ITS Assignment 2

David hbd126, Rasmus dht579

September 2023

1 Tasks

2.1

Here the printenv or env command prints out all the envoirement variables. You can filter these variables by specifying what variable you want printed printenv PWD gives you the password for instance.
Then you can set and unset variables using export to set and unset to unset. This allows us to set a variable with a given value for instance:

export TEST_VARIABLE="TEST!"

¡ Would set TEST_Variable to the value "TEST!". If we want to remove it again we just write

unset TEST_VARIABLE

2.2

Here there is a fork function with the parent process not printing anything. This will result in different values being printed. If we use the diff command to compare the child and parent file then we get the following:

diff filechild fileparent ¿ differences.txt

```
< SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/8232,unix/VM:/tmp/.ICE-unix/8232
---
> SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/2021,unix/VM:/tmp/.ICE-unix/2021
12c12
< SSH_AGENT_PID=7878
---
> SSH_AGENT_PID=1981
14d13
< PWD=/home/seed/Desktop/Labsetup
29,31c28,30
< GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen/b463ece3_52c0_4852_ac36_8f5c6b8a3ce7
< INVOCATION_ID=058995345795431dacc483bca4732cc5
< MANAGERPID=7403
---
> GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen/aa01744a_b96e_4937_b4b8_14b67ff4dd4b
> INVOCATION_ID=eb89c32903eb42e0b63f01194a5b0f86
> MANAGERPID=1777
38c37
< GNOME_TERMINAL_SERVICE=:1.93
---
```

```
> GNOME_TERMINAL_SERVICE=:1.157
43c42
< JOURNAL_STREAM=9:42739
---
> JOURNAL_STREAM=9:38640
48d46
< _=./a.out
49a48
> _=./a.out
```

Between each compression we get a string with two integers and a letter in between. The integers are line number and the char is a key that tells you what has happened. c means that there is a change between lines and d means there are different lines in the files.

From this we can see that the child and parent process are similar but they get different outputs.

2.3

The initial program does not print out anytihng as we have not specified an enviroment variable.

When we add the variable environ, then this program prints out the environment variables, which can be used to get details about the session. For instance looking the output we can see that the user is ”seed” and we’re running in Ubuntu among many others

2.4

Yes it does print the system variables.

2.5

When we set up the two files and change the ownership of the program to root we get the following difference:

These are pretty much the same and we are not able to see the path we set up because it is from the user’s shell it is created and the program is now running with root permission instead, so essentially a different user. We would guess this is a security feature to prevent you from seeing things other users are doing, even root level admin.

2.6

Yes we were able to get it to run the malicious code because of the altered Path. This is probably because the file now has root privileges which allow us to run any code and the modified path allows us to get access to the user Path which sis a security loophole that can be exploited to install malware on a users account.

2 Tasks

2.1

Here the printenv or env command prints out all the envoirement variables. You can filter these variables by specifying what variable you want printed printenv PWD gives you the password for instance.
Then you can set and unset variables using export to set and unset to unset. This allows us to set a variable with a given value for instance:

export TEST_VARIABLE="TEST!"

¡ Would set TEST_Variable to the value "TEST!". If we want to remove it again we just write

unset TEST_VARIABLE

2.2

Here there is a fork function with the parent process not printing anything. This will result in different values being printed. If we use the diff command to compare the child and parent file then we get the following:

diff filechild fileparent ¿ differences.txt

```
< SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/8232,unix/VM:/tmp/.ICE-unix/8232
---
> SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/2021,unix/VM:/tmp/.ICE-unix/2021
12c12
< SSH_AGENT_PID=7878
---
> SSH_AGENT_PID=1981
14d13
< PWD=/home/seed/Desktop/Labsetup
29,31c28,30
< GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen/b463ece3_52c0_4852_ac36_8f5c6b8a3ce7
< INVOCATION_ID=058995345795431dacc483bca4732cc5
< MANAGERPID=7403
---
> GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen/aa01744a_b96e_4937_b4b8_14b67ff4dd4b
> INVOCATION_ID=eb89c32903eb42e0b63f01194a5b0f86
> MANAGERPID=1777
38c37
< GNOME_TERMINAL_SERVICE=:1.93
---
```

```
> GNOME_TERMINAL_SERVICE=:1.157
43c42
< JOURNAL_STREAM=9:42739
---
> JOURNAL_STREAM=9:38640
48d46
< _=./a.out
49a48
> _=./a.out
```

Between each compression we get a string with two integers and a letter in between. The integers are line number and the char is a key that tells you what has happened. c means that there is a change between lines and d means there are different lines in the files.

From this we can see that the child and parent process are similar but they get different outputs.

2.3

The initial program does not print out anytihng as we have not specified an enviroment variable.

When we add the variable environ, then this program prints out the environment variables, which can be used to get details about the session. For instance looking the output we can see that the user is "seed" and we're running in Ubuntu among many others

2.4

Yes it does print the system variables.

2.5

When we set up the two files and change the ownership of the program to root we get the following difference:

These are pretty much the same and we are not able to see the path we set up because it is from the user's shell it is created and the program is now running with root permission instead, so essentially a different user. We would guess this is a security feature to prevent you from seeing things other users are doing, even root level admin.

2.6

Yes we were able to get it to run the malicious code because of the altered Path. This is probably because the file now has root privileges which allow us to run any code and the modified path allows us to get access to the user Path which sis a security loophole that can be exploited to install malware on a users account.

3 Short answer questions

3.1 1.

There are many ways to authenticate users. The most common way is a knowledge based authentication, which means that the user has some knowledge required to be authenticated. This could be a password or an answer to a security question.
Another way to authenticate a user could be through possession of a token. This could be a physical ID-card or a mobile-authentication app on your phone.
A third way could be authentication through biometrics. This could be fingerprint, iris-scanning or voice recognition.

Generally these can be categorised as **'Something you know'**, **'Something you have'** and **'Something you are'**.

3.2 2.

To store a password securely we have to avoid having the password in plain text. This means that passwords are not stored, but rather a hash of the password. Using hashing and salting we can create a hashed password which we can then compare with rather than store the password. Hashing is used in this case because its a strong one-way encryption. This means that even if someone hacks our stored passwords they don't have access to the passwords, but only the stored hashes.

3.3 3.

Discretionary access control (DAC), Mandatory access control (MAC) and Role based access control (RBAC).
DAC controls access based on the authorisation given to the identity of a user, based on access rules stating what requestors have access to.
MAC consists of security labels that give users certain levels of security clearance.
RBAC gives users a role, and then limits their access according to the clearance of the specific role.

3.4 4.

The most common ways to mitigate SQL injection attacks is to use parameterised statements, input whitelists, input validation and sanitation among others.
Parameterised statements serve to separate the user input from the SQL code, so that malicious input never reaches the code.
Whitelists serve to remove access to certain escape characters so that users can not input SQL code directly into the input. The reason a whitelist is used rather

than a blacklist, is because its easier to maintain a static list of allowed inputs, rather than a list of disallowed inputs that can grow with time.
Input validation and sanitation is used to clear the input of potentially harmful SQL code and ensure that it conforms to expected forms of input and rejects any disallowed input.