Verify the abuse email address of a domain in Python

Posted on June 28, 2021

In what follows, we'll develop a small Python program based on WhoisXML API's email verification package, python-email-verifier that returns the valid and working abuse e-mail of an Internet domain if it exists.

Assume that you have received an illicit e-mail from a given domain. This may well happen to you as a user, but it is even more vital if you are actually operating a web or mail server and the received emails affect your users. Certainly, you will want to complain to the domain owner (who may not even be aware of the unwanted activity, so your complaint can be very helpful).

The regular way of such a notification is to send an email to the standard address abuse@domain.tld. According to RFC2142,

"For well-known names that are not related to specific protocols, only the organization's top-level domain name is required to be valid. For example, if an Internet service provider'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 sub-domains, as appropriate. Mailbox names must be recognized independent of character case."

Clearly, for the domain domain.tld, the e-mail address abuse@domain.tld has to exist and be able to receive emails. Moreover, if the correspondence comes from a subdomain, like badguy@subdomain.domain.tld, then the standard address abuse@subdomain.domain.tld may exist and might be more important for the case. (Take, for instance, a university with multiple faculties. Frequently the faculties have a subdomain under the University's domain and they use it for correspondence.) Hence, a good strategy would be to start with the lowest-level domain and proceed to the higher level if the abuse address does not exist.

Let us proceed to the implementation, which requires lower intermediate Python skills. We begin with installing two packages with Python's package manager, pip:

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

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

The email-verifier is the aforementioned package by WhoisXML API. It is able to validate an email address against several aspects. In particular, we want to know whether the domain of the checked e-mail address resolves in the DNS and whether the address itself can receive messages on SMTP. (We could also check which are the corresponding mail servers, whether the mail address belongs to a free email provider, or whether it is a disposable e-mail, but these aspects are not relevant in our case.)

The email-verifier package uses the Email verification API in the background, so to use it we need an API key. Visit the API's page to register for 1,000 queries a month for free, or for more queries at a reasonable price. Having registered, you will receive an API key that will be needed to get our program working.

Armed with all the necessary ingredients, let us just jump directly to the code of our little program, 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, so to make it 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 e-mail addresses as a positional argument. 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 is a list of the parts of the domain name, e.g. ['something','com']
- found is the return value of the program for possible use in e.g. a shell script. It will be set to 0 if we find a valid address.
- In the main loop, we go through the possible email addresses, e.g. in 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 the RFC is respected.
- The "verification" variable holds an object which stores the result of the actual address validation.
- If the domain name in the mail address resolves in DNS and it does accept emails 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 address is found, the program will not return anything and will terminate with the return code 1.

Let's see how it works. The examples will be in bash or zsh, and the "$" indicates the input prompt Checking a proper mail address:

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

or a domain:

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

(This subdomain does not have a specific abuse address as it is not used for mailing.) And finally, a domain from the spam folder of the author, from which a definitely unwanted mail had been received:

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

It appears that they do not only send unwanted mails, but also violate the RFC with respect to the abuse email address. Is there a good reason for accepting mail from them?

In conclusion, we have implemented a small Python program to get the abuse email address for a domain or an e-mail address. It can be used interactively or, for instance, as a part of an automated abuse sending system. It will also detect, and indicate by the return code, if the domain violates RFC2142 by not providing a working abuse address. Unfortunately, the implicit violations of this RFC are not so easy to detect. By "implicit violations' ' we mean the ignoring of the mails

sent to the address, or the sending of automated replies with links to web forms for abuse reporting - a way of ignoring the RFC followed even by some big providers nowadays. Nevertheless, abuse complaints can help a lot in running a domain responsively, and the present utility can be helpful when sending them.