Development Team Project: Suspect Sources System for the Dutch National Cyber Security Centre

Table of Contents

Project Introduction

This project serves as the Development Team Project: Coding Output assignment for the Secure Software Development Module of University of Essex Online - March 2021. The assignment focuses on the second deliverable: a practical development of a previously proposed design.

In total, the submitted code for the Development Team Project assignment aims to implement two major parts:

1. The NCSC Suspect Sources Core System and Database
2. Prototype of a Command-Line Interface to Access the System

The National Cyber Security Centre (NCSC) is the Netherlands' consolidated data hub and cyber security knowledge centre. NCSC's objective is to strengthen Dutch society's digital resilience, resulting in a better, broader, and stable digital world. The NCSC provides expert insight into cyber-security innovations, threats, and risks (Government of the Netherlands, N.D.).

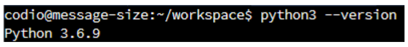
One of NCSC's main tasks is to continuously monitor all suspect sources on the internet and alert public authorities and organisations of new threats (Government of the Netherlands, N.D.).

The System that will be developed for the Dutch NCSC will allow authorised employees to search, amend and create entries for the suspect sources database. These can then be accessed and utilised by other staff members to assist in the work they do. It is assumed that the System will be used by three groups of users, each with its own set of access permissions and controls: Specialists, Administrators and External Authorities.

Setup

Prerequisites

The project has been coded in python 3.6.9, and it is recommended to use the same version when executing the code on your end.



```
codio@message-size:~/workspace$ python3 --version
Python 3.6.9
```

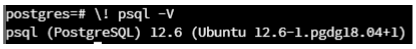
Also, before running the python project, you will need to install a few external libraries: * Argon2 v1.3 for password hashing via the "argon2-cffi" python lib * Fernet symmetric encryption to encrypt and decrypt database credentials and content via .bin file * Psycopg2 as a Python Connector to the PostgreSQL database * Stdiomask, validator_collection and secrets for the interface inputs and validations

You can use the following pip commands to install the libraries directly or follow the requirements.txt file:

```
 pip3 install argon2-cffi
pip3 install cryptography
pip3 install stdiomask
pip3 install psycopg2
pip3 install validator_collection
pip3 install secrets
```

Database Setup

For the System's database, PostgreSQL v12.6 was used. We have added a "setup" folder that includes setup scripts for all of the database tables and some sample data for Users and Sources.



```
postgres=# \! psql -V
psql (PostgreSQL) 12.6 (Ubuntu 12.6-1.pgdg18.04+1)
```

For the default users, you can use the following username/password combinations to authenticate once added to the DB: * Role Administrator: `s.leavitt` / `ZTBJs36*eyzh^msx` * Role Specialist: `r.newton` / `F44aF#rhVgs9hAH8` * Role External Authority: `r.crow` / `3S2PkFEn*#twx*g4`

The Administrator account can then be used to create new users in the System to test all use cases and scenarios.

Database Credential Encryption

To ensure that the CLI can connect to your database, you will need to set up a DB access and include the corresponding encrypted credentials in the "config" folder. The encryption can be done in the following way: 1. Generate database user with sufficient privileges (exact privileges will be explained in the chapter "Security") 2. Format the credentials in the following format: `{database_host}:{default_database}:{username}:{password}` (for reference see the "psql_clear_credentials.txt" file in the 'setup' folder) 3. Generate an own Fernet master key (via "fernet_key_gen.py" in "setup") or use the one currently saved in the "config" folder ("key.bin" in "config") 4. Run the cleartext file through Fernet encryption using the script "credential_encryption.py" in the "setup" folder and save the output as a binary file 5. Rename the output to "credentials.bin" and move it into the "config" folder 6. All done! The CLI should now be able to connect to the DB using your "credentials.bin" file

File Structure

Folder Structure

The project includes the main Python modules and requirements file inside the root folder. In addition, there are three subfolders: 'config', 'setup' and 'policies'.

Config: Config includes three binary files used for the code execution and one HTML file used to send formatted email notifications to users. In an actual deployment, this folder would need to enforce strict access controls to protect the sensitive files inside of it: 1. **banner.bin**: file that contains the initial lines to be printed when the interface is executed. 2. **credentials.bin**: file that includes the encrypted PostgreSQL credentials (encrypted via Fernet). 3. **key.bin**: file that contains the key for the Fernet encryption and which is used to decrypt the credentials when the code is executed. 4. **email_body.html**: file contains the email template used to communicate to the users.

Setup: Setup includes files to understand the initial setup of the project. In an actual deployment, this folder would not exist. In total, the folder includes three files: 1. **credential_encryption.py**: the python script used to encrypt the clear credentials initially and outputs the credentials.bin file as seen in the 'config' folder. 2. **fernet_key_gen.py**: the python script used to generate a new fernet master key and outputs the key.bin file as seen in the 'config' folder. 3. **psql_clear_credentials.txt**: displays the PostgreSQL credentials prior to being encrypted (format: "host:database:user:password"). 4. **sql_table_creation**: includes all of the SQL setup scripts for the database tables and some sample data for the users and sources table.

Policies: Policies includes files that demonstrate the System's compliance to the law and the agreement terms for user to access the System. In total, the folder contains two files: 1. **privacy_policy.bin**: file that includes the System's privacy policy statements to comply with legal obligations. 2. **terms_conditions.bin**: file containing the sets of rules and guidelines that users must agree to and follow to use and access the System.

Python Modules

In total, the Suspect Sources system project includes a total of eight python modules: 1. **main.py**: executes the NCSC Suspect Sources System prototype interface. 2. **interface.py**: main user interface contained as a class that provides handler functions to call the other modules and prompts dialogues and user inputs. 3. **dbconnection.py**: module that decrypts the PostgreSQL credentials and establishes the primary connection to the DB. 4. **authentication.py**: module that handles the login operations, as well as the password hashing functionality. 5. **operations.py**: module that holds the main portions of the specialist and authority user role operations. Includes source creation, search, modification, as well as the change password functionality. 6. **admin_operations.py**: module that handles the main portion of the administration user role operations. Includes user creation, modification, deactivation as well as the unlock functionality. 7. **notification.py**: module that handles the email notification templates and content. It is called by the other modules whenever a notification needs to be triggered. 8. **eventlog.py**: module that handles the log creation of the System to ensure traceability and accountability for users and their actions.

For the execution of the Command-Line Interface, please run either `python main.py` or `python3 main.py` (depending on your Python installation).

Database Structure

The project uses three main databases for the implementation: 1. **Authentication**: contains the user information for authentication. 2. **Data**: contains the suspect source information. 3. **Eventlog**: contains the system logs and traces of all user actions.

Authentication

The Authentication database contains the 'users' table.

When registering a new user, a new data row is inserted into the 'users' table containing the user's id, first_name, last_name, dob, user_role, username, password as an Argon2 hash, last_login, status and email.

Once a user is registered, an automatic email is sent to the user containing the desired username and the hashed password. When logging into the Suspect Sources system for the first time, the user will be prompted to change his/her password, which will then be hashed and saved/updated into the 'users' table to enhance the security of the account.

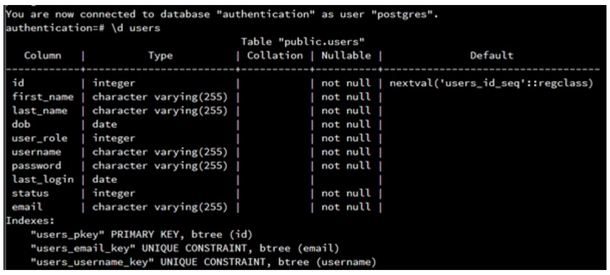
When logging into an already existing account, the user needs to input a username and password. Once done, the entered username is compared against all entries in the table. If it exists, the entered password is hashed, and the hash itself is compared against the saved entry. If the hash and the username line up with the existing data, the user is successfully authenticated in the System and can continue.

The user_role attribute denotes the role for that specific user. In total, this attribute can have one of three values:

```
 1 = Administrator
2 = Specialist
3 = External Authority
```

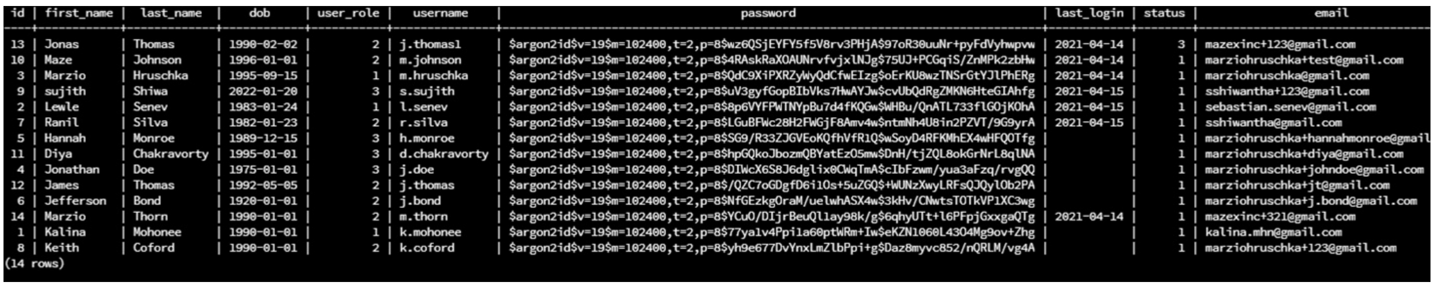
The status attribute denotes the current status of that specific user. In total, this attribute can have one of three values:

```
 1 = Active
2 = Deactivated (soft-deleted)
3 = Locked (due to failed authentication attempts)
```



```
You are now connected to database "authentication" as user "postgres".
authentication=# \d users
                                 Table "public.users"
   Column   |          Type          | Collation | Nullable |               Default
------------+------------------------+-----------+----------+-------------------------------------
 id         | integer                |           | not null | nextval('users_id_seq'::regclass)
 first_name | character varying(255) |           | not null |
 last_name  | character varying(255) |           | not null |
 dob        | date                   |           | not null |
 user_role  | integer                |           | not null |
 username   | character varying(255) |           | not null |
 password   | character varying(255) |           | not null |
 last_login | date                   |           |          |
 status     | integer                |           | not null |
 email      | character varying(255) |           | not null |
Indexes:
    "users_pkey" PRIMARY KEY, btree (id)
    "users_email_key" UNIQUE CONSTRAINT, btree (email)
    "users_username_key" UNIQUE CONSTRAINT, btree (username)
```

Figure 1: 'authentication.users' Columns and Attribute Types



```
 id | first_name |  last_name  |    dob     | user_role |    username     |                                      password                                       | last_login | status |                  email
----+------------+-------------+------------+-----------+-----------------+-------------------------------------------------------------------------------------+------------+--------+------------------------------------------
 13 | Jonas      | Thomas      | 1990-02-02 |         2 | j.thomas1       | $argon2id$v=19$m=102400,t=2,p=8$wz6QSjEYFY5f5V8rv3PHjA$97oR30uuNr+pyFdVyhwpvw | 2021-04-14 |      3 | mazexinc+123@gmail.com
 10 | Maze       | Johnson     | 1996-01-01 |         2 | m.johnson       | $argon2id$v=19$m=102400,t=2,p=8$4RAskRaXOAUNrvfvjxlNJg$75UJ+PCGqiS/ZnMPk2zbHw | 2021-04-14 |      1 | marziohruschka+test@gmail.com
  3 | Marzio     | Hruschka    | 1995-09-15 |         1 | m.hruschka      | $argon2id$v=19$m=102400,t=2,p=8$QdC9XiPXRZyWyQdCfwEIzg$oErKU8wzTNSrGtYJlPhERg | 2021-04-14 |      1 | marziohruschka@gmail.com
  9 | sujith     | Shiwa       | 2022-01-20 |         3 | s.sujith        | $argon2id$v=19$m=102400,t=2,p=8$uV3gyfGopBIbVks7HwAYJw$cvUbQdRgZMKN6HteGIAhfg | 2021-04-15 |      1 | sshiwantha+123@gmail.com
  2 | Lewle      | Senev       | 1983-01-24 |         1 | l.senev         | $argon2id$v=19$m=102400,t=2,p=8$8p6VYFPWTNYpBu7d4fKQGw$WHBu/QnATL733flGOjKOhA | 2021-04-15 |      1 | sebastian.senev@gmail.com
  7 | Ranil      | Silva       | 1982-01-23 |         2 | r.silva         | $argon2id$v=19$m=102400,t=2,p=8$LGuBFWc28H2FWGjF8Amv4w$ntmNh4U8in2PZVT/9G9yrA | 2021-04-15 |      1 | sshiwantha@gmail.com
  5 | Hannah     | Monroe      | 1989-12-15 |         3 | h.monroe        | $argon2id$v=19$m=102400,t=2,p=8$SG9/R33ZJGVEoKQfhVfR1Q$wSoyD4RFKMhEX4wHFQOTfg |            |      1 | marziohruschka+hannahmonroe@gmail
 11 | Diya       | Chakravorty | 1995-01-01 |         3 | d.chakravorty   | $argon2id$v=19$m=102400,t=2,p=8$hpGQkoJbozmQBYatEzO5mw$DnH/tjZQL8okGrNrL8qlNA |            |      1 | marziohruschka+diya@gmail.com
  4 | Jonathan   | Doe         | 1975-01-01 |         3 | j.doe           | $argon2id$v=19$m=102400,t=2,p=8$DIWcX6S8J6dglix0CWqTmA$cIbFzwm/yua3aFzq/rvgQQ |            |      1 | marziohruschka+johndoe@gmail.com
 12 | James      | Thomas      | 1992-05-05 |         2 | j.thomas        | $argon2id$v=19$m=102400,t=2,p=8$/QZC7oGDgfD6i1Os+5uZGQ$+WUNzXwyLRFsQJQylOb2PA |            |      1 | marziohruschka+jt@gmail.com
  6 | Jefferson  | Bond        | 1920-01-01 |         2 | j.bond          | $argon2id$v=19$m=102400,t=2,p=8$NfGEzkgOraM/uelwhASX4w$3kHv/CNwtsTOTkVP1XC3wg |            |      1 | marziohruschka+j.bond@gmail.com
 14 | Marzio     | Thorn       | 1990-01-01 |         2 | m.thorn         | $argon2id$v=19$m=102400,t=2,p=8$YCuO/DIjrBeuQl1ay98k/g$6qhyUTt+l6PFpjGxxgaQTg | 2021-04-14 |      1 | mazexinc+321@gmail.com
  1 | Kalina     | Mohonee     | 1990-01-01 |         1 | k.mohonee       | $argon2id$v=19$m=102400,t=2,p=8$77ya1v4Ppi1a60ptWRm+Iw$eKZN1060L43O4Mg9ov+Zhg |            |      1 | kalina.mhn@gmail.com
  8 | Keith      | Coford      | 1990-01-01 |         2 | k.coford        | $argon2id$v=19$m=102400,t=2,p=8$yh9e677DvYnxLmZlbPpi+g$Daz8myvc852/nQRLM/vg4A |            |      1 | marziohruschka+123@gmail.com
(14 rows)
```

Figure 2: 'authentication.users' Sample Data

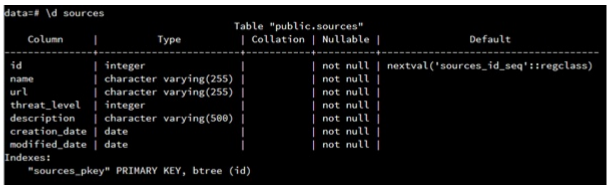
Data

The 'sources' table contains the details of suspect sources inside of the System. New entries can be created by the Specialist user role from within the CLI.

When creating a new source, a new data row is inserted into the 'sources' table containing the source's id, name, url, threat_level, description, creation_date and modified_date.

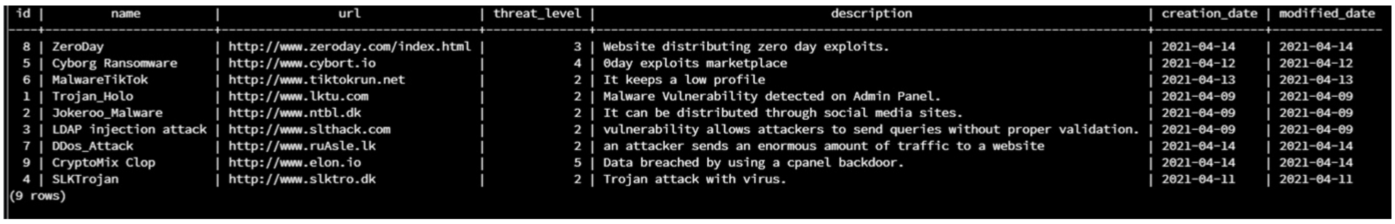
If a new source is added to the System, an email notification is triggered to all users in the System with the role of an external authority.

When searching for sources in the System, the entered data is queried against the appropriate attributes of the 'sources' table, and the details are displayed back to the user. If a user actions a modification, the applicable data row is updated on this database.



```
data=# \d sources
                                    Table "public.sources"
    Column     |          Type          | Collation | Nullable |               Default
---------------+------------------------+-----------+----------+--------------------------------------
 id            | integer                |           | not null | nextval('sources_id_seq'::regclass)
 name          | character varying(255) |           | not null |
 url           | character varying(255) |           | not null |
 threat_level  | integer                |           | not null |
 description   | character varying(500) |           | not null |
 creation_date | date                   |           | not null |
 modified_date | date                   |           | not null |
Indexes:
    "sources_pkey" PRIMARY KEY, btree (id)
```

Figure 3: 'data.sources' Columns and Attribute Types



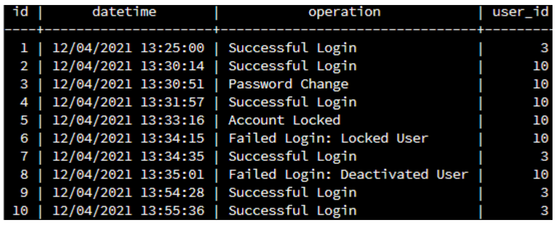
```
 id |         name          |                url                 | threat_level |                              description                                | creation_date | modified_date
----+-----------------------+------------------------------------+--------------+-------------------------------------------------------------------------+---------------+---------------
  8 | ZeroDay               | http://www.zeroday.com/index.html  |            3 | Website distributing zero day exploits.                                 | 2021-04-14    | 2021-04-14
  5 | Cyborg Ransomware     | http://www.cybort.io               |            4 | 0day exploits marketplace                                               | 2021-04-12    | 2021-04-12
  6 | MalwareTikTok         | http://www.tiktokrun.net           |            2 | It keeps a low profile                                                  | 2021-04-13    | 2021-04-13
  1 | Trojan_Holo           | http://www.lktu.com                |            2 | Malware Vulnerability detected on Admin Panel.                          | 2021-04-09    | 2021-04-09
  2 | Jokeroo_Malware       | http://www.ntbl.dk                 |            2 | It can be distributed through social media sites.                       | 2021-04-09    | 2021-04-09
  3 | LDAP injection attack | http://www.slthack.com             |            2 | vulnerability allows attackers to send queries without proper validation. | 2021-04-09    | 2021-04-09
  7 | DDos_Attack           | http://www.ruAsle.lk               |            2 | an attacker sends an enormous amount of traffic to a website            | 2021-04-14    | 2021-04-14
  9 | CryptoMix Clop        | http://www.elon.io                 |            5 | Data breached by using a cpanel backdoor.                               | 2021-04-14    | 2021-04-14
  4 | SLKTrojan             | http://www.slktro.dk               |            2 | Trojan attack with virus.                                               | 2021-04-11    | 2021-04-11
(9 rows)
```

Figure 4: 'data.sources' Sample Data

Eventlog

The eventlog database serves as the log database that stores all traces of user actions on the System. The three main tables of the eventlog are: 'authlogs', 'operationlogs' and 'adminlogs'.

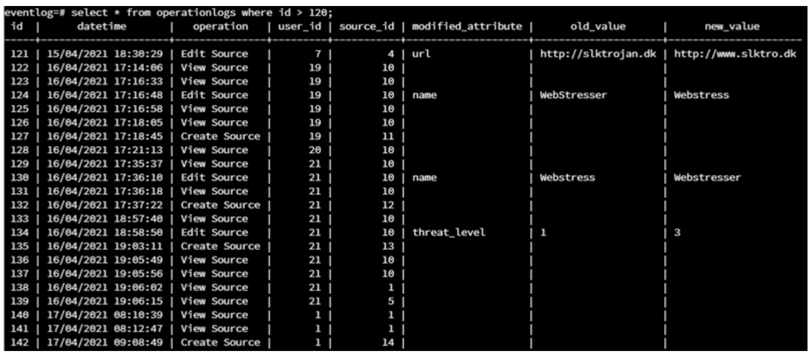
The table 'authlogs' stores logs for authentication actions of existing accounts, such as successful login, change of password, locked account due to failed attempts, and unsuccessful login attempts for users with the status "deactivated" and "locked". For all actions, the data includes the date and time when the action was conducted, what operation it is (successful login, modified password, locked account or unsuccessful login) and who actioned it (user id).



```
 id |      datetime       |             operation             | user_id
----+---------------------+-----------------------------------+---------
  1 | 12/04/2021 13:25:00 | Successful Login                  |       3
  2 | 12/04/2021 13:30:14 | Successful Login                  |      10
  3 | 12/04/2021 13:30:51 | Password Change                   |      10
  4 | 12/04/2021 13:31:57 | Successful Login                  |      10
  5 | 12/04/2021 13:33:16 | Account Locked                    |      10
  6 | 12/04/2021 13:34:15 | Failed Login: Locked User         |      10
  7 | 12/04/2021 13:34:35 | Successful Login                  |       3
  8 | 12/04/2021 13:35:01 | Failed Login: Deactivated User    |      10
  9 | 12/04/2021 13:54:28 | Successful Login                  |       3
 10 | 12/04/2021 13:55:36 | Successful Login                  |       3
```

Figure 5: 'eventlog.authlogs' Sample Data

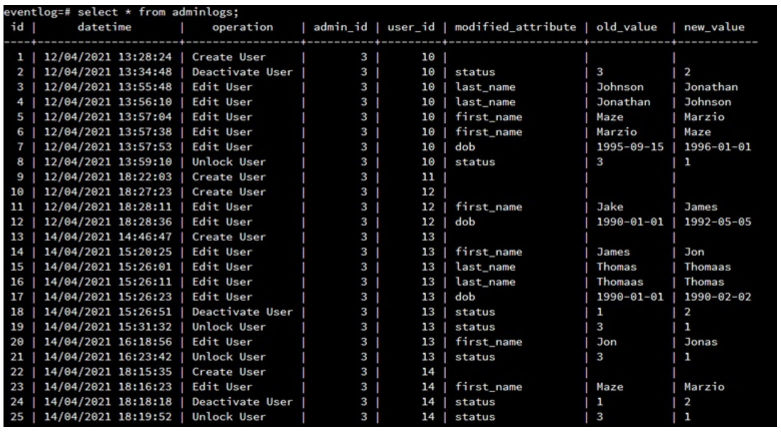
The table 'operationlogs' stores all traces for actions on the source data. For all events the data includes when the action was conducted, what action it is (creation/modification/view), what source was effected (source id) and who actioned it (user id). For modify actions, it includes the record that was modified, the old value before the change and the new value after the change.



```
eventlog=# select * from operationlogs where id > 120;
 id  |      datetime       |   operation    | user_id | source_id | modified_attribute |      old_value       |      new_value
-----+---------------------+----------------+---------+-----------+--------------------+----------------------+----------------------
 121 | 15/04/2021 18:30:29 | Edit Source    |       7 |         4 | url                | http://slktrojan.dk  | http://www.slktro.dk
 122 | 16/04/2021 17:14:06 | View Source    |      19 |        10 |                    |                      |
 123 | 16/04/2021 17:16:33 | View Source    |      19 |        10 |                    |                      |
 124 | 16/04/2021 17:16:48 | Edit Source    |      19 |        10 | name               | WebStresser          | Webstress
 125 | 16/04/2021 17:16:58 | View Source    |      19 |        10 |                    |                      |
 126 | 16/04/2021 17:18:05 | View Source    |      19 |        10 |                    |                      |
 127 | 16/04/2021 17:18:45 | Create Source  |      19 |        11 |                    |                      |
 128 | 16/04/2021 17:21:13 | View Source    |      20 |        10 |                    |                      |
 129 | 16/04/2021 17:35:37 | View Source    |      21 |        10 |                    |                      |
 130 | 16/04/2021 17:36:10 | Edit Source    |      21 |        10 | name               | Webstress            | Webstresser
 131 | 16/04/2021 17:36:18 | View Source    |      21 |        10 |                    |                      |
 132 | 16/04/2021 17:37:22 | Create Source  |      21 |        12 |                    |                      |
 133 | 16/04/2021 18:57:40 | View Source    |      21 |        10 |                    |                      |
 134 | 16/04/2021 18:58:50 | Edit Source    |      21 |        10 | threat_level       | 1                    | 3
 135 | 16/04/2021 19:03:11 | Create Source  |      21 |        13 |                    |                      |
 136 | 16/04/2021 19:05:49 | View Source    |      21 |        10 |                    |                      |
 137 | 16/04/2021 19:05:56 | View Source    |      21 |        10 |                    |                      |
 138 | 16/04/2021 19:06:02 | View Source    |      21 |         1 |                    |                      |
 139 | 16/04/2021 19:06:15 | View Source    |      21 |         5 |                    |                      |
 140 | 17/04/2021 08:10:39 | View Source    |       1 |         1 |                    |                      |
 141 | 17/04/2021 08:12:47 | View Source    |       1 |         1 |                    |                      |
 142 | 17/04/2021 09:08:49 | Create Source  |       1 |        14 |                    |                      |
```

Figure 6: 'eventlog.operationlogs' Sample Data

The table 'adminlogs' stores all traces of administrator actions on user data. For all events, the data includes when the action was conducted, what action it is (creation/modification/unlock/deactivate), what user is affected (user id), and who actioned it (admin id). For modify actions, the logs include the attribute that was modified, the old value before the change and the new value after the change.



```
eventlog=# select * from adminlogs;
 id |      datetime       |    operation    | admin_id | user_id | modified_attribute | old_value  | new_value
----+---------------------+-----------------+----------+---------+--------------------+------------+-----------
  1 | 12/04/2021 13:28:24 | Create User     |        3 |      10 |                    |            |
  2 | 12/04/2021 13:34:48 | Deactivate User |        3 |      10 | status             | 3          | 2
  3 | 12/04/2021 13:55:48 | Edit User       |        3 |      10 | last_name          | Johnson    | Jonathan
  4 | 12/04/2021 13:56:10 | Edit User       |        3 |      10 | last_name          | Jonathan   | Johnson
  5 | 12/04/2021 13:57:04 | Edit User       |        3 |      10 | first_name         | Maze       | Marzio
  6 | 12/04/2021 13:57:38 | Edit User       |        3 |      10 | first_name         | Marzio     | Maze
  7 | 12/04/2021 13:57:53 | Edit User       |        3 |      10 | dob                | 1995-09-15 | 1996-01-01
  8 | 12/04/2021 13:59:10 | Unlock User     |        3 |      10 | status             | 3          | 1
  9 | 12/04/2021 18:22:03 | Create User     |        3 |      11 |                    |            |
 10 | 12/04/2021 18:27:23 | Create User     |        3 |      12 |                    |            |
 11 | 12/04/2021 18:28:11 | Edit User       |        3 |      12 | first_name         | Jake       | James
 12 | 12/04/2021 18:28:36 | Edit User       |        3 |      12 | dob                | 1990-01-01 | 1992-05-05
 13 | 14/04/2021 14:46:47 | Create User     |        3 |      13 |                    |            |
 14 | 14/04/2021 15:20:25 | Edit User       |        3 |      13 | first_name         | James      | Jon
 15 | 14/04/2021 15:26:01 | Edit User       |        3 |      13 | last_name          | Thomas     | Thomaas
 16 | 14/04/2021 15:26:11 | Edit User       |        3 |      13 | last_name          | Thomaas    | Thomas
 17 | 14/04/2021 15:26:23 | Edit User       |        3 |      13 | dob                | 1990-01-01 | 1990-02-02
 18 | 14/04/2021 15:26:51 | Deactivate User |        3 |      13 | status             | 1          | 2
 19 | 14/04/2021 15:31:32 | Unlock User     |        3 |      13 | status             | 3          | 1
 20 | 14/04/2021 16:18:56 | Edit User       |        3 |      13 | first_name         | Jon        | Jonas
 21 | 14/04/2021 16:23:42 | Unlock User     |        3 |      13 | status             | 3          | 1
 22 | 14/04/2021 18:15:35 | Create User     |        3 |      14 |                    |            |
 23 | 14/04/2021 18:16:23 | Edit User       |        3 |      14 | first_name         | Maze       | Marzio
 24 | 14/04/2021 18:18:18 | Deactivate User |        3 |      14 | status             | 1          | 2
 25 | 14/04/2021 18:19:52 | Unlock User     |        3 |      14 | status             | 3          | 1
```

Figure 7: 'eventlog.adminlogs' Sample Data

User Roles and Use Cases

As highlighted in the chapter "Background", it is assumed that the System will be used by three groups of users, each with its own set of access permissions and controls. Each user role has a different CLI view and operations he/she can conduct.

Administrators can: * Create New Users * Modify Existing Users * Deactivate Existing Users (soft delete) * Unlock Existing Users (if locked due to failed login attempts)

Specialists can: * Search for Existing Suspect Sources * Modify Existing Suspect Sources * Create New Suspect Sources * Change their Password

External Authorities can: * Search for Existing Suspect Sources * Change their Password

The "Administrator" role is responsible for User Management. Administrators can create new users, as well as modify, deactivate and unlock existing users. When creating a new user, the administrator will need to input the user's first and last name, the date of birth, the email address, and the user's role (administrator, specialist, authority). The System then automatically generates a username (format: first letter of first name + `.` + full last name + running number if existent already) and a 12-character password including special characters, letters and numbers. The login information is then sent to the user's email address directly.

The "Specialist" role is reserved for internal employees of the NCSC. Specialists are responsible for the data maintenance of the Suspect Sources System. The role can create new source data, search for existing data, and modify the data. When creating a new Suspect Source record in the System, an email notification is triggered to all External Authority users in the System to notify them of a newly identified source. The Authorities can then login to view details and a description of the added threat.

The "Authority" role is reserved for external entities that need access to the Suspect Sources system. These would include public authorities and organisations that need to be kept informed about any new threats that are identified. Authorities are limited to only being able to view existing sources on the database. The user role does not have rights to modify existing data or create new records.

Security

Cryptography

Within the System, two types of encryptions are being utilised: * **Fernet's** symmetric encryption for encrypting and decrypting database credentials, as well as SMTP credentials * **Argon2** for password hashing

Fernet enables encryption of files and messages so that they cannot be manipulated or read without access to the master key. Fernet is an implementation of symmetric (also known as "secret key") authenticated cryptography. Fernet is built on top of several standard cryptographic primitives. Specifically, it uses AES with a 128-bit key for encryption and HMAC using SHA256 for authentication (Gaynor, 2020). Within the System, Fernet is used to encrypt and decrypt the database and SMTP credentials within the "config" folder. Furthermore, it provides the possibility of encoding data within the database if required in the full implementation.

For the System's password hashing, Argon2 is utilised. Argon2 is a password-hashing function that summarises the state of the art design of memory-hard functions. The algorithm has a modern ASIC-resistant and GPU-resistant secure key derivation function. Thus, if configured correctly, it provides a higher degree of cracking resistance than other algorithms, such as PBKDF2, Bcrypt and Scrypt (Nakov, 2018).

SQL Injection

SQL injection attacks are a big point for security concerns and are still regarded as the most critical web application security risk according to the OWASP Top Ten list (OWASP, 2020). The same is valid for this python project as well. The CLI is asking for user inputs, which can be manipulated to execute such injections if no security measures are taken.

All the malicious user would have to do, for instance, is to escape the actual username select query and insert an attack in its place, e.g.:

The statement below passes the username from the client directly to the database without performing any check or validation. Doing so opens the System up to be exploited.

```
 # BAD EXAMPLES. DON'T DO THIS!
cursor.execute("SELECT status FROM users WHERE username = " + username + ";")
```

The above statements should be converted to the below statement where the username is passed as a named parameter. Now, the database will use the specified type and value of username when executing the query, offering protection from Python SQL injection.

```
 # SAFE EXAMPLES. DO THIS!
cursor.execute("SELECT status FROM users WHERE username = %(val)s;", {'val': username})
```

Database Security

For the PostgreSQL database insert, update, and read operations, a database user is required. For this project, the user 'client' was created, which is used for all interface operations. Before executing any operations, the python code needs to authenticate with the database. Instead of adding the PostgreSQL username and password into the code as plain text, we have opted to encrypt the credentials and store them as a binary file in the 'config' folder for security purposes (using Fernet to encrypt). The encryption script and clear text credentials prior to encryption can be viewed in the 'setup' folder. When running the project, the 'dbconnection' module will attempt to decrypt the credentials binary file using the 'credentials.bin' file stored in the same 'config' folder. Only if that is successful, the interface will operate (register, login, create, modify, view). In an actual deployment, you would limit the 'config' folder access controls so that only the code can access the folder and read the 'credentials.bin' and 'key.bin' files, not the user himself. This would make it difficult for the user to obtain the actual credentials and connect to the database directly.

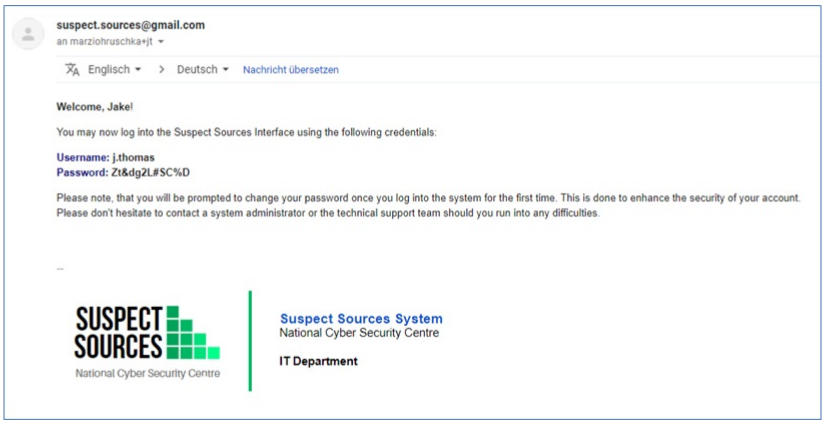
Principle of Least Privilege

Another layer of security is provided by the application of the least privilege principle. The user 'client' is only granted privileges that are vital for the interface operations. These are mainly select (for login and view), insert (for registration and create) and update (for modify) rights for the specific data tables. This ensures that in case of a breach or SQL injection attack using this user, the data that may be affected is limited. A complete list of granted privileges for the user can be seen here:

```
 GRANT SELECT, INSERT, UPDATE ON users TO client;
GRANT SELECT, USAGE, UPDATE ON users_id_seq TO client;
GRANT SELECT, INSERT, UPDATE ON sources TO client;
GRANT SELECT, USAGE, UPDATE ON sources_id_seq TO client;
GRANT INSERT ON authlogs, adminlogs, operationlogs TO client;
GRANT SELECT, USAGE ON adminlogs_id_seq, authlogs_id_seq, operationlogs_id_seq TO client;
```

Email Notification

Throughout the System, we are using three email notifications to be sent out to the users. The emails are sent via an external SMTP (Gmail in the case of this implementation) and can be configured in the "notification.py" module. The three types of notifications that are sent out are: - A ***Registration Email*** that is sent to new users when an Administrator signs them up and contains the auto-generated username and password - A ***Source Creation Notification*** that is sent to all Authority Users (User Role = 3) whenever a Specialist creates a new source - A ***Changed Password Confirmation*** whenever a password of a user is changed from within the CLI



suspect.sources@gmail.com
an marziohruschka+jt

Englisch > Deutsch Nachricht übersetzen

Welcome, Jake!

You may now log into the Suspect Sources Interface using the following credentials:

Username: j.thomas
Password: Zt&dg2L#SC%D

Please note, that you will be prompted to change your password once you log into the system for the first time. This is done to enhance the security of your account. Please don't hesitate to contact a system administrator or the technical support team should you run into any difficulties.

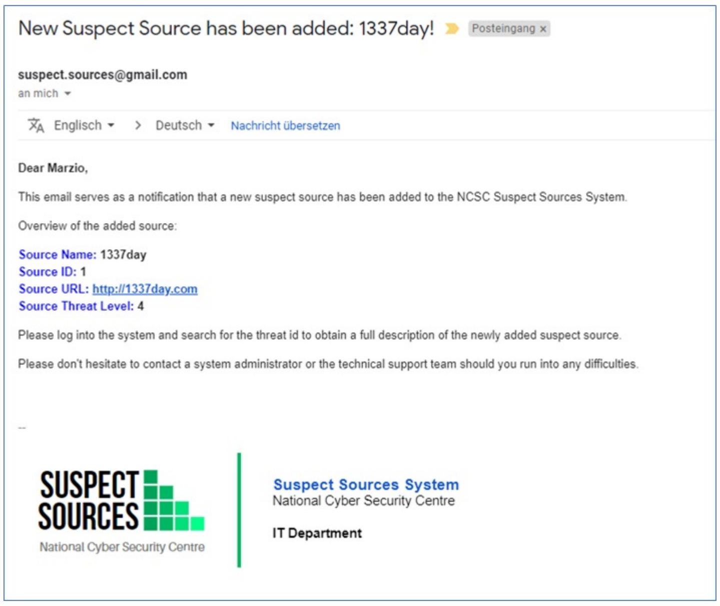
--

SUSPECT SOURCES
National Cyber Security Centre

Suspect Sources System
National Cyber Security Centre

IT Department

Figure 8: Registration Email



New Suspect Source has been added: 1337day! Posteingang

suspect.sources@gmail.com
an mich

Englisch > Deutsch Nachricht übersetzen

Dear Marzio,

This email serves as a notification that a new suspect source has been added to the NCSC Suspect Sources System.

Overview of the added source:

Source Name: 1337day
Source ID: 1
Source URL: http://1337day.com
Source Threat Level: 4

Please log into the system and search for the threat id to obtain a full description of the newly added suspect source.

Please don't hesitate to contact a system administrator or the technical support team should you run into any difficulties.

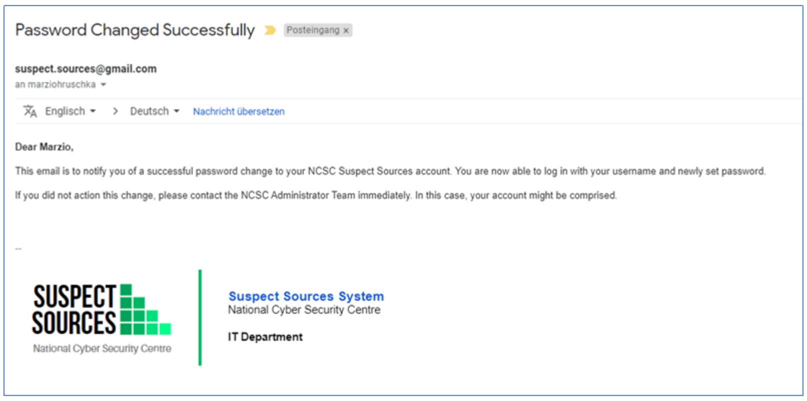
--

SUSPECT SOURCES
National Cyber Security Centre

Suspect Sources System
National Cyber Security Centre

IT Department

Figure 9: Suspect Source Notification



Password Changed Successfully Posteingang

suspect.sources@gmail.com
an marziohruschka

Englisch > Deutsch Nachricht übersetzen

Dear Marzio,

This email is to notify you of a successful password change to your NCSC Suspect Sources account. You are now able to log in with your username and newly set password.

If you did not action this change, please contact the NCSC Administrator Team immediately. In this case, your account might be comprised.

--

SUSPECT SOURCES
National Cyber Security Centre

Suspect Sources System
National Cyber Security Centre

IT Department

Figure 10: Changed Password Confirmation

Testing

There are three levels of testing employed within the System: - **Unit Testing** - **Continuous Review** - **Functional Testing**

Unit testing is a type of software testing where units or components of the software are tested individually. In the Suspect Sources system, this would mean the outputs of the various functions are tested to confirm that they fulfill expectations in terms of typing and values. Ideally, for the complete system implementation, a framework such as "unittest" or "Robot" could be used to create automated test cases that verify the outputs on-demand. For instance, whenever a major code update is pushed, the unit tests could then be run before merging the code to confirm that previous functions are not affected by the update.

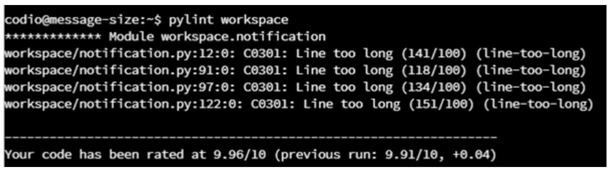
In the case of the prototype implementation, we have instead opted to go with print-based unit tests due to the tight timeline. While developing and adjusting code, print statements were used to verify the outputs of functions and ensure that the individual units of the System are working as expected.

Continuous Source Code review is done to ensure the structure, style, complexity, syntax, and security aspect of the source code as a whole is guaranteed. This can be done with the help of static code analysis tools called Linters. For this project, we decided to go with two different Linters to cover all bases:

Pylint for checking the syntax, complexity, styling and referencing of the code basis.

Bandit for checking security due to its capability of flagging vulnerabilities and their severeness.

By following the Pylint feedback, we raised the final score of the code basis to 9.96/10. The only part that couldn't conform to the standards was the notification module. Some features of the notification module included the HTML styling of the notification emails, which exceeded the line limit of 100 characters.

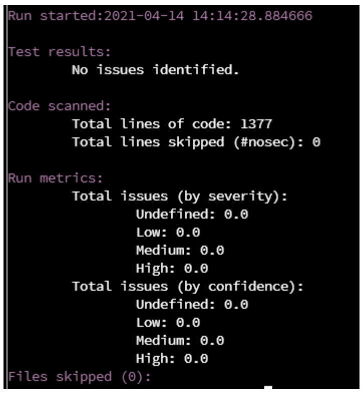


```
codio@message-size:~$ pylint workspace
************* Module workspace.notification
workspace/notification.py:12:0: C0301: Line too long (141/100) (line-too-long)
workspace/notification.py:91:0: C0301: Line too long (118/100) (line-too-long)
workspace/notification.py:97:0: C0301: Line too long (134/100) (line-too-long)
workspace/notification.py:122:0: C0301: Line too long (151/100) (line-too-long)

------------------------------------------------------------------
Your code has been rated at 9.96/10 (previous run: 9.91/10, +0.04)
```

Figure 11: Pylint Final Score for Source Code

Bandit initially flagged a remaining possible SQL injection vector that could have been exploited if overlooked. We ensured that all users inputs were escaped before being passed to the SQL query; however, at one point, the search attribute in a query was still created via a string-based construction. After adjusting this part of the code, the next Bandit run confirmed that our code basis was now issue-free.



```
Run started:2021-04-14 14:14:28.884666

Test results:
        No issues identified.

Code scanned:
        Total lines of code: 1377
        Total lines skipped (#nosec): 0

Run metrics:
        Total issues (by severity):
                Undefined: 0.0
                Low: 0.0
                Medium: 0.0
                High: 0.0
        Total issues (by confidence):
                Undefined: 0.0
                Low: 0.0
                Medium: 0.0
                High: 0.0
Files skipped (0):
```

Figure 12: Bandit's Final Run with No Issues Found.

The last type of testing was done before submitting the final project. Functional Testing ensures that all end-to-end use cases and user scenarios that will be done via the System are included and working. First, we established a Functional Test Plan containing a full checklist of test cases for the individual roles and the System in general. We then proceeded with manually testing each of the test scenarios to ensure that the System as a whole is working as per the use-case specifications. The complete Functional Test Plan can be found below. Doing so gave certainty that all use cases the System set out to cater towards were fulfilled.

The full end-to-end Functional Test Plan can be found here: Functional Test Plan

Reference List

Gaynor, A. (2020) Fernet Documentation. Available from: https://cryptography.io/en/latest/fernet/ [Accessed 17 April 2021].

Government of the Netherlands (N.D.) Fighting Cybercrime in the Netherlands. Government of the Netherlands. Available from: https://www.government.nl/topics/cybercrime/fighting-cybercrime-in-the-netherlands [Accessed 17 April 2021].

Nakov, S. (2018) Practical Cryptography for Developers. Available from: https://cryptobook.nakov.com/mac-and-key-derivation/argon2 [Accessed 17 April 2021].

OWASP (2020) OWASP Top Ten. Available from: https://owasp.org/www-project-top-ten/ [Accessed 10 April 2020].