Authentication

“What was your username again?” *clickety clickety*

— The BOFH

User Authentication

Basic system uses passwords

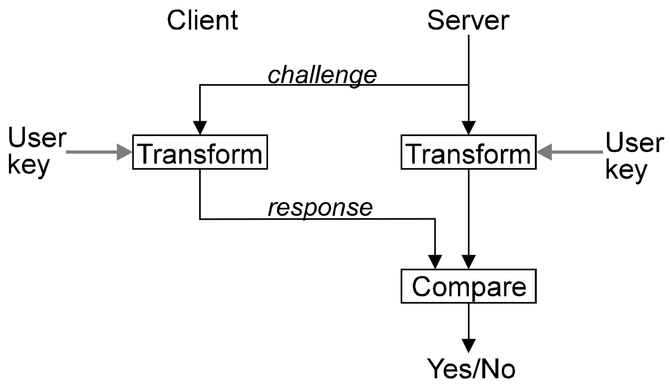
- Can be easily intercepted

Encrypt/hash the password

- Can still intercept the encrypted/hashed form

Modify the encryption/hashing so the encrypted/hashed value changes each time (challenge/response mechanism)

User Authentication (ctd)



Vulnerable to offline password guessing

- Attacker knows the challenge and the encrypted challenge, can try to guess the password used to process it

User Authentication (ctd)

There are many variations of this mechanism but it’s *very* hard to get right

- Replay attacks (replay an old message)
- Impersonation attacks (pretend to be the client or server)
  - In the previous example, only the client is authenticated
  - Could be talking to anyone
  - All too common
- Reflection attacks (bounce the authentication messages elsewhere)
- Steal the client/server authentication database

User Authentication (ctd)

- Oracle attacks (have the client or server transform the data you feed to it)
  - Use the oracle to encrypt/decrypt data
  - Particularly effective with RSA decryption/signing duality
- Subsequent (post-authentication) messages aren't protected
- Modify messages between the client and server
- Chess grandmaster attack
- Mafia attack
- Insert-catchy-name-here attack

User Authentication (ctd)

Example: Needham-Schroeder protocol

- Grandfather of all authentication protocols, introduced in 1978
- New attack discovered via exhaustive state space enumeration in 2002
- A can impersonate self if A and B are the same
  - So what?
- What if A and B are user processes at different privilege levels communicating via RPC?
  - Privilege-separated ssh or Qmail

Needham-Schroeder has been repeatedly proven secure and then broken again over its 25-year history

- Not because it's a bad design, but because it's *very* hard to do

Simple Client/Server Authentication

Client and server share a key K

- Server sends a challenge encrypted with K
  - Challenge should generally include extra identifiable information like the server ID and timestamp
- Client decrypts the challenge, transforms it (e.g. adds one, flips the bits), re-encrypts it with K, and sends it to the server
- Server does the same and compares the two

Properties

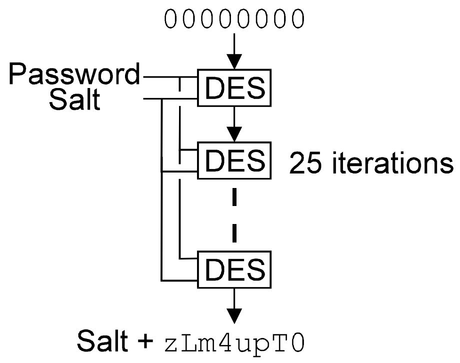
- Both sides are authenticated
- Observer can’t see the (unencrypted) challenge, can’t perform a password-guessing attack
- Requires reversible encryption

(In practice it’s a bit more complex than that)

Unix Password Encryption

Designed to resist mid-70’s level attacks

Uses 25 iterations of modified DES



Salt prevents identical passwords from producing the same output

crypt16

Handles 16 characters instead of 8

- 20 DES crypts for the first 8
- 5 DES crypts for the second 8

Result was weaker than the original crypt

- Search for passwords by suffix
- Suffix search requires only 5 DES crypts

LMHASH

From MS LAN Manager

Like crypt16 but without the salt

- If password < 7 chars, second half is 0xAAD3B435B51404EE
- Result is zero-padded(!) and used as 3 independent DES keys
- 8-byte challenge from server is encrypted once with each key
- 3 × 8-byte value returned to the server

Newer versions added NTHASH (MD4 of data), but the LMHASH is still sent alongside NTHASH data

Subject to rollback attacks (“I can’t handle this, give me LMHASH instead”)

LMHASH (ctd)

L0phtcrack

- Collects hashed passwords via network sniffing, from the registry, or from the SAM file on disk
- Two attack types
  - Dictionary search: 100,000 words in a few minutes on a PPro 200
  - Brute force: All alphabetic characters in 6 hours, all alphanumerics in 62 hours on quad PPro 200
- Parallel attacks on multiple users
- Runs as a background process

Hellman's Time/Memory Tradeoff Attack

Trade off search time for memory, 1980

Step 1: Lookup table creation

$\text{key}_0 \rightarrow \text{keyHash}_0$

$\text{keyHash}_0 \rightarrow \text{key}_1$ — Via reduction function R()

$\text{key}_1 \rightarrow \text{keyHash}_1$

…

$\text{key}_n \rightarrow \text{keyHash}_n$

- Creates a sequence of *n* related entries, where typical *n* = 10,000
- Starting with keyHash_0, we can recreate the entire chain up to keyHash_n

Hellman's Time/Memory Tradeoff (ctd)

Step 2: Store { keyHash_0, keyHash_n } pairs

- Table is sorted by keyHash_n

$$\begin{array}{l|l|}
0 & \text{keyHash}_0, \text{keyHash}_n \\
1 & \text{keyHash}_0, \text{keyHash}_n \\
\dots & \\
m & \text{keyHash}_0, \text{keyHash}_n
\end{array}$$

- Total table size is reduced by the chain length
- $m \times 10{,}000$ table becomes $m \times 2$ table

Hellman's Time/Memory Tradeoff (ctd)

Lookup table use

Step 1: Given kHPW = keyHash(password), generate a chain starting with R(kHPW)

$$\begin{tikzcd}
\text{kHPW}_0 - \dots - \text{kHPW}_x - \dots - \text{kHPW}_n \arrow[d, leftrightarrow] \\
\boxed{\text{keyHash}_0, \text{keyHash}_n}
\end{tikzcd}$$

- Some point of this new chain will correspond to an endpoint of an existing chain

Hellman’s Time/Memory Tradeoff (ctd)

Step 2: Using the stored keyHash$_0$, regenerate the chain

$$\begin{tikzcd}[column sep=tiny] \text{keyHash}_0 - \ldots - & \text{keyHash}_{x\text{-}1} \arrow[d, Rightarrow] & - \text{keyHash}_x \{ - \ldots - \text{keyHash}_n \} \\ & \text{key} & \end{tikzcd}$$

- Knowing the key used to generate the matching keyHash, we now know the password

Hellman’s Time/Memory Tradeoff (ctd)

Problems

- Values in chains can collide, causing the chains to merge

$$\begin{tikzcd}[column sep=small] \ldots - \text{keyHash}a_6 - \text{keyHash}a_7 \arrow[dr, dash] & \\ & \text{keyHash}a_8/b_{17} - \text{keyHash}a_9/b_{18} \\ \ldots - \text{keyHash}b_{15} - \text{keyHash}b_{16} \arrow[ur, dash] & \end{tikzcd}$$

  - Efficiency of a table decreases with size as more collisions occur
- Chains can loop, a variant of the collision problem

Hellman’s Time/Memory Tradeoff (ctd)

Instead of using one big table, use *m* smaller tables with *m* different reduction functions

- Different reduction functions means that chains can collide, but never merge

Lookup becomes more expensive, since we need to generate a chain over *m* tables, not 1

Rivest’s Optimisation

Limit chain size by using distinguished points as endpoints, 1982

- Distinguished point is one for which some simple criterion holds
- Example: Low 8 bits of the value are zero

Loops can be detected

- If no distinguished point is found after *n* iterations, we may be looping

Merges can be detected

- Merging chains will have the same endpoints
- Discard merging chains and replace them with new ones

Rivest’s Optimisation (ctd)

Problem: Chains are now variable-length

- Longer chains have a higher probability of collisions/merges
- Longer chains take more time to check
  - Regeneration of long chains due to false alarms is expensive

Oechslin’s Optimisation

Vary the reduction function R() for each step in the chain, 2003

- c.f. LRW encryption mode

Chains will only collide if the collision appears at the same location in both chains

- Otherwise, both chains will continue with a different reduction function, and won’t merge

Chains don’t loop

Rainbow Tables

Lookup differs from the usual brute-force chain generation method

- Apply $R_{n-1}()$ to kHPW, check for a match
- Apply $R_{n-2}() + R_{n-1}()$ to kHPW, check for a match
- …
- Apply $R_0() + R_1() + \ldots + R_{n-1}()$ to kHPW, check for a match
- On average this requires half as many calculations as the standard methods
  - $n^2/2$ calculations on one table rather than n calculations on m tables

Rainbow table reference implementation available, unlike earlier methods

Rainbow Tables (ctd)

Example: Cracking Windows (LANMAN) passwords

- Seven times faster than the non-rainbow table method
- Using 1.4GB of data, can recover 99.9% of all alphanumeric passwords in 13.6s on a 1.5GHz P4

Reduction function is a mapping from the ciphertext space to a key

- Algorithms by Narayanan and Shmatikov to adapt the rainbow attack to any search space for which there’s an efficient algorithm to compute the i^{th} element, 2005

Use Markov models to generate a dictionary of probably passwords

- Use rainbow tables to search the dictionary space

NT Domain Authentication

One-stop shop for demonstrating authentication protocol weaknesses

Joining

- Client and server exchange challenges CC and SC
- Session key = CC + CS encrypted with the machine password (NTHASH of the LMHASH of the machine name)
- Result is used as an RC4 key

Anyone on the network can intercept this and recover the initial key

NT Domain Authentication (ctd)

User logon

- Client sends RC4-encrypted LMHASH and NTHASH of the user password
- Server decrypts and verifies the LMHASH and NTHASH of the password

RC4 key is reused, can be recovered in the standard manner for a stream cipher

Auxiliary data (logon script, profile, SID) aren’t authenticated

This is why Win2K switched to Kerberos

Attacking Domain Authentication over the Net

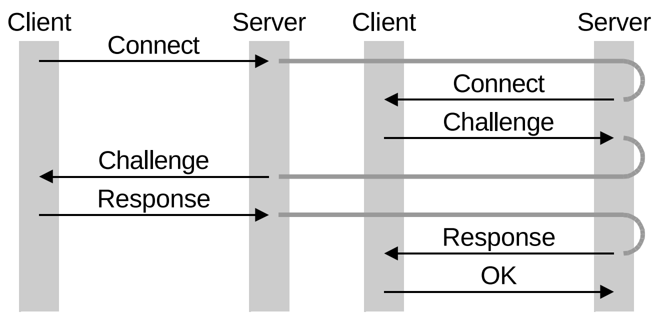
Create a web page with an embedded (auto-loading) image

- Image is held on an SMB Lanman server instead of an HTTP server
- Windows connects to the server
- Server sends a fixed all-zero challenge
- Windows responds with the username and hashed password

All-zero challenge allows the use of a pre-computed dictionary

Attacking Domain Authentication over the Net (ctd)

Reflection attack



Server has now connected to the Windows machine without knowing the password

GSM

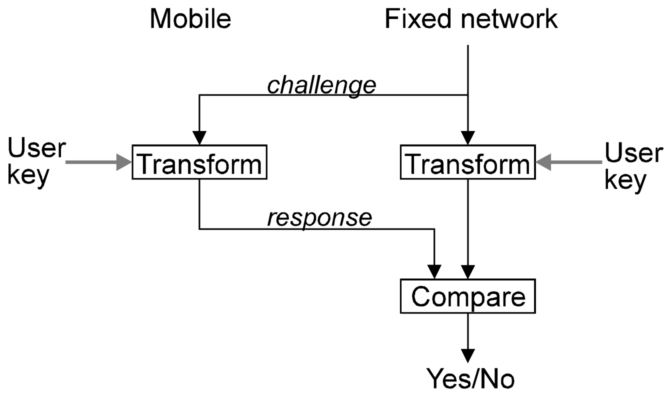
GSM subscriber identity module (SIM) contains

- International Mobile Subscriber Identity (IMSI)
- Subscriber identification key K_i

Used for authentication and encryption via a simple challenge/response protocol

- A3 and A8 algorithms provide authentication (usually combined as COMP128)
- A5 provides encryption

GSM (ctd)



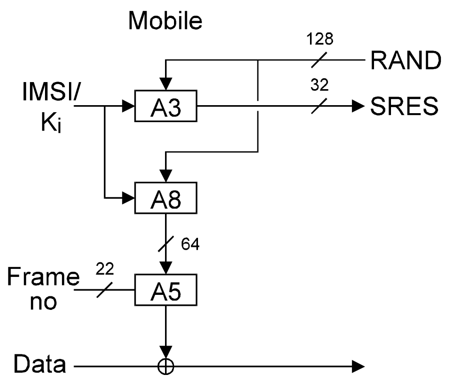
Authentication is a simple challenge/response using A3 and IMSI/K_i

- c.f. basic (broken) challenge/response from earlier

GSM Security

A3 is used to generate the response

A8 is used to generate the A5 key



GSM Security (ctd)

1. Base station transmits 128-bit challenge RAND
2. Mobile unit returns 32-bit signed response SRES via A3, which usually employs the COMP128 algorithm
3. RAND and K_i are combined via A8 to give a 64-bit A5 key
4. 114-bit frames are encrypted using the key and the frame number as input to A5

GSM Security (ctd)

GSM security was broken in April 1998

- COMP128 is weak, allows the IMSI and K_i to be extracted
  - – Direct access to secret SIM data (cellphone cloning)
  - – Performed via over-the-air queries to the phone
- Some cards were later modified to limit the number of COMP128 queries
- A5 was deliberately weakened by zeroing 10 key bits
  - – Even where providers don’t use COMP128, all shorten the key
- Claimed GSM fraud detection system didn’t seem to exist
- Affected 80 million GSM phones

GSM Security (ctd)

Key weakening was confirmed by logs from GSM base stations

```
BSSMAP GSM 08.08 Rev 3.9.2  (BSSM) HaNDover REQuest (HOREQ)
-------0 Discrimination bit D BSSMAP
0000000- Filler
00101011 Message Length    43
00010000 Message Type      0x10
Channel Type
00001011 IE Name           Channel type
00000011 IE Length         3
00000001 Speech/Data Indicator    Speech
00001000 Channel Rate/Type Full rate TCH channel Bm
00000001 Speech encoding algorithm      GSM speech algorithm
Encryption Information
00001010 IE Name           Encryption information
00001001 IE Length         9
00000010 Algorithm ID      GSM user data encryption V.1
******** Encryption  Key   C9 7F 45 7E 29 8E 08 00
Classmark Information Type 2
```

GSM Security (ctd)

Many countries were sold a weakened A5 called A5/2

- A5 security: Breakable in real time with 2^{40} precomputations
- A5/2 security: None (5 clock cycles to break)
- Another attack is to bypass GSM entirely and attack the base station or land lines/microwave links

GSM security was compromised at every level

- Deliberately weakened key generation
- Broken authentiction
  - GSM MoU knew of this nearly a decade ago but didn't inform its members
- A5/1 was known to be weak, A5/2 was deliberately designed to be weak

GSM security was described by one security researcher as "multiple-redundant compromise"

GSM Security (ctd)

Most other cellphone security systems have been broken too

- Secret design process with no public scrutiny or external review
- Fear of government interference ensured poor security

S/Key

One of a class of software tokens/one-time-password (OTP) systems

Freely available for many OS’s, `http://www.yak.net/skey/`

Uses a one-way hash function to create one-time passwords

- pass1 = hash(hash(hash(password)))
- pass2 = hash(hash(password))
- pass3 = hash(password)
  - Actual hash includes a server-specific salt to tie it to a server

S/Key (ctd)

Each hash value is used only once

- Server stores value *n*, verifies that hash(*n*-1) = *n*
- Can tie password *n* + 1 to password *n*
- Can’t derive password *n* from password *n* + 1

Knowing hash(hash(password)) doesn’t reveal hash(password)

Values are transmitted as 16 hex digits or 6-word phrases

Later refinements added new algorithms, more rigorous definitions of the protocol

OPIE

One-time Passwords in Everything, developed by (US) Naval Research Laboratory

Freely available for many OS's, `ftp://ftp.nrl.navy.mil/pub/security/opie/`

Enhancement of S/Key with a name change to avoid trademark problems

TANs

Per-transaction PINs

- User has a list of TANs, one per transaction

  ~~`UZDCOQwG`~~

  ~~`XeSnvszE`~~

  `0uZVSJ2U`
- Bank sends new TANs when the old list is about to expire
  - TANs are sent out with bank statements
  - Marginal cost is close to zero

Remarkably effective against online credit card theft/fraud

- The one thing you can't do online is intercept paper mail

PPP PAP/CHAP

Simplest PPP authentication is PAP, password authentication protocol

- Plaintext user name + password
- Response is Ack or Nak

Challenge handshake protocol (CHAP) was created to fix PAP

- Standard (weak) challenge/response protocol using a hash of the challenge and a shared secret

Other PAP Variants

SPAP

- Shiva {Proprietary|Password} Authentication Protocol
- PAP with a few added bells and whistles

ARAP

- Appletalk Remote Access Protocol
- Bidirectional challenge/response using DES
  - Authenticates client to server, server to client

MS-CHAP

- Microsoft CHAP
- DES-encrypts 8-byte challenge using LMHASH/NTHASH
- Server stores the hash rather than the plaintext password
- Subject to the usual LMHASH attacks

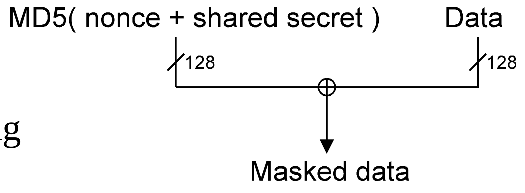
RADIUS

Remote authentication for dial-in user service

Provides an authentication server for one or more clients (dial-in hosts)

- Later extended to handle all manner of other authentication tasks (like COBOL, it just keeps going and going)

Client communicates with the server via encrypted (masked) communications using a shared secret key



- Subsequent blocks are masked with MD5(prev.block + shared secret)

RADIUS (ctd)

RADIUS protocol:

- Client forwards the user access request to a RADIUS server
- Server replies with
  - Reject access
  - Allow access (based on a user-supplied password)
  - Challenge (for a challenge-response protocol, e.g. CHAP)
- If challenge-response is used, client forwards the challenge to the user, user sends their response to the client, which forwards it to the server

One RADIUS server may consult another (acting as a client)

DIAMETER

RADIUS with the second-system effect

- Extensions handled in a more rigorous manner via attribute-value pairs (AVPs)
- Extremely flexible authentication, accounting, and authorisation (AAA) solution
- Adds message integrity and message flow integrity protection via HMAC and sequence numbers

TACACS/XTACACS/TACACS+

Based on an obscure ARPANET access control system for terminal servers, later documented and extended by Cisco

- Forwards the username and password to a TACACS server, returns an authorisation response

XTACACS, Extended TACACS

- Adds support for multiple TACACS servers, logging, extended authorisation
- Can independently authorise access via PPP, SLIP, telnet

TACACS/XTACACS/TACACS+ (ctd)

TACACS+

- Separation of authentication, authorisation, and accounting functions with extended functionality
- Password information is encrypted using RADIUS-style encryption
- Password forwarding allows use of one password for multiple protocols (PAP, CHAP, telnet)
- Extensive accounting support (connect time, location, duration, protocol, bytes sent and received, connect status updates, etc)
- Control over user attributes (assigned IP address(es), connection timeout, etc)

Sorting out the *xxxxx*S's

RADIUS, TACACS = Combined authentication and authorisation process

DIAMETER = Authentication, authorisation, and accounting (AAA), and then some

XTACACS = AAA separated

TACACS+ = XTACACS with extra attribute control and accounting

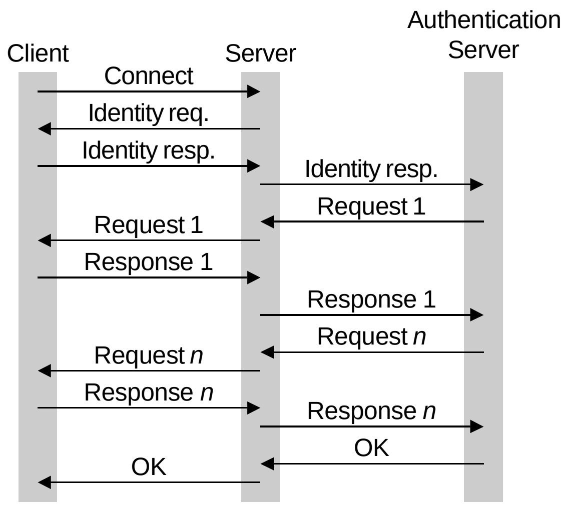
Common shortcomings

- No integrity protection (except DIAMETER)
- No replay protection (except DIAMETER)
- Packets leak data due to fixed/known formatting

General Usage Schema for the *xxxxx*S's

Server consults *xxxxx*S back-end system for authentication information

- Server acts as a proxy for authentication processing
- A general-purpose mechanism to handle this would be useful…



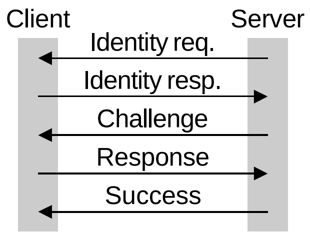
Extensible Authentication Protocol (EAP)

General-purpose authentication protocol framework

- Originally designed for PPP
- Later adopted for many other uses, e.g. 802.x authentication

RADIUS-like exchange

- Server sends requests
  - Client identity
  - MD5 challenge (CHAP-like exchange)
  - One-time password
- Client sends responses
- Server responds to each one with Success or Failure



EAP Extensions

Intended as a universal network-internal authentication protocol

- Requires extra protection when used over insecure channels
- Typical protection mechanism is EAP tunnelled over TLS (EAP-TTLS)

One variant uses TLS for client and server authentication (EAP-TLS)

- EAP-TLS != EAP-TTLS
- Requires client-side certificates, little-used

EAP Extensions (ctd)

Cisco created Lightweight EAP (LEAP) for 802.11 use

- Defeats the point of an *extensible* protocol
  - How much more lightweight does it need to be anyway?
- Use of LEAP conveniently ties you into using Cisco gear
- Insecure, relies on (weak) MS-CHAP
  - LEAP-breaking software (anwrap, THC-LEAPcracker, asleap) is openly available

Followup was Protected EAP (PEAP) from Cisco, Microsoft, and RSADSI

- Like EAP-TTLS only different
- Microsoft PEAP != Cisco PEAP
  - *EAP decisions were driven by politics rather than security

Kerberos

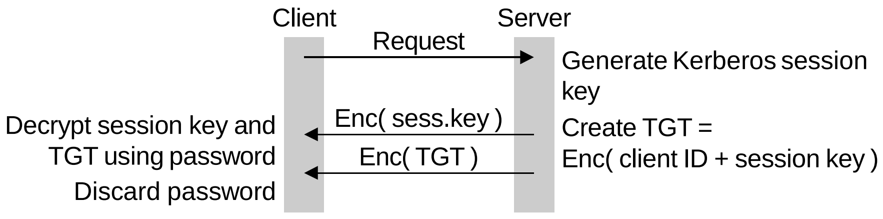
Designed at MIT based on Needham-Schroeder protocol

Relies on a key distribution centre (KDC) to perform mediated authentication

KDC shares a key with each client and server

Kerberos Sign-on

To avoid long-term password storage, the user's password is converted into a short-term client key via the KDC


Client
Server
Request
Generate Kerberos session key
Enc(sess.key)
Create TGT =
Enc(client ID + session key)
Decrypt session key and TGT using password
Enc(TGT)
Discard password

- KDC sends a short-term client key encrypted with the user's password to the client
- User decrypts the short-term client key and discards the password
- Future KDC ↔ client communications use the short-term client key

Kerberos Sign-on (ctd)

KDC also sends out a ticket-granting ticket (TGT)

- TGT contains the client ID, short-term client key, and an expiry time encrypted with the KDC key
- TGT is used in all further requests
- Based on a theoretical Kerberos model that separates the authentication server and ticket-granting server
  - KDC/AS issues the TGT to talk to the KDC/TGS
- Separation is rarely used in practice

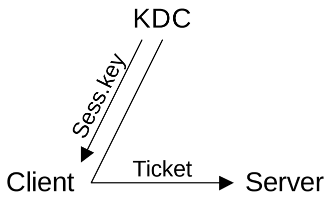
This process converts the user password to a limited-duration equivalent TGT

- Kerberos v5 includes pre-authentication for the request (encrypted timestamp) to avoid offline password-guessing attacks

Kerberos Use

When a client wants to connect to a server

- Client sends TGT and authenticator (encrypted timestamp) to the KDC
- KDC sends to the client
  - Session key encrypted with the client's key
  - Ticket = client ID + session key encrypted with the server's key
- User forwards the ticket to the server



Kerberos Use (ctd)

- User decrypts the session key, server decrypts the ticket to recover the client ID and session key
  - Only the client can recover the client-encrypted session key
  - Only the server can recover the server-encrypted session key

Ticket identifies the client and client's network address (to stop it from being used elsewhere)

Mutual Authentication

Parties exchange session-key encrypted timestamps

- Only holders of the shared session key can get the encryption right
- Replays detected by checking the timestamp on each exchange
- Messages older than a certain time are automatically rejected

KDC's can be replicated to increase availability

- Kerberos is designed so that most database accesses are read-only, making replication easier

Kerberos Realms

Problems with a single KDC database

- Compromising a KDC can compromise many users
- Big KDC databases are difficult to manage
- Big databases lead to name clashes

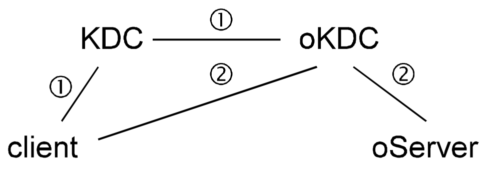
Solution is to use multiple realms

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

Kerberos Realms (ctd)

When a client connects to a server in another realm

- KDC authenticates the client to the other realm’s KDC
- Other realm’s KDC authenticates the client to the other realm’s server



Multi-KDC chaining is disallowed for security reasons (a rogue KDC in the chain could allow anyone in)

Kerberos V5

Extends V4 in various ways

- Extended ticket lifetimes (V4 max = 21 hours)
- Allows delegation of rights
- Allows hierarchical realms
- Adds algorithms other than DES
- V4 used ad hoc encoding, V5 uses ASN.1

Win2K uses Kerberos V5 with Windows-specific extensions

Ticket Lifetimes

V4 allowed a maximum 21 hour lifetime, V5 allows the specification of

- Start time
- End time
- Renewal time (long-term tickets must be renewed periodically)

This added flexibility allows for features like postdated tickets

Delegation

Request a TGT for different machine(s) and/or times

- TGT can be set up to allow forwarding (converted for use by a third party) and proxying (use on a machine other than the one in the TGT)

Delegation is a security hole, Kerberos makes it optional

- TGT's are marked as forwarded or proxied to allow them to be rejected

Realms

V4 required a KDC to be registered in every realm

V5 (tries to) fix the problem with rogue KDC chaining by including in the ticket all transited realms

(client →) foo.com → bar.com (→ server)

vs.

(client →) foo.com → hacker.com → bar.com (→ server)

Realms (ctd)

Requires trusting KDC's

- Trust chain only goes back one level
- Current KDC can alter the ID of previous KDC's

Kerberos abdicates responsibility for trust in chaining to application developers (who will invariably get it wrong)

- When chaining, use the shortest path possible to limit the number of KDC's in the path that must be trusted

Other Changes in V5

V4 used DES, V5 added MD4 and MD5

V4 allowed offline password-guessing attacks on TGT's

- Request a TGT for the victim
- Try passwords on the TGT

V5 adds pre-authentication data to the TGT request

Kerberos-like Systems

KryptoKnight (IBM)

- Closer to V4 than V5
- Can use random challenges instead of synchronised clocks
- Either party can contact the KDC
  - In Kerberos, only the initiator can do this
- Encryption is CDMF (40-bit DES)
  - Required by former US export controls

Kerberos-like Systems (ctd)

SESAME (EU)

- European Kerberos clone mostly done by ICL, Bull, and Siemens
  - Uses XOR instead of DES
  - XOR in CBC mode cancels out 50% of the “encryption”
  - Keys are generated from the current system time
  - Only the first 8 bytes of data are authenticated
- Apparently users were expected to find all the holes and plug in their own secure code
- Later versions added public-key encryption support
- Vendor-specific versions provided enhanced security services

Kerberos-like Systems (ctd)

DCE (OSF)

- Distributed Computing Environment uses Kerberos V5 as a security component
- DCE adds privilege and registration servers to the Kerberos KDC
  - Privilege server provides a universal unique user ID and group ID (Kerberos uses system-specific names and IDs)
  - Registration server provides the database for the KDC and privilege server
- DCE security is based on ACLs (access control lists) for users and groups
- Data exchanges are protected via native DCE RPC security

Authentication Tokens

Physical device used to authenticate the device owner to the server

Two main types

- Challenge-response calculators
- One-way authentication data generators
  - Non-challenge-response nature fits the “enter name and password” authentication model

Authentication Tokens (ctd)

SecurID

- Uses a clock synchronised with a server
- Token encrypts the time, sends it to the server in place of a password
  - 64-bit key (seed), 64-bit time, produces a 6-8 digit output (cardcode)
    - Key can be recovered in 2^{40} operations (Contini and Yin)
  - Card can be protected by a 4-8 digit PIN which is added to the cardcode
- Server does the same and compares the result
- Timestamp provides automatic replay protection, but needs to be compensated for clock drift
- Several attempts to replace it with better technology, but like COBOL it just keeps on going

Authentication Tokens (ctd)

Challenge-response calculators

- Encrypt a challenge from the server and return the result to the server
- Server does the same and compares the result
- Encryption is usually DES (ANSI X9.9)
- Encryption key is random (rather than a fixed password) which makes offline password guessing much harder

Authentication Tokens (ctd)

Possible attacks

- Wait for all but the last character to be entered, then sieze the link
- Hijack the session after the user is authenticated

However, any of these are still vastly more secure than a straight password

- It’s very difficult to choose an easy-to-guess password when it’s generated by crypto hardware
- It’s very difficult to leak a password when it’s sealed inside a hardware token

Authentication Tokens (ctd)

Why aren’t they more widely used?

- Anyone with the capability to make them universal has no interest in doing so; anyone with the interest in doing so has no capability to do so
- Visa, Mastercard, etc, don’t want to have a token that anyone else can use
- Microsoft, Verisign, Liberty Alliance, etc, want a token that anyone can use, as long as they’re the middlemen
- See also: Multi-application smart cards, e-cash

Result: Many, many individual incompatible tokens

ANSI X9.26

Sign-on authentication standard ("Financial institution sign-on authentication for wholesale financial transmission")

DES-based challenge-response protocol

- Server sends challenge (TVP = time variant parameter)
- Client responds with encrypted authentication information (PAI = personal authentication information) XORd with TVP

Offline attacks are prevented through the use of the secret PAI information

Variants include

- Two-way authentication of client and server
- Authentication using per-user or per-node keys but no PAI

Public-key-based Authentication

Simple PKC-based challenge/response protocol

- Server sends a challenge
- Client signs the challenge and returns it
- Server verifies the client's signature on the challenge

Vulnerable to oracle attacks

- Server can have the client sign anything
- Algorithm-specific attacks (e.g. RSA signature/encryption duality)

FIPS 196

Entity authentication using public key cryptography

- Extends and clarifies the ISO 9798 entity authentication standard

Signed challenge/response protocol:

- Server sends server nonce SN
- Client generates client nonce CN
- Client signs SN and CN and returns it to the server
- Server verifies the signature on the data

FIPS 196 (ctd)

Mutual authentication uses a three-pass protocol

- Server sends the client signed SN as the final step

Inclusion of the CN prevents the previous oracle attacks

- Vulnerable to other attacks unless special precautions are taken

Netware Authentication

Netware 3 used a challenge-response method

- Server sends challenge
- Client responds with MD4(MD4(serverID/salt, password), challenge)
- Server stores (server-dependant) inner hash
  - Not vulnerable to server compromise because of the serverID/salt

Netware 4 added public-key encryption managed via the Netware Directory Services (NDS)

Netware Authentication (ctd)

User’s public/private keys are stored by NDS, accessed with a modification of the V3 protocol

- Server sends challenge
- Client responds with server-public-key encrypted V3 hash and session key
- Server returns user’s RSA key encrypted with the session key

Client performs a Kerberos-like conversion to a short-term authentication key

- RSA key is converted to a Gillou-Quisquater (GQ) key
- RSA key is deleted
- GQ key is used for authentication

Netware Authentication (ctd)

Compromise of the GQ key doesn't compromise the RSA key

GQ is much faster than RSA for both key generation and authentication

GQ is used to authenticate the user

- Authenticate the GQ key using the long-term RSA key
- Sign requests with the short-term GQ key

All valuable information is deleted as quickly as possible

- Only the short-term GQ key remains active

Netware Authentication (ctd)

Protocol had a design/implementation flaw in the initial V3 authentication step used to obtain the private key

- Server stored the hash of the password used to authenticate the user
- Since this was used to obtain the private key, it was equivalent to storing the private key in plaintext

Biometrics

Capture data via cameras or microphones

Verification is relatively easy, identification is very hard

Fingerprints

- Small and inexpensive
- ~10% of people ("goats") are difficult or impossible to scan
- Associated with criminal identification

Voice authentication

- Upset by background noise, illness, stress, intoxication
- Can be used over phone lines

Eye scans

- Intrusive (scan blood vessels in retina/patterns in iris)

Biometrics (ctd)

Facial scans

- High error rates

See my biometric security tutorial for more information

Biometrics (ctd)

Advantages

- Everyone carries their ID on them
- Very hard to forge
- Easy to use

Disadvantages

- Easy to defeat
  - In multiple tests, fingerprint readers showed a 100% failure rate
- Unreliable
  - Facial scanners were so unreliable that users disabled them
- You can't change your password
- Expensive

PAM

Pluggable Authentication Modules

- OSF-designed interface for authentication/identification plugins

Administrator can configure authentication for each application

Service	Module
`login`	`pam_unix.so`
`ftp`	`pam_skey.so`
`telnet`	`pam_smartcard.so`
`su`	`pam_securid.so`

PAM (ctd)

Modules are accessed using a standardised interface

```
pam_start( &handle );
pam_authenticate( handle );
pam_end( handle );
```

Modules can be stacked to provide single sign-on

- User is authenticated by multiple modules at sign-on
  - Avoids the need to manually invoke each sign-on service (`kinit`, `dce_login`, `dtlogin`, etc)
- Password mapping allows a single master password to encrypt per-module passwords

PAM in Practice

A typical implementation is Linux-PAM

- Extended standard login (checks time, source of login, etc)
- `.rhosts`, `/etc/shells`
- cracklib (for password checking)
- DES challenge/response
- Kerberos
- S/Key, OPIE
- RADIUS
- SecurID