User Authentication

<*Reference*: chapter 3 of the textbook>

- Password
- Token
- Biometric Authentication
- Remote Users

1

RFC 2828

RFC 2828 defines user authentication as:

"The process of verifying an identity claimed by or for a system entity."

2

Authentication Process

- fundamental building block and primary line of defense
- basis for access control and user accountability

- identification step
  - presenting an identifier to the security system
- verification step
  - presenting or generating authentication information that corroborates the binding between the entity and the identifier

CS3750 (Instructor: Dr. Zhu) 3

3

User Authentication

Four means of authenticating user identity are based on:

something the individual knows	something the individual possesses (token)	something the individual is (static biometrics)	something the individual does (dynamic biometrics)
• password, PIN, answers to prearranged questions	• smartcard, electronic keycard, physical key	• fingerprint, retina, face	• voice pattern, handwriting, typing rhythm

CS3750 (Instructor: Dr. Zhu) 4

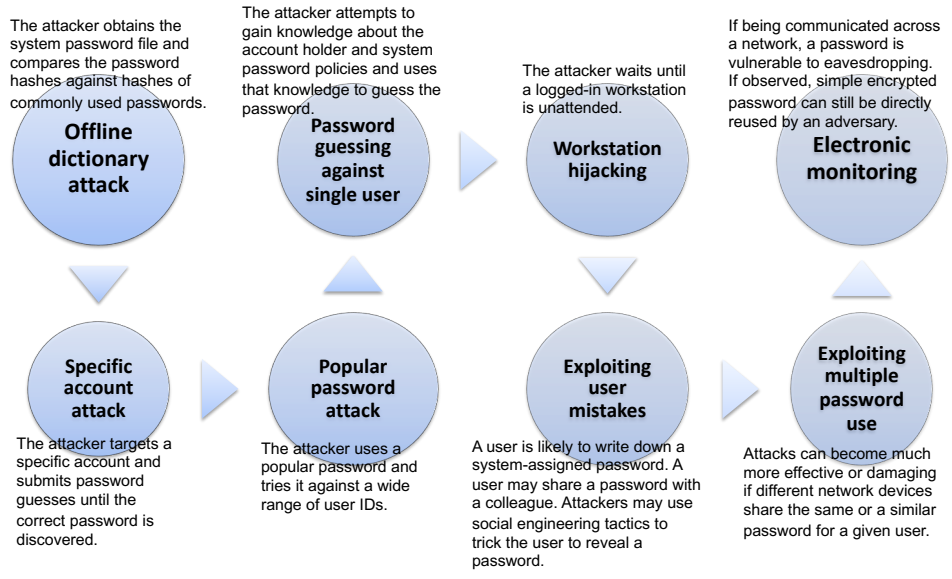
4

Password Authentication

- widely used line of defense against intruders
  - user provides name/login and password
  - system compares password with the one stored for that specified login
- the user ID:
  - determines that the user is authorized to access the system
  - determines the user's privileges
  - is used in discretionary access control
    - E.g., by listing the IDs of the other users, a user may grant permission to them to read files owned by that user.

5

Password Vulnerabilities



6

Hashed Passwords

- A widely used ***password security*** technique: hashed password and salt value
  - virtually all UNIX variants as well as a number of other operating systems
  - has been shown to be secure against a variety of cryptanalytic attacks
- Procedure:
  1. the user selects or is assigned a ***password***.
  2. **Combine** this ***password*** with a fixed-length ***salt value*** **(older: related to** the time at which the password is assigned to the user; **newer:** a pseudorandom or random number).
  3. Serve *password* and *salt* as inputs to a *slow* hashing algorithm to produce a fixed-length ***hash code***. (be *slow* to execute to prevent/delay attacks.)
  4. Store the ***hash code*** (a.k.a. ***hashed password***), a *plaintext copy* of the ***salt***, in the **password file** for a given ***user ID***.
- When a user attempts to log on to a UNIX system,
  1. the user provides an ID and a password
  2. The operating system uses the ID to index into the password file and retrieve the ***plaintext salt*** and the ***hashed password***.
  3. Use the *salt* and *user-supplied password* to the slow hash routine. If the result matches the stored value, the password is accepted.

7

Use of Hashed Passwords

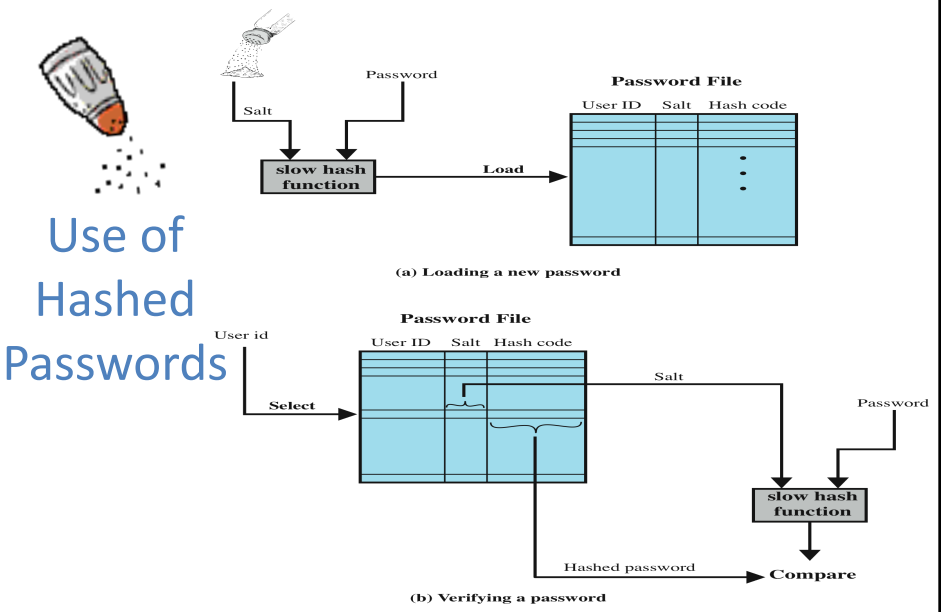


Figure 3.1 UNIX Password Scheme

8

Why Salt?

1. It prevents duplicate passwords from being visible in the password file.
2. It greatly increases the difficulty of offline dictionary attacks. For a salt of length b bits, the number of possible passwords is increased by a factor of 2^b.
   - An offline dictionary attack: the attacker obtains a copy of the password file.
   - The attacker's goal is to guess a single password.
   - Without using the salt, the attacker submits a large number of likely passwords to the hashing function. If *any* of the guesses matches one of the hashes in the file, then the attacker has found a password that is in the file.
   - Using the UNIX scheme, the attacker must take each guess and submit it to the hash function *once* for *each* salt value in the password file, multiplying the number of guesses that must be checked.
3. It becomes nearly impossible to find out whether a person with passwords on two or more systems has used the same password on all of them.

9

UNIX Implementation: Crypt(3)

- Traditional DES-based scheme since the 7th edition of UNIX (1979, Bell Laboratories)

1. Each user selects a password of up to **eight** printable characters in length.
2. Convert it into a **56-bit** value (using 7-bit ASCII), to be used as the **key** to DES encryption
3. The one-way hash routine, known as **crypt(3)**, is based on a *modified* DES encryption algorithm
   - **12-bit** salt, chosen by the system, is incorporated in the modified DES to perturb the encryption algorithm
   - The modified DES algorithm is executed to encrypt a **64-bit** block of ZERO
     - The 64-bit data output of the 1st DES encryption then serves as the 64-bit data input for a 2nd DES encryption. This repeated for a total of 25 modified DES encryptions.
   - The final **64-bit** output is then translated into an **11-character** sequence (Radix-64 encoding)
   - **crypt(3)** routine is designed to discourage guessing attacks.
     - Software implementations of DES are slow compared to hardware versions,
     - the use of 25 DES encryptions in each round of crypt(3) makes it *slow* hash routine.

- now regarded as inadequate
  - still often required for compatibility with existing account management software or multivendor environments

10

crypt(3) functions in C on Linux

- crypt(3): crypt() or crypt_r()

```
#define _XOPEN_SOURCE
#include <unistd.h>
char *crypt(const char *key, const char *salt);

#define _GNU_SOURCE
#include <crypt.h>
char *crypt_r(const char *key, const char *salt, struct crypt_data *data);
```


11

Improvements

1. much stronger hash/salt schemes available for Unix
2. recommended hash function is based on MD5, i.e., md5crypt()
   - salt of up to 48-bits
   - password length is unlimited
   - produces 128-bit hash (22 characters in Radix-64 encoding)
   - uses an inner loop with 1000 iterations to achieve slowdown, much slower than crypt(3)
3. OpenBSD uses Blowfish block cipher-based hash algorithm called bcrypt()
   - most secure version of Unix hash/salt scheme
   - password length is up to 55 characters
   - use 128-bit salt and has a digest size of 184 bits
   - create 184-bit hash value (31 characters in Radix-64 encoding)
   - Include a cost variable: cost variable ↑ → time to perform a Bcrypt hash routine ↑
   - An example of a bcrypt hash string (e.g., https://bcrypt-generator.com/)

```
$2y$10$mSEmB2Mbnwmj6XcgFFfXg.viYzti66p3ZmiFnSzTjsJzPYMPz9XUu
\__/\__/\_____________________/\_____________________________/
 Alg  Cost          Salt                        Hash
```

$2y$: The hash algorithm identifier (bcrypt), which also could be $2a$, or $2b$, or 2 (originally Bcrypt spec)
10: Cost factor (2^{10} ==> 1,024 rounds)
mSEmB2Mbnwmj6XcgFFfXg.: 16-byte (128-bit) salt (Radix-64 encoded to 22 characters)
viYzti66p3ZmiFnSzTjsJzPYMPz9XUu: the resulting 184-bit hash value (Radix-64 encoded to 31 characters)

12

Password Cracking (1)

- dictionary attacks
  - develop a large dictionary of possible passwords and try each against the password file
  - each password must be hashed using each salt value and then compared to stored hash values
- rainbow table attacks
  - Generate a large dictionary of possible passwords.
  - pre-compute tables of hash values for all salts
  - a mammoth table of hash values
    - For example, [OECH03] showed that using 1.4 GB of data, he could crack 99.9% of all alphanumeric Windows password hashes in 13.8 seconds.
  - can be countered by using a sufficiently large salt value and a sufficiently large hash length
    - Both the FreeBSD and OpenBSD approaches should be secure from this attack for the foreseeable future

13

Password Cracking (2)

- Password crackers
  - Exploit the fact that people choose easily guessable passwords
  - Guessable passwords: own name, street name, a dictionary word, etc.
  - Shorter password lengths are also easier to crack
- John the Ripper
  - Open-source password cracker first developed in 1996
  - Uses a combination of brute-force and dictionary techniques

14

Modern Approaches

- **Complex password policy**
  - Forcing users to pick stronger passwords
- **However password-cracking techniques have also improved**
  - The processing capacity available for password cracking has increased dramatically
  - The use of sophisticated algorithms to generate potential passwords
  - Studying examples and structures of actual passwords in use

15

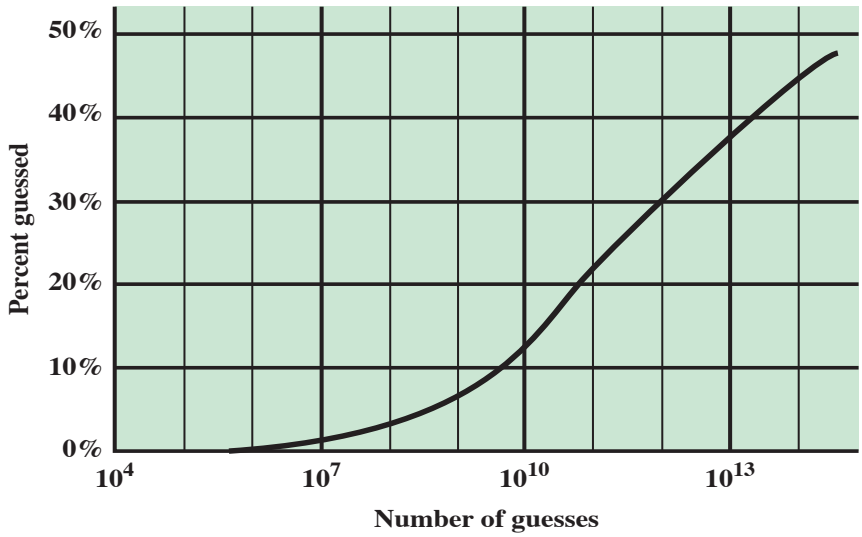


Figure 3.3 The Percentage of Passwords Guessed After a Given Number of Guesses

16

16

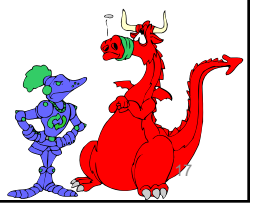
Password File Access Control

Can block offline guessing attacks by denying access to hashed passwords

- make available only to privileged users
- shadow password file

vulnerabilities

- weakness in the OS that allows access to the file
- accident with permissions making it readable
- users with same password on other systems
- access from backup media
- sniff passwords in network traffic



17

Password Selection Techniques

- user education
  - users can be told the importance of using hard to guess passwords and can be provided with guidelines for selecting strong passwords
- computer generated passwords
  - users have trouble remembering them
- reactive password checking
  - system periodically runs its own password cracker to find guessable passwords
  - openware John the Ripper password cracker (openwall.com/john)
- Complex password policy
  - user is allowed to select their own password, however the system proactively checks to see if the password is allowable, and if not, rejects it
  - goal is to eliminate guessable passwords while allowing the user to select a password that is memorable

18

Proactive Password Checking

- **1st approach: password cracker**
  - compile a large dictionary of passwords not to use
  - Two problems: space for storing large dictionary, time of searching and checking likely permutations
- **2nd approach: rule enforcement**
  - specific rules that passwords must adhere to, e.g., length, containing one each of uppercase, lowercase, numeric digits, punctuation marks, etc.
  - Coupled with advice to the user
  - Not sufficient to prevent password crackers since it even alerts crackers as to which passwords not to try
  - Proactive password checker: openware pam_passwdqc (openwall.com/passwdqc/)
- **3rd approach: Bloom filter -** a technique for developing an effective and efficient proactive password checker.

19

Bloom Filter

- A Bloom filter of order k consists of a set of k independent hash functions H1(x), H2(x), … , Hk(x), where each function maps a password in the **large dictionary of passwords not to use** into a hash value in the range 0 to N – 1. That is,

  $H_i(X_j) = y \qquad 1 <= i <= k;\ 1 <= j <= D;\ 0 <= y <= N - 1$

where X_j = jth word in password dictionary; D = # of words in password dictionary; $N = 2^n$ if hash values are all n-bit long; then

1. A **hash table** of *N* bits is defined, with all bits initially set to 0.
2. For each password, its k hash values are calculated, and the corresponding bits in the hash table are set to 1. E.g., if H1(X5) = 35, the bit #35 in the hash table is set.
3. If the bit already has the value 1, it remains at 1.
4. When a new password is checked, its k hash values are calculated.
   - If all the corresponding bits of the hash table are 1, then the password is rejected.
   - All passwords in the dictionary will be rejected.
   - " false positives": passwords not in the dictionary but producing a match in hash table

- e.g., if k = 2, H1(undertaker)=25, H1(hulkhogan)=83, H1(xG%#jj98)=665, H2(undertaker)=998 H2(hulkhogan)=665, H2(xG%#jj98)=998
  - xG%# jj98 will be rejected even though it is not in the dictionary because hT[25], hT[83], **hT[665]**, **hT[998]** all have been set as **1**

20

Performance of Bloom Filter

- the **probability of a false positive**, ***p***, can be approximated by

$$p \approx (1 - e^{-kD/N})^k = (1 - e^{-k/R})^k$$

or equivalently,

$$R = -k / \ln(1 - p^{1/k})$$

where **k**=number of hash functions,

N=number of bits in hash table,

D=number of words in dictionary,

R=N/D, ratio of hash table size (in **bits**) to dictionary size (in **passwords**)

21

Performance of Bloom Filter (2)

- Figure 3.2 on next page plots P as a function of R for various values of k.
- An example
  - a dictionary of 1 million words (D=1,000,000), 8 bytes per word by average
  - wish to have a 0.01 probability of rejecting a password not in the dictionary (P=0.01)
  - choose six hash functions (k=6)
  - the required ratio from the formula is 9.62, calculated as below

  $R = - k/(\ln(1 - p^{(1/k)}) = - 6 / (\ln(1 - 0.01^{1/6}) = 9.62$

  N (in bits) = R * D (in passwords) = 9.62 * 1000000 bits = 9.62×10^6 bits = 1.20×10^6 Bytes = 1.14 MB

  Dictionary size = 1000000 x 8 Bytes = 7.63 MB
- Benefits of using the Bloom filter.

1. Thus, need a **hash table** of 9.62×10^6 bits = 1.14 MB of storage.
   - Storage of the entire **dictionary** = 8,000,000 Bytes = 7.63 MB.
   - A compression factor of 6.69, resulting from 7.63MB / 1.14 MB
2. Furthermore, password checking involves the straightforward calculation of six hash functions and is independent of the size of the dictionary,
   - with the use of the full dictionary, there is substantial searching.

22

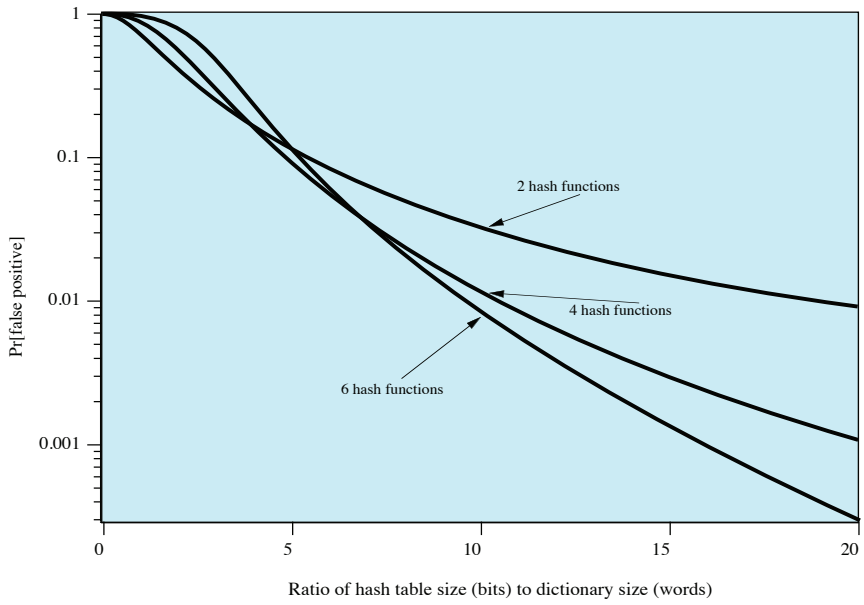


Figure 3.4 Performance of Bloom Filter

23

23

Table 3.2 Types of Cards Used as Tokens

Card Type	Defining Feature	Example
Embossed	Raised characters only, on front	Old credit card
Magnetic stripe	Magnetic bar on back, characters on front	Bank card
Memory	Electronic memory inside	Prepaid phone card
Smart Contact Contactless	Electronic memory and processor inside Electrical contacts exposed on surface Radio antenna embedded inside	Biometric ID card

* Please read the textbook for more details on smart tokens and biometric authentication.

CS3750 (Instructor: Dr. Zhu) 24

24

Memory Cards

- Can store but do not process data
- The most common is the magnetic stripe card
- Can include an internal electronic memory
- Can be used alone for physical access
  - Hotel room
  - ATM
- Provides significantly greater security when combined with a password or PIN
- Drawbacks of memory cards include:
  - Requires a special reader
  - Loss of token
  - User dissatisfaction

25

Smart Tokens

- Physical characteristics:
  - Include an embedded microprocessor
  - A smart token that looks like a bank card
  - Can look like calculators, keys, small portable objects
- Interface:
  - Manual interfaces include a keypad and display for interaction
  - Electronic interfaces communicate with a compatible reader/writer
- Authentication protocol:
  - Classified into three categories:
    - ✓ Static
    - ✓ Dynamic password generator
    - ✓ Challenge-response

26

Smart Cards

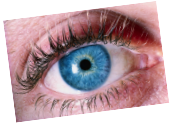
- Most important category of smart token
  - Has the appearance of a credit card
  - Has an electronic interface
  - May use any of the smart token protocols
- Contain:
  - An entire microprocessor: processor, memory, I/O ports
- Typically include three types of memory:
  - Read-only memory (ROM)
    - ✓ Stores data that does not change during the card's life
  - Electrically erasable programmable ROM (EEPROM)
    - ✓ Holds application data and programs
  - Random access memory (RAM)
    - ✓ Holds temporary data generated when applications are executed

CS3750 (Instructor: Dr. Zhu) 27

27

Biometric Authentication

- attempts to authenticate an individual based on unique physical characteristics
- based on pattern recognition
- is technically complex and expensive when compared to passwords and tokens
- physical characteristics used include:
  - facial characteristics
  - fingerprints
  - hand geometry
  - retinal pattern
  - iris
  - signature
  - voice



CS3750 (Instructor: Dr. Zhu) 28

28

Remote User Authentication

- **Authentication over a network, the Internet, or a communication link is more complex**
- **Additional security threats such as:**
  - eavesdropping, capturing a password, replaying an authentication sequence that has been observed
- **Countermeasure:** generally rely on some form of a challenge-response protocol to counter threats

29

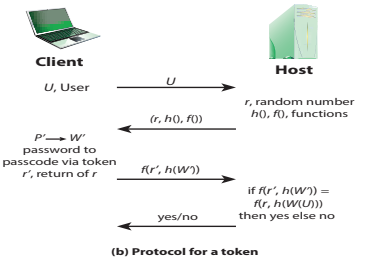
Challenge-Response Protocol for a **password**

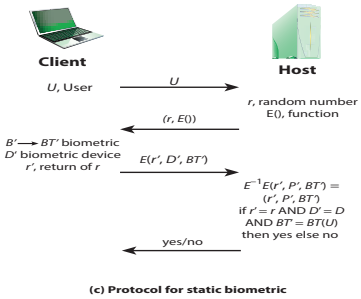
Client	Transmission	Host
U, user	$U \rightarrow$	
	$\leftarrow \{r, h(), f()\}$	random number h(), f(), functions
P' password r', return of r	$f(r', h(P')) \rightarrow$	
	$\leftarrow$ yes/no	if $f(r', h(P')) = f(r, h(P(U)))$ then yes else no

1. user transmits identity to remote host
2. host generates a random number (nonce)
3. nonce is returned to the user
4. host stores a hash code of the password
5. function in which the password hash is one of the arguments
6. use of a random number helps defend against an adversary capturing the user's transmission

30

Challenge-Response Protocol for authentication via a **token, static biometric, or dynamic biometric**





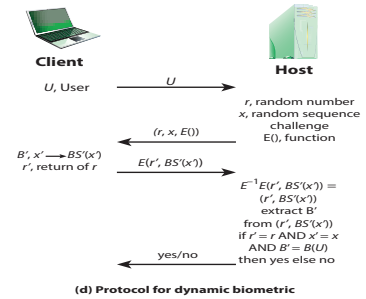
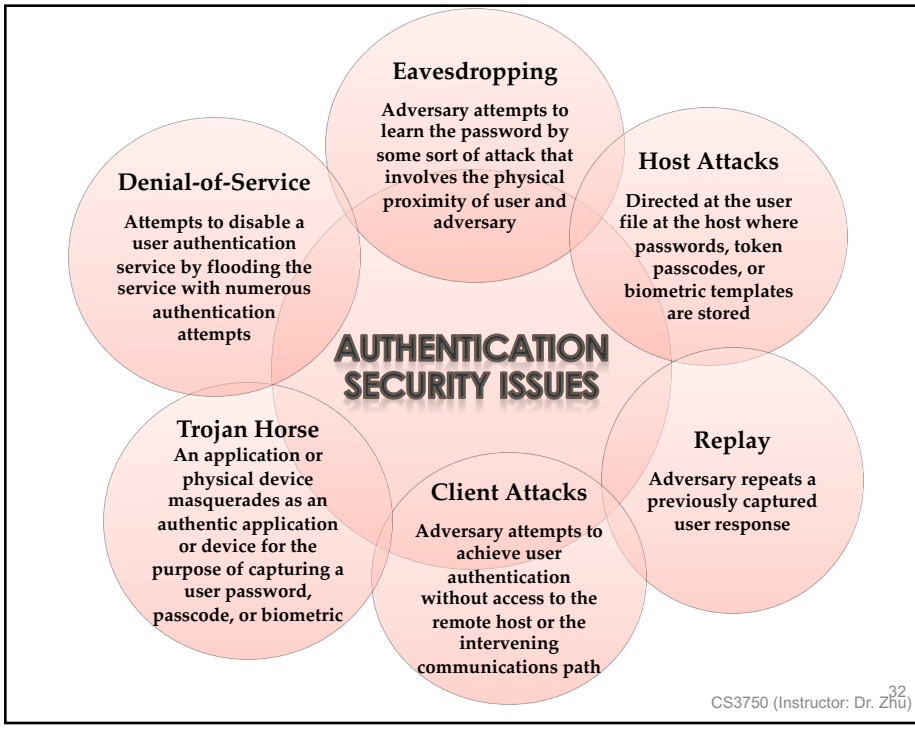


Figure 3.12 Basic Challenge-Response Protocols for Remote User Authentication

31

31



32

Attacks	Authenticators	Examples	Typical defenses
Client attack	Password	Guessing, exhaustive search	Large entropy; limited attempts
	Token	Exhaustive search	Large entropy; limited attempts, theft of object requires presence
	Biometric	False match	Large entropy; limited attempts
Host attack	Password	Plaintext theft, dictionary/exhaustive search	Hashing; large entropy; protection of password database
	Token	Passcode theft	Same as password; 1-time passcode
	Biometric	Template theft	Capture device authentication; challenge response
Eavesdropping, theft, and copying	Password	"Shoulder surfing"	User diligence to keep secret; administrator diligence to quickly revoke compromised passwords; multifactor authentication
	Token	Theft, counterfeiting hardware	Multifactor authentication; tamper resistant/evident token
	Biometric	Copying (spoofing) biometric	Copy detection at capture device and capture device authentication
Replay	Password	Replay stolen password response	Challenge-response protocol
	Token	Replay stolen passcode response	Challenge-response protocol; 1-time passcode
	Biometric	Replay stolen biometric template response	Copy detection at capture device and capture device authentication via challenge-response protocol
Trojan horse	Password, token, biometric	Installation of rogue client or capture device	Authentication of client or capture device within trusted security perimeter
Denial of service	Password, token, biometric	Lockout by multiple failed authentications	Multifactor with token

Table 3.4

Some Potential Attacks, Susceptible Authenticators, and Typical Defenses