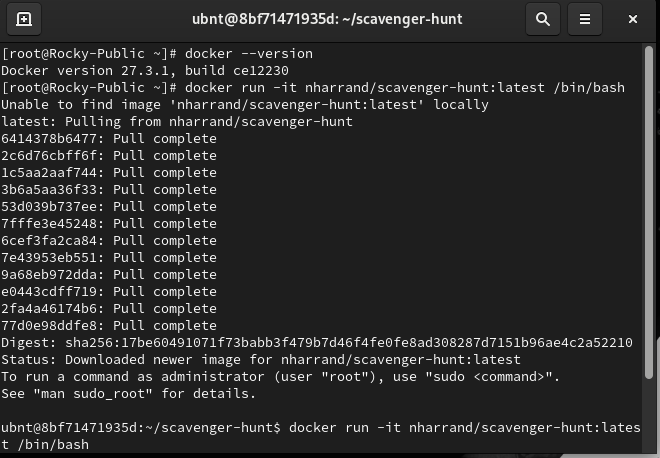
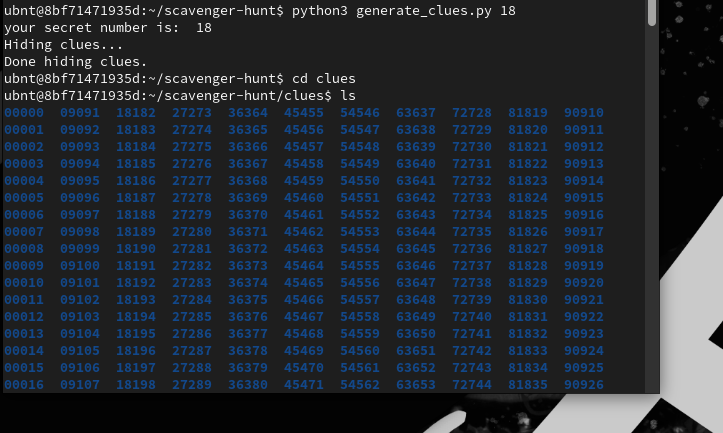
0

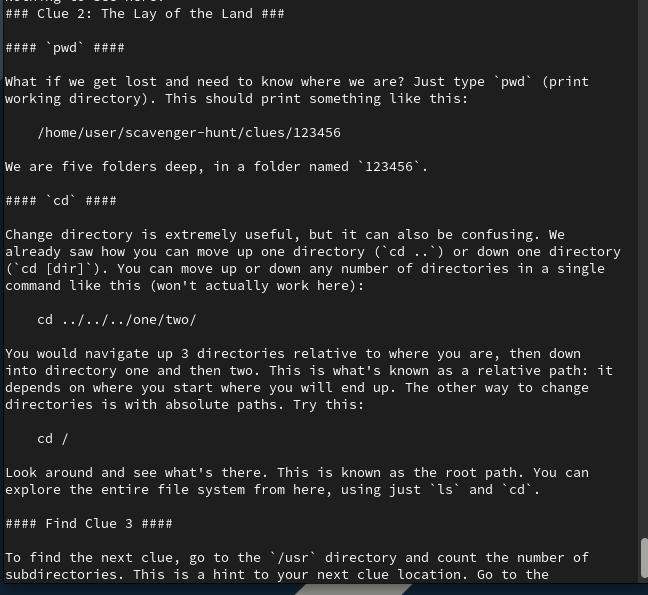
1. Challenge 0.1 (Simple): Find the login/password combination used in Lab 0's scavenger hunt.  
   login ubnt  
   Password :ubnt



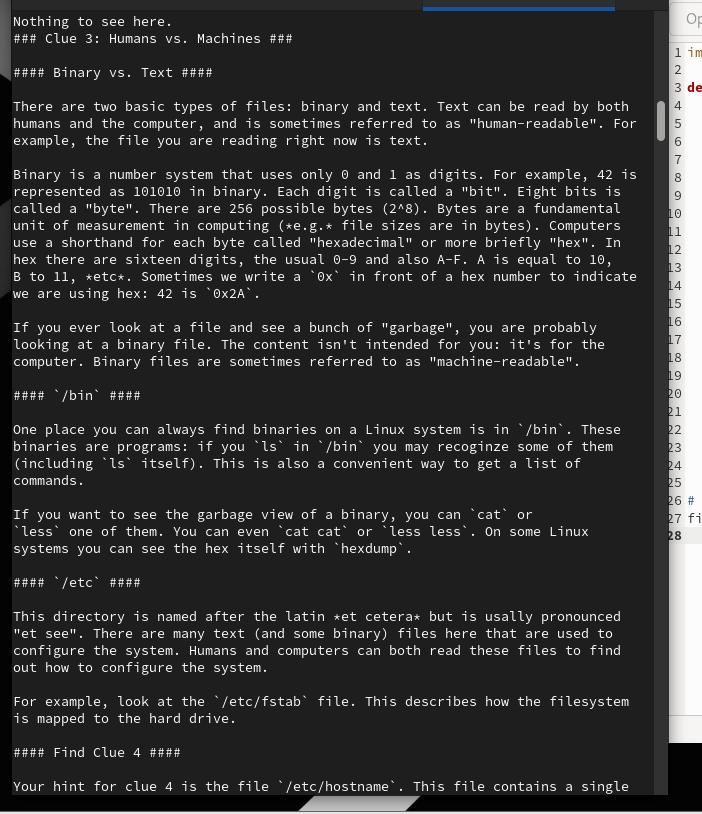


1. Challenge 0.2 (Normal): Write a script (or a one-liner) that finds all the clues at once in Lab's 0 scavenger hunt.





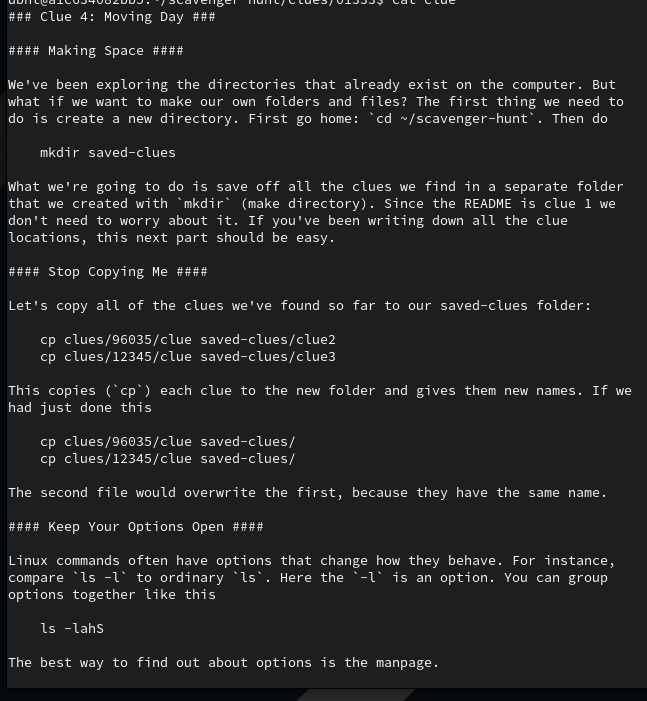
3rd clue



A screenshot of a computer

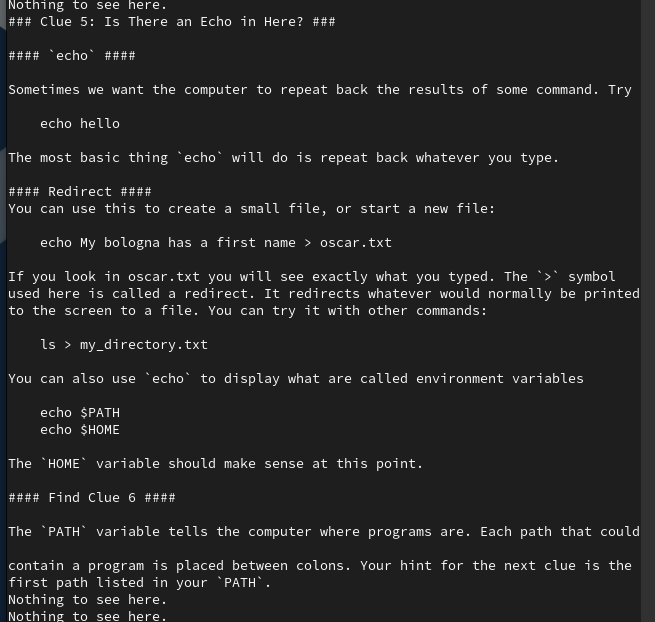
Description automatically generated

4th clue



A screenshot of a computer

Description automatically generated



A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

1

1. Challenge 1.1 (Simple):
   1. D\_AZ\_5H7S006\_9WHF6BHD\_33HX\_5  
      VHSAH3WS0AHIJHX3SY0H064WH6XH  
      AZW4HS9WHX\_3WH5S4WVHX3SYH5HT  
      BAH064WA\_4W0HAZWHX3SYH\_0HZ\_V  
      VW5H\_5HS56AZW9HX\_3WHAZ\_0H4W0  
      0SYWH\_0HAZWHS50DW9HA6HUZS33W  
      5YWHIHV6AHI

WITHIN\_PASSOIRE\_YOU\_WILL\_FIND

\_AT\_LEAST\_12\_FLAGS\_SOME\_OF\_THEM

\_ARE\_FILE\_NAMED\_FLAG\_N\_BUT\_SOMETIMES

\_THE\_FLAGS\_ARE\_HIDDEN\_IN\_ANOTHER\_FILE

\_THIS\_IMAGE\_IS\_THE\_ANSWER\_TO\_CHALLENGE\_10.DOT\_01

.

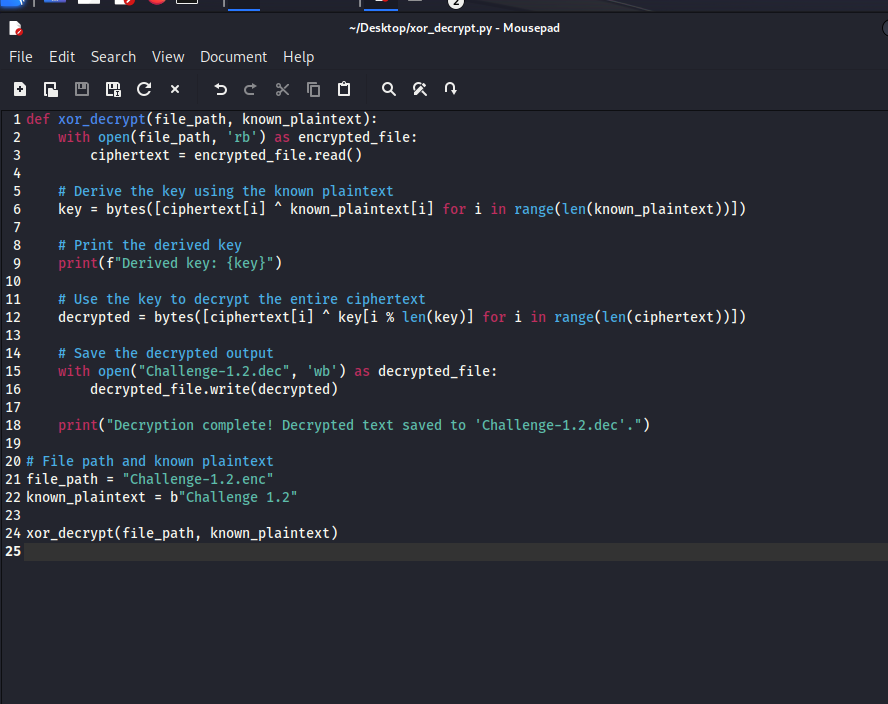
1. Challenge 1.2 (Normal): Decrypt the file '[Challenge-1.2](https://nextilearn.dsv.su.se/mod/resource/view.php?id=18567https://nextilearn.dsv.su.se/mod/resource/view.php?id=18567https://nextilearn.dsv.su.se/mod/resource/view.php?id=18567https://nextilearn.dsv.su.se/mod/resource/view.php?id=18567https://nextilearn.dsv.su.se/mod/resource/view.php?id=18567https://nextilearn.dsv.su.se/mod/resource/view.php?id=18567https://nextilearn.dsv.su.se/mod/resource/view.php?id=18567https://nextilearn.dsv.su.se/mod/resource/view.php?id=18567https://nextilearn.dsv.su.se/mod/resource/view.php?id=18567https://nextilearn.dsv.su.se/mod/resource/view.php?id=18567https://nextilearn.dsv.su.se/mod/resource/view.php?id=18567https://nextilearn.dsv.su.se/mod/resource/view.php?id=18567https://nextilearn.dsv.su.se/mod/resource/view.php?id=18567https://nextilearn.dsv.su.se/mod/resource/view.php?id=18567https://nextilearn.dsv.su.se/mod/resource/view.php?id=18567https://nextilearn.dsv.su.se/mod/resource/view.php?id=18567https://nextilearn.dsv.su.se/mod/resource/view.php?id=18567https://nextilearn.dsv.su.se/mod/resource/view.php?id=18567https://nextilearn.dsv.su.se/mod/resource/view.php?id=18567https://nextilearn.dsv.su.se/mod/resource/view.php?id=18567https://nextilearn.dsv.su.se/mod/resource/view.php?id=18567https://nextilearn.dsv.su.se/mod/resource/view.php?id=18567)'. Hint: It has been encrypted with a key 120bits long, but the file starts with the string "Challenge 1.2" (The plain text is in ascii. Make sure you're not using another encoding when attempting to read the decrypted version.)

Challenge 1.2

This is a simple file that was encrypted. But you already know that since you can read this message.To reward you for your effort, here is a clue for the group project:

Somewhere in the Passoire Docker image, there is a file encrypted with a similar method (albeit slightly harder to decrypt). A combination of a known plaintext attack, brute force, and reverse engineering can help you. Of course, it may be easier to decrypt the file on your own Docker image during Phase II first and think about how you can then decrypt a similar file hidden in the Docker image of the other groups during Phase III, encrypted with a different key.

Good luck!



A computer screen with white text

Description automatically generated

2

1. Challenge 2.1 (Hard): Length Extension Attack:
   1. Extend the following message: "I give you the following amount of SEK coded in binary:\x12". (The message is encoded in ascii.)
   2. "HMAC" follows the insecure pattern h(key | msg), the hash function is SHA-1. The key length is 7 bytes long. (Don't bruteforce it!)
   3. HMAC = 67452301EFCDAB8998BADCFE10325476C3D2E1F0.
   4. You need to produce a message extend the number coded in binary in the given message, and produce a valid "HMAC" = h(key | extended\_msg).
   5. You do not know the key.

Run the python script

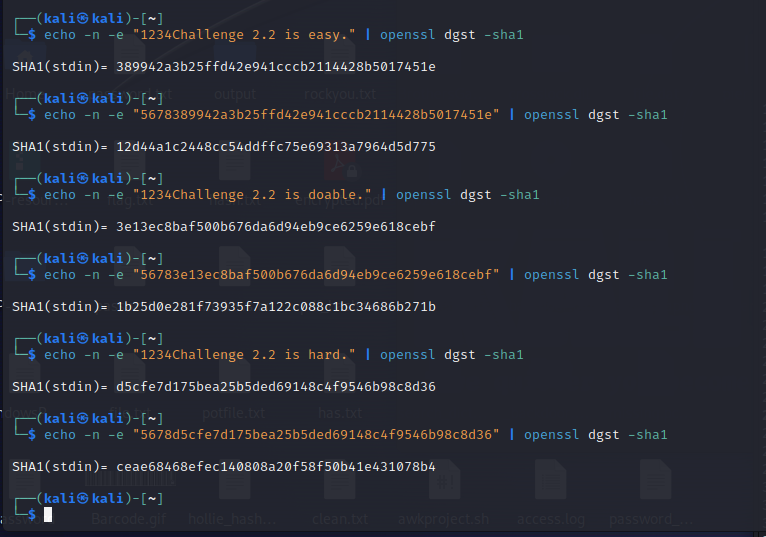
A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

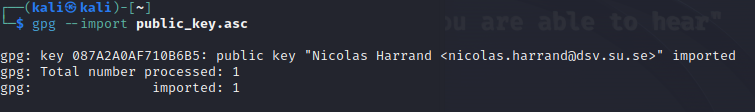
1. Challenge 2.2 (Simple): Verify the HMACs
   1. Key\_1: "1234", Key\_2: "5678", HMAC follows the pattern h(key\_2 | h(key\_1 | msg)), the hash function is SHA-1. The inner hash is used an ascii string. So computing the second hash should look like h("56785aa5787118c3816d7f80c497d557176fae450b93").
   2. Message 1: "Challenge 2.2 is easy." HMAC: 12d44a1c2448cc54ddffc75e69313a7964d5d775
   3. Message 2: "Challenge 2.2 is doable." HMAC: 1b25d0e281f73935f7a122c088c1bc34686b271b
   4. Message 3: "Challenge 2.2 is hard." HMAC: aec64e480f251c6811686597305b04edcc25da35
   5. Which (if any) of these messages have been tampered with.

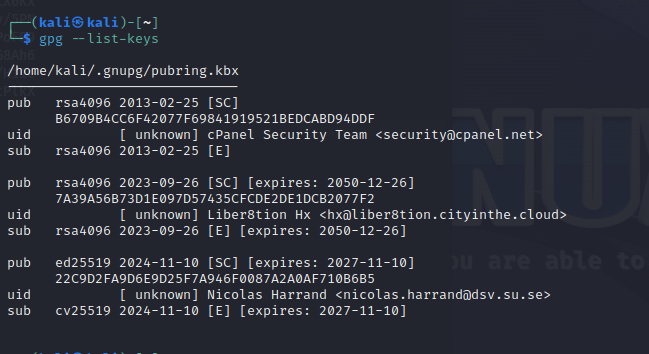


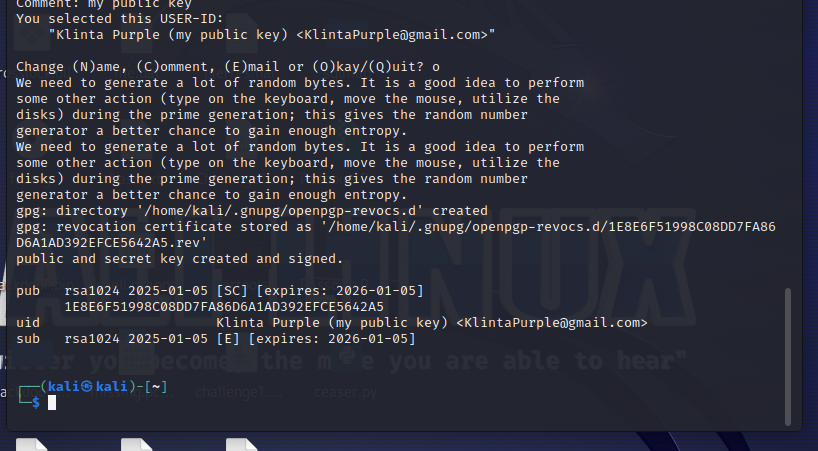
This indicates that **Message 3 has been tampered with**.

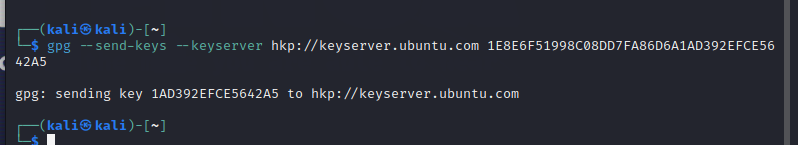
* **Message 1**: Valid (the HMAC matches).
* **Message 2**: Valid (the HMAC matches).
* **Message 3**: Invalid (the HMAC does not match), so this message was likely altered.

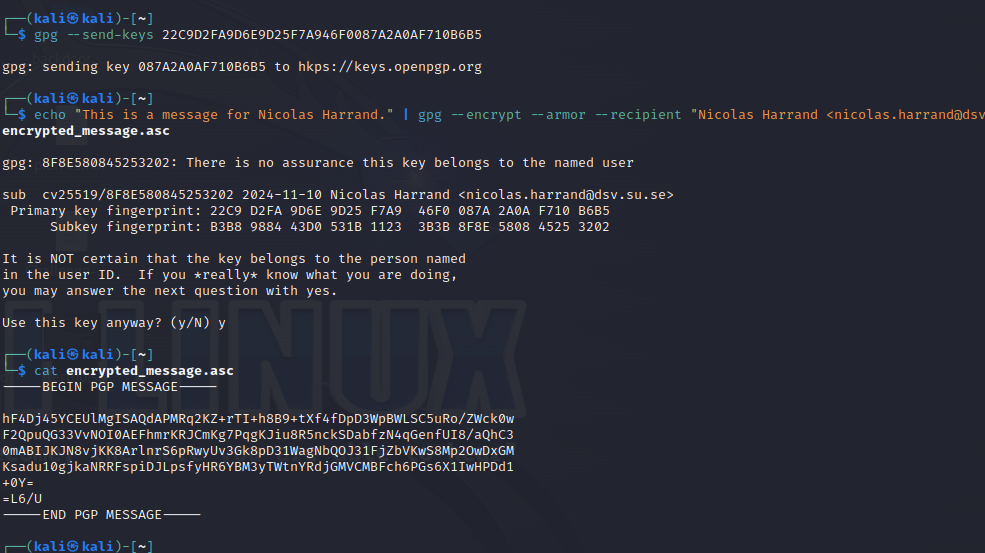
1. Challenge 2.3 (Normal): Send me an email encrypted with PGP:
   1. Read about Web of Trust.
   2. Get my public key from NextILearn.
   3. Send me an email including a link to a source on Web of Trust and your public key.
   4. The email's subject must be "[INTSEC] Challenge 2.3".
   5. Upload your public key on [https://keyserver.ubuntu.com](https://keyserver.ubuntu.com/)
   6. I will answer you using your public key, giving you the secret to put as an answer for the challenge.













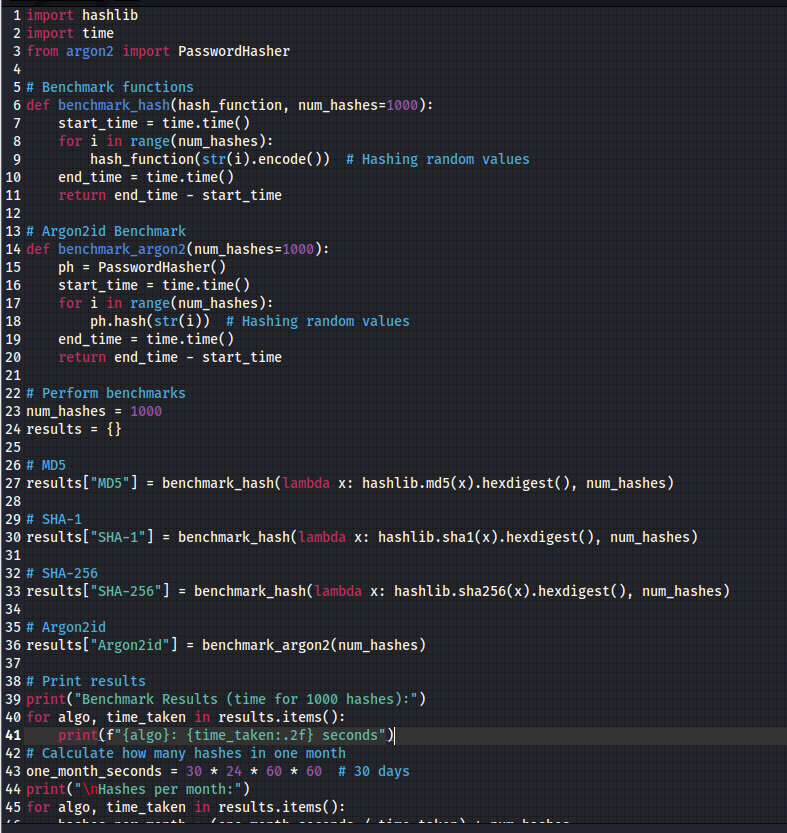
3

1. Challenge 3.1 (Normal): Break weak passwords (with no salt)
   1. They follow the format: Hash-Algo, Hash, Hint
   2. SHA1, 30139264c3ec85759ce4f83c2fe286ecb63e6d43, PIN code: **8695**
   3. md5, c49078e81caafab96c08390197cf6a96, you need to find the right key(s) **5678uiop**
   4. SHA-256, b81848b9e4857c5ed8da601fa6ba92d9c2ee6c6aceabcf5e09813b427dab7bfc, Common password **malachi**
2. Challenge 3.2 (Normal): Break weak passwords with salt h(Salt | password)  
   1. Hash-Algo, Hash, Salt, Hint
   2. SHA1,57536215cfe9781d21733fcab27a653e9db92577,1fa6,PIN code: **3649**
   3. SHA256

8421f0e3432bb339f3671341bc1ec96f6eb283dbf65bb56793065458c20cf945,cb63, L1tera11y a password

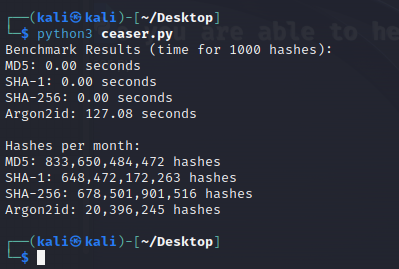
* 1. md5,e75a0b86d4f30e2e56a73cbe9d7dbf07,e098, You need to know something (obvious) about me.

1. Challenge 3.3 (Normal): Benchmark  
   1. Try to compute 1000 hashes of any random values with: (MD5, SHA-1, SHA-256, Argon2id)
   2. How many attempt could you make in a month for each? (Let me know if you used a GPU.)



A screen shot of a computer code

Description automatically generated



Benchmark Results (time for 1000 hashes):

MD5: 0.00 seconds

SHA-1: 0.00 seconds

SHA-256: 0.00 seconds

Argon2id: 127.08 seconds

Hashes per month:

MD5: 833,650,484,472 hashes

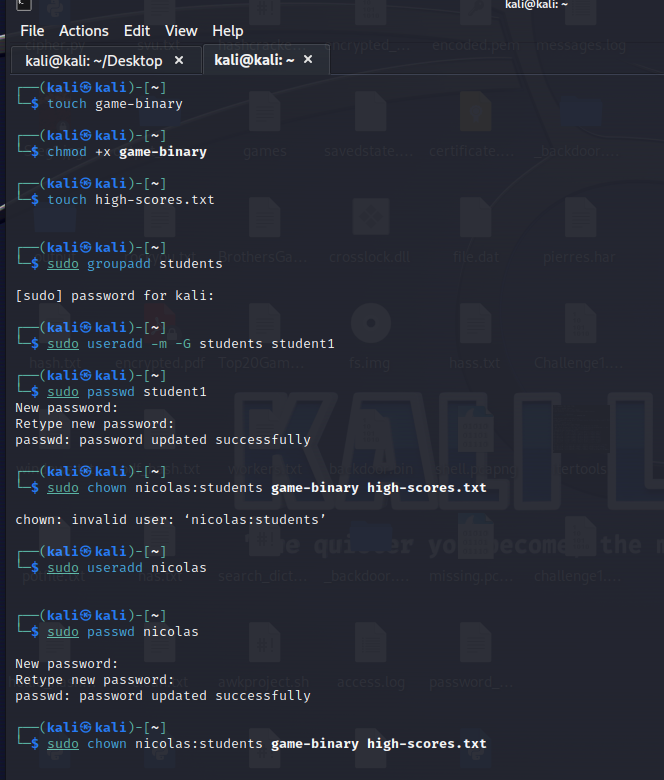
SHA-1: 648,472,172,263 hashes

SHA-256: 678,501,901,516 hashes

Argon2id: 20,396,245 hashes

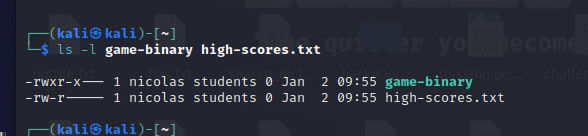
4

1. Challenge 4.1 (Simple):  
   1. Fictional scenario: Imagine I wrote a game for the student of the course. I put it on a linux server where all students have an account and are part of the group students. Here are the files related to the game:
   2. - \_ \_ \_ \_ \_ \_ \_ \_ \_ nicolas students game-binary
   3. - \_ \_ \_ \_ \_ \_ \_ \_ \_ nicolas students high-scores.txt
   4. Set the rights to let anyone in the group "students" play the game and save their best score, without allowing anyone to cheat. (Apply the principle of least privilege and Fail-safe defaults). You can assume that a process regularly saves and flushes the content of high-scores.txt. (Credit for this problem: Alan Davidson.)



A computer screen shot of white text and black text

Description automatically generated



With these settings:

* **Game binary** can be executed by nicolas and any user in the students group.
* **High scores** can be modified by nicolas and read by members of the students group, but others cannot access or modify them.

1. Challenge 4.2 (Simple):  
   1. Go read the sources of Samy and/or the explanations of its author to answer the following questions:
   2. Myspace prevented users to add most html tags such as <script>, where did Samy put his code?
   3. Myspace prevented user to use essential keywords of javascript such as innerHTML, how did Samy bypassed this limitation?
   4. Write a one-sentence explanation for each of these questions.

**Myspace prevented users to add most HTML tags such as <script>, where did Samy put his code?**

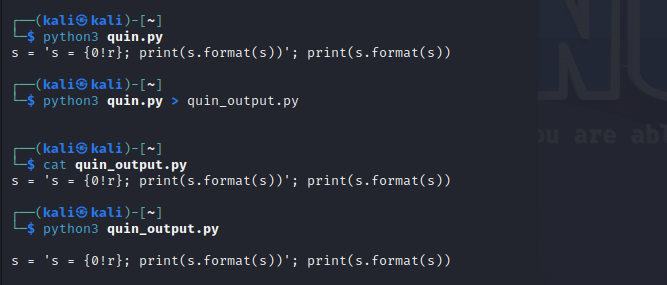
* Samy placed his malicious JavaScript code inside an image tag, using an event handler like onerror to execute the script when the image failed to load.

**Myspace prevented users to use essential keywords of JavaScript such as innerHTML, how did Samy bypassed this limitation?**

* Samy used a combination of different obfuscation techniques, including encoding the JavaScript code and using less obvious properties like document.location to bypass filters on restricted JavaScript keywords.

5:

1. Challenge 5.1 (Hard): Quine
   1. Write a quine in the language of your choice. (Not a trivial quine like "1".)
   2. Don't cheat! If you look for the solution online, you will never again have the opportunity to find it on your own.
   3. If you succed you should be able to run your program, save the output in a file, run this new program, save its output in a file, run, etc and the files should be the same. (Do test it!)
   4. Then you can read Reflections on trusting trust, it's only 3 pages, and it's brilliant.



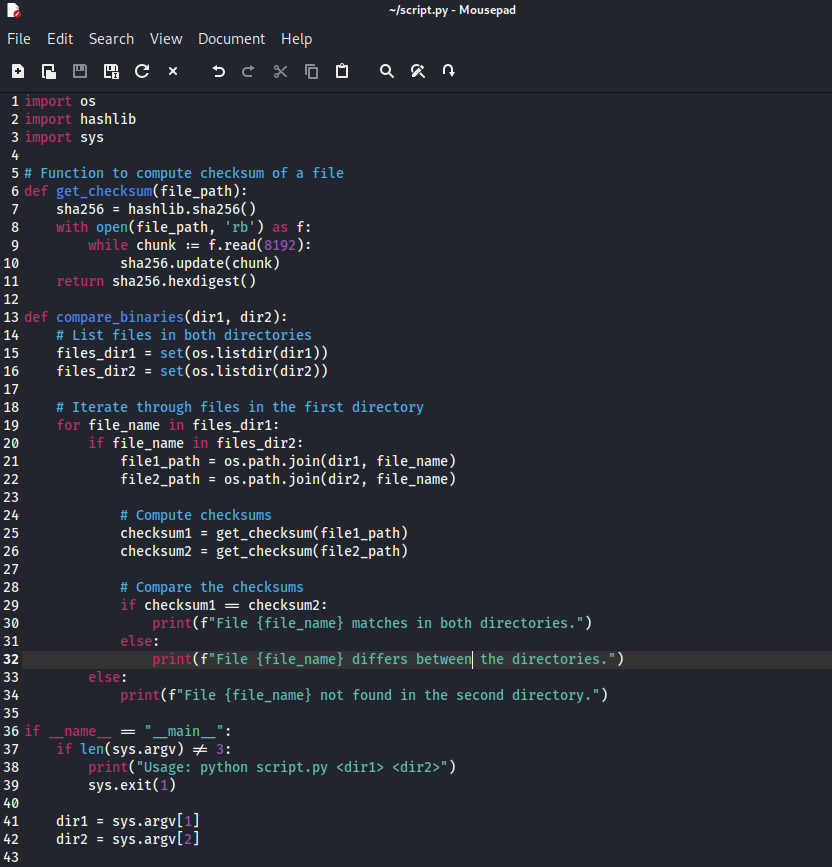
A screen shot of a computer

Description automatically generated

* I wrote a quine that outputs its own source code.
* saved the output of the quine into a new file.
* ran the new file, and it correctly printed the original source code.

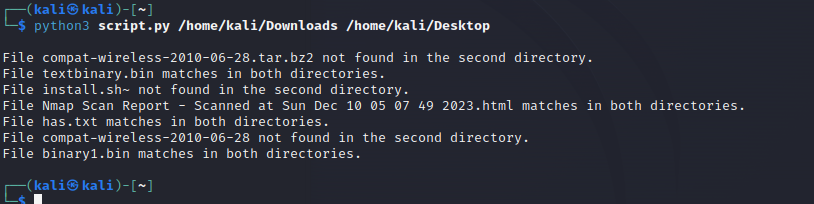
1. Challenge 5.2 (Normal): Checking binaries
   1. Write a short script that takes two directories as input.
   2. For each files in one directory, check that there is another file of the same name in the other directory.
   3. Compute a checksum for each files, and compare.

create a short Python script to compare files in two directories by checking for matching filenames and comparing their checksums



A screen shot of a computer program

Description automatically generated



1. Challenge 5.3 (Normal): Tests
   1. Write 2 to 3 tests for the XOR encryption function you  
      used in Challenge 1.2 (even if you used an external  
      service).
   2. Test the full extent of possible input characters.
   3. Use a unit test framework.
   4. Put in the comments which part is the initialization,  
      function under test, and oracle.
   5. Let me know if you found a bug.



**Initialization (setUp)**:

* This part is executed before each test. In this case, we define a test input string ("Hello, World!") and a key (123) that will be used in the tests.

**Test 1 - test\_basic\_xor**:

* This test checks if the XOR encryption and decryption work as expected.
* We encrypt the string, and then we decrypt it using the same key. The result should be the original string.

**Test 2 - test\_empty\_string**:

* This test checks how the XOR encryption handles an empty string. The encrypted and decrypted result for an empty string should also be an empty string.

**Test 3 - test\_all\_printable\_chars**:

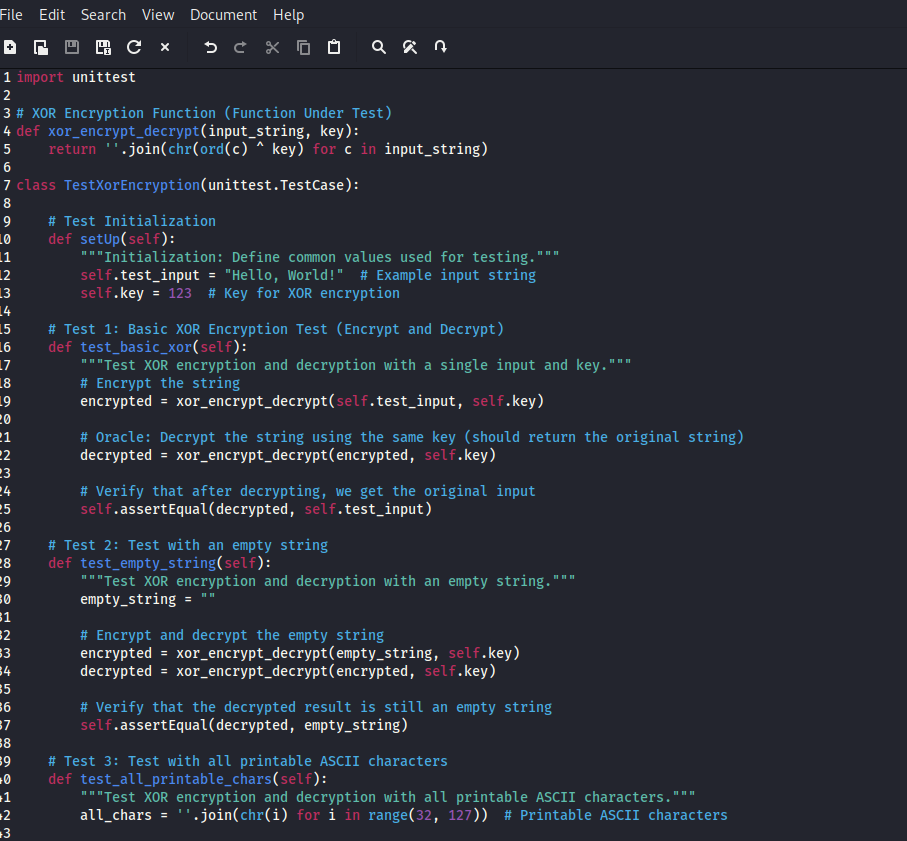
* This test checks if the XOR encryption works with all printable ASCII characters. It ensures that every character between ASCII 32 (space) and 126 (tilde ~) is correctly encrypted and decrypted.

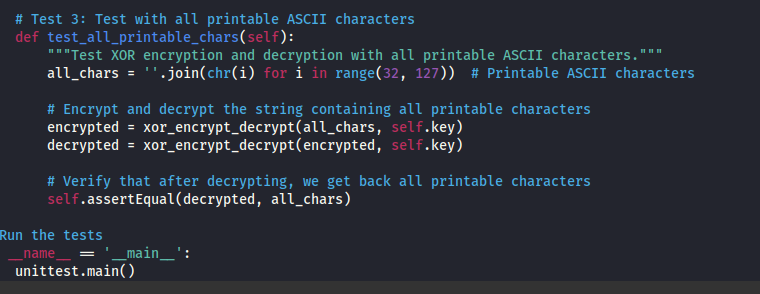
# XOR Encryption Function

def xor\_encrypt\_decrypt(input\_string, key):

return ''.join(chr(ord(c) ^ key) for c in input\_string)

**script**





6

1. Challenge 6.1 (Hard):  
   1. Consider the implementation of indoor security cameras within a household shared by a multi-age family, consisting of members from different age groups. Identify and describe potential security and privacy concerns that may arise due to the presence of such technology. Consider the possible flaws or shortcomings in the design of the camera system, focusing on what might go wrong when deployed in a family setting. From a human-centred perspective, think how these concerns might impact different members of the household, particularly considering generational differences.
   2. Apply both human-centred principles and a feminist perspective to explore the situation. Identify and suggest at least one practical/technical solution that addresses these concerns you have outlined.
   3. Note that avoiding the use of the camera entirely is not an option or  
      solution.
   4. Write your propositions in a separate pdf. (You should metion the pdf name in your csv.)

**1. Identifying Security and Privacy Concerns**

* **Surveillance Misuse**: Cameras may be used to spy on family members, creating distrust.
* **Hacking**: Unauthorized access to the cameras could compromise privacy.
* **Data Storage Risks**: If recordings are stored on insecure servers, sensitive moments could be leaked.
* **Generational Tensions**:
  + Younger members might feel over-monitored or have their autonomy undermined.
  + Older members might feel uncomfortable or excluded from technology-related discussions.
* **Bias in Monitoring**: Cameras might disproportionately focus on specific areas or individuals, reflecting implicit biases.
* **Children's Privacy**: Children may not fully understand the implications of constant surveillance.
* **Impact on Daily Behavior**: Over-surveillance can create a sense of restriction, affecting spontaneity in daily activities.

**2. Applying Human-Centred Principles**

* **Empathy**: Understand the feelings and needs of each family member.
* **Inclusivity**: Design systems that cater to all age groups equally.
* **Transparency**: Ensure all members understand how cameras work, where they are placed, and how the data is used.
* **Control**: Allow family members to have some control over the system, such as pausing recording in private spaces.

**3. Applying a Feminist Perspective**

* **Intersectionality**: Recognize how gender, age, and cultural factors influence privacy concerns.
* **Power Dynamics**: Consider how the camera system might reinforce existing family power structures (e.g., parents monitoring children excessively or one partner monitoring another).
* **Agency**: Ensure all members, particularly vulnerable groups, can voice concerns and participate in decision-making.

**4. Suggesting Practical/Technical Solutions**

* **Selective Monitoring**: Implement privacy zones where cameras don’t record (e.g., bathrooms, bedrooms).
* **Access Controls**:
  + Role-based access to camera feeds and recordings.
  + Multi-factor authentication for remote access.
* **Data Encryption**: Use encrypted storage and transmission to prevent hacking.
* **Notification System**: Notify household members when someone is accessing the camera feeds.
* **User Consent**: Implement opt-in features for specific recordings.
* **Family Agreements**: Establish clear rules on how and when cameras are used.
* **AI-Powered Analysis**: Use AI to alert for potential intrusions while ignoring routine activities, reducing over-surveillance.

7:

1. Challenge 7.1 (Normal): SQL injection
   1. Run sqli-dojo ([Link](https://github.com/h1pmnh/sqli-dojo-docker))
   2. Get the secret value from the "GET parameter injection in the middle of a parameter" page.
   3. Write in your csv the series of SQL injections you needed to perform to obtain the secret value.
2. Challenge 7.2 (Hard): Find the backdoor in passoire's original image. (And no I don't mean just the suspicious user, even though it's related.) You can work on this one with your group, but you all need to submit individually the answer. (I'm expecting detailed explanations.)