System Security Project

Fake Data Prevention with Conventional Cryptotools

Jaime Hernández Pérez 583770
Maciej Wieteska 579536

Demonstration of how to prevent the injection of fake data into a system using asymmetric cryptography (RSA) and JSON Web Tokens (JWT).
Linux Mint distribution is used in a virtual machine for testing.

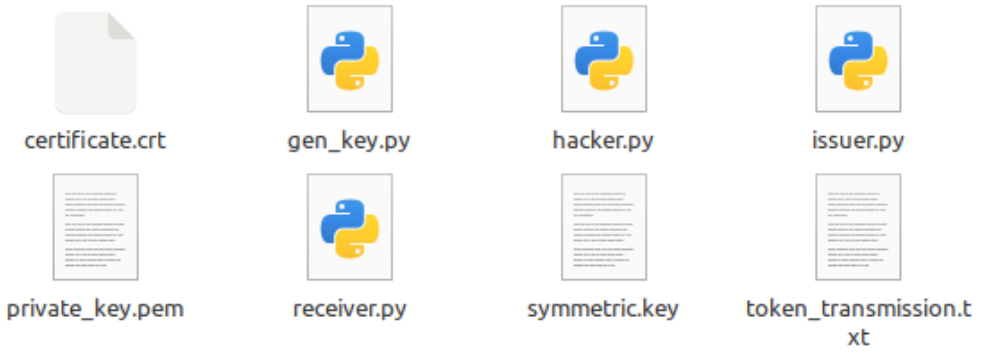
INDEX

Core Components & Roles

- **Issuer:** Generates commands, encrypts the sensitive payload for confidentiality (AES), and signs the token using a Private Key (RSA) for authenticity.

- **Receiver:** Validates the sender's identity using a trusted Digital Certificate (X.509) and decrypts the payload (only if the signature is valid).

- **Attacker:** Intercepts the token but fails to inject fake data because they lack the Private Key to sign the forgery, and fails to read the command because they lack the Symmetric Key to decrypt it.



Libraries & Tools

1. **pyjwt**

- **Standardization**: It packs our data into the official JWT format (Header.Payload.Signature) so it's easy to transmit.
- **The Gatekeeper**: Does Verification. It checks the signature math and enforces our rule to *only* accept RS256 tokens. This blocks the hacker from tricking the system with a weaker algorithm.

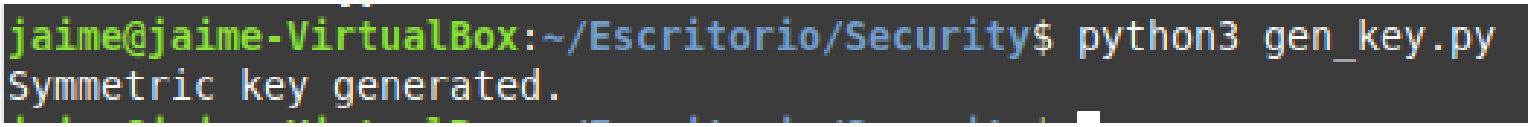
2. **cryptography**

- **Confidentiality (Fernet)**: Applies AES encryption. It turns our command "OPEN_GATE" into unreadable gibberish (gAAAA...) so the hacker can't see what we are sending.
- **Identity (X.509)**: Instead of using raw keys, we use this to read Digital Certificates. It extracts the public key from the certificate file, allowing us to verify the sender's identity professionally.

```
jaime@jaime-VirtualBox:~/Escritorio/Security$ pip install pyjwt cryptography
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: pyjwt in /usr/lib/python3/dist-packages (2.3.0)
Requirement already satisfied: cryptography in /usr/lib/python3/dist-packages (3.4.8)
jaime@jaime-VirtualBox:~/Escritorio/Security$
```

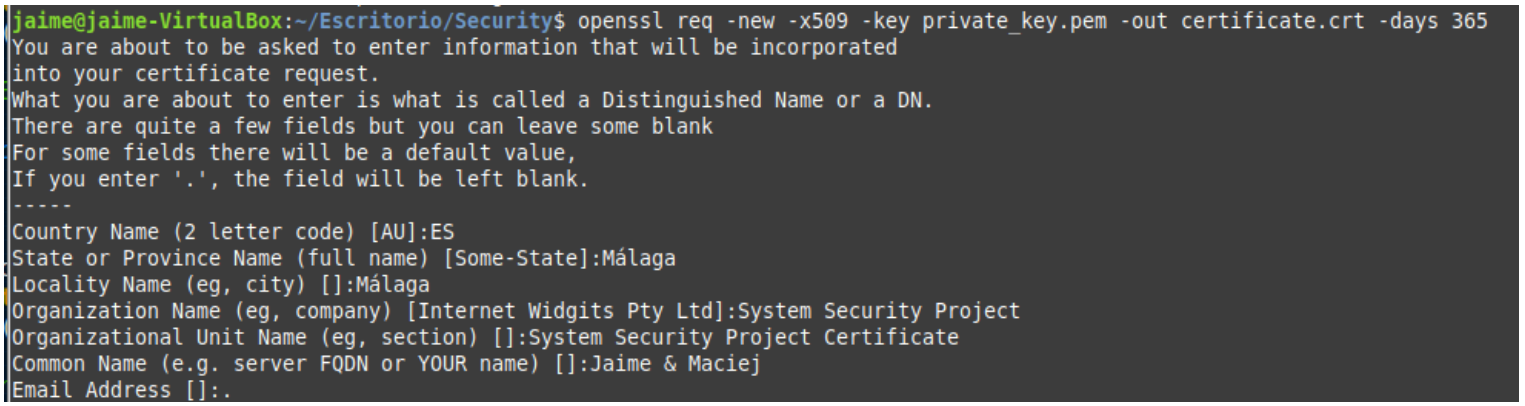
Commands needed in the cmd

- Generate the symmetric key



- Generate the private key
  2048 → key length in bits. Current industry standard

- Extracts the public key, adds the identity, and signs the certificate with their own private key.



Python codes

issuer.py

- This Python code simulates a server that issues the command

```
import jwtimport datetime

from cryptography.fernet import Fernet # Fernet does Symmetric


# Load Symmetric Key (Encryption)
with open("symmetric.key", "rb") as k:
      cipher_suite = Fernet(k.read())

# Load Private Key (Digital Signature)
try:
      with open("private_key.pem", "rb") as f:
            # rb means read binary
            private_key = f.read()
except FileNotFoundError:
            print("ERROR: private_key.pem not found. Run OpenSSL
commands first.")
            exit()

raw_command = "OPEN_MAIN_GATE"
encrypted_command = cipher_suite.encrypt(raw_command.encode()).decode()

# Define the Data Payload
data_payload = {
      "user": "sys_admin",
      "role": "superuser",
      "command": encrypted_command,
      "amount": 10000, # Random value, imagine this is dollars in a bank
      "timestamp": str(datetime.datetime.now()) # To prevent replay
attacks
}

print(f"--- ISSUER: Generating Token ---")
print(f"Original Data: {data_payload}")
print(f"Raw Command: {raw_command}")

# Sign the data and create the JWT (Authenticity & Integrity)
# Integrity - Signing - Encoding
signed_token = jwt.encode( # Create a hash of our data
      payload=data_payload,
```

```
      key=private_key,
      algorithm="RS256" # RSA Signature with SHA-256 for Asymmetric
Cryptography
)

print(f"Generated Token: {signed_token[:20]}... (truncated)") # :20 for
shortening

# Simulate Network Transmission
# Save token to file to simulate sending it
with open("token_transmission.txt", "w") as f:
      # In real life we would send it through an HTTP POST request
(TCP/IP packets)
      # and the hacker would sniff the packet with Proxy tools
      f.write(signed_token)

print("Status: Token sent to network.\n")
```

receiver.py

- This Python code simulates the receiving device. It only trusts data signed by the correct Private Key.

```
import jwt
from cryptography.fernet import Fernet # Fernet does Symmetric
from cryptography import x509
from cryptography.hazmat.backends import default_backend

# Load the Public Key generated by OpenSSL
# The receiver uses this key to verify the signature.
try:
      with open("certificate.crt", "rb") as f:
            cert_pem = f.read()
            # We use the x509 library to extract public key from
certificate
            certificate = x509.load_pem_x509_certificate(cert_pem,
default_backend())
            public_key = certificate.public_key()
except FileNotFoundError:
      print("Error: certificate.crt not found.")
      exit()
print("--- RECEIVER: Processing Data ---")

# Load the Symmetric Key
try:
      with open("symmetric.key", "rb") as k:
```

```
            cipher_suite = Fernet(k.read())
except FileNotFoundError:
      print("Error: symmetric.key not found.")
      exit()

print("---RECEIVER: Symmetric Key loaded---")

# Read the token from the "network"
try:
      with open("token_transmission.txt", "r") as f:
            received_token = f.read()
except FileNotFoundError:
      print("Error: No token found in the network.")
      exit()

print(f"Token Received: {received_token[:20]}...") # :20 again

# Verify the Signature and Decode
try:
      decoded_data = jwt.decode(
            jwt=received_token,
            key=public_key,
            algorithms=["RS256"] # In brackets cause there might be
multiple
      )

      print("\n[SUCCESS] INTEGRITY VERIFIED.")
      print("The signature is valid. Executing command:")

      encrypted_cmd = decoded_data['command']
      decrypted_cmd =
cipher_suite.decrypt(encrypted_cmd.encode()).decode()
      print(f"Decrypted Command: {decrypted_cmd}")
      print(f"User Authorized: {decoded_data['user']}")

except jwt.InvalidSignatureError:
      print("\n[CRITICAL ERROR] SIGNATURE MISMATCH!")
      print("Security Alert: The data has been tampered with (Fake Data
detected).")
      print("Action: Request Rejected.")

except Exception as e:
      print(f"\n[ERROR] Token invalid: {e}")
```

hacker.py

- This Python code simulates Man-in-the-middle attack. The attacker intercepts the token, modifies the data, and tries to forward it.

```
import jwt
import json
import base64

print("--- ATTACKER: Man-in-the-Middle Interception ---")

# Intercept the token from the transmission file
try:
      with open("token_transmission.txt", "r") as f:
            intercepted_token = f.read()
except FileNotFoundError:
      print("No token found to intercept.")
      exit()

# Decoding the JWT without a key
payload_only = jwt.decode(intercepted_token,
options={"verify_signature": False})
# verify=False because the hacker doesn't have the Public Key, so it
shows what's inside

print(f"Hacker sees this Payload: {payload_only}")
# The hacker sees: "command": "gAAAAABl..."

# Since they don't have symmetric.key, they don't know what it says
print(f"\nHacker can't read the command, but will try to change it
anyway")

# Injecting Fake Data
payload_only["user"] = "HACKER_ENTITY"
payload_only["command"] = "FAKE_ENCRYPTED_DATA"

print(f"Modified Payload: {payload_only}")

# Attempting to forge the token
# The hacker doesn't have 'private_key.pem', so they sign it with a fake
key
fake_key = "random-string"
forged_token = jwt.encode(payload_only, fake_key, algorithm="HS256") #
Wrong algorithm too

# Inject the forged token back into the network
with open("token_transmission.txt", "w") as f:
      f.write(forged_token)
```

```
print("\nStatus: Forged token injected.")
```

gen_key.py

- Symmetric Key generator

```
from cryptography.fernet import Fernet

# Generate the key
key = Fernet.generate_key()
with open("symmetric.key", "wb") as f:
    f.write(key)
print("Symmetric key generated.")
```

Demonstration

Case 1 → Legitimate Transaction

```
jaime@jaime-VirtualBox:~/Escritorio/Security$ python3 issuer.py
--- ISSUER: Generating Token ---
Original Data: {'user': 'sys_admin', 'role': 'superuser', 'command': 'gAAAAABphQNUKM4zc0AtIlQAZFM2cOJmtxtB5ZpBWLhnrhzDhITSauMHu2Bt365y1yDF9O4Sfu0CMQhWS801JF60qx
eErHW2sQ==', 'amount': 10000, 'timestamp': '2026-02-05 21:53:40.199261'}
Raw Command: OPEN_MAIN_GATE
Generated Token: eyJ0eXAiOiJKV1QiLCJh... (truncated)
Status: Token sent to network.

jaime@jaime-VirtualBox:~/Escritorio/Security$ python3 receiver.py
--- RECEIVER: Processing Data ---
---RECEIVER: Symmetric Key loaded---
Token Received: eyJ0eXAiOiJKV1QiLCJh...

[SUCCESS] INTEGRITY VERIFIED.
The signature is valid. Executing command:
Decrypted Command: OPEN_MAIN_GATE
User Authorized: sys_admin
```

Case 2 → Fake Data Attack

```
jaime@jaime-VirtualBox:~/Escritorio/Security$ python3 issuer.py
--- ISSUER: Generating Token ---
Original Data: {'user': 'sys_admin', 'role': 'superuser', 'command': 'gAAAAABphQJ89ihn9k9NKPtBjZdTB0LIz5113tI7H4JYr7iQKfBCaM-Bp0_pgOoHhHBORd38nfuAGItfH7_Lnt1BU4
1l4bRqlw==', 'amount': 10000, 'timestamp': '2026-02-05 21:50:04.090525'}
Raw Command: OPEN_MAIN_GATE
Generated Token: eyJ0eXAiOiJKV1QiLCJh... (truncated)
Status: Token sent to network.

jaime@jaime-VirtualBox:~/Escritorio/Security$ python3 hacker.py
--- ATTACKER: Man-in-the-Middle Interception ---
Hacker sees this Payload: {'user': 'sys_admin', 'role': 'superuser', 'command': 'gAAAAABphQJ89ihn9k9NKPtBjZdTB0LIz5113tI7H4JYr7iQKfBCaM-Bp0_pgOoHhHBORd38nfuAGIt
fH7_Lnt1BU41l4bRqlw==', 'amount': 10000, 'timestamp': '2026-02-05 21:50:04.090525'}

Hacker can't read the command, but will try to change it anyway
Modified Payload: {'user': 'HACKER_ENTITY', 'role': 'superuser', 'command': 'FAKE_ENCRYPTED_DATA', 'amount': 10000, 'timestamp': '2026-02-05 21:50:04.090525'}

Status: Forged token injected.
jaime@jaime-VirtualBox:~/Escritorio/Security$ python3 receiver.py
--- RECEIVER: Processing Data ---
---RECEIVER: Symmetric Key loaded---
Token Received: eyJ0eXAiOiJKV1QiLCJh...

[ERROR] Token invalid: The specified alg value is not allowed
```