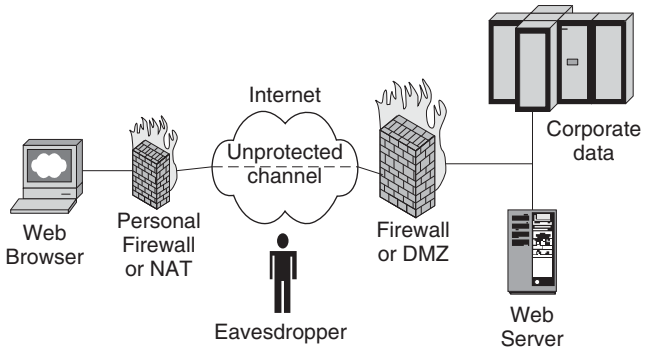
13.2 Secure Channels

Messages passing across any public network—particularly the Internet—can be intercepted. The information contained in such messages is thus potentially available to an eavesdropper. For sensitive communication across a public network, create encrypted SECURE CHANNELS (434) to ensure that data remains confidential in transit.

Example

A typical Internet-based application will exchange a variety of information with its users. Some of this information about people, products and services will be sensitive in nature. Typical examples include credit card numbers when making on-line purchases or bookings, or product plans and shipment schedules exchanged between business partners.



An unprotected channel

It is relatively straightforward to secure data on the client and the server. The client and server can be protected by different firewall mechanisms, in the case of the server maybe even a fully-fledged DEMILITARIZED ZONE (449), to make it difficult for an attacker to penetrate the systems and gain access to the data they hold. If the data is of a particularly sensitive nature, such as credit card numbers, it may even be stored in encrypted form following INFORMATION OBSCURITY (426). However, data on the Internet itself has no protection from intruders, and straightforward encryption mechanisms that can be used in a managed environment cannot be applied between two unrelated machines across the Internet. Because of this, data is passed across an unprotected channel, as shown in the figure on the previous page.

A private channel could be set up between client and server, but this would rely on a private networking mechanism, which defeats the object of delivering services cheaply and conveniently across a public network such as the Internet.

Context

The system delivers functionality and information to clients across the public Internet through one or more Web servers. Larger systems may use multiple Web servers and multiple application servers to deliver this functionality, all protected by a DEMILITARIZED ZONE (449). The application must exchange data with the client. A percentage of this data will be sensitive in nature.

Problem

How do we ensure that data being passed across public or semi-public space is secure in transit?

The solution to this problem must resolve the following forces:

- Much application data is non-sensitive, but data that is sensitive needs protection when it is made available outside the system's defence mechanisms.
- Encrypting data is a significant overhead on system performance.
- It is easier to provide encryption solutions with known partners, but many customers of the system cannot or will not install specific software or hardware for this purpose.

Solution

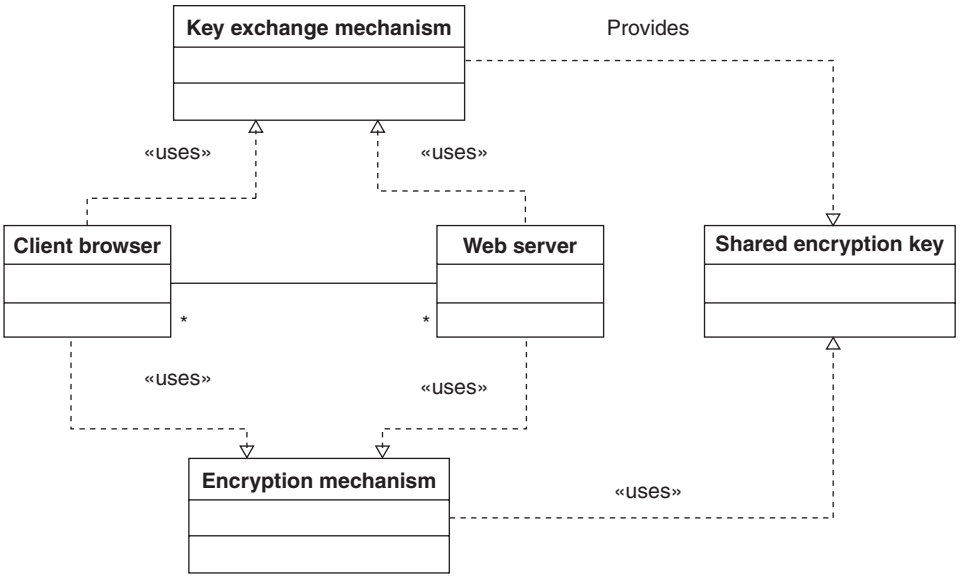
Create secure channels for sensitive data that obscure the data in transit. Exchange information between client and server to allow them to set up encrypted communication between themselves. Reduce the associated overhead on the system by using ordinary communication channels for non-sensitive data.

Structure

SECURE CHANNELS (434) requires the following elements:

- A Web server, which provides access to the application's functionality and information. The Web server software could be one of many common types such as Apache or Internet Information Services, but it must support the encryption and key exchange mechanisms being employed.

- A client browser, the universal client used to access the system. The browser can be any generic browser such as Internet Explorer or Netscape, but it must support the encryption and key exchange mechanisms being employed.
- A shared encryption key. To exchange encrypted data in a secure but efficient manner, the client browser and Web server must share a secret value. This usually takes the form of a symmetrical encryption key that can be used to both encrypt and decrypt data.
- A key exchange mechanism such as an agreed protocol that the client browser and Web server use to exchange the shared encryption key securely.
- An encryption mechanism. The client browser and Web server use an agreed encryption mechanism that is applied to sensitive data. Armed with the shared encryption key and an algorithm to implement this encryption mechanism, both client and server can encrypt and decrypt confidential data.



Composition of elements for SECURE CHANNELS

Dynamics

Most of the time the client and server exchange information over a normal, unencrypted channel. When they wish to exchange sensitive data, they set up a secure channel using encryption. This section focuses on such an encrypted exchange. In a typical encrypted exchange, two parties, Alice and Bob, wish to exchange information. They can use a shared, symmetric encryption key to pass information

privately across a public space, such as the Internet. Eve, the eavesdropper trying to intercept the message, cannot decrypt the message even if she captures it, because she does not possess the shared key. This is the basic privacy mechanism used by Secure Sockets Layer (SSL), the most prevalent secure channel mechanism on the Internet.

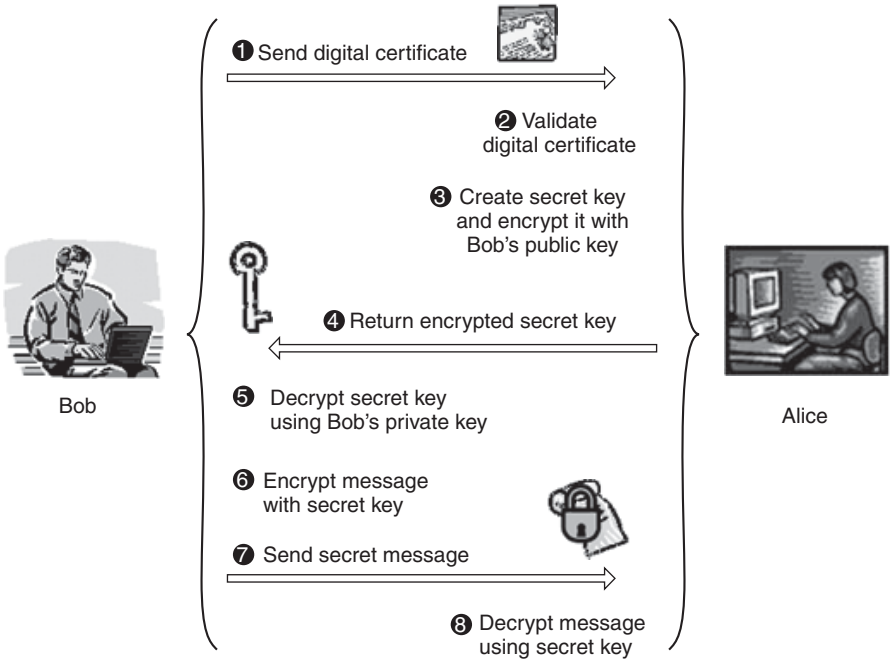
In Internet terms, the roles of Bob and Alice are played by the Web server and the browser or other client software. However, this presents us with a problem: usually the client and server do not know each other well enough to have established a shared key. The shared key cannot just be built into the Web server and browser software, otherwise everyone could decrypt everybody else's messages. Each SSL session uses a unique shared key—hence it is also called the 'session key' in SSL. What we need is a way for the Web server and the client to exchange the session key to be used.

The exchange of session keys is based on the use of a digital certificate. The owner of the Web server must obtain a server certificate that associates a given public key with the server's DNS name, for example `www.securitypatterns.org`. Once the server certificate has been installed on the Web server, a client requests a secure channel by accessing a resource using a URL that starts with 'https:' rather than 'http:' This request causes the server to send the client its digital certificate to prove its identity, and to provide the client with its public key. The client then checks the digital certificate to make sure that it is issued by a trusted third party and that it matches the DNS name with which it is accessing the server.

If the certificate looks valid, the client generates a symmetrical encryption key (the session key) and uses the public key in the certificate to encrypt it. The server uses its private key to decrypt the session key, then starts using the session key to exchange encrypted messages with the client. This exchange has achieved two things: the client now believes that the server is genuine, and the client and server now have a shared, secret key with which to exchange private messages—in this case, the contents of HTTP POST requests. This key exchange can be extended to allow the client to authenticate itself with the server, which is important for KNOWN PARTNERS (442), but is not essential for most SECURE CHANNELS (434) across the Internet. The essence of the key exchange is shown in the figure below, with Bob in the role of the Web server. For a more detailed description of how the whole exchange works, see [And01]. See figure on page 438.

You may at this point be wondering why we do not just use the public/private key pairs to encrypt the data passing back and forth. The answer is that the symmetrical session key and its associated algorithm are respectively shorter and quicker to run than those for public/private key encryption. Most machines do not currently have the necessary resources to encrypt the amount of data passing between a Web client and server in an appropriate time using public key cryptography. This is a trade-off between performance and security.

Because the session key is shorter and its algorithm simpler, ciphertext based on it is easier to crack than ciphertext based on public key cryptography. If the same session key is used all the time between a client and server, it becomes increasingly



Key exchange for SECURE CHANNELS

possible to work out the shared key using statistical analysis of the messages being passed based on the number of times particular words appear in the language in which the messages are written. To counter this, the session key is changed on a regular basis using a mechanism similar to the initial exchange of the session key described earlier.

Implementation

Because much information provided by the system is non-sensitive, it can be distributed by normal Internet mechanisms such as HTTP. When the system need to exchange sensitive data with a user, a SECURE CHANNEL is set up specifically for that exchange. The most common mechanism for creating SECURE CHANNELS (434) across the Internet is the Secure Sockets Layer (SSL). SSL capabilities are built into all major current Web browsers, and also into popular development platforms such as J2EE and .NET. Any application that wishes to use these capabilities merely needs to obtain a server certificate for SSL that can then authenticate the server to the client and can be used as the basis for secure session key exchange.

One issue to consider is that the increased security delivered by SECURE CHANNELS (434) may conflict with other desired non-functional characteristics. One obvious conflict is between the use of SSL and performance. In theory we could use SSL for all exchanges between client and server—in practice, this imposes too great an overhead on the exchange of non-sensitive information. Another less obvious conflict is between SSL and the application of LOAD BALANCED ELEMENTS [Dys04] to the Web servers to improve availability and scalability. When load balancing is combined with SECURE CHANNELS (434) it presents a problem, because if the client were to be routed to a different server than the one that began the SSL session, the new server would not possess the session key for that SSL exchange. One solution here is to 'pin' a particular client to a particular server for the duration of its SSL exchange, a technique that is sometimes termed 'server-affinity.' However, this impacts on the availability and scalability of the solution.

To use load balancing and SECURE CHANNELS (434) in combination, it is best to use load balancing hardware that understands secure channels and that can itself participate in the secure channel on behalf of the server. This solution avoids issues of server-affinity, but does open up a further security gap, as unencrypted information is exchanged between the load balancer and the servers. To address this problem we introduce a totally new set of Web servers and load balancers. Any traffic that enters the outermost switch will either arrive on port 80, the default HTTP port, or port 443, the default HTTPS port. Traffic on port 80 is switched to a standard Web server via the standard load balancer. Traffic on port 443, however, can only go to a secure Web server via a secure load balancer. The packets passed between the secure load balancer and the selected secure Web server are still unencrypted, but we now have the opportunity to put in place additional security measures to harden the channel from secure load balancer to secure Web server. This effectively extends the secure channel from the browser right down to the secure Web server.

The use of SSL between the client and the Web server is fairly standard and is the obvious place to apply SECURE CHANNELS (434): this applies for both B2C e-commerce and B2B[2] e-commerce. However, this is not the only place that such security should be considered. Even if a site has been protected as described in DEMILITARIZED ZONE (449), it may be possible for an attacker to penetrate one of the routers, or even a Web server on the DMZ. From this vantage point they can potentially monitor traffic within the DMZ as it passes between the Web servers and the application servers. If this traffic is not encrypted, it is then available to the eavesdropper. To avoid this possibility, you can set up a virtual private network (VPN) between the Web servers and the application servers. This VPN makes sure that data is encrypted as it passes through the DMZ, the firewall and the internal router.

[2] B2B: business-to-business, B2C: business-to-customer.

Example Resolved

The Internet application uses SSL between browser and Web server to create a SECURE CHANNEL. Such channels are used to protect application data in transmission in different scenarios:

- Passing payment information between client and server
- Viewing of order status by customers
- Logging in by customers
- Changing of details by customers

For a high-availability system, load balancing content switches would be used to process SSL so that the SSL session is between the client browser and the load balancer (as opposed to between the client and the server). Although a VPN using IPSec would provide peer-to-peer security between the Web and application servers, this would be an unnecessary overhead for most applications given the sensitivity of the information passed back and forth.

Variants

Asynchronous Secure Channel. So far this pattern has discussed synchronous secure channels between clients and Web servers. However, there are other ways to implement secure channels. One option is to use an asynchronous messaging system and to encrypt the contents of the messages. Asynchronous operation gives us better performance and availability characteristics, at the expense of the additional processing that is required to correlate messages and to recover from failure.

In terms of Internet technology, we can use MIME-encoded e-mail messages with encrypted payloads as a secure, asynchronous channel. Alternatively, we can use encrypted XML/SOAP messages, as defined in the WS-Security specification [OASIS]. These messages can be delivered synchronously (HTTP), pseudo-asynchronously (one-way HTTP message), or asynchronously (e-mail). While the use of asynchronous messaging is generally useful, you may well need to write more custom code to support this unless you find some good products to help you out.

Known Uses

SECURE CHANNELS (434) is implemented in all mainstream Web browsers (Internet Explorer, Netscape, Mozilla, Opera, Firefox) and Web servers (IIS, Apache) through the provision of SSL functionality. Support for SSL is also included in development platforms such as .NET and J2EE. There are other, less commonly-used variations on SECURE CHANNELS (434), such as IPSec, TLS and various VPN protocols.

Consequences

The following benefits may be expected from applying this pattern:

- Security is improved, because even in the event of an attack that captures data in transit, the data is not usable by the attacker.
- Common implementations of SECURE CHANNELS (434) are built into most Internet software.
- Key exchange allows previously-unknown partners to conduct confidential conversations.
- The mechanism does not impact the exchange of non-sensitive data, because it is only used when sensitive data is to be exchanged.

The following potential liabilities may arise from applying this pattern:

- Performance is impacted by the processing overhead associated with the encryption mechanism.
- Scalability is potentially impacted if the encryption mechanism causes server-affinity, which would undermine effective load balancing.
- Availability is potentially impacted if the encryption mechanism causes server-affinity, which would undermine effective fail-over.
- Cost is increased and maintenance overhead is added, because you must obtain and maintain one or more server certificates for your SECURE CHANNELS (434). Also, you may need to increase the hardware specification of your Web servers or buy dedicated encryption hardware to mitigate the associated performance overhead.