79 Slides

Authentication

Computer Security Terminology

Source: From RFC 2828, Internet Security Glossary, May 2000

- Adversary
  - An entity that attacks or is a threat to a system
- Security Policy
  - A set of rules and practices that specify how a system or organization provides security services to protect sensitive and critical system resources.
- Vulnerability
  - A flaw or weakness in a system’s design, implementation, or operation that could be exploited to violate the system’s security policy.

Computer Security Terminology

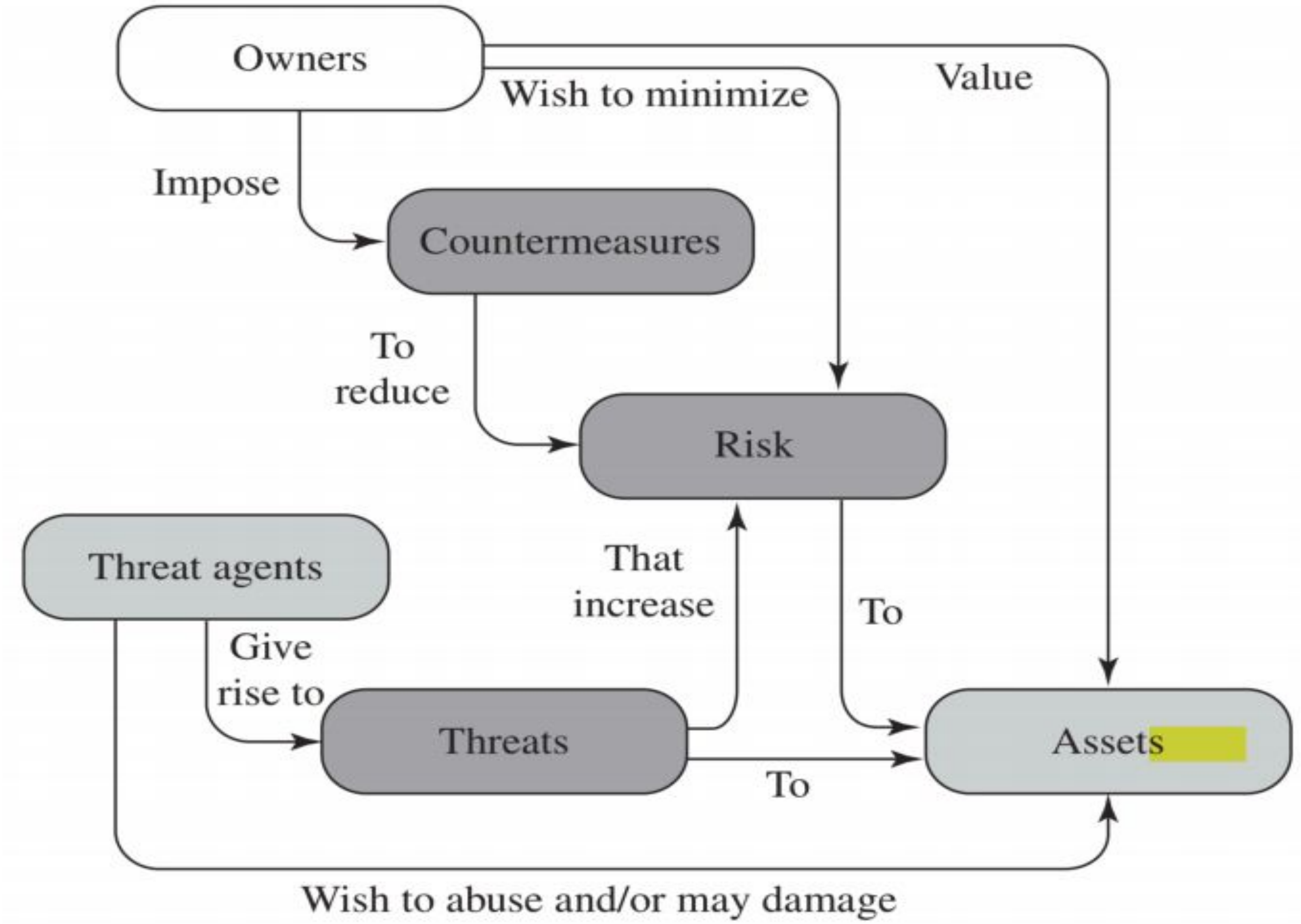
- Threat
  - A potential for violation of security, which exists when there is a circumstance, capability, action, or event, that could breach security and cause harm.
  - Threat is a possible danger that might exploit a vulnerability

Computer Security Terminology

- Attack
    - A deliberate attempt to evade security services and violate the security policy of a system
    - Countermeasure
        - An action that reduces a threat, a vulnerability, or an attack by eliminating or preventing it, by minimizing the harm it can cause, or by discovering and reporting it so that corrective action can be taken

- System Resource (Asset)
    - Data contained in an information system
    - Service provided by a system
    - System capability, such as processing power or communication bandwidth; or an item of system equipment (i.e., a system component— hardware, firmware, software, or documentation); or a facility that houses system operations and equipment.

Concepts & Relations



Other Terms

- Unauthorized Disclosure
  - An entity gains access to data for which it is not authorized
  - Exposure: Sensitive data is directly released to unauthorized entity
  - E.g. Facebook leaks
- Interception:
  - Unauthorized entity directly accesses sensitive data traveling between authorized source and destination
  - E.g. Man-in-the-middle attacks

Other Terms

- Inference:
  - Unauthorized entity indirectly accesses sensitive data (need not directly access the data in communication) by reasoning from characteristics or by-products of communication
  - E.g. attacker can gain information by observing traffic patterns on a network
- Intrusion:
  - Unauthorized entity gains access to sensitive data by circumventing security of system.
  - E.g. Backdoor access to a system

Other Terms

- Deception
  - An authorized entity is fed false data and made to believe it true
- Masquerade
  - An unauthorized entity performs a malicious act by pretending to be an authorized entity
  - E.g. Performing malicious operation on a system by using a Trojan horse
- Falsification: Deceiving an authorized entity using false data
  - E.g. Tampering student grades in AUMS

Other Terms

- Repudiation:
  - An entity deceives another by falsely denying responsibility for an act
  - E.g. user denies sending an email or sending a whatsapp message
- Misappropriation
  - Unauthorized use of services or operating system resources
  - E.g. Cryptojacking
- Misuse
  - Causes a system component to perform a function or service that is detrimental to system security.
  - E.g. hacker that has gained access to a system

Software Security

- Software security is a branch of computer security that focuses on the design and implementation of software
- Using best language, tools and methods
- Focuses on avoiding software vulnerabilities
- Focus of study: This makes it a white-box approach, where as most other approaches are black-box as it ignores the software's internals

Why Software security

Why is code of significance?

- Software defects are the root-cause of security problems
- Other defenses ignore software security and build defenses around it
- Firewalls, Anti-virus tools try to protect the system by building defenses around it
- But when software security persists then attackers often find ways to bypass the defenses

How to Improve Software Security

Blackhat

- What are the security relevant defects that constitute vulnerabilities?
- How are they exploited ?

White Hat

- How do we prevent security-relevant defects (before deploying) ?
- How do we make vulnerabilities we don’t manage to avoid harder to exploit

Other Common Terms

- Malware

  Stands for ‘Malicious Software’ . Any type of code or program used to perform malicious actions

- Vulnerability

  Any weakness that can exploited by malware (attackers)

- Exploit

  Code designed to take advantage of a vulnerability

Other common terms

- Patch

Update to a vulnerable program or a system. A common practice to keep your computer or mobile secure is to install the latest vendors patches in a timely manner

- Social Engineering

A psychological attack used by attackers to deceive their victims into taking an action that will place them at a risk

User Authentication

User Authentication

Fundamental building block and the primary line of defense

- Process of establishes an identity claimed by or for a system entity
- Identification: Presenting a proof of a user claimed identifier to the security system
- Verification: Generating authentication information corroborating the binding between the user and the identifier

Means of Authentication

4 general means of authenticating a user’s identity

- What the user has
  - E.g. electronic key cards or atm cards, physical keys
- What the user is
  - E.g. Face, fingerprint or retina scan
- What the user knows
  - E.g. password, PIN
- What the user does
  - E.g. Typing rhythm, voice pattern

• (sometimes where the user is) e.g. IP address

Password-based Authentication

- Almost all systems have a publicly known user ID and a secret used as a line of defense
- The ID is used to Establish whether the user is authorized to gain access to the system
- Decide on the privileges granted to the user (e.g. as in DAC)
- The password is compared to a previously stored password for that ID maintained in a system password file
- Not stored in clear – they are either encrypted or hashed

Vulnerability of Passwords

- Offline Dictionary Attack
    - An attacker obtains the system password file and compares the password hashes against hashes of commonly used passwords
    - If a match is found, the attacker can gain access
    - Countermeasures:
        - Intrusion detection controls to prevent unauthorized access to the password file
        - Measures to identify a compromise, and rapid reissuance of passwords
- Specific Account Attack
    - The attacker targets a specific account and submits password guesses until the correct password is discovered.
    - Countermeasure - an account lockout mechanism, which locks out access to the account after a number of failed login attempts.
    - Typically, 3-5 attempts

- Popular password attack:

    - Use a popular password and try it against a wide range of user IDs.(exploits the fact that a user’s tendency is to choose a password that is easily remembered)

    - Countermeasures: Policies to inhibit selection of common passwords and scanning the IP addresses of authentication requests

- Workstation Hijacking
    - The attacker waits until a logged-in workstation is unattended
    - Countermeasure is automatically logging the workstation out after a period of inactivity
    - Intrusion detection schemes can be used to detect changes in user behavior

- Exploiting multiple password use
    - Attacks occurs when different devices/accounts share the same or a similar password for a given user.
    - Countermeasures include a password policy that forbids the same or similar password on particular network devices.
- Electronic monitoring.
    - A password transmitted across a network to log on to a remote system is vulnerable to eavesdropping.
    - Simple encryption will not fix this problem, because the encrypted password is, in effect, the password and can be observed and reused by an adversary.

Password Attack

Password Attack

- Any attempt to crack, decrypt or steal password from a system
- Oldest form of attack and there are many automated tools

Password Cracking

The process of guessing or recovering a password

- Brute force
- Dictionary attack
- (why is dictionary attack successful?)

How does Linux store passwords?

Traditional Linux systems stored passwd in /etc/passwd

- Password was stored encrypted, but the file is world readable
- Other info in the file – user id, group id, home dir etc. that must be accessible to other users/processes.

Contents of passwd file

```
$cat /etc/passwd

root:3RaraB1..1ssZ:0:0:root:/root:/bin/bash
bin:*:1:1:bin:/bin:
daemon:*:2:2:daemon:/sbin:
adm:*:3:4:adm:/var/adm:
lp:*:4:7:lp:/var/spool/lpd:
sync:*:5:0:sync:/sbin:/bin/sync
shutdown:*:6:0:shutdown:/sbin:/sbin/shutdown
halt:*:7:0:halt:/sbin:/sbin/halt
mail:*:8:12:mail:/var/spool/mail:
news:*:9:13:news:/var/spool/news:
uucp:*:10:14:uucp:/var/spool/uucp:
operator:*:11:0:operator:/root:
games:*:12:100:games:/usr/games:
gopher:*:13:30:gopher:/usr/lib/gopher-data:
ftp:*:14:50:FTP User:/home/ftp:
nobody:*:99:99:Nobody:/:
xfs:*:100:102:X Font Server:/etc/X11/fs:/bin/false
gdm:*:42:42::/home/gdm:/bin/bash
```

Each line stores one record and each record consists of 7 fields.

Format of /etc/passwd (original)

```
mark:x:1001:1001:mark,,,:/home/mark:/bin/bash
[--] - [--] [--] [-----] [--------] [--------]
|    |   |    |     |         |        |
|    |   |    |     |         |        +-> 7. Login shell
|    |   |    |     |         +----------> 6. Home directory
|    |   |    |     +--------------------> 5. GECOS
|    |   |    +--------------------------> 4. GID
|    |   +-------------------------------> 3. UID
|    +-----------------------------------> 2. Password
+----------------------------------------> 1. Username
```

https://linuxize.com/post/etc-passwd-file/

S.No.	Field	Description
1	Username	String we type when we log into the system.Maximum length is restricted to 32 characters
2	Password	In older linux system, the users encrypted password was stored in /etc/passwd.But in modern system tjis field is set as x and password is stored in /etc/shadow
3	UID	Number assigned to each user
4	GID	User’s group identifier number Typically name of group is same as username. Secondary groups are listed in /etc/groups
5	GECOS	Full name of the user. Contain list of comma separated values Includes User’s fullname, room no, work phone no, home phone no, other contact information

Password Shadowing

Linux Shadow Suite to remove passwords /etc/passwd

- Shadowing is a technique in which the /etc/passwd file remains readable but does not contain passwords
- User passwords are stored in /etc/shadow and is readable only by the root
- Shadow Suite Implements:
  - Enables Password Aging – limits lifespan of a password
  - Automatic account lockout when users don’t change their passwords
- Combine password string with a number of bits known as the salt.

Format of /etc/shadow

The file consists of one record per line broken into nine colon-delimited fields:

S.No.	Field
1	User name
2	Hashing algorithm used
3	Salt
4	Encrypted password
5	#Days since last password change

Modular Crypt Format

Password hash format in this scheme:

$<identifier>$rounds=<number-of-

rounds>$<salt>$<password-hash>

OR

$<identifier>$<salt>$<password-hash>

E.g. Identifier = 6

number of rounds: 4000

salt:ZVzZ72hf

actual hash value:
Tf19cHUK0g.nf.IBpn5jd3jokKMEAIHssRW2OEUGfneuTUzkhNmGv9iDhjfeDpJtqOyGjtSeXSq8

The “Identifier”

The identifier refers to the password hashing scheme

Password Hashing Scheme	Identifier
md5_crypt	1
bcrypt	2
bcrypt	2a
bcrypt	2x
bcrypt	2y
bsd_nthash	3
sha256_crypt	5
sha512_crypt	6
sun_md5_crypt	md5
sha1_crypt	sha1

What is a hash function?

- A hash function takes a variable sized input message and produces a fixed-sized output.
- The output is called as the ‘hash code’ or ‘hash’ or the message digest.
- E.g. SHA-512 hash function takes for input messages of length up to 2128 bits and produces as output a 512-bit message digest
- Hash code is a fixed sized finger print of a variable sized message

An MD depends on all the bits in the input message, any alteration in the input can cause the hash value to change

```
Message:           "A hungry brown fox jumped over a lazy dog"
SHA1 hash code:    a8e7038cf5042232ce4a2f582640f2aa5caf12d2

Message:           "A hungry  brown fox jumped over a lazy dog"
SHA1 hash code:    d617ba80a8bc883c1c3870af12a516c4a30f8fda
```

When is a hash function secure?

One-way property

If it is computationally infeasible to find a message that corresponds to a given hash code.

• Strong collision resistance

If it is computationally infeasible to find two different messages that hash to the same hash code value.

• Is hashing and encryption the same?

Then, what is a difference?

Is Simple Hashing Enough?

Can we directly apply a hash algorithm such as SHA-512

to a user entered password? (minus salting)

• No, the password would still be crackable even though hash

function is still cryptographically secure.

• E.g. password of 6 characters all in lower case. The number of

unique password strings possible = 266

• Easy to construct a lookup table for all such hashes and acquire

the password in the blink of an eye

• The intruder doesn’t need all passwords – simply needs a password or two

Salt

A salt is a randomly chosen bit pattern that is combined with the actual password before it is hashed by a hashing algorithm

• In the example, ZVzZ72hf is the Base64 characters, each standing for 6 bits. This is a 48bit word that will be combined with user‘s password before hashing.

• Older Unix systems

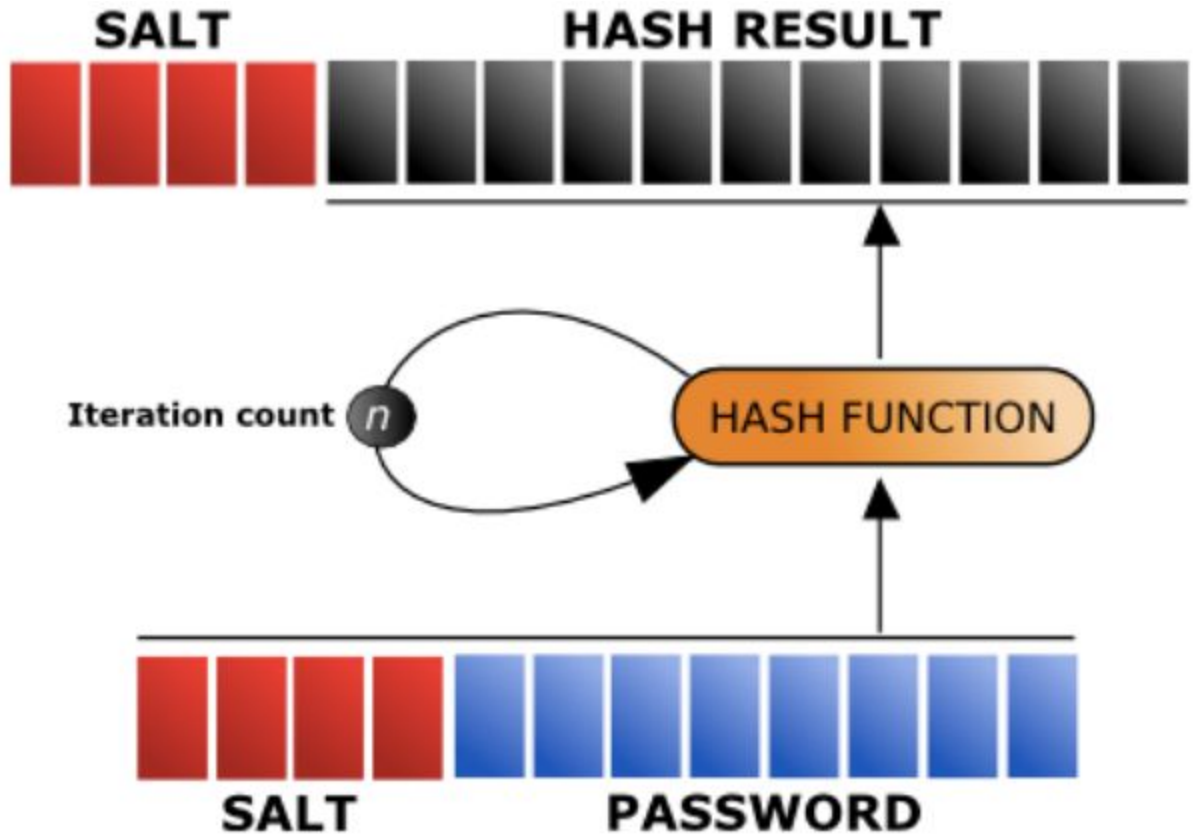
• Password limited to 8 characters and used 12 bit salts

• => # of unique salts = 212 = 4096

• Assuming 95 printable ascii characters ~= 2767.9 TB

• Do-able with modern multi-terabyte data drives

Graphically



Benefits of Salting

- It prevents duplicate passwords from being visible in the password file (even if two users choose the same password) salt values.
- 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 2b
- Makes it nearly impossible to find out whether a person with passwords on two or more systems has used the same password on all of them.

Salt Strategies

Fixed Salt

• A fixed sequence of bytes that will be used before hashing every password

• Variable Salt

• Generated by computer separately for each password

• Allows each stored password to be decoupled from the others so that even if 2 users use the same password, the hash code generated will be different

Number of Rounds

Modern hashing scheme hashes a password along with its salt multiple times.

If the intruder has access to the hashing scheme and the number of rounds, why then do we need to hash multiple times?

- Hashing multiple times makes it harder to crack a password through look up

Currently most password hashes uses a default of 5000

- Reason – protection provided by salts is considered to be strong enough for a few more years to come

- Additional protection can be provided by introducing a variable number of rounds.

Conclusion

Salt in itself is capable of thwarting rainbow table/ lookup attacks using tables off the internet.

• Would take years for the attacker to create his own rainbow tables that account for every possible salt

• Number of rounds adds one more level of complexity making dictionary attacks harder

• Randomizing number of rounds can offer more resistance to dictionary attacks – but there’s no need for it now

Why is Password Cracking Possible

- Simple answer : Hashing functions are not resistant to collision. E.g. MD5, LM Hash

- SHA family are more secure but even that will fail

- The following two facts have given much importance to the development of password cracking methods during the last twenty years:

- The older versions of the Microsoft Windows platform used an extremely weak method for hashing passwords

- The near universality of the Windows machines all around the world.

The Dictionary Attack

- The list of most commonly used passwords is called Dictionary
- Why does Dictionary Attack work?
  - Users typically choose weak passwords OR
  - Users choose short passwords – number of transformations required to encrypt one is small.
  - Crack, John The Ripper

LM Hash

Password hashing used on Windows System

• The password hashing used in the older versions of the Windows platform is known as the LM Hash where LM stands for LAN Manager.

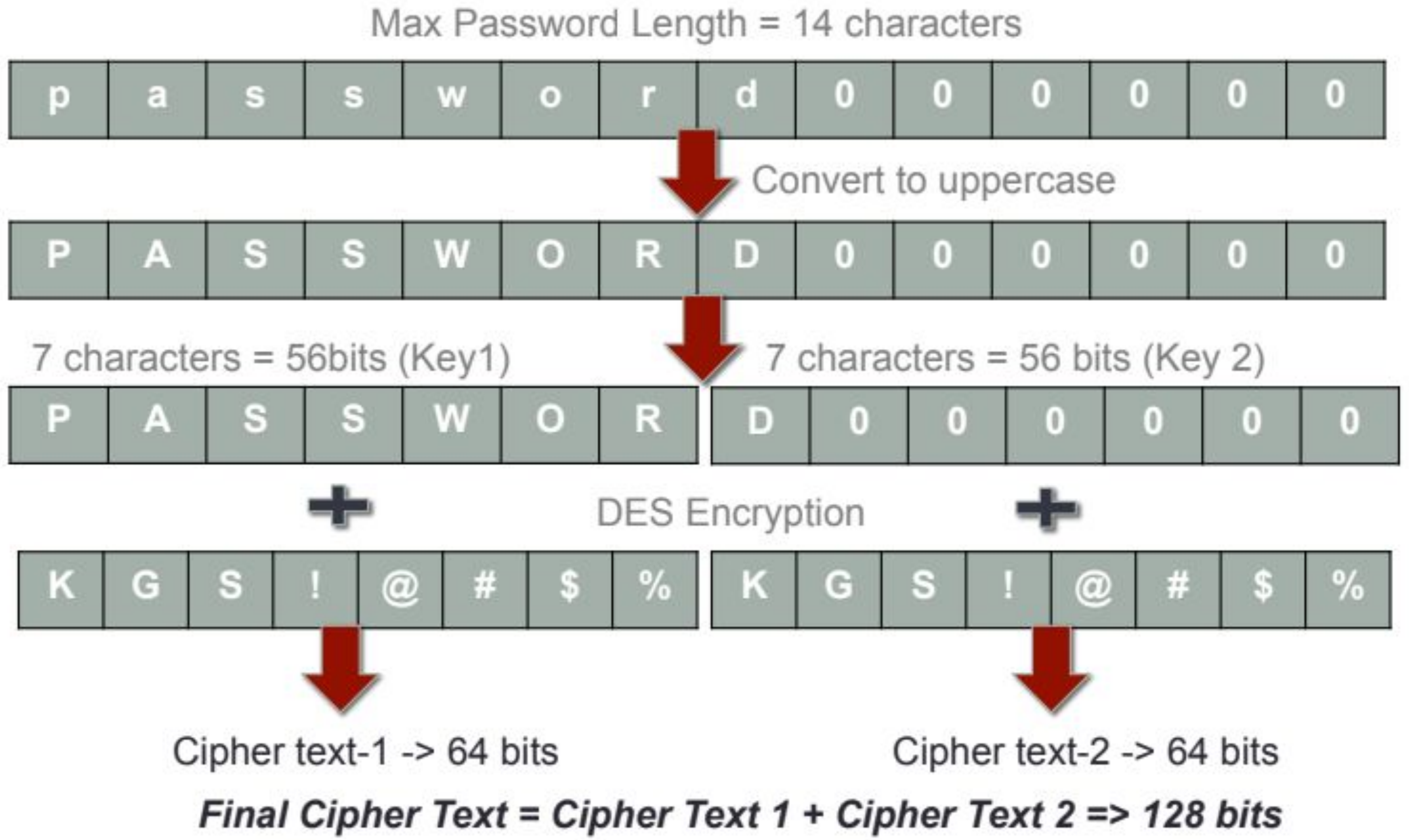
• This hashing function is so weak that a password can be cracked in just a few seconds

LM Hash Algorithm

Password Length limited to a maximum of 14 ascii characters

- Any lower case characters are converted to upper case
- The string is divided into two substrings 7 characters each
- 56bits of each substring is then used to encrypt the 8 char plain text KGS!@#$%
- KGS stands for Key of Glen & Steve

LM Hash Algorithm



LM Hash Algorithm

Each half produces a 64-bit cipher text and two cipher text

bit streams are simply concatenated together to create a

128-bit pattern that is stored as the password

• # of unique password strings possible for each 7

character half :

Total printable ascii chars = 95

When only upper upper case = 95-26 = 69

697 ~= 243 unique strings

Vulnerabilities of LM Hash

Limitations of DES Algorithm

- Easy to guess if the original password string is shorter than 8 chars
- Two halves of the hash value can be attacked separately
- Approximately 243 distinct hash values possible – not a very large number
- If len(password) < 8, then result is predictable
- Since the second half of the input string is all zeros whose DES encryption is given by the hex 0xAAD3B435B51404EE

Lookup Table Hash Maps

What if we constructed the hash values of the most

common passwords and stored it as HashMaps?

• <hash,password> pairs can be stored in a giant disk-

based hash table.

• To construct a lookup table for LM Hash:

• For each half of the password approx 243 password strings are

likely

Stroage for LM Hash

243 ~= 9 x 1012 strings

1 string = 7bytes

9 x 1012 strings = 63 x 1012 bytes of storage

~= 63TB of storage

$50 for 1TB

Very inexpensive today!!!

You can further bring the space down by using meaningful passwords

(Assume no collisions)

Thwarting Dictionary Attack With Log Scanning

- How it starts: Scanning IP Addresses for vulnerabilities at open ports.
- Example : SSH daemon uses port 22
- The attacker tries a large number of common user names and passwords.
- On a Ubuntu system, authentication log will show such scans

/var/log/auth.log

Trying for user names

#Account name tried: staff

Apr 10 13:59:59 moonshine sshd[32057]: Invalid user staff from 61.163.228.117

Apr 10 13:59:59 moonshine sshd[32057]: pam_unix(sshd:auth): check pass; user unknown

Apr 10 13:59:59 moonshine sshd[32057]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh

Apr 10 14:00:01 moonshine sshd[32057]: Failed password for invalid user staff from 61.163.228.117 port 40805 ssh2

#Account name tried: sales

Apr 10 14:00:08 moonshine sshd[32059]: Invalid user sales from 61.163.228.117

Apr 10 14:00:08 moonshine sshd[32059]: pam_unix(sshd:auth): check pass; user unknown

Apr 10 14:00:08 moonshine sshd[32059]: pam_unix(sshd:auth):

authentication failure; logname= uid=0 euid=0 tty=ssh

Apr 10 14:00:10 moonshine sshd[32059]: Failed password for invalid user

sales from 61.163.228.117 port 41066 ssh2 ruser= rhost=61 ruser=

rhost=61

Trying for password, user = root

Apr 10 16:23:20 moonshine sshd[32301]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 Apr 10 16:23:22 moonshine sshd[32301]: Failed password for root from 202.99.32.53 port 42273 ssh2

Apr 10 16:23:29 moonshine sshd[32303]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 Apr 10 16:23:32 moonshine sshd[32303]: Failed password for root from 202.99.32.53 port 42499 ssh2

Apr 10 16:23:39 moonshine sshd[32305]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 Apr 10 16:23:41 moonshine sshd[32305]: Failed password for root from 202.99.32.53 port 42732 ssh2

Sample log file for scenario when sshd tries to reverse map the source ip address and fails

Apr 10 21:41:58 moonshine sshd[757]: reverse mapping checking .....

[78.153.210.68] failed - POSSIBLE BREAK-IN ATTEMPT!

Apr 10 21:41:58 moonshine sshd[757]: pam_unix(sshd:auth):

authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=78.1

Apr 10 21:41:59 moonshine sshd[757]: Failed password for root from

78.153.210.68 port 43828 ssh2

Apr 10 21:42:01 moonshine sshd[759]: reverse mapping checking .....

[78.153.210.68] failed - POSSIBLE BREAK-IN ATTEMPT!

Apr 10 21:42:01 moonshine sshd[759]: pam_unix(sshd:auth):

authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=78.1

Apr 10 21:42:02 moonshine sshd[759]: Failed password for root from

78.153.210.68 port 43948 ssh2

Apr 10 21:42:03 moonshine sshd[761]: reverse mapping checking .....

[78.153.210.68] failed - POSSIBLE BREAK-IN ATTEMPT!

Apr 10 21:42:04 moonshine sshd[761]: pam_unix(sshd:auth):

authentication failure; logname=uid=0 euid=0 tty=ssh ruser= rhost=78.1

Apr 10 21:42:06 moonshine sshd[761]: Failed password for root from

78.153.210.68 port 44058 ssh2

A Quick Aside

Simplest way to prevent dictionary attack is by using /etc/hosts.allow , /etc/hosts.deny

• For instance, if you want to SSH into your home computer from your work and want to block all other connections out:

/etc/hosts.allow : sshd:xxx.xxx.xxxx.xxx

/etc/hosts.deny : ALL : ALL

Logscanning Example Tool

Using a tool such as DenyHosts by Phil Schwartz

• Script for Linux admins to help thwart SSH attacks.

• DenyHosts scans logs to detect repeated unsuccessful attempts from an IP address automatically blacklists it. In case of repeated attacks for SSH servers

• Automatically adds an IP addr to /etc/hosts.deny

• Poweful synchronization feature that allows it to download IP addresses that have been blacklisted elsewhere.

Denyhosts Example

tried to connect as vanessa:

Apr 25 16:29:33 moonshine sshd[31049]: reverse mapping .... [190.12.41.50] failed - POSSIBLE BREAK-IN ATTEMPT!

Apr 25 16:29:33 moonshine sshd[31049]: Invalid user vanessa from 190.12.41.50

Apr 25 16:29:33 moonshine sshd[31049]: pam_unix(sshd:auth): authentication

failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=190.12.41.50

Apr 25 16:29:34 moonshine sshd[31049]: Failed password for invalid user vanessa from 190.12.41.50 port 54406 ssh2 tried to connect as alyson:

Apr 25 16:29:38 moonshine sshd[31051]: reverse mapping .... [190.12.41.50]

failed - POSSIBLE BREAK-IN ATTEMPT!

Apr 25 16:29:38 moonshine sshd[31051]: Invalid user alyson from

190.12.41.50

Apr 25 16:29:38 moonshine sshd[31051]: pam_unix(sshd:auth): authentication

failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=190.12.41.50

Apr 25 16:29:39 moonshine sshd[31051]: Failed password for invalid user

Denyhosts Example

tried again to connect as root:

Apr 25 16:29:42 moonshine sshd[31053]: reverse mapping .... [190.12.41.50] failed - POSSIBLE BREAK-IN ATTEMPT!

Apr 25 16:29:42 moonshine sshd[31053]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=190.12.41.50

Apr 25 16:29:44 moonshine sshd[31053]: Failed password for root from 190.12.41.50 port 54509 ssh2

AND FINALLY CAUGHT BY DENYHOSTS:

Apr 25 16:29:50 moonshine sshd[31060]: refused connect from ::ffff:

190.12.41.50 (::ffff:190.12.41.50)

Twarting Dictionary Attack

iptables is a tool written by Rusty Russell

• Originally a userspace program but now a part of kernel

• Tool to insert & deletes rules from the kernels packet

filtering table.

• Packet filter is a piece of software that looks at the header

of packets and decides the fate of the entire packet

Lookup-table attack

Lookup-table attack: a password hash is attacked by looking up a table of previously computed hashes

- (Why does it store hashes?)
- Stores all possible passwords
- But constant look-up
- Lookup tables are available for download or can be got on a physical media from different vendors
- Net admins uses such tables to test for password attacks
- Bad guys use it to crack passwords

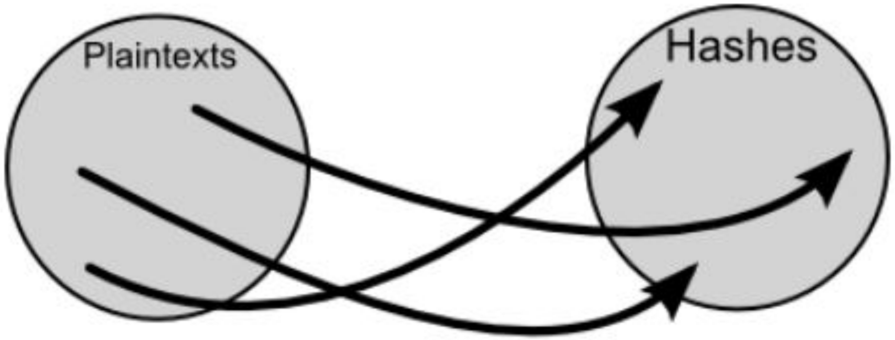
Rainbow Table

Rainbow table trades time for memory

- i.e not constant lookup time
- Optimizes on the amount of disk space used by using hash chains.
- Idea of rainbow tables was invented by Phillipe Oecshlin and is described in his paper “Making a Faster

Cryptanalytic Time-Memory Trade-Off” that appeared in Lecture Notes in Computer Science in 2003.

What is a hash



Hash function is a one way function that takes a variabl size input to produce a fixed size output.

Hash Chain

A sequence of values derived via consecutive applications of a cryptographic hash function to an initial input.

- Same properties of a hashing function apply
- Relatively easy to calculate successive values in the chain
- But given a particular value, it is infeasible to determine the previous value.

Reduction Function

Reduction function is the basis of a hash chain

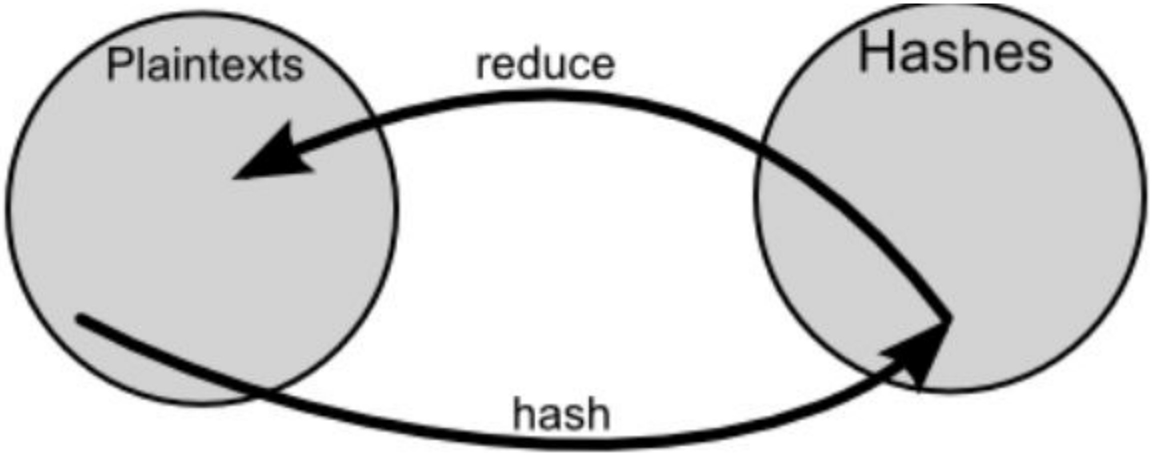
• A reduction function maps a hash to a character string that looks like a password

• Not an inverse operation

• Take the last few bytes of the hash and create any sort of a mapping from those bytes into the space of all possible passwords.

• Maps more than one hash to the same password

Pictorially



Even though reduction function does the reverse of a hash, it is not an inverse hash.

It gives some other plain text, not the original.

Eg. MD5(“435678”) = "222f00dc4b7f9131c89cff641d1a8c50“

Reduction maybe as simple as taking the first 6 numbers

Some Logic

Let, p = plain text password

c = hash of the password i.e. c = H(p)

Let R be a reduction function when applied to c gives a string that looks like a plain text p1

p1 = R(c)

Here’s a Hash Chain – application of one hash and one reduction

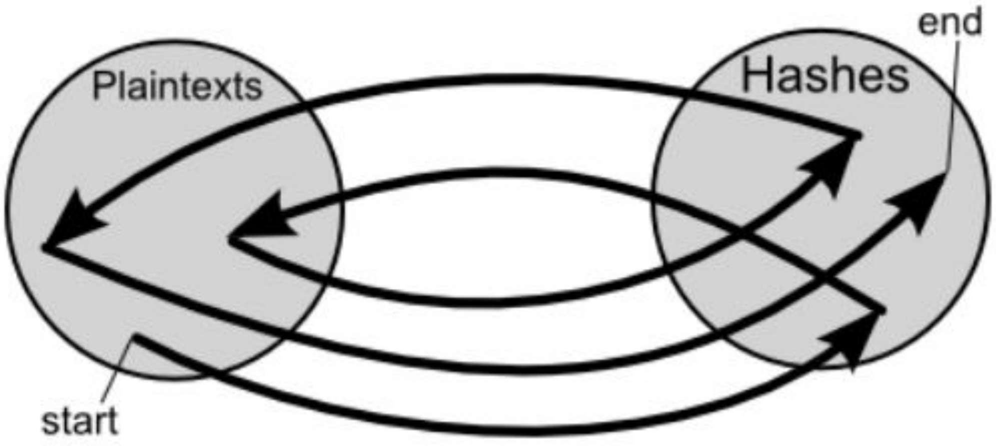
c1 = H(p1) -> p2 = R(c1) 1st link of Hash chain

c2 = H(p2) -> p3 = R(c2) 2nd link of Hash chain

ck-1 = H(pk-1) ->pk= R(ck-1) kth link of Hash chain

Where k is the length of the reduction chain.

Pictorially



The chains which make up rainbow tables are chains of one way hash and reduction functions starting at a certain plaintext, and ending at a certain hash.

Some more logic

Let’s say we store the p1(starting plain text) and pk (ending plain text) in the table as:

Starting point Plain text	Endpoint Also plain text After k steps of R(H(Pk))
$p1^1$ p12 p13 ...	pk1 pk2 pk3 ...

The Actual Cracking

Lets say the attacker wants to use the table to crack a password hash c

• Cracker creates a chain called test hash chain

• Apply the reduction function to get p2

• Then apply H() and R() to p2 and continue

• If any of the plain texts in the test chain matches the end point plain text in second column then there is a high probability that the password that the cracker is looking for in the chain corresponding to that row.

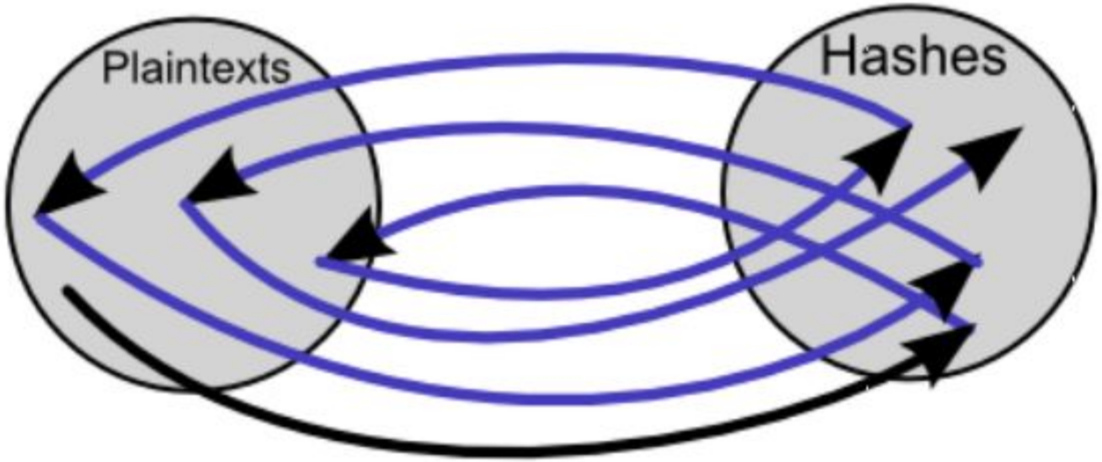
The Actual Cracking Continued

How long to grow the test chain when we look for a match?

• The cracker grows the hash chain k steps which is the same number of steps used in the hash chain table

• If no match then the password does not exist in the table

Pictorially



Start with a user's hashed password

Reduce it , Check for a match

Hash and Reduce

Repeat

The Actual Cracking Continued

What if the endpoint plain text is a match, but we are unable to find the hash in the row?

• This is a False Positive

• Continue to grow the test chain and look for another end point match.

• The passwords stored implicitly in all the chains should span the space of all passwords

• If a legitimate password is not in the starting, end or interior of any chains then there would be no way to get this password from the hash.

Limitations

Any R() will map multiple hashes to the same password string

• If Chain1 has a string at step i and Chain2 has the same string at step j where i != j the 2 chains will traverse the same transitions even though their endpoints will be different.

• Chain1 and Chain2 will occupy different rows in the table even though the passwords overlap

Limitations

The overlap in hash chains is called merging

- The overlap cannot be detected because we only store the starting and ending points.

- Reduces the ability of a hash chain table to crack a password because of reduced overall sampling of the space of passwords

Rainbow Tables

To reduce incidence of chain merging

• Instead of applying the same reduction function through out, use k different reduction functions {R1(), R2()... Rk()}

• Several websites that provide pre-computed rainbow tables for different hash functions

http://www.freerainbowtables.com/en/tables2/

http://project-rainbowcrack.com/

Programming Assignment-1

Adopted & modified from Herbert Bos's course on Network Security

- Given:
- /etc/passwd file
- /etc/shadow file
- A dictionary file
- Goal:
- To recover as many user passwords as possible using a dictionary

of words commonly used in passwords.

Programming Assignment-1

Write a unshadow command (just like in John the ripper),

that combines the contents of the /etc/passwd and /etc/

shadow files to create a combined file called

‘passwordfile.txt’. Your command to run will look like this.

Use Makefile to generate the executable

unshadow /tmp/password /tmp/shadow

Programming Assignment-1

Inputs to the C program is passwordfile.txt and the dictionary file

- To compile – use make

- To run

‘make runall’

The make runall command must automatically run

guessword –i hash.txt –d dictionary.txt –o all

guessword –i hash.txt –d dictionary.txt –o current

guessword –i hash.txt –d dictionary.txt –o root

HINT: Use getopt

Program output

A txt file called ‘allcrackpasswd.txt’ which contains a list of

cracked passwords in the format

username:password

- How many passwords you cracked
- Time you took to crack those many passwords
- Can you improve on your time?

Programming Assignment-1

Program must :

• Be indented & documented properly.

• Be written entirely by yourself

• Use proper coding standards – if you don’t know what it is

– google it. J

• Not invoke external programs

• Not use external libraries other than GNU libc and –lcrypt

• Compile and run on a standard installation of Ubuntu