend the policy coverage to enforce policies that are defined around individual users.

When an SGFW performs DPI to scan a flow for an application signature, as Chapter 7 covers, multiple packets may have passed through the firewall before an identification can be made. These leaked packets may have already provided the black hats with useful information to further their attacks. Because DPI relies on pattern matching, prevalent in the form of regular expression matching, the operation is computationally intensive. The performance impact of DPI on firewall throughput determines how much content is scanned on a per-packet basis and how much stateful data is kept for correlation when conducting analytics. As such, firewalls with a DPI engine obtain scalability through a hardware-based regex processor that typically increases the cost of the overall solution. Firewalls in general had become commoditized in the late 1990s. The cost factor determined whether a firewall had a built-in DPI engine and what capabilities that DPI engine offered.

There are many issues that render DPI ineffective. First, DPI does not work on an encrypted payload. An encrypted payload is indistinguishable from random byte streams and thus cannot match any known patterns. Other data obfuscation techniques, such as compression, encoding, and tunneling, can achieve the same effectiveness in defeating DPI.

Second, firewalls with DPI engines cannot modify the content even when malicious content has been identified: entire packets must be discarded that will impact the overall sessions. Here is the reason why: as Figure 1-1 illustrates, the firewall rules are formulated against the fields from the layer-3 and layer-4 headers, in this example, from the IP header and TCP header. Any alteration made to the packet can cause a TCP checksum error, unless the TCP checksum is recomputed by the firewall. Because TCP checksum covers all of the payload data, re-computing the TCP checksum is an expensive operation. The firewall may need to perform packet reassembly due to IP layer fragmentation, thus incurring additional processing overhead. Revising the TCP checksum is insufficient and will not work in cases where, for example, an Internet protocol security authentication header (IPSec AH) is employed to verify end-to-end message integrity; in other words, any modification of the original message by intermediate systems, in this case the firewall, would fail the AH integrity check at the final destination.

Although an SGFW can provide better visibility by recognizing certain unencrypted applications by means of DPI, its enforceable actions are still as limited as

the traditional firewall. This coarse protection method can impede the usability of other defensive systems against sophisticated attacks.

IDS/IPS and Firewall

A firewall is the first line of defense, but it has limited visibility into the content while it makes traffic-filtering decisions. Because a firewall is commonly deployed at the ingress and egress points of a network, all traffic paths will converge and traverse through the firewall. Therefore, the performance and scalability of a firewall affects the network as a whole. For this reason, although some firewalls may incorporate a DPI engine, a firewall is designed to execute a limited set of actions against each packet, even when hardware acceleration is activated in the firewall. When an attack circumvents the firewall, an IDS extends the security coverage by inspecting the network and the end systems for evidence that corroborates whether some network events and security alerts were instigated by attacks or malicious infiltrations. An IDS generates alarms and reports to network management systems upon detecting abnormal or suspicious traffic.

An IDS examines packets for signatures that are associated with known viruses, malware, and other malicious traffic. In addition to pattern scanning within the packets, an IDS analyzes overall traffic patterns to detect anomalies and known attacks. Some examples of known attacks are denial-of-service (DoS), port scanners that search for vulnerable network services, buffer overflow exploits, and self-propagating worms. Examples of anomalies include malformed protocol packets and traffic patterns that deviate from the norm. An IDS is divided into two main categories: a network-based intrusion detection system (NIDS) and a host-based intrusion detection system (HIDS). NIDS and HIDS differ in where the IDS is deployed, which consequently dictates the types of data collected and analyzed by that specific type of IDS. Figure 1-4 illustrates an example deployment of IDS systems behind a firewall.

As shown in Figure 1-4, the NIDS is deployed inside the organization's internal networks, behind the firewall. A NIDS monitors the activities of the entire network and examines both intranet traffic and Internet-bound traffic. On the other hand, the firewall concentrates on traffic that flows into and out of the internal network to the Internet.

The traditional NIDS scans packets against a database of signatures of known attacks. Similar to the open source IDS tool Snort, each signature in the data is often implemented as a matching rule. This *signature-based IDS* runs the packets through these matching rules or signatures to detect attacks. Another approach is the *statistical-based* or *anomaly-based NIDS*, which is also known as the *behavior-based NIDS*. With a statistical-based NIDS, a profile of the network under protection is built over time, based on evolving historical data, which represents the norm of the network. Some examples of data collected and compiled into a profile that represents the network operating under normal conditions include

the following: the number of new applications that are discovered per day on the network and the average traffic volume generated by each type of application; the average number of DNS queries transmitted from a specific IP address at a given time interval; the average overall aggregate throughput of the network; and the average number of HTTP transactions issued per minute from a specific IP address. Any deviation observed by the NIDS may be interpreted as anomalies or misuse that instigates responses as defined by corresponding security directives.

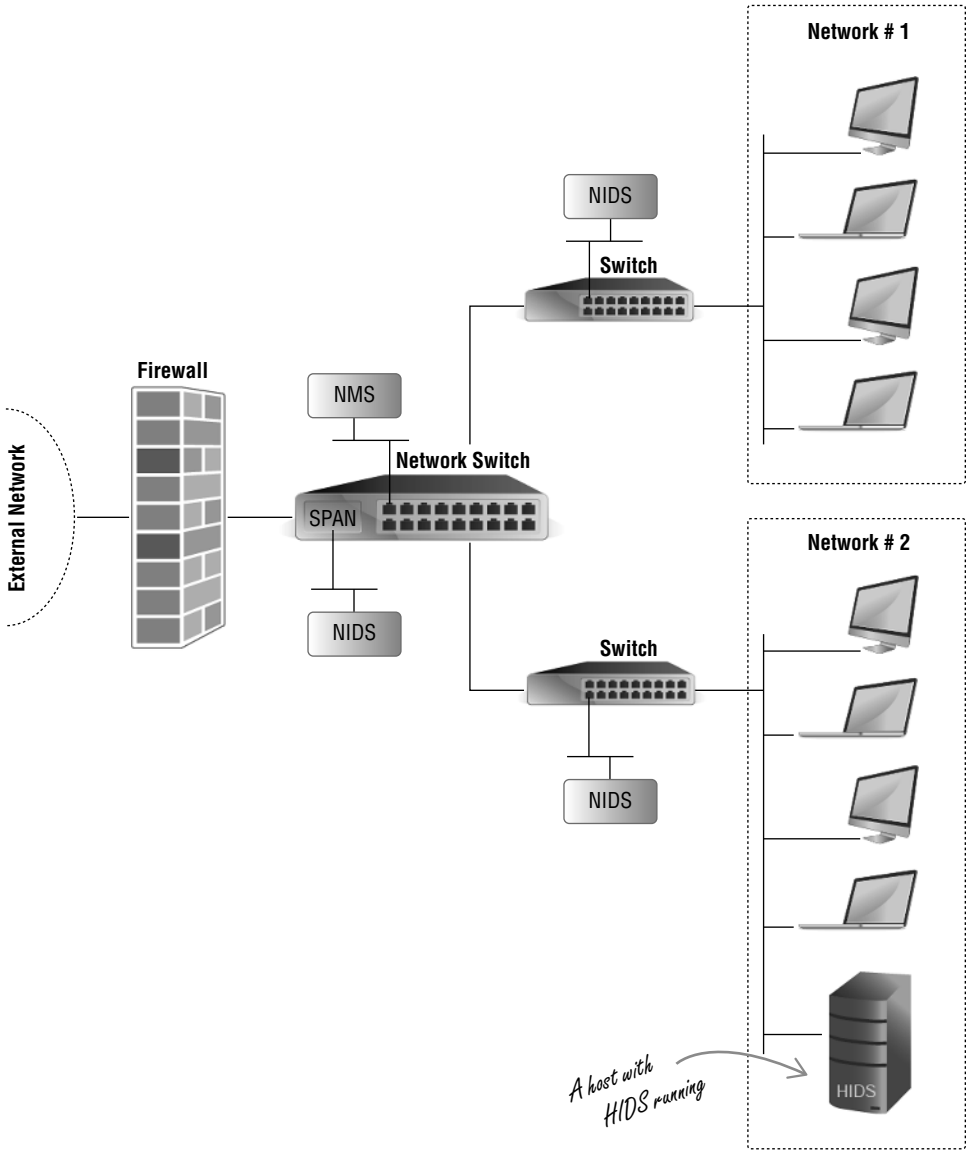

Figure 1-4: IDS and Firewall

The key to the success of a signature-based NIDS is the richness in the collection of the attack signatures. Identifying a unique and effective signature for a new attack, especially a complex attack, takes time to develop and evolve. As new attacks propagate across the networks and infrastructures, the signature-based NIDS is incapable of detecting these attacks while the new signatures are being implemented. The success of the statistical-based NIDS depends on the knowledge or heuristics of the network characteristics that are considered as normal and serve as the baseline. Establishing the boundaries of normal network behavior is challenging as the network fosters a wide range of protocols and applications and hosts a user base with a diverse spectrum of online behaviors that can trigger sporadic traffic patterns. A statistical-based NIDS can be effective against new attacks because new attacks can incite network behaviors that alarm the NIDS.

A host-based IDS (HIDS) is purposefully built, either for an operating system or for a specific application, and operates in individual end systems. The HIDS analyzes the operating system process identifier (PID), system calls, service listeners, I/O and file system operations, specific application runtime behavior, and system and application logs to identify evidence of an attack.

Firewalls are called active protection systems because a firewall is in the path of all traffic, known as inline deployment. This enables the firewall to examine live traffic, and when the firewall identifies an attack, it is capable of blocking that attack while it is in progress. In other words, upon detection, a firewall can prevent malicious traffic from reaching a targeted system.

Intrusion detection systems can be categorized as passive protection systems because an IDS is typically connected to a SPAN (Switched Port Analyzer) port on a network switch or to a network tap that duplicates packets for an entire link. While an IDS can also examine every packet, however, the packets under analysis have successfully passed through a firewall and cannot be filtered by the IDS; those packets may also have already reached the intended targets and enacted malicious activities. In other words, an IDS identifies an attack that may have already taken place, at which point the IDS begins to remediate the damage by executing countermeasures, for example, sending alerts and notifications to monitoring and management systems. The passive traffic-processing nature of an IDS implies the performance of an IDS does not have any impact on active live traffic. As such, an IDS can perform much more in-depth analysis, and correlate more data sets, than a firewall. A firewall fulfills a security role that prevents the firewall from being a replacement for an IDS.

DPI is also an integral part of the IDS. Using the open source Snort software, here is an example of a rule created by the Sourcefire Vulnerability Research Team. The rule scans for the signature of the Flashpack/Safe/CritX exploit kit that attempts to download a malicious file as part of the attack:

```
alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any (msg:"EXPLOIT-KIT
Flashpack/Safe/CritX exploit kit jar file download";
```

```
flow:to_client,established; file_data; content:"filename="; http_header;
content:".jar"; within:4; distance:24;
pcre:"/filename\=[a-z0-9]{24}\.jar/H";
metadata:policy balanced-ips drop, policy security-ips drop,
  service http;
reference:url,
  www.malwaresigs.com/2013/06/06/flashpack-exploit-kit-safepack/;
classtype:trojan-activity; sid:26892; rev:2;)
```

This example illustrates that as the IDS scans for attack signatures, it suffers from the same inherent deficiencies in the DPI engines as those found in the firewall. Evasion techniques that are used against DPI engines are also effective in defeating the signature-based IDS engines. In this example, the code in bold face is a Perl Compatible Regular Expression (PCRE). The question is, what if the exploit kit uses HTTPS to download the payload, resulting in the payload being protected by the SSL encryption so that this rule cannot be applied at all?

Unlike the passive network monitoring of an IDS, an IPS takes the active role of performing mitigation actions in real-time once attacks are detected. An IPS possesses all of the capabilities of an IDS, but an IPS is deployed physically inline in the network, which enables the IPS to drop attack packets, reset TCP connections, or activate filters to block the source of the attack. An IPS can perform other functions such as configuring dynamic policies in security devices, such as a firewall, to interrupt the malevolent maneuvering and prevent further damage to the network.

Unified Threat Management and Next-Generation Firewall

The most significant limitations of the traditional firewall are its inability to perform payload inspection and to distinguish applications. The concept of Unified Threat Management (UTM) gained visibility and momentum in 2004 to address the security gaps in firewalls, and to offer a solution for the lack of unified policy management across the various security control technology products commonly deployed together in an enterprise network. The UTM strategy is to combine multiple security features such as a firewall, NIDS, IPS, gateway-based antivirus, and content filtering into a single platform or appliance to offer multiple layers of security protection with simplified management and ease of policy implementation. The security posture continued to increase its focus on users and their applications, as the transformation in UTM took place in parallel.

Then, Gartner Inc., an information technology research and advisory company, claimed to be the first to define the *Next-Generation Firewall (NGFW)*. In its NGFW definition, the three key attributes of an NGFW are its ability to detect application-specific attacks, to enforce application-specific security policies, and

to intercept and decrypt SSL traffic. The NGFW includes all of the capabilities of the traditional firewall and incorporates the full functionality of a signature-based IPS. Another key characteristic of the NGFW is its inline deployment as a bump-in-the-wire. In addition, the NGFW can collaborate with external services to incorporate additional security-relevant data and feeds to enhance its enforcement capabilities.

The NGFW definition has a large overlap with that of the UTM. The articulated differences have limited technical merits, and the deviations are largely a result of verbiage manipulation. The NGFW concept seems to be a desired byproduct of combining the UTM with the unique features of the secure proxy. The conceptualization of the NGFW, with such a rich set of security features, processing network traffic at multi-gigabit wire speed, and without any performance degradation, would be the ultimate goal of security system design architects and developers. However, as we will illustrate in this book, firewall and proxy are fundamentally incompatible with respect to the policies each is designed to interpret and to enforce. The process and method of application classification collides with the operation of proxy interception.

Security Proxy: A Necessary Extension of the End Point

A firewall, even with UTM, performs primarily syntactical analysis of traffic that is largely signature driven and is capable of enforcing security with limited actions. Without the ability to decrypt content for analysis when encountering encrypted sessions, a firewall is confined to simply denying traffic in environments with restrictive enforcement policies. In enterprise networks, a legitimate but encrypted session could be blocked, causing discontinuity in both business and productivity. A security solution that can decrypt SSL cipher text, then feed the plain text into other security technologies, is a mandatory step to combat advanced and fast-evolving threats.

The secure proxy was invented long before NGFW was conceptualized. The demand for the secure proxy in enterprises in the financial sector, defense industry, and many others has flourished since 2002. Even the design for SSL interception was in full swing at that time. In essence, the secure proxy is the result of combining a secure web gateway with application proxies, operating with a complex and expressive policy engine at its core.

A *security proxy*, sometimes referred to as a *secure proxy* or simply a proxy unless stated otherwise, performs semantic analysis in the context of individual protocols, most importantly layer-5 to layer-7 application protocols. At the time of this writing, the majority of proxies have some capability to decrypt SSL traffic. A proxy is a

security enforcement companion to a firewall, an IDS and IPS, an enterprise-grade virus scanning appliance, analytics engines, and many other security solutions. As illustrated in Figure 1-5, a proxy is the data hub that feeds decrypted traffic to any attached companion system that performs one or more dedicated security functions. Each companion system requires a different type of input. The proxy is capable of extracting mail attachments, web URLs, and executable files from the payload and feeding these inputs to its security attachments accordingly.

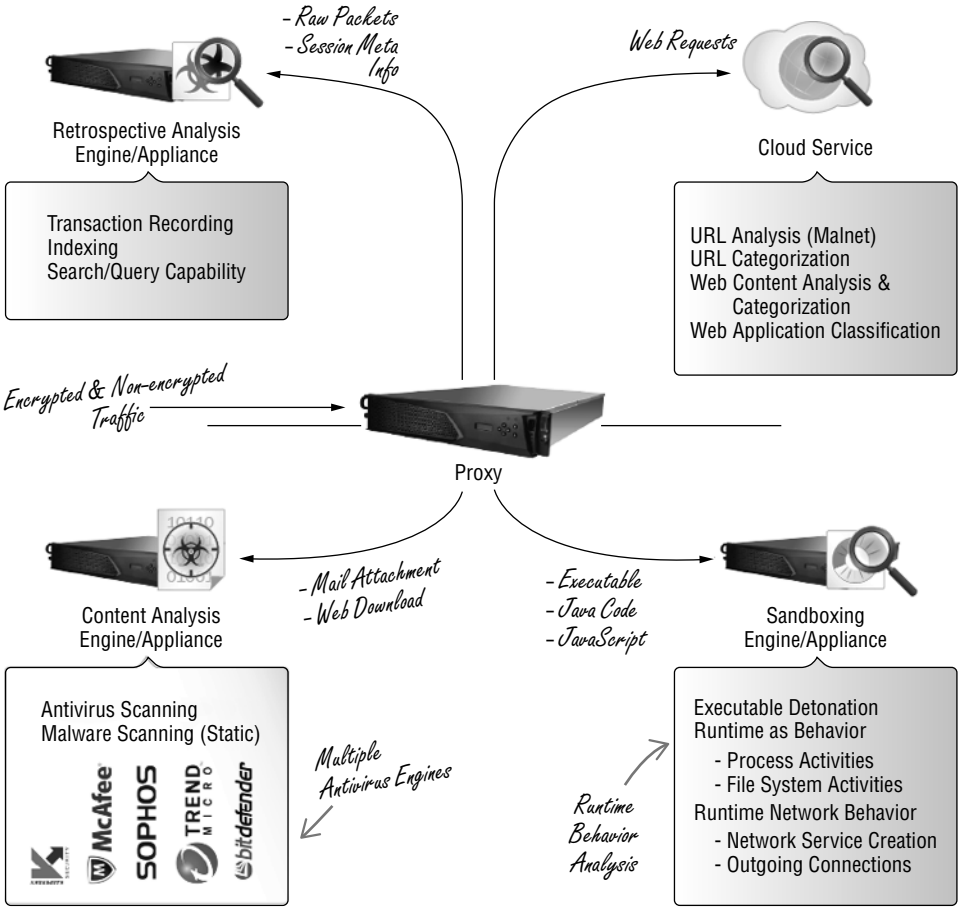

Figure 1-5: Secure Proxy as a Data Hub

A proxy is predominantly deployed inside a firewall-protected network. The secure proxy performs proxy functions beyond just analyzing the web traffic. We define web traffic as that which is carried over the HTTP or HTTPS protocols. The secure proxy can intercept more protocols than just HTTP. However, the proxy concept is best illustrated in Figure 1-6 using HTTP as an example.

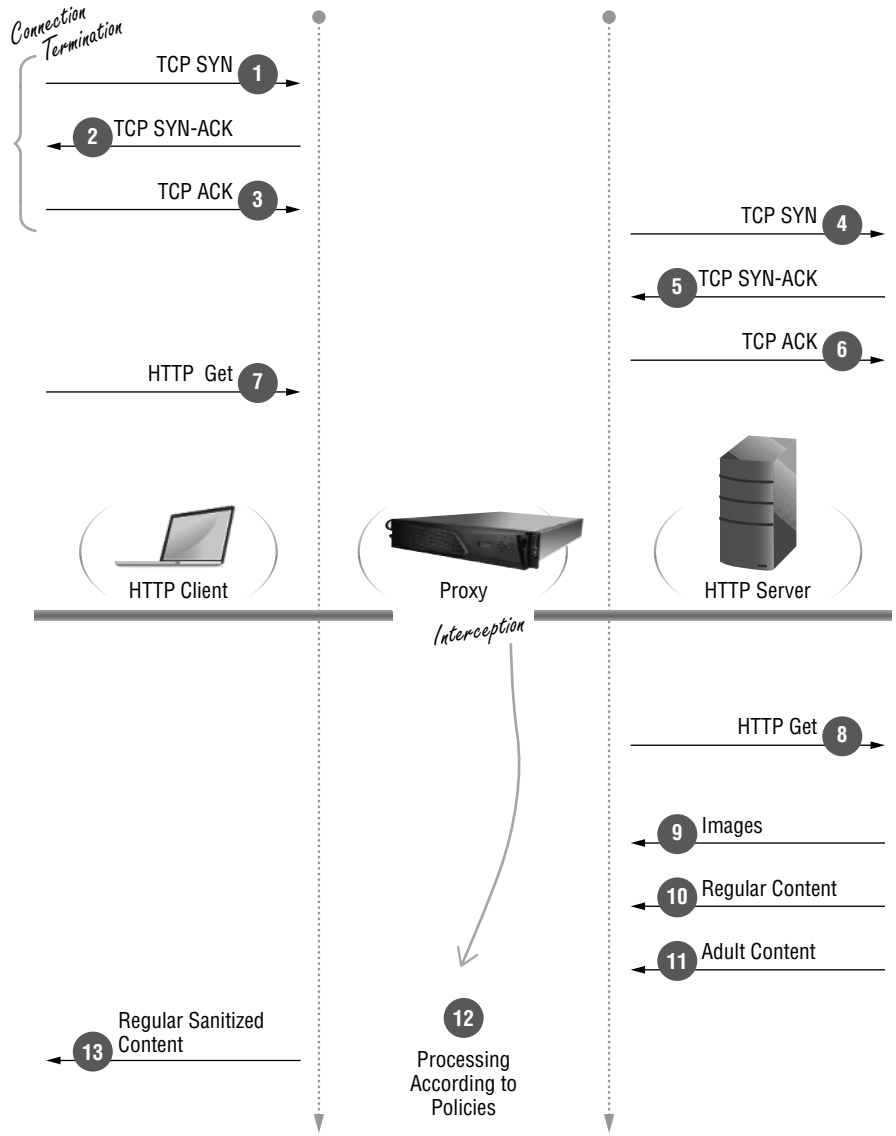

Figure 1-6: Proxy Concept

As shown in Figure 1-6, the first and most important action a proxy exerts on a connection is *interception*. Connection interception is achieved through connection *termination*. We will use the term *client* to refer to the initiator of the connection request, and the term *server* to refer to the original intended recipient of the connection request. In TCP, connection termination involves the proxy completing the TCP three-way handshake to establish the connection with the HTTP client. The next step in the interception process is for the proxy to establish

another TCP connection with the server. In this example, the original destination is Google. Once both connections have been established successfully, the next act of the interception procedure is for the proxy to receive traffic from one connection and then inject that traffic, either unmodified or transformed, into the other connection. In other words, the proxy splices the traffic between these two TCP connections. Unlike a firewall, a proxy can modify any packet and manipulate any content exchanged in these connections. In the example shown in Figure 1-6, the proxy detects the presence of adult material in the returned content and strips away that material as part of the configured policy. The sanitized content is then transmitted back to the HTTP client. This example illustrates that a proxy performs intrusive maneuvering of communication exchanges that are visible to the proxy. The payload obfuscation techniques used to defeat a firewall are proven ineffective against the proxy. Because the proxy terminates the connection, the proxy will reassemble packets and decode the content type before subjecting the session to higher-layer processing.

A real-life example of a proxy in action is free WiFi access at airports. When you connect to a WiFi access point, your computer indicates it is connected and has obtained an IP address. Yet, without opening a web browser you are unsuccessful when you try to run any application that needs the Internet. This is because you have not agreed to the terms and conditions of use. When you open the browser for the very first time, a legal agreement web page displays, and you can proceed to use the Internet once you accept that agreement. This legal agreement page displays as long as you have not accepted that agreement, regardless of how many times you choose to close and reopen the web browser. This is called a *captive portal*, which impels a user to fulfill some action, such as responding to user authentication queries. A captive portal is also used by hotels that offer Internet access, where a web page prompts the user to review and agree to the charges on first use. A web proxy (or HTTP proxy) is one of many techniques and an effective approach in implementing a captive portal.

Transaction-Based Processing

A proxy also keeps state information on the connections it processes, but unlike a firewall, a proxy participates in the connection activities, exchanging packets as a communicating peer both to the originator of the connection and to the originally intended destination. As such, there are some notable differences when comparing the firewall state table against the proxy state table, as shown in Figure 1-7. Each entry in the firewall state table represents a single connection. As Figure 1-7 illustrates, the proxy must maintain the state information that correlates the two TCP connections with the two corresponding HTTP transactions as belonging to a single user transaction that was initiated from that specific HTTP client. When the firewall processes incoming packets, only the packet headers are applied when updating connection state information. In the proxy

case, entire packets are processed and may be stored as part of the transaction state information. Remember, the proxy must receive traffic from the client-side connection and then transmit that traffic, either modified or verbatim, to the server-side connection, and vice versa.

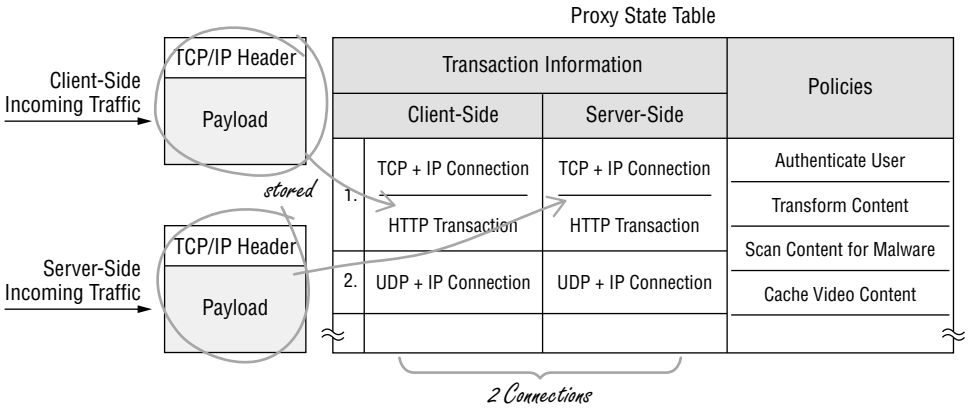

Figure 1-7: Proxy State Table

Some application-level protocols are carried over UDP. Because UDP is connectionless, the proxy must have the ability to track when a UDP-based transaction begins and ends. Similar to the TCP case, the proxy needs to create two UDP flows and update the state information according to the transaction. For example, a DNS proxy creates the UDP flow entry in its state table when it processes the DNS query message. The DNS proxy may modify the query message before sending it onward to the identified DNS server. The proxy must maintain this DNS transaction until the proxy receives the corresponding answer message, regardless of how the proxy may have processed the query content. The DNS proxy may change the time-to-live (TTL) of a particular entry in the DNS answer, or it may remove entries from the answer due to policy restrictions. The DNS proxy removes the transaction from the state table once it has transmitted the final DNS response to the client. In this example, the DNS proxy treats the DNS query and the subsequent response as the complete DNS transaction.

The Proxy Architecture

The DNS proxy example illustrates that a proxy must have in-depth knowledge about a specific application protocol and also about its structure and operational detail in order to perform proper interception. Depending on the protocol in question, examples of this knowledge may include the following: whether a centralized directory service is involved in locating a service or peer; how the connection is established between the communicating peers;

the type of authentication mechanism employed; the encryption methods that are available and the negotiation approach; the types of requests and the associated payloads; the types of responses and the associated payloads; and how the various content is encoded and transported. For every application protocol that may exist in an enterprise network, if that application requires management other than the simple allow-or-deny type of enforcement, then there exists a specific proxy designed and built for that application protocol, performing the necessary interception and processing requests and responses according to defined security policies.

A *secure proxy* is an appliance that incorporates various application proxies into a single platform, with the proxies collaborating with one another to process application traffic and enforce policies. Figure 1-8 shows the high-level architecture of a proxy. As shown in the figure, the proxy is comprised of three main components:

- The protocol detection and application classification engine (PACE)
- Various protocol and application proxies
- The policy engine

When the proxy receives a transaction for the first time, the PACE dispatches traffic to the proxy according to the default port designation. The PACE first terminates the connection and then transfers the established connection to a specific proxy. The connection transfer is done through a dispatcher that has the knowledge of the various well-known ports and the designated proxies. As shown in Figure 1-8, the ports table maps a proxy to a well-known port. For example, the DNS proxy is assigned to handle port 54, and the SSH proxy is assigned to port 22. Because malicious traffic attempts to evade the firewall by utilizing the well-known ports, a specific proxy must accurately detect if a given traffic flow is in fact from the protocol that the proxy is built to handle. For example, port 443 is used by HTTPS sessions. The first set of data packets exchanged on the established connection must be the SSL handshake traffic. Each proxy scans the payload for specific known signatures belonging to the protocol or application in question. In the HTTPS example, when the SSL proxy accepts the connection from the dispatcher, the SSL proxy expects to receive the SSL `ClientHello` record, which begins with the byte pattern: `0x16 0x03 0x01 0x02 0x00 0x01 0x00 0x01 0xfc 0x03 0x03`. The SSL proxy redirects the transaction back to the PACE to perform further protocol detection if it cannot interpret the payload as SSL traffic.

The keen reader will now oppugn some of the statements just made in the last paragraph: if a proxy scans the payload for specific known signatures, then how is the proxy different from a firewall or IDS system with a built-in DPI engine? How can the SSL proxy scan for a predefined byte stream in encrypted traffic? And how can a proxy scan encrypted content?

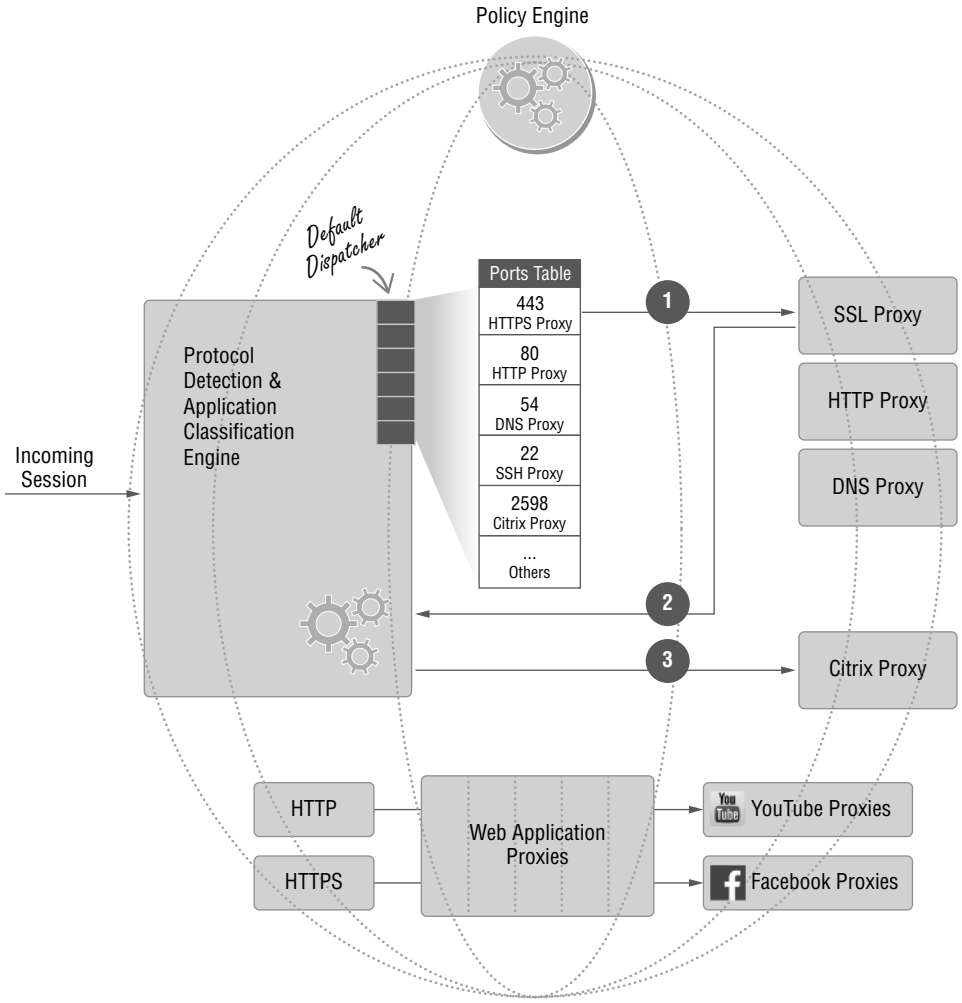

Figure 1-8: Proxy Architecture

The first question, asked differently, is: if the PACE has the ability to classify the traffic against specific protocols and applications, is the PACE duplicating work that is performed by the proxies? The answer is that there is no work duplication, and here is the reason why: The HTTP protocol is an ASCII protocol. The PACE parses the payload for keywords such as "HTTP/", "GET", "Content-type", "Content-length", "Accept:", and "<HTML>". Together these keywords provide a high probability that the payload belongs to an HTTP request. Therefore, the PACE forwards the transaction to the HTTP proxy. Once the HTTP proxy receives the transaction and encounters the keyword "GET", it interprets this keyword as a method and parses the subsequent bytes to look for the parameter (such as a filename) for this method.

The PACE parses the payload and classifies the traffic according to the HTTP *protocol syntax*. The HTTP proxy understands the full *semantics* of the HTTP protocol and, as such, can enforce security policies that are written specifically around the HTTP protocol. For example, a policy rule can be

```
Deny if (http.method == GET) and (Host == www.adserver.com)
```

Each proxy has its own nuances. In each proxy, security policies are designed to operate on specific aspects of a protocol. Therefore, as this simple example demonstrates, the proxy cannot apply any security policies unless it can, as a first step, accurately detect the application or the protocol in question. As we will show in more detail, a proxy identifies an application by specific payload signatures and according to the sequence of events and exchanges that must take place, combined with the runtime behavior of the application.

Application classification and protocol detection may require multiple packets before reaching the conclusion on what the application or protocol is. Each enterprise has a different level of stringent policies on how many packets can be permitted to flow through the proxy unrestricted before the proxy interrupts the flow and closes down the transaction. Therefore, the PACE and the specific proxy must work collaboratively to quickly identify the traffic. If the proxy cannot classify the protocol, the PACE can choose from two main options when proceeding: the first option is for the PACE to stop and end the transaction immediately; the second option is for the PACE to re-inject the packets received from the initiator connection into the other connection unmodified. In either case, the PACE may log this transaction for the administrator. A proxy that chooses the second approach is concerned more with preventing communication disruption than with strict security enforcement where packet leaks are to be kept to a minimum.

The policy engine executes in the context of all modules and components and across all layers between layer 2 and layer 7. The policy engine is covered in detail in Chapter 3.

The SSL messages transmitted at the early stages of the handshake exchange are not encrypted. These messages contain sufficient detail for an SSL proxy to determine if it will perform interception on a specific transaction. Other proxies rely on the SSL proxy to decrypt cipher text and offer these other proxies the plain text for further analysis and processing.

SSL Proxy and Interception

The remaining discussion in this chapter will focus on the HTTPS proxy because it depends on the SSL proxy. The SSL proxy is challenging to design, implement, and deploy not only because of privacy concerns but also because the SSL proxy performs identity emulation, and it must enforce authentication and the trust model, which are essential in secure communication. Figure 1-9 illustrates two

main SSL interception scenarios. As shown in the figure, the proxy must masquerade as the server when communicating with the client. Similarly, the proxy must assume the identity of the client when it connects to the server. In essence, the proxy acts as the man-in-the-middle (MITM), and if the proxy does a good job, its presence remains undetected throughout its operational lifetime. The proxy can succeed in interception only if both the client and the server trust the proxy.

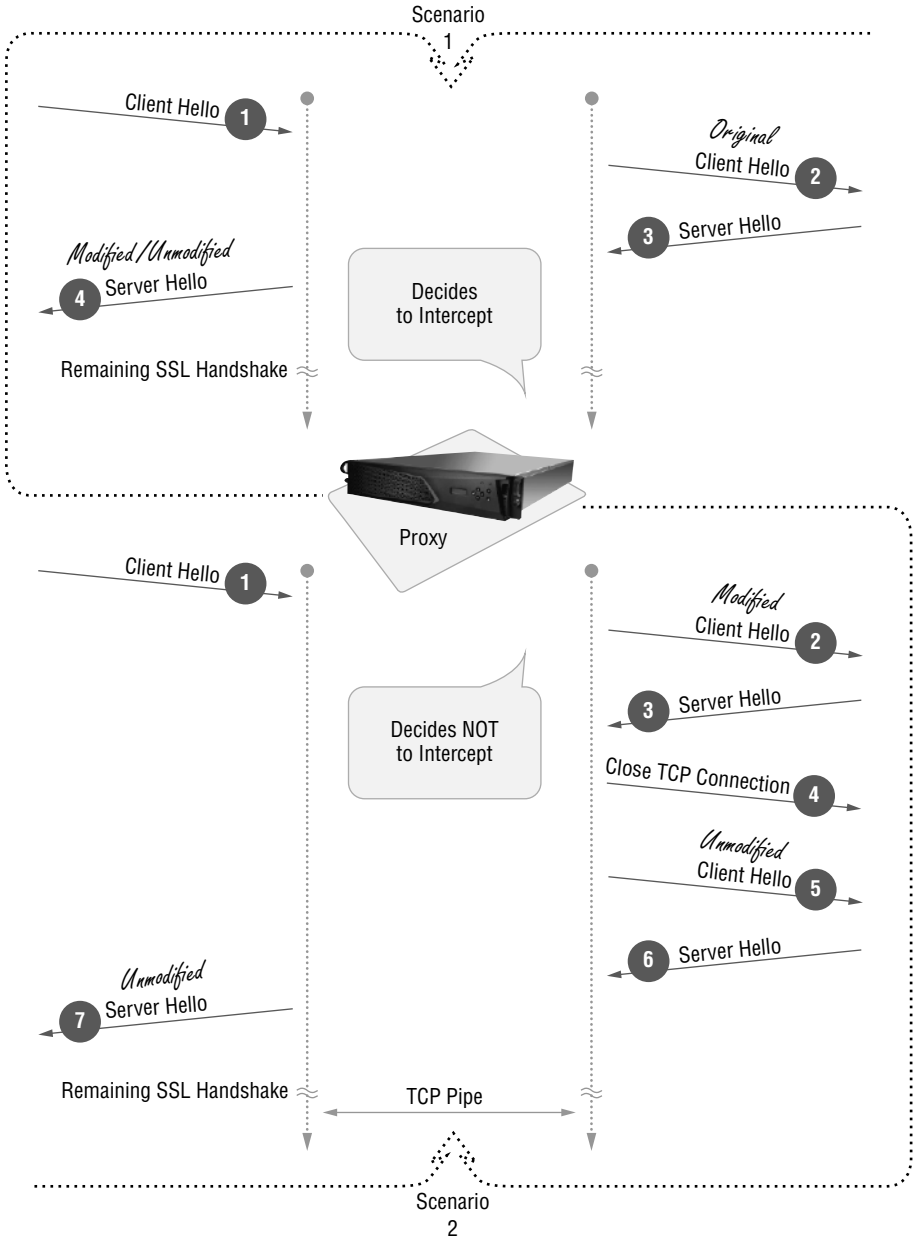

Figure 1-9: SSL Interception

In the first scenario, when the proxy receives the SSL `ClientHello` message at the beginning of the SSL handshake, the proxy forwards this `ClientHello` message to the server unmodified. When the corresponding `ServerHello` message reaches the proxy, the proxy makes the interception decision by applying the configured policies to the `ServerHello` message. At this point the proxy may modify the `ServerHello` message before transmitting it back to the client. The proxy does not modify the `ClientHello` message for a good reason. The final SSL message exchanged between the client and the server is the `Finished` message. The `Finished` message contains the MD5 digest of all of the handshake messages combined with the negotiated master secret. If the proxy decides not to intercept this connection but it has modified any of the handshake messages, such as the initial `ClientHello` message, then the MD5 digest will fail verification at both the client and the server ends, resulting in the client and the server failing to complete the handshake even after the proxy has decided not to intercept that transaction.

There are several challenges the proxy must consider during its interception of SSL traffic. The client may offer a cipher suite that is not supported by the proxy. In this case, the proxy must modify the `ClientHello` message to substitute a cipher suite that the proxy supports. Other negotiation parameters such as the version, whether it is TLS 1.1, TLS 1.2, or SSL 3.0, could cause similar incompatibility issues, and these fields may be modified by the proxy en route to the server. For example, the proxy may replace and substitute a cipher suite it supports in the `ClientHello` message. This case is illustrated in the second scenario in Figure 1-9. In this scenario the proxy first saves a copy of the original `ClientHello` message before making the necessary modifications to its content and then transmits the new `ClientHello` message to the server. Then the proxy decides not to intercept the traffic after processing the `ServerHello` message. Because the `ClientHello` message was modified, the proxy must close the server-side TCP connection. Next, the proxy reconnects to the server with a new TCP connection and then sends the saved original `ClientHello` message to the server as a new SSL handshake negotiation. The client is unaware of any of these server-side activities. When the proxy forwards the `ClientHello` message to the server unmodified, however, the `ServerHello` message indicates the server has chosen a set of parameters that are not supported by the proxy; in this case the proxy will handle the transaction in the exact same way as it did in the previously described processing scenario. In this second scenario, once the SSL handshake completes, the SSL proxy acts as a packet forwarding system that splices the two connections into a TCP tunnel. The packets that flow across this TCP tunnel are encrypted packets, and the proxy performs only the packet forwarding action.

Interception Strategies

The second SSL interception scenario alludes to an interesting question: can SSL interception be accomplished without the TCP termination? There are

two main SSL interception strategies: one leverages the full TCP connection termination, while the other relies only on SSL encryption and decryption. The two SSL interception scenarios presented in Figure 1-9 can be summarized as the SSL interception strategy illustrated in Figure 1-10. This SSL interception strategy offers the most flexible and intrusive policy-driven processing of the content: content insertion, deletion, and transformation are all possible with this approach.

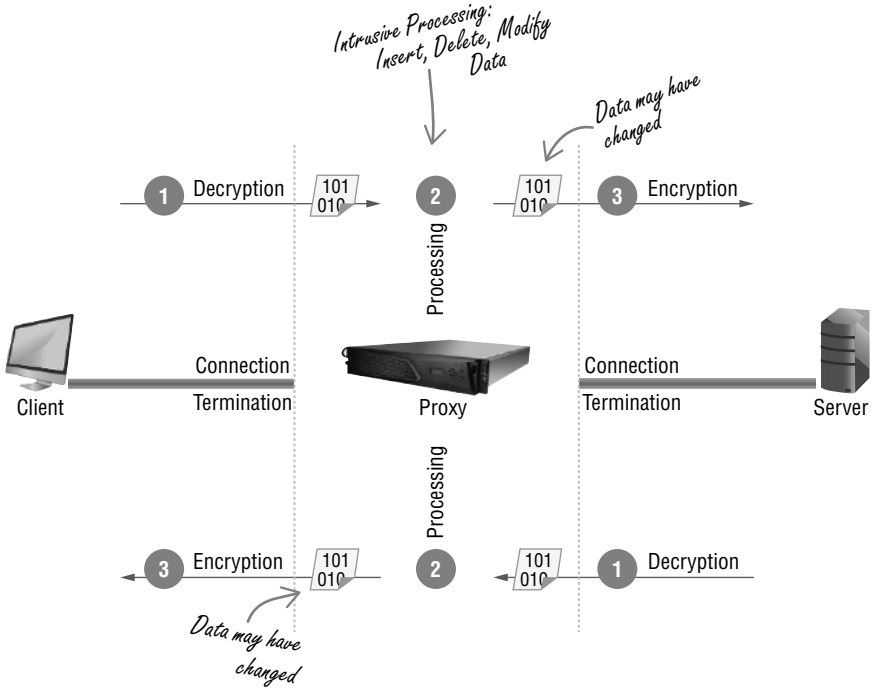

Figure 1-10: Type-I SSL Interception

Another SSL interception strategy is depicted in Figure 1-11. With this interception strategy, a single TCP connection is established between the client and server; in other words, the proxy does not terminate the TCP connection. The proxy has the ability to decrypt and encrypt the content within the SSL session, but the proxy cannot modify the content and must keep the content fully intact. Here is the reason why: SSL protects and transmits the data using a record protocol. The SSL record protocol is similar to the IP fragmentation and reassembly mechanism, where the data is divided into fragments and each fragment is independently encrypted and transmitted. On the receiving end, each encrypted payload is independently decrypted, verified, and reassembled back into the original data. Any modification applied to any of the SSL records would alter the original data and may render the data invalid.

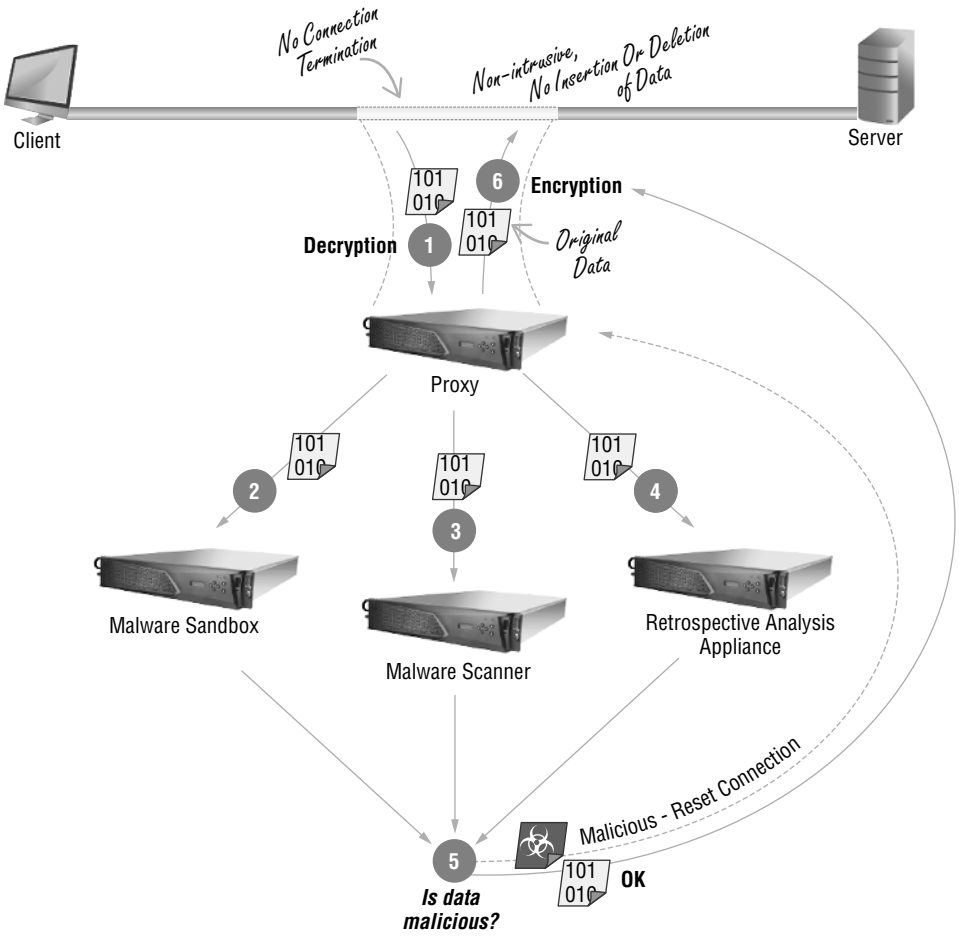

Figure 1-11: Type-II SSL Interception

Unlike the termination-based interception, the main course of action the proxy can enforce is to reset that single TCP connection, thus breaking the connectivity between the two end points. As depicted in Figure 1-11, the decision to break the TCP connection can come from a variety of sources. Once the encrypted payload has been transformed from cipher text into plain text, the SSL proxy can redirect the plain text to a diverse set of security devices to perform various in-depth content-centric analysis. For example, as shown in Figure 1-11, the content can be sent to a malware scanner first, and if something suspicious is discovered, the malware scanner returns an indication to the proxy. At that point, the proxy places the suspicious content into a malware sandbox to detonate the potential malware and investigate the outcome of the controlled execution. As soon as the malicious nature of the content is confirmed and the malware has been identified, the proxy begins retrospective analysis of the historical data to

discover the earliest exposure to that vulnerability and begins the construction of countermeasures. In parallel, the proxy shuts down the TCP connection to prevent further damage.

The proxy communicates with the other security devices by utilizing regular TCP or UDP connections and transmitting the plain text over these standard communication channels to maximize interoperability, thus eliminating the need for these devices to make any software modifications.

Referring back to the proxy architecture illustrated in Figure 1-8, the concept of transaction handoff was discussed in the context of application recognition: the SSL proxy transfers the transaction to another proxy through the PACE when the transaction operates over a protocol on top of the SSL. Another purpose for the transaction handoff is when one or more proxies must work collaboratively to manage and manipulate a transaction. The transaction handoff concept is detailed in Figure 1-12.

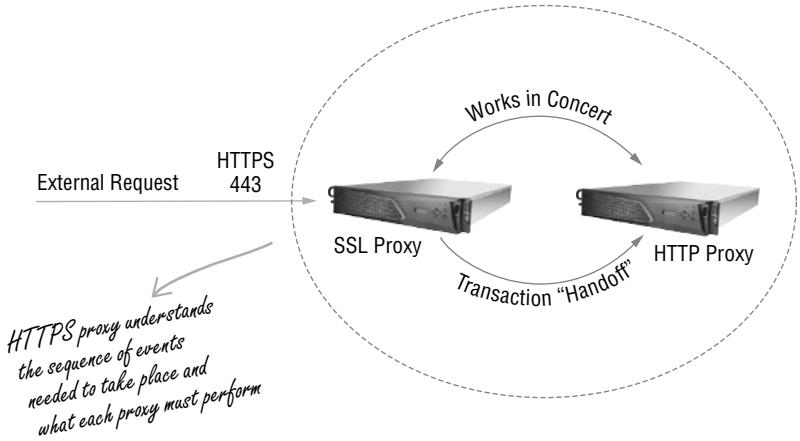

Figure 1-12: Transaction Handoff

As shown in Figure 1-12, the HTTPS proxy is conceptually comprised of two proxies: the SSL proxy and the HTTP proxy. In practice, the HTTP proxy is fully aware of the SSL processing detail when it handles the HTTPS request. Once the SSL handshake is complete, the SSL proxy must transfer this transaction to the HTTP proxy for HTTP-based proxy operation. Another example is the *stunnel proxy*, which is comprised of the SSL proxy and the TCP tunnel proxy. Therefore, the SSL proxy is typically implemented as a common proxy that provides services at the SSL layer to other proxies. Based on HTTP-specific policies, the HTTP proxy may instruct the SSL proxy to initiate certain operations, for example, performing a *rehandshake*, disallowing session resumption, or perhaps requiring client certificate authentication in future transactions with a specific client.

Certificates and Keys

One of the SSL design goals is to facilitate server authentication. For example, if you are making an online purchase at Amazon, your web browser must ascertain whether it is indeed communicating with an Amazon server before transmitting your credit card information to that server. An X.509 certificate binds a public key to a specific entity. A trusted third party, the *certificate authority* (CA), verifies the identity of the entity that owns the certificate and ensures the entity possesses the corresponding private key. At the completion of successful validation, the CA signs the certificate with its digital signature as proof of the certificate's authenticity. A CA-signed certificate guarantees the public key contained in the certificate belongs to the entity as claimed in that certificate. The CA's digital signature can be verified using the CA's public key.

The fact that an SSL proxy can perform traffic decryption and re-encryption after transaction interception implies that the SSL proxy possesses the server's private key if server authentication is mandatory. In practice and in the majority of cases, the SSL proxy will not have the server's private key. Can you imagine the SSL proxy having private keys from Google, Amazon, Facebook, Netflix, or any other commercial websites? Figure 1-13 illustrates how the SSL proxy achieves the keying mechanism necessary to perform decryption on intercepted traffic.

As shown in Figure 1-13, when the proxy receives the server certificate in Step 4, it modifies the certificate before sending it to the client. The proxy changes the certificate *issuer* to be the proxy itself. Because each certificate has a pair of keys—one public and one private—associated with it, the proxy replaces the original server's public key with its own public key. After making all of the necessary changes to the server certificate, the proxy signs the modified certificate using the private key of a preinstalled CA certificate and then replaces the *signature* field with the new signature value. The proxy transmits this newly transformed certificate to the client. How does the client respond when it receives this certificate from the proxy and begins server authentication?

The server certificate verification will complete successfully and uneventfully if the proxy is a legitimate intermediate CA holding certificate signing authority, and it is a part of a certificate chain that terminates at a client-trusted root CA. However, in most deployment situations the proxy will not have certificate signing authority. In this case, the client will neither trust nor accept the certificate fabricated by the proxy without user intervention. The common visual indication of a problematic certificate is a web browser pop-up window, similar to the one shown in Figure 1-14. In this example a proxy is deployed between the client and the Internet. When the user tries to access the Google website, the proxy modifies the Google certificate, subsequently triggering

the browser pop-up window at the client end. As shown in Figure 1-14, the browser pop-up window states the server certificate cannot be verified as the reason for user notification.

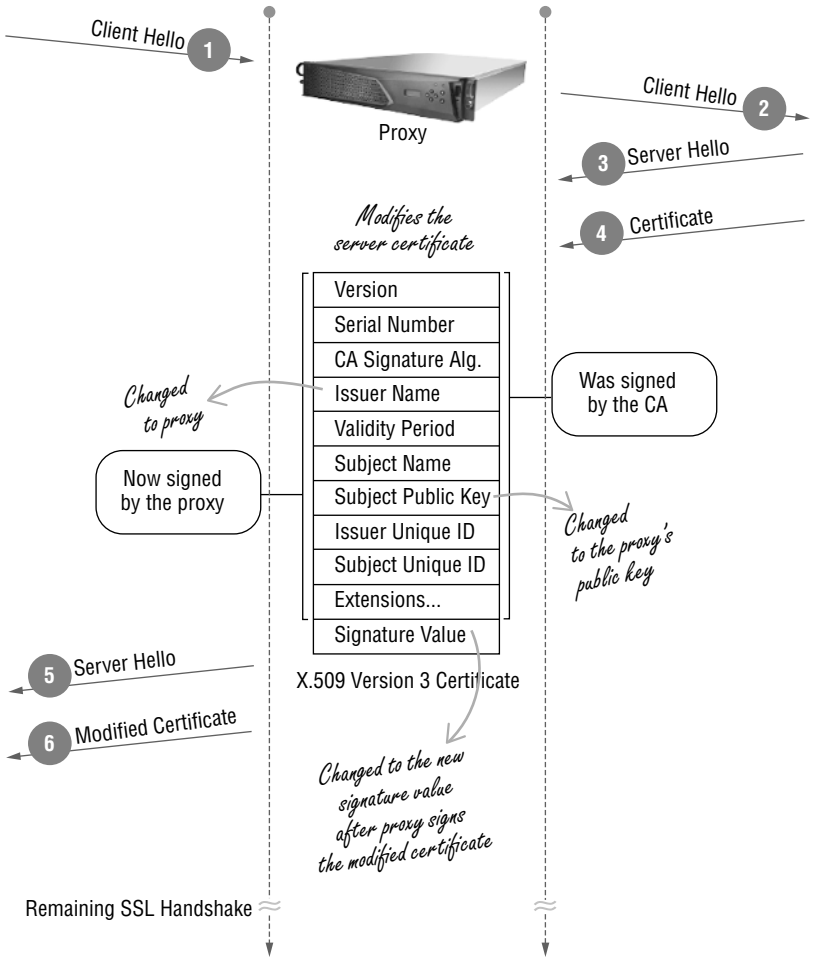

Figure 1-13: Server Certificate Modification

In the enterprise environment, such a browser pop-up window is a good indication that a corporate proxy is present in the network, which enforces the corporate use policy. Outside the corporate network, such an alert triggered by a non-verifiable certificate is strong evidence that a MITM attack may be taking place somewhere in the infrastructure.

SSL interception is confronted with another challenge when secure servers require client authentication. Client authentication may be necessary in situations where the client is granted access to restricted or highly confidential resources and services, such as military systems, only after the client authenticates

successfully. The client certificate must be issued by an externally known and trusted CA. Client authentication is controlled by the server. The server that demands client authentication sends a certificate request to the client during the SSL handshake process.

Figure 1-14: Browser Issued Warning about Proxy's Certificate

Installing an individual client certificate in the proxy so that the proxy can offer the right client certificate upon request by the server is a possible solution in enterprise networks. However, such a solution is neither scalable nor practical in any organization with a large number of users. One approach to solving the scalability problem is using a technique called *client certificate emulation*. Figure 1-15 illustrates an example of such a practice.

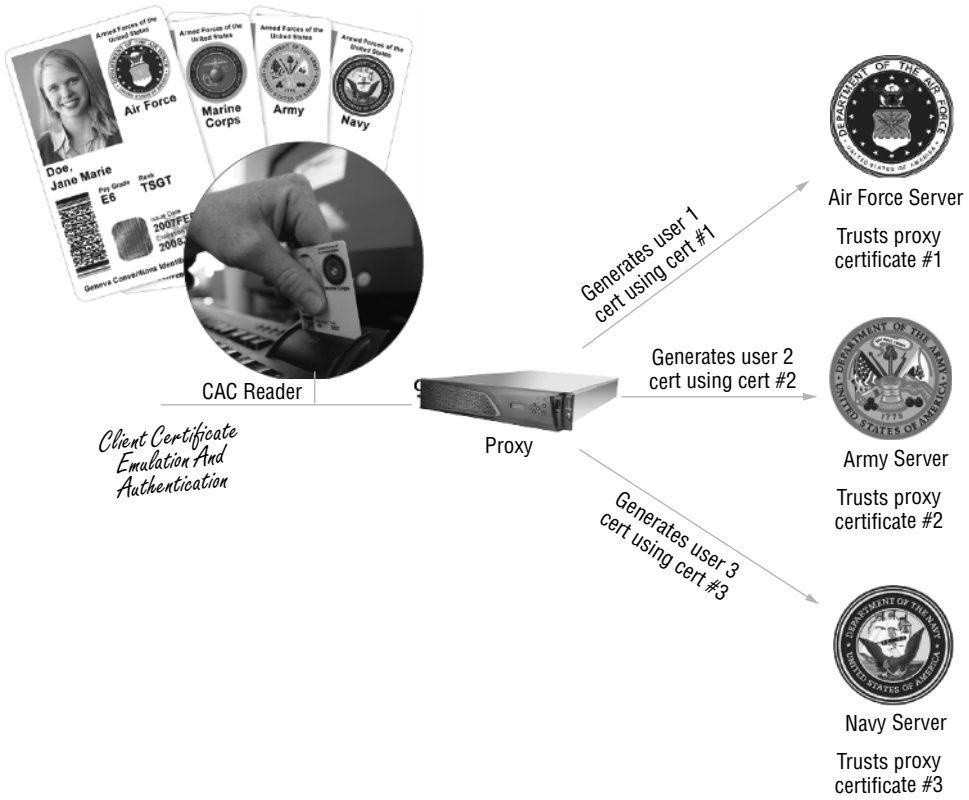

Figure 1-15: Client Certificate Emulation

The Common Access Card (CAC) serves as standard identification for military and DoD personnel. The CAC is a smart card that uniquely identifies an individual with that individual's private key embedded in it. In the fictitious scenario depicted in Figure 1-15, the same trusted CA is installed in the proxy and in the servers, and the proxy is given intermediate signing authority by this trusted CA. Once the proxy authenticates an individual, the proxy generates a certificate (possibly with a limited lifetime) that identifies this specific individual and the associated key pair and signs this certificate. When the server demands client authentication for that individual, the proxy offers the generated certificate to the server. The server can verify this certificate because the proxy has the signing authority issued by a CA that is also trusted by the server. As shown in Figure 1-15, instead of a single trusted CA, the proxy could install three different trusted CAs, one for the Air Force server, one for the Army server, and one for the Navy server. These CAs are trusted by each server, respectively.

In the non-termination–based SSL interception strategy, as depicted in Figure 1-11, the proxy must examine the SSL handshake exchange and modify the server certificate similar to the termination-based interception. The proxy transmits the modified server certificate to the client and uses that modified certificate along with its own key pairs to negotiate a master secret with the client. This master secret is used for traffic encryption and decryption between the client and the proxy. The proxy exchanges the master secret with the server using the server's original certificate. This master secret is used by the proxy to re-encrypt the decrypted client content and then transmit that newly encrypted traffic to the server.

With both interception strategies, if the server's private keys are installed in the proxy, then the proxy can avoid modifying the server certificate completely. In that case, the proxy has full capability to decrypt any content transmitted to the server using the server's private key.

Certificate Pinning and OCSP Stapling

Certificate pinning is a solution that attempts to solve a MITM attack when an entity tries to communicate with a peer securely using SSL, but the attacker assumes the identity of the peer by intercepting the certificate validation process using a rogue but valid certificate that masquerades as the peer.

How can a rogue but valid certificate be created in the first place? Such an attack was first made possible due to the fact there were still CAs that used the MD5 cryptographic hash function to generate certificate signatures. The MD5 hash function has known collision vulnerabilities that were discovered back in 1993; in a nutshell, it means that two different inputs to the same MD5 hash function can produce the same exact hash output. In 2005 researchers demonstrated a practical method to craft a pair of X.509 certificates, each having a different public key, to result in the same computed MD5 digest. The collision vulnerability attack against MD5 demonstrates that similar attacks could be made against other cryptographic hash functions, such as SHA-1, that are in use by CAs. Once an intermediate rogue CA certificate with certificate signing authority can be constructed, such a rogue CA certificate could be used to sign any fabricated certificate bearing the identity of any entity.

In recent years, there have been known incidents where CAs have issued questionable intermediate or subordinate root certificates. For example, in early 2012 Trustwave revoked a subordinate root certificate it issued to an unnamed company, which enabled that company to forge and issue unlimited certificates claiming the identities of any server or organization. The subordinate root certificate and the forged certificates it generated were all stored inside a hardware security module (HSM), and that specific certificate was issued for that company's internal use; however, such a certificate could have been misused, which warranted the revocation.

Another venue for attackers to gain access to rogue certificates is by breaching a CA and then obtaining rogue certificates through that CA. In late 2011 DigiNotar, a Dutch CA, was hacked and its certificate issuing servers were compromised by the hackers. Through DigiNotar, the hackers issued rogue certificates as well as signing rogue certificates. The breach came to light only after a third party made a public disclosure, and rogue certificates continued in circulation after the discovery and after DigiNotar claimed to have revoked all such rogue certificates.

With certificate pinning, the peer's certificate is included in the application when the application is built. For example, Google pins its certificates in its Chrome web browser, and when users download the Chrome browser, the Google certificates are already embedded inside the executable file. A peer's certificate can be manually inserted into a trusted certificate list after that certificate has been obtained through a secure and trusted channel. Certificate pinning eliminates the need to validate a certificate at runtime, during the secure connection establishment phase. Because the public key is the most important element of the certificate, pinning the public key instead of the certificate is another viable solution. There are no known workable solutions for an intercepting proxy to circumvent the certificate pinning mechanism other than holding the actual pinned certificate at the proxy.

A related identity validation concept is Online Certificate Status Protocol (OCSP) stapling, formally known as the Certificate Status Request TLS feature extension. OCSP stapling places the burden of identity verification on the peer, who must include an OCSP-signed and time-stamped response proving its certificate is valid during the TLS or SSL handshake. OCSP stapling also challenges the proxy's ability to perform transparent interception.

SSL Interception and Privacy

Privacy laws differ from country to country and region to region. Therefore, a proxy must sometimes obtain explicit consent from a user before intercepting any user traffic. When the proxy has intermediate certificate signing authority issued by a trusted root CA, any modified server certificate will not trigger a browser pop-up warning message because this modified certificate can be verified. In this case, how could the end user prevent proxy interception if the user has the right to choose the action?

Due to privacy concerns, Blue Coat Systems took the approach of utilizing the client authentication mechanism to determine if the client explicitly grants permission to allow for proxy interception. The proxy is programmed to parse two preformatted certificates, one that has the certificate common name "Yes Sir", and the other that has the common name "No Sir". The client installs two such certificates in its key ring, as shown in Figure 1-16.

Figure 1-16: Client Consent Certificate

Policies configured in the proxy would instruct the proxy to request client certificates for a pre-determined set of destinations that show up in client requests. Because the client has multiple certificates installed in its key ring, the browser prompts the user to select the certificate to return to the server. This process is shown in Figure 1-17.

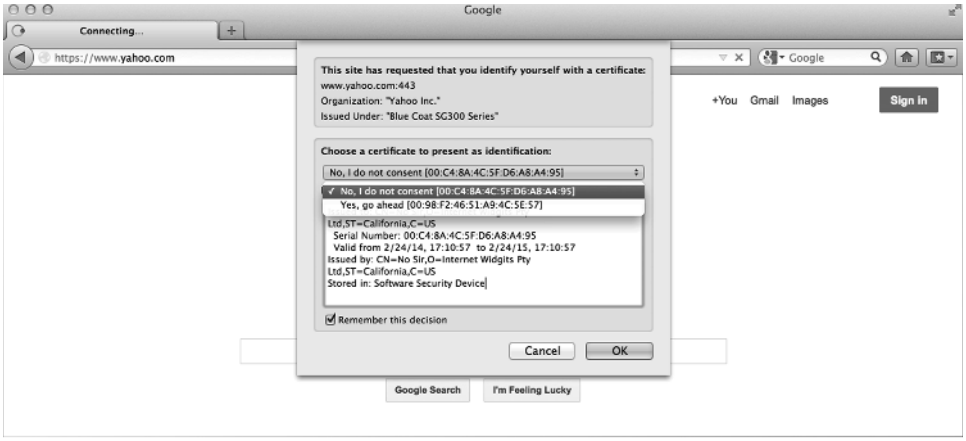

Figure 1-17: Client Consent Pop-up

Now the user has direct control over whether the proxy should intercept that particular session by selecting the right certificate. In other words, by choosing the "No Sir" certificate to transmit to the proxy, when the proxy parses this client certificate and sees the common name "No Sir", the proxy takes that as the cue and bypasses the session without SSL interception.

Summary

Firewalls are security devices that analyze traffic according to syntactical rules. The security policies enforced by a firewall are based on limited actions such as Allow or Deny. Firewalls provide the first level of traffic filtering. Intrusion detection systems perform traffic analysis based on historical data and heuristics and can scan for known threats. Both firewalls and intrusion detection systems become inoperable with encrypted traffic. This fundamental challenge is solved by proxies. Proxies are complex to design and implement. A proxy operates with deep knowledge of the semantics of an application or a protocol. The SSL proxy is a testament to just how sophisticated a proxy must be in order to successfully intercept and process various transactions. A significant part of this chapter was devoted to explaining the inner workings of the SSL proxy. The next chapter focuses on the various types of proxy deployments and discusses associated deployment challenges and solutions.