6.857 Project: QRAttack

Eric Pence, Rami Rustom, and Lindsey McAllister

May 20, 2021

Contents

1 Introduction

QR codes are increasingly ubiquitous in public spaces for a wide variety of purposes, but they were never designed for security. While most QR codes in the wild are legitimate, malicious actors can use this capability for a variety of attacks. They can direct users to phishing sites, have users download malware, or force reset users' phones, just to name a few [2,3,4]. The COVID-19 pandemic's emphasis on contact-free communication has made these potential attacks all the more worrisome: everything from restaurant menus to contact tracing now uses QR codes [5]. Since QR codes are not human-readable and efforts to customize or aesthetically alter QR codes have seen limited success [6], it is difficult for individuals to discern between safe and dangerous QR codes. Some scanners include various security measures, but there are a wide and heterogeneous variety of apps and scanning mechanisms in use [7].

Given this landscape of several mechanisms for exploitation, we were interested in attacks that involve malicious manual edits to QR codes (think of using a back-pocket sharpie). Our ultimate goal was to build a tool that scans a QR code, and calculates the minimum number of edits (with a sharpie or black pen) that a person needs to make in order to change the QR code payload into any payload of their choice. Our tool would also be able to sabotage a QR code and destroy it so it becomes unreadable to a scanner. We successfully built such a tool [16]. Further, we are interested in whether people will still scan a code that someone has edited with a sharpie after using our tool. We hypothesize that people will still scan manually doctored QR codes in the wild, and we wanted to test this empirically with a user study. If this is in fact true, such a tool would highlight the very real danger of naively displaying or scanning QR codes.

2 Background

2.1 QR Code Description

QR codes are two-dimensional bar codes developed in 1994 for manufacturing, and have become increasingly ubiquitous as a convenient, low cost mechanism for distributing digital

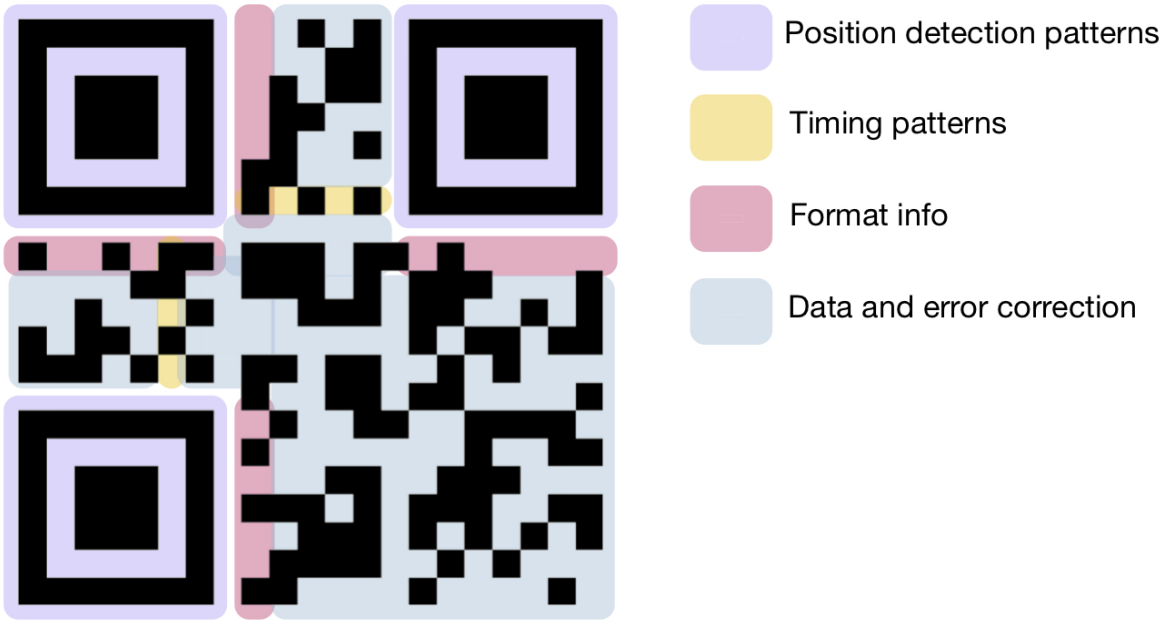
information. A QR-code can encode up to 4296 ASCII characters and several data types, including URLs, email-addresses, contact information, SMS/MMS/FaceTime codes, and wifi settings [1].

Each square on a QR code is called a module. There are versions 1 through 40, which are used to encode variable amounts of information; however, there are some features which all QR codes share, each highlighted in Figure 1. First, the position detection patterns are the large squares around a square in three corners, which help the scanner determine the orientation. They are always 7 modules by 7 modules surrounded by a quiet zone of width 1 module. Next, there are timing patterns, which help the scanner determine how large the code is. These timing patterns consist of one horizontal and one vertical strip of alternating black and white modules. Then, there are spaces reserved for format information and version information. The scanner reads these first once oriented to know what kind of code to expect and how to read it [14]. The format information includes the error correction level and the mask pattern. It is always 15 bits: 5 bits for the error correction level and mask pattern and 10 error correction bits, which are XOR-ed with a mask pattern (101010000010010). There are four error correction levels: low, medium, quartile, and high, which each leave different amounts of space on the code reserved for error correction modules. The mask is one of 8 patterns applied across the data modules of the QR code to invert some of the modules to prevent large chunks of same-colored modules that are hard for scanners to read. The version information specifies which of the 40 versions the code is. Finally, there are the data and error correction modules, which encode the actual information stored in the code. QR codes use Reed-Solomon error correction, and the ratio of data to error correction data modules depends on the code's level of error correction, or how much of the code could be lost and still work. [15]

2.2 QR Code Editing Potential

QR Codes still work when manually edited. The scanners in modern-day smartphones can tolerate imperfect edges and angles. Some businesses decoratively use colors or non-square shapes in their displayed QR codes for aesthetic purposes, but the same property that makes this work also makes manual alterations work. Drawing on a QR code with a marker can

Figure 1: Format of QR codes



induce the scanner to read once-white modules as black ones. The layout of QR codes and existing content on a QR codes does restrict how much someone with a marker can change the payload of the code.

3 QRAttack Tool

3.1 Description of Destroy Algorithm

Our first goal with our tool is destroying QR codes. The tool would scan an existing target QR code and give the user an image of the modules they should color in order to break the target QR code. This means a QR code reader should not be able to scan the code at all. Further, we want the QR code to still appear legitimate. Breaking it by scribbling over the entire thing is not an interesting attack.

We targeted the modules that encode the formatting information. Since the scanner needs to know the error correcting level and the mask in order to read the modules that encode the payload, removing the modules that specify them as well as all the error correction modules

prevents the scanner from reading the code at all. Because of the mask and error correcting modules, the 15 modules of the format information should never all be black by themselves.

In every QR version, there are 15 modules in the same location relative to the the position detection patterns. The four sections that contain these 15 modules are immediately adjacent to the edges of some of the position detection patterns, as seen in Figure 1. Since the position detection patterns are always 7 modules in width and height, our tool can easily locate them, and thus format information. An example of a working code and then our tool's alteration of that code to not work are shown in Figure 2.

Figure 2: A working QR code and the output of the "destroy" option of our tool

3.2 Description of Change Algorithm

Our algorithm for changing a target code into a malicious one is borrowed from [13], with a key optimization that allows for the algorithm to be computed efficiently. We summarize the algorithm below, and describe our optimization in the next section.

As described above, the attack setting consists of an adversary with a black marker and a public QR code (ex: a restaurant menu, brochure, business card, etc...). Thus, the adversary can only change white modules to black. The attack outlined in [13] consists of the following steps:

1. Scan the target QR code Q_T, and get the target message M_T. We will assume this is a URL.

2. Generate several malicious candidate messages M_i for $i = 1, ..., n$ such that they visually resemble the target message.

3. Generate the QR codes Q_i for $i = 1, ..., n$ corresponding to the messages. Use the same version and mask as the target code Q_T.

4. Calculate the symmetric difference $D_i = Q_T \triangle Q_i$ for each candidate code. This represents the set of modules in the same position the differ in color between the candidate and target codes. Since the QR codes can be represented as 2D matrices with the black squares as 1 and the white ones as 0, this can be computed using the element-wise XOR of the two matrices.

5. Calculate the matrix $R_i = Q_i \wedge D_i$, which is the logical AND of the candidate and symmetric difference matrices. This represents the modules in the candidate code that need to get changed from white to black. Then, compute the ratio
$$r_i = \frac{\| R_i \|_1}{\| D_i \|_1}$$
of modules that need to be changed from white to black to the number of different modules using the 1-norm.

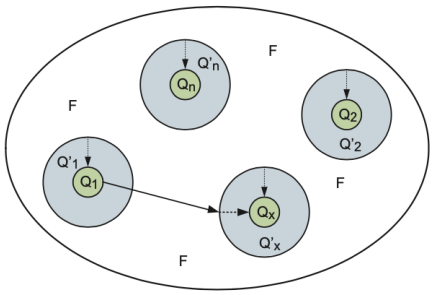
6. Order the candidate codes by r_i by descending order, omitting codes where the number of codewords (8-bit contiguous sections) that need to be changed is greater than the error correction level of the code.

7. For each candidate code in order, color the white modules in Q_T that are black in Q_0 black, and check whether the code decodes to something other than M_T. This can be done using the element-wise OR operation $Q'_i = Q_T \vee R_i$. If it does, then a malicious message M_i has been found.

The intuition behind this attack is that each target QR code lies at a point Q_i in the space of possible codes, and this small region is expanded by using the error correction codes to a space Q'_i, such that any code in that space will map back to Q_i. This simplifies the task of finding collisions, since we only need to find any overlap between two regions Q'_i and Q'_j for some malicious code Q_j. This can be visualized in Figure 3, which we borrow from [13].

The authors also emphasized that the generated malicious URL should resemble the target URL, since oftentimes in mobile QR codes readers (usually, the camera application), the URL is displayed to the users.

Figure 3: A visualization of the search space



3.3 Optimizations

The paper leaves out a crucial detail which determines the feasibility of this attack: how to generate the candidate messages M_i. As stated above, we are optimizing for similarity to target payload M_T for the purposes of executing a practically feasible phishing attack.

Considering strictly messages that are URLs, we can break up the payload of the target QR code into the following parts:

1. Scheme: This is the network protocol, and for our purposes is usually either "http://", "https://", or omitted entirely.

2. Domain Name: This is the website domain, and can potentially include subdomains of the form "subdomain.domain.TLD", where TLD is the top level domain (ex: .com, .org).

3. File Path: This region contains anything after the slash, and before any query parameters.

4. Parameters: This contains any HTTP GET query parameters

5. Anchor: This includes anything after the octothorpe (#), and points to a certain location on the webpage.

In our implementation so far, we have only taken the scheme, domain name, and path into consideration.

We experimented with several methods for generating payloads and thus QR codes that are similar, primarily relying on Hamming distance and Levenshtein distance. The former

Figure 4: Parts of a URL

simply consists of character substitutions to the original message, and the latter additionally includes additions and deletions of characters. Given that the message can potentially be modified but not shortened because the message length is fixed in the QR code, we decided to focus on the Hamming Distance.

Since each character in the message could be modified, the search space is exponentially large and thus impractical for a feasible attack. By experimenting with different messages, we discovered that modifying just the domain name is sufficient to find messages reliably and efficiently. It is unlikely that changing the scheme leads to anything useful (hutps isn't a valid scheme) and any arbitrary path can be created by the malicious agent if she owns the domain. We therefore decided to keep the same scheme, TLD, and path the same from the target QR code.

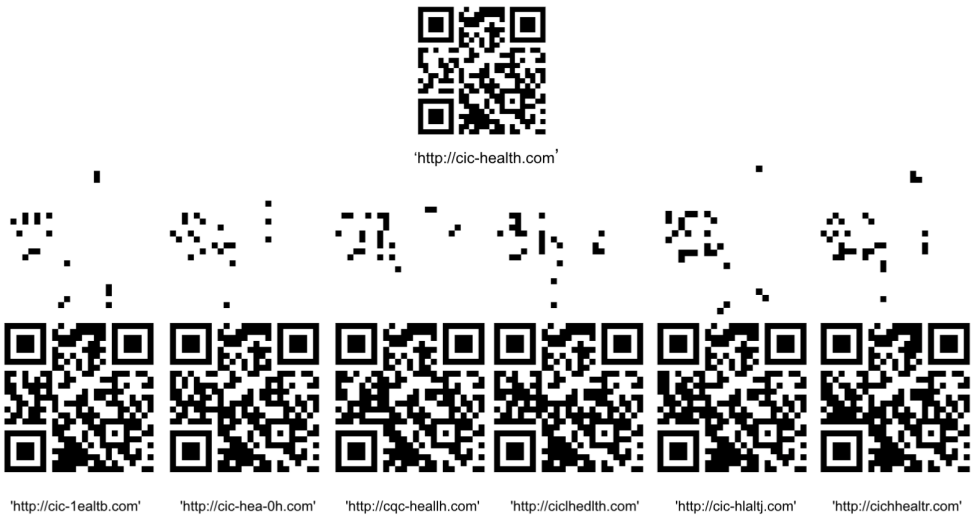
We apply the algorithm described above on the messages $M_{i,h}$ for $h = 1, ..., |M_T|$, where h is the Hamming Distance. We therefore run the algorithm on all messages with increasing Hamming Distance from M_T until a successful candidate is identified. By Hamming Distance, we are referring to the number of characters that are substituted, not the bits of the ASCII representation. We do this until a valid target code - one that encodes a new message changing only white to black modules - Q_T is found.

3.4 Results

We implemented this algorithm and ran it on a web server. We built a simple web interface that uses a device's camera to scan the target QR code. Once it recognizes the code, it displays the payload and offers the user the option to either destroy or tamper with the code. The interface outputs the new QR code with the modules a user would need to color in highlighted in red if it finds a valid one. We have tested this on several codes and gotten consistent results. An examples is pictured in figure 6 below, which shows the QR code for Massachusetts Covid vaccine administration (shown at vaccination sites!) and the changes

that could be made it to make it lead to a different site. The different site that it would lead to is listed below each set of new QR codes, and the difference in modules between the target and the malicious codes.

Figure 5: A working QR code and all the valid outputs of the "tamper" option of our tool



3.5 Further Investigations

To further validate the potential impact of this attack, we evaluated our attack on QR codes using commercially available, popular QR code generators that targets are likely to use in the wild. We use qr-code-generator.com [17], qrcode-monkey.com [18], qrstuff.com [19], and shopify.com [20]. We were curious about the Error correction, version, and mask values that they used on their QR codes. We also compared these values to the optimal assignment of those values using the Python library qrcodegen [21], which optimizes for using up all of the bits in a QR code, and maximizing error correction. We present these results below, as well as some of the malicious messages we were able to produce. In the first table, for each generator, we try a variety of URLs, trying out different combinations of schemes, domains, subdomains, TLDs, and filepaths. For each, we report the error correction level, version, and mask, respectively.

Figure 6: The web app allows a user to first scan the QR code, choose their attack type, then highlights the modules to color in

M_T	qr-code-generator	qrcode-monkey	qrstuff	optimal
http://cic-health.com	0, 2, 2	1, 2, 2	0, 2, 6	1, 2, 2
https://flourbakery.com/menu	0, 2, 2	1, 3, 2	0, 2, 3	0, 2, 4
https://news.mit.edu	0, 2, 2	1, 6, 2	0, 2, 0	2, 2, 6
https://news.mit.edu/press	0, 2, 2	1, 2, 6	0, 2, 0	1, 2, 6
https://bit.ly/3f7yVJ0	0, 2, 2	1, 2, 3	0, 2, 0	1, 2, 6

M_T	Collision
http://cic-health.com	None
https://flourbakery.com/menu	https://flourbawury.com/menu
https://flourbakery.com/menu	https://olourbakery.com/menu
https://news.mit.edu	https://dews.mmt.edu
https://news.mit.edu/press	https://nfwq.mit.edu/press
https://bit.ly/3f7yVJ0	https://cit.ly/3f7yVJ0

We were able to find collisions using only Hamming distances of 1 and 2 using all of the websites for almost all of the codes above.

4 Experiment

4.1 Methods

To assess whether people actually scan manually doctored codes and thus how dangerous our tool really is, we conducted a user study. We also wanted to know if the degree of obviousness of manipulation affected scanning behavior. To do this, we displayed both normal QR codes and ones manually modified to various degrees on posters and recorded scanning rates. We hypothesized that many people would still scan doctored QR codes.

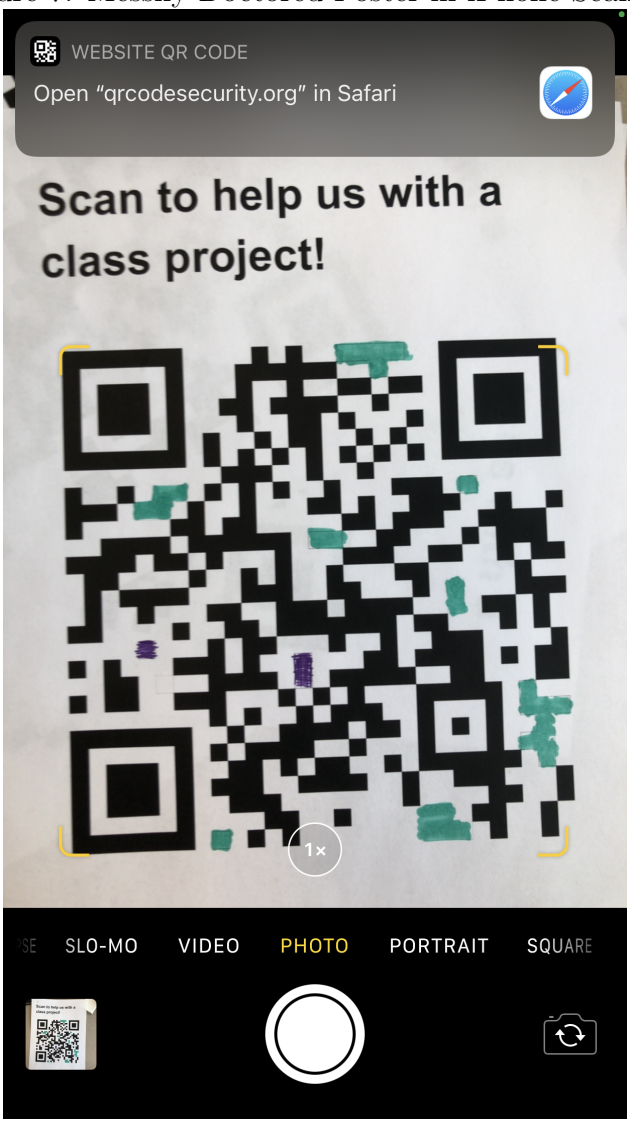
We chose four locations on the MIT campus to hang posters with a QR code. Locations 1, 2, and 3 were undergraduate dorms, and Location 4 was an on-campus Covid testing center. The posters read, "Scan to help us with a class project!" followed by a large QR code. Each location had 2 different versions of the poster: one with a control QR code with no modifications and the other with modifications with a sharpie. We also included different types of modifications. For locations 1 and 2, the modifications were neatly colored-in with black sharpie. For locations 3 and 4, the modifications were outside the lines, in different colors, and imperfectly filled in.

All posters led to different sites with identical content about QR code security. Each version at each location led to a different page so that we could count the number of times those posters had been scans. We kept a timestamp of every page visit but no other information. We left the posters up in each location for 4 days.

4.2 Results

Each page visit triggered a recording of the page identifier and the timestamp on our server so that we could evaluate scanning rates. Figure 8 shows the total scans of each type of poster at each location, and figure 9 shows the hourly number of scans of each type over time.

Figure 7: Messily Doctored Poster in iPhone Scanner



4.3 Analysis

Our results indicate that people still scan obviously doctored QR codes, even though it may be at lower rates. Interestingly, they scan messily doctored ones at about the same rate as regular ones, but scan neatly doctored ones at about half the rate. We think this may be because the messily doctored ones with bright colors and obvious markings are more visually interesting, so people may be scanning them out of curiosity.

While the raw results at location 3b suggest people may scan the messily doctored ones at even higher rates than the normal ones, we think that about 20 of the scans may not be

Figure 8: Poster Scan Count

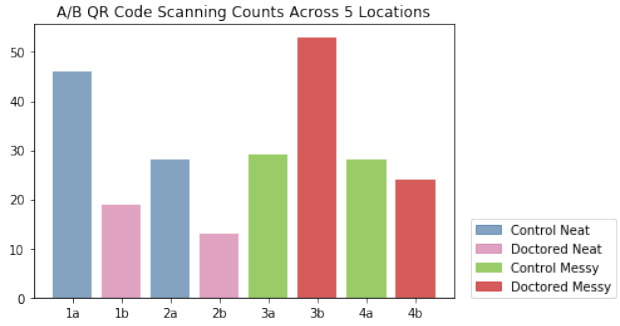
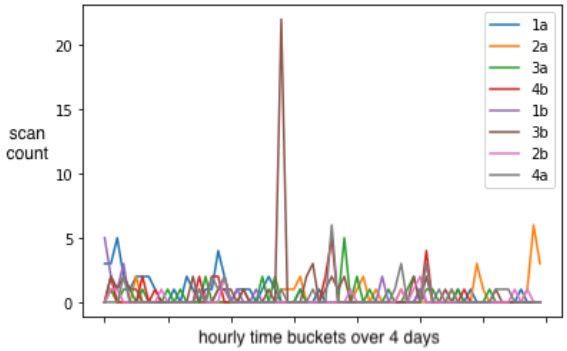


Figure 9: Scan Counts Over Experimental Period



legitimate ones. Looking at the timestamps, the activity across all versions looks spread out over time and relatively consistent with each other; however, the anomaly is the 3b poster set, which was scanned 22 times between midnight and 1am the first night they were up. We believe this may have been a single user or group of users who scanned several times out of curiosity or for humor. Subtracting 20 scans from the 3b count brings it very close to its control group, 3a, which fits with the idea of users scanning this type of modification at the same rate they scan other kinds of modifications.

Ultimately, this experiment confirmed that people do in fact scan QR codes that are manually doctored. Messy alterations even increase this rate. This indicates that in the wild, a malicious agent who draws on QR codes can do so with the reasonable expectation that users will scan them. This makes the potential our tool offers to maliciously edit QR codes dangerous in practice.

5 Potential Extensions

As we look toward future work, there are several next steps we plan to take. The first and simplest is examining how we can manipulate Top Level Domains (TLDs) and their positioning within a URL to expand the space of potential malicious URLs that we generate. In our current implementation, we never consider different TLDs (e.g. '.com') when generating our list of potentially malicious QR codes. We only perform manipulations on the subdomains and file path (i.e. what comes after the '/'). This means that we never attempt to convert '.com' into '.org', for example. Since there are over a thousand top level domains, we are drastically reducing the size of our search space by not considering these possibilities. Additionally, we never attempt to shift the location of the TLD as we generate malicious URLs. If 4 characters come before the TLD in the original URL and 5 characters come after, we only consider malicious URLs with 4 characters before the TLD and 5 after. We could dramatically expand our search space by considering shifts in the positioning of the TLD. With these changes comes the task of determining whether expanding our search space in such a fashion yields a higher likelihood of finding a collision. While we can benchmark this likelihood through extensive testing, ideally we could derive this likelihood formally, which motivates the second and third key steps in our future work.

Second, we would like to determine if we are guaranteed to find a transformation, using only black to white modifications, from any given QR code to a malicious one given sufficient time and computation resources. Answering this question will require us to understand the size and composition of the entire space of QR codes encoding URLs. This information will help us to determine if every QR code that encodes a URL has a 'parent' which can encoded with a larger number of black squares. In order to determine this, we will need to examine the underlying structure of Reed-Solomon error correction further, as well as the effects QR codes masks have on the white-to-black mapping between QR codes. Importantly, when looking for a malicious QR code, we do not need to map directly from one un-altered URL encoding to another. Instead, we can map from one un-altered URL encoding to an error-corrected version of another URL encoding.

Third, we would like to determine the likelihood of finding a malicious modification that

results in a new URL within a given hamming distance of the original. For example, if we know our machine is only capable of testing QR code contenders within a hamming distance of 3, what are the odds we find a collision? Determining this will require us to discover the the mathematical relationship between the QR code representation of URLs that are similar in character composition. A good place to start on this work will be investigating the interaction between QR code masks and the UTF-8 or ASCII encoding schemes.

Fourth, we would like to explore what effect altering the version and mask, which is encoded in the same place on every QR code, has on our ability to find mappings to malicious QR codes. Initially, this exploration will take the form of trial and error. In this process we will find out if changing the version number and mask of a QR code yields a new, valid QR code. If so, we will have identified an easy way to create a malicious QR code. If not, we will have a 'broken' QR code but we may be able to determine if a valid QR code is reachable with white to black modifications. However, this will require us to solve the problem of identifying malicious codes from an entirely different angle. Instead of solving backward from URLs, we will need to solve forward from the black and white squares of a URL.

Fifth, we would like to examine QR code encodings of payloads that are not URLs. We would like to see if we are able to convert commonly used payloads, like URLs, into commands that force victims to dial phone numbers or alter their phone settings.

Finally, we would like to implement a simple feature in our back-end that searches Namecheap.com and other domain-buying platforms to see if the malicious URLs we identify are up for sale. Ideally, our back-end would purchase the URL if it is available, and set up a website automatically.

6 Potential Defenses

For individuals who wish to publish QR codes for the public to scan safely, we have several recommendations. The first is that they print the payload under the QR code. This is an easy, effective way to enable anyone scanning a QR code to verify that the URL their phone is headed to looks legit and matches what is encoded in the QR code.

Second, we recommend that publishers use the QR code version that is closest in size to

the payload they are encoding. This will limit the empty space in a QR code and reduce the space of possible malicious QR codes for adversaries to search.

Third, we recommend that publishers select that mask version that results in the largest number of black squares in the QR code. Most commercial QR code generators already do this, so publishers should not worry too much about doing this manually.

Fourth, we recommend publishers put QR codes in hard-to-alter places if possible. For example, putting a QR code inside a plastic holder or behind glass makes our attack much harder.

For individuals who wish to scan QR codes, we have one simple recommendation: don't scan QR codes from unknown or un-trusted sources. If it looks like it's been altered by hand, don't scan! We urge anyone scanning a QR code to read the URL encoded before opening it and double check that they are comfortable following the link.

7 Conclusion

QR codes were never designed for security, and a world increasingly inundated with QR codes with them may soon come to realize this. Our work highlights both the vulnerability of QR codes and the practical danger. We designed a tool that allows a malicious actor armed with a marker to destroy or tamper with QR codes in the wild. This ability could easily be used to redirect unsuspecting users to a malicious site. Underscoring the practical risk to such users in the wild, the empirical results of our user experiment indicate that obvious manual altering does not deter scanning. This means someone armed with our tool could realistically redirect users to malicious endpoints in the real world. This alarming conclusions highlights the need for continued work in this area and far greater caution from all users of QR codes. This information could help the individuals and organizations publishing QR codes determine whether it is in fact the best choice and collectively whether QR codes should even be used on the wide scale we see today.

Bibliography:

1. A. Kharraz, E. Kirda, W. Robertson, D. Balzarotti and A. Francillon, Optical Delusions: A Study of Malicious QR Codes in the Wild, 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Atlanta, 2014.

2. Peter Kieseberg, Manuel Leithner, Martin Mulazzani, Lindsay Munroe, Sebastian Schrittwieser, Mayank Sinha, and Edgar Weippl. 2010. QR code security. In Proceedings of the 8th International Conference on Advances in Mobile Computing and Multimedia (MoMM '10). Association for Computing Machinery, New York, NY, USA, 430–435.

3. Vidas T., Owusu E., Wang S., Zeng C., Cranor L.F., Christin N. (2013) QRishing: The Susceptibility of Smartphone Users to QR Code Phishing Attacks. In: Adams A.A., Brenner M., Smith M. (eds) Financial Cryptography and Data Security. FC 2013. Lecture Notes in Computer Science, vol 7862. Springer, Berlin, Heidelberg.

4. Krombholz K., Frühwirt P., Kieseberg P., Kapsalis I., Huber M., Weippl E. (2014) QR Code Security: A Survey of Attacks and Challenges for Usable Security. In: Tryfonas T., Askoxylakis I. (eds) Human Aspects of Information Security, Privacy, and Trust. HAS 2014. Lecture Notes in Computer Science, vol 8533. Springer, Cham.

5. Victoria Turk, WIRED UK. "In a Touch-Free World, the QR Code Is Having Its Moment." Wired, Conde Nast, 18 Aug. 2020.

6. M. Xu et al., "Stylized Aesthetic QR Code," in IEEE Transactions on Multimedia, vol. 21, no. 8, pp. 1960-1970, Aug. 2019

7. “Secure QR Code Scanner.” Sophos Search, docs.sophos.com/esg/smsec/7-0-5/help/en-us/mobile/concepts/QRCodeScanner.htm.

8. Kieseberg, Peter, et al. ”QR code security.” Proceedings of the 8th International Conference on Advances in Mobile Computing and Multimedia. 2010.

9. Krombholz, Katharina, et al. ”QR code security: A survey of attacks and challenges for usable security.” International Conference on Human Aspects of Information Security, Privacy, and Trust. Springer, Cham, 2014.

10. Peng, Kevin, et al. ”Security overview of QR codes.” Student project in the MIT course 6.14 (2014).

11. Narayanan, A. Sankara. ”QR codes and security solutions.” International Journal of Computer Science and Telecommunications 3.7 (2012): 69-72.

12. Tan, Longdan, et al. ”Robust Visual Secret Sharing Scheme Applying to QR Code.” Security and Communication Networks 2018 (2018).

13. Kieseberg, Peter, et al. ”Malicious pixels using QR codes as attack vector.” Trustworthy ubiquitous computing. Atlantis Press, Paris, 2012. 21-38.

14. ”History of QR Code.” QRcode.com, www.qrcode.com/en/history/.

15. Thonky. ”Format and Version Information.” Thonky.com

16. https://github.com/RRustom/qrattack

17. https://www.qr-code-generator.com/

18. https://www.qrcode-monkey.com/

19. https://www.qrstuff.com/

20. https://www.shopify.com/pos/qr-code-generator

21. https://github.com/nayuki/QR-Code-generator