Authentication

"What was your username again?" *clickety clickety*

— The BOFH

User Authentication

Basic system uses passwords

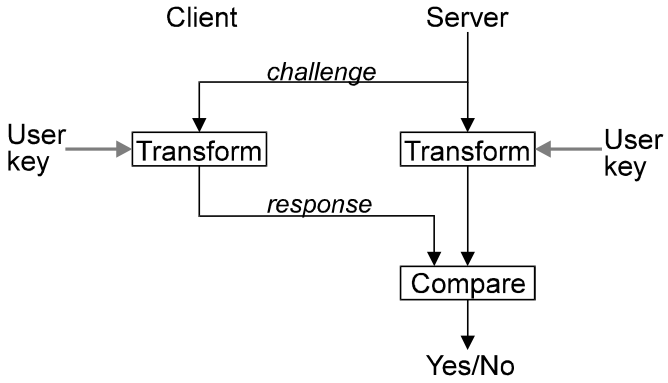
- Can be easily intercepted

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 challenge and 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

- Impersonation attacks (pretend to be client or server)
- Reflection attacks (bounce the authentication messages elsewhere)
- Steal client/server authentication database
- Modify messages between client and server
- Chess grandmaster attack

Simple Client/Server Authentication

Client and server share a key K

- Server sends challenge encrypted with K
  - Challenge should generally include extra information like the server ID and timestamp
- Client decrypts challenge, transforms it (eg 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

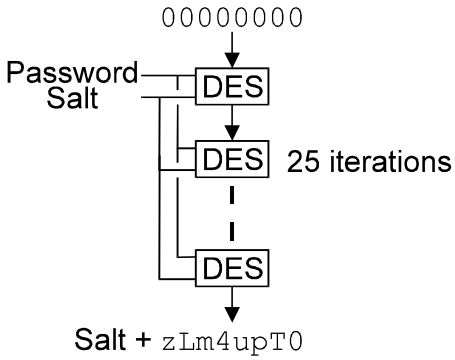
- Both sides are authenticated
- Observer can't see the (unencrypted) challenge, can't perform a password-guessing attack
- Requires reversible encryption

Something similar is used by protocols like Kerberos

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 independant DES keys
- 8-byte challenge from server is encrypted once with each key
- 3×8-byte value returned to server

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

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

LMHASH (ctd)

l0phtcrack, `http://www.l0pht.com/l0phtcrack/`

- Collects hashed passwords via network sniffing, from the registry, or from 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

NT Domain Authentication

Joining

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

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

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

This is why NT 5 will use 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
- NT connects to server
- Server sends fixed all-zero challenge
- NT responds with the username and hashed password

All-zero challenge allows the use of a precomputed dictionary

Attacking Domain Authentication over the Net (ctd)

Reflection attack

- NT connects to the server as before
- Server connects back to NT
- NT sends challenge to server, server bounces it back to NT
- NT responds with the username and hash, server bounces it back to NT

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

Netware Authentication

Netware 3 used 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 serverID/salt

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

Netware Authentication (ctd)

Users 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 users RSA key encrypted with the session key

Once the user has their RSA key pair on the workstation, they can use it for further authentication

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

Netware Authentication (ctd)

Compromise of 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

- Sign request with short-term GQ key
- Authenticate GQ key with long-term RSA key (from NDS)

All valuable information is deleted as quickly as possible

- Only the short-term GQ key remains active

Kerberos

Designed at MIT based on late-70's work by Needham and Schroeder

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

KDC shares a key with each client and server

Kerberos (ctd)

When a client connects to a server

- KDC sends to client
  - Session key encrypted with clients key
  - Session key + client ID encrypted with servers key
- User forwards the latter (a ticket) to the server
- User decrypts session key, server decrypts ticket to recover 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

Kerberos (ctd)

Ticket identifies the client and clients network address (to stop it being used elsewhere)

To avoid long-term password storage, the users password is converted to a short-term client key via the KDC

- KDC sends a short-term client key encrypted with the users password to the client
- User decrypts the short-term client key, decrypts the password
- Future KDC $\leftrightarrow$ client communications use the short-term client key

Kerberos (ctd)

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

- TGT contains the client short-term key encrypted with the KDC key
- Based on a theoretical Kerberos model which separates the authentication server and ticket-granting server
  - KDC/AS issues the TGT to talk to the KDC/TGS

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

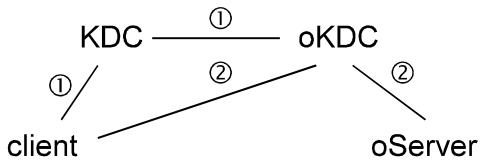
Solution is to use multiple realms

Interrealm 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 client to other realm's KDC
- Other realm's KDC authenticates client to other realm's server



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

Kerberos V5

Extended V4 in various ways

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

Ticket Lifetimes

V4 allowed maximum 21 hour lifetime, V5 allows specification of

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

This added flexibility by providing 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 other 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

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

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

When chaining, use the shortest path possible to limit the number of KDC's in the path which 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)

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 ID’s)
  - Registration server provides the database for the KDC and privilege server
- DCE security is based on ACL’s (access control lists) for users and groups
- Data exchanges are protected via native DCE RPC security

Authentication Tokens

Physical device used to authenticate owner to 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 clock synchronised with server
- Token encrypts the time, sent to server in place of password
  - 64-bit key (seed), 64-bit time, produces 6-8 digit output (cardcode)
  - Card can be protected by 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
- Proprietary ACE server protocol will be replaced by RSA-based SecurSight in early ‘99

Authentication Tokens (ctd)

Challenge-response calculators

- Encrypt a challenge from the server and return result to server
- Server does the same and compares the result
- Encryption is usually DES
- 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

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

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

Each hash value is used only once

- Server stores value *n*, verifies that hash(*n*-1) = *n*

S/Key (ctd)

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 name change to avoid trademark problems

PPP PAP/CHAP

Simplest PPP authentication is PAP, password authentication protocol

- Plaintext user name + password

Challenge handshake protocol was created to fix PAP

- Standard challenge/response protocol using a hash of challenge and shared secret

Other PAP Variants

SPAP

- Shiva {Propietary|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

MSCHAP

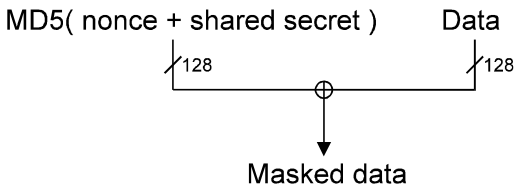
- Microsoft CHAP
- DES-encrypts 8-byte challenge using LMHASH/NTHASH
- Server stores the hash rather than the plaintext password
- Subject to the usual LMHASH attacks

RADIUS

Remote authentication for dial-in user service

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

Client communicates with RADIUS server via encrypted communications using a shared secret key



RADIUS (ctd)

RADIUS protocol:

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

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

TACACS/XTACACS/TACACS+

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

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

XTACACS, Extended TACACS

- Adds support for multiple TACACS servers, logging, extended authorisation
- Can independantly 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 addresse(s), connection timeout, etc)

Sorting out the *xxxxx*S's

RADIUS, TACACS = Combined authentication and authorisation process

XTACACS = Authentication, authorisation, and accounting separated

TACACS+ = XTACACS with extra attribute control and accounting

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) XOR'd with TVP

Offline attacks prevented using secret PAI

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 challenge
- Client signs challenge and returns it
- Server verifies clients signature on the challenge

Vulnerable to chosen-protocol attacks

- Server can have client sign anything
- Algorithm-specific attacks (eg RSA signature/encryption duality)

FIPS 196

Entity authentication using public key cryptography

Extends and clarifies 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 to server
- Server verifies signature on the data

FIPS 196 (ctd)

Mutual authentication uses a three-pass protocol

- Server sends client signed SC as final step

Inclusion of CN prevents the previous chosen-protocol attacks

- Vulnerable to other attacks unless special precautions are taken

Biometrics

Capture data via cameras or microphones

Verification is relatively easy, identification is very hard

Fingerprints

- Small and inexpensive
- ~10% of people 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)

Advantages

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

Disadvantages

- You can’t change your password
- Expensive
- No real standards (half a dozen conflicting ones as well as vendor-specific formats)
- User acceptance problems

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