Using ec2 instances as sneaker bid bots pt 3.

Date: 2023-11-28
By: csr13

Download as PDF

For part three.

- How to be PCI compliant when storing credit card information on your premises.
- How to scrape nike products via react store 'storage'.

How to be PCI compliant for any app

Code to use when storing credit card information, or any other information, for example, in healtcare, patient confidential data, to be in compliance with HIPPA standards.

```
import os
import json

from cryptography.fernet import Fernet
from aws.loader import AwsSecretLoader


class CreditCardMixin(object):

    @classmethod
    def encrypt_cc_data(cls, data):
        key = None
        key_loader = None

        try:
            key_loader = AwsSecretLoader()
            key = json.loads(key_loader.get_aws_secret("FERNET_KEY"))
            key = key["FERNET_KEY"]
        except:
            ########################################################################
            # Double check if the key is in the global server environment as variable
            # This is insecure and should raise
            ########################################################################
            key = os.getenv("FERNET_KEY")[1:]

        key = key.encode()
        fernet = Fernet(key)
        cc_expiry = f"{data['cc_expiry'].month}/{data['cc_expiry'].year}"
        data = dict(
            cc_name=fernet.encrypt(data["cc_name"].encode()),
            cc_number=fernet.encrypt(data["cc_number"].encode()),
            cc_code=fernet.encrypt(data["cc_code"].encode()),
            cc_expiry=fernet.encrypt(cc_expiry.encode()),
        )
        return data

    @classmethod
    def decrypt(cls, value):
        key_loader = AwsSecretLoader()
        try:
            key: str = key_loader.get_aws_secret("WRAPPER")
            key: dict = json.loads(key)
            key: str = key["FERNET_KEY"]
        except:
            return ""

        key: bytes = key.encode()
        fernet = Fernet(key)

        try:
            value = fernet.decrypt(value).decode("utf-8")
        except Exception as error:
            Logger.exception(str(error))
            return ""
        return value
```

Instead of writing a thousand line selenium (browser automation, guided by selecting elements, classes, id's, or xpaths) we only really need one regex. I converted this into a microservice that I can write a caller executed inside a cron job that fetches me new nike products, from the release soon page daily.

I am using sanic instead of Flask, I like sanic, the logo got me (a cheap drawing of Sonic, but it's very fast).

```python
import json
import io
import random
import re

import requests
from sanic import Sanic
from sanic.response import json as json_response

app = Sanic(__name__)
config = {"host": "127.0.0.1", "port": 8080, "debug": True}

##############################################################
# We only need this compiled regular expression which
# matches the window state, well in this case they use redux
# a shitty state management library that has big gaping holes.
##############################################################

ps_re = re.compile(r"window.__PRELOADED_STATE__\s=\s(\{.*?\});")


##############################################################
# load a random user agent, for the request.
##############################################################

def _get_random_agent():
    with io.open("user-agents.txt") as ts:
        agents = [_.strip("/n") for _ in ts.readlines()]
        agent = random.choice(agents).strip("\n")
    return agent


@app.route("/random-agent", methods=("GET",))
def get_random_agent(request):
    with io.open("user-agents.txt") as ts:
        agents = [_.strip("/n") for _ in ts.readlines()]
        agent = random.choice(agents).strip("\n")
    body = {"status": "success", "message": agent}
    return json_response(body=body, status=200)

####################################################################
# we could use my docker tor proxy to trick nike.com so their bot
# detection libraries won't blacklist the origin of my IP,
# which is a good and basic technique.
# works every time. for now, one daily request won't block us,
# since this is not a browser bot, just an http request.
###################################################################

@app.route("/nike-products")
async def nike_releases(request):
    data = {}
    url = "https://www.nike.com/launch"
    try:
        headers = {"User-Agent": _get_random_agent()}
        chunk = requests.get(url, headers)
        target = json.loads(ps_re.search(chunk.text).group(1))
    except Exception as error:
        data.update({"status": "error", "message": str(error)})
        return json_response(body=data, status=400)
    data.update({"status": "success", "message": target})
    return json_response(body=data, status=200)


if __name__ == "__main__":
    app.run(**config)
```

That's it, this is the end of the series, I won't write how to parse and traverse a store, that is the data anlyst job, not mine. Needless to say, you do need to write a selenium/puppeter bot that makes the purchase, the checkout.service on the t2.micro/ec2 instance.

I personally found some github code, and hacked it to fit my purpose, since I had to do encrypt and decrypt credit card information (because it travels through the wire), and write custom endpoints that my main command and ontrol server could hit with the purchase information of user, because this app operates not in this fashion. It instead loads the info unencrypted, since it never leaves the codebase and I ended up adding 1000 + lines of node.js code.

Here is the base purchase component repository. https://github.com/samc621/SneakerBot, it works well, developer updates regularly, and has great documentation, the developers added many other stores other than nike.

This post is part of a series, check out the other parts of this series of notes

Using ec2 instances as sneaker bid bots pt 2.

Using ec2 instances as sneaker bid bots pt 1.

Related Notes

1) Using ec2 instances as sneaker bid bots pt 2.
Download PDF
Date published: 2023-11-27
bots python aws series

2) Whatsapp chatbot with Python and Twilio
Download PDF
Date published: 2023-11-15
bots python whatsapp business

3) Using ec2 instances as sneaker bid bots pt 1.
Download PDF
Date published: 2023-11-21
bots python aws series

4) Real Time Language Translation Agent System for Call Centers
Download PDF
Date published: 2023-11-16
voip telephony python systems

5) Using ec2 instances as sneaker bid bots pt 2.
Download PDF
Date published: 2023-11-27
bots python aws series

6) Whatsapp chatbot with Python and Twilio
Download PDF
Date published: 2023-11-15
bots python whatsapp business

7) Backend Celery task manager dashboard via Flower
Download PDF
Date published: 2023-10-29
backend tasks python

8) Using ec2 instances as sneaker bid bots pt 1.
Download PDF
Date published: 2023-11-21
bots python aws series

9) Using ec2 instances as sneaker bid bots pt 2.
Download PDF
Date published: 2023-11-27
bots python aws series

10) Using ec2 instances as sneaker bid bots pt 1.
Download PDF
Date published: 2023-11-21
bots python aws series

11) Using ec2 instances as sneaker bid bots pt 2.
Download PDF
Date published: 2023-11-27
bots python aws series

12) Using ec2 instances as sneaker bid bots pt 1.
Download PDF
Date published: 2023-11-21
bots python aws series