Version control

Version	Date	Author	Description
1.0	2019	Hud Daannaa	Attack Surface Management Complete *(ASMC)* Application's Documentation

Introduction

An attack surface is the total sum of vulnerabilities that can be exploited to carry out a security attack. Attack surfaces can be physical or digital

Purpose

This document is written to describe in detail the ASM aspect of the GSOC service. Focusing on the respective solutions *(Tenable SC and Metasploit)* integrated to achieve a common solution pillar to be built as part of the program. It is the indent of this document to provide the entire high-level design details of each technology & the integration probes for the GSOC eco-system.

Objective

The core objective of this application is to empower the cyber exposure element and minimize the attack surface area a GSOC. The application would leverage the respective functionalities of Tenable SC and Metasploit Pro, a MongoDB database and the result would be visualized in Tableau.

Significance

The key objectives are as follows:

- To help minimize treat response time *(Creating, on-demand or scheduled vulnerability scan templates, attaining the results and validating them on the fly).*
- The program also provides to ability to utilized scan templates that have already been created in tenable SC.
- To promote efficiency and accuracy, since the element of human error is minimized, hence the program automates repetitive tasks that would be done by analysts.
- To provide a means of accessing and managing hosts in a multi-tenant environment.
- The validation of scans via Metasploit and the provision of their respective exploits together with their severity levels.
- To be able to fine tune vulnerability results in other to focus on critical ones.
- To provide a connection to the mongo DB database in other to feed vulnerability and exploitable information for later visualizations and other forms of data analytics on Tableau.
- To help enrich vulnerability information to champion the multi-tenancy feather in the GSOC ecosystem.

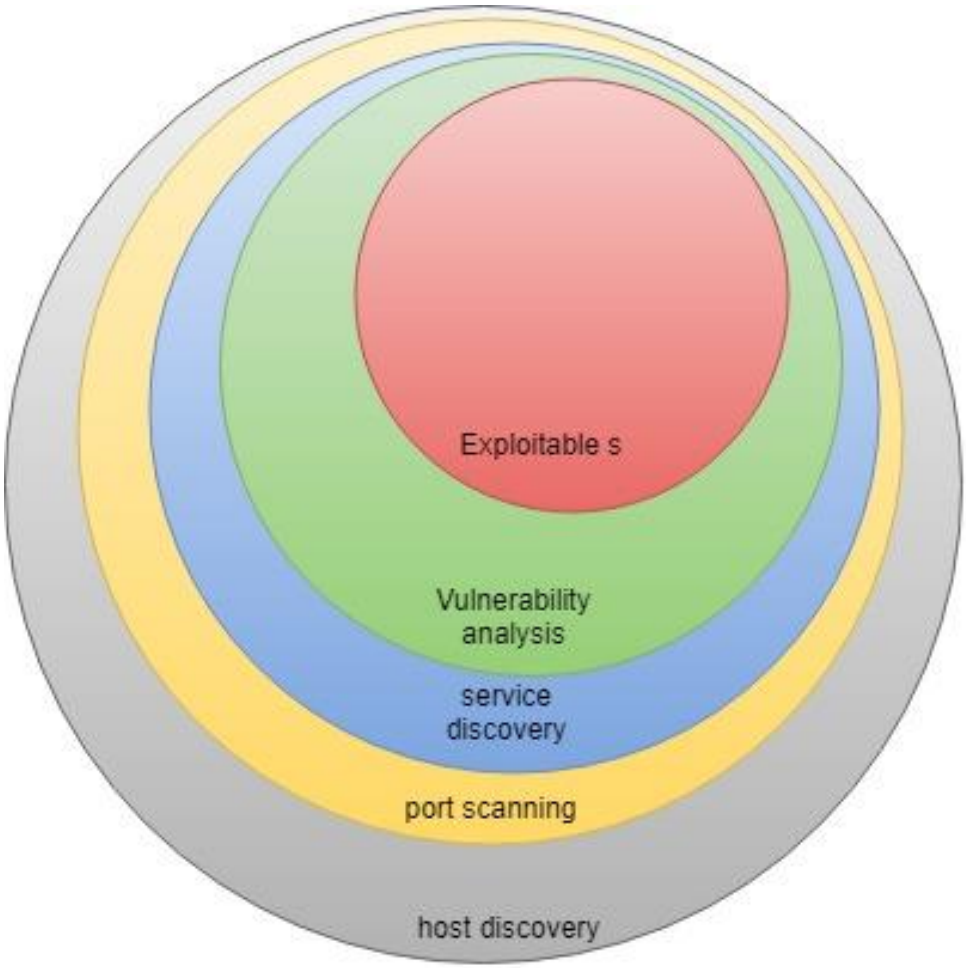
NOTE: *THE APPLICATION CONTAINS SEVERAL README FILES IN SPECIFIC FOLDERS, TO HELP GUIDE AN ANALYST DURING COMMAND LINE OPERATIONS*

System Goals

The system designs use the approach of a penetration test, as outlined below:

- Host discovery
- Port scanning
- Service discovery
- Vulnerability analysis
- Exploitation
- Gaining access

The diagram below illustrates the ordering and location of each phase:



Overview of system

The System is design for the Attack Surface Management *(ASM)* section of a GSOC and fits in as a vulnerability solution to aid in the reduction of the attack surface. It performs scans, enrichment to support a given number of tenants *(Assets)* and provides vulnerability validation. It is basically an integration between two solutions, namely Tenable SC and Metasploit Pro.

System technologies

	Solution	Core functionality
Phase 1 technology	Tenable Security Center *(SC)*	Reconnaissance Port Scanning Service enumeration Vulnerability scanning
Phase 2 technology	Metasploit Pro *(MSF)*	Passive vulnerability exploitation

Requirements

Software specifications

Operating system	Centos, Ubuntu
Language *(Driver)*	Python
Dependencies	PyTenable msgpack Bson PyMongo

Hardware specifications

CPU	2GHz +
RAM	8GB +
Hard disk/ Storage	50GB +

System design

Overview

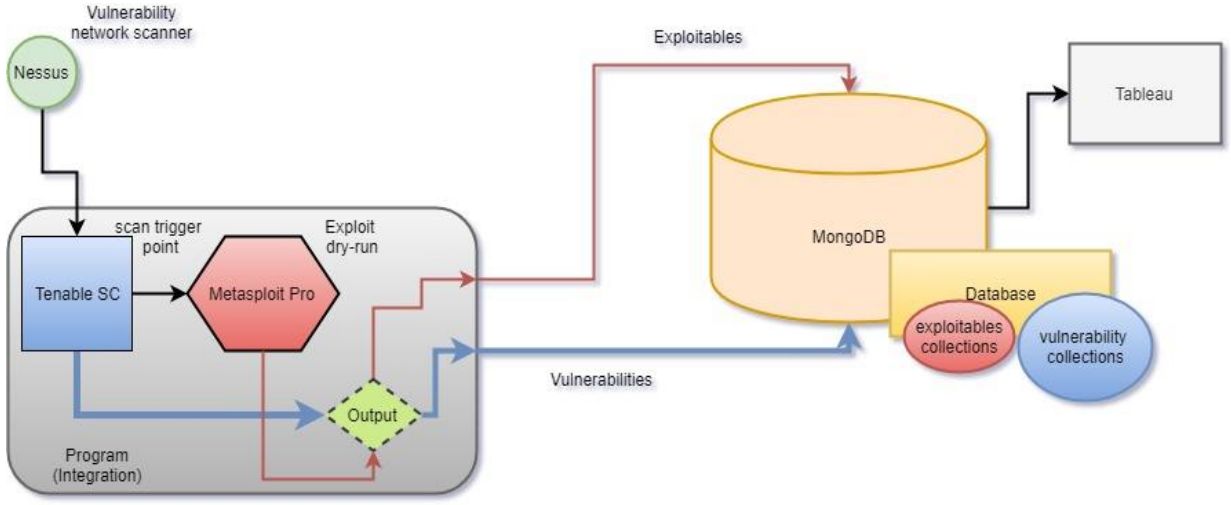
The system comes in two *(2)* modes or parts, the main aim is to help facilitate ASM in a GSOC environment. The core functionality is to convey data from a Nessus scan to the Tableau visualization data analytics platform. This data in question are known as vulnerabilities *(vulns)*.

The movement of vulns from a given Nessus scan on a set network segment goes through several systems in order to come up with a decision-making step which is normally done in the visualizer. Vulns are triggered via SC using several user interface methodologies.

These vulns are enriched and can also be validated depending on the options chosen at the initialization of the of the system. The images below outline the system design and the data flow of operations.

ASM Overview

The image below illustrates the system design and flow of data in the ASM only:



The image above also runs on the background of a server bearing the outlined specification stated in the document. This mode is a user input driven system, simply put, its waits for a user to input data for it to perform given tasks like, running a scan via SC, validations via MSF.

NOTE: All operations still undergo an enrichment process to help power multi-tenancy.

Main components of the Application

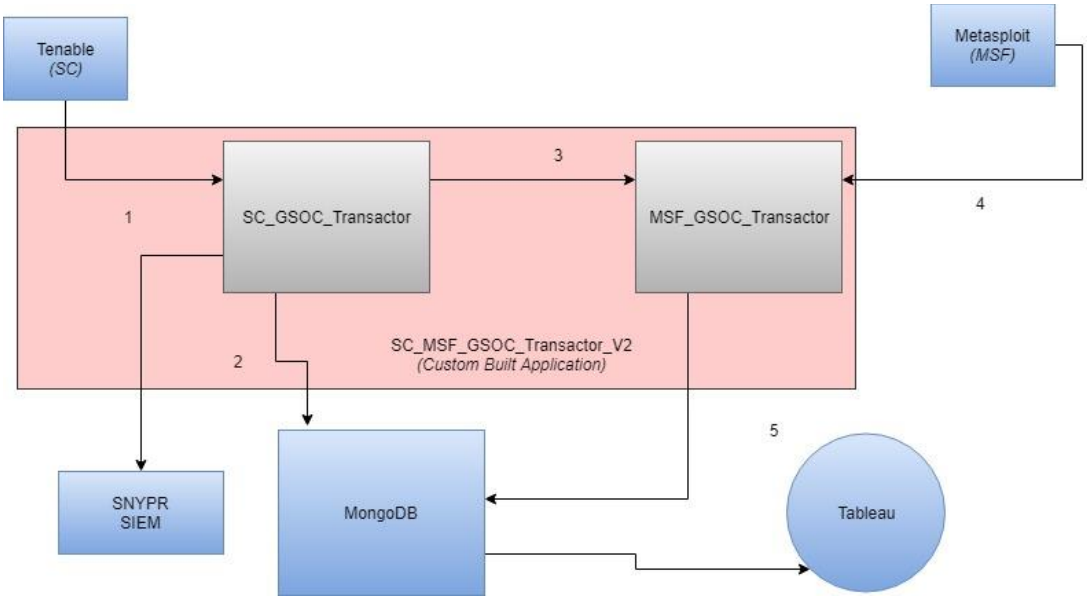
The application can be categorized in three (3) folds, they are:

1. SC Scan Collector
2. MSF Validator
3. ASMCD Control Base

These three components come together to form the SC MSF GSOC Transactor

The General Overview of SC_MSF_GSOC_Transactor_V2

The image below illustrates the system design and flow of data in the SC_MSF_GSOC_Transactor:



The above image runs on the background of a given server bearing the requirement outlined in the previous chapter. The system still collects enriches and forwards scans results to mongo dB for Tableau to visualize

System Characteristics

Performing scans

- Vulnerability scans via SC
- Passive exploitation scans on vulnerabilities via MSF

Data Enrichment

- Multi – Tenant *(Ability to utilize integrated solutions over one or more assets)*
- Time series data *(for trending analysis on Tableau)*
- Exploit *(authenticated Vulnerabilities)* enrichment

Integration to 3rd party solutions like:

- Syslog forwarder *(for SIEM/ Log management integration)*
- MongoDB for Storage *(Scan data/(vuln)result)*
- MongoDB for Credential Storage *(for API keys & Asset information)*

User-interfaces

- Via Configuration file
- Via Tenable SC
- Via Emails *(template)*

System Analysis

SC and MSF Application interactions

This part of the application runs in two folds. The first fold runs through every asset and collects the scan results from SC. These scans are scans that are performed via this application or via SC, hence any scan that is run will be collected so far as SC is involved. These scan results are enriched and pushed to a database *(In a collection called Vulnerabilities).*

The second fold takes all scan results that have been processed by tenable SC and imports them into MSF, where a passive exploitation takes place to produce authenticated vulnerabilities. These vulnerabilities *(auth)* are also pushed to the same database, into a collection called "Exploitables"

The part 1 makes it possible to use the interface of Tenable SC to perform scan *(not necessarily using this application to trigger scan)*, the scans are all enriched, collected to mongo DB, passively exploited and is made visible on Tableau as dash boards.

NOTE: The database in question denotes an asset i.e. and organization.

*ASM_ControlBase_*

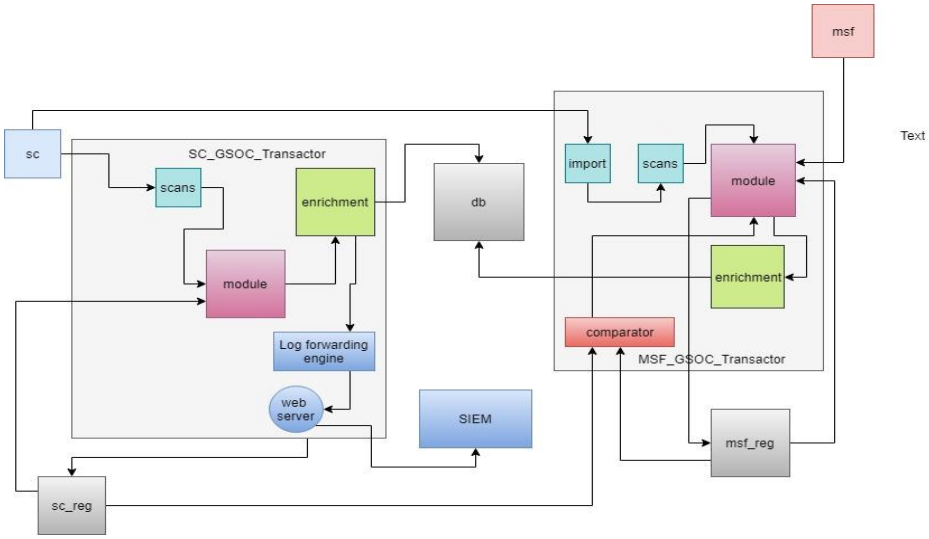
Using emails or a configuration file as interface, one has the option to create a scan template or utilize a predefined scan template in Tenable SC. The is also an option called validator where a user can take the name of a scan result feed it in and validate scans using MSF.

From the above diagram, we leverage the API's of SC and MSF, triggered scan from SC are sent across the application where the resultant scans are enriched and can also be validated with respect to the options selected.

NOTE: All results from both use cases are collected in mongo DB

System Architecture of how SC Scan collector and MSF Validator interact

SC_MSF_GSOC_Transactor_V2 *(Details schematics of from the system design view)*

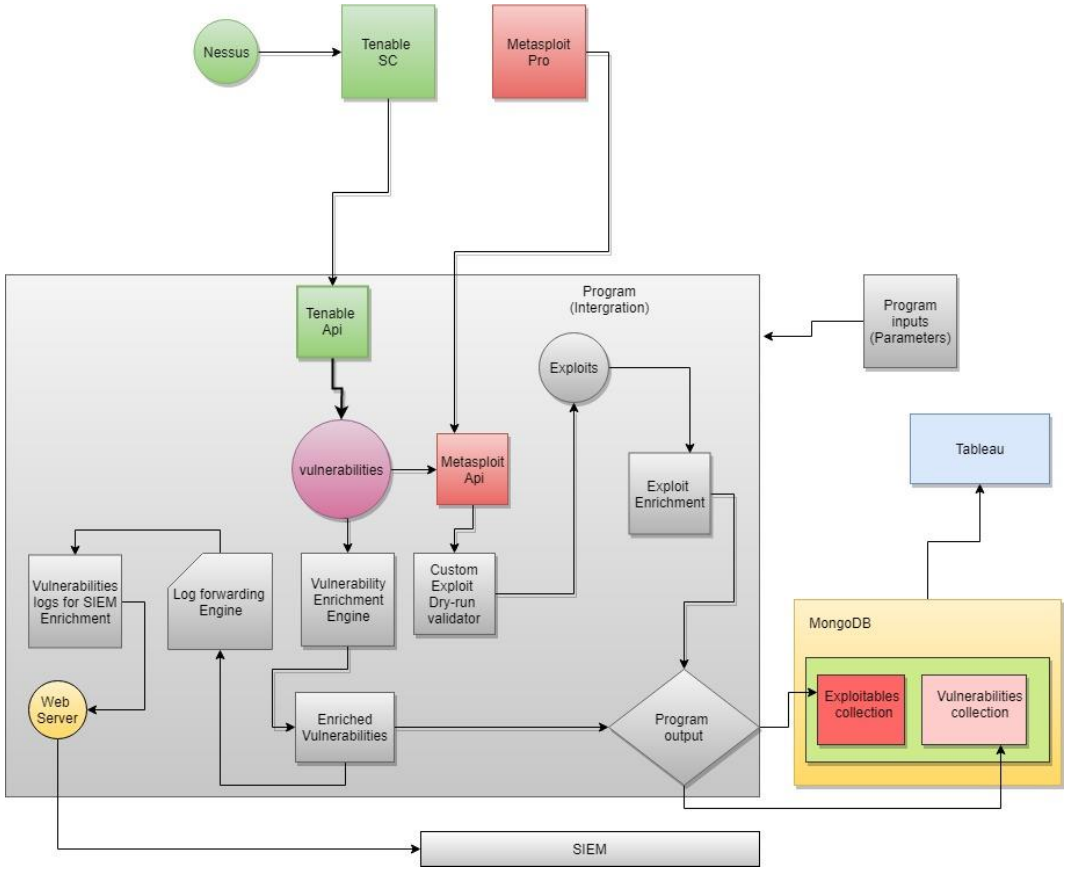


System architecture analysis

From the above diagram:

SC	controls the Nessus scanners from will command or signals are sent for scans to be performed. More than one Nessus scanners are linked to tenable SC. This solution is also the main point of data *(vuln)* of data collection *(indexer)* for the system.
Metasploit Pro	is the pen testing solution on which validation depends on.
Scans	are the data generated *(collected)* from the Nessus scanners and the data indexed and stored in SC. This data is the main driving force for the system.
Enrichment	Without this step there would be no multi- tenancy in ASM
Module	is one of the main engine core components of the designed system and helps to power passive exploitation.
SC_Reg	This is the register for the SC GSOC Transactor *(helps to keep track of scans)*
MSF_Reg	This is the register for the MSF GSOC Transactor *(helps to keep track of validates)*
comparator	This is the key decision point, which determines, what scan data to validate.
DB	It is the main storage facility of the GSOC. Every onboarded asset is assigned a database. Every database is made up of a series of collections that depicts the various aspects and key functionalities of the GSOC. It is fair to say, exploitables and Vulnerabilities will be collects in every database assigned to an asset
Import	It is the step responsible to the data *(scans/vulns)* entry point into MSF
SIEM	

ASM Architecture with log forwarding and API integrations into the application *(Details schematics of from the system design view)*



System architecture analysis

From the above diagram:

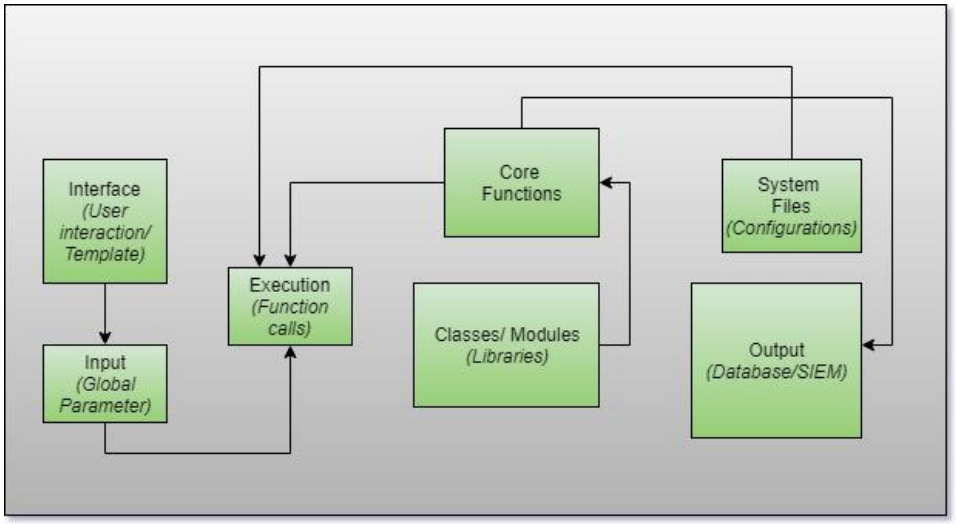
Nessus scanner	this acts as the sensor or main point of data collection for the system
Tenable SC	controls the Nessus scanners from will command or signals are sent for scans to be performed. More than one Nessus scanners are linked to tenable SC. This solution is also the main point of data *(vuln)* of data collection *(indexer)* for the system.
Metasploit Pro	is the pen testing solution on which validation depends on.
SC & MSF APIs	are the means of connection for the system
Vulnerabilities	are the data generated *(collected)* from the Nessus scanners and the data indexed and stored in SC. This data is the main driving force for the system.

Exploits	These are the resultant data generated by the custom module engine.
Custom_Exploit_dry-run_Module	is one of the main engine core components of the designed system and helps to power passive exploitation.
Exploit enrichment	Without this step there would be no multi- tenancy in ASM
MongoDB	It is the main storage facility of the GSOC. Every onboarded asset is assigned a database. Every database is made up of a series of collections that depicts the various aspects and key functionalities of the GSOC. It is fair to say, exploitables and Vulnerabilities will be collects in every database assigned to an asset
Tableau	It is the main visualizer of the GSOC
Log forwarding engine	This is the engine responsible for vulnerability structuring into event and parsing the events into an actionable format for a given SIEM
Vulnerabilities log and enrichment	This is the refined vulnerability events that are readily available to be sent to the SIEM
SMTP server	The system contains an SMTP server, when configured, helps to send feedbacks on triggered activities like commands via email *(IMAP)*
Web server	The web server is a means to export the vulnerability enriched events to the SIEM, from the log forwarding directory, the web server uses that as a root directory and makes the log file accessible to the SIEM
SIEM	The SIEM is a Security information and event management (SIEM) is an approach to security management that combines SIM (security information management) and SEM (security event management) functions into one security management system

System Development Methodology

High Level Code Architecture and data flow *(For Transactor & ASM Control Base)*

The image below outlines the flow of data between the source code components of the Transactor_V2.



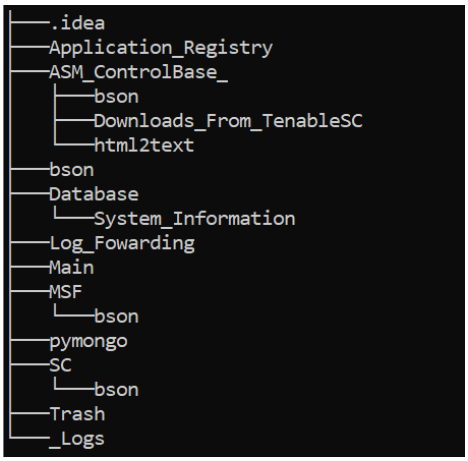
The diagram above illustrates a high-level structure of the source code for both ASM controller base and SC MSF Transactor. Using block diagram as the visual, the source code has seven *(7)* component. These components come all play key roles and respects the principle of segregation of duties.

The seven *(7)* components operates as follows:

Interface	This is the main source of data entry/input into the system. This is usually in a form of a template *(Emails/Configuration file)* or Tenable SC.
Global params	This is collection of all data inputs from the user interface. Where the inputs are validated and filtered.
Execution	This is the functions call section of the source code, were functions are invoked from the libraries *(Classes/modules),* system configuration files are feed here, and the global parameters are feed.
Core functions	This is mainly invoked functions from the libraries *(Classes/modules)*
Classes/ Modules	This is the driving engine behind the whole system. It contains algorithmic and logic modules on which the function call section feeds off.
System files	The part contains server information like, IP address, ports and server keys and credentials
Output	The is the final stage of the application, were data is piped out.

Directory Structure for Source code files *(For Transactor & ASM Control Base)*

The diagram shows the various source code files and the respective directories they dwell in:



Elaborating on the above:

	Folders	Description
1	*Main*	This folder contains the file options.config that which bears the template for controlling ASM Control base
2	*Application Registry*	This contains SC and MSF registers *(SC_register, MSF_register)*
3	*Database*	This folder contains book.config and the system information
4	*System Information*	This folder is found in the database and contains the email register and config.config
5	*SC*	This contains system files and modules for SC
6	*MSF*	This contains system files and modules for MSF
7	*ASM Control base*	The ASM Control Base resides in the root folder
8	*SC MSF GSOC Transactor*	This is the root folder bearing all the application files
9	*Trash*	Program output residue
10	*Log Forwarding*	This is the directory for log and SIEM enrichment files
11	*Pymongo*	This is a python library for mongo db
12	*Global parameters*	This is the global parameters directory, for the applications input
NOTE:		The highlighted areas contain the modules *(libraries)*

Getting Started

This section details how to run ad configure the application. Taking into consideration the hardware and software requirements, and the out lined dependencies. The application consists of a root folder which contains sub folders. The app is meant to reside at root "/".

Every directory worth configuring, or any words of caution has a readme file that should be read for further considerations

Configuring system parameters:

Navigate to the root folder of the application and go to **Database,** in the database folder, go to system information. At system information, there are two (2) config files by name:

1. Email_register
2. Config.config

Configure the **config.config** file with the right parameters as outlined in the readme

```
MONGODB CONFIG
====
3.Host=
4.Port=27017


TENABLE SC CONFIG
====
9.Host=
10.Port= 443
11.Collection_name=Vulnerabilities


METASPLOIT PRO CONFIG
====
16.Host=
17.Port=3790
18.SSL=False
19.Token=
20.Collection_name=Exploitables_2

IMAP SERVER CONFIG
====
24.file_name = /SC_MSF_GSOC_Transactor_V2/asmcd_template.config
25.imap_host =
26.imap_user =
27.imap_pass =

SMTP SERVER CONFIG
====
31.smtp_host =
32.smtp_return =

NOTE:   DO NOT ALTER, CONTACT AUTHOR OR SENIOR
        TECHNICAL ADMINISTRATOR
```

NOTE: *THE ORDER AND ALIGNMENT OF TEXT WITH RESPECT TO THE NUMBERED LINED SHOULD NOT UNDER ANY CIRCUMSTANCE BE ALTERED*

Also, go to the **Email_register.config** and specify the users authorized to send emails for scan triggers.

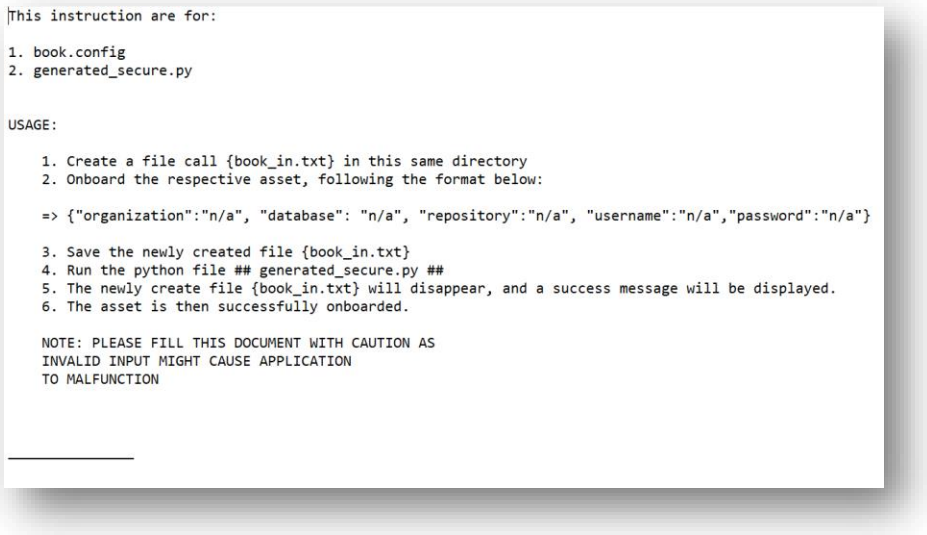
```
somebody@mail.ae
to_whom@it_may_concern.ae


```

Database *(Knowledge base file generation)*

The application has a database which is called "**book.config**" this contains organizational or Asset information. This file serves as the database and is the most important component since any wrong input will render the application in-operable. The content of this file can be pushed to a given database like a mongo dB *(in case of future upgrades or for future integrations).* The content of *book.config* is encoded and is housed in the database section of the application. The generated_secure.py is a script within the program that takes a file called book_in.txt as input and generates the *book.config*. this means that the book_in.txt will contain the organizational data *(this data is aligned in a specific format).* Refer to the README file within the database folder to gain more insight of the respective operation. The operations are outlines below:

Go back to the database folder and take note of the file ***generated_secure.py*** and refer to the readme, this states that:

Readme:



```
This instruction are for:

1. book.config
2. generated_secure.py


USAGE:

    1. Create a file call {book_in.txt} in this same directory
    2. Onboard the respective asset, following the format below:

    => {"organization":"n/a", "database": "n/a", "repository":"n/a", "username":"n/a","password":"n/a"}

    3. Save the newly created file {book_in.txt}
    4. Run the python file ## generated_secure.py ##
    5. The newly create file {book_in.txt} will disappear, and a success message will be displayed.
    6. The asset is then successfully onboarded.

    NOTE: PLEASE FILL THIS DOCUMENT WITH CAUTION AS
    INVALID INPUT MIGHT CAUSE APPLICATION
    TO MALFUNCTION


_____
```

From the above, **book_in.txt** is the file responsible for *onboard assets/ organization on the application*. The readme specifies the right format to write asset metadata, which will be used by the application. The below information details the format in the readme file:

{"organization":"n/a", "database": "n/a", "repository":"n/a", "username":"n/a","password":"n/a"}

After running the command: *python generated_secure.py*

The **book.config** file will be generated is configured *(This forms the data or knowledge base of the application).*

IMPORTANT: After the above configuration, *applications flow* :

kickstart app >> Perform Scans >> View Logs >>View Report >> Drink Green tea