SNOW CRASH

Using QR codes as a means of cyber attack

Research Manual

2020-11-13

Brendan D. Burke

C00232110@itcarlow.ie

Abstract

QR codes are ubiquitous in modern life and serve a multitude of functions including automatic Wi-Fi login, descriptions of museum exhibits, and even making SEPA compliant bank transfers. They have enjoyed enormous popularity and been adopted for a wide variety of use cases. While there is a growing awareness of the dangers of phishing sites the same awareness has not extended to what may be contained inside a QR code. The growth and adoption of QR codes has occurred in tandem with the growth of mobile computing. Similarly, there has been a marked growth in the amount of malware specifically targeting mobile devices. This begs the question; how safe are QR codes?

This paper aims to give an overview of the technologies involved, examples of cyber-attacks and scams that utilised QR codes, and some of the mitigations and security controls. The goal of this project will be to test several attacks that utilise QR codes, examine their technical feasibility and to attempt to squeeze an entire executable payload into a QR code.

The title Snow Crash was taken from Neal Stephenson's 1992 novel of the same name. To quote Stephenson "when the computer crashed and wrote gibberish into the bitmap, the result was something that looked vaguely like static on a broken television set—a 'snow crash'" (Stephenson, 1999).

Contents

Introduction

QR (Quick Response) codes are two-dimensional bar codes that are used to encode information in a machine-readable format. They are cheap to produce and implement, have high information density (compared to traditional bar codes), and have high error correction capabilities. They can be used to encode multiple different types of information and have been utilised in some strange and novel ways. Cyber criminals have also made use of them to steal money, download malware and phish sensitive information from victims. We are going start by looking at how QR codes work and then move on to some attacks and mitigations.

Overview of QR codes

QR codes were originally developed by Denso Wave, a subsidiary of Toyota, in 1994. They were developed by Masahiro Hara, an engineer at Denso Wave, who says he was inspired by the popular board game Go. QR codes were initially designed for quick tracking and identification of vehicle parts across a manufacturing line. Since their inception they have grown in scope and usage due to three main factors:

1. Denso Wave opted to exercise their patents on QR code technology in a limited fashion and their use is licensed freely provided users follow the relevant JIS or ISO standards.
2. The mass adoption of the personal smartphone which acts as a QR code reader.
3. The level of access to high-speed mobile internet and increased availability of public Wi-Fi networks.

Today QR codes have become ubiquitous and serve a multitude of functions including but not limited to:

- Advertising and marketing.
- Digital payment processing.
- Parcel/shipment tracking.
- Connecting to a Wi-Fi network.
- Food traceability systems.

The encoding and storing of information in a QR code depends on the version, the level of error correction and the encoding mode used. There are 40 different versions of QR code ranging from the smallest v1 (21 x 21 modules) to the largest v40 (177 x 177 modules). Each version adds 4 modules to the previous version i.e.

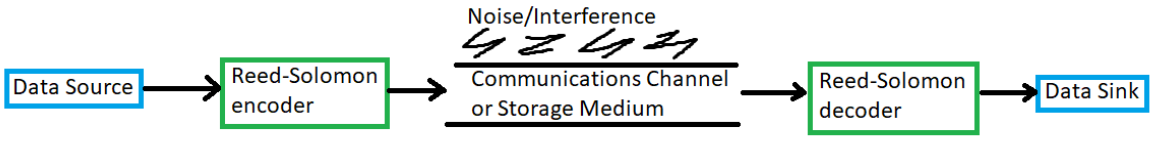
- v1 – 21 x 21 modules
- v2 – 25 x 25 modules
- v3 – 29 x 29 modules
- …
- V40 – 177 x 177 modules

(Images from left to right show a v1, v10 and v40)

As shown, a v40 QR code is much more information dense than a v1 or v10. However, the version number has no bearing on the size of a QR code. This means the QR code must be printed to a relevant size and the resolution offered by the QR scanner high enough to capture all the information. There is an obvious preference for smaller QR codes as they are usually easier to scan and decode. By using technology like URL-shorteners it is often possible to redirect someone to the correct webpage using a much smaller QR code version (Denso Wave, 2020).

Every QR code has built in error correction based upon Reed-Solomon codes. Reed-Solomon codes were originally developed by Irving S. Reed and Gustave Solomon in 1960. They have seen widespread application in digital media and can be found in CDs, DVDs, RAID, and satellite broadcasting. They were also used to encode and send digital pictures from the Voyager space probes back to Earth.

Reed-Solomon codes are based upon a simple system of adding and checking parity bits. Before sending or storing a data block it will be passed through a Reed-Solomon encoder where additional parity bits will be added to the message bits. When the data gets decoded it is first passed through a Reed-Solomon decoder which will use the parity bits to correct any errors in the data source. It will then go to the endpoint or data sink. The number of parity bits added will increase the error-correction available to the system (Riley & Richardson, 1996).



(Diagram of Reed-Solomon error correction)

When a QR scanner first reads a QR code it compares the data bits and the error correction bits. With this comparison it can work out if the QR code has been read correctly and, if it has not, it can correct the data bits to what they should read. The advantage is that if part of a QR code is obscured

or damaged the data can possibly still be retrieved from the image and used. The disadvantage is that because of this redundancy you lose some of the space that could otherwise be used for more data. Some companies make use of this error correction to place branding, logos, or advertising into the QR code.

(QR code with artistic embellishment that can still be read thanks to error correction)

There are 4 levels of error correction specified in the QR standard.

Error Correction Level	**Error Correction Capability (Approximate)**
L (Low)	7% of data bytes can be restored
M (Medium)	15% of data bytes can be restored
Q (Quartile)	25% of data bytes can be restored
H (High)	30% of data bytes can be restored

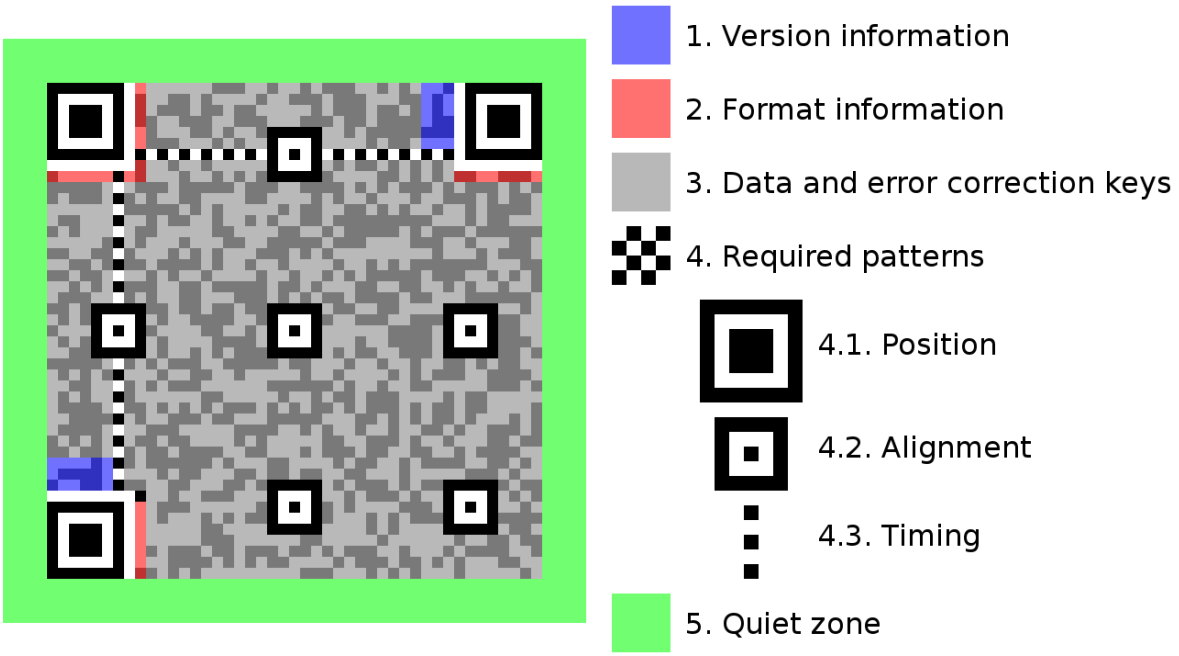
Using the correct level of error correction will depend entirely on the use case e.g. a QR code used on a factory floor may be more prone to wear and thus a higher level of error correction should be

used. Compare this to a QR code on a poster in a shopping centre. It is indoors, under glass and less likely to suffer any distortion so a low level of error correction will suffice.

QR codes are typically used to encode a string of text into a machine-readable format. The QR standard specifies four main modes for encoding data as shown in the table below. Each mode is optimized to generate the shortest possible string from the bits given. The values for the **Max Characters** column are based on a v40-L QR code (A **Version 40** with **Low** error correction) which currently represents the upper limit of information that can be stored using a QR code.

Mode	Max Characters	Encoding/Charsets
Numeric	7,089	0 - 9
Alphanumeric	4,296	0 - 9, A - Z (upper-case only), space, $, %, *, +, -, ., /, :
Binary/Byte	2,953	ISO 8859-1 (also known as Latin1)
Kanji/Kana	1,817	Shift JIS X 0208 (2-byte Japanese character encoding)

Note: Kanji/Kana is a 2-byte encoding for Japanese characters, some Chinese characters, Russian and English alphabets. Since it requires the greatest number of bits it does not see a lot use outside of Asia.



The above is a technical diagram of the make-up of a typical QR code. The blue sections contain the version information. They are mirror images of one another rotated 90 degrees. The red sections contain the format information. There is one copy on the top left corner and a second copy split between the top right and bottom left corners. This duplication and splitting up of information is one of the error correction traits of QR codes. By spreading out the information it means that areas of localised damage can be accounted for and corrected.

The recognisable corner and alignment tiles found in every QR code are there to help the scanner generate a clear, correctly orientated image. A QR code can be scanned from multiple angles and multiple orientations (sideways, upside-down, etc.) and a scanner will still be able to decode the image. The timing patterns are an alternating pattern of black and white modules used to provide support for QR code density and for quickly determining row/col coordinates.

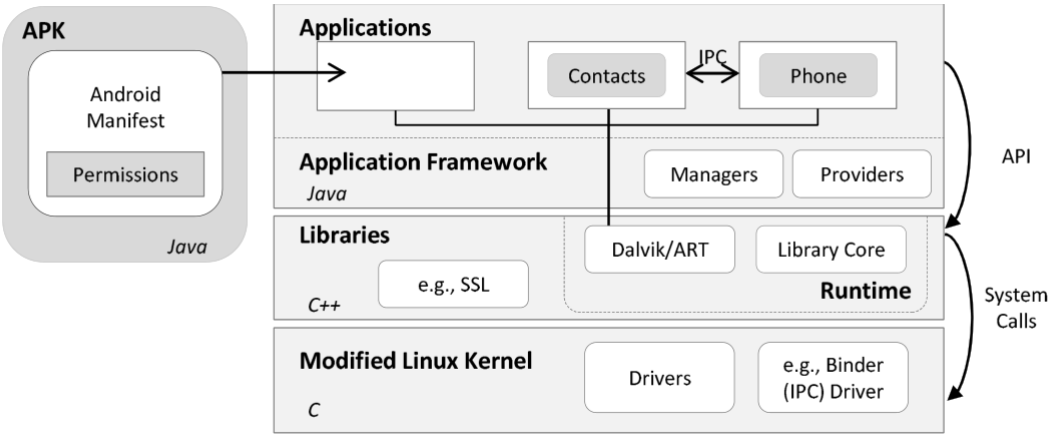
Finally, the data and error correction bits in the QR code will be spread out using 1 of 8 masking patterns. These masking patterns are used to break up large clusters of black or white modules that could be misread by the QR scanner. Masking also helps spread modules so that the QR code is again more resistant against localised damage. QR code generators will generally be able to select the appropriate masking pattern automatically when the information is encoded.

Overview of mobile malware

The growth of the home desktop market in the last decade led to an increase in malware targeting Windows and Mac. Similarly, mobile malware has been growing at an accelerated pace with the massive expansion of mobile computing this decade (Tam, et al., 2017). The focus of this section will be on the Android operating system as it currently holds roughly 75% of the mobile OS market share (Liu, 2020). The vast majority of mobile malware is Android-centric due to its widespread popularity, the open-source nature of the OS, and the ability to self-publish on application stores (Google Play and third party). These factors have made it a much-targeted platform for malware developers.

Mobile devices have some unique caveats that can make them more vulnerable to malware than a traditional desktop. They easily travel across multiple networks and domains which makes them incredibly attractive to worms trying to replicate themselves across other devices. There is also the risk of a device travelling from an infected home network into a work/industrial network or vice versa. The greater mobility of computing has thus led to the greater mobility of malware.

Mobile devices also present a greater attack surface due to technologies like GPRS, HSCSD, 3G, LTE, Bluetooth, SMS, MMS, accelerometers, etc. These are typically not found in traditional desktops or laptops, so they often present a novel route to infection either via poor implementation or via firmware drivers (Tam, et al., 2017). A recent example of this is a type of SMS phishing that was conducted using the OMA CP protocol (Open Mobile Alliance Client Provisioning). An attacker, pretending to be a vendor, would send an OTA (over-the-air) communication to a victim's phone capable of changing handset settings. This method could change the device's network settings and route traffic through a proxy owned by the attacker (Ilascu, 2019).



(Overview of the Android Operating System Architecture)

Android's security model is permission based and is handled at two core levels: the kernel level and the Application Framework level. The Linux kernel at the centre of Android restricts access to its resources and operations based upon user and group permissions. Each application on the device is uniquely IDed and restricted to accessing only the resources it has been explicitly granted permissions for. This generally provides a measure of separation/sandboxing of the individual applications, but researchers have proven there are ways of breaking out of this sandbox and reading arbitrary files using symbolic links (Taylor & Martinovic, 2018).

At the Application Framework there is another security control in the form of the manifest file (*AndroidManifest.xml*). This is where an application declares all the permissions it requires to run. These permissions are then sent as a request to the user at install time. Unfortunately, a large percentage of the Android userbase do not read the permissions requested by applications thus defeating one of the main security controls offered by the Android platform. Android applications are also capable of defining and declaring their own custom permissions (Elenkov, 2015). The permissions in the manifest file are ranked from lowest to highest in 4 protection levels.

1. **Normal**: This is the default value and defines a permission with low risk to the system or other applications. Permissions with the normal protection level are granted automatically without any user input. One example is ACCESS_NETWORK_STATE (gain access to network information).

2. **Dangerous**: Permissions with the dangerous protection level require a user's consent before being granted. They usually give access to user data or some form of device control. Examples are READ_SMS (application can read SMS message) and CAMERA (access to the device camera).

3. **Signature**: Signature permissions are granted only to applications that have been signed with the same key as the application that defined the permission. This means that applications using signature permissions are usually controlled by the same developer since they signed both applications with the same key. There are also built-in signature

permissions that are typically used by system applications to perform device management tasks.

4. **SignatureOrSystem**: These permissions are granted to applications that are either signed with the same key as the declaring application (as with signature permissions) of if they are part of the system image. This makes SignatureOrSystem a slightly compromised version of Signature protection level. This allows vendors that have their applications preinstalled on a device to access system functionality they ordinarily would not have access to. These applications must be installed in the */system/priv-app/* directory to be granted SignatureOrSystem permissions. If malware manages to install itself to this folder it can become incredibly difficult to remove from the system.

Despite Android's security controls there are numerous case studies that have shown that these mechanisms can be outflanked (Kambourakis, et al., 2018). Modern Android malware has caught up with its PC relatives in a relatively short time and today employs many sophisticated methods to obstruct and evade analysis and reverse engineering efforts. This means that hybrid systems of static and dynamic analysis techniques are usually employed to work out what the malware is doing. These hybrid analysis techniques must be employed by both security researchers and by Android anti-virus programs.

Some of the evasion techniques employed by modern Android malware are listed below.

1. **Java Reflection**: This is a feature of the Java programming language that allows an executing program to examine and manipulate its own internal properties. Lots of Android developers leverage the *java.lang.reflect* package to determine, at runtime, what classes and methods are available before making an API call. This means the same application can use newer APIs while still retaining support for older Android versions (Conder & Darcey, 2010). Android malware can utilise this mechanism to check if it is being sandboxed, check connectivity status and system time, hide API names, and generally obfuscate what it is doing.

2. **Dynamic Code Loading**: Dynamic Code Loading or DCL allows a program to load additional code at runtime from both inside and outside the application source code. DCL can be used to bypass conventional malware detection by only loading the malicious code from a remote server after the application has been successfully installed on a victim's device. This technique is widely employed by a type of mobile malware known as "droppers". Droppers are a class of Trojan usually bundled with a seemingly legitimate application. By themselves they exhibit little malicious behaviour and can bypass numerous security controls e.g. Google Bouncer. Once installed a dropper can then load one or more malicious payloads remotely. Some cybercrime gangs have started the practice of renting install space on their dropper for other gangs to load their payloads similar to the practice of renting botnets (Cimpanu, 2018). Important to note is that some benign applications also open themselves up to code injection vulnerabilities from malicious applications by using DCL. While Google's content policy states all application updates must go through its marketplace (Google Play) it is entirely possible for developers, both benign and malicious, to violate this policy and dynamically load remote code at runtime (Qu, et al., 2017).

3. **Obfuscation**: Decompiling and retrieving the original Java source code from an APK is trivial and so developers commonly obfuscate their code using tools such as ProGuard. ProGuard and DexGuard are commonly used because they are built into the Android toolkit. This is usually done to protect commercial interests and prevent other parties from copying code and stealing intellectual property. The same methods are employed by malware creators to obfuscate malware. One example of a common obfuscation technique is string encryption. Strings inside binaries will become unreadable and are only decrypted during runtime. There is an associated performance penalty, but it forces a reverse engineer to use dynamic analysis methods. Another common obfuscation technique is control flow obfuscation which makes the decompiled code look like a complete mess of inverted logic and jumping function calls, but which will execute as perfectly valid code (PreEmptive, 2020).

One recent example of a sophisticated strain of Android trojan is the Shedun family of adware. Shedun, also known as Shuanet, Shiftybug, and Kemoge, first appeared in August 2015. It was repacked with legitimate, popular applications such as Facebook, Candy Crush, Twitter, etc and then released via third party stores. These trojanised apps were perfectly functional but, in the background, Shedun would auto-root the device and begin serving ads and installing applications. Check Point Software Technologies managed to trace the origin of Shedun to a cybercrime gang in China associated with the legitimate mobile ad server company Yingmob. Yingmob are based in Chongqing, in southwest China, and Check Point estimates the Shedun malware generates about $300,000 per month for the company (Rogan, 2016). Shedun, once installed, can auto-root the device it is on and install itself to the system partition meaning it can survive factory-resets. This makes it incredibly persistent and has led to its classification as “aggressive adware”. The average user is usually incapable of flashing their device with a valid rom image from their manufacturer and thus their only option is to purchase a new device (Bentley, 2015).

QR codes as an attack vector

QR codes offer great advantages but also pose security risks when the conveniences they provide are taken advantage of by criminals. China currently leads the world in terms of electronic payments via mobile and QR codes. WeChat, owned by the Tencent conglomerate, and Alipay, owned by the Alibaba group accounted for 90% of the $17 trillion mobile payments market in 2017 (CGAP, 2019). These mobile payments are made by selecting an amount in the app and then scanning a QR code to transfer the funds. The receipt is then printed on both the buyer and the seller’s device. QR codes have become so commonplace as a means of payment in China that they are even placed on tickets issued for parking violations and similar misdemeanours.

Criminals in Shanghai took advantage of this practice and began to issue fake parking tickets to vehicles parked in restricted areas. Payments would then go to a private WeChat account where the profile picture appeared to be a police officer so people would not immediately realise they were being scammed (Goula, 2019). Probably the largest such scam to date occurred in Guangdong

province where one criminal gang managed to generate about 90 million yuan (US $13 million) by pasting their own QR codes over legitimate ones. There have been many similar cases involving the replacing of QR codes on a merchant's stand or replacing the QR code on a rental bicycle. Payments made to the merchant would go to the scammer's account instead (Tao, 2017).

One of the earliest examples of using a QR code as a means of attacking mobile devices goes back to 2011. Argentinian security researcher Augusto Pereyra described a method he named "Attaging" which involves a victim scanning a QR code containing a URL that would redirect them to a remote server running an instance of Metasploit (Pereyra, 2011). The same year Kaspersky Labs announced that they had discovered the very first malicious QR code being used in the wild by cyber-criminals in Russia. The QR code, once scanned, would automatically download a version of the Jimm-mobile ICQ client infected with the *TrojanSMS.AndroidOS.Jfake.f* trojan which automatically sent SMS messages to premium rate numbers (Kharraz, et al., 2014).

In 2012 security expert Ravi Borgaonkar demonstrated a novel 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). Other novel forms of attack have been discussed such as placing a secondary malicious QR code or Aztec code (2D barcode, similar to QR codes) inside a primary benign one (Dabrowski, et al., 2014). Another demonstrated attack involved using a QR code for a Wi-Fi login to a network controlled by the attacker so they could sniff sensitive information. The versatility of QR codes have lent themselves to many different ways in which they can be abused by bad actors.

Placing a malicious URL inside a QR code is an obvious attack vector and could be used to download malware, steal private device data, or display shock images for an attacker's amusement. The most practical attacks are phishing attacks e.g. present a QR code, redirect the user with a URL that appears untoward and having users input sensitive data on a seemingly legitimate site. However, it is also possible to place an entire binary executable inside a QR code and have it run when scanned as proven by Australian hacker MattKC (MattKC, 2020). The underground phenomenon of boot-sector games and demoscene subculture have long proven that it is possible to squeeze a surprising amount of functional code into a very limited space. Boot-sector viruses and malware are also not uncommon. A modern example would be the CIA's Angelfire project of which Solartime is the component that modifies the boot sector (Cimpanu, 2017).

Mitigations against QR code attacks

QR codes do not have any inbuilt security mechanisms i.e. they are merely meant to store and represent data. They are also not human-readable meaning that any deciphering between a benign and a malicious QR code must be performed by the QR scanner. As of the time of writing there currently exists no standard mechanism for proving the authenticity and integrity of a QR code. This means that any data coming from a QR code should be considered untrusted.

In China Alipay has said it owns a technology that can determine whether a QR code was generated by its own systems, or if it contains a malicious link produced by a scammer. If a security risk is detected the Alipay app will warn the user and allow them to decide if they want to proceed with a payment or not. Many vendors in China have moved to placing their QR codes where they can still be scanned by a customer but out of reach of anyone who might try to tamper with it (Tao, 2017).

Other improvements that have been suggested is the use of public key cryptography to check a digital signature embedded into the QR code. The disadvantages of such a system are the space limitations of QR codes and the extra computational time needed to verify the signature. However, researchers have shown that it is feasible and that using the ECDSA and RSA schemes QR codes can be digitally signed and verified (Focardi, et al., 2018). Important to note is that digital signatures provide authentication, integrity, and non-repudiation but not confidentiality. They also do not guarantee the contents of the QR code are safe. They merely prove which private keys were used to sign the QR code. Some have suggested a system of digital signatures and verification via a third-party Certificate Authority much in the same way that TLS currently operates (Mavroeidis & Nicho, 2017). However, this would require anyone who creates a QR code to now get a certificate and sign it. This extra effort would probably be viewed as a detriment to adoption and usability by many developers. It also requires QR scanners to be rewritten and Certificate Authorities to begin to issue certificates for individual QR codes. We have also seen sophisticated phishing sites that use TLS certificates so this cannot be considered a comprehensive solution to the core issues of QR code security.

Some QR code scanners make use of “Google Safe Browsing”, a public API that can detect phishing attacks and malware infected content. Safe Browsing also protects users on Android; however, a user can often choose to ignore the warnings and proceed with a download or visiting a site. Another issue is that Google Safe Browsing only flags the offending page and not necessarily the full domain. It is possible to evade this mechanism by redirecting users from the landing URL to the phishing page (Mavroeidis & Nicho, 2017).

Security Researchers in 2017 showed that displaying a URL to a user should be the most basic of security features and that blocking confirmed malicious URLs is a must. It should not be left to the user to discern whether a URL is truly malicious. A large amount of QR scanners (even those provided by anti-virus companies like Avira and Kaspersky) failed to detect URL redirection through a QR code and would direct users to a site containing malware (Dudheria, 2017). Most smartphone users are perhaps not aware that QR codes can contain malicious URLs just like a phishing email. For attackers, encoding a URL into a QR code requires minimal effort and may produce better results given the lack of awareness surrounding the dangers.

The human – QR interaction

As with other forms of social engineering the presentation of the scam is key to the effectiveness of the campaign. We see today that generalised phishing emails are slowly going the way of the dodo and modern attempts are a lot more targeted and refined. This had led to the term “spear-phishing”. We have seen an increase in phishing emails that are properly formatted, appear official, have nice clean graphics, may address a person by name, etc.

In relation to QR codes this presents the question: Are QR codes more effective when they are presented with logos and accompanying text or is a plain QR code more enticing to scan? In 2014 security researchers performed a simulated phishing attack with 3 different designs of QR code poster leading to an online survey. The first was a plain QR code with no additional information, the second came with a description and the third was a QR code surrounded by cute cat images. It transpired that the plain QR codes with no information did best (Krombholz, et al., 2014). While not to be taken as final and conclusive, the fact remains that a sizeable population of users will scan a QR code with no accompanying information. This is the equivalent to clicking on malicious URLs blindfolded. While the best advice would be to include an indication of what a QR code does on the same poster/sticker it seems that curiosity may lead to better results both for bad actors and those with marketing interests.

One point of further research could be what kind of results a plain QR code versus a snazzy version would generate in a phishing campaign. As with any phishing campaign done for research or educational purposes this would have to be kept within strict ethical guidelines. The goal should be to gather data to see where users make security mistakes and then educate appropriately on the dangers. Some questions that could be answered by such a campaign are:

- Does the average user consider a QR code with a logo to be more “official” and therefore more trustworthy? Or does the surrounding poster make more of an impact than the QR code itself? How do logos, branding, etc affect levels of trust?
- Can you incentivise users into scanning with the promise of some form of compensation e.g. a voucher or coupon?
- Is the proposition of being “entered into a draw” with a chance to win a larger prize more attractive than an immediate smaller prize or compensation?
- Are certain sites trusted because of their popularity e.g. can you phish sensitive information using SurveyMonkey or a Google Forms survey?

(3 QR codes, generated with QR Customizer Pro, all pointing to https://www.itcarlow.ie/)

Of course, the key to generating enough data as to be significant would rely on how much footfall there is and the adoption rate of QR usage in that region. In China QR adoption has been massive (Chiu, 2018) and so an urban centre such as Beijing or Shanghai would probably produce more data than Dublin even taking population density into account. Another significant factor is the current COVID-19 lockdown restrictions which have reduced urban footfall. This would naturally affect results and accuracy of any such study.

In most security systems the human element is the weakest link in the chain. To err is human nature and only by collecting data on where we err can we begin to properly educate for the safer use of technology. QR codes offer many advantages and, given that everyone carries a QR scanner in their pocket, seem to be a perfect means of storing small pieces of data. However, as they have no security features, it is up to the developers of QR readers and users to be vigilant and reticent of the dangers.

Tools and Technologies

Generating QR codes

Libqrencode: Libqrencode is the most popular and widely used QR code generator on the Linux platform. It is an open-source command-line utility that provides all the major functionality needed to create a QR code from a given string of input. It can output QR codes in multiple image format types e.g. png, jpg, bitmap, etc. It is licensed under the GNU LGPL v2.1 or later.
https://fukuchi.org/works/qrencode/

QtQR: QtQR, as the name suggests, is based on the popular Qt framework. It is a graphical program for creating QR codes in Linux. The backend library is python-qrtools. It can create many different types of QR code, provides a means to adjust individual pixel size and margins, and even decode and read back QR codes. It is licensed under the GNU GPL v3.
https://launchpad.net/qr-tools

Portable QR-Code Generator: This is a cross-platform QR code generator written in Java. It can produce a wide variety of QR codes including WLAN sign-in, VCard, URLs, GPS co-ordinates, etc. Generated QR codes can be printed, saved as BMP, GIF or PNG, or copied to clipboard to use in other applications. It is distributed as freeware under the GNU GPL v3.
https://sites.google.com/site/qrcodeforwn/home/qr-code-generator-for-wireless-networks

QR Code Studio: QR Code Studio is a popular GUI-based QR code generator that works on Windows (will work on Linux using WINE) and MacOS. It is highly functional and can create various typical QR codes including Text, URL, VCard, Mobile Tagging, SEPA payments, etc. It is released under a freeware license by TEC-IT.

https://www.tec-it.com/en/download/free-software/qrcode-studio/Download.aspx

QR Customizer Pro: QR Customizer Pro is another popular GUI-based QR code generator that works on Windows. Its unique selling point is that it can be used to customise QR codes with custom art, colours, logos, designs, etc. This is useful for making a QR code appear "official" if this was required. The trial/free version contains most of the key features with the Pro version adding several extra graphical effects. It is licensed under a custom EULA.
https://www.qrcustomizerpro.com/

h0nus/QRGen: This is not a QR code generator per se but rather a python script for generating malicious QR codes from a wordlist. These payloads are designed to test how a QR scanner's parser handles malicious input and fuzzing. It also offers the option of creating QR payloads from a custom wordlist. It is licensed under the GNU GPL v3.
https://github.com/h0nus/QRGen

The reason for choosing so many different QR code generators is to check for discrepancies in how different generators and readers handle data. An example is in QtQR adding or removing the UTF8-BOM character will change how certain readers parse the data. This is because the QR standard specifies the use of ISO-8859-1 instead of UTF8. Comparing results between different generators and readers should lead to quick error correction. Some of the chosen generators also have features that the others do not. All the above are available for free (as in beer) and under licenses that are permissive for their intended use in the project.

Reading QR codes

ZXing Barcode Reader: This is an old established QR code reader based on the popular open-source ZXing (Zebra-Crossing) library. It has numerous settings that can be finetuned depending on what type of QR code is being scanned. The project is now in maintenance mode, so it is not receiving active development. There are numerous ports of the ZXing library to other languages e.g. C++, PHP, etc so its codebase appears in numerous QR code readers. It is released under an Apache 2.0 license.
https://play.google.com/store/apps/details?id=com.google.zxing.client.android&hl=en_US&gl=US

Barcode Scanner+ (Plus): This is an improved version of the ZXing Barcode Reader created by one of the original ZXing developers (Sean Owen). It was made free and open-source as part of the project's 10-year anniversary. It has the ability to decode QR codes from image files which is a significant advantage over the original ZXing scanner.
https://play.google.com/store/apps/details?id=com.srowen.bs.android

QR & Barcode Scanner Gamma Play: This is one of the most popular QR code scanners on the Google Play store with over 100M+ downloads. It can read many different types of QR codes and bar codes both from camera and from image files in the device Gallery.
https://play.google.com/store/apps/details?id=com.gamma.scan

QR & Barcode Reader TeaCapps: After Gamma Play's QR & Barcode Scanner this is the second most popular app on the Google Play store (at time of writing) with 50M+ downloads. It specifically mentions its use of "Google Safe Browsing" in its info section. It also has the ability to scan from both camera and Gallery.
https://play.google.com/store/apps/details?id=com.teacapps.barcodescanner

FREE QR Scanner: Barcode Scanner & QR Code Scanner: An app developed by Simple Design Ltd, this is another popular QR scanner on the Google Play store with over 10M+ downloads. It is fully featured like the previously mention scanners. It also claims to only require camera permissions in order to "keep user privacy 100% safe".
https://play.google.com/store/apps/details?id=qrcodereader.barcodescanner.scan.qrscanner

ZBar Bar Code Reader: ZBar Bar Code Reader is an open-source software suite for reading various types of barcodes. It is available for Windows and Linux, has language bindings for C, C++, Python 2, and Perl as well as GUI widgets for Qt, GTK, and PyGTK 2.0. It serves as the main library for a variety of different scanners on different platforms. Given it's incredibly small size it is found in numerous embedded systems. It is licensed under the GNU LGPL v2.1
https://github.com/mchehab/zbar

A selection of QR readers were chosen based upon popularity. One of the goals will be to see if they implement different ways of parsing QR data and how they handle dangerous input. They also ask for different permissions which may or may not make them more vulnerable. ZBar and ZXing are two of the most popular libraries of QR based tools and are found in multiple apps and programs. ZBar is slightly more popular, lightweight, and can be used in Linux shell scripts.

Emulating Android

Android-x86: Android-x86 is an open-source project that allows you to run the Android operating system in a virtual machine. It also allows for multiple versions of Android (6.0, 7.1, 8.1, LineageOS roms, etc.) to be installed. These ports are fully featured and can perform many of the functions a real mobile device can including access the Google Play store. It is released under an Apache 2.0 license.
https://www.android-x86.org/

VirtualBox: VirtualBox is a popular, free, and open-source hypervisor developed by the Oracle Corporation. It runs on Windows and Linux and can host a vast number of host systems. VirtualBox will be used in combination with Android-x86 images to sandbox and isolate an Android device before allowing any malicious code to execute. It is licensed under the GNU GPL version 2.
https://www.virtualbox.org/

Developing Android APKs

Android Studio: Android Studio is official IDE provided by Google for Android app development. It provides support for coding in Java, Kotlin and C++. It also provides a feature-rich emulator that can be used to test an app on multiple versions of Android and on different devices including tablets, smartwatches, etc. While this may be good for testing certain payloads, its main use will be writing and debugging code.
https://developer.android.com/studio

Languages

Java: Java is the main programming language on the Android platform. While Android apps can be developed in other languages Java is the most popular. Android apps run in the ART (Android Runtime) environment which replaced the old Dalvik VM with the introduction of Android 5.0. Android applications come in apk files which are converted to Java bytecode and then transformed into native instructions by the ART. Of note is that Android provides its own GUI classes and does not use Java AWT, Swing or JavaFX.

Kotlin: Kotlin is a newer language designed to interoperate fully with Java and the JVM. Kotlin is Google's preferred programming language for Android apps and comes fully supported in Android Studio. Kotlin has some advantages over Java e.g. it is easy to learn, you can avoid NullPointExceptions and other Java specific issues, there is less boilerplate code due to the leaner syntax, etc.

C/C++: Android is based upon a modified Linux kernel which is written in C. All the low-level drivers and core functionality of the system relies on C. Android's system libraries and the ART are written in C++. Android apps written in C++ generally run faster than their Java counterparts (due mainly to C++ being a compiled language and Java being interpreted). Existing C and C++ libraries can be incorporated into an app by using the Android NDK (Native Development Kit). This builds the code into a native library that can be packaged with the APK. Sometimes developers will write an app in C or C++ and call the native library from a Java or Kotlin class. C or C++ are generally used when access is needed to things like device sensors, gyroscopes, or battery. They can also be used when it is easier for a developer to reuse an existing C/C++ library than to reimplement it in Java or Kotlin.

Assembly: Many viruses and malware take advantage of low-level programming in order to compromise a system. Being able to reverse-engineer payloads/malware and understand how they work on a particular architecture is a valuable skill for anyone trying to create malware or build more robust systems. The vast majority of mobile malware works on the ARM platform so the high-level code disassembles to ARM assembly.

HTML, CSS, JavaScript, PHP: A malicious QR code can contain a URL that points to a remote server. When a request is made this server may try to run some form of malicious JavaScript in a user's browser, or download a malicious apk file, or perhaps serve a phishing site. This would have to be done with some combination of web languages running on a XAMPP stack or similar.

Conclusion

QR codes serve numerous functions and are an incredibly useful technology. They have become arguably the most popular and usable 2D barcode in the world today. There are numerous ways in which QR codes can be taken advantage of by bad actors. QR codes, like some other technologies and protocols still in use, were never designed with any security features. One classic example of this is HTTP, which serves every page in plaintext and does not use any encryption. HTTP was then succeeded by a secure, redesigned version: HTTPS. While there are some modern 2D barcodes such as JAB (Just Another Barcode) they have yet to see mass adoption similar to QR codes which are the de facto standard.

Only by thoroughly testing existing security controls and educating users can we improve our overall security posture. Seeing how practical real-world attacks function can give us insight into how to mitigate these attacks. Sometimes cyber attacks can occur using very novel or unpredictable means e.g. sending a fax (Itkin, et al., 2018).

The QR code protocol does not offer any inbuilt security features and thus, their safe use can only be determined by the parsing software and the end users. Given the recent COVID-19 pandemic QR codes have experienced a huge surge in popularity due to the increased consciousness surrounding contaminated surfaces (Horswill, 2020). They are currently being deployed for easy check-in using COVID tracker apps, to sign digital forms, collect shopping vouchers, and to make digital payments. This upsurge in popularity means that now, more than ever, we should be aware of the security risks of QR codes.

References

Bentley, M., 2015. *Lookout discovers new trojanized adware; 20K popular apps caught in the crossfire.* [Online]
Available at: https://blog.lookout.com/trojanized-adware
[Accessed 05 November 2020].

CGAP, 2019. *China: A Digital Payments Revolution.* [Online]
Available at: https://www.cgap.org/research/publication/china-digital-payments-revolution
[Accessed 05 November 2020].

Chiu, K., 2018. *How the QR code conquered China.* [Online]
Available at: https://www.scmp.com/abacus/who-what/what/article/3028233/how-qr-code-conquered-china
[Accessed 11 November 2020].

Cimpanu, C., 2017. *CIA Developed Windows Malware That Alters Boot Sector to Load More Malware.* [Online]
Available at: https://www.bleepingcomputer.com/news/security/cia-developed-windows-malware-that-alters-boot-sector-to-load-more-malware/
[Accessed 12 November 2020].

Cimpanu, C., 2018. *Droppers Is How Android Malware Keeps Sneaking Into the Play Store.* [Online]
Available at: https://www.bleepingcomputer.com/news/security/droppers-is-how-android-malware-keeps-sneaking-into-the-play-store/
[Accessed 04 November 2020].

Conder, S. & Darcey, L., 2010. *Learn Java for Android Development: Reflection Basics.* [Online]
Available at: https://code.tutsplus.com/tutorials/learn-java-for-android-development-reflection-basics--mobile-3203
[Accessed 03 November 2020].

Dabrowski, A., Krombholz, K., Ullrich, J. & Weippl, E. R., 2014. *QR Inception: Barcode-in-Barcode Attacks.* Scottsdale, AZ, Association for Computing Machinery.

Denso Wave, 2020. *What is a QR Code?.* [Online]
Available at: https://www.qrcode.com/en/about/
[Accessed 02 November 2020].

Dudheria, R., 2017. *Evaluating Features and Effectiveness of Secure QR Code Scanners.* Nanjing, China, IEEE.

Elenkov, N., 2015. *Android Security Internals.* First Printing ed. s.l.:No Starch Press, Inc..

Felt, A. P. et al., 2011. *A survey of mobile malware in the wild.* Chicago, Illinois, CCS, pp. 3-14.

Focardi, R., Luccio, F. L. & Wahsheh, H. A. M., 2018. *Usable cryptographic QR codes.* Lyon, France, IEEE.

Garateguy, G. J., Arce, G. R., Lau, D. L. & Villarreal, O. P., 2014. QR Images: Optimized Image Embedding in QR Codes. *IEEE Transactions on Image Processing*, 02 May, 23(7), pp. 2842 - 2853.

Goula, J., 2019. *Traffic ticket QR code scams cause problems in China.* [Online]
Available at: https://www.qrcodepress.com/traffic-ticket-qr-code-scams-cause-problems-in-china/8536452/
[Accessed 30 October 2020].

Horswill, I., 2020. *The QR code is booming in the time of COVID-19.* [Online]
Available at: https://www.theceomagazine.com/business/news/qr-code-covid-19/
[Accessed 14 April 2021].

Ilascu, I., 2019. *Android SMS Phishing Can Stealthily Enable Malicious Settings.* [Online]
Available at: https://www.bleepingcomputer.com/news/security/android-sms-phishing-can-stealthily-enable-malicious-settings/
[Accessed 02 Novermber 2020].

Itkin, E., Livneh, Y. & Balmas, Y., 2018. *Faxploit: Sending Fax Back to the Dark Ages.* [Online]
Available at: https://research.checkpoint.com/2018/sending-fax-back-to-the-dark-ages/
[Accessed 14 April 2021].

Kambourakis, G., Shabtai, A., Kolias, C. & Damopoulos, D., 2018. *Intrusion Detection and Prevention for Mobile Ecosystems.* s.l.:Taylor & Francis Group, LLC.

Kharraz, A. et al., 2014. *Optical Delusions: A Study of Malicious QR Codes.* Atlanta, GA, IEEE.

Kieseberg, P. et al., 2010. *QR code security.* Paris, France, Association for Computing Machinery, pp. 430-435.

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.

Krombholz, K. et al., 2015. *QR Code Security - How Secure and Usable Apps Can Protect Users Against Malicious QR Codes.* Toulouse, France, IEEE.

Li, C. M., Hu, P. & Lau, W. C., 2015. *AuthPaper: Protecting paper-based documents and credentials using Authenticated 2D barcodes.* London, UK, IEEE.

Lin, P.-Y. & Chen, Y.-H., 2017. High payload secret hiding technology for QR codes. *EURASIP Journal on Image and Video Processing*, 08 February, Issue 14.

Liu, S., 2020. *Android - Statistics & Facts.* [Online]
Available at: https://www.statista.com/topics/876/android/
[Accessed 05 November 2020].

MattKC, 2020. *Snake in a QR code.* [Online]
Available at: https://itsmattkc.com/etc/snakeqr/
[Accessed 23 October 2020].

Mavroeidis, V. & Nicho, M., 2017. *Quick Response Code Secure: A Cryptographically Secure Anti-Phishing Tool for QR Code Attacks.* Warsaw, Poland, Cham Springer, pp. 313-324.

Narayanan, S. A., 2012. QR Codes and Security Solutions. *International Journal of Computer Science and Telecommunications,* 3(7).

Peng, K., Sanabria, H., Wu, D. & Zhu, C., 2014. *Security Overview of QR Codes,* Massachusetts: MIT.

Pereyra, A., 2011. *KaOtiCo NeUtRaL.* [Online]
Available at: https://kaoticoneutral.blogspot.com/2011/09/using-qr-tags-to-attack-smartphones_10.html
[Accessed 06 November 2020].

PreEmptive, 2020. *What is Obfuscation?.* [Online]
Available at: https://www.preemptive.com/obfuscation
[Accessed 03 November 2020].

Qamar, A., Karim, A. & Chang, V., 2019. Mobile malware attacks: Review, taxonomy & future directions. *Future Generation Computer Systems*, August, Volume 97, pp. 887-909.

Qu, Z. et al., 2017. *DyDroid: Measuring Dynamic Code Loading and Its Security Implications in Android Applications.* Denver, CO, IEEE.

Riley, M. & Richardson, I., 1996. *Reed-Solomon Codes.* [Online]
Available at: https://www.cs.cmu.edu/~guyb/realworld/reedsolomon/reed_solomon_codes.html
[Accessed 02 November 2020].

Rogan, M., 2016. *Android Trojanized Adware 'Shedun' Infections Surge.* [Online]
Available at: https://www.linkedin.com/pulse/android-trojanized-adware-shedun-infections-surge-mike-rogan
[Accessed 05 November 2020].

Sharma, V., 2010. A Study of Malicious QR Codes. *International Journal of Computational Intelligence and Information Security*, May, 3(5), pp. 21-26.

Stephenson, N., 1999. *In the Beginning was the Command Line,* s.l.: Published online.

Tam, K. et al., 2017. The Evolution of Android Malware and Android Analysis Techniques. *ACM Computing Surveys*, January, 49(4).

Tao, L., 2017. *QR code scams rise in China, putting e-payment security in spotlight.* [Online]
Available at: https://www.scmp.com/business/china-business/article/2080841/rise-qr-code-scams-china-puts-online-payment-security
[Accessed 30 October 2020].

Taylor, V. F. & Martinovic, I., 2018. Attacking Smartphone Security and Privacy. In: J. Zhou & P. Samarati, eds. *Intrusion Detection and Prevention for Mobile Ecosystems.* s.l.:Taylor and Francis Group, pp. 27-29.

Yong, K. S. C., Chiew, K. L. & Tan, C. L., 2019. *A survey of the QR code phishing: the current attacks and countermeasures.* Sarawak, Malaysia, IEEE.

Zhou, Y. & Jiang, X., 2010. Dissecting Android Malware: Characterization and Evolution. *2012 IEEE Symposium on Security and Privacy,* 20 May.

Image references

Cover Page: QR code with cover page text created by me.

Page 1: QR code pointing to my LinkedIn profile.

Page 4: 3 QR code images taken from Wikimedia Commons licensed under Creative Commons License. In order:
https://commons.wikimedia.org/wiki/File:Qr-1.svg
https://commons.wikimedia.org/wiki/File:Qr-code-ver-10.svg
https://commons.wikimedia.org/wiki/File:Qr-code-ver-40.svg

Page 4: Reed-Solomon high level diagram created by me in MS paint.

Page 5: Artistic QR code example. Released to Public Domain by creator.
https://commons.wikimedia.org/wiki/File:2_150_150DPI_ty_oerny_08_2011.jpg

Page 6: Technical diagram of QR code. Creative Commons.
https://commons.wikimedia.org/wiki/File:QR_Code_Structure_Example_3.svg

Page 8: Android Operating System Architecture diagram taken from page 5 of:
Tam, K. et al., 2017. The Evolution of Android Malware and Android Analysis Techniques. *ACM Computing Surveys*, January, 49(4).

Page 13: 3 QR codes, generated with QR Customizer Pro, all pointing to https://www.itcarlow.ie/