2019 GSoC OWASP Proposal

Personal Profile

My name is Bocheng Zhang(Mars Zhang) and I am a sophomore student majoring in Computer Science at Hangzhou Dianzi University(HDU), China.

My Coding Skill Sets

Basically, I am well equipped with Python, C, C++ and Machine Learning knowledge due to my study and project experience. And I am also familiar with Windows and Linux system, especially with KALI and Centos.

My Project Experience

I mainly program on windows system and Linux system (KALI and centos). I am fluent in C and python as the programming language for research and production code usage.

Since last year, I have been working as a researcher with an Information Security Research team at my university, an innovative group aiming at the security of the industrial control equipment. The project I mainly contribute to is to is a system which has four functional modules: on-line verification and scanning of industrial control systems; non-intrusive verification of industrial control systems; plug-in verification of industrial control systems; customized search and visualization of situational awareness. My role is implementing scan identification, as well as using HMM model and TF-IDF algorithm to our security identification system. This experience not only stimulates my enthusiasm in information security, but also helps develop my ability to update my own vulnerability information and the multi-process crawls the latest vulnerability information of the first-line security website. This link shows a demo of the initial implementations of our system.

I am committed to building and managing databases and successfully implementing scan identification. I also use NLP to implement real-time automatic updates to the database and to eliminate redundancy.

In the meantime, I work for HDU INPAINT Research Group which focuses on the research on Image Restoration Technology. Photographs are often unrecognizable due to late damage or unrelated factors on the spot. The goal of this group is to fix these photos with impurities. I tried a variety of methods introduced in cutting-edge conference papers such as Object Removal by Exemplar-Based Inpainting, combined with deep learning to fix photos.

My Open Source Experience

Open Source

I have always liked the form of open source. I think open source can draw more people's wisdom and inspiration so that the project can be more dynamic and innovative. I have used the Nmap package before, and I have read the relevant code that is exposed on GitHub. BLT has attracted me so far, I also cloned it to my computer and tried to run it. These experiences have made me feel that open source is a very good form of the project, both for users and developers.

OWASP

Contributition to DefectDojo Project

I'm working on contributions to DefectDojo OWASP. I wrote the unittest for parser.py in the nsp folder.

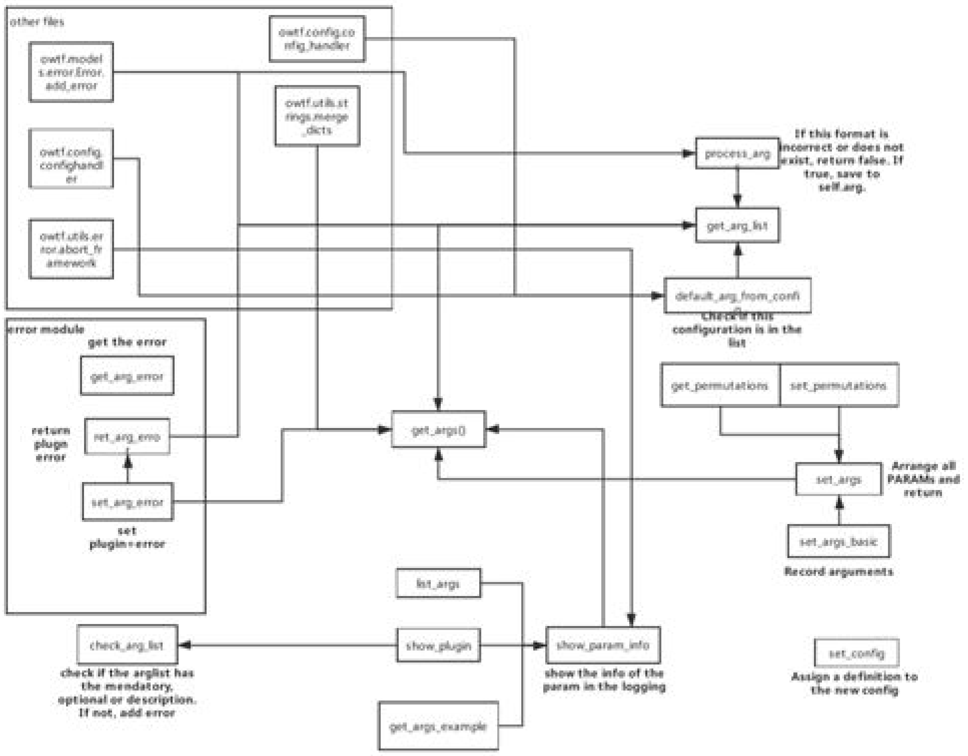
Fix bug #975

- At the same time, I am also working on bugs for network scanner. Some progress on this bug: there is no get_all function in the target_manager class of owtf.managers.target in the process_plugin_list function, which cause the errors. According to the performance of the process_plugin_list function, the get_all function should be used to obtain the ID information of the target. After examining at the target_manager class, I replaced the get_all function with the get_all_target() function with the help of mohit and it worked fine.
- I also raised my own doubts about this and the inference of possible reasons: In the 'for http_ports in http' loop, the plugin_group value passed to the process_plugins_for_target_list function is “web”. The function has one priority is to judge ‘if plugin_group == "network" ’, so no subsequent operations are performed.
- In the ‘for loop‘ mentioned above, the passed target_list parameters are {"https://{}".format(target.split("//")[1])} and {target}. But the target does not be in the list format, so it may lead to the repetition of the following steps.
- With the help of Viyat, I think the reason why it doesn't work is: In the 'loop of the target', the 'lastwave' is always the value of 'wave[0]' after the second loop, which causes the parameters of the subsequent ‘get_tcp_ports’ function always is two equal values.

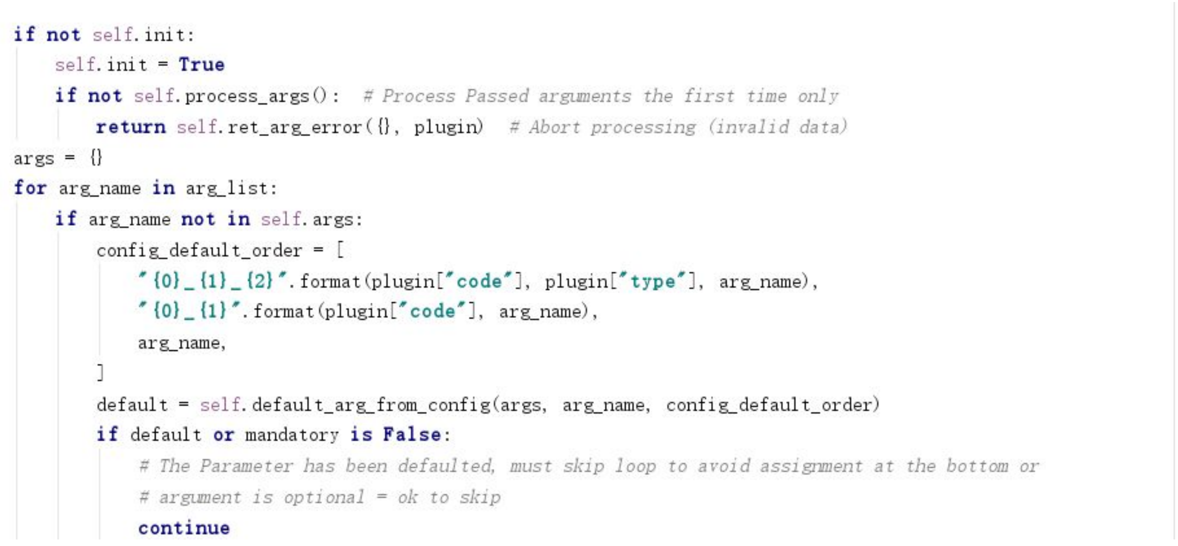
Other Issues

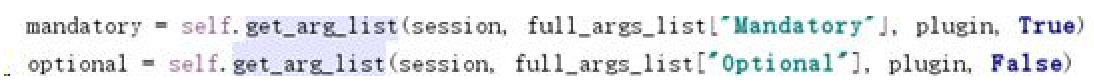
Function Bug

The following is a UML I created for modules related to param.py.


other files
owtf.models.error.Error.add_error
owtf.config.config_handler
owtf.utils.strings.merge_dicts
owtf.config.confighandler
owtf.utils.error.abort_framework
process_arg
If this format is incorrect or does not exist, return false. If true, save to self.arg.
get_arg_list
default_arg_from_confi
Check if this configuration is in the list
error module
get the error
get_arg_error
return plugin error
ret_arg_erro
set_arg_error
set plugin=error
get_args()
get_permutations
set_permutations
set_args
Arrange all PARAMs and return
set_args_basic
Record arguments
list_args
check_arg_list
check if the arglist has the mendatory, optional or description. If not, add error
show_plugin
show_param_info
show the info of the param in the logging
get_args_example
set_config
Assign a definition to the new config

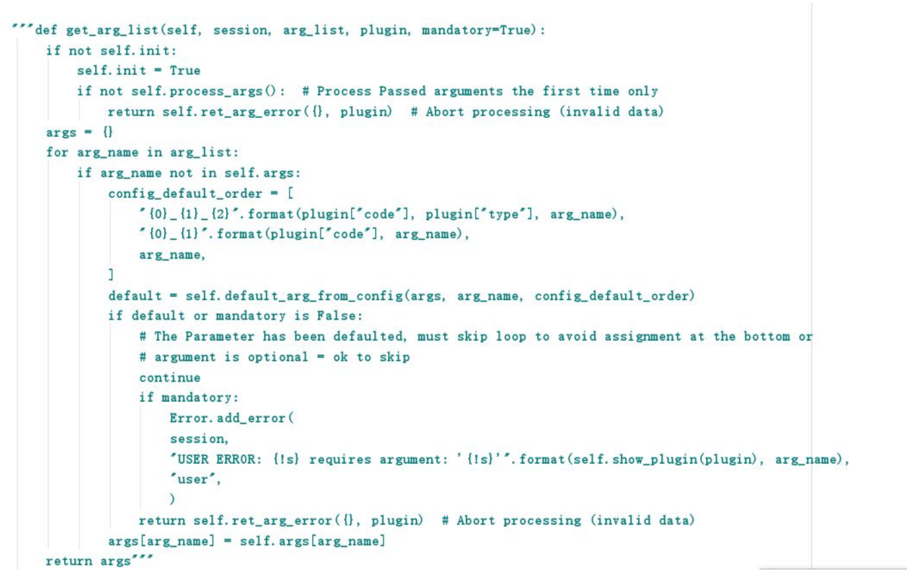
It seems 'get_arg_list' function does not work properly. When argument is optional, it also jumps out of the loop directly through return, which causes the error to be output and the loop to be terminated when the function calls and enters 'mandatory=true'. Function issue can be seen as belows.


```python
if not self.init:
    self.init = True
    if not self.process_args():  # Process Passed arguments the first time only
        return self.ret_arg_error({}, plugin)  # Abort processing (invalid data)
args = {}
for arg_name in arg_list:
    if arg_name not in self.args:
        config_default_order = [
            '{0}_{1}_{2}'.format(plugin['code'], plugin['type'], arg_name),
            '{0}_{1}'.format(plugin['code'], arg_name),
            arg_name,
        ]
        default = self.default_arg_from_config(args, arg_name, config_default_order)
        if default or mandatory is False:
            # The Parameter has been defaulted, must skip loop to avoid assignment at the bottom or
            # argument is optional = ok to skip
            continue
```



```
mandatory = self.get_arg_list(session, full_args_list["Mandatory"], plugin, True)
optional = self.get_arg_list(session, full_args_list["Optional"], plugin, False)
```

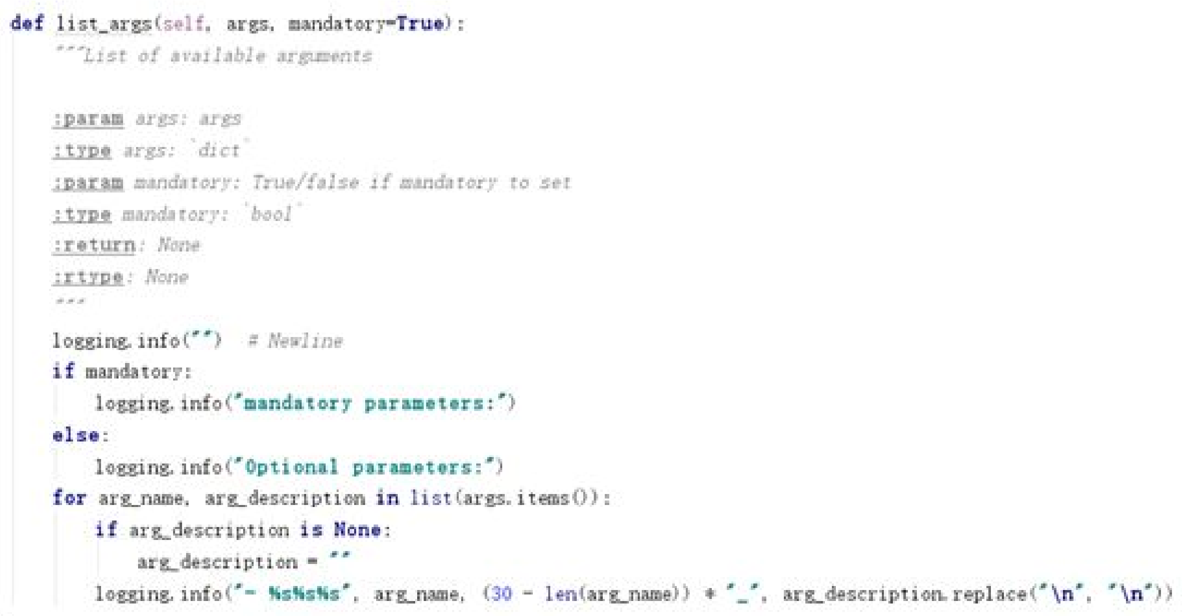
I tried to make the following corrections:



```
"""def get_arg_list(self, session, arg_list, plugin, mandatory=True):
    if not self.init:
        self.init = True
        if not self.process_args():  # Process Passed arguments the first time only
            return self.ret_arg_error({}, plugin)  # Abort processing (invalid data)
    args = {}
    for arg_name in arg_list:
        if arg_name not in self.args:
            config_default_order = [
                "{0}_{1}_{2}".format(plugin["code"], plugin["type"], arg_name),
                "{0}_{1}".format(plugin["code"], arg_name),
                arg_name,
            ]
            default = self.default_arg_from_config(args, arg_name, config_default_order)
            if default or mandatory is False:
                # The Parameter has been defaulted, must skip loop to avoid assignment at the bottom or
                # argument is optional = ok to skip
                continue
                if mandatory:
                    Error.add_error(
                    session,
                    "USER ERROR: {!s} requires argument: '{!s}'".format(self.show_plugin(plugin), arg_name),
                    "user",
                    )
                return self.ret_arg_error({}, plugin)  # Abort processing (invalid data)
            args[arg_name] = self.args[arg_name]
    return args"""
```

Name Repeat Issue

From the code, it looks that the ‘args parameter of list_args’ function and the ‘full_args_list of get_args_example’ function are equal, but their titles are different. The following values and keys seem to have the same issue.



```
def list_args(self, args, mandatory=True):
    """List of available arguments

    :param args: args
    :type args: `dict`
    :param mandatory: True/false if mandatory to set
    :type mandatory: `bool`
    :return: None
    :rtype: None
    """
    logging.info("")  # Newline
    if mandatory:
        logging.info("mandatory parameters:")
    else:
        logging.info("Optional parameters:")
    for arg_name, arg_description in list(args.items()):
        if arg_description is None:
            arg_description = ""
        logging.info("- %s%s%s", arg_name, (30 - len(arg_name)) * "_", arg_description.replace("\n", "\n"))
```

PEP-8 Format Issue

Regarding the project guidance, PEP-8 style is required to keep code standard. However currently there are much code need to be converted.

Project Information

Project Name: OWASP OWTF_New Plugin Architecture

Project Description

The experience of existing OWTF plugin system could be improved when we browse plugins. And most of the plugins have code issues. In order to improve it, I will make a clear classification for OWTF's plugin and implement it by refactoring the original API. Additionally, I will also add new plugin types and write the complete unittest. This may also involve a more complete scanner.py to ensure the correct operation of each plugin.

Why I am interested in OWASP OWTF

I am very interested in OWTF because I think the workflow is impressive especially combined with my previous security protection project. First, it would be very good user friendly design with clear guidance to follow up. For example, after it scans the destination WebUI, OWTF could automatically analyses and generates a detailed report for user to determine whatever they want to test. What’s more, users can pause and restart their work at any time. And it also has a lot of functions to match different types of security requests, such as TCP with plugins.

Also I believe in its idea of that can provide a good detection effect for penetration attacks. So I hope it can be better improved and broadly used to detect the existence of the attack early and reduce the losses.

I chose New Plugin architecture project idea because I worked on similar projects before, so I am much confident with capability to complete project. Meanwhile, this project idea requires some new functions and plugins to work on, which is very interesting to me. So I am eager to contribute.

Deliverables

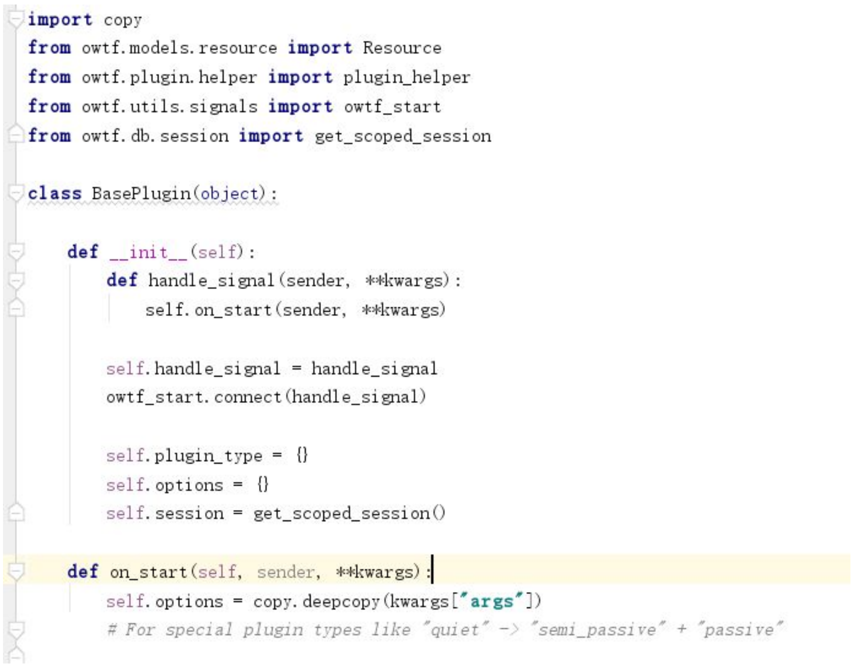
Fix Issues

- Refactor the related plugin API to solve the issues and make it work properly and efficiently.
- Refactor its code to conform to the PEP-8 format.

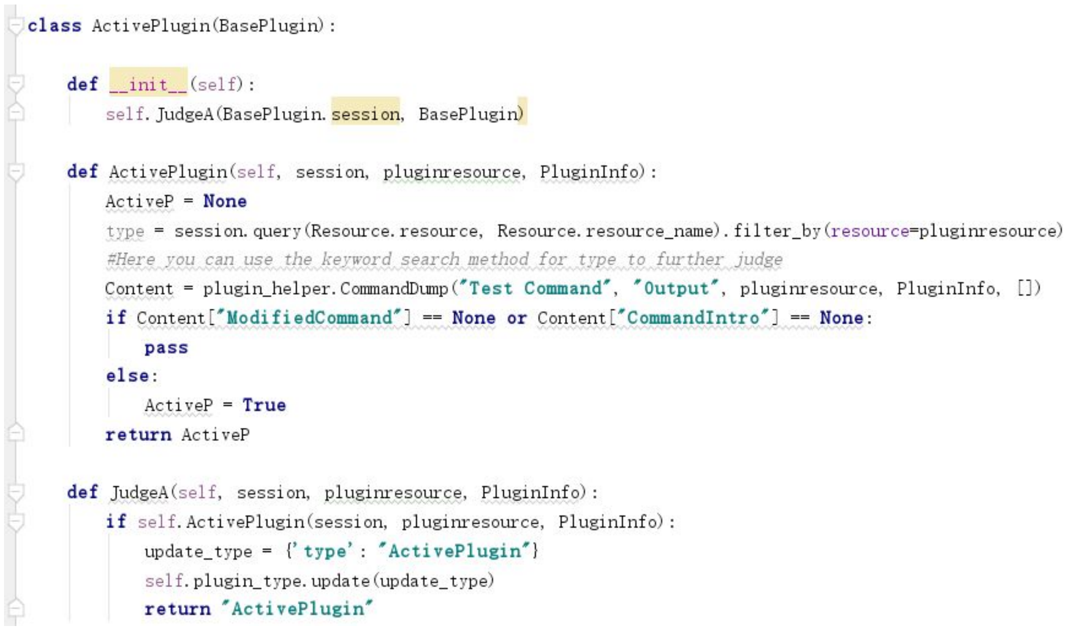
- Improve its naming standards to make it more uniform and clear.
- Reproduce its functionality and correct problems such as incorrect value passing and incomplete operation.

Design New Plugin Classification

- Define different plugins according to OWTF's existing plugin.
- Write a BasePlugin, which is the integration of different types of plugins, and also contains the existing plugin_type to make the entire OWTF plugin more tidy and simple (Sample class signature and APIs listed below).
- Inherit to use more plugin types, including (Sample class signature and APIs listed below).
  - PassivePlugin is characterized that can be triggered without an active command
  - ActivePlugin requires commands such as ModifiedCommand to trigger actively, and the result will be different RawOutput, PluginOutputDir, etc. depending on the command.
  - SemiPassivePlugin is very similar to ActivePlugin and often requires active commands to trigger. But SemiPassivePlugin needs to be connected to the relevant URL and usually needs to use the cache.
  - GrepPlugin do NOT send traffic to the target and only grep the HTTP Transaction Log.
  - ExternalPlugin is used to assist external functions such as calling test files of peripherals.
  - NetworkPlugin is related to network-side plugins, such as those involving ftp and http.
  - AuxPlugin has its own categories, or subcategories, and they have different mandatory and optional.
- Plugins design plan
  - Next up is my design on Baseplugin and its Activeplugin:

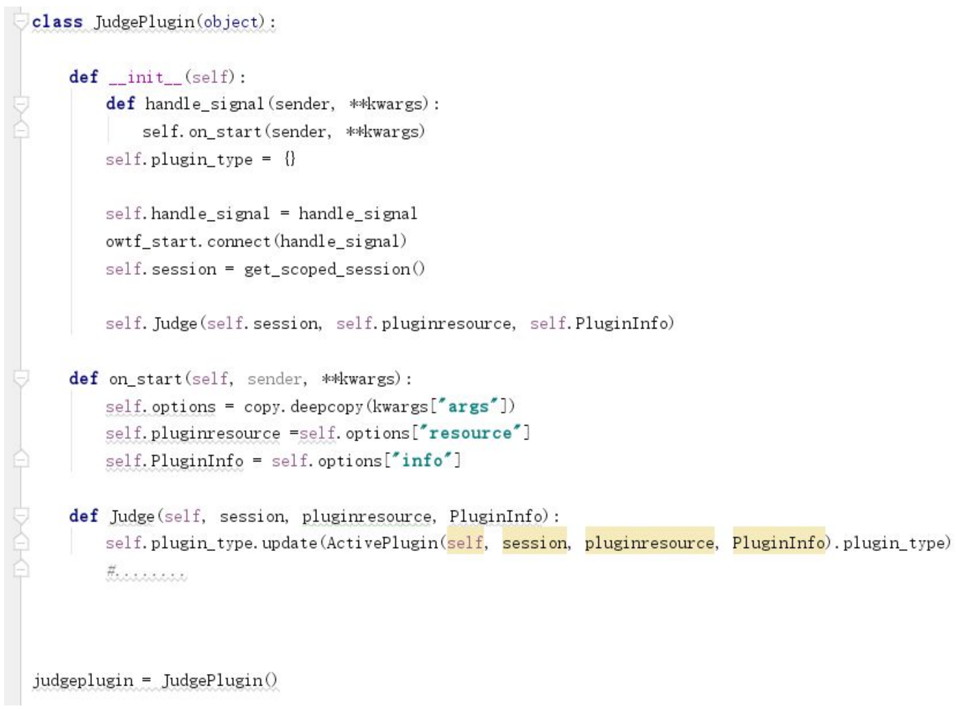


```python
import copy
from owtf.models.resource import Resource
from owtf.plugin.helper import plugin_helper
from owtf.utils.signals import owtf_start
from owtf.db.session import get_scoped_session

class BasePlugin(object):

    def __init__(self):
        def handle_signal(sender, **kwargs):
            self.on_start(sender, **kwargs)

        self.handle_signal = handle_signal
        owtf_start.connect(handle_signal)

        self.plugin_type = {}
        self.options = {}
        self.session = get_scoped_session()

    def on_start(self, sender, **kwargs):
        self.options = copy.deepcopy(kwargs["args"])
        # For special plugin types like "quiet" -> "semi_passive" + "passive"
```



```python
class ActivePlugin(BasePlugin):

    def __init__(self):
        self.JudgeA(BasePlugin.session, BasePlugin)

    def ActivePlugin(self, session, pluginresource, PluginInfo):
        ActiveP = None
        type = session.query(Resource.resource, Resource.resource_name).filter_by(resource=pluginresource)
        #Here you can use the keyword search method for type to further judge
        Content = plugin_helper.CommandDump("Test Command", "Output", pluginresource, PluginInfo, [])
        if Content["ModifiedCommand"] == None or Content["CommandIntro"] == None:
            pass
        else:
            ActiveP = True
        return ActiveP

    def JudgeA(self, session, pluginresource, PluginInfo):
        if self.ActivePlugin(session, pluginresource, PluginInfo):
            update_type = {'type': "ActivePlugin"}
            self.plugin_type.update(update_type)
            return "ActivePlugin"
```

- There will be an unified Judgeplugin to call all the previous classes to get the final plugin_type, as follows:



```python
class JudgePlugin(object):

    def __init__(self):
        def handle_signal(sender, **kwargs):
            self.on_start(sender, **kwargs)
        self.plugin_type = {}

        self.handle_signal = handle_signal
        owtf_start.connect(handle_signal)
        self.session = get_scoped_session()

        self.Judge(self.session, self.pluginresource, self.PluginInfo)

    def on_start(self, sender, **kwargs):
        self.options = copy.deepcopy(kwargs["args"])
        self.pluginresource =self.options["resource"]
        self.PluginInfo = self.options["info"]

    def Judge(self, session, pluginresource, PluginInfo):
        self.plugin_type.update(ActivePlugin(self, session, pluginresource, PluginInfo).plugin_type)
        #........


judgeplugin = JudgePlugin()
```

- Improve on this basis and add new judgment methods such as keyword search by NLP.

Design Each New Plugin's Unittest File

- Modify and rewrite the current plugins to conform to the classification.
- Add the new plugin type described above to the runner.py and other files to update the plugin's classification to make it more accurate and neat.

Add New Features

- Support user supplied custom plugin groups and enable persisting custom plugin groups for future use by the user. This gives users the freedom to choose the plugins they need and perform targeted tests.
- We can also collect the selection of a large user group and get a report of the test needs within a certain period of time. This can also help us build a near-term possible threat direction to pass this information to the relevant security workers.

Update My Work

- Keep my work updated to my GitHub so that subsequent contributors can clearly understand my workflow.
- Communitate with mentors and group to get feedback.

TimeLine

Community Bonding Period (May 06-27, 2019)

- Week 1 (May 06-13,2019): Further discussion about proposal with community.
- Week 2 (May 13-20,2019): Refactor and finalize the plugin API from owtf.plugin.params.
- Week 3 (May 20- 27,2019): Refactor and finalize the plugin API from owtf.help.py.

First Evaluation Period Deliverables (May 27 -Jun 29,2019)

- Week 4 (May 27-June 03,2019): Fix the bug for runner.py and make it work with both files.
- Week 5 (June 03-June 10,2019): Write a BasePlugin, which contains several new plugin types mentioned above.
- Week 6 (June 10-June 17,2019): Complete the test file exclusive to the above functions.
- Week 7 (June 17-June 24,2019): Synchronize these new plugin formats into individual API files to make them available.
- First Evaluation time:Research and communicate with mentors About previous progress(June 24 -29,2019)

Second Evaluation Period Deliverables (Jul 01- 26,2019)

- Week 8 (July 01-July 08,2019): Ensure that the new API will run successfully and that the new plugin_type will be displayed on the user interface.
- Week 9 (July 08-July 15,2019): Build a data collection function for the user's choice of plugin.

- Week 10 (July 15-July 22,2019): Create functions that analyze user-selected data and build reports in real time.
- Second Evaluation time(July 22 -26,2019)**:**Research and communicate with mentors About previous progress

Final Evaluation Period Deliverables (Jul 26 -Aug27,2019)

- Week 11 (July 26-Aug 2,2019):Implement the above two functions on the web interface.
- Week 12 (Aug 2-Aug 9,2019):Integrate and add the above two features to the API and ensure correct operation.
- Week 13 (Aug 9-Aug 16,2019):Integrate the functionality of my own implementation and organize the logs.
- Week 12 (Aug 16-Aug 27,2019): submit

Why You Should Choose Me?

My Ability to Contribute

As mentioned in self-introduction and project experience, I have solid background and hand-on project experience with Python and security-related programming projects. With the experience of implementing a complete and effective background database automatic collection and update function and related APIs for my research team, I am confident in the implementation of OWTF's high-performance new plugin and related APIs. After supplementing with more in-depth learning, I can make some better fixes for OWTF's backend environment.

Self-learner with High Motivition

I am good at self-learning and seeking out solutions whatever the case is. For example, since I started my GSoC journey, I have been reading organization documents, watching tutorial videos on Youtube and talking with community. Now I am familiar with the relevant code I am working on, as well as plugins such as runner.py, etc.

Teamwork with Community

When I was working on my proposal, I enjoy communication and coordination with community and mentors, which helps me get used to community rules and teamwork.I appreciate everyone’s help and I hope I will successfully finish my project to as return to our community.

My Summer Plans

For this summer, I do not have any other plans except GSoC so far. If I am accepted, I will be able to work on GSoC project for 30-40 hours per week. I will always keep in touch with the OWTF organization and mentor and actively complete my work.

Reference

- Automating Security Testing with the OWTF
- OWASP OWTF Botnet Mode
- Define the classification plug-in
- Nmap Network Scanning