1 Cyber Security Fundamentals

1.1 Network and Security Concepts

1.1.1 Information Assurance Fundamentals

Authentication, authorization, and nonrepudiation are tools that system designers can use to maintain system security with respect to confidentiality, integrity, and availability. Understanding each of these six concepts and how they relate to one another helps security professionals design and implement secure systems. Each component is critical to overall security, with the failure of any one component resulting in potential system compromise.

There are three key concepts, known as the CIA triad, which anyone who protects an information system must understand: confidentiality, integrity, and availability. Information security professionals are dedicated to ensuring the protection of these principals for each system they protect. Additionally, there are three key concepts that security professionals must understand to enforce the CIA principles properly: authentication, authorization, and nonrepudiation. In this section, we explain each of these concepts and how they relate to each other in the digital security realm. All definitions used in this section originate from the National Information Assurance Glossary (NIAG) published by the U.S. Committee on National Security Systems.[1]

1.1.1.1 Authentication Authentication is important to any secure system, as it is the key to verifying the source of a message or that an individual is whom he or she claims. The NIAG defines *authentication* as a "security measure designed to establish the validity of a transmission, message, or originator, or a means of verifying an individual's authorization to receive specific categories of information."

FACTOR	EXAMPLES
Something You Know	Information the system assumes others do not know; this information may be secret, like a password or PIN code, or simply a piece of information that most people do not know, such as a user's mother's maiden name.
Something You Have	Something the user possesses that only he or she holds; a Radio Frequency ID (RFID) badge, One-Time-Password (OTP) generating Token, or a physical key
Something You Are	A person's fingerprint, voice print, or retinal scan—factors known as biometrics

Exhibit 1-1 Factors of authentication.

There are many methods available to authenticate a person. In each method, the authenticator issues a challenge that a person must answer. This challenge normally comprises requesting a piece of information that only authentic users can supply. These pieces of information normally fall into the three classifications known as *factors of authentication* (see Exhibit 1-1).

When an authentication system requires more than one of these factors, the security community classifies it as a system requiring *multifactor authentication*. Two instances of the same factor, such as a password combined with a user's mother's maiden name, are not multifactor authentication, but combining a fingerprint scan and a personal identification number (PIN) is, as it validates something the user is (the owner of that fingerprint) and something the user knows (a PIN).

Authentication also applies to validating the source of a message, such as a network packet or e-mail. At a low level, message authentication systems cannot rely on the same factors that apply to human authentication. Message authentication systems often rely on cryptographic signatures, which consist of a digest or hash of the message generated with a secret key. Since only one person has access to the key that generates the signature, the recipient is able to validate the sender of a message.

Without a sound authentication system, it is impossible to trust that a user is who he or she says that he or she is, or that a message is from who it claims to be.

1.1.1.2 Authorization While authentication relates to verifying identities, authorization focuses on determining what a user has permission to do. The NIAG defines *authorization* as "access privileges granted to a user, program, or process."

After a secure system authenticates users, it must also decide what privileges they have. For instance, an online banking application will authenticate a user based on his or her credentials, but it must then determine the accounts to which that user has access. Additionally, the system determines what actions the user can take regarding those accounts, such as viewing balances and making transfers.

1.1.1.3 Nonrepudiation Imagine a scenario wherein Alice is purchasing a car from Bob and signs a contract stating that she will pay $20,000 for the car and will take ownership of it on Thursday. If Alice later decides not to buy the car, she might claim that someone forged her signature and that she is not responsible for the contract. To refute her claim, Bob could show that a notary public verified Alice's identity and stamped the document to indicate this verification. In this case, the notary's stamp has given the contract the property of *nonrepudiation*, which the NIAG defines as "assurance the sender of data is provided with proof of delivery and the recipient is provided with proof of the sender's identity, so neither can later deny having processed the data."

In the world of digital communications, no notary can stamp each transmitted message, but nonrepudiation is still necessary. To meet this requirement, secure systems normally rely on asymmetric (or public key) cryptography. While symmetric key systems use a single key to encrypt and decrypt data, asymmetric systems use a key pair. These systems use one key (private) for signing data and use the other key (public) for verifying data. If the same key can both sign and verify the content of a message, the sender can claim that anyone who has access to the key could easily have forged it. Asymmetric key systems have the nonrepudiation property because the signer of a message can keep his or her private key secret. For more information on asymmetric cryptography, see the "State of the Hack" article on the subject published in the July 6, 2009, edition of the *Weekly Threat Report*.[2]

1.1.1.4 Confidentiality The term *confidentiality* is familiar to most people, even those not in the security industry. The NIAG defines *confidentiality* as "assurance that information is not disclosed to unauthorized individuals, processes, or devices."

Assuring that unauthorized parties do not have access to a piece of information is a complex task. It is easiest to understand when broken down into three major steps. First, the information must have protections capable of preventing some users from accessing it. Second, limitations must be in place to restrict access to the information to only those who have the authorization to view it. Third, an authentication system must be in place to verify the identity of those with access to the data. Authentication and authorization, described earlier in this section, are vital to maintaining confidentiality, but the concept of confidentiality primarily focuses on concealing or protecting the information.

One way to protect information is by storing it in a private location or on a private network that is limited to those who have legitimate access to the information. If a system must transmit the data over a public network, organizations should use a key that only authorized parties know to encrypt the data. For information traveling over the Internet, this protection could mean using a virtual private network (VPN), which encrypts all traffic between endpoints, or using encrypted e-mail systems, which restrict viewing of a message to the intended recipient. If confidential information is physically leaving its protected location (as when employees transport backup tapes between facilities), organizations should encrypt the data in case it falls into the hands of unauthorized users.

Confidentiality of digital information also requires controls in the real world. Shoulder surfing, the practice of looking over a person's shoulder while at his or her computer screen, is a nontechnical way for an attacker to gather confidential information. Physical threats, such as simple theft, also threaten confidentiality. The consequences of a breach of confidentiality vary depending on the sensitivity of the protected data. A breach in credit card numbers, as in the case of the Heartland Payment Systems processing system in 2008, could result in lawsuits with payouts well into the millions of dollars.

1.1.1.5 Integrity In the information security realm, *integrity* normally refers to data integrity, or ensuring that stored data are accurate and contain no unauthorized modifications. The National Information Assurance Glossary (NIAG) defines integrity as follows:

> Quality of an IS (Information System) reflecting the logical correctness and reliability of the operating system; the logical completeness of the hardware and software implementing the protection mechanisms; and the consistency of the data structures and occurrence of the stored data. Note that, in a formal security mode, integrity is interpreted more narrowly to mean protection against unauthorized modification or destruction of information.[3]

This principal, which relies on authentication, authorization, and nonrepudiation as the keys to maintaining integrity, is preventing those without authorization from modifying data. By bypassing an authentication system or escalating privileges beyond those normally granted to them, an attacker can threaten the integrity of data.

Software flaws and vulnerabilities can lead to accidental losses in data integrity and can open a system to unauthorized modification. Programs typically tightly control when a user has read-to-write access to particular data, but a software vulnerability might make it possible to circumvent that control. For example, an attacker can exploit a Structured Query Language (SQL) injection vulnerability to extract, alter, or add information to a database.

Disrupting the integrity of data at rest or in a message in transit can have serious consequences. If it were possible to modify a funds transfer message passing between a user and his or her online banking website, an attacker could use that privilege to his or her advantage. The attacker could hijack the transfer and steal the transferred funds by altering the account number of the recipient of the funds listed in the message to the attacker's own bank account number. Ensuring the integrity of this type of message is vital to any secure system.

1.1.1.6 Availability Information systems must be accessible to users for these systems to provide any value. If a system is down or responding too slowly, it cannot provide the service it should. The NIAG defines *availability* as "timely, reliable access to data and information services for authorized users."

Attacks on availability are somewhat different from those on integrity and confidentiality. The best-known attack on availability is a denial of service (DoS) attack. A DoS can come in many forms, but each form disrupts a system in a way that prevents legitimate users from accessing it. One form of DoS is resource exhaustion, whereby an attacker overloads a system to the point that it no longer responds to legitimate requests. The resources in question may be memory, central processing unit (CPU) time, network bandwidth, and/or any other component that an attacker can influence. One example of a DoS attack is network flooding, during which the attacker sends so much network traffic to the targeted system that the traffic saturates the network and no legitimate request can get through.

Understanding the components of the CIA triad and the concepts behind how to protect these principals is important for every security professional. Each component acts like a pillar that holds up the security of a system. If an attacker breaches any of the pillars, the security of the system will fall. Authentication, authorization, and nonrepudiation are tools that system designers can use to maintain these pillars. Understanding how all of these concepts interact with each other is necessary to use them effectively.

1.1.2 Basic Cryptography

This section provides information on basic cryptography to explain the history and basics of ciphers and cryptanalysis. Later sections will explain modern cryptography applied to digital systems.

The English word *cryptography* derives from Greek and translates roughly to "hidden writing." For thousands of years, groups who wanted to communicate in secret developed methods to write their messages in a way that only the intended recipient could read. In the information age, almost all communication is subject to some sort of eavesdropping, and as a result cryptography has advanced rapidly. Understanding how cryptography works is important for anyone who wants to be sure that their data and communications are safe from intruders. This section discusses cryptography, starting with basic ciphers and cryptanalysis.

The ancient Egyptians began the first known practice of writing secret messages, using nonstandard hieroglyphs to convey secret messages as early as 1900 bc. Since that time, people have developed many methods of hiding the content of a message. These methods are known as *ciphers*.

The most famous classical cipher is the substitution cipher. Substitution ciphers work by substituting each letter in the alphabet with another one when writing a message. For instance, one could shift the letters of the English alphabet as shown:

abcdefghijklmnopqrstuvwxyz
nopqrstuvwxyzabcdefghijklm

Using this cipher, the message "the act starts at midnight" would be written as "gur npg fgnegf ng zvqavtug." The text above, showing how to decode the message, is known as the *key*. This is a very simple substitution cipher known as the *Caesar cipher* (after Julius Caesar, who used it for military communications) or *ROT13* because the characters in the key are rotated thirteen spaces to the left.

Cryptography is driven by the constant struggle between people who want to keep messages secret and those who work to uncover their meanings. Substitution ciphers are very vulnerable to cryptanalysis, the practice of breaking codes. With enough text, it would be simple to begin replacing characters in the ciphertext with their possible cleartext counterparts. Even without knowing about the Caesar cipher, it is easy to guess that a three-letter word at the beginning of a sentence is likely to be *the*. By replacing all instances of the letters *g*, *u*, and *r* with *t*, *h*, and *e*, the ciphertext changes to

the npt ftnetf nt zvqavtht

Next, the analyst might notice that the fourth word is only two letters long and ends with *t*. There are two likely possibilities for this word: *at* and *it*. He chooses *at* and replaces all occurrences of *n* in the sentence with an *a*.

the apt ftaetf at zvqavtht

With *at* in place, the pattern is clearer, and the analyst guesses that if the letter *g* translates to *t*, the adjacent letter *f* may translate to *s*.

the apt staets at zvqavtht

The word *sta_ts* now looks very close to *starts*, and the analyst makes another substitution, indicating that *rst* is equivalent to *efg*, which reveals the full pattern of the cipher and the message. While the message is now clear, the meaning of "the act starts at midnight" is not. Code words are an excellent way of hiding a message but, unlike

LETTER	FREQUENCY	LETTER	FREQUENCY
e	12.70%	m	2.41%
t	9.06%	w	2.36%
a	8.17%	f	2.23%
o	7.51%	g	2.02%
i	6.97%	y	1.97%
n	6.75%	p	1.93%
s	6.33%	b	1.49%
h	6.09%	v	0.98%
r	5.99%	k	0.77%
d	4.25%	j	0.15%
l	4.03%	x	0.15%
c	2.78%	q	0.10%
u	2.76%	z	0.07%

Exhibit 1-2 Frequency of letters in the English language.

cryptography, cannot hide the meaning of arbitrary information without agreement on the meaning of the code words in advance.

Short messages can be difficult to decrypt because there is little for the analyst to study, but long messages encrypted with substitution ciphers are vulnerable to frequency analysis. For instance, in the English language, some letters appear in more words than others do. Exhibit 1-2 shows the frequency of each letter in the English language.

E is by far the most common letter in the English language and, as such, is also the most likely character in an article written in English. Using the table above, an analyst could determine the most likely cleartext of any ciphertext encrypted with a substitution cipher. As shown in the example sentence above, while the ciphertext appears to be random, patterns remain that betray the original text.

The ultimate goal of any cipher is to produce ciphertext that is indistinguishable from random data. Removing the patterns inherent in the original text is crucial to producing ciphertext that is impossible to decode without the original key. In 1917, Gilbert Vernam

Exhibit 1-3 A lottery cage randomizes the number selection.

developed the *one-time pad*, a cryptographic cipher that, with a properly randomized key, produces unbreakable ciphertext. A one-time pad is similar to a substitution cipher, for which another letter based on a key replaces a letter, but rather than using the same key for the entire message, a new key is used for each letter. This key must be at least as long as the message and not contain any patterns a cryptanalyst could use to break the code.

Imagine a room filled with lottery cages such as the one shown in Exhibit 1-3. Each cage contains twenty-six balls numbered 1–26. A person stands next to each cage, turning the crank until a single ball rolls out; that person records the number on a pad of paper, and puts the ball back into the cage. Doing this repeatedly would eventually generate a very long string of random numbers. We can use these numbers to encrypt our message with a one-time pad. In the first row in the key shown below, we have our original cleartext ("Clear") and, below that, the numbers generated by our lottery cage ("Cage"). To apply the one-time pad, we perform the same rotation of the alphabet as in the substitution cipher above, but we rotate the alphabet by the random number, resulting in the ciphertext ("Cipher") in the third row.

Clear	T	h	e		a	c	T		s	t	a	r	t	s		a	t		m	i	d	n	i	g	h	t
Cage	22	19	2	11	5	12	19	5	16	12	6	11	5	2	19	15	24	20	18	2	21	6	5	19	17	21
Cipher	O		g	k	f	o	L	e	h	e	g	b	y	u	s	p	q	t	d	k	y	t	n	z	y	n

The letter *a* at the beginning of *act* is rotated five spaces to the right, resulting in the letter *f*; however, the letter *a* at the beginning

Exhibit 1-4 The German Enigma coding machine.

of *at* is rotated fifteen spaces, resulting in the letter *p*. The recipient can decrypt the text by reversing the function, rotating the alphabet left by the number specified in the key rather than right. A frequency analysis will fail against this cipher because the same character in the ciphertext can be the result of different inputs from the cleartext. The key to the one-time pad is only using it one time. If the cryptographer uses the numbers in a repeating pattern or uses the same numbers to encode a second message, a pattern may appear in the ciphertext that would help cryptanalysts break the code. The study of cryptography advanced greatly during World War II due to the invention of radio communication. Anyone within range of a radio signal could listen to the transmission, leading both sides to spend countless hours studying the art of code making and code breaking.

The problem with one-time pads is that they are cumbersome to generate and have a limited length. If a submarine captain goes to sea for six months, he must have enough one-time pads with him to encode every message he intends to send to central command. This dilemma led to the development of machines that could mimic the properties of a one-time pad but without the need to generate long keys and carry books of random numbers. The most famous machine

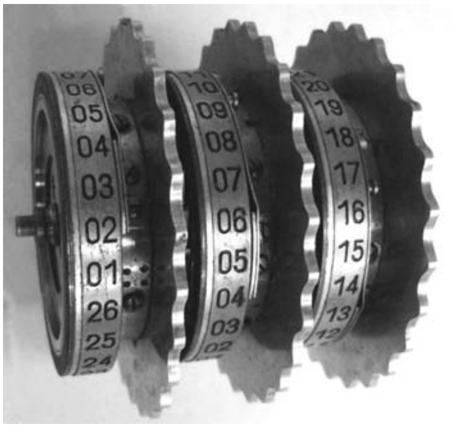


Exhibit 1-5 Enigma rotors.

of this type is the Enigma, invented by the German engineer Arthur Scherbius at the end of World War I.[4] The Enigma (see Exhibit 1-4)[5] used a series of rotors (see Exhibit 1-5)[6] to encrypt each letter typed into it with a different key. Another user with an enigma machine could decode the message because their system had the same combination of encoded rotors.

The Enigma could not perfectly replicate a one-time pad because any system that does not begin with random input will eventually reveal a pattern. British mathematicians eventually discovered patterns in Enigma messages, giving them the capability to read many German military secrets during World War II. Since the invention of modern electronic computers, cryptography has changed significantly. We no longer write messages on paper pads or speak them character by character into a microphone but transmit them electronically as binary data. The increase in computing power also gives cryptanalysts powerful new tools for analyzing encrypted data for patterns. These developments have led to new algorithms and techniques for hiding data. The next section provides some detail about modern cryptography and how the principles of classical cryptography are applied to digital systems.

1.1.3 Symmetric Encryption

Although symmetric encryption requires a shared key and therefore depends upon the secrecy of that key, it is an effective and fast method

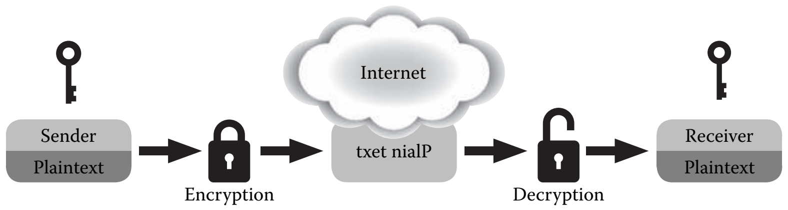


Exhibit 1-6 Symmetric encryption: the sender and receiver use the same key.

for protecting the confidentiality of the encrypted content. In this section we explain the basics of symmetric encryption and how it differs from asymmetric algorithms. Symmetric encryption is a class of reversible encryption algorithms that use the same key for both encrypting and decrypting messages.

Symmetric encryption, by definition, requires both communication endpoints to know the same key in order to send and receive encrypted messages (see Exhibit 1-6). Symmetric encryption depends upon the secrecy of a key. Key exchanges or pre-shared keys present a challenge to keeping the encrypted text's confidentiality and are usually performed out of band using different protocols.

Algorithms in this category are usually fast because their operations use cryptographic primitives. As previously discussed in Basic Cryptography we explained how the cryptographic primitive substitution works. Permutation, or altering the order, is another cryptographic primitive that many symmetric algorithms also use in practice.[7]

1.1.3.1 Example of Simple Symmetric Encryption with Exclusive OR (XOR) At its most basic level, symmetric encryption is similar to an exclusive OR (XOR) operation, which has the following truth table for input variables *p* and *q*:

P	Q	= P XOR Q
True	True	False
True	False	True
False	True	True
False	False	False

The XOR operation is nearly the same as one would expect for OR, except when both *p* and *q* are true. The properties of XOR make it ideal for use in symmetric cryptography because one of the inputs (*p*) can act as the message and the other input (*q*) can act as the key. The recipient of an encrypted message (*p XOR q*) decrypts that message by performing the same XOR operation that the sender used to encrypt the original message (*p*).

P XOR Q	Q	= (P XOR Q) XOR Q
False	True	True
True	False	True
True	True	False
False	False	False

The operation above shows how to decrypt the encrypted message (*p XOR q*) to obtain the original message (*p*). Applying this technique to larger values by using their individual bits and agreeing on a common key (*q*) represents the most basic symmetric encryption algorithm.

Encryption using XOR is surprisingly common in unsophisticated malicious code, including shellcode, even as a means to hide logging or configuration information. Due to its simplicity, many unsophisticated attackers use either one-byte XOR keys or multibyte XOR keys to hide data. The Python script below demonstrates how to brute force single-byte XOR keys when they contain one of the expected strings: *.com*, *http*, or *pass*.

```
count = len(data)
for key in range(1,255):
 out = ‘’
 for x in range(0,count):
 out += chr(ord(data[x]) ^ int(key))
 results = out.count(‘.com’) + out.count(‘http’) +
out.count(‘pass’)
 if results:
print “Encryption key: \t%d matched: %d” % (key,results)
print out
```

While this script is effective when the original message contains a URL or password string, analysts could use other techniques to identify expected results such as character distribution or words in the English language.

The reason it is possible to brute force an XOR key that uses just one byte is that the length of the key is so small. One byte (8 bits) allows for only 256 possible key combinations. A two-byte (16 bits) key creates 65,536 possible keys, but this number is still quite easy to brute force with modern computing power. Modern cryptographic ciphers typically use 128-bit keys, which are still infeasible to brute force with today's computing power.

The XOR operation is an example of a stream cipher, which means that the key operates on every bit or byte to encrypt a message. Like traditional substitution ciphers, XOR leaves patterns in ciphertext that a cryptanalyst could use to discover the plaintext. Performing an XOR operation on the same data twice with the same key will always result in the same ciphertext. Modern stream ciphers like RC4, designed by Ron Rivest in 1987, avoid this problem by using a pseudo-random number generation (PRNG) algorithm. Instead of performing an XOR on each byte of data with a key, a PRNG receives a chosen key, used as a "seed." A PRNG generates numbers that are close to random but will always be the same given the same seed. RC4 uses the PRNG to create an infinitely long, one-time pad of single-byte XOR keys. This technique allows the sender to encrypt a message with a single (relatively short) key, but for each individual byte, the XOR key is different.

1.1.3.2 Improving upon Stream Ciphers with Block Ciphers Block ciphers are more common in symmetric encryption algorithms because they operate on a block of data rather than each character (bit or byte). PRNG algorithms used in stream ciphers are typically time intensive. Block ciphers are the best choice for bulk data encryption. Stream ciphers remove patterns from ciphertext using PRNGs, but block ciphers use a more efficient method called *cipher block chaining* (CBC).

When using a block cipher in CBC mode, both a key and a random initialization vector (IV) convert blocks of plaintext into ciphertext. The initialization vector and plaintext go through an XOR operation, and the result is an input to the block cipher with the chosen key (see Exhibit 1-7). This ensures that the resulting ciphertext is different, even if the same key was used to encrypt the same plaintext, as long as the IV is different and sufficiently random with each execution of the algorithm.

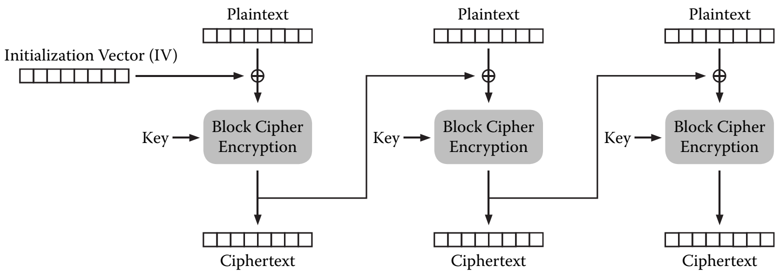


Exhibit 1-7 Cipher block chaining (CBC) mode encryption. *Source*: Cipher block chaining http://en.wikipedia.org/wiki/File:Cbc_encryption.png.

The next block will be encrypted with the same key, but instead of using the original IV, CBC mode uses the ciphertext generated by the last function as the new IV. In this way, each block of cipher text is chained to the last one. This mode has the drawback of data corruption at the beginning of the file, resulting in complete corruption of the entire file, but is effective against cryptanalysis.

All of the most popular symmetric algorithms use block ciphers with a combination of substitution and permutation. These include the following:

- 1977 DES
- 1991 IDEA
- 1993 Blowfish
- 1994 RC5
- 1998 Triple DES
- 1998 AES

iDefense analyzed several malicious code attacks that encrypt data using the popular algorithms shown in this list. Due to attackers including the decryption or encryption key on an infected system, analysts can attempt to decrypt messages of this type. Additionally, analysis of the system memory before encryption or after decryption may be effective at revealing the original message.

Programmers may wish to write custom encryption algorithms, in the hopes that their infrequent or unusual use will detract attackers; however, such algorithms are usually risky. As an example of this, consider how a programmer who applies the data encryption standard (DES) algorithm twice could affect the strength of the message. Using double DES does not dramatically increase the strength of a message over DES. The reason is that an attacker can compare the decryption of the ciphertext and the encryption of the plaintext. When both of these values match, the attacker has successfully identified both keys used for encrypting the message.

Symmetric encryption can be very fast and protect sensitive information provided the key remains secret. The grouping of larger blocks of data in the encryption algorithm makes it more difficult to decrypt without the key. Key exchange and protection are the most important aspects of symmetric cryptography because anyone who has the key can both encrypt and decrypt messages. Asymmetric algorithms are different because they use different keys for encryption and decryption, and in this way, public key encryption can solve other goals beyond symmetric algorithms that protect confidentiality.

1.1.4 Public Key Encryption

This section continues this series with a brief discussion of asymmetric encryption, more commonly referred to as public key encryption.

Public key encryption represents a branch of cryptography for which the distinguishing attribute of the system is the use of two linked keys for encryption and decryption, rather than a single key. While a variety of public key encryption solutions have been proposed, with some implemented and standardized, each system shares one common attribute: each public key system uses one key, known as the *public key*, to encrypt data, and a second key, known as the *private key*, to decrypt the encrypted data.

Public key encryption solves one of the major issues with symmetric key encryption, namely, the use of a shared key for both sides of the conversation. In public key systems, the intended recipient of a secure communication publishes his or her public key. Anyone wishing to send a secure datagram to the recipient uses the recipient's public key to encrypt the communication; however, those in possession of the public key cannot use the key to decrypt the communication. The use of a public key is a one-way cryptographic operation. This allows recipients to give out their public keys without the risk of someone using the same public keys to reveal the original content of the messages sent. This is the most obvious advantage over symmetric encryption. To decrypt the encrypted message, the recipient uses his or her

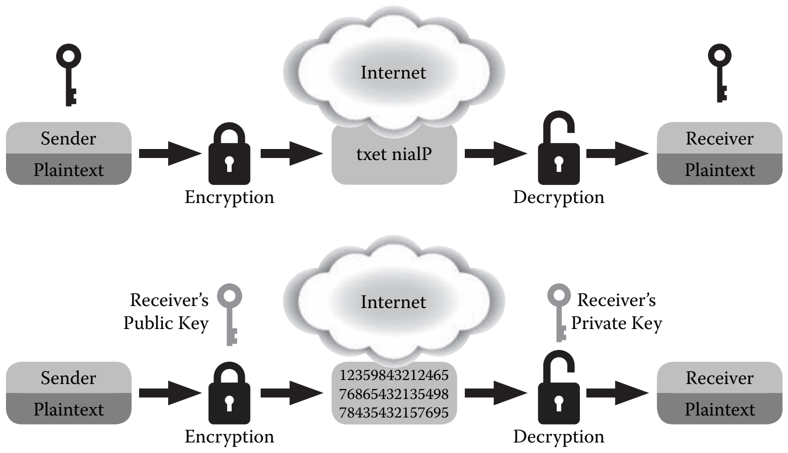


Exhibit 1-8 Symmetric encryption (top) versus public key encryption (bottom).

private key. The private key has a mathematical relationship to the public key, but this relationship does not provide an easy way for an attacker to derive the private key from the public key. Given the fact that the recipient uses the private key to decrypt messages encoded with the public key, it is paramount that the owner of the private key keeps it secure at all times.

Visually, the process of encrypting and decrypting a message using the public key method is similar to the process of using symmetric encryption with the notable exception that the keys used in the process are not the same. Exhibit 1-8 illustrates this disconnect.

One of the simplest analogies for public key encryption is the lock box analogy. In essence, if an individual (Blake, for example) wanted to send a message to another individual (Ryan, for example) without exchanging a shared cryptographic key, Blake could simply place his communication in a box and secure it with a lock that only Ryan could open. For Blake to possess such a lock, the box would need to be publicly available. In this case, that lock represents Ryan's public key. Blake could then send the locked box to Ryan. Upon receiving the box, Ryan would use his key to unlock the box to retrieve the message. In this situation, once Blake has locked (encrypted) his message to Ryan into the lock box with Ryan's lock (public key), Blake, or anyone else who may come in contact with the lock box, will be unable to access the contents. Only with Ryan's private key to the lock box will the message become retrievable.

Unlike symmetric encryption schemes that rely on a shared key and the use of substitutions and permutations of the data stream, public key encryption systems use mathematical functions. Researchers have developed a variety of public key–asymmetric encryption schemes, some more practical than others, but each of these schemes relies on the use of mathematical functions to encrypt and decrypt the data stream. A key attribute of the process is the fact that while both the public key and private key are mathematically related, it is practically impossible, given a finite time frame, to derive the private key from the public key. This fact allows the unbiased distribution of the recipient's public key without the fear that an attacker can develop the private key from the public key to decrypt the encoded message.

Whitfield Diffie and Martin Hellman developed one of the first asymmetric encryption schemes in 1976.[8] Their original work focused on the framework of establishing an encryption key for communication between two parties that must talk over an untrusted and insecure communication medium. Later, in 1979, researchers at MIT (Ron Rivest, Adi Shamir, and Leonard Adleman)[9] expanded on this research to develop one of the widest used public key encryption systems in use today. Known as the *RSA system*, a name derived from the original inventors' last names, the system uses large prime numbers to encrypt and decrypt communication. While the math involved is somewhat cumbersome for the confines of this text, in essence the RSA process works as such:

1. The recipient generates three numbers: one to be used as an exponential (e), one as a modulus (n), and one as the multiplicative inverse of the exponential with respect to the modulus (d). The modulus n should be the product of two very large prime numbers, p and q. Thusly, $n = pq$.
2. The recipient publishes his or her public key as (e, n).
3. The sender transforms the message (M) to be encrypted into an integer whose value is between 0 and (n–1). If the message cannot fit within the confines of this integer space, the message is broken into multiple blocks.
4. The sender generates the ciphertext (C) by applying the following mathematical function:

$$C = M^e \bmod n$$

5. The sender transmits the ciphertext to the recipient.
6. The recipient uses the pair (d, n) as the private key in order to decrypt the ciphertext. The decryption process uses the following mathematical transform to recover the original plaintext:

$$M = C^d \bmod n$$

The power of the RSA scheme lays in the use of the large prime numbers p and q. Factoring an extremely large prime number (on the order of 2^{1024} or 309 digits) is an exceedingly difficult task—a task for which there is no easy solution. To understand how the RSA scheme works in simpler terms, it is best to use a simpler, smaller example:[10]

1. The recipient chooses two prime numbers: for example, $p = 17$ and $q = 11$.
2. The recipient calculates n by multiplying the two prime numbers together: ($n = 187$).
3. The recipient chooses an exponent such that the exponent is less than $(p-1)(q-1)$, which is 160, and the exponent is relatively prime to this number. In this scenario, a recipient could choose the number 7, as it is less than 160 and relatively prime to 160.
4. The value of d is calculated by solving $de = 1 \pmod{160}$ with $d < 160$. The math behind this calculation is beyond the scope of this book; however, in this scenario, d has the value of 23.
5. At this point in the scenario, the recipient could have developed a private key of (23, 187) and a public key of (7, 187).

If the sender were to encrypt the message of 88 (which is between 0 and 186) using the RSA method, the sender would calculate 88^7 mod 187, which equals 11. Therefore, the sender would transmit the number 11 as the ciphertext to the recipient. To recover the original message, the recipient would then need to transform 11 into the original value by calculating 11^{23} mod 187, which equals 88. Exhibit 1-9 depicts this process.

As seen in the previous example, public key encryption is a computationally expensive process. As such, public key encryption is not

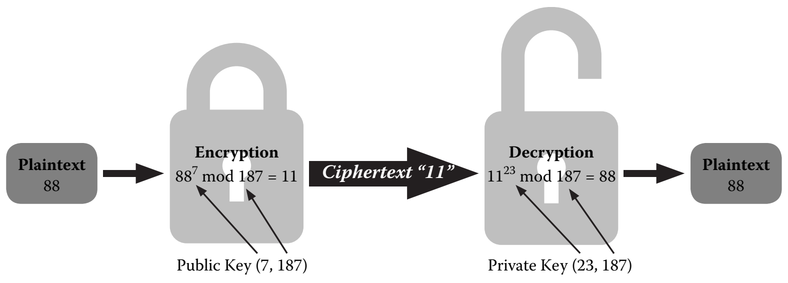


Exhibit 1-9 An RSA encryption–decryption example. *Note*: *RSA* stands for Ron Rivest, Adi Shamir, and Leonard Adleman, its inventors.

suited for bulk data encryption. The computational overhead resulting from public key encryption schemes is prohibitive for such an application. Smaller messages and symmetric encryption key exchanges are ideal applications for public key encryption. For example, secure socket layer (SSL) communication uses public key encryption to establish the session keys to use for the bulk of the SSL traffic. The use of public key encryption to communicate the key used in a symmetric encryption system allows two parties communicating over an untrusted medium to establish a secure session without undue processing requirements.

Compared to the old symmetric encryption, public key encryption is a new technology revolutionizing the field of cryptography. The encryption scheme allows parties to communicate over hostile communication channels with little risk of untrusted parties revealing the contents of their communication. The use of two keys—one public and one private—reduces the burden of establishing a shared secret prior to the initial communication. While the mathematics involved in public key encryption is complex, the result is an encryption system that is well suited for untrusted communication channels.

1.1.5 The Domain Name System (DNS)

This section explains the fundamentals of the domain name system (DNS), which is an often overlooked component of the Web's infrastructure, yet is crucial for nearly every networked application. Many attacks, such as fast-flux and DNS application, take advantage of weaknesses in the DNS design that emphasize efficiency over security. Later sections will discuss some attacks that abuse the DNS and will build upon the base information provided in this section.

DNS is a fundamental piece of the Internet architecture. Knowledge of how the DNS works is necessary to understand how attacks on the system can affect the Internet as a whole and how criminal infrastructure can take advantage of it.

The Internet Protocol is the core protocol the Internet uses. Each computer with Internet access has an assigned IP address so that other systems can send traffic to it. Each IP address consists of four numbers between 0 and 255 separated by periods, such as 74.125.45.100. These numbers are perfect for computers that always deal with bits and bytes but are not easy for humans to remember. To solve this problem, the DNS was invented in 1983 to create easy-to-remember names that map to IP address.

The primary goal that the designers of the DNS had in mind was scalability. This goal grew from the failure of the previous solution that required each user to download a multithousand-line file named *hosts.txt* from a single server. To create a truly scalable system, the designers chose to create a hierarchy of "domains." At the top of the hierarchy is the "root" domain under which all other domains reside. Just below the root domain are top-level domains (TLD) that break up the major categories of domains such as .com, .gov, and the country code TLDs. Below the TLDs are second-level domains that organizations and individuals can register with the registry that manages that TLD. Below second-level domains are the third-level domains and so forth, with a maximum of 127 levels. Exhibit 1-10 shows how

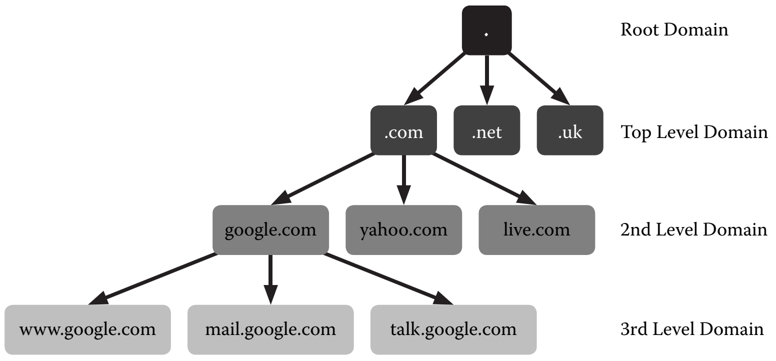


Exhibit 1-10 The hierarchical structure of the domain name system (DNS).

the hierarchical nature of the DNS leads to a tree-like structure consisting of domains and subdomains.

Separating domains in this way allows different registries to manage the different TLDs. These registries are responsible for keeping the records for their assigned TLD and making infrastructure available to the Internet so users can map each domain name to its corresponding IP address.

The DNS uses computers known as *name servers* to map domain names to the corresponding IP addresses using a database of records. Rather than store information for every domain name in the system, each DNS server must only store the information for its domain. For instance, the name server gotgoogle.com keeps information for www.google.com and mail.google.com but not for www.yahoo.com. Name servers are granted authority over a domain by the domain above them, in this case .com. When a name server has this authority, it aptly receives the title of *authoritative name server* for that domain.

The hierarchical nature that defines the DNS is also a key to the resolution process. *Resolution* is the process of mapping a domain to an IP address, and *resolvers* are the programs that perform this function. Due to the nature of the resolution process, resolvers fall into two categories: recursive and nonrecursive. Exhibit 1-11 shows the steps required for a resolver to complete this process. The first step in resolving www.google.com is contacting the root name server to find out which name server is authoritative for .com. Once the resolver has this information, it can query the .com name server for the address of the google.com name server. Finally, the resolver can query the google.com name server for the address of www.google.com and pass it on to a Web browser or other program.

Exhibit 1-11 depicts the most common way for systems to resolve domain names: by contacting a recursive DNS server and allowing it to do the work. A nonrecursive resolver (like the one used by a home PC) will only make a single request to a server, expecting the complete answer back. Recursive resolvers follow the chain of domains, requesting the address of each name server as necessary until reaching the final answer. Using recursive DNS servers also makes the system much more efficient due to caching. Caching occurs when a DNS server already knows what the answer to a question is, so it does not need to look it up again before responding to the query. The addresses

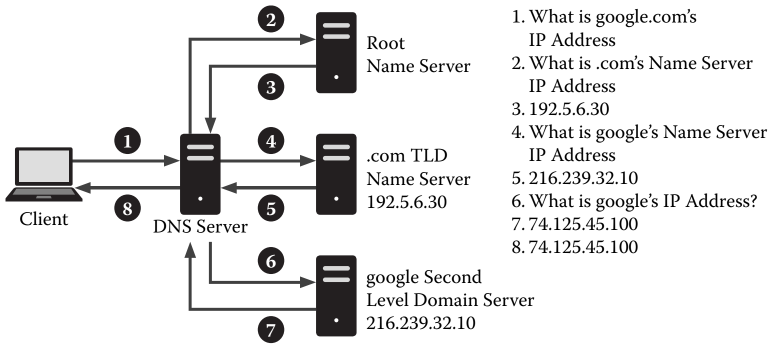


Exhibit 1-11 Resolution of google.com using a recursive DNS server.

of the root server and the .com server are usually cached due to the frequency with which systems request them.

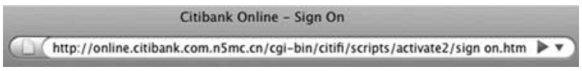
The DNS stores information in Resource Records (RR). These records are separated by type, and each one stores different information about a domain. RFC1035 defines the variety of different RR types and classes, including the most common types: A, NS, and MX.[11] An *A record* maps a domain to an IP address. NS records provide the name of that domain's authoritative name server. The *NS record* includes an additional section with the type A records for the name servers so the resolver can easily contact them. The *MX records* refer to mail exchange domains used to send e-mail over Simple Mail Transfer Protocol (SMTP). Like an NS record, MX records include an additional section to provide type A records to the domains included in the MX record. The following is an example of a query for the www.google.com A record and the resulting answer.

```
;; QUESTION SECTION:
;; NAME                   CLASS     TYPE
;www.google.com.          IN        A
;; ANSWER SECTION:
;; NAME              TTL  CLASS  TYPE   DATA
www.google.com.      180  IN     A      64.233.169.147
```

In the question section, the resolver has specified that it wants the A record for www.google.com in the Internet class (specified by IN). During the development of DNS, additional classes were created, but the Internet class is the only one commonly used today. The answer session includes the information from the question, the IP address for the domain, and a time-to-live (TTL) value. The TTL specifies the number of seconds for which the data in the record are valid. This value is the key to the caching system described above; as without a TTL, the servers would not know how long any particular data could be cached.

1.1.5.1 Security and the DNS As a fundamental part of the modern Internet, the security of the DNS is important to all Internet users. In the previous discussion of how the DNS system works, it is important to note that no authentication of results ever occurred. This makes the system vulnerable to an attack known as *DNS cache poisoning*, wherein an attacker tricks a DNS server into accepting data from a nonauthoritative server and returns them to other resolvers. Extensions to the DNS protocol, known as *DNSSEC*, solve this problem using cryptographic keys to sign RRs.[12] While this system has not yet been widely adopted, VeriSign, the company responsible for management of the root domain, deployed DNSSEC for the root DNS servers on July 16, 2010. This is an important step in the deployment of DNSSEC as it provides a single trust anchor that other domains can use to streamline deployment.[16] Protection of credentials used to manage domains with registrars is also key to the security of the DNS. In December 2008, attackers gained access to the credentials that control many domains, including checkfree.com, and used them to install a banking Trojan on visitors' systems. Legitimate use of DNS also has implications for security professionals. Fast-flux networks rely on very short DNS TTL values to change the IP address associated with a domain rapidly. Phishing attacks that employ domain names similar to those registered by financial institutions also employ the DNS. Attackers can exploit the length of a domain name to create phishing domains that mimic legitimate banking domains to steal information. Exhibit 1-12 shows a five-level domain of online.citibank.com.n5mc.cn that may appear to belong to CitiBank but is actually a subdomain of n5mc.cn.

Organizations that want to issue takedown requests for these domains need to understand how the DNS works so they can take the correct actions.


Citibank Online – Sign On

http://online.citibank.com.n5mc.cn/cgi-bin/citifi/scripts/activate2/sign on.htm

Exhibit 1-12 A long phishing domain appearing to belong to CitiBank.

1.1.6 Firewalls

The Internet of today is in stark contrast to the close-knit group of research networks that made up the Internet forty years ago. As the Internet has grown, the need to protect networks and even individual computers has become a major concern. To this end, devices and software that fall under the banner of "firewall" have become a necessity for any and all computers connected to the Internet that the user wants to remain safe. While the term firewall may conjure different images for different people, the basic concept of a firewall is simple: Keep the bad people out of our computer. In this section, explore the concept of firewalls, what they really do, and how they do it.

1.1.6.1 History Lesson The Internet turned forty years old in 2009, but the use of devices to separate one network from another, undesirable network, did not occur until the late 1980s.[13] At the time, network administrators used network routers to prevent traffic from one network from interfering with the traffic of a neighboring network. Enhanced routers introduced in the 1990s included filtering rules. The designers of these routers designated the devices as *security firewalls*. The configuration of these special routers prevents unnecessary or unwanted traffic from entering a company's network boundaries. The routers used filtering rules to determine which network traffic administrators considered good and which traffic they considered bad, but the use of router filtering rules was cumbersome to maintain as networks continued to evolve.

The next generation of security firewalls improved on these filter-enabled routers. During the early 1990s, companies such as DEC, Check Point, and Bell Labs developed new features for firewalls. Check Point, for instance, eased the technical expertise requirements for configuring firewalls by providing user-friendly interfaces while at the same time providing administrators with new configuration options for refined rule sets.

1.1.6.2 What's in a Name? The question remains: what exactly is a firewall? Firewalls are network devices or software that separates one trusted network from an untrusted network (e.g., the Internet) by means of rule-based filtering of network traffic as depicted in Exhibit 1-13. Despite the broad definition of a *firewall*, the specifics of what makes

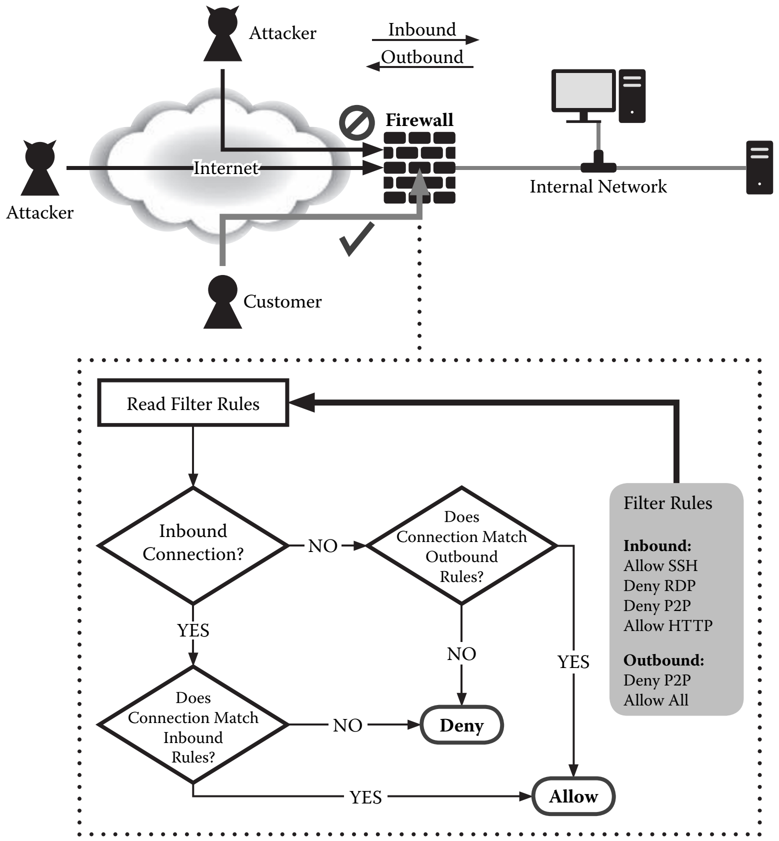


Exhibit 1-13 A basic firewalled network.

up a firewall depend on the type of firewall. There are three basic types of firewall: packet-filtering firewalls, stateful firewalls, and application gateway firewalls. While each of these different firewall types performs the same basic operation of filtering undesirable traffic, they go about the task in different manners and at different levels of the network stack.[14]

While Exhibit 1-13 identifies the firewall as a separate physical device at the boundary between an untrusted and trusted network, in reality a firewall is merely software. This does not mean that physical, separate devices are not firewalls, but merely that these devices are simply computers running firewall software. Host-based firewalls have found their way into most operating systems. Windows XP and later versions have a built-in firewall called the Windows Firewall.[15] Linux- and Unix-based computers use ipchains[16] or iptables[17] (depending on the age and type of the operating system [OS]) to perform firewall functionality;

therefore, it is important to understand that firewalls can exist at different locations within a network, not just at the perimeter of a network.

1.1.6.3 Packet-Filtering Firewalls The most rudimentary of firewalls is the packet-filtering firewall. Packet-filtering firewalls work at the IP level of the network. Most routers integrate this type of firewall to perform basic filtering of packets based on an IP address. The principle behind packet-filtering firewalls is that the firewall bases the decision to allow a packet from one network into another network solely on the IP address of the source and destination of the packet. For instance, if the firewall administrator defines the following packet-filtering rules, and if a packet from host 1.1.1.1 is destined for host 2.2.2.2, the firewall allows the packet to pass.

```
ALLOW host 1.1.1.1 to host 2.2.2.2
DENY ALL
```

On the other hand, if the packet originated from 2.2.2.2 with a destination of 3.3.3.3, the firewall would stop the packet from traversing the network any further due to the DENY ALL rule. Generally, firewalls contain an implied DENY ALL rule. If the administrator fails to include the DENY ALL rule, the firewall, after exhausting all of the rules in the filter set, will default to the DENY ALL rule and deny the traffic, since it did not match any defined rule.

Packet-filtering firewalls can also expand on the basic principle of IP-address-only filtering by looking at the Transmission Control Protocol (TCP) or User Diagram Protocol (UDP) source and destination ports. In this mode, the firewall operates in nearly the same fashion as the packet-filtering firewalls operating on the IP address. For a packet to pass through the firewall, the source IP and port and the destination IP and port must match at least one rule in the filter list. More advanced routers and even some higher-end switches offer this functionality.

To limit the exposure of a system to only necessary ports, administrators use port filtering. For example, a collocated Web server (a server hosted by a third party) will typically open HTTP and HTTPS ports for a customer's server to the Internet, but the administrator will restrict the secure shell (SSH) port on the firewall to only allow connections from the hosting company's network. This technique combines the use of IP and port filtering to allow administration of resources from specific networks (e.g., the company's network or trusted third-party networks) while allowing public services (e.g., HTTP) the necessary Internet exposure.

It is worth pointing out a common design feature of firewalls: the rule set's priority. The majority of (if not all) firewalls will use the first rule that exactly matches the conditions of the packet under observation. This means that in the previous example, the first rule matched the packet originating from 1.1.1.1 destined for 2.2.2.2 and the firewall did not continue to apply the remaining rules. Similarly, for the packet originating from 2.2.2.2 destined for 3.3.3.3, the last rule, DENY ALL, matched, which resulted in the firewall dropping the packet. This behavior leads to an interesting optimization problem for firewall administrators. To reduce network delay introduced by the firewall, administrators will typically move the most likely packet-filtering rule to the top of the list, but in doing so, they must ensure that the rule does not negate another rule further down the list. An example of such an optimization in which the administrator erroneously organized the packet filter list would be

```
ALLOW host 1.1.1.1 to host 2.2.2.2
DENY ALL
ALLOW host 3.3.3.3 to 1.1.1.1
```

In this example, the firewall administrator has placed an ALLOW rule after the DENY ALL rule. This situation would prevent the processing of the last ALLOW rule.

1.1.6.4 Stateful Firewalls Simple packet-filtering firewalls suffer from one significant downside: they do not take into consideration the state of a connection, only the endpoints of the connection. Stateful firewalls allow only properly established connections to traverse the firewall's borders. While packet filtering is still a key element of these firewalls, the firewall also pays attention to the state of the connection.

Once the firewall allows a successful connection between two hosts using the three-way TCP handshake,[18] the firewall records the occurrence of a valid session between the two hosts. If an attacker attempts to generate an invalid session, such as by sending an ACK (acknowledgment) prior to sending a SYN (synchronize), the firewall identifies the packet as an invalid state and subsequently blocks the connection.

After a host establishes a valid session, however, communication between the two hosts can occur unrestricted and without requiring the firewall to rerun the list of packet filters.

It is the ability to determine the order and state of a communication session that allows stateful firewalls to make faster determinations about incoming packets. Of course, it is important that these firewalls do not run out of memory from storing the state of stale connections. To avoid this problem, stateful firewalls will purge state information for sessions that have "gone quiet" for a significantly long period. Once a session has expired, the next packet originating from either host will result in the firewall verifying the packet against packet-filtering rules and the establishment of a new session.

1.1.6.5 Application Gateway Firewalls Application gateway firewalls, also known as *proxies*, are the most recent addition to the firewall family. These firewalls work in a similar manner to the stateful firewalls, but instead of only understanding the state of a TCP connection, these firewalls understand the protocol associated with a particular application or set of applications. A classic example of an application gateway firewall is a Web proxy or e-mail-filtering proxy. A Web proxy, for instance, understands the proper HTTP protocol and will prevent an improperly constructed request from passing. Likewise, an e-mail-filtering proxy will prevent certain e-mails from passing based on predefined conditions or heuristics (for example, if the e-mail is spam).

These proxies also prevent unknown protocols from passing through. For example, a properly configured HTTP proxy will not understand an SSH connection and will prevent the establishment of the connection (see Exhibit 1-14). This level of packet inspection cannot occur with either a packet-filtering or stateful firewall, as neither firewall type looks at the application layer of the network stack. By identifying improperly constructed packets for a given protocol, the application gateway firewalls may prevent some types of protocol-specific attacks; however, if a particular protocol's definition allows for such a vulnerability, the gateway will provide no protection.

1.1.6.6 Conclusions Firewalls come in a variety of forms, from simple packet filtering to the more complex proxy. The topic of firewalls is complex and extremely well documented. Authors from the IT

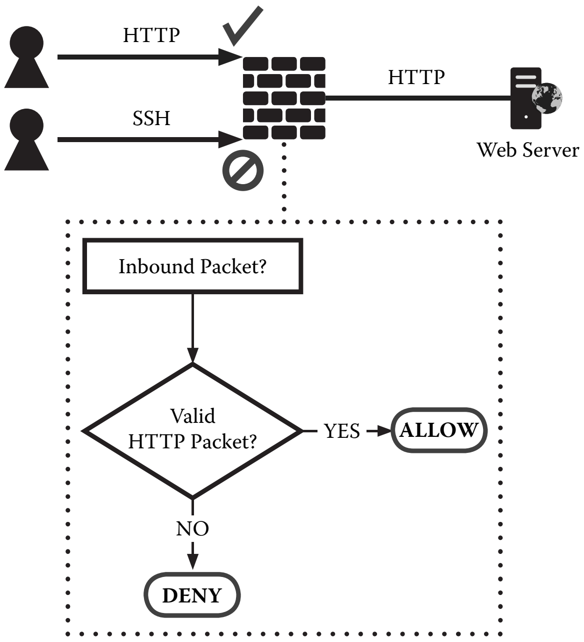


Exhibit 1-14 An application gateway filtering known and unknown protocols.

security community have dedicated entire books to the subject of designing, administering, and implementing firewalls. To understand the importance of firewalls, the minute details of their operation can be avoided, but it is critical to understand the high-level concepts of their operation. Understanding the basics of how firewalls process traffic and how that processing prevents unwanted intrusions is the key to understanding the security of firewalls.

Like antivirus solutions, the impression that firewalls will stop all evils of the Internet is overstated at best. Firewalls provide a single layer of defense in the larger scheme of defense in depth. While firewalls can reduce the attack surface of a server by blocking unnecessary ports from the Internet at large, firewalls cannot protect resources that are vulnerable to specific vulnerabilities such as buffer overflows and privilege escalation attacks.

1.1.7 Virtualization

Technology has advanced to the point that server consolidation through virtualization can help tame the cost of infrastructure deployment and operation by reducing the number of servers required to perform the same level of operational standards, given that enterprises typically underutilize the full capacity available in physical servers. This section explores the history, concepts, and technologies of virtualization.

1.1.7.1 In the Beginning, There Was Blue … Infrastructure resources such as servers are expensive. This expense comes from the cost of the physical hardware, the cost associated with supplying power to the servers, the cost to cool and maintain the proper operating environment for the servers, and the cost of administering the servers. For large infrastructures with deployments of tens to tens of thousands of servers, the cost of running these servers can quickly balloon, resulting in extremely high operational costs. To alleviate some of these administrative costs, organizations are turning to virtualization.

Virtualization at its most fundamental level is the simulation or emulation of a real product inside a virtual environment. Recent efforts by many companies to capitalize on the wave of cloud computing have given the term new importance in the IT and business communities, but the term *virtualization* is older than most realize. In the 1960s, researchers at the IBM Thomas J. Watson Research Center in Yorktown, New York, created the M44/44X Project. The M44/44X Project consisted of a single IBM 7044 (M44) mainframe that simulated multiple 7044s (44X). The M44/44X Project was the first to use the term *virtual machine* (VM) to describe simulating or emulating a computer inside another computer using hardware and software.

For decades, the use of virtual machines inside mainframes has been common practice. The use of these virtual machines gives mainframes the ability to act not as a single machine but as multiple machines acting simultaneously. Each virtual machine is capable of running its operating system independent of the other virtual machines running on the same physical machine. In this sense, the mainframe effectively turns one machine into multiple machines. Mainframes only represent the beginning of the virtualization technology and are by no means the only systems that provide the service.

1.1.7.2 The Virtualization Menu Virtualization comes in many forms such as platform and application virtualization. The most recognized form of virtualization is platform virtualization and is the method of virtualization detailed in this section. Platform virtualization is a broad category that contains several variations on a theme. The most predominant platform virtualization techniques[19] include full virtualization, hardware-assisted virtualization, paravirtualization, and operating system virtualization. Each of these techniques accomplishes the task of virtualization in different ways, but each results in a single machine performing the function of multiple machines working at the same time.

With the exception of operating system virtualization, the high-level representation of a virtual machine is largely consistent amongst the various virtualization techniques. Each technique, to varying degrees, presents a virtual hardware platform on which a user can deploy an operating system. Unlike emulation systems explained later in this section, virtualization systems require that the virtual machine match the basic architecture of the host machine (the machine running the virtual machines). This means that a standard x86 host is incapable of hosting a virtual PowerPC-based system (such as the older Apple Macintosh systems). The distinction between the different virtualization techniques exists due to the way the virtual machine application, commonly referred to as the *virtual machine monitor* (VMM) or *hypervisor*, partitions the physical hardware and presents this hardware to virtual machines.

Virtualization systems consist of several key components: a VMM, physical hardware, virtual hardware, virtual operating systems, and a host (or real) operating system. Exhibit 1-15 illustrates the relationship of these components. The key component, the component that makes virtualization possible, is the VMM.

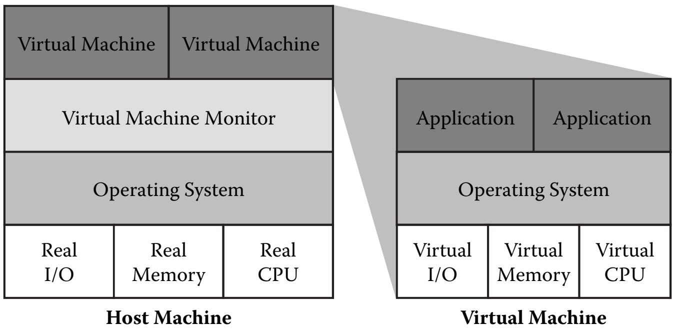


Exhibit 1-15 The relationship between virtual machines and a host machine.

The VMM is the application layer between the various virtual machines and the underlying physical hardware. The VMM provides the framework for the virtual machine by creating the necessary virtual components. These components include, but are not limited to, hardware devices like network interface cards (NICs), sound cards, keyboard and mouse interfaces, a basic input–output system (BIOS), and virtual processors. It is the responsibility of the VMM to mate the needs of the virtual machine with the available physical resources. The manner in which the VMM handles these needs dictates the type of virtualization technique employed.

1.1.7.3 Full Virtualization Full virtualization, as the name implies, strives to provide the most realistic, completely accurate virtual representation of the real hardware. For x86-based architecture, this is problematic. The x86 family of processors offers different levels of privilege to running code. Known as *rings*, these levels of protection are in place to prevent lower privileged code, such as that found in a standard application, from interfering or corrupting higher privileged code such as the kernel of the operating system.

The most privileged code level, known as *ring-0*, typically houses the kernel, or core, of the computer's operating system. Code that executes in ring-0 can manipulate the most sensitive components of the computer freely. This ability is required for operating systems to manage memory, allocate time slices to particular processes (used for multitasking), and monitor and maintain input–output (I/O) operations like hard drive and network activity. When a VMM uses full virtualization, the VMM attempts to execute code in the virtual machine in the exact manner a physical machine would. The VMM must ensure that while faithfully executing the VM's code, the VM's code does not interfere with the host machine or other VMs.

To make virtualization faster and more efficient, virtual machine applications such as VMware utilize the host machine's processor to execute instructions requested by the virtual machine. For example, if the virtual machine requests to move memory from one location to another location, the VMM would execute the instructions natively on the host machine and post the result to the virtual machine. This requires significantly less processor time and fewer resources than emulating the CPU, resulting in a faster virtual machine.

The problem that many in the virtualization sector faced was the way certain x86 ring-0 instructions operate. Based on its architecture, the x86 cannot virtualize several of its instructions without encountering unknown or undesired effects. To avoid this hurdle, the VMware family of virtual machine applications runs the virtual machine in a less privileged ring (such as ring-3 or -1) while placing the VMM in ring-0. When the virtual machine's operating system, for instance, attempts to execute a ring-0 command, an exception occurs that the CPU gives to the handler in ring-0. With the VMM residing in ring-0, the VMM can translate the offending instruction into a series of virtual machine operations that produce the same result without requiring the host machine to execute an instruction that would produce instability. VMware refers to this technique as *binary translation*.

1.1.7.4 Getting a Helping Hand from the Processor As the virtualization technology has matured from a software perspective, the hardware manufacturers have begun to show interest in the field, which opens the door for hardware-assisted virtualization. Recently, Intel and AMD released newer x86-based processors that incorporate features known as *processor extensions* to aid virtualization. The processor extensions give Intel's Virtualization Technology (VT)[20] and AMD's AMD-V[21] chip-level solutions to the issue of privileged x86 instructions that the VMM cannot virtualize. These technologies provide an even more privileged layer than ring-0 in which the VMM resides.

With hardware-assisted virtualization, the VMM operates in a new root mode privilege level—a level below ring-0. The processor extensions allow the VMM to operate in this sub-ring-0 privilege level while allowing the virtual machine's operating system access to the privileged ring-0. When the virtual machine's operating system executes an instruction that would cause instability in the host machine's operating system, the hardware passes the request to the VMM, which resides in a separate processor space by virtue of the processor extensions in order to handle the offending instruction. This allows the virtual machine's operating system to run largely unobstructed (thus reducing overhead). At the same time, the host processor ensures that the virtual machine's operating system does not impede the host operating system since the VMM will handle conditions that would cause instability between the two competing operating systems.

Hardware-assisted virtualization is an extension of full virtualization. Like full virtualization, hardware-assisted virtualization provides the virtual machine with a completely virtual hardware system. The advantage of hardware-assisted virtualization is the possibility that with a suitably designed system, the CPU can more efficiently handle instructions generated from the guest operating system that would otherwise cause instability.

1.1.7.5 If All Else Fails, Break It to Fix It Developed prior to the introduction of hardware-assisted virtualization technologies in the x86 architecture, paravirtualization provides a solution to the nonvirtualizable instruction problem present in the x86 processors. Unlike full virtualization, which runs the virtual machine's operating system in a ring with less privilege than ring-0, paravirtualization allows the virtual machine's operating system to run in ring-0 after modifying the system to restrict the dangerous x86 instructions. Paravirtualization breaks instructions that would otherwise cause instability in the host machine and replaces these instructions with calls to the VMM to allow the VMM to handle the instructions using the appropriate actions. The result is a virtual machine's operating system and applications running in the rings that the developers originally intended, but at the cost of modifying the kernel of the virtual machine's operating system.

The obvious disadvantage of paravirtualization is the required modification to the virtual machine's operating system. For closed-source operating systems, it is difficult to modify the kernel fully to meet the requirements of paravirtualization. Most paravirtualization-based virtual machines run modified Linux operating systems. An example of a paravirtualization system is the open-source Xen[22] application for the Linux operating system. Commercial applications such as VMware support the paravirtualization mode, but the choice of the virtual machine's operating system limits the usefulness of paravirtualization.[23]

1.1.7.6 Use What You Have Operating system-assisted virtualization differs dramatically from the underlying concept that ties full

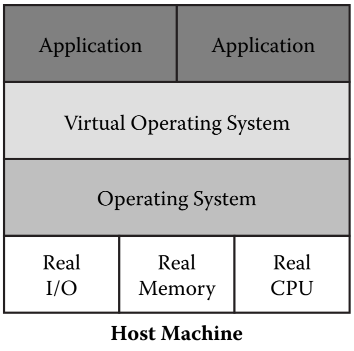


Exhibit 1-16 Operating system-assisted virtualization.

virtualization, paravirtualization, and hardware-assisted virtualization together. Instead of providing a realistic virtualized machine complete with dedicated I/O, memory, and processors, operating system-assisted virtualization provides an application with the illusion of a dedicated operating system. This virtualization technique is common on Linux- and Unix-based systems via chroot,[24] FreeVPS,[25] FreeBSD Jail,[26] and others.

Whereas the other virtualization techniques provide a virtual machine capable of supporting ring-0 instructions, operating system-assisted virtualization provides only user mode resources. This means that the virtual environment is unable to run privileged instructions, which require ring-0. This type of system allows a single operating system instance to run multiple applications in isolation while still providing them with the necessary operating system resources such as disk and network access. Exhibit 1-16 depicts this form of virtualization.

1.1.7.7 Doing It the Hard Way Emulators operate on the same basic principles as virtualization systems except that the requirement that the host machine must match the same basic architecture as the virtual machine does not limit emulators. Emulators, as the name implies, emulate all aspects of the virtual machine's hardware. While virtualization systems will offload the execution of a virtual machine's operating systems or applications to the host machine's processor, emulators do not. Emulation systems translate the virtual machine's instructions into instructions that can run on the host machine.

The CPU of a virtual machine inside an emulator can be radically different from the host machine's CPU. For instance, emulators exist that allow x86 architectures to run virtual machines that emulate older Apple Macintosh operating systems.[27] The power of emulators to run radically different architectures than the host machine's architecture comes at a price. For every CPU instruction that a virtual machine's CPU executes, the host machine must translate the instruction into a series of instructions that the host machine's CPU can execute. This constant translation of CPU instructions from the virtual CPU to the host CPU can result in a significant amount of overhead. The overhead, of course, results in a significant performance penalty.

Emulators are not strictly for dissimilar architectures. Emulators can run virtual machines of the same architecture as the host machine. VMware, if specifically configured to do so, can emulate the x86 architecture including the CPU within a virtual machine. The advantage of this behavior is to provide an even more realistic virtual environment that does not rely on the translation of certain ring-0 instructions.

1.1.7.8 Biting the Hand That Feeds Virtualization of infrastructure resources may reduce the number of physical servers required; at the same time, it is important to understand that virtualization may introduce risks. While many virtualization systems attempt to provide rigid boundaries between the host system and the virtual machines running on the host system, the possibility exists that malicious actors may attempt to breach the boundaries. As virtualization systems have gained popularity, attackers have begun focusing on the weaknesses within these systems.

Regardless of the virtualization method, the fact remains that the virtual machine's operating system and its associated applications run on the host system at some point in time. When the VMM gives the virtual machine access to physical resources such as video devices, the possibility exists for the separation between the virtual and host machine to crumble. In 2009, researchers from Immunity released a presentation[28] at Black Hat 2009 in which they demonstrated that from within a virtual machine, an attacker could gain access to the host machine's memory. Similarly, in 2009, researchers at Core Labs[29] released an advisory describing a method for accessing the host operating system from within a virtual machine.

Virtualization systems are complex systems and, as such, are prone to vulnerabilities. A vulnerability in an operating system or an application may lead to the compromise of a single server within an infrastructure. When that vulnerable operating system or application is running within a virtual machine that is itself vulnerable, the effects of a single compromise can amplify across all other virtual machines within the same physical machine. Moreover, since cloud computing heavily relies on virtualization, this class of vulnerability can affect not only a single enterprise but also any enterprise that operates within the same virtual infrastructure. Therefore, it is important to understand that separating sensitive virtual machines (i.e., VMs that handle personally identifiable information) from public virtual machines (i.e., VMs that run a company's public Web server or mail server) can reduce the impact associated with the VM boundary vulnerability.

1.1.7.9 Conclusion Virtualization has many advantages ranging from server consolidation to program isolation. While the technology has been available in some form for decades, advancements in modern computing hardware have led to a more widespread adaptation of the technology. Even at its current level of development, virtualization is already making major inroads into the IT community. Virtualization is a key component of the recent influx of new cloud-computing technologies currently on the market. The growth of the virtualization market is far from reaching its peak.

Before deploying a large virtualized infrastructure, it is important to understand the risks associated with virtualization. When the boundary between a virtual machine and a host machine becomes transparent (through vulnerabilities), the risk of significant data exposure and system compromise increases dramatically. Classifying the data and types of virtual machines that run on the same physical machine can reduce this exposure.

1.1.8 Radio-Frequency Identification

At the 20XX DEFCON conference, Chris Paget of H4RDW4RE LLC presented his talk on debunking the myths around radio-frequency identification (RFID). While many organizations use these devices for authentication, they often are not aware of how the technology works or how secure it is. In this section, RFID and the security and privacy concerns around the technology are explained.

The term RFID does not describe one particular technology, but a group of technologies used for the purposes of identification using radio waves. RFID devices, commonly referred to as tags, are commonplace in everyday life. To name just a few of their many uses, the devices enable electronic tollbooths, inventory tracking, and authentication systems. RFID has been the source of much controversy in the last decade as security and privacy concerns began to emerge. Depending on the way people use RFID tags and the security measures deployed to protect them, these concerns range from minor to severe. To understand the security concerns of RFID, it is first important to understand how they operate. In RFID communication, there are two actors: the interrogator (reader) and the device (tag). The reader is a device, typically connected to a computer, capable of receiving and interpreting data from an RFID tag. The tag is a device varying in complexity that sends back the specific identification information unique to the tag. Some tags simply emit the same information each time the reader interrogates them, and others include processing systems capable of complex cryptographic operations.

There are three primary types of RFID tags when categorized by power sources. These types include *passive*, *battery-assisted passive*, and *active*. Both types of passive tags activate when they receive a signal from the reader. Passive tags that operate without any battery power use the power in the signal sent by the reader to power them and send back their responses. Battery-assisted passive tags activate after the reader sends a signal, but use battery power to construct and send their responses. Because passive tags only use the power they can scavenge from the reader's signal, they have limited ranges compared to battery-assisted devices. The third type of RFID device is an active tag. Unlike their passive cousins, active tags can transmit signals without activation by a reader.

1.1.8.1 Identify What? The data that an RFID tag contains vary depending on its application. The simplest and most common RFID tag is the electronic product code (EPC). EPCs are the RFID

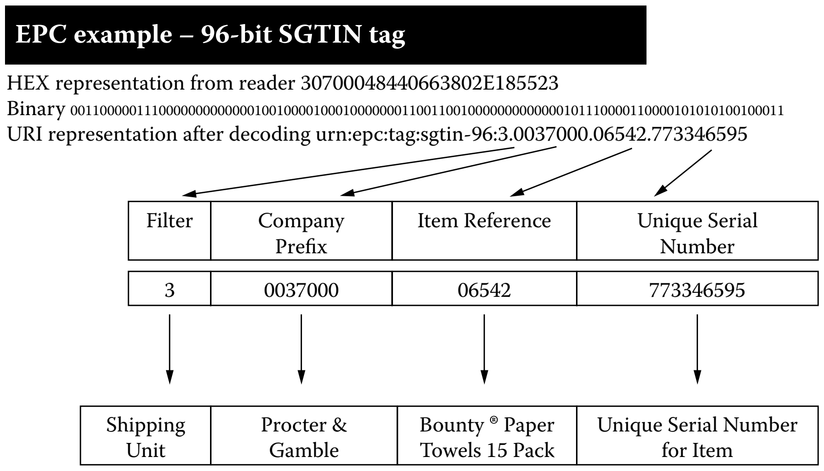

EPC example – 96-bit SGTIN tag
HEX representation from reader 30700048440663802E185523
Binary 001100000111000000000000000100100001000100000000110011001000000000000000101110000110000101010100100011
URI representation after decoding urn:epc:tag:sgtin-96:3.0037000.06542.773346595
Filter
Company Prefix
Item Reference
Unique Serial Number
3
0037000
06542
773346595
Shipping Unit
Procter & Gamble
Bounty ® Paper Towels 15 Pack
Unique Serial Number for Item

Exhibit 1-17 An example electronic product code (EPC). *Source*: http://assets.devx.com/articlefigs/16814.jpg.

equivalent of the bar code, and replacing barcodes is their primary function. EPC tags are passive RFID tags, and organizations frequently integrate them into stickers. EPCs contain information similar to that found on Universal Product Codes (UPC) but can store much more information. The data stored in a typical EPC is a 96-bit number using the specification shown in Exhibit 1-17. The data stored in an EPC is merely this number, which has no value without the ability to decode its meaning. For product tags, this number represents the product's manufacturer, type, and serial number.

While UPC codes can store enough information to enumerate all types of products, such as a pack of paper towels, EPC codes include an additional 36 bytes of data, allowing for the use of more than 600 billion unique serial numbers. Rather than identifying a general product type, such as a pack of paper towels, RFIDs can identify a specific pack of paper towels. An organization can place EPCs on any product or group of products that it would like to track. In 2005, Wal-Mart Stores, Inc. mandated that all of its suppliers must tag shipments with RFID tags. Libraries have also begun to use EPC tags to expedite book check-ins and check-outs.

Organizations and governments are using RFID tags to identify much more than common household products. Many organizations deploy RFID-equipped ID cards (commonly known as *proxy cards*, or *proximity cards*) that grant access to buildings and systems. In this

Exhibit 1-18 An American Express card with CSC functionality.

case, the number returned by the card corresponds to information about a specific individual stored in a database. If the card readers receive 0001 from John Doe's card, the security system can look up this record in its database of users and allow or deny access to the secured area. Identifying people and authorizing access based on RFID tags are fundamentally different uses of the technology compared to merely identifying products.

Obviously, there is no practical value in copying or cloning an EPC tag attached to a bag of potato chips, but copying an RFID access card can be quite valuable. If an access card worked like an EPC tag, it would always return the same 96-bit number. Anyone who could read the card could easily duplicate it and gain access to a building. To prevent this, another class of RFID tag, known as the *contactless smart card* (CSC), is much more complex than an EPC.

Similar to traditional smart cards, CSCs have the ability to store and process information. Rather than simply responding to each interrogation with the same number, CSCs use cryptography to hide their information and sometimes confirm the identity of the reader before divulging sensitive information.

Examples of CSC products are contactless credit cards issued by VISA, MasterCard, and American Express (see Exhibit 1-18); most access control badges; and the new U.S. electronic passport. The security of these devices is incredibly important, because cloning or tampering with them could allow an attacker to steal the owner's money or identity without ever coming into contact with the owner.

1.1.8.2 Security and Privacy Concerns The implementation of RFID security measures and the privacy concerns that wireless identity tags present are the subjects of much controversy. In 2005, researchers at Johns Hopkins University, led by Dr. Avi Rubin, broke the encryption used by millions of RFID-enhanced car keys and Exxon's Speedpass RFID payment system.[30] These RFID-enhanced car keys use RFID technology as a lock-picking prevention mechanism. If the proper RFID tag is not in proximity of the reader when a user turns the key in the ignition, the car will not start. Speedpass allows Exxon customers to link a credit card to their keychain tokens to make purchases from Exxon gas stations. Rubin's team discovered that both of these devices only use 64-bit encryption to protect the tags. While this encryption may have been sufficiently complex to prevent brute force attacks when manufacturers introduced the system in 1993, this level of protection is no longer sufficient.

At DEFCON 17, Chris Paget, a security researcher with H4RDW4RE LLC, presented his talk on RFID myth busting. In his talk, Paget debunked the myth that readers can only interrogate RFID tags at short ranges. One ID card Paget has researched is the U.S. enhanced driver's license (EDL). EDLs contain RFID tags and act as passports for passage between the United States and its bordering countries. These cards can easily be read at distances of more than twenty feet and contain no encryption at all. Earlier this year, Paget posted a YouTube video in which he demonstrated collecting EDL information from victims without them ever being aware of his presence.[31] To target a specific group, the attacker could build an antenna into a doorframe and collect the ID of each person who entered the room. Because of these cards' lack of encryption, attackers can easily clone them to steal victims' IDs without their knowledge.

Beyond the security concerns of identity and data theft, RFID tags also have ramifications for personal privacy. Because attackers can read the tags at a distance, they can read them without the user's knowledge. Even tags that do not contain any identifying information may identify a specific person when grouped with additional information.

Imagine that every shoe made contained an RFID tag that the manufacturer could use to track inventory. This RFID tag alone is not a significant privacy concern, but if a person buys this shoe with a credit card, that specific RFID tag would then link to the buyer's name in a retailer's database. The retailer could then scan every user entering the store to see if the shopper was wearing any clothing associated with a specific customer. The retailer could use this to display targeted advertisements to each customer and track his or her location in each store, similar to the scenario played out in the film *Minority Report*.

Any individual or organization considering deploying RFID technology or carrying RFID-enabled devices should seriously study these concerns. Reading RFID tags at long distances allows attackers to track carriers without their knowledge. RFID wallets, which block signals transmitted by the devices, can provide protection against RFID readers. These wallets are typically made of metallic material through which radio frequency radiation cannot pass.

RFID tags have many advantages over technologies that require optical scans or physical contact. RFID readers can interrogate hundreds of tags at a time to perform complete inventories in a fraction of the time required for hand counting; however, using these devices for identification and authentication requires the implementation of countermeasures to protect against cloning and modification.

1.2 Microsoft Windows Security Principles

1.2.1 Windows Tokens

Access tokens and control lists limit a user's or program's access to certain systems. Granting a user the least privilege required and developing programs to require minimal privileges are effective ways of containing the potential of full system compromise to privilege escalation vulnerabilities.

1.2.1.1 Introduction The inner workings of Microsoft Windows access tokens and access control lists for objects such as processes and threads are not widely understood. Windows uses access tokens (hereafter simply referred to as *tokens*) to determine if a program can perform an operation or interact with an object. In this section, we will explain the concept of Windows tokens and process and thread access control lists.

1.2.1.2 Concepts behind Windows Tokens Tokens provide the security context for processes and threads when accessing objects on a system. These objects, also known as *securable objects*, include all named

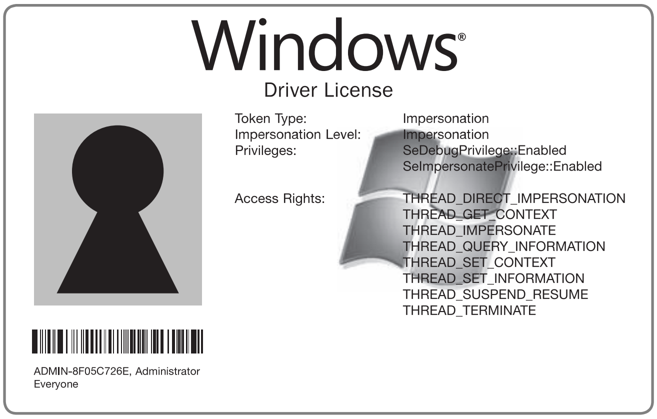

Windows®
Driver License
Token Type: Impersonation
Impersonation Level: Impersonation
Privileges: SeDebugPrivilege::Enabled
SeImpersonatePrivilege::Enabled
Access Rights: THREAD_DIRECT_IMPERSONATION
THREAD_GET_CONTEXT
THREAD_IMPERSONATE
THREAD_QUERY_INFORMATION
THREAD_SET_CONTEXT
THREAD_SET_INFORMATION
THREAD_SUSPEND_RESUME
THREAD_TERMINATE
ADMIN-8F05C726E, Administrator
Everyone

Exhibit 1-19 A Windows token represented as a driver's license.

objects ranging from files and directories to registry keys. Tokens have four parts that include an identity, privileges, type, and access controls. Conveniently, Windows tokens share similarities with a driver's license at the conceptual level, and this idea draws analogies between the two. Exhibit 1-19 shows a license that is a visual representation of a token, and throughout this report, examples will attempt to bridge these two concepts together.

A Windows token has a portion called an identity. The token's identity describes to whom the token belongs, much like a driver's license includes the owner's name. The identity consists of two parts: a user and a group, just as the name on a driver's license consists of both a first and last name. If a person were to use a credit card, the merchant might ask to see the person's license to check if the name on the credit card is the same as the name listed on the driver's license. If the names matched, the merchant would allow the person to use the credit card. In Windows, if a user has access to a directory, Windows would check to see if the token had the same user listed. If it is the one listed, Windows would grant access to the directory.

The second piece of a token's identity is the concept of group memberships. Several users can belong to the same group to simplify resource access management. For example, if a man visits a local community center that his family paid to attend, and an employee of the facility checks the man's driver's license to see if his family's name is registered, the employee would let him use the facility if the names matched. Exhibit 1-19 shows the identity of the token by displaying the user, group name, administrator, and ADMIN-8F05C726E, respectively, as it would on a driver's license.

A token can have a variable number of groups that allow programs to restrict permissions in a more granular fashion. For example, when a police officer pulls over a motorcyclist and asks for his or her driver's license, the officer is ensuring that the rider has a motorcycle license by checking for an M rating, signifying that the individual completed motorcycle riding and safety tests. Similarly, a Windows program may not want certain operations carried out by programs using a specific token. The program may restrict or add groups to a token to allow even more fine-grained control. For example, in Exhibit 1-19 above, the administrator user belongs to the ADMIN-8F05C726E group but also resides in the everyone group.

The privileges of a token specify certain operations that the holder of the token can perform. Two of the most familiar privileges are SeDebug and SeImpersonate. These privileges specify to the kernel what operations the user can perform with kernel objects before access control checks are considered.

The SeDebug privilege tells the kernel that the program holding this privilege can perform operations on processes and threads, which are objects a debugger would need to be able to access, without considering the access control restrictions on those objects. This concept is similar to the organ donor area on a driver's license. If someone involved in a fatal car accident has properly indicated on his or her driver's license that he or she is an organ donor, a hospital may remove the organs that the holder has designated without requiring the consent of the person's surviving relatives.

The SeImpersonate privilege allows the user to impersonate other users' tokens. Typically seen granted to system services, this privilege allows a user to acquire the access and permissions of someone else after the user authenticates. When used by a service, the service impersonates a client while accessing requested resources on the server. An example of this privilege would be if one person took another person's driver's license to use as his or her own when driving a car.

The token also has a type: it can be either a primary token or an impersonation token. Primary tokens identify processes, and impersonation tokens identify threads. Other than this assignment, the only other difference is that impersonation tokens also have an associated impersonation level.

The impersonation levels are anonymous, identification, impersonation, and delegation. With an anonymous token at its processing whim, a program cannot identify the user of the token, nor can it impersonate the token. Anonymous tokens do little more than fill function requirements that a token exists. Anonymous tokens are like a motorist having no driver's license at all; the motorist is not identifiable.

Identification tokens are the next impersonation level. A program that possesses an identification token can inspect the user of the token, the group memberships of the token, and any privileges that the token has enabled. Identification tokens are useful when a program would like to perform its own access checks against a user and is not concerned about allowing the operating system to check permissions. Identification tokens are like a motorist having a valid driver's license; the motorist is identifiable.

An impersonation-level token allows a program to perform operations on behalf of the user of the token on the local system. The program that possesses an impersonation-level token can call all of the Win32 application programming interfaces (APIs) and have the operating system perform access checks against the user. Impersonation tokens are like having the ability to change the picture and physical description on a driver's license—one can allow anybody to assume the identity listed on the driver's license.

The final level of token is a delegation token. Armed with a delegation token, a program can access both the local system and network resources on behalf of the user of the token. Delegation token use is common in situations in which the program requires the local operating system to determine if the user has access to a resource and remote systems to check if the user can perform the operation. Delegation tokens resemble the ability to modify the picture, physical description, and issuing state of a driver's license—one can allow anybody to assume the identity on the driver's license and have any state believe it is a valid local driver's license.

1.2.1.3 Access Control Lists Tokens have access control lists that describe the access that identities may request when accessing the token. These entries on the access control list either explicitly allow or explicitly deny specific types of operations on the token. The token can allow or deny reading information from the token, writing information to the token, and various other operations on the token to specific identities on the system. Together, these components form the basis behind tokens on Windows.

Tokens support access restrictions by using groups classified as denied. When determining if a token has access to a specific resource, Windows checks the access control list first to see if the token has access. Then, it will check again to see if the access requested matches the access control entry's access and group. If they match, Windows will grant the token access. This is similar to optical restrictions on a driver's license. A police officer may first check to see if the person driving the motor vehicle has a license to drive. Then the officer may check to see if the driver requires vision correction gear to drive and will consider both these pieces of information when developing a case against the driver.

These access control lists apply to both processes and threads. They allow administration of the level of access granted to various groups and users. Process and thread control lists both offer standard access rights,[32] but differ in the process- and thread-specific access rights.

Process-specific rights are a list of fourteen access permissions that apply to processes only. These rights include granular access controls that range from reading and writing to creating and terminating processes. In addition to these granular permissions, Windows includes an all-encompassing right known as PROCESS_ALL_ACCESS, which permits all process-specific rights to a user.

Thread-specific rights are thirteen access rights that apply to threads only. The permissions allow interaction with threads and include rights to suspend, resume, and terminate threads, to name a few. Like the process-specific rights, THREAD_ALL_ACCESS permits all thread-specific rights to the user.

1.2.1.4 Conclusions Access tokens and control lists limit the amount of access a user or program has to a system. Administrators should grant users the lowest level of privileges necessary to limit the amount of damage caused by compromise or a rogue user. Developers should also follow the least privilege stance while coding to reduce the impact of application misuse.

As far as an attacker is concerned, under normal circumstances, impersonation tokens with levels of impersonation and delegation are the most valuable because they increase an attacker's access to systems. Therefore, access controls gained from proper token use can limit the exposure to privilege escalation vulnerabilities and lower the chances of full system compromise.

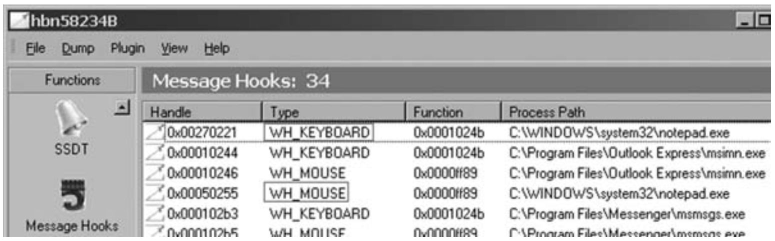
1.2.2 Window Messaging

The window-messaging queue, which handles events such as mouse clicks and keyboard input, allows program windows on Microsoft operating systems to interact. Unfortunately, malicious software may also utilize this functionality and even add message hooks for capturing data and covert communication. Monitoring window message hooks can reveal malicious behavior such as key logging or graphical user input.

Programs that run on Microsoft operating systems with visible windows can accept and handle new events using window messaging (and the window-messaging queue). Processes may send these messages to communicate with other processes. For example, window messages allow users to interact with a window and input text or use the mouse. Sending a window message activates a message hook to execute code to handle the event.

Malicious and nonmalicious programs install message hooks to process message events. For instance, notepad.exe installs a message hook for keyboard (WH_KEYBOARD) and mouse (WH_MOUSE) messages to accept user input. Exhibit 1-20 shows message hooks that IceSword detects when the user opens notepad.exe.

SetWindowsHookEx is a Windows API function that initiates a hook for these messages. It allows the author to execute a new handling



Message Hooks: 34

Handle	Type	Function	Process Path
0x00270221	WH_KEYBOARD	0x0001024b	C:\WINDOWS\system32\notepad.exe
0x00010244	WH_KEYBOARD	0x0001024b	C:\Program Files\Outlook Express\msimn.exe
0x00010246	WH_MOUSE	0x0000ff89	C:\Program Files\Outlook Express\msimn.exe
0x00050255	WH_MOUSE	0x0000ff89	C:\WINDOWS\system32\notepad.exe
0x000102b3	WH_KEYBOARD	0x0001024b	C:\Program Files\Messenger\msmsgs.exe
0x000102b5	WH_MOUSE	0x0000ff89	C:\Program Files\Messenger\msmsgs.exe

Exhibit 1-20 IceSword is one tool for viewing message hooks.

function whenever the program receives a new message.[33] Message hooks operate correctly under administrator and limited user accounts because they are necessary for users to interact with windows. Multiple processes can initialize hooks for the same message type (see Exhibit 1-20). In cases where there are multiple hooks, the most recently initialized handling functions determine whether to pass the message to other handling functions for the same message type.

The system delivers messages using a first in, first out (FIFO) message queue or by sending messages directly to a handling function. Each thread that has a graphical user interface (GUI) has its own message queue, and there is a special message queue for system messages. Whenever a window does not accept these messages within a timeout period of a few seconds, the window may show "Not Responding" until the program handles the message.

When a user logs on, he or she will also call the CreateDesktop function, which associates the desktop with the current window station. With fast user switching, for instance, multiple users can log on at the same time and each has a unique desktop. A desktop restricts window messages from other desktops, preventing a user from sending window messages to another active desktop. Each session has a unique ID, which may contain one or more desktops. The first user to log on always has session zero (Windows XP/2003 or earlier), and subsequent users have sessions one, two, and so on. Services also run in the same session (zero) as the user who logs on first, allowing the user to send or receive window messages for services.

1.2.2.1 Malicious Uses of Window Messages Malicious code authors can use window messages and hooks for malicious purposes, including monitoring, covert communication, and exploiting vulnerabilities. One malicious use of window messages is for monitoring. An attacker can use the SetWindowsHookEx function with WH_KEYBOARD to install a key logger. The diagram in Exhibit 1-21 shows message hooks for the legitimate notepad application and an additional WH_KEYBOARD message hook for a malicious key logger program, which tries to log all the keystrokes that the user types.

Malicious programs may also propagate via autorun with removable devices and use WM_DEVICECHANGE hooks to determine when users insert new devices. Even programs running with limited

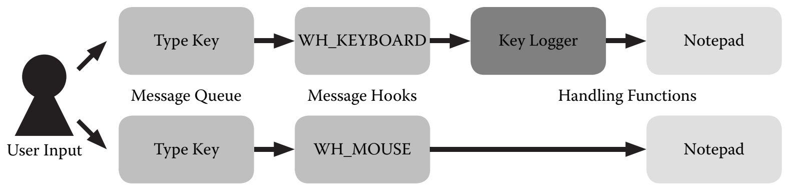


Exhibit 1-21 Malicious and benign message hooks for a keylogger and notepad.

user permissions can intercept messages intended for other processes using these types of hooks.

Window messages can also serve as a covert communication channel that is often invisible to users and administrators. Rootkits or other malicious programs can communicate using custom window messages. In this way, they can signal other processes and initiate new functionalities. Trojan horses can also install backdoor code and inject code from dynamic link libraries (DLLs) into other running processes using message hooks as the technique to activate the code. Attackers also try to hide message hooks to avoid analysts identifying their uses or malicious behaviors. Normally, programs initiate a message hook within the context of a desktop, which the system creates when it initiates a user's session; however, attackers may also call the CreateDesktop function, which allows them to create new window-messaging contexts. Trojans like Tigger avoid monitoring by creating a new desktop, which prevents messages on the default desktop from interacting with it.

Window message processing can introduce security vulnerabilities. Application code to handle messages can cause programs to crash if they do not handle unusual window messages. Privileged processes with active windows running on a limited user's desktop can also allow privilege escalation. In 2002, Chris Paget published a paper on "shatter attacks" detailing how design flaws could allow privilege escalation in the Win32 API. Paget described the design flaws through the window-messaging system that allow a process with limited user permissions to gain local system privileges. In this attack, an attacker already on the system can send a WM_TIMER message to a process running as a local system and pass a second parameter containing an address for the timer callback function, which will execute with local system privileges.[34] Microsoft disabled some more dangerous functions related to sending window messages, which could also allow privilege escalation. According to Paget, the Microsoft fixes for this vulnerability only disable certain vulnerable functions but do little to prevent the privilege escalation vulnerabilities in the window-messaging system.

1.2.2.2 Solving Problems with Window Messages Windows Vista is less susceptible to shatter attacks due to greater separation of interactive sessions. In Vista, users log on to user sessions starting at one instead of zero (which Vista reserves for system services). In this way, user applications cannot interact with system services that previously exposed their window-messaging functionalities. Microsoft has fixed problems related to privilege escalation in many of its own interactive services; however, other privileged processes from third-party developers still pose a risk to non-Vista users. More information is available from the Microsoft Developer Network (MSDN) and Larry Osterman's MSDN blog.[35]

Services that must run with increased permissions should not use interactive windows when they run as a limited user. Alternatives are available on Microsoft Windows platforms to perform interprocess communication and to limit the effects of privilege escalation attacks. Using named pipes and remote procedure calls (RPCs) are some alternatives that do not depend on support for sessions, and developers should use these instead of window messages to remain compatible with Windows Vista.[36]

Analysts and researchers should monitor window message hooks to identify behavior in running programs. Attackers may use window messaging for a variety of malicious purposes including monitoring, covert communication, and exploitation. Each window message hook may reveal functionality of unknown suspicious programs, such as graphical user input, key-logging functionality, spreading via removable devices, and custom functionality.

1.2.3 Windows Program Execution

Most people rarely consider the mechanics behind the scenes when running a program on their computers. Users have become accustomed to the fact that by simply double clicking an executable or typing in a command, the operating system will magically load and run the application desired. On the Windows system, this process is much more involved than simply clicking a button. The process by which the operating system loads, executes, and schedules programs is complex. This section delves into the process of running a program from the moment the operating system begins loading the program into memory until the moment the program actually begins to execute.

A Windows executable is nothing more than a well-structured binary file that resides on a computer's hard drive. It is not until the operating system loads the executable into memory, properly initializes it, and generates a process context that the executable becomes an actual program. The procedure by which the operating system turns an executable image file into a running process varies slightly depending on which internal Windows application programming interface (API) function call loads the image.

The native Windows API contains a surprisingly large number of process generation functions, as seen in the top level of the hierarchy depicted in Exhibit 1-22. Ultimately, each of these functions (with the exception of the CreateProcessWithLogon and CreateProcessWithToken) ends in a call to CreateProcessInternalW, which in turn calls NtCreateProcessEx. The exceptions to this trend are the CreateProcessWith… functions that ultimately end in a remote procedure call (RPC) via NdrClientCall. The RPC call also terminates with a call to the internal NtCreateProcessEx function.

Regardless of the API function used to initiate the process creation, the basic steps to create the process are the same, since all functions end in a call to NtCreateProcessEx. These eight steps, as defined by Microsoft Press' "Windows Internals," are displayed in order in Exhibit 1-22. These steps make up the core of the Windows program execution system. Given the complexity of each of the steps, the remainder of this section will explore each step to provide a better understanding of what each step involves and how each contributes to the execution of a new process.

1.2.3.1 Validation of Parameters The call to NtCreateProcessEx contains a variety of parameters that the function must verify before it can attempt to load an executable image. NtCreateProcessEx must determine if these parameters are indeed valid and, if so, how they will affect subsequent operations. The API function allows the caller

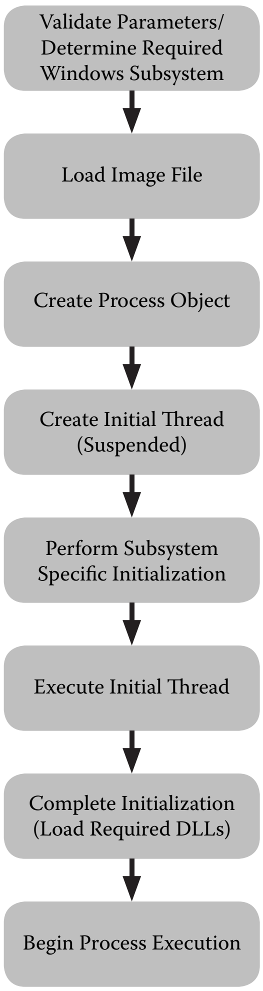

Validate Parameters/ Determine Required Windows Subsystem
Load Image File
Create Process Object
Create Initial Thread (Suspended)
Perform Subsystem Specific Initialization
Execute Initial Thread
Complete Initialization (Load Required DLLs)
Begin Process Execution

Exhibit 1-22 Windows execution steps.

to specify the scheduling priority of the new process. Windows provides a wide range of scheduling priorities that dictate how much of the central processing unit's (CPU's) time a particular process and its associated threads will receive with respect to the other processes running on the operating system at any given time; however, to set the scheduling priority, NtCreateProcessEx must determine the requested scheduling priority. The scheduling priority parameter consists of a set of independent bytes, each of which specifies a particular priority class such as Idle, Below Normal, Normal, Above Normal, High, and Real-Time. The function determines the lowest priority scheduling

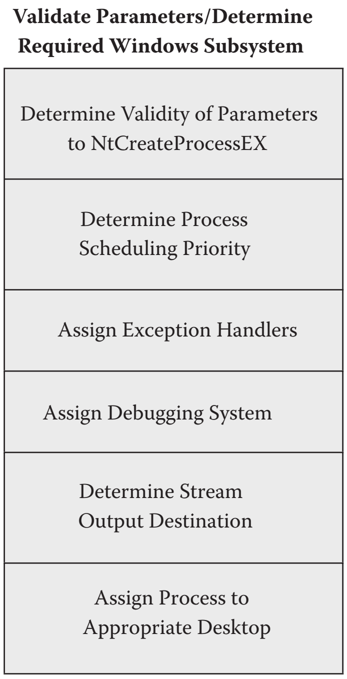

Validate Parameters/Determine Required Windows Subsystem
Determine Validity of Parameters to NtCreateProcessEX
Determine Process Scheduling Priority
Assign Exception Handlers
Assign Debugging System
Determine Stream Output Destination
Assign Process to Appropriate Desktop

Exhibit 1-23 Substeps of the *validate parameters* phase.

class specified and uses this as the priority for the new process. If the caller to NtCreateProcessEx specifies that the only scheduling priority class allowed is Real-Time (a class that attempts to take as much of the CPU's time as possible) and the caller of the function does not have sufficient scheduling privileges, NtCreateProcessEx will downgrade the request to High but will not prevent the call to NtCreateProcessEx from succeeding. When a program calls NtCreateProcessEx without an explicit scheduling priority defined, the function defaults to the Normal priority.

During the validation-of-parameters phase, NtCreateProcessEx assigns various handles and subsystems to address low-level events that may occur during the normal course of program execution. The function assigns exception-monitoring routines to handle exceptions that may occur in the program. The function also establishes whether a debugging system is required to handle debugging events. Finally, the function determines where the operating system will send data stream output if the program chooses not to use a console window (as would be the case with graphic applications). If the caller of NtCreateProcessEx does not specify a Windows desktop for the process, the function associates the current user's desktop with the process.

This prevents a process run by one user on a multiuser version of Windows (such as Windows 2003 or 2008) from starting a program on the desktop of another user logged onto the same system by accident. Exhibit 1-23 details the components of the validation-of-parameters phase.

1.2.3.2 Load Image, Make Decisions Exhibit 1-24 depicts the steps for the Load Image phase. The Windows API supports a limited set of executable images. When a user double clicks a .doc file or an .xls file, Explorer does not call one of the various CreateProcess functions with the given file, but instead maps the file extension to the appropriate program using the settings in the registry. The program responsible for the .doc or .xls file runs via the CreateProcess API functions. From this point on, we will use the generic function name *CreateProcess* for any of the process creation functions shown in Exhibit 1-22.

CreateProcess handles only a few extensions directly (or semidirectly, as will be explained shortly). The list of valid extensions that a caller can pass to the CreateProcess functions includes .exe, .com, .pif, .cmd, and .bat. One of the first steps the CreateProcess function takes during this phase is to determine the type of executable image the caller is requesting. Windows can support a variety of different applications, as indicated in Exhibit 1-25. CreateProcess does not make the determination of the executable image type by extension alone. The function loads the image into a section object[37] that maps a view of the file into a shared memory region. From the header of the image, the API can

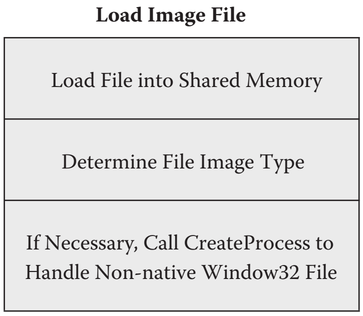

Load Image File
Load File into Shared Memory
Determine File Image Type
If Necessary, Call CreateProcess to Handle Non-native Window32 File

Exhibit 1-24 Load image file steps.

APPLICATION TYPE	EXTENSIONS	RESPONSIBLE WINDOWS IMAGE
Windows 32/64-bit	.exe	run directly via CreateProcess
Windows 16-bit	.exe	run via ntvdm.exe
MS-DOS	.exe, .com, .pif	run via ntvdm.exe
MS-DOS Command File	.bat, .cmd	run via cmd.exe/command.exe
POSIX		run via Posix.exe
OS2 1.x		run via Os2.exe

Exhibit 1-25 A support image type by CreateProcess AP.

determine if the image contains a Windows 32- or 64-bit image, an MS-DOS image, a POSIX image, or a Windows 16-bit image.

If CreateProcess determines that the image is not a native 32- or 64-bit Windows image, the exact type of image is determined and the function transfers control to the appropriate Windows image processor. For example, if the image is determined to be a portable operating system for a Unix (POSIX) image, CreateProcess calls itself again to start the Posix.exe image loader and passes the current executable (and its associated arguments) as arguments to Posix.exe. In this way, CreateProcess transfers the responsibility for loading the image to the image loader (Posix.exe), which has the resources to properly load the image into memory and provide the necessary subsystem support to execute the image. The image loader is a Windows 32- or 64-bit executable, therefore allowing the CreateProcess procedure to continue as it would with a native Windows executable.

1.2.3.3 Creating the Process Object So far in this section, we have used the term *process* as the part of the executable that Windows executes, but this is not exactly the case. A process is merely a container or an object. A process contains the necessary information for the scheduling system of Windows and the other various Windows subsystems to maintain the context (or state) of one or more related threads. A thread is a self-contained set of executable instructions that interacts with the operating system and its related resources through the system API. A process must contain a minimum of one thread. Threads inside the same process can share memory, but processes do not share memory without using special API calls.

Before CreateProcess can create the initial thread, the operating system must establish a suitable working environment from the executable image. This requires the operating system to construct several key data structures such as the Windows EPROCESS[38] block, the initial memory address space for the executable image, the kernel process block (KPROCESS),[39] and the program environment block (PEB).[40] Each of the data structures mentioned plays a key role in the execution cycle of a process's threads and, by extension, the process overall. As part of the EPROCESS initialization, the system gives the process a process identifier (PID).

The creation of the address space involves establishing the virtual memory for the new executable image. Once established, the operating system maps the section object containing the executable image to the new virtual memory space at the base address specified in the header of the image. This establishes the new process's memory space. Upon completion of this task, the operating system maps the ntdll.dll DLL into the new virtual memory space.

At this point, if the system-auditing component of Windows tracks the creation of new processes, the operating system generates an entry in the Security event log chronicling the existence of the new process.

Finally, CreateProcess registers the new process object with the operating system, initiating a series of internal functions responsible for the management of the new process. This concludes the initialization and setup of the process object, but additional work remains before the process and its initial thread are ready to execute. Exhibit 1-26 illustrates this step.

1.2.3.4 Context Initialization While the initialization of the process object sets the stage for the initial thread, at this point in the CreateProcess procedure, the function has yet to establish the thread. As a result, the process contains no executable component, merely a container where the executable component can exist. To establish the initial thread, CreateThread passes control to the kernel, which in turn constructs the necessary thread working environment. The kernel creates the initial thread in a suspended state, since at this point the thread contains insufficient resources to operate. These insufficiencies include a missing stack and execution context.

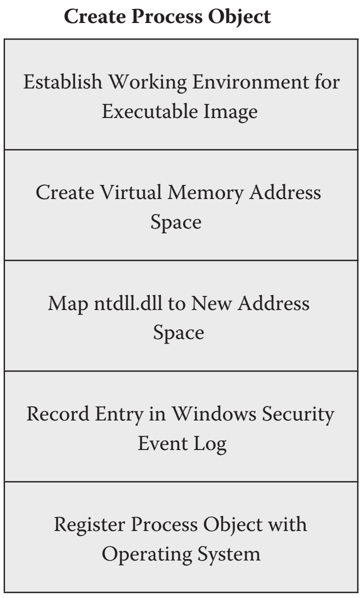

Create Process Object
Establish Working Environment for Executable Image
Create Virtual Memory Address Space
Map ntdll.dll to New Address Space
Record Entry in Windows Security Event Log
Register Process Object with Operating System

Exhibit 1-26 Create process object steps.

The kernel uses the execution context when switching between threads. The execution context stores the current state, or context, of a thread prior to switching to a new thread. Likewise, when the kernel reactivates a thread once it has been given CPU time, the kernel uses the context information to restore the thread's execution at the point that it was last active. With the context and stack established, CreateProcess calls the kernel to insert the thread into the list of threads. This process generates a new Thread ID for the thread and initializes several data structures. The function leaves the thread suspended at this point since the function still must load the remaining subsystems and dependencies required by the image. Exhibit 1-27 visualizes the steps associated with this phase.

1.2.3.5 Windows Subsystem Post Initialization With the majority of the process container and initial thread initialized and ready, CreateProcess must initialize the Windows subsystem. The Windows subsystem is responsible for the interface between the user and the kernel spaces. This subsystem establishes the working environment for applications by providing support for console windows, graphical user interface (GUI) windows, thread and process management services,

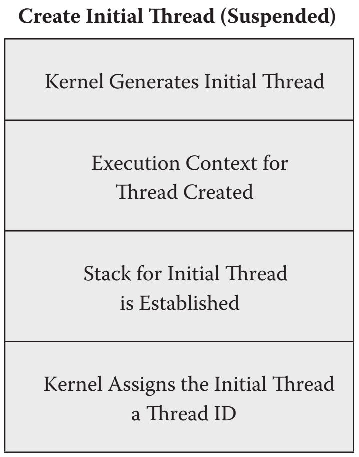

Create Initial Thread (Suspended)
Kernel Generates Initial Thread
Execution Context for Thread Created
Stack for Initial Thread is Established
Kernel Assigns the Initial Thread a Thread ID

Exhibit 1-27 The initial thread creation phase.

and miscellaneous other services. Without a suitable environmental subsystem, an application would be unable to function given the lack of interface between the application and the kernel.

As part of the Windows subsystem post initialization, the operating system checks the validity of the executable to determine if the subsystem should permit the executable image to run. This check involves verifying the executable against group policies established by the administrator, and in the case of Windows Web Server 2008 and Windows HTC Server 2008, the subsystem verifies the imported APIs to ensure the image does not use restricted APIs. CreateProcess instructs the Windows subsystem that a new process (and its thread) is waiting for initialization, requiring the subsystem to perform more low-level initializations. While exploring each of these low-level initialization steps is outside the scope of this article, what is important to understand is how the Windows subsystem handles the introduction of a new process object.

The Windows subsystem (via the Csrss.exe process) receives a copy of the process and its thread's handles along with any necessary flags. CreateProcess gives the Windows subsystem the PID of the process responsible for the call to the CreateProcess function. The Windows subsystem in turn allocates a new process block inside the csrss process and ties in the necessary scheduling priorities, as defined earlier in the CreateProcess procedure, along with a default exception handler. The subsystem stores this information internally and registers the new

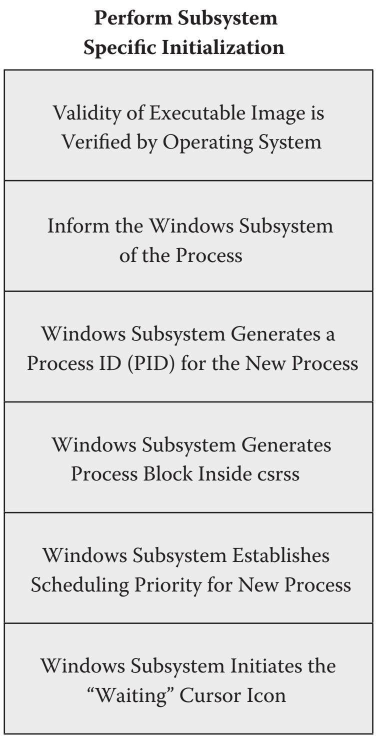

Perform Subsystem Specific Initialization
Validity of Executable Image is Verified by Operating System
Inform the Windows Subsystem of the Process
Windows Subsystem Generates a Process ID (PID) for the New Process
Windows Subsystem Generates Process Block Inside csrss
Windows Subsystem Establishes Scheduling Priority for New Process
Windows Subsystem Initiates the "Waiting" Cursor Icon

Exhibit 1-28 The subsystem initialization phase.

process object within the list of subsystem-wide processes. The process's initial thread is still in the suspended state at this point, but the Windows subsystem activates the application start cursor (the cursor with the small hourglass or the circular icon on Vista or later). This icon will appear for up to two seconds while waiting for the primary thread to engage the GUI of the application. As seen in Exhibit 1-28, the subsystem initialization phase requires more steps than the other phases detailed thus far.

1.2.3.6 Initial Thread … Go! By the end of the subsystem initialization phase, the process has all of the necessary information and access control tokens[41] required to begin execution. Unless the caller of CreateProcess specified the CREATE_SUSPENDED flag set for the process, the operating system begins the initial thread to continue the last step of the initialization process. The initial thread begins by running KiThreadStartup[42] to set up the necessary kernel-level attributes such as the interrupt request level (IRQL).[43] KiThreadStartup in turn calls PspUserThreadStartup, which begins by setting the locale ID and processor type in the thread execution block (TEB) specific to the executable's header.

If the process has a user mode or kernel mode debugger attached, the kernel informs the appropriate debugger of the creation of the new process. If the debugger requests that the kernel kill the thread, the operating system immediately terminates the thread. If the administrator has enabled prefetching on the system, the prefetcher[44] activates. The prefetcher allows the operating system to load a binary faster by using a single data block to reference information from the last time the same binary ran. Coordinating the necessary information into a data structure that the prefetcher can load in a single disk read significantly reduces the time associated with excessive random access disk reads.

The function PspUserThreadStartup initializes the system-wide stack cookie if it has not done so already. This cookie prevents general stack overflow attacks[45] by setting a value near the end of a function's stack frame. Before a function returns, the stack cookie's integrity is verified. If the function cannot verify the integrity of the cookie, the function generates an exception that the binary must address or allows the operating system to terminate the process as a safety precaution. Exhibit 1-29 displays the steps required by this phase.

1.2.3.7 Down to the Final Steps The system initializes the thread local storage (TLS) and fiber local storage (FLS) arrays. The result of this is the possible creation of a preemptive thread as defined in the transport layer security (TLS) configuration.

Once the necessary data structures are established, the system processes the import table of the executable image. This table results in the various required DLLs loading and their entry points being called. For each DLL loaded, the loader passes the entry point function, the DLL_PROCESS_ATTACH flag, to indicate to the DLL that a new process has loaded it. Exhibit 1-30 details this short phase.

The CreateProcess function has now initialized the executable image, registered the necessary data structures with kernel and the Windows subsystem, and loaded the necessary resources to allow the initial thread to execute. With this in mind, the system begins the execution of the initial thread for the new process. After the

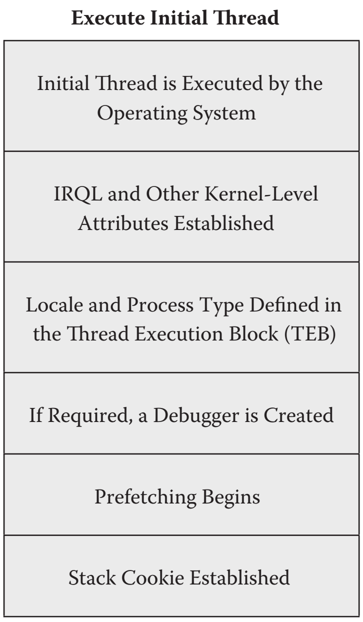

Execute Initial Thread
Initial Thread is Executed by the Operating System
IRQL and Other Kernel-Level Attributes Established
Locale and Process Type Defined in the Thread Execution Block (TEB)
If Required, a Debugger is Created
Prefetching Begins
Stack Cookie Established

Exhibit 1-29 Execution of the initial thread phase.

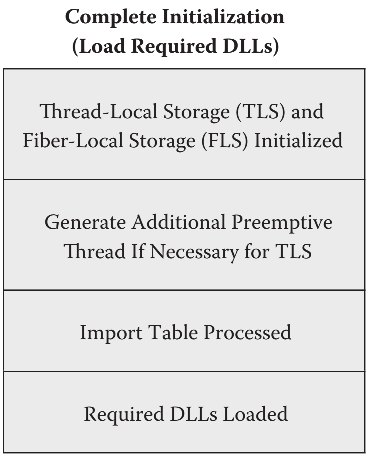

Complete Initialization (Load Required DLLs)
Thread-Local Storage (TLS) and Fiber-Local Storage (FLS) Initialized
Generate Additional Preemptive Thread If Necessary for TLS
Import Table Processed
Required DLLs Loaded

Exhibit 1-30 Completion of the process initialization phase.

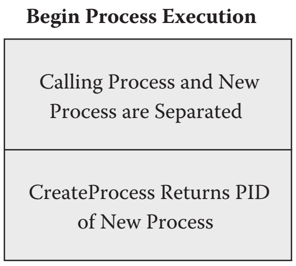

Begin Process Execution
Calling Process and New Process are Separated
CreateProcess Returns PID of New Process

Exhibit 1-31 The process begins.

process's thread begins, the separation between the process that called CreateProcess and the new process is complete, and CreateProcess returns to the caller with the new process's PID. Exhibit 1-31 shows this final phase.

1.2.3.8 Exploiting Windows Execution for Fun and Profit Given the variety of data structures and steps required to generate and execute a process (and its threads), there are undoubtedly areas where malicious actors may exploit these data structures to hide their nefarious activities. One of the most common methods, rootkits, can hide a process by masking out the process object. The function ZwQuerySystemInformation provides information about various system attributes, including the list of running processes. When malicious code hooks this function, it is possible to prevent the caller from seeing all processes, effectively hiding a running malicious executable.

During the DLL loading phase, the operating system queries a registry entry called HKEY_LOCAL_MACHINE\Software\Microsoft\WindowsNT\CurrentVersion\Windows\AppInit_DLLs to determine additional DLLs to load at the start of the new process's initial thread execution. This gives attackers the ability to load their own custom DLLs into every running process on victims' systems without the affected process's binaries explicitly requesting the malicious DLL.

As described above, the process to take an executable program file and turn that file into running code is highly involved. Fortunately, the Windows API masks the majority of the operation behind a subset of API functions. The encapsulation of the low-level details of process creation frees the developer from writing code that could lead to potentially devastating results if done poorly, although accessing functions directly still provides plenty of opportunity to construct malicious executables. While Windows does attempt to hide the majority of the underlying data structures associated with process management, malicious code authors, especially those who develop rootkits, have managed to exploit key aspects of the process management system to hide their processes. Sometimes, having a bit more transparency might help situations like this. At the very least, additional controls should be in place to prevent such tampering from going unnoticed.

1.2.4 The Windows Firewall

With the release of Windows XP SP2, Microsoft Corporation provided users with a built-in firewall in hopes of protecting its users from network threats. Before this release, to protect themselves from network attacks, Windows users had to purchase third-party firewall products such as Zone Alarm or Black Ice. Many users did not understand the need for firewall software, and as a result, the users of the older Windows operating systems were left largely exposed.

With the introduction of the Windows Firewall, also known as the Internet Connection Firewall (ICF), Microsoft institutionalized the use of a limited-functionality firewall to prevent a large number of network attacks. The Windows Firewall is a *stateful firewall*, meaning that the firewall monitors network connections that originate from the user and denies connections that do not originate from the user. By default, this type of firewall denies any incoming network connection that the user did not initiate while allowing user-initiated connection out to the network. Moreover, the Windows Firewall in Vista and Windows Server 2008 has the ability to deny outbound connections on a port-by-port basis. Exhibit 1-32 depicts this behavior.

The Windows Firewall consists of relatively few components:[46] a network driver, a user interface, and a network service. The core of the Windows Firewall is located in the IPNat.sys network driver. This driver is responsible for not only the Windows Firewall but also the network address translation (NAT) functionality of the operating system. The driver registers itself as a "firewall hook driver" to determine if the firewall should allow or disallow a connection (inbound or outbound). The determination of which connections to allow or disallow is derived from the list of approved applications and ports supplied by

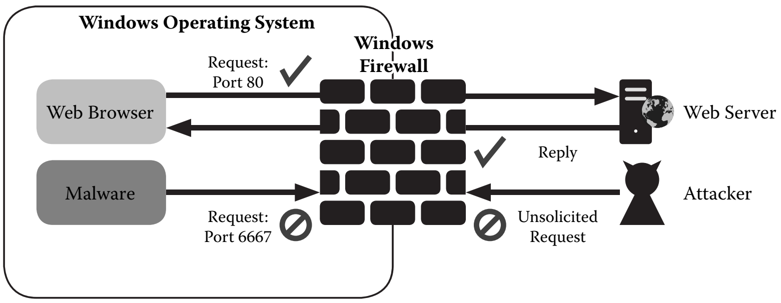


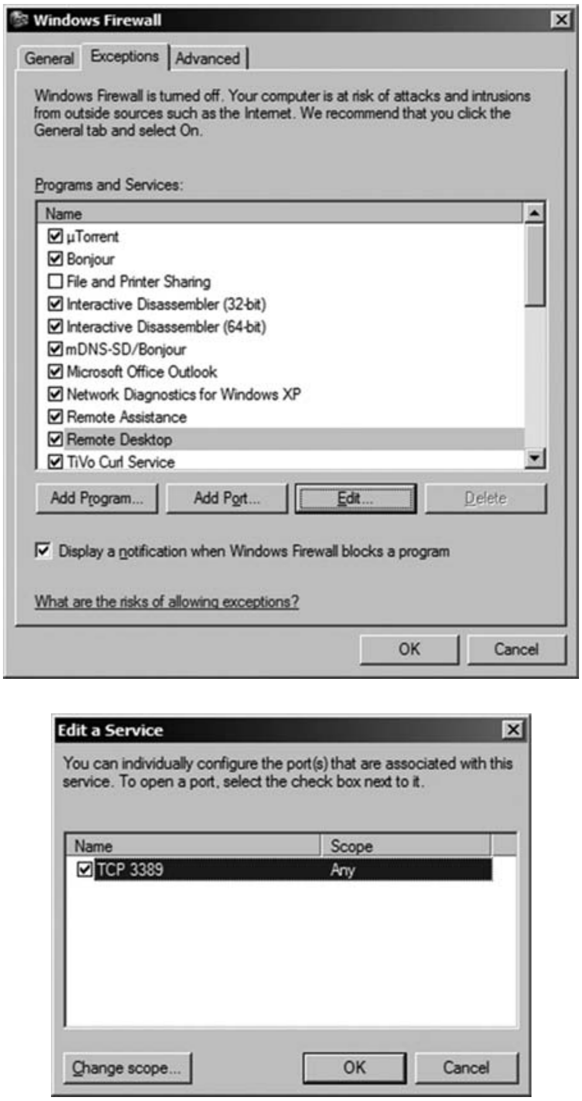
Exhibit 1-32 Windows Firewall's stateful firewall operation.

the user through the user interface component. The user interface, as seen in Exhibit 1-33, allows the user to define which applications and which ports the system allows in and out of the Windows Firewall.

The Internet Connection Firewall and Internet Connection Sharing (ICF/ICS) service handles the translation of the firewall rules from the user interface to the IPNat.sys driver. In the event that the ICF/ICS server shuts down, the firewall functionality of the operating system is disabled. This is one way that malicious code can circumvent the restrictions imposed by the Windows Firewall.

The overall system involved in the Windows Firewall appears, on the surface, to be a rather simplistic three-component system and, from a malicious code author's perspective, there are numerous ways to get around the firewall. As mentioned previously, simply disabling the ICF/ICS service is the fastest way to evade the firewall. The side effect of this action is the immediate security popup, seen in Exhibit 1-34, indicating that the firewall is now disabled. This brute force approach to disabling the firewall can be immediately apparent; however, malicious code authors can suppress the security alert using the Windows application programming interface (API). Regardless of the suppression of the alert, the fact remains that the malicious code has clearly disabled the service, giving the victim warning that something has compromised the system.

A more subtle approach adopted by many malicious code families involves adding the offending malicious code to the list of approved applications. As seen previously in Exhibit 1-33, the Windows Firewall retains a list of applications that the user has approved for network access. When the user first engages the Windows Firewall, only a very small number of entries are present in the list. In recent Windows



Windows Firewall is turned off. Your computer is at risk of attacks and intrusions from outside sources such as the Internet. We recommend that you click the General tab and select On.

Programs and Services:

Name
☑ µTorrent
☑ Bonjour
☐ File and Printer Sharing
☑ Interactive Disassembler (32-bit)
☑ Interactive Disassembler (64-bit)
☑ mDNS-SD/Bonjour
☑ Microsoft Office Outlook
☑ Network Diagnostics for Windows XP
☑ Remote Assistance
☑ Remote Desktop
☑ TiVo Curl Service

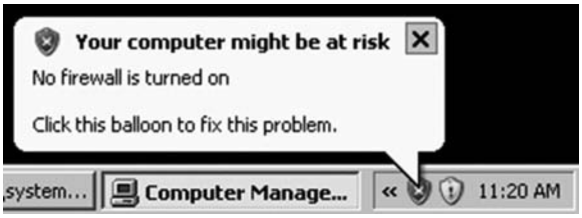
☑ Display a notification when Windows Firewall blocks a program

What are the risks of allowing exceptions?

You can individually configure the port(s) that are associated with this service. To open a port, select the check box next to it.

Name	Scope
☑ TCP 3389	Any

Exhibit 1-33 Windows Firewall user interface.


Your computer might be at risk

No firewall is turned on

Click this balloon to fix this problem.

Exhibit 1-34 Security alert after disabling the Internet Connection Firewall and Internet Connection Sharing (ICF/ICS) service.

versions, such as Vista and Server 2008, the user must explicitly enable common applications, such as Internet Explorer, for the application to have access to the network. For the malicious code author to add the malicious code to a list of authorized applications, the author must either modify the registry directory or use the Windows API. To add itself to the list of Windows Firewall–authorized applications, the malicious code adds an entry to the following registry branches:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile\AuthorizedApplications\List
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\SharedAccess\Parameters\FirewallPolicy\DomainProfile\AuthorizedApplications\List

These branches contain a list of all of the Windows Firewall–authorized applications in a series of strings that indicate the path to the application along with key attributes. The malicious code adds its information here as if the application is actually allowed to traverse the firewall.

Windows provides an alternative to directly modifying the registry by providing an API interface[47] to modify the Windows Firewall directly. Using a few lines of code, as seen in Exhibit 1-35, malicious code authors can add their creations to the list of Windows Firewall–authorized applications. Using the API may reduce the likelihood of the malicious code raising suspicion about the victim or tripping antivirus solutions that monitor direct registry modifications.

Adding a program to the authorized programs list only allows the malicious code to access the network; it does not allow the malicious code to run as a server application. For the Windows Firewall to allow

```
// Retrieve the authorized application collection.
fwProfile->get_AuthorizedApplications(&fwApps);
// Create an instance of an authorized application.
CoCreateInstance(
        __uuidof(NetFwAuthorizedApplication),
        NULL,
        CLSCTX_INPROC_SERVER,
        __uuidof(INetFwAuthorizedApplication),
        (void**)&fwApp
        );
// Allocate a BSTR for the process image file name.
fwBstrProcessImageFileName = SysAllocString(fwProcessImageFile
Name);
// Set the process image file name.
fwApp->put_ProcessImageFileName(fwBstrProcessImageFileName);
// Allocate a BSTR for the application friendly name.
fwBstrName = SysAllocString(fwName);
// Set the application friendly name.
hr = fwApp->put_Name(fwBstrName);
// Add the application to the collection.
hr = fwApps->Add(fwApp);
```

Exhibit 1-35 Programmatically adding a program to the Windows Firewall authorized programs list.

access to incoming network connections, the malicious code author must "poke a hole" in the firewall. Much the same way that authorized programs are contained within the registry, the firewall retains the list of ports that allow unsolicited network connections in the registry under the branches:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile\GloballyOpenPorts\List
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\SharedAccess\Parameters\FirewallPolicy\DomainProfile\GloballyOpenPorts\List

Windows gives each opened port a set of configuration items defining the network (or networks) allowed to make connections, the protocol of the port (TCP or UDP), and the name of the service running the port. Malicious code authors can insert their own port definitions into the list to open a hole for the malicious code. For malicious code such as Waledac, which can provide HTTP and DNS services on Internet-facing victims, it is important to disable the Windows Firewall to allow the malicious code to provide the necessary services.

```
fwProfile->get_GloballyOpenPorts(&fwOpenPorts);
// Create an instance of an open port.
CoCreateInstance(
        __uuidof(NetFwOpenPort),
        NULL,
        CLSCTX_INPROC_SERVER,
        __uuidof(INetFwOpenPort),
        (void**)&fwOpenPort
        );
// Set the port number.
fwOpenPort->put_Port(portNumber);
// Set the IP protocol.
fwOpenPort->put_Protocol(ipProtocol);
// Allocate a BSTR for the friendly name of the port.
fwBstrName = SysAllocString(name);
// Set the friendly name of the port.
fwOpenPort->put_Name(fwBstrName);
// Opens the port and adds it to the collection.
fwOpenPorts->Add(fwOpenPort);
```

Exhibit 1-36 Programmatically opening a port through the Windows Firewall.

Waledac, however, does not directly modify the Windows registry. Instead, Waledac uses the Windows Firewall API to open firewall ports. Exhibit 1-36 illustrates the Microsoft example for programmatically opening a port in the Windows Firewall.

Antivirus vendors may detect a modification to the firewall settings of Windows Firewall. When malicious code needs to remain stealthy, modifying the Windows Firewall settings to connect to the command-and-control (C&C) server may prove problematic. Many families of malicious code, notably the BBB group B family from 2008,[48] use programs such as Internet Explorer to circumvent the Windows Firewall restrictions.

As mentioned previously, Windows Vista does not give Internet Explorer access to the Internet by default. With the prevalence of Internet Explorer, many attackers assume that victims have given Internet Explorer access to the Internet by adding the application to the Windows Firewall–authorized programs list.

Windows XP systems, on the other hand, automatically allow Internet Explorer network requests to traverse the Windows Firewall, regardless of whether or not the user explicitly lists the browser in the authorized application list. Attackers can exploit this fact by injecting code into a running Internet Explorer instance. The Windows

Firewall does not concern itself with what part of Internet Explorer is requesting access through the firewall, only that the overall application itself is requesting the access. Using the Windows API functions WriteProcessMemory, an attacker can inject code into the Internet Explorer process. Then, using the CreateRemoteThread API, the attacker can activate the code under the context of Internet Explorer, giving the attacker's code the ability to access the Internet without disturbing the Windows Firewall. Typically, when an instance of Internet Explorer is unavailable, malicious code authors create a new instance of Internet Explorer using WinExec or CreateProcess while starting the new instance as a hidden application. This prevents the new instance from appearing on the taskbar and alerting the victim of the presence of the malicious code.

Given the fact that the Windows Firewall is little more than software running on a potentially infected host, malicious code could install additional network drivers to intercept traffic before the Windows Firewall can intercede. Using a new network stack, such as is done by the Srizbi family of malicious code, quickly defeats the Windows Firewall but requires a higher degree of operating system knowledge on the part of the malicious code author.

Given the multiple ways around the Windows Firewall, it is apparent that, by itself, the Windows Firewall is not sufficient protection against malicious code threats. With a variety of malicious code designed to handle the obstacles presented by the Window Firewall, the enterprise cannot rely on the firewall itself as the sure-fire solution. Thus, enterprises should not rely on the Windows Firewall alone, but should consider the system as an additional layer in the overall "defense-in-depth" strategies deployed by most network administrators. The most effective enterprise firewall solution comes when using firewall devices external to the Windows computer the company wishes to protect.

References

1. "National Information Assurance Glossary," June 2006, http://www.cnss.gov/Assets/pdf/cnssi_4009.pdf.
2. "State of the Hack: iDefense Explains … Public Key Encryption," iDefense *Weekly Threat Report*, ID# 490870, July 6, 2009.