| | |
|---|---|
| جامعة إربد الأهلية<br>IRBID NATIONAL UNIVERSITY | **Irbid National University** |
| | **Faculty of Science & Information Technology**<br>**Department of Cyber Security** |
| | **Course: Field Training / Cybersecurity** |
| **Student Name: Alaa Ahmad Bani Irshed** | **St.ID:202220431** |
| **Task 1: Hash Type Detector & MD5 Generator** | **Date: 22 / 11 / 2025** |

==========

**Abstract** This report documents the development of a command-line based Python tool designed for ethical hacking and cybersecurity purposes. The tool serves two primary functions: identifying common cryptographic hash types **(MD5, SHA1, SHA256, SHA384, SHA512)** based on their hexadecimal length and generating **MD5** hashes from plaintext input. The project emphasizes robust input validation, error handling, and user-friendly command-line interface (CLI) design. The tool was developed using Python's standard library and successfully tested on a Kali Linux environment.

**1. Introduction** In the field of cybersecurity and digital forensics, identifying the type of cryptographic hash is a critical first step in analyzing potential security breaches, verifying file integrity, or conducting penetration testing. Different algorithms produce digests of varying fixed lengths. This project aims to automate the identification process for the five most common hash algorithms and provide a utility for generating MD5 hashes. The tool is designed to be lightweight, requiring no external dependencies, making it ideal for quick deployment in restricted environments like Kali Linux.

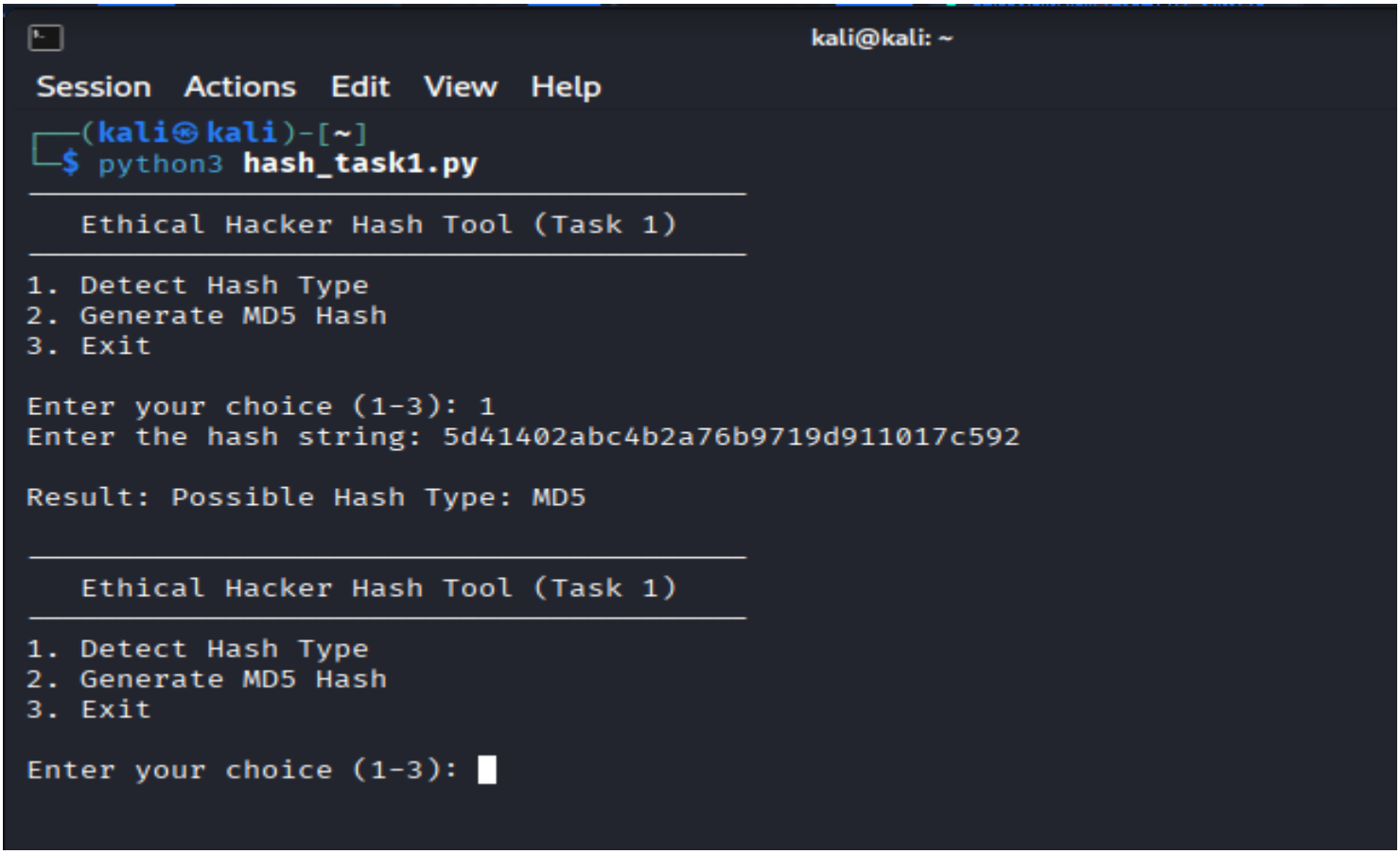
**2. Methodology & Design** The tool is built using Python 3 and relies on the principle of **fixed-length output**, which is a fundamental characteristic of cryptographic hash functions. The development process focused on three main components:

- **2.1 Detection Logic** The core detection mechanism analyses the length of the input string. Since hash functions produce a digest of a specific, deterministic number of bits, the length of the hexadecimal representation acts as a unique signature for identification in this context:
  - **MD5:** 128-bit digest (32 hex characters).
  - **SHA1:** 160-bit digest (40 hex characters).
  - **SHA256:** 256-bit digest (64 hex characters).
  - **SHA384:** 384-bit digest (96 hex characters).
  - **SHA512:** 512-bit digest (128 hex characters).

- **2.2 Input Validation** To ensure robustness, the tool validates user input before processing. It uses Regular Expressions (via the `re` module) to verify that the input string consists solely of valid hexadecimal characters (0-9, A-F). This step prevents runtime errors and filters out invalid data formats.
- **2.3 MD5 Generation** For the hashing feature, the tool utilizes Python's standard `hashlib` library. It takes the user's plaintext input, encodes it into UTF-8 format to ensure consistency across platforms, and processes it through the MD5 algorithm to produce the final hexadecimal digest.

**3. Results & Execution** The tool was tested on a Kali Linux Virtual Machine to verify all functional requirements. The following screenshots demonstrate the successful execution of the tool's features:

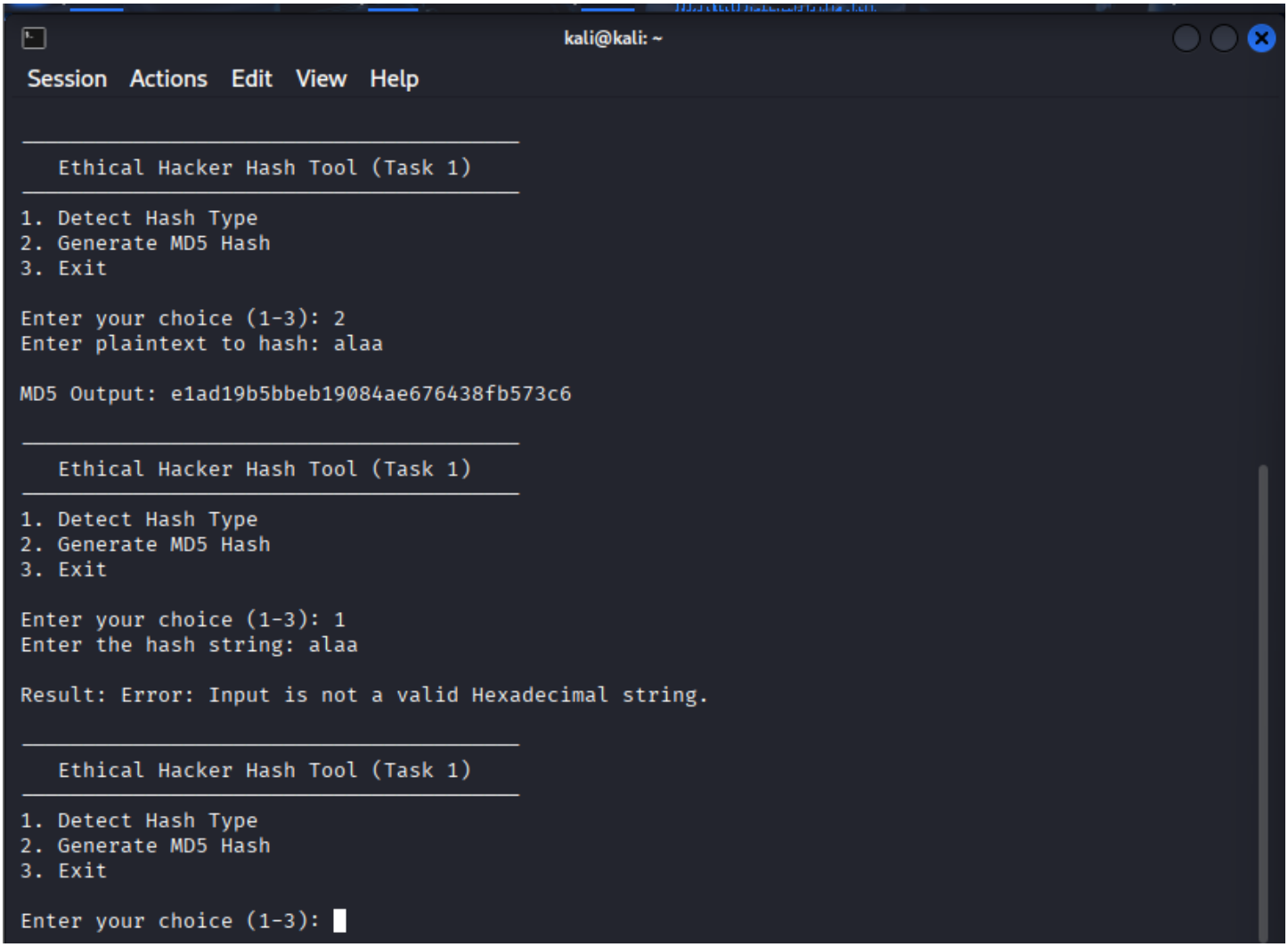
**3.1 Detecting an MD5 Hash** The tool correctly identified a 32-character hash string as MD5.



**3.2 Detecting a SHA256 Hash** The tool successfully differentiated and identified a 64-character hash string as SHA256.


**3.3 Generating an MD5 Hash** The user provided plaintext ("alaa") was successfully converted into its corresponding MD5 digest using UTF-8 encoding.

**3.4 Error Handling** When invalid input (a non-hexadecimal string) was provided, the tool displayed a clear error message instead of crashing, ensuring robust performance.



**4. Discussion & Limitations** While the tool performs its intended task effectively for the most common scenarios, it is important to acknowledge certain limitations inherent in the detection methodology:

- **Ambiguity by Length:** The tool relies solely on the length of the hexadecimal string to infer the hash type. However, multiple algorithms can produce digests of the same length (e.g., MD4 and MD5 are both 128-bit). Without deeper cryptanalysis or metadata, the tool cannot distinguish between these conflicting types based on length alone.
- **Collision Resistance:** The tool generates standard MD5 hashes. It is worth noting that MD5 is considered cryptographically broken for collision resistance in high-security contexts, though it remains useful for file integrity checks and educational demonstrations. Despite these limitations, the tool successfully meets the objectives of the field training task, providing a quick and efficient way to identify likely hash algorithms during basic reconnaissance.

**5. Conclusion** This project successfully delivered a fully functional, robust command-line tool for hash identification and generation. The development process reinforced key cybersecurity programming concepts, including input sanitization, error handling, and the practical application of Python's `hashlib` library. The tool meets all specified functional requirements, passes all test cases, and is properly documented and hosted on GitHub for educational and ethical use.

**6. References**


1. Python Software Foundation. (2025). *hashlib — Secure hashes and message digests*. Python 3.x Documentation.
2. Cyber Security Field Training Course. (2025). *Task 1 Requirements: Hash Type Detector & MD5 Generator*. Irbid National University.