Samuel Steinberg

September 26th, 2019

CS366

Project 2

Assignment 1

Problem 1:

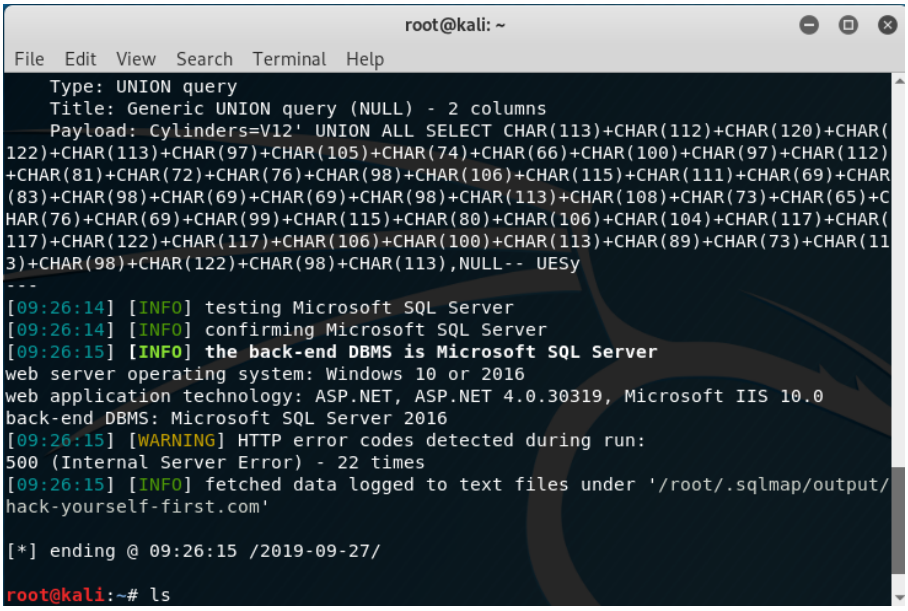
a) Logic bomb: This program has a perfectly legitimate host program, but when certain conditions are met or at a predefined time (in this case Friday the 13th) a malicious function that crashes the computer will be set off.
b) Trapdoor: This program has a trapdoor allowing the user "133t h4ck0r" to login without providing a valid username or password. A trapdoor is a secret entry point into an application, service, etc. that can allow one with malicious intent to cause havoc. Trapdoors allow those who know the backdoor to bypass normal security measures, which is the case in this piece of code.

Problem 2:

a) Paul has no read or write access to the document (SECRET, {B, C}).
b) Anna has no read or write access to the document (CONFIDENTIAL, {B}).
c) Jesse only has read access to the document (CONFIDENTIAL, {C}).
d) Sammi only has read access to the document (CONFIDENTIAL, {A}).
e) Robin only has write access to the document (CONFIDENTIAL, {B}).

Assignment 2

Task 1:



Question 1: After observing the sqlmap execute in real time and reviewing the output afterwards, the ‘Cylinders’ parameter on the website is vulnerable. The ‘Cylinders’ parameter was shown to have UNION based columns, which when vulnerable can allow for the appendage of malicious queries onto seemingly legitimate queries. This will be especially helpful since the sqlmap also returned some of the structure of the ‘Cylinders’ table with this response: “[INFO] GET parameter 'Cylinders' is 'Generic UNION query (NULL) - 1 to 20 columns' injectable”. To sum up the vulnerabilities, the sqlmap found five vulnerable injection points: Boolean-based blind, error-based, stacked queries, time-based blind, and UNION query with the UNION injection point offering the largest payload.

Question 2: For the SQL injection: 1 OR 1=1 the command works because 1=1 is always true and the logical operator OR only requires a single true statement, so the query will always execute and can give users access to data they might not even be eligible to access. For example, consider a query such as the following:

SELECT * FROM Users WHERE id = 100 OR 1=1;

Here, all rows from the Users table will be returned because the condition OR 1=1 is always true, so the statement will always execute (assuming no defense). In more general terms, the data returned will be whatever the user specifies in the statement before that command, unless limited by a command after (such as LIMIT 1, etc.)

The SQL injection: 1 EXEC XP_ will create a shell or another system command by which a malicious user can create havoc. The EXEC keyword in SQL runs a stored procedure or code and XP_ (such as xp_cmdshell) allows users to run a shell command. The prepend of 1

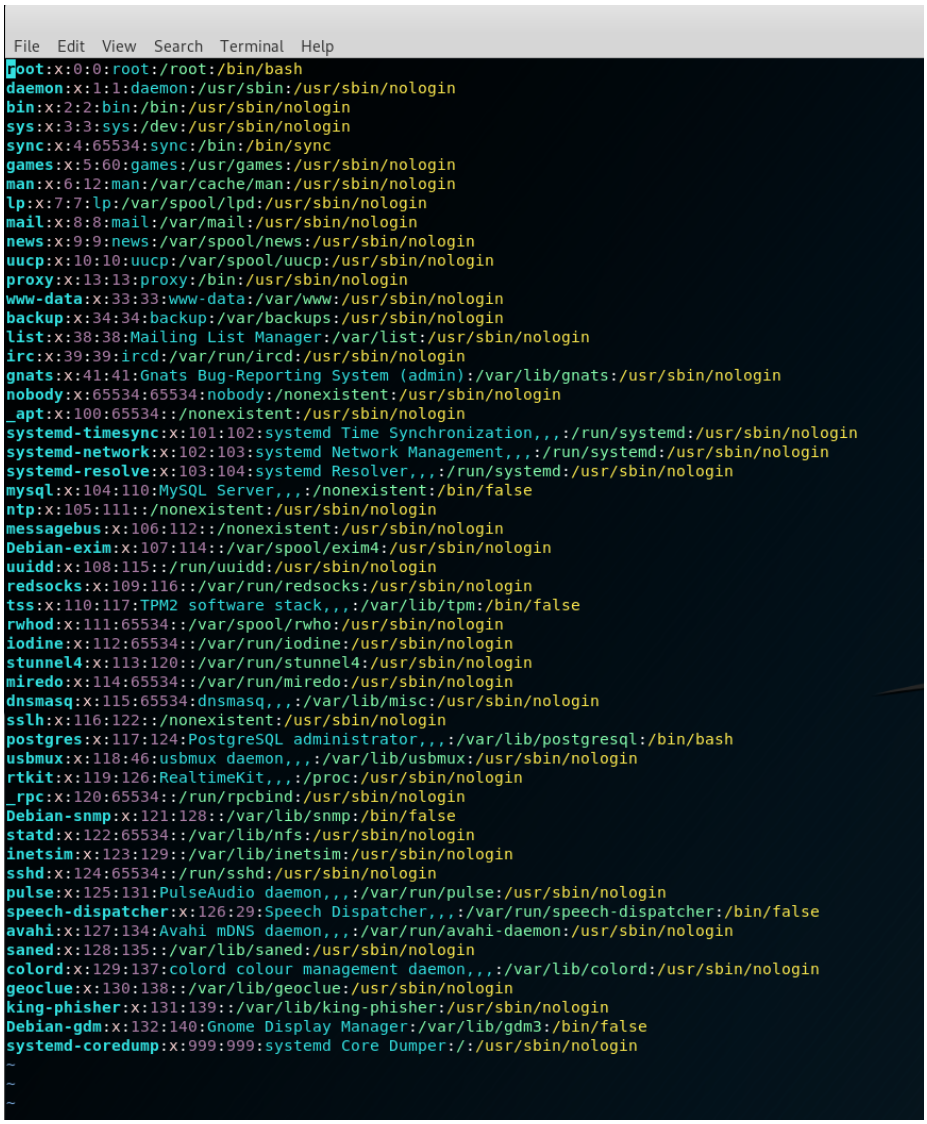
makes sure that the EXEC will always run. In the database this allows users to run free. Consider the following (truncated) command:

1 EXEC xp_cmdshell ‘bcp select * from Users’ … ‘ “outputfile.txt” ’

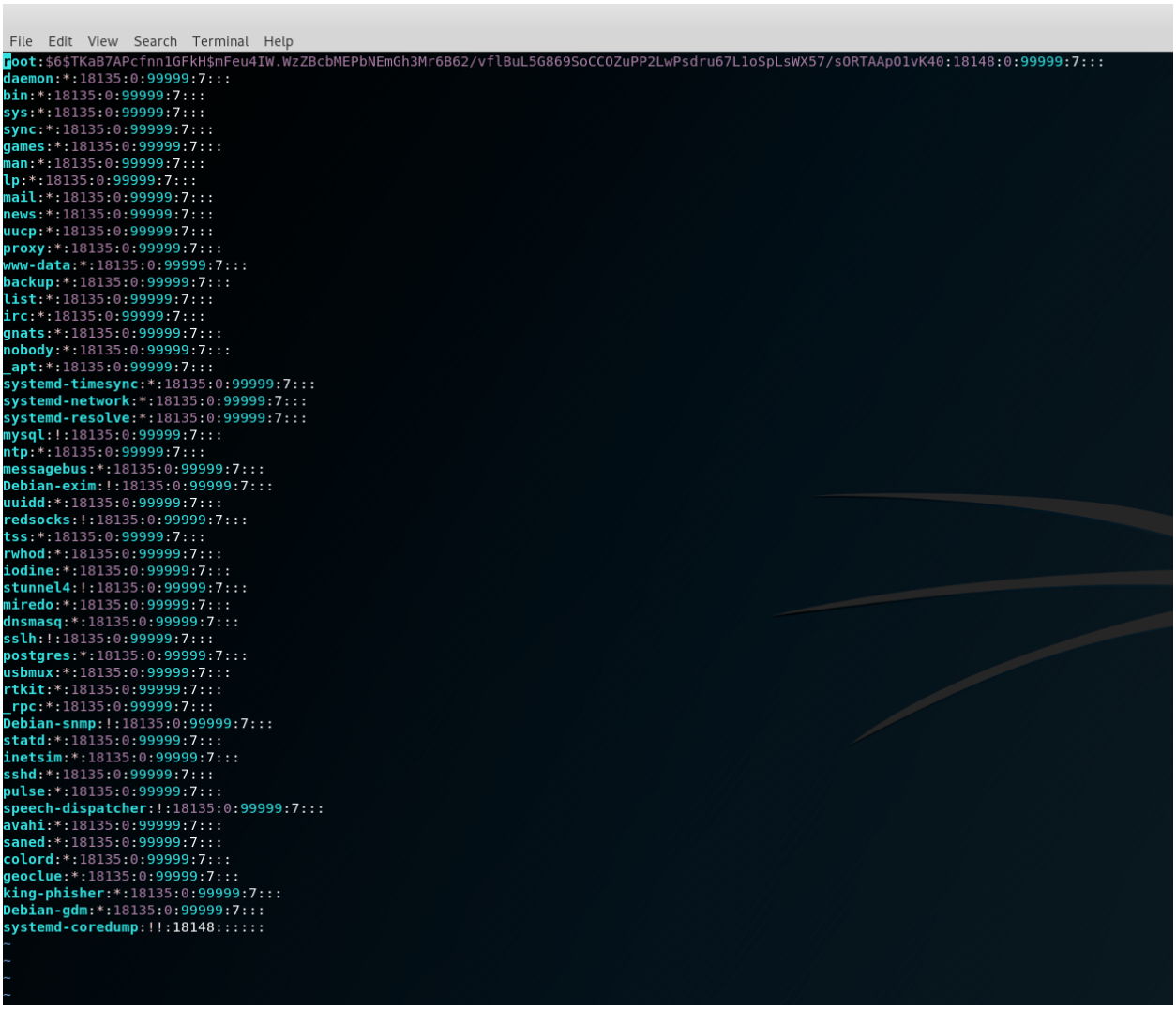
To clarify, in this command the ‘…’ represents more commands that might be input by the user to compile specific data, and in addition there might be more information added to the end of the statement for more information. As for the actual statement above, the 1 will ensure that EXEC xp_cmdshell will always run. Here, the command is running a bulk copy program (bcp) on a database where it is selecting all the information from the Users column and sending it into a text file named “outputfile.txt”. With no defense, commands like this can be run in a shell to harvest malicious data and run commands like ‘bcp’ or other utilities offered by SQL servers.

Task 2:

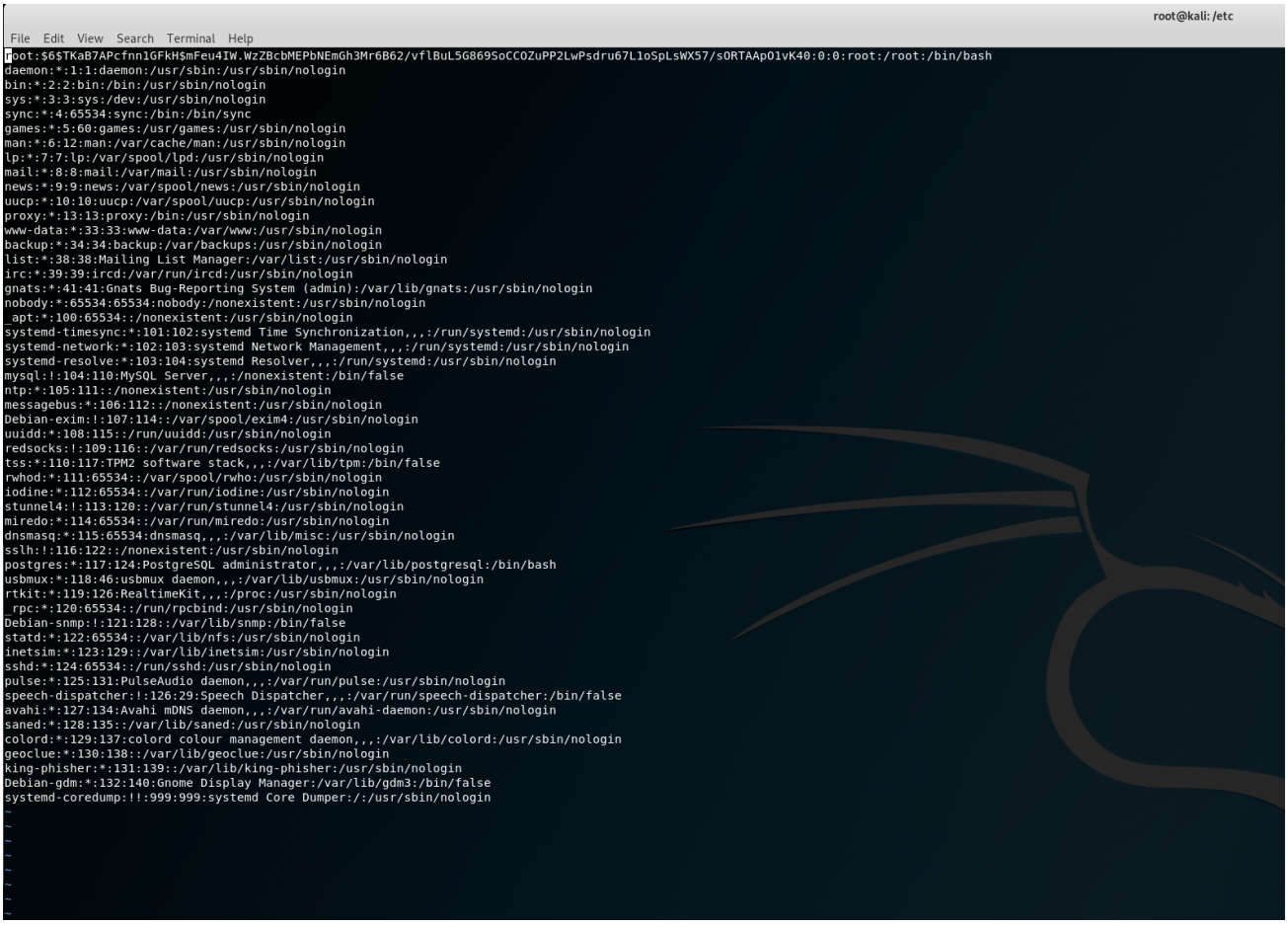
/etc/passwd:



/etc/shadow:


File Edit View Search Terminal Help
root:6TKaB7APcfnn1GFkH$mFeu4IW.WzZBcbMEPbNEmGh3Mr6B62/vflBuL5G869SoCCOZuPP2LwPsdru67L1oSpLsWX57/sORTAAp01vK40:18148:0:99999:7:::
daemon:*:18135:0:99999:7:::
bin:*:18135:0:99999:7:::
sys:*:18135:0:99999:7:::
sync:*:18135:0:99999:7:::
games:*:18135:0:99999:7:::
man:*:18135:0:99999:7:::
lp:*:18135:0:99999:7:::
mail:*:18135:0:99999:7:::
news:*:18135:0:99999:7:::
uucp:*:18135:0:99999:7:::
proxy:*:18135:0:99999:7:::
www-data:*:18135:0:99999:7:::
backup:*:18135:0:99999:7:::
list:*:18135:0:99999:7:::
irc:*:18135:0:99999:7:::
gnats:*:18135:0:99999:7:::
nobody:*:18135:0:99999:7:::
_apt:*:18135:0:99999:7:::
systemd-timesync:*:18135:0:99999:7:::
systemd-network:*:18135:0:99999:7:::
systemd-resolve:*:18135:0:99999:7:::
mysql:!:18135:0:99999:7:::
ntp:*:18135:0:99999:7:::
messagebus:*:18135:0:99999:7:::
Debian-exim:!:18135:0:99999:7:::
uuidd:*:18135:0:99999:7:::
redsocks:!:18135:0:99999:7:::
tss:*:18135:0:99999:7:::
rwhod:*:18135:0:99999:7:::
iodine:*:18135:0:99999:7:::
stunnel4:!:18135:0:99999:7:::
miredo:*:18135:0:99999:7:::
dnsmasq:*:18135:0:99999:7:::
sslh:!:18135:0:99999:7:::
postgres:*:18135:0:99999:7:::
usbmux:*:18135:0:99999:7:::
rtkit:*:18135:0:99999:7:::
_rpc:*:18135:0:99999:7:::
Debian-snmp:!:18135:0:99999:7:::
statd:*:18135:0:99999:7:::
inetsim:*:18135:0:99999:7:::
sshd:*:18135:0:99999:7:::
pulse:*:18135:0:99999:7:::
speech-dispatcher:!:18135:0:99999:7:::
avahi:*:18135:0:99999:7:::
saned:*:18135:0:99999:7:::
colord:*:18135:0:99999:7:::
geoclue:*:18135:0:99999:7:::
king-phisher:*:18135:0:99999:7:::
Debian-gdm:*:18135:0:99999:7:::
systemd-coredump:!!:18148:::::

After running the command: **unshadow /etc/passwd /etc/shadow > combined.txt** the following was output to a new file “combined.txt”:



Question 3: The user root has a password hash of:

> mFeu4IW.WzZBcbMEPbNEmGh3Mr6B62/vflBuL5G869SoCCOZuPP2LwPsdru67L1oSpLsWX57/sORTAAO1vK40

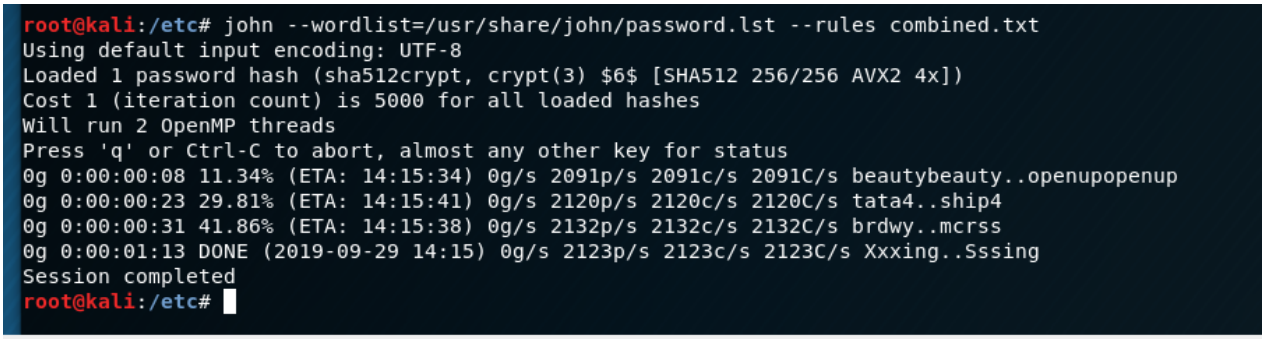
Question 4: The hash algorithm used to generate the hash is SHA-2. This can be found by analyzing the first numeric value in the password hash (between the first two $ signs), which in this case is a 6 (denoting SHA-2). SHA-2 has a numerous different hash sizes, including 224, 256, 384, and 512 bits along with 512/224 and 512/256.

Question 5: Yes, this password is salted. In this case, the salt for root is:

> TKaB7APcfnn1GFkH

A salt is a unique value added to the end of a password to create a different hash value, which is extremely helpful for password security (such as more safety from brute force attacks). This will mean that even the same passwords will be hashed differently. The salt is located after the hash algorithm number and is $-sign separated.

After running: **john –wordlist=/usr/share/john/password.lst –rules combined.txt**



After running the above command and: **john combined.txt**

After running the command: **john –show combined.txt**