SNOW CRASH

Using QR codes as a means of cyber attack

Functional Specifications

2020-11-27

Brendan D. Burke

C00232110@itcarlow.ie

Contents

Abstract

The Snow Crash project consists of a collection of attacks utilising QR code technology. The purpose of this document is to set out a practical approach to the project. This document details the functionality of the core components of the project and will discuss the main technologies used to implement the attacks. It will also consider design and implementation issues in order to keep the project goals realistic.

Introduction

This project will be broken up into 3 main sections:

1. Create a tool/program for generating and testing malicious QR codes.
2. Present practical, reproduceable attacks that utilize QR code technology.
3. Create novel QR code malware and test whether the executable can be retrieved from the QR code.

The first section will focus on creating a Linux-based, CLI tool for generating dangerous/malicious QR codes. In the second section we will present scenarios that could be used to attack QR code users via social engineering means. The final section will be an attempt to create a piece of malware that can be delivered via scanning a QR code.

Project Scope

Goal

Create a simple tool for generating and testing QR codes. Present a series of attacks that test the security, or lack of, in QR technologies and document them appropriately for other security researchers and developers. All attacks should be testable and reproduceable.

Technology Stack

Tools

QR code generators: There are numerous QR code generators available on every platform and in every language. A total of six different generators were chosen to quickly root out any discrepancies in how they produce QR codes. An example is in QtQR adding or removing the UTF8-BOM character will change how certain readers parse the data. Some of the chosen generators also have features that the others do not e.g. adding logos and branding. All were chosen because they are free (as in beer) and under licenses that are permissive for the projects intended use. A comprehensive list of the generators and their respective licenses can be found in the project research manual under the "Tools and Technologies" section. The main generator used in the tool will be libqrencode while the others are for extra features and comparing output.

QR code readers: Five different Android QR code readers were chosen based upon their popularity in the Google Play Store. These will be tested to see how they handle malicious input. These are all based on the ZBar or ZXing libraries. The ZBar library seems to be more popular overall so it will be used in the command line tool for testing.

Android Emulation: Android emulation will be performed using Android-x86 images inside of VirtualBox. This will make it easy to roll back to a working installation if something gets broken. Android-x86 also provides many different images ranging from Android 1 (donut) to Android 9 (pie).

Android Studio: Android Studio is the official IDE for developing, testing, and debugging Android apps. It also contains an emulator for testing apps; however, this emulator is not sandboxed and should not be used for testing malware.

VirtualBox: Any malware or dangerous payloads contained within a QR code should only be scanned and executed inside a properly sandboxed environment on a virtual machine. This also means they can be rolled back to a previously working state if a payload manages to break the device. VirtualBox will be used to emulate both Android and Windows platforms for testing QR codes.

Java, Kotlin, C++, C, Assembly: These are the main languages used in Android application development. While the vast majority are written in Java, where space or speed is concerned it sometimes becomes necessary to write in C or Assembly language.

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.

Libraries

ZBar Library: A popular, well-known QR code reading library. It is open-source and has a very small memory footprint. It is the main library used in the ZBar Android app. ZBar is available in most repositories and can be used on the command line and in scripts.

ZXing Library: Another popular QR code library that has a corresponding Android app. It is written in Java, with additional ports to other languages.

Operating Systems

Windows: ZBar is available on Windows and can be used for scanning and testing QR codes that contain malicious strings or executables.

Linux: Linux offers a number of robust command line tools for generating and testing QR codes. The ZBar library was written mainly for the Linux platform and is utilised in many Android QR code scanners and in many embedded systems around the world. Kali Linux will be used for one of the attacks in section 2. Everything else will be done on MX-Linux which is a Debian-based distro.

Android: Android is the world's most popular mobile OS and the vast majority of QR codes are scanned using an Android app.

Main Deliverables

Section 1: A command line tool/program for creating and testing malicious QR codes in Linux. The tool should be able to quickly generate numerous malicious QR codes based on both default and custom wordlists. A user should be able to pick from a variety of QR code types e.g. WiFi login, send an SMS, etc. and generate the desired QR code. These will be stored in a gallery to be scanned manually by a real device or automatically sent to a parsing library for testing. This will be done using ZBar as the main test library. QR code generation will be handled by libqrencode.

Malicious codes will be passed to the ZBar library for parsing and the results will be printed to screen and written to a logfile. This is to see if any QR codes were not read correctly or if they managed to somehow break the parser. This only really works for string encoded QR codes as binary QR codes will display odd symbols that cannot be read or understood by humans. The tool should be able to test both single and multiple QR codes.

The program will also allow for encoding binary data (such as apk files) into QR code format. These potentially dangerous binaries should only be tested in a properly sandboxed virtual machine. All of these binary files cannot be larger than the maximum of 2,953 bytes offered by a v40 QR code. While there are systems that allow for scanning data across multiple QR codes these are rare and fall outside of the specifications of this tool.

The tool should also be able to display the created QR codes in a slideshow fashion testing with a real hardware-based device such as a ticketing or postage system.

Section 2: This section is made up of 3 attacks that utilise QR codes. These attacks are theoretical and will be conducted in a controlled environment. The QR codes used in the second and third sections will be created using the tool developed in the first section.

1. A phishing campaign using QR codes that are advertised as a survey or competition but lead to a phishing URL. This is a simple, practical attack that can be implemented easily but highlights the dangers presented by QR codes lack of human-readability and perceived trust in the surrounding poster information. The phishing site will be done as a SurveyMonkey or Google Forms survey. This attack is not particularly advanced in terms of technical requirements but, given the prevalence and danger of phishing attacks (Rosenthal, 2021), it would be remiss of the project to not cover them in some regard.

2. A second attack using a QR code as an automatic Wi-Fi login that sets up a man-in-the-middle position for the attacker. Since QR codes can be used to quickly authenticate with a wireless network this attack demonstrates an attack where a user connects to an attacker-controlled network unknowingly and an attacker can sniff and decrypt the network traffic. This will be done using an Alfa AWUS036ACH wireless adapter, Kali Linux, and wifipumpkin3 which is a wireless hacking framework akin to Metasploit.

3. A third attack focusing on the use of UUSD (Unstructured Supplementary Service Data) and MMI (Man-Machine-Interface) codes. This attack is limited in that it requires the use of actual hardware. This is because UUSD and MMI codes require a sim card and a cellular

connection in order to function correctly. UUSD and MMI code can be encoded into QR codes. These will then be tested on a real device to see which codes execute successfully and if device manufacturers have put any security controls in place. These codes will be stored as part of a custom wordlist in Section 2 for reuse depending on the hardware to be tested.

Section 3: A QR code containing a small apk will be scanned and sideloaded onto an Android device. It will then ask for permissions before installing and, upon successful installation, demonstrate some simple behaviour. If a malicious payload is to be used and executed this will be done only inside a properly sandboxed environment. This is a proof of concept to show a piece of malware could be placed into a QR code and executed when scanned. A real virus would try to exploit a vulnerability in the camera or QR scanner and then execute on the device without the user's knowledge.

Assumptions

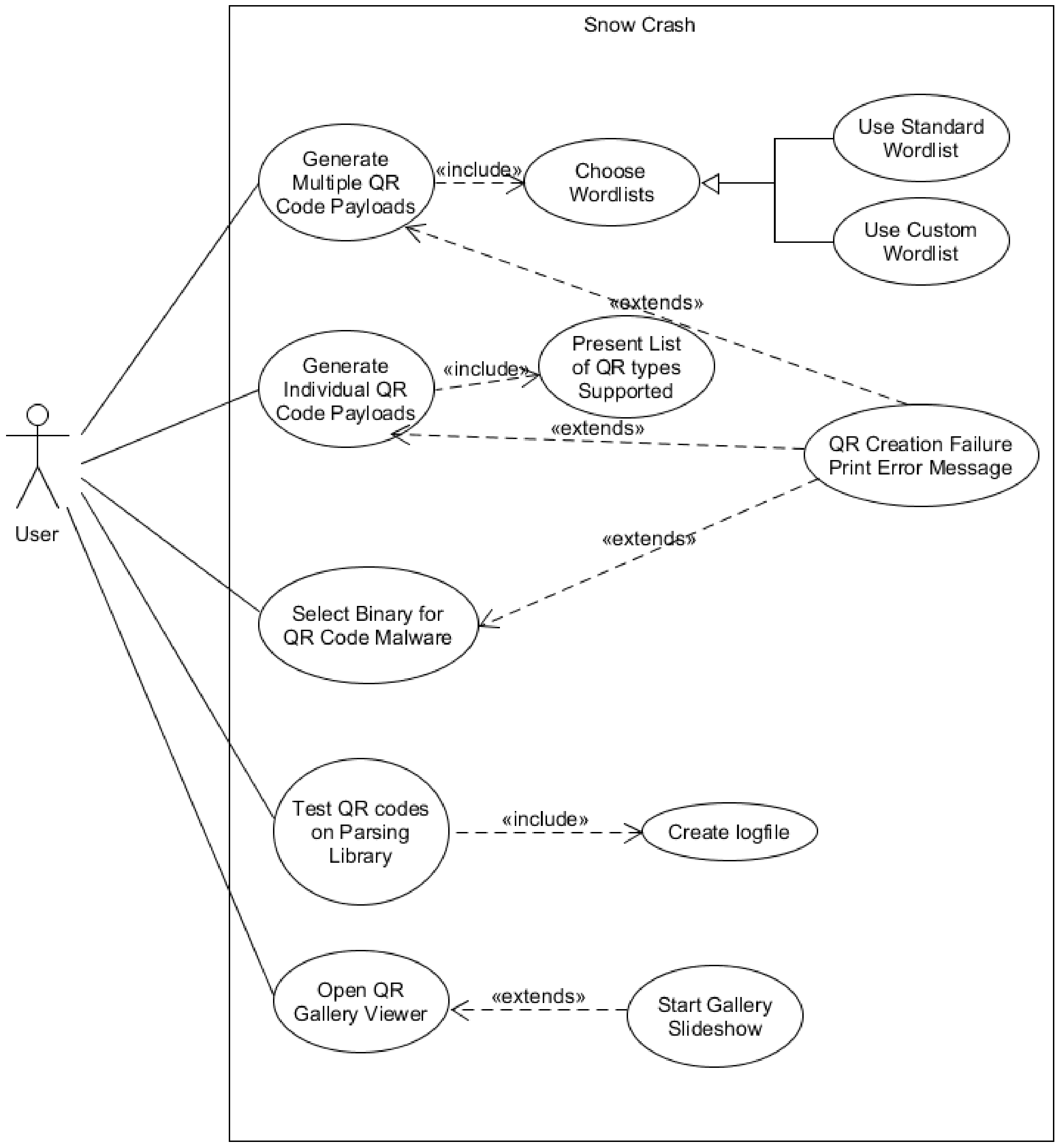
- I will have time to plan, execute, and properly document every attack outlined in this document.
- That every section is technically feasible and within my abilities e.g. that it is possible to sideload and install an apk to an Android device via a QR code.

Risks

- Time runs out before all deliverables are completed.
- Some part of the technology stack does not work as intended or takes too much time to set up properly.
- While every precaution will be taken, there can be no guarantee that some malicious QR code will not cause damage to software or hardware during testing.

Use Cases

Section 1


Snow Crash
User
Generate Multiple QR Code Payloads
«include»
Choose Wordlists
Use Standard Wordlist
Use Custom Wordlist
«extends»
Generate Individual QR Code Payloads
«include»
Present List of QR types Supported
«extends»
QR Creation Failure Print Error Message
«extends»
Select Binary for QR Code Malware
Test QR codes on Parsing Library
«include»
Create logfile
Open QR Gallery Viewer
«extends»
Start Gallery Slideshow

Generate multiple QR codes using wordlists method.

Actors	User
Preconditions	User has installed the tool on a Linux system running either BASH or ZSH.
Actions	1. The user starts the tool and selects the option to create multiple QR codes from a wordlist. 2. They can choose to use either standard or custom wordlists. 3. The user can then select individual wordlists they want, and the corresponding QR code payloads will be created.

Expected Results	QR codes will be created and placed in a folder with the wordlists name e.g. the wordlist “01-sql-injection” will create a folder of the same name and will contain QR code containing SQLi payloads.
Alternatives	User edits a standard list or provides a custom list with a value that cannot be turned into a QR code. This results in an appropriate error message.

Generate individual QR code.

Actors	User
Preconditions	User has installed the tool on a Linux system running either BASH or ZSH.
Actions	1. The user starts the tool and selects the option to create an individual QR code. 2. They are presented with a list of supported QR codes e.g. send SMS, login for a Wi-Fi network, SEPA Payment, etc. 3. Selecting an option from the list will ask the user to fill out the appropriate fields before creating the QR code. 4. Users can also supply values/string manually using the manual option.
Expected Results	The user supplied values are turned into the appropriate QR code and stored using a filename supplied by the user.
Alternatives	User provides a value that cannot be turned into a QR code. This results in an appropriate error message.

Generate a QR code containing executable binary code.

Actors	User
Preconditions	User has installed the tool on a Linux system running either BASH or ZSH.
Actions	1. The user starts the tool and selects the option to create an individual QR code using a binary as the input. 2. They select the file using a built-in file explorer and the QR code is created.
Expected Results	The user supplied file is turned into a QR code and stored using a filename supplied by the user.
Alternatives	User provides a file that cannot be turned into a QR code. This results in an appropriate error message.

Test QR codes with ZBar QR library

Actors	User

Preconditions	User has installed the tool on a Linux system running either BASH or ZSH.
Actions	1. The user starts the tool and selects the option to test QR codes. 2. They select the individual file(s) or folder(s) they want to test and start the testing procedure. 3. The tests will be written to screen and out to a logfile.
Expected Results	ZBar will store the QR code output to a logfile.
Alternatives	The program passes a QR code to ZBar that cannot be properly read. This results in an appropriate error message being written to the logfile.

Viewing QR codes.

Actors	User
Preconditions	User has installed the tool on a Linux system running either BASH or ZSH.
Actions	1. The user starts the tool and selects the gallery option. 2. They can then choose to view single QR codes or start a slideshow of multiple QR codes. 3. Viewing options are selected. 4. User starts slideshow.
Expected Results	The appropriate QR codes are displayed on screen either statically or in a slideshow.
Alternatives	None.

Section 2, Attack 1

Actors	Victim, Bad Actor
Preconditions	Bad Actor creates phishing QR code and tests that it works.
Actions	1. Bad Actor hangs up a poster with phishing QR code. 2. Victim scans the QR code. 3. Google Safe Browsing API or similar provide minimal or now warnings i.e. phishing URL is not blocked. 4. Victim visits phishing URL and enters details. 5. Bad Actor steals victim’s private information and uses it another attack e.g. credential stuffing attack.
Expected Results	Bad Actor is able to successfully phish victim’s information.
Alternatives	URL is blocked by QR scanner and user avoids getting phished.

Section 2, Attack 2

Actors	Victim, Bad Actor
Preconditions	Bad Actor creates Wi-Fi Login QR code and places in public pace. Bad Actor creates appropriate corresponding wireless AP.
Actions	1. Victim notices and scans Wi-Fi Login QR code. 2. Victim is shown captive portal that asks for email or similar details. 3. Victim enters details and is connected to rouge AP. 4. Bad Actor can sniff network traffic and/or inject packets to send back to the victim.
Expected Results	Bad Actor successfully sets up MITM attack and captures sensitive user.
Alternatives	QR reader blocks URL from loading and phishing attack is prevented.

Section 2, Attack 3

These attacks involve scanning QR codes containing UUSD and MMI codes. 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). UUSD and MMI codes rely on cellular networks and sim cards and so must be conducted on real hardware instead of in an emulator. Since some codes are manufacturer-specific this will not be thoroughly conclusive as this project does not have the funding to purchase multiple devices from various manufacturers. Every common UUSD and MMI code available will be stored in a custom wordlist and provided with the tool created in section 1.

Section 3

Section 3 will attempt to fit a small Android apk into a QR code (upper limit 2,953 bytes). This apk will then be sideloaded and installed on the device using only the QR scanner. The first step will be to try this with a harmless hello-world apk. If the delivery method works, the next step will be to improve the functionality of the apk to include extra features. Anything that could be considered malware or dangerous will be tested in a sandboxed environment using the appropriate security controls.

This will be the most challenging part of the project. Several previous attacks using QR codes have utilised their ability to make payments automatically or to send a victim to a malicious URL either for phishing or to run malicious JavaScript in the victim's browser (e.g. cryptominers). To my knowledge none to date have tried to squeeze a malicious binary executable inside a QR code. The earlier sections will confirm that these previously demonstrated attacks are still possible and dangerous. This section will attempt to push the envelope.

In the event that this is not feasible on Android the backup will be to perform the same attack on the Windows platform. From my research I know it to be possible to load an executable from a QR code. The goal will be to place a malicious .exe file into a QR code, read it out, and execute the program on a Windows 10 machine. This will likely be written in C or x86 assembly and compiled using Visual Studio.

FURPS+

FURPS+ is an acronym for the headings below. It is a model, originally developed at Hewlett-Packard, used to classify software attributes. It is used to describe the requirements of a project, both functional and non-functional.

Functionality

Core functionality has been described in the use cases above. A user should be able to produce and test numerous QR codes using wordlists or custom binaries. Testing results will have a synopsis printed to screen and a detailed log file produced as well. Users will also be able to display QR code images in a slideshow fashion.

Usability

The tool is a command line utility with some TUI (Text-based User Interface) elements. It should be quick and responsive. The navigation, selections, and options should be clear and apparent without additional documentation or explanation. The TUI based interaction will be limited to number keys, space bar for selection, and arrow keys for navigation. If there is an error the tool will attempt to recover and print an informative error message.

Reliability

The tool has limited functionality (do one thing and do it well) and should be able to recover from a user supplying input that cannot be turned into a QR code. This type of incorrect input will be main source of errors. The tool operates on the command line and should start quickly and reliably like other terminal-based tools e.g. htop, cmus, youtube-dl, etc.

Performance

QR code creation and testing should prioritise accuracy and precision over speed. A QR code must be encoded correctly for it to work. The tool is a command line utility with minimal overhead and should function on a system with limited resources.

Supportability

The tool is intended to be a Linux based tool. It should function on any Linux system running Bash or ZSH. There will be some dependencies that will be listed in the documentation.

Additional

There are no additional attributes of the project.

Inspirations

My initial inspiration came from a video titled "Can you fit a whole game into a QR code?" by MattKC. Matt managed to squeeze an entire game of snake into a v40 QR code and got it to execute on Windows using a webcam. In Matt's case the game .exe ran without any prompts or warnings which he highlighted as a security issue in his video (MattKC, 2020). This got me thinking about the idea of storing a virus or malicious payload inside a QR code.

The concept of a virus in a QR code is similar to the "Snow Crash" virus in Neal Stephenson's 1992 Sci-Fi novel of the same name. In the book the Snow Crash virus, a black and white matrix pattern, can be shown to people inside virtual reality or on a screen and causes real-world cerebral damage. The goal of this project is not to cause any physical harm but rather to explore the possibilities of some novel forms of cyber-attack. I shamelessly borrowed the title of this project from the novel.

Our smartphones have become central to our modern lives, carrying a wealth of personal information, and identifying metadata. This makes them a prime target for bad actors. Especially given the recent COVID-19 pandemic we have seen an increase in the use of QR codes for checks-ins with COVID tracker apps, for payment means, and even for restaurant menus. This is done in the interest of hygiene and so people can avoid touching possibly contaminated surfaces.

Metrics

Section 1 is devoted to creating a tool for creating and testing malicious QR codes. Section 2 and 3 are example attacks using QR codes created using the tool from Section 1. Below are some of the functional requirements the different sections should fulfil.

Section 1

Functional Requirements

1. Create malicious QR codes and place them in appropriate folders.
2. Successfully use custom wordlists used for QR creation.
3. Provide a comprehensive set of inbuilt wordlists.
4. Create QR files from binary executables.
5. Provide a means for a slideshow of QR codes for hardware cameras.
6. Run tests against ZBar QR reading library and log the output.
7. Create logfiles and provide feedback and debug information on how the parser handled the malicious QR code.

Non-Functional Requirements

1. Provide the option for incorporating other libraries like ZXing or custom user libraries.
2. Provide a means for creating other types of 2D barcodes e.g. Aztec codes.
3. Provide a means of placing 2D barcodes inside one another for barcode inception style attacks described by (Dabrowski, et al., 2014).
4. Provide a means of creating artistic/stylish QR codes and accompanying posters with text.

Section 2, Attack 1

Functional Requirements

1. QR code works as intended and directs victim to phishing site.
2. URL loads with minimal or no warnings.
3. URL is not blocked by Google Safe Browsing API.
4. Sensitive data can be retrieved by an attacker.
5. Comparison of different QR scanners and see how they behave.

Non-Functional Requirements

1. Conduct a real phishing campaign and collect data on QR usage and effectiveness as a means of phishing. Compare data against a mass phishing campaign sent via email and against a more targeted spear-phishing campaign.
2. Create multiple posters and see which is most effective in drawing potential victims.
3. Test URL forwarding capabilities on a custom domain to evade Google Safe Browsing and similar APIs.

Section 2, Attack 2

Functional Requirements

1. Create a QR code that logs a user into a network automatically.
2. Set up an AP with Kali to sniff user traffic.
3. Create and present a fake captive portal to the user.
4. Harvest credentials from the portal.
5. Compare different QR scanners and see how they behave.

Non-Functional Requirements

1. Demonstrate packet-injection or other advanced MITM techniques.

Section 2, Attack 3

Functional Requirements

1. Get any MMI code to execute on a phone via QR code.
2. Get any UUSD code to execute on a phone via QR code.
3. Create a custom wordlist of various UUSD and MMI codes.
4. Create a corresponding gallery of UUSD and MMI QR codes.

Non-Functional Requirements

1. Get a dangerous UUSD or MMI code to execute e.g. change PIN code, factory reset data, call premium number, etc.

Section 3

Functional Requirements

1. Keep any apk or exe used under the upper size limit of 2,953 bytes.
2. Sideload an apk onto a device using only the QR scanner.
3. In the case of Windows, load and execute the exe without being blocked by an anti-virus or Windows Defender.
4. Add features to the initially benign apk or exe.

Non-Functional Requirements

1. Get the apk onto the device without "install unknown apps" setting turned on i.e. exploit a vulnerability in the camera or QR code reader.
2. Add ransomware capability to the apk or exe.
3. Add the ability to communicate with a remote C2 server.

Bibliography

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

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.

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