«EAST» penetration testing framework. Documentation.

Pentest framework environment is the basis of IT security specialist's toolkit.

This software is essential as for learning and improving of knowledge in IT systems attacks and for inspections and proactive protection.

The need of native comprehensive open source pen test framework with high level of trust existed for a long time. That is why EAST framework was created for native and native friendly IT security markets.

EAST is a framework that has all necessary resources for wide range exploits to run, starting from Web to buffer overruns.

EAST differs from similar toolkits by its ease of use. Even a beginner can handle it and start to advance in IT security.

Main features:

- **Framework security.**
  Software used for IT security must have a high level of user trust. Easy to check open source Python code realized in EAST. It is used for all parts of the framework and modules. Relative little amount of code eases its verification by any user. No OS changes applied during software installation.
- **Framework maximum simplicity.**
  Archive downloads, main python script start.py launches, which allows exploits start-stop and message traffic. All handled local or remotely via browser.
- **Exploits simplicity of creation and editing.**
  Possibility to edit and add modules and exploits on the fly without restart. Module code body is easy and minimal in terms of amount.
- **Cross-platform + minimal requirements and dependencies.**
  Tests for Windows and Linux. Should function everywhere where Python is installed. Framework contains all dependencies and does not download additional libraries.
- **Full capacity of vanilla pen test framework.**
  In spite of simplicity and "unoverload" the framework has all necessary resources for wide range exploits to run, starting from Web to buffer overruns.
- **Wide enhancement possibilities.**
  Third party developers can create their own open source solutions or participate in EAST development by use of Server-client architecture, message traffic API and support libraries.

EAST and alternate frameworks comparison table

Characteristics	EaST	Core	Canvas	Metasploit
Ease of use and operation	+	–	–	+/-
Code verifiability (without backdoors)	100%	0%	50%	50%
Educational manuals (in Eng,	+	–	–	+/-

Rus, Am)				
Content (exploits, exploit packs, tools)	+	+	+	+

Documentation

1. General scheme

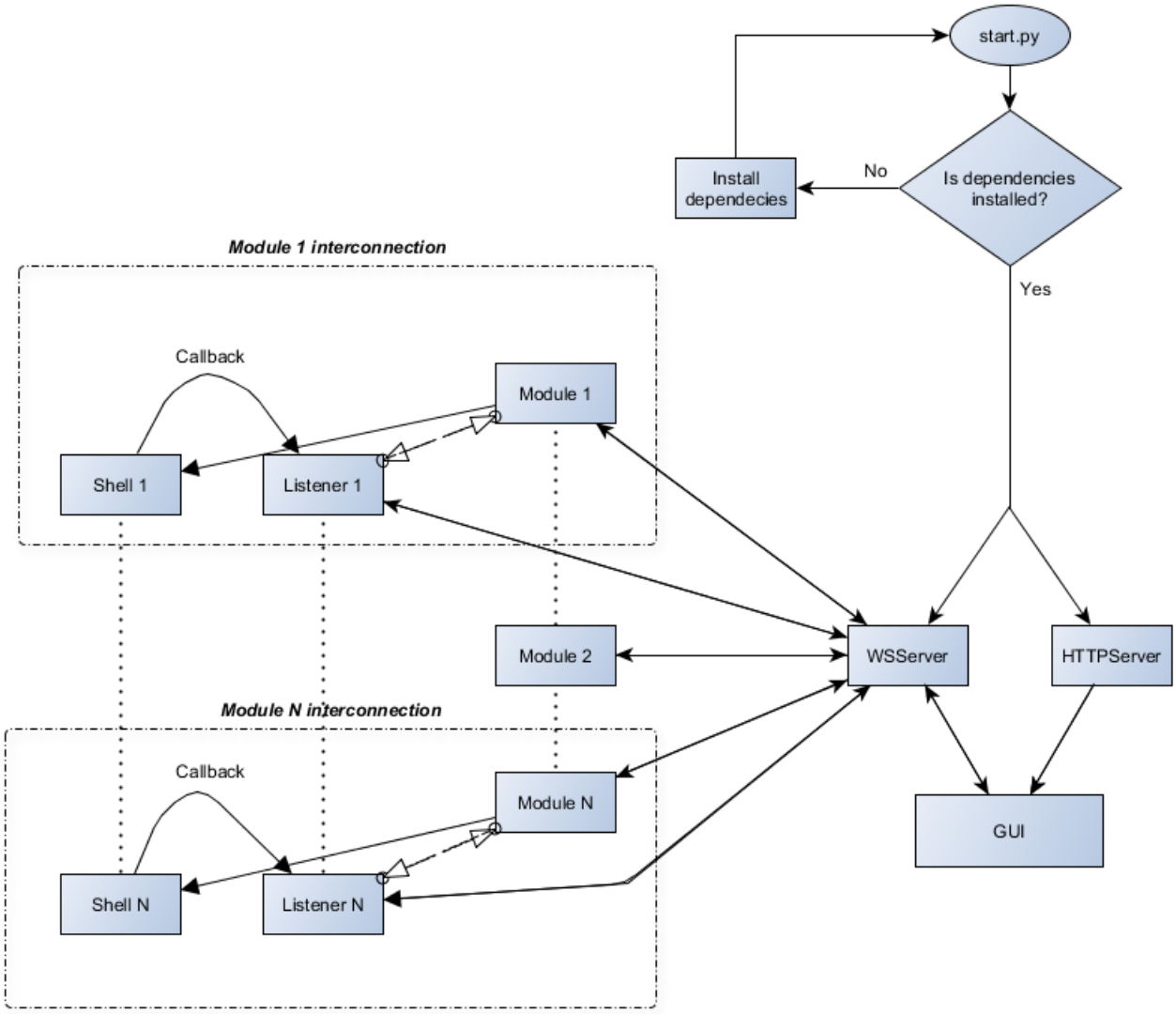


Рис.1.1. main structure scheme

During framework launch, third party python modules check initiated.

The framework automatically installs missing modules from “3rdPartyTools” folder. With availability of all modules then goes launch of:

a) web socket server (WSServer) which is the core and

b) HTTPServer, responsible for GUI visualization. GUI connection information displayed to console (host and port) for GUI connection through web browser.

1.1. Commands system.

Websocket server listens defined port. Clients can connect to it by websocket protocol. All framework entities (except server itself), such as exploits, listeners and GUI, play clients role. Request standard form looks like an item (dictionary) with the structure indicated below:

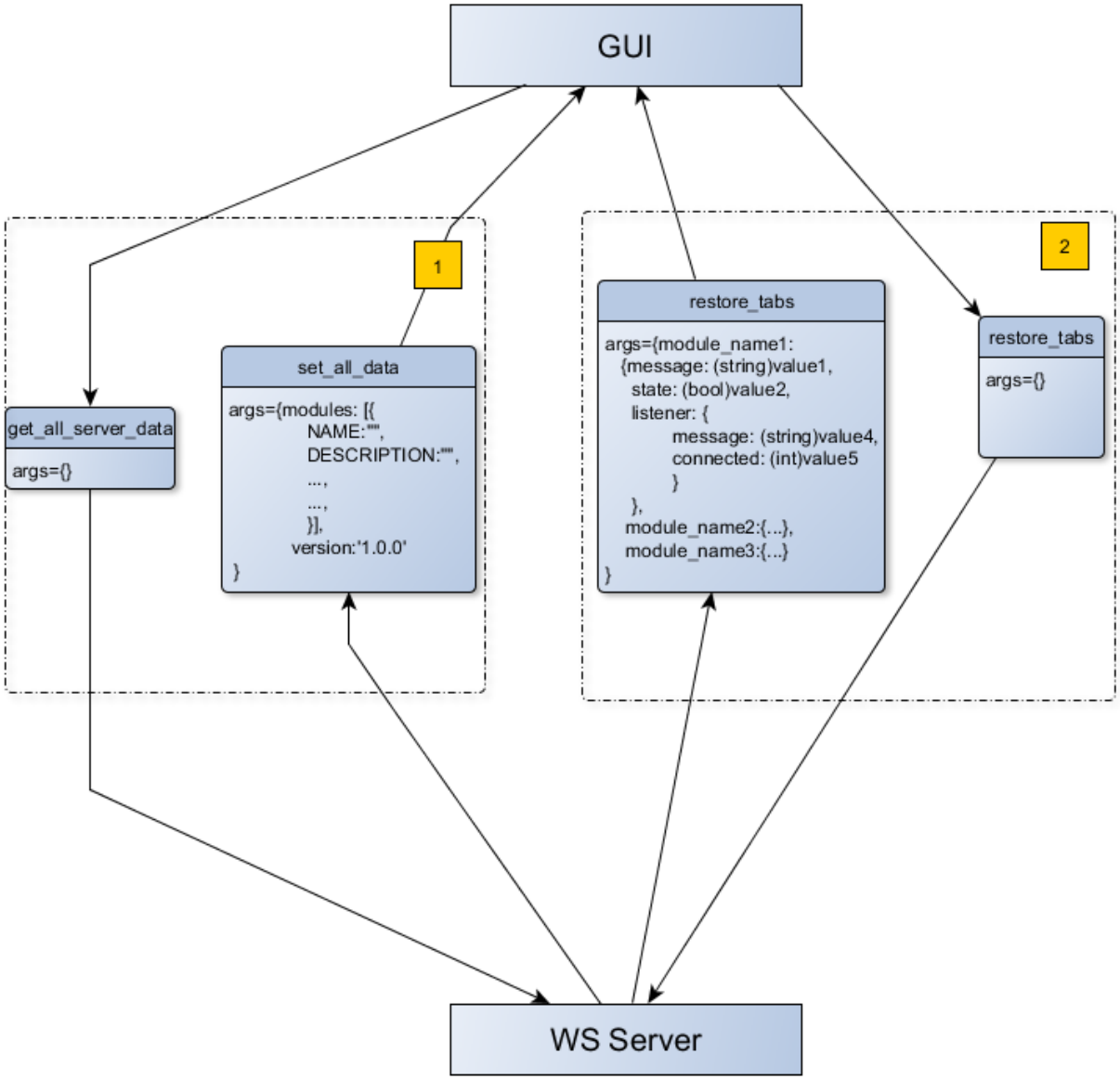
```
1 {
2 “command”: “some_command”,
3 “args”: {
4 “arg1”: value,
5 “arg2”: value2
6 }
7 }
```

Next, this item serialized in JSON. Server has specified commands list. JSON – string is sent to the server. Then JSON->dict parsing happens and dictionary
with ***“command”*** and ***“args”*** compulsory keys appears. “***commands”*** contains on server initialization command. Depending on command, server functions differently, argument for functions is ***“args”*** value, as well as ***request*** item which allows to send a response to the client who initiated current command. Standard form of command agent function looks like:

```
1 1: def some_command_handler(self, args, request):
2 2: a = args[‘a’]
3 3: b = args[‘b’]
4 4: result = self.do_something(a, b)
5 5: request.send_message(json.dumps(result))
```

Explanation: In strings 2,3 ‘a’ and ‘b’ keys value assignment occurs in args dictionary. Usually key entity test is made before assignment. Onwards received values used as any function argument that can recall any value. After this value serialized in json and sent back to the client who can have its own command agent (with the same structure).

2. GUI- EastMainServer interaction



2.1. GUI initialization

When EAST framework user connects to the web interface (GUI) and loads appropriate content, that triggers an “event” which causes the webServer to connect to websocket main Server.

Right after successfull connection, the command «get_all_server_data» is generated.

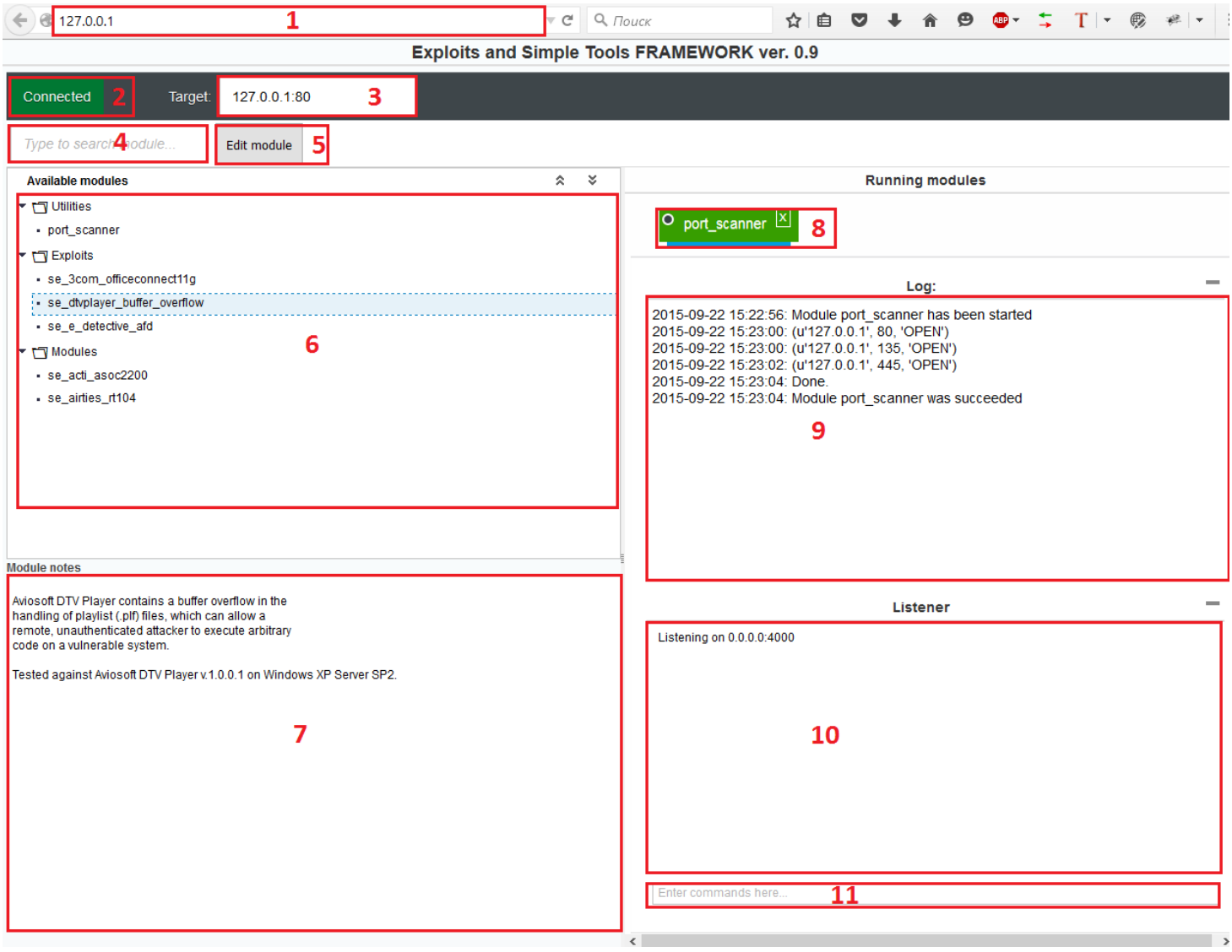
After the receiption of that command the Server collects all modules data, which contains in the INFO dict of each module.

There is a PATH key in the **Info** dict which represents module path. The modules GUI tree is generated based on that keys.

The main server script – start.py also defines framework version. All that info influences the generation of the tree.

After the initial tree generation the «restore_tabs» command is sent to already started modules (exploits or tools), listeners – so that all the information could be displayed in GUI and to make available correct interaction between Server and modules. GUI is also responsible for modules management and messages logging/displaying.

2.1. Интерфейс

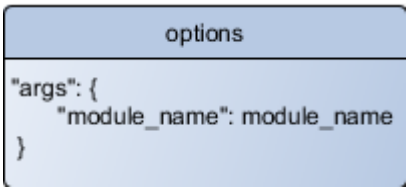


1) http address;
2) reconnect button (shows current connection state);

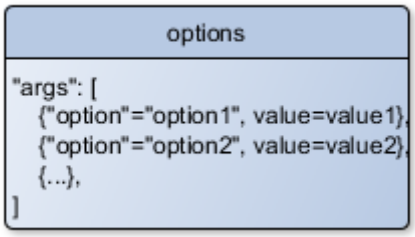
3) field for default target “host and port” used for current exploit;
4) exploits search field;
5) Button to start “on the fly” editor for chosen exploit;
6) Modules Tree;
7) Info about selected module;
8) Module Tab (with color showing the state of the module);
9) Messages received from the module;
10) Messages received from the Listener;
11) Commands to be sent to the Listener.

2.2. Modules launching.

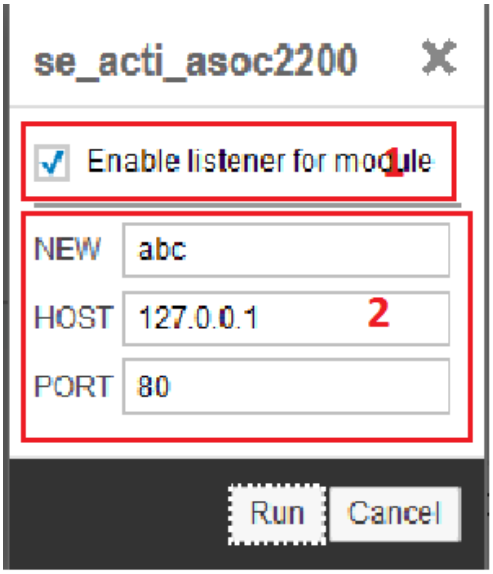
After the double clicking the module, – GUI sends the Server a query in order to receive available module options. Query structure:



module_name – module name for which we are going to receive options. Server responds with the following structure:



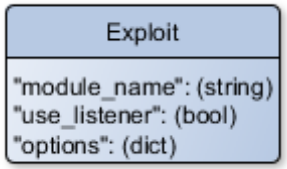
With the help of such options GUI displays appropriate dialog window (pic. 2.3)..



Pic. 2.3. Module options dialog window. 1) Checkbox for listener 2) Modules options

Exploit developers could set appropriate option types, which will be displayed as more complex dialog windows. *More info about complex option types could be found in developers section of the Manual.*

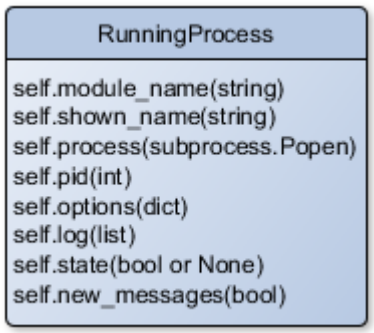
After [Run] buttton is pressed GUI sends the following command with parameters:



1. module_name, 2. isListenerEnabled , 3. options – dict with keys being option names, and values defined from GUI (pic.2.3.).

Server executes each module as a separate process, and also start listener if needed. Options are stored inside the memory of each process. Module and server always have access to stored options (see Developers Manual).

Each started module has its own Data structure (pic. 2.4). This structure could be accessed from outside the process by PID or module name:



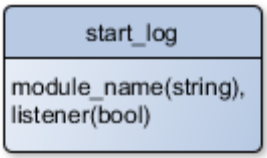
Pic. 2.4. Module data structure.

Pic 2.4.:

- *module_name* – uniqe module name, which Server sets for each launched module. In case of several instances of the same module there is a suffix «(n)», which corresponds to the number of the instances.
- *shown_name* – originally shown name.
- *process* – Process instance which is started with subprocess.Popen() It is possible to kill the process at any time.
- *pid* – PID .
- *options* – modules options to be set from GUI. These options are used by modules to define internal parameters.
- *log* – list structure. Contains ModuleMessageElement class, with 2 attributes: time and message. Examples of such class are created when module sends message to the Server.

- *state* – Current modules stage: None – keep running, True – finished succsessfully, False – failed
- *new_messages* – True if there are new messages from that module waiting to be sent to GUI; False – no new messages.

After the module start the Server sends the following command to GUI (webserver):



module_name – unique module name, listener – variable which defines whether to show Listener window in GUI or not.

After the receiption of that command GUI starts timer and then «status» command without arguments (once per 300 ms).

Server collects the data from all started modules (pic. 2.4) stores it in the from of dict, with keys being module names. If module is started with Listener, dictionary also contains info about its Listener instance.

When module Tab is closed in the GUI, Server sends command «kill_process» with «module_name» argument.

Listener – server interaction

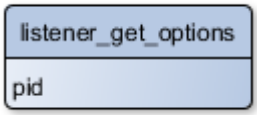
3. Listener – server interaction

Description

Listener is working like asynchronous socket server, which listens on a user defined port for incoming connections from client (shell). When connection is established, messages and data could be sent in both directions.

Listener is started when there is a command from GUI to start module with listener.

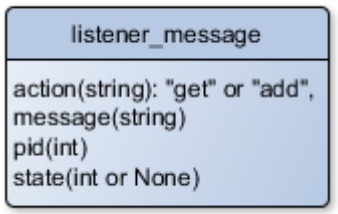
Server automatically set the port number for listener and add that info to module OPTIONS (see developers section). Listener instance, which is web socket client instance (could be several instances at once) sends the following command:



pid – PID of the listener process. Server responses to listener with options for it and then waits for connect back.

Listener universal command:

With this command listener could receive messages from GUI, transmits them to the shell instances, or vice versa from the shell instances to the Server (Server further could send them to GUI):



action – the string (“get” or “add” values): defines whether to receive or send msg from/to GUI;

message – msg for GUI (when action=”add”);

pid – PID of the listener process,

state – listener state (None – waiting for incoming connection, 1 – shell established connection, 2 – connection closed).

Listener also has protection from “short term” connections, like connections from port scanners.

4. How to write exploit modules for EAST

4. Modules (exploits) development

All modules are inherited from main class – Sploit. This class contains basic methods for interaction with a server.

4.1. Main modules writing rules:

- Module should contain INFO dictionary with the keys «NAME», «DESCRIPTION», «NOTES», – which defines the name, brief description, detailed description respectively.
- Module could contain OPTIONS dict, with appropriate user defined keys which influence GUI and could be altered from GUI .
- args(self, OPTIONS) method allows for GUI altered parameters and options import so that a user could use them in module. When module has been started with listener being autorun, – listener port could be obtained like so: listener_port = Sploit.args(self, OPTIONS)[‘listener’][‘PORT’]
- Method self.log(msg) – send messages to GUI and writes them down to text log file.
- Method self.finish(state) is used when module finished to operate… state = True, for successfull exploitation complete, state = False, when modules failed for some reason

Option type:

Depending on the option type, GUI options are displayed differently.

Simple option type could be set like: OPTION['int'] = 10 or OPTION['bool']=True.

More complex option type allowing to choose from list: OPTION['list'] = dict(options=[a,b,c], selected=c)

4.2. Auxillary classes

There are several auxillary classes which could be of use for exploit writers:

- PortScannerMT. Simple multi threaded scanner allowing to define whether port is open or closed on the remote machine

    - Shellcode generator for several OS.