Verify the Abuse Email Address of a Domain in Python

Posted on June 28, 2021

This post teaches users to develop a Python program based on WhoisXML API's email verification package, python-email-verifier that returns the valid and working abuse email address of a domain if it exists. But let's start with the basics. What is an abuse email address, exactly?

What Is an Abuse Email Address?

An abuse email address, also known as an "abuse contact address" is a means for Internet users to let your organization know that you've been sending spam or even malicious messages. It usually takes the form abuse@domain[.]tld, following the standard set by RFC 2142.

Think of it as a suggestion box in your store where customers and prospects can leave you complaints or messages suggesting product/service improvements.

Who Needs an Abuse Email Address?

Practically any company that employs email marketing needs an abuse email address. Why? Because threat actors are always on the lookout for email accounts to hack and send malicious messages with. If the problem is not dealt with, your email domain would likely be flagged by email service providers (ESPs) and Internet service providers (ISPs) as a spammer. That would disallow anyone in your organization from sending messages to any Internet user.

An abuse email address would help in that users can contact you in case you've been sending spam or, worse, malicious messages. Think of it this way: When you receive illicit emails from a specific domain, which could happen to any user, you'll want to complain to the domain owner who may not even be aware of the activity, wouldn't you? You can do that by sending a message to the domain owner's abuse email address address.

Why Verify an Abuse Email Address and How Do You Do So in Python?

The Internet Engineering Task Force (IETF) created RFC 2142 for a reason—to ensure complaints about a particular domain get to intended recipients. The standard has this to say about an abuse email address:

"For well-known names that are not related to specific protocols, only the organization's top-level domain (TLD) name is required to be valid. For example, if an ISP's domain name is COMPANY.COM, then the <ABUSE@COMPANY.COM> address must be valid and supported, even though the customers whose activity generates complaints use hosts with more specific domain names like SHELL1.COMPANY.COM. Note, however, that it is valid and encouraged to support mailbox names for subdomains, as appropriate. Mailbox names must be recognized independent of character case."

It's clear then that the domain domain[.]tld should be valid to make the email address abuse@domain[.]tld exist and be able to receive messages. Moreover, if the correspondence comes from a subdomain like badguy@subdomain[.]domain[.]tld, then the standard address abuse@subdomain[.]domain[.]tld may exist and be more important for the case. (Take, for instance, a university with multiple faculties. Often, each faculty has a subdomain under the university's domain, which they use for correspondence.) Hence, a good strategy would be to start with the lowest-level domain and proceed to the highest level if the abuse address does not exist. Let us proceed to the implementation, which requires lower to intermediate Python skills. We begin by installing two packages with Python's package manager, pip:

```
pip install tld email-verifier
```

The first one, tld, is able to get the TLD from a domain name. We need this because of second- or lower-level domains in certain country-code TLDs (ccTLDs) that act as TLDs. For instance, when dealing with abuse@department[.]company[.]co[.]uk, we don't want to check abuse@co[.]uk.

The email-verifier package mentioned above was made by WhoisXML API. It can validate an email address against several aspects. In particular, we want to know if the domain of the queried abuse email address resolves in the Domain Name System (DNS) and can receive messages via the Simple Mail Transfer Protocol (SMTP). We can also check for the abuse email address's corresponding mail servers, if it belongs to a free ESP, and if it is disposable, although these aspects are not relevant to this case.

The email-verifier package uses Email verification API in the background, so to use it, we need an API key. Visit the API page to register for 1,000 queries a month free of charge or more at reasonable prices. After registering, you will receive an API key that you will need to get our program working.

Armed with all the necessary ingredients, let us just jump directly to the code of our little program, which we will call “get_abuse_email.py”:

```
#!/usr/bin/env python3

import sys

from tld import get_tld

from emailverifier import Client
from emailverifier import exceptions

API_KEY = 'YOUR_API_KEY'

client = Client(API_KEY)

domain = sys.argv[1].strip().split('@')[-1]
```

```
tld_level = len(get_tld(domain, fix_protocol=True).split("."))

domain_elements =  domain.split('.')

found = 1
for level in range(0, len(domain_elements) - tld_level):

    this_level_address = "abuse@" + "".join(
[c + "." for c in domain_elements[level:]]).strip(".")

    verification = client.get(this_level_address)

    if verification.dns_check and verification.smtp_check:
print(this_level_address)
found = 0
break

sys.exit(found)
```

The API_KEY variable should hold your actual API key. To make the code work, replace the string “YOUR_API_KEY” with it. Let’s see how it works.

- The variable domain holds the domain name. The program accepts domain names as well as email addresses as positional arguments. In the latter case, we get rid of the account name and the @ character.

- TLD level determines the level of the TLD, for instance, it will be 1 for something[.]com and 2 for something[.]co[.]uk.

- domain_elements refers to a list of the parts of the domain name, such as ['something','com'].

- found refers to the return value of the program for possible use in a shell script, for example.

It will be set to 0 if we find a valid address.

- In the main loop, we go through the possible email addresses. In the case of badguys[.]evildomain[.]co[.]uk, we will try abuse@badguys[.]evildomain[.]co[.]uk and abuse@evildomain[.]co[.]uk. If the former works, it is more specific, whereas the second one should work if RFC 2142 was followed.

- The verification variable holds an object that stores the result of the actual address validation.

- If the domain name of the email address resolves in the DNS and it does accept messages sent via SMTP (i.e., verification.dns_check and verification.smtp_check is true) we are done. We report the address, set the return code to 0, and break the loop.

- If no valid abuse email address is found, the program will not return anything and terminate with the return code 1.

Let’s see how it works. The examples will be in bash or zsh, and $ indicates the input prompt. Checking a proper abuse email address will return this:

```
$ ./get_abuse_email.py support@whoisxmlapi.com
abuse@whoisxmlapi.com
$ echo $?
0
```

Checking a valid abuse email domain will give this:

```
$./get_abuse_email.py drs.whoisxmlapi.com
abuse@whoisxmlapi.com
$ echo $?
0
```

The subdomain does not have a specific abuse email address since it is not used for mailing.

Finally, a domain from the author’s spam folder where a definitely unwanted message was received would return this:

```
$./get_abuse_email.py dinaf.gob.hn
$ echo $?
1
```

From the example, it appears that the email addresses do not only send unwanted messages but also violate RFC 2142 with respect to abuse email address formatting. Is there a good reason for accepting emails from them?

In conclusion, we implemented a simple Python program to validate the abuse email address of a domain. It can be used interactively or, for instance, as a part of an automated abuse sending system. It will also detect and indicate with particular return codes if the domain violates RFC 2142 by not providing a working abuse email address. Unfortunately, implicit violations of RFC 2142 are not easy to detect. By “implicit violations,” we mean ignoring emails sent to the address or sending automated replies with links to web forms for abuse reporting—typically by some big providers today. Nevertheless, abuse complaints can help a lot for organizations to run a domain responsively, and the utility we created can be helpful when sending abuse email notifications.