BUECE

Boston University

Electrical and Computer Engineering

EC521 -- Cyber Security Project -- Final Report

QR Code Security

$100
AMAZON GIFT
CARD GIVEAWAY!
FILL OUT
SURVEY TO
WIN:

Team Members:

George Kent-Scheller, georgeks@bu.edu
Sam Krasnoff, krasnoff@bu.edu
John Kircher, kircherj@bu.edu
Julian Padgett, jpadgett@bu.edu

For access to the Github repository, please contact: georgetks@gmail.com

Description

QR codes are a machine-readable code consisting of an array of black and white squares, typically used for storing URLs or other information for reading by the camera on a smartphone. Since the beginning of the COVID-19 Pandemic, usage of QR codes has increased to eliminate the need for physical human interaction and reduce the risk of infection. Today, QR codes have become ubiquitous in many facets of modern life. However, QR codes are not very safe or secure to use. Phishing attacks can be executed by getting victims to scan QR codes in the same way that an attacker could get a victim to click on a dangerous link. While many people are educated on the importance of not clicking on suspicious links, there seems to be less public awareness of the importance of not scanning suspicious QR codes.

Approach / Measurement

In order to understand the extent of the problem, our team created several fake flyers with our own QR codes, and posted over 80 of them around the BU campus and surrounding areas. These flyers included a sign-up form for a nonexistent basketball tournament, a fake Amazon gift card giveaway, and free tickets to a fictional nightclub event.

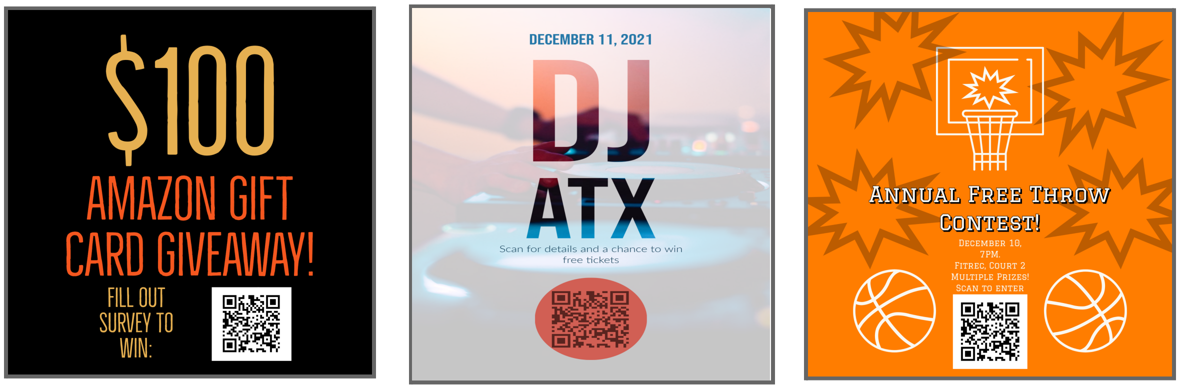


Figure 1: This image depicts our Amazon, Party, and Basketball QR code flyers

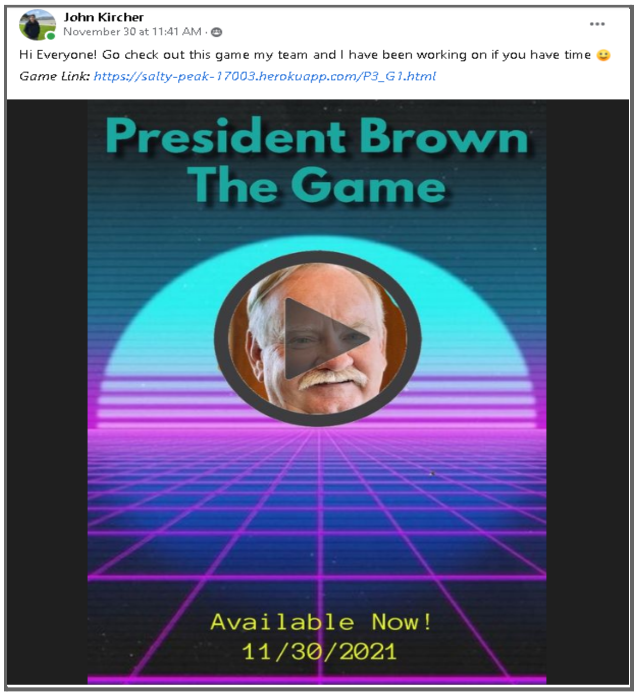
Over one week, 55 people scanned the Amazon flyers, 10 people scanned the basketball flyers, and 5 people scanned the club flyers. After scanning the QR codes attached to the flyers, multiple steps were taken to redirect users and record their data:

1. After scanning a QR code, a javascript program accessed data from the user including:
   a. Google Analytics Cookies, IP, QR Code Scanned, Device, Operating System, Search Engine, and Time.
2. Users are then redirected to a website hosted using Heroku. This website included a brief description of our project, as well as the ability to fill out a survey about how they accessed the QR code, their awareness and understanding of cyber security, and the ability to leave any notes or thoughts about the experiment. (A view of our website can be seen in Figure 9, page 8)

3. POST requests are made to another javascript program (app.js) with all of the collected data.

4. This data is then added as a document to a collection in a FireStore database. Furthermore, if users filled out the survey via the web page, their data was also added to another collection in FireStore.

We also decided to add a control to our data collection. This control was a fake game-link that was shared via Facebook student groups, and Reddit student groups.


John Kircher
November 30 at 11:41 AM ·

Hi Everyone! Go check out this game my team and I have been working on if you have time 😀
Game Link: https://salty-peak-17003.herokuapp.com/P3_G1.html

Figure 2: This image depicts our President Brown The Game advertisement

The link was supposed to act as a control to see how many more people were susceptible to QR codes over random "sketchy" links. However, as seen from our data in the next section, more people clicked on the link than scanned QR codes in total. Ultimately, we also included the data in our research because it still exposes the vulnerability that people will scan/click on images and codes with no thought to security risks.

Results

In total, we received 184 documents through our FireStore database. These included 70 (38%) from various QR codes, and 114 (62%) from the game link. 78.5% of the total scans were from Amazon, 14.3% from the basketball flyers, and the final 7.2% from the party fliers. Histograms showing the distribution of data are shown below in

Figure 3. This data allows us to draw numerous conclusions. Among the flyers, individuals were much more likely to scan something with immediate monetary gain, rather than arbitrary events that may just be interesting to the individual. We were genuinely surprised, as the Amazon flyer was by far the least official looking. In terms of carrier/IP breakdown, 44% of the data came from users directly on the BU network. The university has sent out numerous emails warning users to be wary of suspicious links, and other connections to digital mediums, like QR codes, which were clearly ignored. The rest came from various LTE carriers spread evenly over most of the major US companies. The most traction we got was from the game link, posted in a private Facebook group and on BU reddit. Though the post did seem genuine, social media accounts are often hacked and exploited, and users should always be careful of clicking on unknown links, as there was a clearly fake hyperlink. Overall, this experiment shows a consistent lack of good digital security practices, and that to protect the lowest knowledge user, there must be a secure alternative.

Furthermore, we also write a python script accessing the xlwt library to get all of our data from FireStore and write it to an Excel sheet for a more simple analysis. An example/screenshot of this sheet can be seen in figure 4.

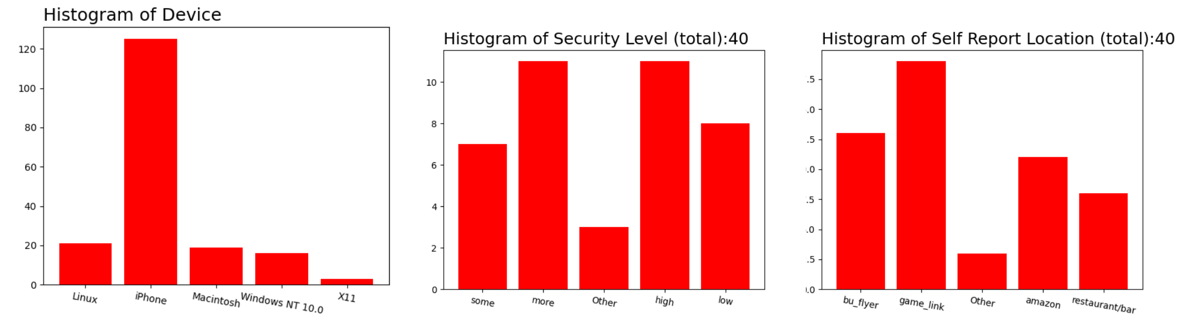
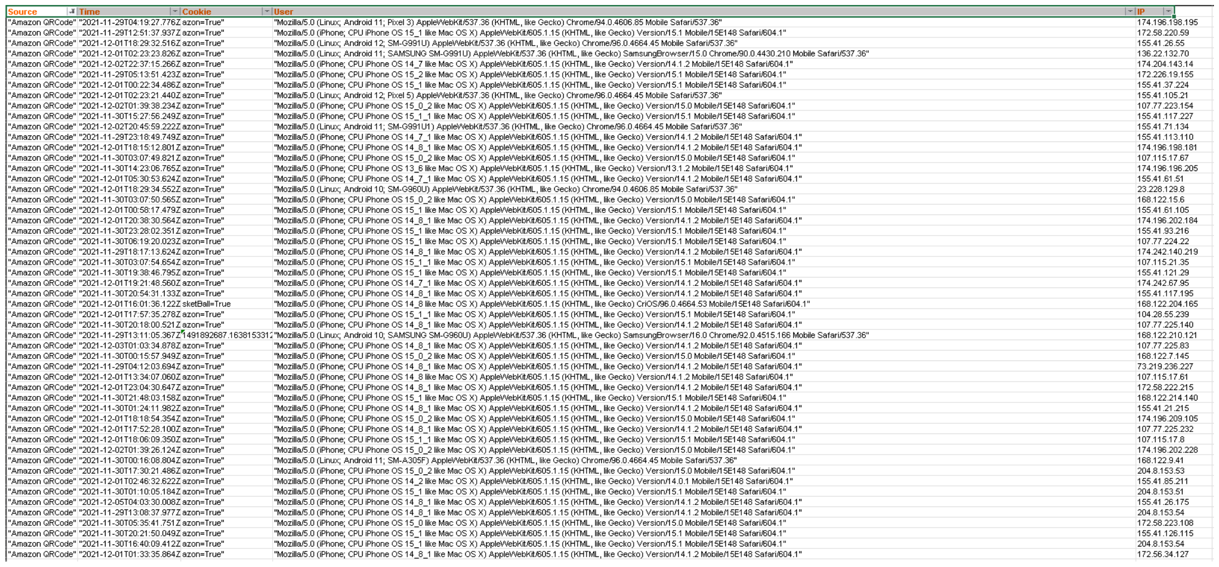


Figure 3: These graphs represent data collected from individuals who scanned the QR codes and visited our website



Source	Time	Cookie	User	IP
"Amazon QRCode"	"2021-11-29T04:19:27.776Z	azon=True"	"Mozilla/5.0 (Linux; Android 11; Pixel 3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.85 Mobile Safari/537.36"	174.196.198.195
"Amazon QRCode"	"2021-11-29T12:51:37.937Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 15_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/15.1 Mobile/15E148 Safari/604.1"	172.58.220.59
"Amazon QRCode"	"2021-12-01T18:29:32.516Z	azon=True"	"Mozilla/5.0 (Linux; Android 12; SM-G991U) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.45 Mobile Safari/537.36"	155.41.26.55
"Amazon QRCode"	"2021-12-01T02:23:23.826Z	azon=True"	"Mozilla/5.0 (Linux; Android 11; SAMSUNG SM-G991U) AppleWebKit/537.36 (KHTML, like Gecko) SamsungBrowser/15.0 Chrome/90.0.4430.210 Mobile Safari/537.36"	136.22.132.70
"Amazon QRCode"	"2021-12-02T22:37:15.266Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 14_7 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.1.2 Mobile/15E148 Safari/604.1"	174.204.143.14
"Amazon QRCode"	"2021-11-29T05:13:51.423Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 15_2 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/15.1 Mobile/15E148 Safari/604.1"	172.226.19.155
"Amazon QRCode"	"2021-12-01T00:22:34.486Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 15_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/15.1 Mobile/15E148 Safari/604.1"	155.41.37.224
"Amazon QRCode"	"2021-12-01T02:23:21.440Z	azon=True"	"Mozilla/5.0 (Linux; Android 12; Pixel 5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.45 Mobile Safari/537.36"	155.41.105.21
"Amazon QRCode"	"2021-12-02T01:39:38.234Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 15_0_2 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/15.0 Mobile/15E148 Safari/604.1"	107.77.223.154
"Amazon QRCode"	"2021-11-30T15:27:56.249Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 15_1_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/15.1 Mobile/15E148 Safari/604.1"	155.41.117.227
"Amazon QRCode"	"2021-12-02T20:45:59.222Z	azon=True"	"Mozilla/5.0 (Linux; Android 11; SM-G991U1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.45 Mobile Safari/537.36"	155.41.71.134
"Amazon QRCode"	"2021-11-29T23:18:49.749Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 14_7_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.1.2 Mobile/15E148 Safari/604.1"	155.41.113.110
"Amazon QRCode"	"2021-12-01T18:15:12.801Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 14_8_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.1.2 Mobile/15E148 Safari/604.1"	174.196.198.181
"Amazon QRCode"	"2021-11-30T03:07:49.821Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 15_0_2 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/15.0 Mobile/15E148 Safari/604.1"	107.115.17.67
"Amazon QRCode"	"2021-11-30T14:23:06.765Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 13_6 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/13.1.2 Mobile/15E148 Safari/604.1"	174.196.196.205
"Amazon QRCode"	"2021-12-01T05:30:53.624Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 14_7_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.1.2 Mobile/15E148 Safari/604.1"	155.41.61.51
"Amazon QRCode"	"2021-12-01T18:29:34.552Z	azon=True"	"Mozilla/5.0 (Linux; Android 10; SM-G960U) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.85 Mobile Safari/537.36"	23.228.129.8
"Amazon QRCode"	"2021-11-30T03:07:50.565Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 15_0_2 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/15.0 Mobile/15E148 Safari/604.1"	168.122.15.6
"Amazon QRCode"	"2021-12-01T00:58:17.479Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 15_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/15.1 Mobile/15E148 Safari/604.1"	155.41.61.105
"Amazon QRCode"	"2021-12-01T20:38:30.564Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 14_8_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.1.2 Mobile/15E148 Safari/604.1"	174.196.202.184
"Amazon QRCode"	"2021-11-30T23:28:02.351Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 15_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/15.1 Mobile/15E148 Safari/604.1"	155.41.93.216
"Amazon QRCode"	"2021-11-30T06:19:20.023Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 15_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/15.1 Mobile/15E148 Safari/604.1"	107.77.224.22
"Amazon QRCode"	"2021-11-29T18:17:13.624Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 14_8_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.1.2 Mobile/15E148 Safari/604.1"	174.242.140.219
"Amazon QRCode"	"2021-11-30T03:07:54.654Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 15_1_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/15.1 Mobile/15E148 Safari/604.1"	107.115.21.35
"Amazon QRCode"	"2021-11-30T19:38:46.795Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 15_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/15.1 Mobile/15E148 Safari/604.1"	155.41.121.29
"Amazon QRCode"	"2021-12-01T19:21:48.560Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 14_7_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.1.2 Mobile/15E148 Safari/604.1"	174.242.67.95
"Amazon QRCode"	"2021-11-30T20:54:31.133Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 14_8_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.1.2 Mobile/15E148 Safari/604.1"	155.41.117.195
"Amazon QRCode"	"2021-12-01T16:01:36.122Z	sketBall=True	"Mozilla/5.0 (iPhone; CPU iPhone OS 14_8 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) CriOS/96.0.4664.53 Mobile/15E148 Safari/604.1"	168.122.204.165
"Amazon QRCode"	"2021-12-01T17:57:35.278Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 15_1_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/15.1 Mobile/15E148 Safari/604.1"	104.28.55.239
"Amazon QRCode"	"2021-11-30T20:18:00.521Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 14_8_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.1.2 Mobile/15E148 Safari/604.1"	107.77.225.140
"Amazon QRCode"	"2021-11-29T13:11:05.367Z	1491892687.1638153312	"Mozilla/5.0 (Linux; Android 10; SAMSUNG SM-G960U) AppleWebKit/537.36 (KHTML, like Gecko) SamsungBrowser/16.0 Chrome/92.0.4515.166 Mobile Safari/537.36"	168.122.210.121
"Amazon QRCode"	"2021-12-03T01:03:34.878Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 14_8_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.1.2 Mobile/15E148 Safari/604.1"	107.77.225.83
"Amazon QRCode"	"2021-11-30T00:15:57.949Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 15_0_2 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/15.0 Mobile/15E148 Safari/604.1"	168.122.7.145
"Amazon QRCode"	"2021-11-29T04:12:03.694Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 14_8_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.1.2 Mobile/15E148 Safari/604.1"	73.219.236.227
"Amazon QRCode"	"2021-12-01T13:34:07.060Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 14_8 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.1.2 Mobile/15E148 Safari/604.1"	107.115.17.61
"Amazon QRCode"	"2021-12-01T23:04:30.647Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 14_8_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.1.2 Mobile/15E148 Safari/604.1"	172.58.222.215
"Amazon QRCode"	"2021-11-30T21:48:03.158Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 15_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/15.1 Mobile/15E148 Safari/604.1"	168.122.214.140
"Amazon QRCode"	"2021-11-30T01:24:11.982Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 14_8_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.1.2 Mobile/15E148 Safari/604.1"	155.41.21.215
"Amazon QRCode"	"2021-12-01T18:18:54.354Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 15_0_2 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/15.0 Mobile/15E148 Safari/604.1"	174.196.209.105
"Amazon QRCode"	"2021-12-01T17:52:28.100Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 14_8_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.1.2 Mobile/15E148 Safari/604.1"	107.77.225.232
"Amazon QRCode"	"2021-12-01T18:06:09.350Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 15_1_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/15.1 Mobile/15E148 Safari/604.1"	107.115.17.8
"Amazon QRCode"	"2021-12-02T01:39:26.124Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 15_0_2 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/15.0 Mobile/15E148 Safari/604.1"	174.196.202.228
"Amazon QRCode"	"2021-11-30T00:16:08.804Z	azon=True"	"Mozilla/5.0 (Linux; Android 11; SM-A305F) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.45 Mobile Safari/537.36"	168.122.9.41
"Amazon QRCode"	"2021-11-30T17:30:21.486Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 15_0_2 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/15.0 Mobile/15E148 Safari/604.1"	204.8.153.53
"Amazon QRCode"	"2021-12-01T02:46:32.622Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 14_2 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.0.1 Mobile/15E148 Safari/604.1"	155.41.85.211
"Amazon QRCode"	"2021-11-30T01:10:05.184Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 15_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/15.1 Mobile/15E148 Safari/604.1"	204.8.153.51
"Amazon QRCode"	"2021-12-05T04:03:30.008Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 14_8_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.1.2 Mobile/15E148 Safari/604.1"	155.41.26.175
"Amazon QRCode"	"2021-11-29T13:08:37.977Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 14_8_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.1.2 Mobile/15E148 Safari/604.1"	204.8.153.54
"Amazon QRCode"	"2021-11-30T05:35:41.751Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 15_0 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/15.0 Mobile/15E148 Safari/604.1"	172.58.223.108
"Amazon QRCode"	"2021-11-30T20:21:50.049Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 15_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/15.1 Mobile/15E148 Safari/604.1"	155.41.126.115
"Amazon QRCode"	"2021-11-30T16:40:09.412Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 15_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/15.1 Mobile/15E148 Safari/604.1"	204.8.153.54
"Amazon QRCode"	"2021-12-01T01:33:35.864Z	azon=True"	"Mozilla/5.0 (iPhone; CPU iPhone OS 14_8_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.1.2 Mobile/15E148 Safari/604.1"	172.56.34.127

Figure 4: This is our Excel Sheet holding all of our data parsed from JSON to a list in python

While we did receive favorable results from our experiment, we could have had many more scans had we placed QR codes in more vulnerable locations. For example, if we replaced restaurant menu QR codes with our own, or replaced official BU QR codes with our own. However, we did not want to intrude or adversely affect businesses or do anything illegal. Finally, we also received various forms of feedback through our survey that was included on the website. Though we had many responses (40 total, as seen from histograms in Figure 3), only 15 included their thoughts which can be seen below in Figure 5. These ranged from helpful comments to some of our peers expressing their surprise and frustration with the experiment.

Other: "rip my tuition"
Game_link: "Did someone hack me"
Bu_flyer: "nice"
Amazon: "Really interesting, when I saw this I pop up I was like I've never rlly thought about that"
Amazon: "False advertising bullshit"
Game_link: "This is kind of a stupid experiment. Why would I expect a phasing scam on a student Facebook group with user authentication required to enter?"
restaurant/bar: "Okay Yes"
Bu_flyer: "I have client sided protection that would halt and alert before anything was executed or downloaded."

Amazon: “Hmm”
Game_link: “Interesting”
Game_link: “I love it!”
Bu_flyer: “Cool. Good way to see how dumb people are (clearly I am lol)”
Amazon: “Convincing”
restaurant/bar: “Yes”
Bu_flyer: “this exp gave me a descent scare. its a good way of spreading important information”

Figure 5: This is a table showing 15 written responses received from our survey

Mitigation

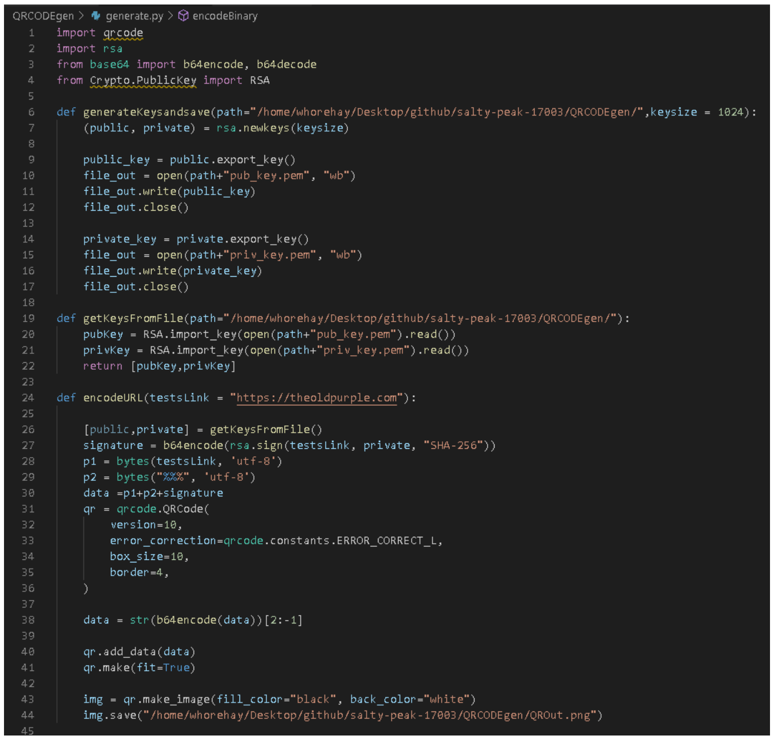
Our primary mitigation was the website. Although campus IT routinely sends out emails warning people about specific security risks, our project proves that people still don't understand the dangers of blindly clicking or scanning. This website not only helped us see what people were most likely to scan but also was meant as a prototype for how an I.T. website could work. If there are more fake websites that exist to demonstrate security vulnerabilities without actually exploiting them, we believe that it would improve security awareness. Many of our survey comments reflected peoples appreciation of the learning experience.

The secondary purpose of our project was to make QR codes more secure. Our solution is to generalize https. We believed that this security standard would help preserve the power of QR codes. One of the really cool things about QR codes is their ability to store raw bytes. This meant that we wanted to write a program to sign the actual QR codes rather than just checking for https after the scan. However there is a big issue with our mitigation.

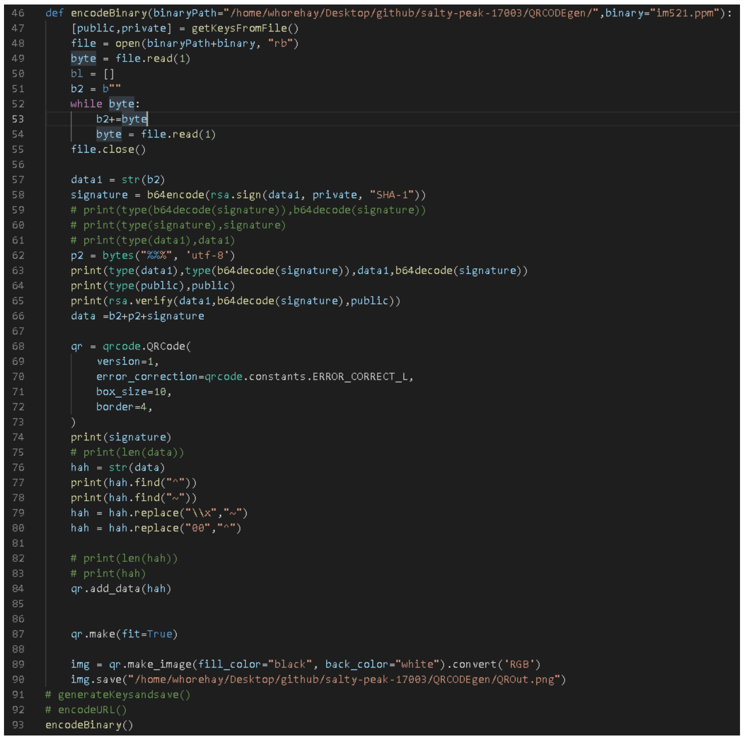
Although it is not well documented, there is a lack of standards in QR code scanners. Many lower quality QR code scanner libraries are not able to scan the largest types of QR codes. This is an issue because the ability to write programs, music files, or images to a QR code cannot take full advantage of the 2953 bytes offered by the version 40 QR codes. This makes the space on the QR code at premium. In our example, we minimize space used by utilizing SHA1 hashing for our signing. The user could increase security by writing a more robust QR code scanning library that supports the largest sized QR codes. They could also implement a system where they have a signature on one QR code and use the other for data only.

For our version we used RSA signing on a SHA1 hash of the data. We then appended the signature to the end of the data that we wanted to put on the QR code. The data was then turned into a QR code and scanned by the user. The QR code was separated into data and signature sections. The signature was then verified against the data with a public key. There would be a certificate authority who would be able to sign important QR codes. Additionally there would be the option for one party to provide non-repudiability of data on a QR code to a separate party. Figure 7 (below) shows prototypes of the secure QR code, with an embedded hash, but no authority to reference. In a proper implementation, the user would scan, confirm trust, and then forward the user to their data. If it was an untrusted source, an intermediate page would interrupt the process, similar to an HTTP (non-secure) link on a desktop browser.

As the images show below, we were able to make a decently robust proof of concept QR code generator. In Python, we used a QR code package downloaded through PIP, as well as the built in crypto library. Running the main generator function will take the inputted URL/information and append a SHA1 hash of a trusted source.

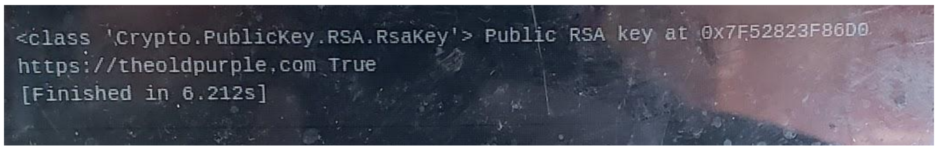


```python
import qrcode
import rsa
from base64 import b64encode, b64decode
from Crypto.PublicKey import RSA

def generateKeysandsave(path="/home/whorehay/Desktop/github/salty-peak-17003/QRCODEgen/",keysize = 1024):
    (public, private) = rsa.newkeys(keysize)

    public_key = public.export_key()
    file_out = open(path+"pub_key.pem", "wb")
    file_out.write(public_key)
    file_out.close()

    private_key = private.export_key()
    file_out = open(path+"priv_key.pem", "wb")
    file_out.write(private_key)
    file_out.close()

def getKeysFromFile(path="/home/whorehay/Desktop/github/salty-peak-17003/QRCODEgen/"):
    pubKey = RSA.import_key(open(path+"pub_key.pem").read())
    privKey = RSA.import_key(open(path+"priv_key.pem").read())
    return [pubKey,privKey]

def encodeURL(testsLink = "https://theoldpurple.com"):

    [public,private] = getKeysFromFile()
    signature = b64encode(rsa.sign(testsLink, private, "SHA-256"))
    p1 = bytes(testsLink, 'utf-8')
    p2 = bytes("%%%", 'utf-8')
    data =p1+p2+signature
    qr = qrcode.QRCode(
        version=10,
        error_correction=qrcode.constants.ERROR_CORRECT_L,
        box_size=10,
        border=4,
    )

    data = str(b64encode(data))[2:-1]

    qr.add_data(data)
    qr.make(fit=True)

    img = qr.make_image(fill_color="black", back_color="white")
    img.save("/home/whorehay/Desktop/github/salty-peak-17003/QRCODEgen/QROut.png")

```



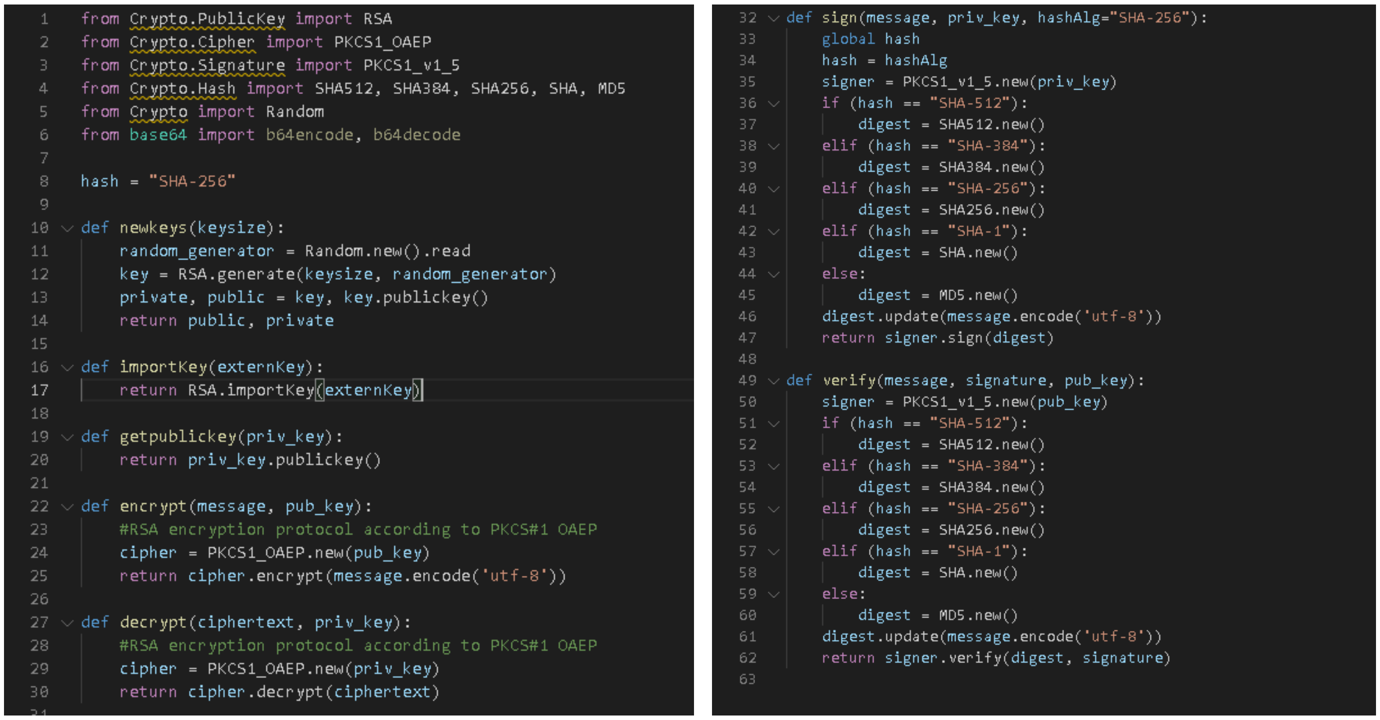
```python
def encodeBinary(binaryPath="/home/whorehay/Desktop/github/salty-peak-17003/QRCODEgen/",binary="im521.ppm"):
    [public,private] = getKeysFromFile()
    file = open(binaryPath+binary, "rb")
    byte = file.read(1)
    bl = []
    b2 = b""
    while byte:
        b2+=byte
        byte = file.read(1)
    file.close()

    data1 = str(b2)
    signature = b64encode(rsa.sign(data1, private, "SHA-1"))
    # print(type(b64decode(signature)),b64decode(signature))
    # print(type(signature),signature)
    # print(type(data1),data1)
    p2 = bytes("%%%", 'utf-8')
    print(type(data1),type(b64decode(signature)),data1,b64decode(signature))
    print(type(public),public)
    print(rsa.verify(data1,b64decode(signature),public))
    data =b2+p2+signature

    qr = qrcode.QRCode(
        version=1,
        error_correction=qrcode.constants.ERROR_CORRECT_L,
        box_size=10,
        border=4,
    )
    print(signature)
    # print(len(data))
    hah = str(data)
    print(hah.find("^"))
    print(hah.find("~"))
    hah = hah.replace("\\x","~")
    hah = hah.replace("00","^")

    # print(len(hah))
    # print(hah)
    qr.add_data(hah)


    qr.make(fit=True)

    img = qr.make_image(fill_color="black", back_color="white").convert('RGB')
    img.save("/home/whorehay/Desktop/github/salty-peak-17003/QRCODEgen/QROut.png")
# generateKeysandsave()
# encodeURL()
encodeBinary()
```

Figure 6: This is our generate.py file that generates the keys and encodes the url and binary



```
<class 'Crypto.PublicKey.RSA.RsaKey'> Public RSA key at 0x7F52823F86D0
https://theoldpurple.com True
[Finished in 6.212s]
```

Figure 7: This is our signed QR code output for the URL "https://theoldpurple.com" next to the scanner method with the first return value the data and the second the verification status.

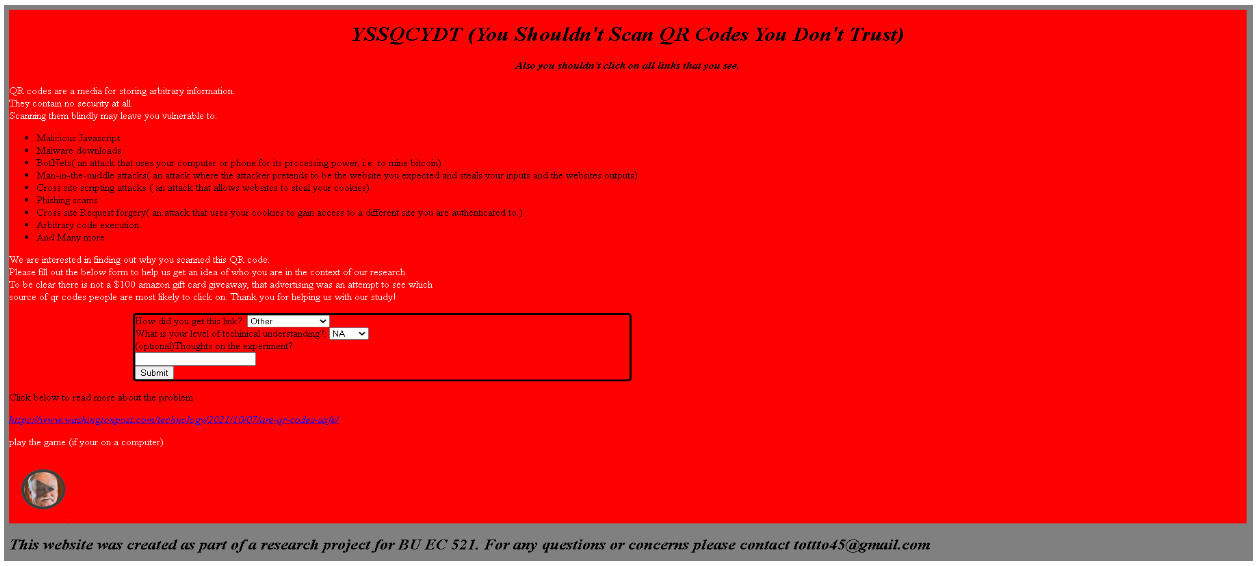


```python
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
from Crypto.Signature import PKCS1_v1_5
from Crypto.Hash import SHA512, SHA384, SHA256, SHA, MD5
from Crypto import Random
from base64 import b64encode, b64decode

hash = "SHA-256"

def newkeys(keysize):
    random_generator = Random.new().read
    key = RSA.generate(keysize, random_generator)
    private, public = key, key.publickey()
    return public, private

def importKey(externKey):
    return RSA.importKey(externKey)

def getpublickey(priv_key):
    return priv_key.publickey()

def encrypt(message, pub_key):
    #RSA encryption protocol according to PKCS#1 OAEP
    cipher = PKCS1_OAEP.new(pub_key)
    return cipher.encrypt(message.encode('utf-8'))

def decrypt(ciphertext, priv_key):
    #RSA encryption protocol according to PKCS#1 OAEP
    cipher = PKCS1_OAEP.new(priv_key)
    return cipher.decrypt(ciphertext)
```

```python
def sign(message, priv_key, hashAlg="SHA-256"):
    global hash
    hash = hashAlg
    signer = PKCS1_v1_5.new(priv_key)
    if (hash == "SHA-512"):
        digest = SHA512.new()
    elif (hash == "SHA-384"):
        digest = SHA384.new()
    elif (hash == "SHA-256"):
        digest = SHA256.new()
    elif (hash == "SHA-1"):
        digest = SHA.new()
    else:
        digest = MD5.new()
    digest.update(message.encode('utf-8'))
    return signer.sign(digest)

def verify(message, signature, pub_key):
    signer = PKCS1_v1_5.new(pub_key)
    if (hash == "SHA-512"):
        digest = SHA512.new()
    elif (hash == "SHA-384"):
        digest = SHA384.new()
    elif (hash == "SHA-256"):
        digest = SHA256.new()
    elif (hash == "SHA-1"):
        digest = SHA.new()
    else:
        digest = MD5.new()
    digest.update(message.encode('utf-8'))
    return signer.verify(digest, signature)
```

Figure 8: This is our RSA.py that uses RSA encrypt and decrypt to verify and sign the keys

Our Website:



YSSQCYDT (You Shouldn't Scan QR Codes You Don't Trust)

Also you shouldn't click on all links that you see.

QR codes are a media for storing arbitrary information.
They contain no security at all.
Scanning them blindly may leave you vulnerable to:

- Malicious Javascript
- Malware downloads
- BotNets(an attack that uses your computer or phone for its processing power, i.e. to mine bitcoin)
- Man-in-the-middle attacks(an attack where the attacker pretends to be the website you expected and steals your inputs and the websites outputs)
- Cross site scripting attacks (an attack that allows websites to steal your cookies)
- Phishing scams
- Cross site Request forgery(an attack that uses your cookies to gain access to a different site you are authenticated to.)
- Arbitrary code execution.
- And Many more

We are interested in finding out why you scanned this QR code.
Please fill out the below form to help us get an idea of who you are in the context of our research.
To be clear there is not a $100 amazon gift card giveaway, that advertising was an attempt to see which source of qr codes people are most likely to click on. Thank you for helping us with our study!

How did you get this link?: Other
What is your level of techinical understanding?: NA
(optional)Thoughts on the experiment?

Click below to read more about the problem

https://www.washingtonpost.com/technology/2021/10/07/are-qr-codes-safe/

play the game (if your on a computer)

This website was created as part of a research project for BU EC 521. For any questions or concerns please contact tottto45@gmail.com

Figure 9: This is an image of our website hosted using Heroku