Error-Correcting Codes as Source for Decoding Ambiguity

Adrian Dabrowski
SBA Research
Vienna, Austria
Email: adabrowski@sba-research.org

Isao Echizen
National Institute of Informatics
Tokyo, Japan
Email: iechizen@nii.ac.jp

Edgar R. Weippl
SBA Research
Vienna, Austria
Email: eweippl@sba-research.org

***Abstract*—Data decoding, format, or language ambiguities have been long known for amusement purposes. Only recently it came to attention that they also pose a security risk. In this paper, we present decoder manipulations based on deliberately caused ambiguities facilitating the error correction mechanisms used in several popular applications. This can be used to encode data in multiple formats or even the same format with different content. Implementation details of the decoder or environmental differences decide which data the decoder locks onto. This leads to different users receiving different content based on a language decoding ambiguity. In general, ambiguity is not desired, however in special cases it can be particularly harmful. Format dissectors can make wrong decisions, e.g. a firewall scans based on one format but the user decodes different harmful content.**

We demonstrate this behavior with popular barcodes and argue that it can be used to deliver exploits based on the software installed, or use probabilistic effects to divert a small percentage of users to fraudulent sites.

I. Introduction

Albertini [1] and Magazinius et al. [2] researched so called *binary polyglots* that are for example valid PDF, JPEG, and ZIP files at once [3]. In contrast to computer language polyglots (source code) which compile in different programming languages, these are binary formats carefully stuffed together to confirm to multiple file standards or at least be understood by those file parsers.

However, ambiguity is in general not desired. Especially when it leads to different decoders reading different content, e.g. a firewall scans a JPEG but the user decodes a ZIP archive with harmful content out of the same data stream. Jana et al. [4] and Alvarez et al. [5] have shown how to abuse file-type fingerprinting and parsing differences of anti-virus tools to evade detection.

On a more theoretical level polyglots lead to a language decoding ambiguity. This gets even more interesting when a decoder supports multiple languages but makes a hard decision on reading and interpreting it one way or another. Any ambiguity is therefore a potential security risk [4]–[6].

In this paper we show that error-correcting codes (ECC) are a convenient tool to construct and provoke such ambiguities. It is a generalization of our previous paper on *Barcode-in-Barcode Attacks* [7] which we will use for demonstration purposes.

The rest of this paper is structured as follows: In Section II we discuss the connection to Language Security and the necessary background on ECC and polyglots. In Section III we show how they can be combined and in Section IV and V we present a proof-of-concept implementation based on barcodes and their evaluation with different decoders. *Future Work* (Section VI) and the conclusion in Section VII discuss the implications, countermeasures and open problems.

II. Background

For this paper we will use *file format*, *data format* (e.g. on a network) and *language* interchangeably. After all, a file format specification defines semantics, symbols, and a grammar and therefore a language. A parser by itself also implicitly defines a language that under best circumstances is equivalent to the specification or at least contains the specification as a subset or substantially overlaps it.

Likewise, a *file*, a *transmission*, a *data stream*, or a single *data packet* are also the same on an abstract level: data arrives from an external source and has to be decoded and/or parsed (often multiple times, based on layered network and software structure) to be turned into an useful (internal) representation. This includes storage of data and transmissions over wire, radio, optical networks, or visual symbologies (e.g. barcodes, as used in this paper).

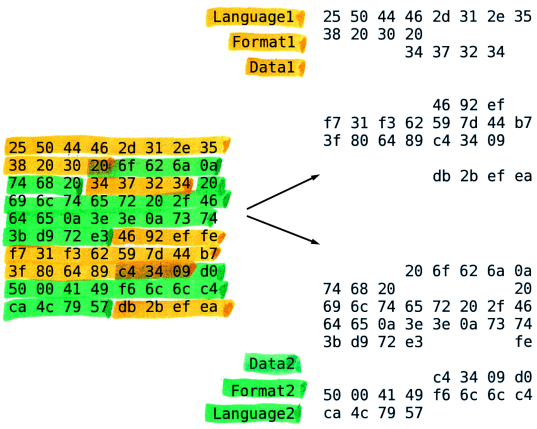

Figure 1. Visualization of a binary polyglot: data is interleaved using references with unused space in between, reserved or unused fields, etc.

A. Binary Polyglots

In computing, a polyglot used to be a source code valid in multiple programming languages. However, this is easily extendable to binary formats, as each parser forms its own language.

As seen in the visualization (Figure 1), some parts of the file are used exclusively for one format (language) or another, whereas other parts can be shared. This stuffing is made possible by the different format semantics, loosely validating parsers, lax or ambiguous format specifications, and extensive use of *comment blocks* as well as *ignored*, *reserved*, and fields *for future use*.

This way, for example, a file can simultaneously be a valid PDF document, JPEG picture, and ZIP archive. Typically, parsers start their work by finding an identifier and proceed from there. In above example, JPEG needs a correct signature string right at the start, PDF within the first 1024 bytes, and ZIP starts decoding archives from the end. (Binary) polyglots can be build as academic challenge, or to deceive file type dissectors and bypass scanning or filtering.

B. Error Correction

Forward error correction (FEC) is a technique used to cope with errors in data transmission over unreliable or noisy communication channels without the need for a reverse channel. The sender encodes their message in a redundant way by using an error-correcting code (ECC). Thus allowing the receiver to detect and correct a limited number of errors that may occur anywhere in the message without retransmission. Simple codes such as Parity and Hamming codes can only detect and correct a very limited number of errors, advanced codes such as Reed-Solomon, MDS codes, or Turbo Codes provide versatile configuration options. For example, on noisy channels, the code can be configured to allow 30% of the data to be damaged without an impact on consistency.

The model behind these techniques is a noisy transmission line, random physical errors on a data carrier, or interference in transmission. They have not been designed to withstand a crafted attack. In their effort to protect and reconstruct the original data they can be abused up to a point where (part of) the data can mean something completely different.

Extensive error correction is used in many applications such as digital radio (DAB) and video transmission (terrestrial and via satellite, DVB-T, DVB-S, DVB-C), cellular phones (GSM, UMTS, LTE), wireless networks (WIFI), digital tapes, hard discs, optical discs, raid arrays, flash drives, cloud storage, server RAM, and barcodes. We will use barcodes for demonstration purposes.

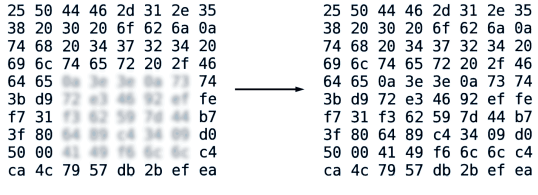

Figure 2. Visualization of an error-correction code: the destroyed data is reconstructed.

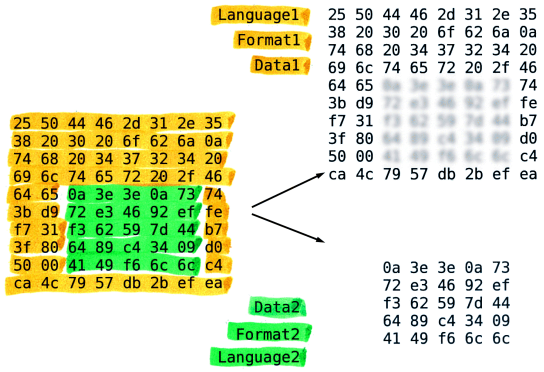

Figure 3. Visualization of embedding alien data into an ECC protected data: the alien data can overwrite big (consecutive) chunks as long as the error margin is not overstressed.

III. Error Correction as a Hideout

For the rest of the paper, ECC can be viewed as configurable black boxes for encoding data into code words and vice versa. Among other parameters, the configuration determines the recovering abilities (error margin); i.e. the fraction of destroyable code words without an effect on the decoded data. In normal operation the modified code words remain transparent to the reading application as error correction is typically done as one of the first steps in acquiring data (Figure 3). The amount of foreign data is limited primarily by the error recovering abilities of the used code. It is not dependent on the actual type of code used (e.g. Turbo Code, Reed-Solomon).

We can utilize the latter to override chunks with parasitic data at almost freely chooseable places, as long as the error margin is not overstressed and no other vital information is harmed (e.g. ECC-header, synchronization). This includes additional synchronization patters and headers if necessary. The parasitic data can be of a different language or the same language as the carrier data. The former will exploit multi-language synchronizers, dissectors and parsers which have to decide on the type of data. The latter will primarily exploit the inability of simple synchronizers to distinguish between different data streams.

This remotely resembles Packet-in-Packet attacks on radio devices and protocol decoding mismatch in network protocols [8]. In contrast to Packet-in-Packet attacks, we do not have to modify the content of the user data. The latter is transparently restored by the ECC mechanism.

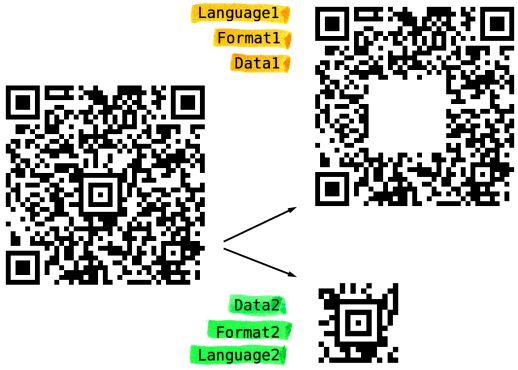

Figure 4. Similar to Figure 3 a Barcode-in-Barcode can be constructed. Note: both parts are valid barcodes.

IV. Implementation using Barcodes

While the visualization in Figure 3 might look far-fetched or abstract, we can do *exactly that* with barcodes (Figure 4).

Linear barcodes or *1D codes* are used to provide a machine-readable form of printed information. In cases where higher data density is required, *matrix* or *2D barcodes* are preferentially deployed. Such codes are used in industrial applications, e.g. logistics or tracking of individual components during the production process.

There are various *2D* or *matrix barcode* symbologies. Each of them tends to be dominant in one or more particular fields of application. This makes it necessary for many devices to support more than one standard. An optical transmission media (printed barcode scanned with a camera under imperfect conditions) is subject to noise, distortions, camera artifacts, uneven illumination, and other effects. Thus, these symbologies typically employ multiple strategies to cope with them, one of which is the extensive use of *error-correcting codes*.

In everyday life, electronic tickets are issued with 2D barcodes, and web links are conveyed via 2D barcodes on billboards and in printed ads. Additionally, they are used in security-sensitive applications such as monetary transactions: Paypal and Bitcoin allow shoppers to pay for goods and services using apps that generate QR codes readable by merchants' existing scanning devices [9]. Threema [10] uses QR codes to exchange keys and authenticate users.

The application of such codes is not without security risks: Different ways of using QR codes as an attack vector have been proposed [11]–[13]. In 2012, hackers showed that *Unstructured Supplementary Service Data* (USSD) codes encoded in 2D barcodes can be used to wipe a phone or execute other system functions [14]. On some phones, they can be used to generate premium rate SMS messages. QR codes can also be used to trigger vulnerabilities in the reader software, the operating system, or a remote site such as SQL injections [11]. The *Ninjhax* exploit [15] uses a custom QR code to perform a buffer overflow on the (locked down) Nintendo 3DS portable game console allowing it to install custom software. Peng et al. [16] found code injection vulnerabilities in several QR libraries. QR Codes are also used to spread malware [13] and for phishing attacks.

Figure 5. Popular 2D Barcodes with rectengular pixels: Quick Response, Aztec, Data Matrix

For our proof-of-concept implementation we used three popular code types with rectangular pixels, thus ensuring a uniform visual appearance when used together: Quick Response (QR) [17], Aztec [18], and Data Matrix (DM) [19]. All codes employ ECC codes, but for practical reasons (Section IV-C) QR codes are suited best for hosting alien data. With the exception of Aztec, all standards mandate a white space (quiet zone) around the symbol, but our tests have shown that most decoders require much less if not none.

A crafted 2D barcode that conforms to multiple standards (or an embedded barcode inside another) is hardly detectable by an untrained human viewer.

A. Full Scan: Multiple Standards Ambiguity

The primary case we are discussing here is that of a full scan. The decoder is presented with a choice of different codes within the same area. Decoding software usually employs multiple computationally cheap finders for specific symbologies, e.g. a detector for a specific visual marker of a symbology (Figure 6). In other contexts, this is called *synchronization pattern*, *preamble*, *magic value*, or *format signature*. When one is found, an appropriate decoder retrieves the data and presents it to the user or the calling application. Although the symbologies are standardized, the dissector decision tree (and its detection order) is not.

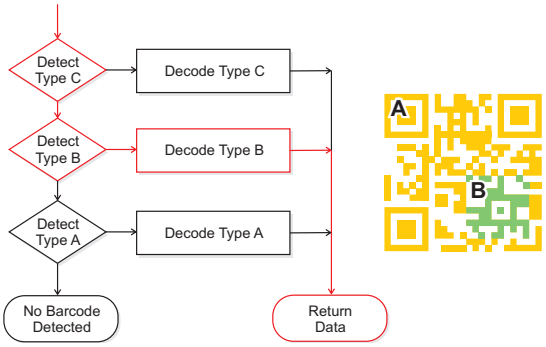

Figure 6. Decoding ambiguity: the detector for a particular code is tested first, therefore the others are not considered.

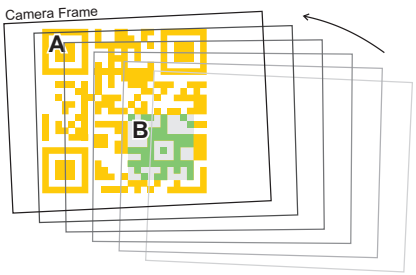

Figure 7. Sliding over the barcode will make the smaller inner barcode fully visible before the entire (outer) barcode.

B. Partial Scan

A partial scan (e.g the user trying to find the right angle and distance for scanning) makes it highly probable that an (*inner*) embedded code is inside the imaging frame before the full *outer* (or *host*) barcode, favoring the detection of the inner barcode.

C. Embedding Criteria

Embedding one code into another requires distinct characteristics of the standards for the outer as well as the inner code.

The outer code has to (a) provide a continuous area of a certain size to shelter the other, and (b) a sufficiently robust data correction (or another way to include alien data).

QR and DataMatrix provide a relatively large continuous area to hide other codes. In our tests, QR's error correction performed much better than Data Matrix's. It is configurable in 4 steps from 7% to 30% (i.e. 30% of the data can be destroyed and still reconstructed). Therefore, currently, the QR symbology provides the best host platform to embed other codes. As versatile as QR's error correction code is, not all parts are protected equally. Some elements are vital and needed before the FEC bits can even be read or applied. Therefore, the embedded code must not interfere with these elements (Figure 8):

1) Finder or Location Markers: These visually prominent markers (including the quiet zone around them) are used by the detector to locate a barcode in an image and correct possible distortions.

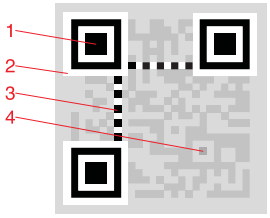

Figure 8. Critical areas of an QR Code: location markers (1), quiet zone (2), timing pattern (3), and alignment markers (4).

Table I
TESTED APPLICATIONS AND THEIR BARCODE STANDARD SUPPORT

OS/Type	Name	QR	DM	Aztec
iPhone	NeoReader [20]	✓	✓	✓
	Qrafter [21]	✓	✓	✓
	i-nigma [22]	✓	✓	–
	QR Code Reader and S. [23]	✓	✓	✓
	ScanLife [24]	✓	✓	–
Android	ZXing Barcode Reader [25]	✓	✓	–
	UberScanner [26]	✓	✓	✓
	ScanLife [27]	✓	✓	–
	i-nigma [28]	✓	✓	–
	AT&T Code Scanner [29]	✓	✓	–
	NeoReader [30]	✓	✓	✓
	ShopSavvy [31]	✓	✓	–
Handheld	Symbol DS6708 [32]	✓	✓	✓

2) Quiet Zone: The QR standard defines a large white space around each barcode. Most readers still require at least 1 pixel white border around the location pattern, whereas some also manage without a quiet zone.

3) Timing Patterns: These dotted patterns run horizontally and vertically between the inner corners of the three location markers. They are used to synchronize rows and column pixels and are essential for most readers.

4) Alignment Markers: They are only built into bigger codes to help handling distortions. They are less important for most decoders and a limited number of them can be destroyed without reducing readability.

V. Experimental Results

We tested 13 readers (iPhone, Android, and dedicated hardware) with ten different barcode inclusions as presented in [7]. For this paper, we picked three notable examples. All codes were scanned ten times and recognition of one code or another was noted, including additional observations (such as strong preference for one outcome). Results can depend on many external factors such as light, state of the reader, or distance. Thus, we refrain from providing non-representative numbers.

A. Aztec in QR

Aztec is a very good choice for being embedded into another code. By standard it does not require a quiet zone. However, our tests have shown that corner placement (and therefore offering a partial quiet zone) provides a higher decodability rate with the Symbol device.

Qrafter was neither able to decode the inner nor the outer barcode, while NeoReader strongly prefers the inner Aztec code. This is probably a case where the Aztec finder is called before the QR finder. All other decoders non-deterministicly return one code or the other.

App/Device	Outer	Inner
NeoReader	✓	✓pref.
Qrafter	✗	✗
i-nigma	✓	–
QR Code R.S.	✓	✗
ScanLife	✓	–
ZXing B.S.	✓	–
UberScanner	✓	✓
ScanLife	✓	–
i-nigma	✓	–
AT&T Code S.	✓	–
NeoReader	✓	✓
ShopSavvy	✓	–
DS6708	✓	✓

Figure 9. Aztec in QR: NeoReader on iOS strongly prefers Aztec over QR. (✓ decoded, ✗ not decoded, – unsupported)

B. Data Matrix in QR

The weakness of Data Matrix is the lack of a distinct visual marker. On the one hand, this makes the code very compact, on the other hand the decoder gets fewer visual clues.

In this example, the embedded Data Matrix code was positioned in the center of the QR code. A thin white border (Figure 10) was added for better decodeability. (In different tests [7], we also shared the border with the outer code.) This example is interesting, because NeoReader on iOS completely ignores the outer QR code. On Android, ScanLife and the AT&T Scanner only decoded the inner Data Matrix when panning over the image.

App/Device	Outer	Inner
NeoReader	✗	✓
Qrafter	✓	✓
i-nigma	✓	✓
QR Code R.S.	✓	✗
ScanLife	✓pref.	✓
ZXing B.S.	✓	✓
UberScanner	✓	✓
ScanLife	✓	(✓swipe)
i-nigma	✓	✓
AT&T Code S.	✓	(✓swipe)
NeoReader	✓	✓
ShopSavvy	✓	✓
DS6708	✓	✓

Figure 10. Data Matrix in QR

C. QR in QR

QR in QR is a special case. The finder markers compete against each other and may confuse the detector. Additionally, it is easier to be noticed by a human. The results suggest, that the white space around the whole barcode as defined in [17] is not a necessity for any of the tested readers and a thin border around the finder markers is enough (Figure 11). Without the latter, most applications only decode the outer barcode.

App/Device	Outer	Inner
NeoReader	✗	✓
Qrafter	✗	✓
i-nigma	✓	✓
QR Code R.S.	✗	(✓)
ScanLife	✗	✓
ZXing B.S.	✗	✓
UberScanner	✗	✓
ScanLife	✗	✓
i-nigma	✓	✓
AT&T Code S.	✗	✓
NeoReader	✗	✓
ShopSavvy	✗	(✓)
DS6708	✓pref.	✓

Figure 11. QR in QR, center with white space

Qrafter and ShopSavvy need noticeably longer for decoding, but do so only for the embedded code. i-nigma on Android prefers the outer code when the phone is held further away, and the inner code when it is held closer to the barcode. The QR Code Reader and Scanner on iOS has major troubles with decoding. In our tests it eventually returned the inner code and in one case returned a garbage string.

Presumably, these implementations prefer markers in close vicinity to each other, except for the handheld device by Symbol. The latter prefers the largest area between markers.

VI. Discussion, Countermeasures and Future Work

We demonstrated the ECC hiding and decode ambiguity problem with popular barcodes. We argue that it can be used to deliver exploits based on the software installed, or use probabilistic effects to divert a small percentage of users to fraudulent sites (e.g. a donation site where some transactions are diverted to a different account). It could also be used for fare-dodging or circle-routing parcels, as different stations along the logistics chain read different tracking IDs.

More in general, the technique can be used anywhere where ECC is employed. This includes satellite or terrestrial digital video transmission (e.g. DVB-S and DVB-T) where different content is decoded by different viewers. This might also have implications on computer (anti-)forensics when dealing with ECC-protected data (tapes, hard disc arrays, ECC RAM).

The main conceptional problem with countermeasures is that ECC are designed to transparently hide any modifications from the processing layers above. The host data as well as the parasitic data are actually valid and conform to the

specifications. Therefore, effective mitigation often heavily depends on the threat model, the application, and whether this case can be escalated to the layers above – potentially reaching a user interactively.

A. Countermeasures for Barcodes

For the barcode example, several mitigation strategies arise. It should be noted that user involvement is an easy option for interactive applications, but less suitable for automated processes (e.g. sorting machines in logistics).

1) Stringent priority: While the code formats themselves have been standardized, the order of detection is chosen by the software designers. As this is the root cause for multi-standard code ambiguity, a stringent prioritization should be defined.

2) Notification on all codes found: Barcode readers detecting the presence of code ambiguity (same standard or multiple) should present all of them to the user. This requires that software does not stop after the detection of the first code.

3) Alien data warning: A reader that detects alien data, multiple standards, or multiple decodings using the same standard should warn the user. This might include standards that it might not be able to decode but is able to detect based on its marker signature. However, this also bears the risk of false positives, as the decoder cannot verify if the marker actually belongs to a valid code.

4) Scanned photo excerpt: Interactive barcode readers can present the decoded image and highlight the area of interest containing the decoded barcode for visual inspection.

5) Only decode what you are looking for: Limit decoding to only the standards the intended purpose needs.

B. Generic Countermeasures

Generic solutions are hard to come by, as both data transmissions are perfectly valid considering the specification. The threat model determinates whether the decoder should escalate the condition to a higher layer, fail-safe by discarding all data, or try to decide on the benign data by itself.

The latter is not a trivial problem. In our visual QR examples above, dissecting parasitic data from the host transmission is much easier than the general case. Spotting multiple synchronization markers on an asynchronous data channel and verifying their belonging to non-overlapping data packets is potentially a hard problem as it might include decoding and verification of all the possible data interpretations itself.

Another (simpler) heuristic could involve sorting possible decoding variants of the data by its amount of bits (whether before or after ECC) and assuming that the longest data stream is the host and therefore legit.

However, any heuristic (or *guessing*) is a risk, and might lead to exactly the problems described in Section I. These are open problems left for future work.

VII. Conclusion

Data decoding, format, or language ambiguities have been long known for amusement purposes. Only recently it came to attention that they also pose a security risk, for example by deceiving file dissectors of firewalls and virus scanners. These binary polyglots mostly arise from poor parsers, lax data format specifications, and undefined multi-standard compatibility. In contrast to carefully crafted polyglots, data formats protected by error-correcting codes (ECC) provide a very convenient way of constructing decoder or parser ambiguities.

This can be used to encode data in multiple formats or even the same format with different content. Implementation details of the decoder or environmental effects decide which data the decoder locks onto. This leads to different users receiving different content based on a language decoding ambiguity.

We demonstrated this behavior with popular 2D barcodes but the method is not limited to these. However, evasion or mitigation strategies are not easy transferable and mostly application specific.

The main conceptional problem with countermeasures is that ECCs are designed to transparently hide any modifications from the processing layers above. The host data as well as the parasitic data are actually valid and conform to the specifications. Where applicable, stringent language descriptions that include dissection rules for multiple languages/standards and multiple synchronization markers are a good start. In the general case, spotting multiple synchronization markers on an asynchronous data channel and verifying their belonging to non-overlapping data packets is not a trivial task. The area of defenses leaves plenty of opportunities for future work.

Acknowledgments

Part of this work arose during an internship at the National Institute of Informatics, Tokyo. Special thanks to Johanna Ullrich, Katharina Krombholz and Manuel Leithner for their valuable feedback and their previous work.

This research was partially funded by the COMET K1 program by FFG (Austrian Research Funding Agency) and the Austrian Science Fund (FWF): P 26289-N23.

References

[1] A. Albertini, "corkami: Reverse engineering and visual documentations," http://code.google.com/p/corkami/#Binary_files, accessed September 6th 2014.

[2] J. Magazinius, B. K. Rios, and A. Sabelfeld, "Polyglots: crossing origins by crossing formats," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 753–764.

[3] A. Albertini, "This PDF is a JPEG; or, This Proof of Concept is a Picture of Cats," in *PoC || GTFO 0x03*, March 2014, http://corkami.googlecode.com/svn/trunk/doc/pocorgtfo/pocorgtfo03.pdf.

[4] S. Jana and V. Shmatikov, "Abusing File Processing in Malware Detectors for Fun and Profit," in *Proceedings of the 33rd IEEE Symposium on Security & Privacy*, San Francisco, CA, May 2012.

[5] S. Alvarez and T. Zoller, "The death of AV defense in depth? - revisiting anti-virus software," 2008, http://cansecwest.com/csw08/csw08-alvarez.pdf.

[6] L. Sassaman, M. L. Patterson, S. Bratus, M. E. Locasto, and A. Shubina, "Security Applications of Formal Language Theory," in *IEEE Systems Journal, Volume 7, Issue 3*, Sept. 2013.

[7] A. Dabrowski, K. Krombholz, J. Ullrich, and E. R. Weippl, "QR Inception: Barcode-in-Barcode Attacks," in *Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*. ACM, 2014, pp. 3–10.

[8] T. Goodspeed, S. Bratus, R. Melgares, R. Shapiro, and R. Speers, "Packets in packets: Orson welles' in-band signaling attacks for modern radios." in *Proceedings to WOOT 2011*, August 2011, pp. 54–61.

[9] D. Tam, "PayPal offers QR codes for retail-store purchases," October 2013, http://www.cnet.com/news/paypal-offers-qr-codes-for-retail-store-purchases/, accessed July 24th 2014.

[10] Threema GmbH, "Threema," https://threema.ch/, accessed July 17th 2014.

[11] P. Kieseberg, S. Schrittwieser, M. Leithner, M. Mulazzani, E. Weippl, L. Munroe, and M. Sinha, "Malicious Pixels Using QR Codes as Attack Vector," in *Trustworthy Ubiquitous Computing*, ser. Atlantis Ambient and Pervasive Intelligence, I. Khalil and T. Mantoro, Eds. Atlantis Press, 2012, vol. 6, pp. 21–38. [Online]. Available: http://dx.doi.org/10.2991/978-94-91216-71-8_2

[12] A. Kharraz, E. Kirda, W. Robertson, D. Balzarotti, and A. Francillon, "Optical Delusions: A Study of Malicious QR Codes in the Wild," in *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 06 2014.

[13] M. DeCarlo, "AVG: QR code-based malware attacks to rise in 2012," 2012, http://www.techspot.com/news/47189-avg-qr-code.html, accessed July 18th 2014.

[14] B. Naik, "QR Code: USSD attack," 2012, http://resources.infosecinstitute.com/qr-code-ussd-attack/, accessed July 18th 2014.

[15] J. Rabet, "NINJHAX - 3DS Homebrew Exploit," http://smealum.net/ninjhax/, accessed March 3rd 2015.

[16] K. Peng, H. Sanabria, D. Wu, and C. Zhu, "Security Overview of QR Codes," 2014, MIT Student Paper, available online https://courses.csail.mit.edu/6.857/2014/files/12-peng-sanabria-wu-zhu-qr-codes.pdf.

[17] *ISO/IEC 18004: Information technology – Automatic identification and data capture techniques – QR Code 2005 bar code symbology specification*, International Organization for Standardization Std. ISO/IEC 18 004.

[18] *ISO/IEC 24778: Information technology – Automatic identification and data capture techniques – Aztec Code bar code symbology specification*, International Organization for Standardization Std. ISO/IEC 24 778.

[19] *ISO/IEC 16022: Information technology – Automatic identification and data capture techniques – Data Matrix bar code symbology specification*, International Organization for Standardization Std. ISO/IEC 16 022.

[20] NeoMedia Technologies, Inc., "NeoReader," Apple App Store, https://itunes.apple.com/us/app/id284973754, accessed July 17th 2014.

[21] Kerem Erkan, "Qrafter," Apple App Store, https://itunes.apple.com/us/app/id416098700, accessed July 17th 2014.

[22] 3GVision, "i-nigma," Apple App Store, https://itunes.apple.com/en/app/id388923203, accessed July 17th 2014.

[23] ShopSavvy Inc., "QR Code Reader and Scanner," Apple App Store, https://itunes.apple.com/en/app/qr-code-reader-and-scanner/id388175979, accessed July 17th 2014.

[24] Scanbuy Inc., "ScanLife Barcode & QR Code Reader with Prices, Deals, & Reviews," Apple App Store, https://itunes.apple.com/us/app/scanlife-barcode-reader-qr/id285324287, accessed July 17th 2014.

[25] ZXing Team, "Barcode Scanner," Google Play Store, https://play.google.com/store/apps/details?id=com.google.zxing.client.android, accessed July 17th 2014.

[26] Ubercoders, "UberScanner," Google Play Store, https://play.google.com/store/apps/details?id=org.ubercoders.uberscanner, accessed July 17th 2014.

[27] Scanbuy Inc., "ScanLife QR & Barcode Reader," Google Play Store, https://play.google.com/store/apps/details?id=com.ScanLife, accessed July 17th 2014.

[28] 3GVision, "i-nigma Barcode Scanner," Google Play Store, https://play.google.com/store/apps/details?id=com.threegvision.products.inigma.Android, accessed July 17th 2014.

[29] AT&T Services Inc., "AT&T Code Scanner: QR,UPC & DM," Google Play Store, https://play.google.com/store/apps/details?id=com.mtag.att.codescanner, accessed July 17th 2014.

[30] NeoMedia Technologies Inc., "NeoReader QR & Barcode Scanner," Google Play Store, https://play.google.com/store/apps/details?id=de.gavitec.android, accessed July 17th 2014.

[31] ShopSavvy Inc., "ShopSavvy Barcode Scanner," Google Play Store, https://play.google.com/store/apps/details?id=com.biggu.shopsavvy, accessed July 17th 2014.

[32] M. Inc., "Symbol DS6708 Digital Scanner Product Reference Guide," 2009, http://www.motorolasolutions.com/web/Business/Products/Bar%20Code%20Scanning/Bar%20Code%20Scanners/General%20Purpose%20Scanners/_Documents/static_file/ds6708.pdf.