1 Contents

2 Introduction

This project is about the security issues surrounding how supermarkets in the UK have been using Auto-ID technologies to automate Point-Of-Sale (POS) interactions with customers. Most people are already quite familiar with the experience, so little explanation is required: All purchasable items in the store are affixed with barcodes that uniquely identify them, and when a customer takes an item to checkout the POS will read the barcode and lookup the information associated with that item. Most importantly in stores with self-checkouts, the information accessed will include the price that the item is being sold for, and the expected weight of the product.

Typically, this information is all stored on a centralised server which all POS systems will have access to. Most items have barcodes that are either 8 or 13 digits, which serve as a unique identifier and encode no other information. This is a somewhat adequate solution from a digital security standpoint, as long as the POS can be sure that the barcode that is scanned corresponds to the correct product. This is trivial at a manual checkout till, but at self-checkout this is done by measuring the mass of items placed down, and comparing it to a reference measurement of that item. The resolution of this measurement is high enough that a customer cannot pass off one item as another.

Assuming an attacker can't just bypass the physical security procedures in place to avoid shoplifting, if they want to decrease the price they pay at a POS then they have two choices: they must either modify the price reported by the centralised server for that item, or trick the POS into thinking the item they're scanning is a different, lower value item. The latter is often referred to as the "Banana Trick" [5], as shoplifters sometimes tell the self-checkout terminal that they are about to weigh bananas, but instead place an item of greater value on the weighing scale. This exploits the fact that the self-checkout can only differentiate between items based on their mass.

Issues begin to arise when information such as the price of the item is also encoded in the barcode, and the customer is still in control of the barcode that gets scanned for a given item. This makes it such that the price reported by the centralised server is bypassed completely, in favour of the new price.

All major supermarkets in the UK have a system in place to apply price reductions to their stock, so that they can incentivise the purchase of consumables approaching their expiration date, damaged items or stock otherwise designated for clearance. This is often done by generating a barcode that encodes a new price for the item in addition to the item's original identifier, and these "discount" barcodes essentially override the original price of the item.

This project can be split into two sections: In the first I present how an attacker can exploit these discount barcodes to manipulate the pricing for items to their will. To achieve this, I reverse engineer the methods used by all the major grocery stores of the UK, and develop a mobile application that can produce "counterfeit" discounts for all these stores. The targets will be

The second section of my project will be the development of a cryptographically secure alternative to the discounting procedures currently in place. I will analyse the attacks that current systems are vulnerable to, and propose a robust solution that is resilient to these attacks. The proposed solution must also take into consideration the constraints of operating in these existing environments and be viable to implement in place. Ideally, it will be as close to a plug-and-play solution as I can achieve, while bringing a large security improvement over the existing system.

3 Section A – Developing the Exploit

I first encountered this vulnerability when I came across a discount while shopping at Sainsbury's. I had no background knowledge of how discounts were produced in any store, and until after I finished the working prototype for Sainsbury's I hadn't found past work in the field.

3.1 Background Information about Barcodes

When items at Sainsbury's are discounted, they get new barcodes placed over the original barcode, the intention being to avoid accidentally scan the old barcode. I bought a discounted item (A loaf of Hovis bread) and brought it home to analyse how the barcode works. Below is the original barcode of the item, that was covered up by the discount code.

Figure 1 - The Original EAN-13 Barcode

The original barcode for the item is a 13-digit EAN-13 Barcode. All items in the store have either the 8-digit EAN-8 barcode, or the EAN-13 barcode instead. The European Article Number (EAN), also known as the International Article Number in non-European Countries, is a barcode symbology and numbering system that is standardised by GS1 [6].

EAN-13 Barcodes begin with a 2 or 3 digit GS1 Prefix, as per the GS1 General Specification [6]. In particular, 500-509 are the company prefixes for GS1 United Kingdom [7]. The following 10 or 9 digits (Depending on the prefix length) are the combination of a manufacturer code and a product code. Up until recently, EAN-13 barcodes were known as UCC-13 barcodes and the manufacturer code was fixed to 5 digits, but thousands of potential products codes were being wasted on manufacturers with only a few products, so the manufacturer code was made to be variable length.

Finally comes the check digit for the barcode, in this case "4". The check digit is a form of redundancy check and is calculated from the other digits in the barcode. This is useful for barcode scanners to ensure that they correctly read a barcode, as a single digit change in the data would be guaranteed to be detected. The checksum for GS1 barcodes is calculated as the weighted sum of the digits in the barcode, i.e.

$$\sum_{i=1}^{n} x_i * \begin{cases} 3 \; where \; i \; is \; even \\ 1 \; where \; i \; is \; odd \end{cases}$$

Where n is the length of the barcode, and x_i is the digit of the barcode at the position i. Then the check digit is calculated by subtracting the sum from nearest equal or higher multiple of ten. As you'll see in the next chapter, I wasn't aware that this is also how check digits for the Sainsbury's discount codes is generated as well until after I reverse engineered it, so my algorithm will look different to this, despite achieving the same results.

Code-128 Barcodes, which are used extensively by most the supermarkets that implement discount barcodes, are variable length barcodes that don't have to have a check digit appended to the end like EAN-8 and EAN-13 barcodes. This is because Code-128 barcodes have a mandatory check-symbol section that is part of the barcode, and is processed by barcode scanners and generators without the end user ever having to interact with them directly. The integrated check-symbol is a modulo 103 weighted sum, where the weight for each digit is the position (or index) [6]. Despite this, most supermarkets with Code-128 discount barcodes include another check digit at the end of their barcodes.

3.2 Prototyping for Sainsbury's

Figure 2 - Code 128 Discount Barcode

This is the Code 128 discount barcode for the same loaf of bread that was shown in figure 1, applied over the original barcode. Every discount barcode at Sainsbury's starts with 91, which appears to be the Application Identifier, as described in the GS1 General Specification [6]. Application Identifiers 91-99 are reserved for company internal information.

The following 13 digits are the digits of the original barcode, just as they appear on the original packaging. For items that have the shorter EAN-8 barcode, this will be zero padded on the left.

Next is the new price for the item, the one that it has been reduced to. Note that is not how much it is reduced by, but instead the price it has been reduced to.

The final digit is a new check digit for the entire discount barcode. As it so happens, the check digit for the Code 128 discount barcode is calculated the same way as it would be for EAN-13 and EAN-8 barcodes, but as I was unaware at the time I was working with the following model:

$$Check\ Digit = (\sum_{i=1}^{n} w_i x_i)\ mod\ (10)$$

Where each digit x had its own associated weighting w. Operating under this assumption, I would go through the barcode one digit at a time, and would try incrementing or decrementing this value to see how it would effect the check digit.

Sainsbury's has an app called the Smartshop App. The purpose of this app is to scan your items using your phone as you shop, accelerating the checkout process. The app requires location permissions and only can only be used when the phone reports its position as being inside the store, but this restriction can be bypassed on Android devices by spoofing the GPS location and disabling Google Location Accuracy, which improves location accuracy by using Wi-Fi, mobile networks, and device sensors to help estimate location.

After getting Smartshop working outside the store, I'd have plenty of time for testing. Not only does Smartshop supports scanning discounted items, it would also throw an error regarding the invalidity of the barcode if the check digit was wrong. So to determine how each digit was weighted, I would enumerate through all 10 possible values for each digit, and see which of the 10 possible check digits are accepted by Smartshop:

Table 1 - Brute forcing w_{20} of the check digit

Barcode (Excluding check digit)	**Accepted check digit**	**Check digit without Modulo division**
91501000306528400030	3	3
91501000306528400031	0	10
91501000306528400032	7	17
91501000306528400033	4	24
91501000306528400034	1	31
91501000306528400035	8	38
91501000306528400036	5	45
91501000306528400037	2	52
91501000306528400038	9	59
91501000306528400039	6	66

Remembering that the check digit is the result of a modulo 10 division, it becomes clear that the check digit increases by 7 each time the last digit is incremented. In other words:

$$w_{20} = 7$$

Barcode (Excluding check digit)	Accepted check digit	Check digit without Modulo division
91501000306528400009	9	9
91501000306528400019	8	18
91501000306528400029	7	27
91501000306528400039	6	36
91501000306528400049	5	45
91501000306528400059	4	54
91501000306528400069	3	63
91501000306528400079	2	72
91501000306528400089	1	81
91501000306528400099	0	90

Table 2 - Brute forcing w_20 of the check digit

Repeating this process for the second to last digit, we find the following:

$$w_{19} = 9$$

Since it was feasible to continue manually brute force this by generating barcodes and scanning them with Smartshop, I did just that. I found that $w_{20} = 7,\ w_{19} = 9,\ w_{18} = 7,\ w_{17} = 9$ and $w_{16} = 7$. It became clear after these first few digits that the pattern was alternating:

Figure 3 - Annotated Weightings for Sainsbury's

$$\sum_{i=0}^{n-1} x_i * \begin{cases} 7 \ where\ i\ is\ even \\ 9 \ where\ i\ is\ odd \end{cases} mod(10)$$

$$\begin{aligned} Checkdigit = (&9*9+1*7+5*9+0*7 \\ &+1*9+0*7+0*9+0*7 \\ &+3*9+0*7+6*9+5*7 \\ &+2*9+8*7+4*9+0*7 \\ &+0*9+0*7+0*9+3*7 \\ &+3*9)\ mod\ (10) \\ =\ &416\ mod\ 10 \\ =\ &6 \end{aligned}$$

3.3 Previous Work

Despite the vulnerability still existing in 2022, this issue was brought to light all the way back in 2012 in both Tesco's and Asda's inventory system, but has had no mention on the clearnet since.

3.3.1 Tesco

On Sunday 22nd July 2012, A researcher (who was at the time a Physics student at the University of Manchester) published on his WordPress blog the article *Tesco Discount Barcodes, Cracked* [8]. In it, he showed how the discount barcodes for Tesco could be reverse engineered, just as I'd done with Sainsbury's. Take the following barcode, Matt Evans showed that it can be broken down into:

971 5031021057952 4 00019 0 0

Figure 4 - Tesco discount code for a bottle of milk

971 – means "This is a discounted item"
5031021057952 – The products regular barcode
4 – Matt Evans called this the mystery digit, he couldn't figure out what it meant
00019 – The new price, in pence
0 – A zero (he concludes this is a padding character)
0 – A check digit

His conclusions on the breakdown of the Tesco discount code are very similar to the conclusions I arrived at with Sainsbury's discounting algorithm, since the way Tesco generated discounts back in 2012 was so similar to how it's still done today.

Unfortunately, he was unable to determine the mystery digit at the time, but the user MartinMcGirk on made a relevant comment on Hacker News a couple days later [9], which mentions that the red mystery digit is known internally as the discount-reason-code and represents the reason for the discount, such as "damaged" or "short date (nearly out of date). Further on in the paper I show that any digit works for the reason code.

On the 23rd July 2012, Matt Evans posted on his twitter that he chatted with a Tesco Insider who said that discounts are stored in a local database, which gets wiped at 3am every morning [10]. This would suggest that it wouldn't be possible to generate your own discount barcodes or scan previous ones again, but as I found during testing in a few chapters, this isn't true, and the vulnerability does in fact exist.

Then on 24th July 2012, the Anonymous Twitter account @YourAnonNews published a tweet with a bit.ly link to Matt Evan's blog page about the barcode crack [11]. The next day, 25th July 2012, Matt Evans took down his blog post, at request of Tesco's security department [12]. Tesco never fixed the issue, and the system is still exactly the same almost 10 years later.

3.3.2 Asda

Inspired by the controversy of Matt Evans' Tesco Barcode Crack, the admin of greenhac.org.uk, known only as Peter, pursued a similar project but with Asda's discount codes [13]. Take for example the following discount code from Asda for pork pies, he showed it is broken down like so:

020 518 00 0010
020 – A prefix all reduction codes start with
518 – A "department" or "product type", 518 corresponds to "Deli Pies"
00 – He was uncertain if this was more digits for the price or not
0010 – Reduced price in pence, in this case 10p

As I show in my own research, his conclusions were not accurate, and are also no longer relevant.

3.4 Implementations

I travelled to all the following stores, as well as carried out online research, in order to reverse engineer the algorithms used to produce discounts all supermarkets.

Most supermarkets include a check digit at the end of their discount codes, all of which use alternating weightings for the digits in even and odd positions, so I created a brute forcing program using Python that will find those weights. The program will try all 100 possible pairs of weights for a given barcode, and since there are only 10 possible check digits for a given barcode, there is a 1/10 chance that a random pair of weightings will produce the correct check digit.

To fix this, the program can take any number of barcodes as input, and will only return the weightings that in the correct check digit for all the given barcodes. If you provide 2 barcodes, there will be a 1/100 chance that a random pair of weightings will produce the correct check digit for both, 1/1000 for 3 barcodes, etc.

3.4.1 Sainsbury's

Discounts for Sainsbury's are encoded in Code 128 barcodes. The resultant discount is always 22 digits long, no matter the parameters for price and barcode. As already shown in the prototype, the contents are formed like so:

91 BBBBBBBBBBBB PPPPPP C

1. All Sainsbury's discount barcodes start with 91

2. Afterwards follows the original barcode of the item $BBBBBBBBBBBB$. This is always 13 characters, and if the original item being discounted has an EAN-8 barcode, then it is zero padded on the left with 5 zeros, to increase the length to 13. The last character B_{13} should be the original check digit.

3. Then is the new price $PPPPPP$ encoded as a 6-digit number in pence. This is always zero padded, and allows for prices up to £9999.99.

4. The final check digit C is calculated like so:

$$C = \sum_{i=0}^{n-1} x_i * \begin{cases} 7 \text{ where } i \text{ is even} \\ 9 \text{ where } i \text{ is odd} \end{cases} mod(10)$$

3.4.2 Tesco

Discounts for Tesco's are encoded in Code 128 barcodes. Building on Matt Evans' discoveries, I also determined out what the second to last digit stands for, as it's not just always 0. Whenever an item goes unsold after being reduced, it would get reduced further, and the new discount code would increment the digit in this position. On the first discount this value is typically 0, which is why Matt Evans believed this digit is always 0, but if a product is discounted further, this digit gets incremented. The resultant discount code is always 24 digits long, no matter the parameters for price and barcode:

971 BBBBBBBBBBBB R PPPPP I C

1. All Tesco discount barcodes start with 971.

2. Afterwards follows the original barcode of the item $BBBBBBBBBBBB$. This is always 13 characters, and if the original item being discounted has an EAN-8 barcode, then it is zero padded on the left with 5 zeros, to increase the length to 13. The last character B_{13} should be the original check digit.

3. The next digit is the discount reason R. This can take any value between 0 and 9. If there exists a discount-reason-code that corresponds to part of the item missing or being damaged, this could be of interest as it would be a cue for the self-checkout to ignore the typical weight restrictions for the object.

4. Following that is the new price $PPPPP$, encoded as a 5-digit number in pence. This is always zero padded, and allows for prices up to £999.99.

5. The next digit I is the discount iteration. Starting at 0, each time the item goes through another iteration of discounts, the digit I gets incremented.

6. Finally is the check digit C, calculated the same as with Sainsbury's:

$$C = \sum_{i=0}^{n-1} x_i * \begin{cases} 7 \text{ where } i \text{ is even} \\ 9 \text{ where } i \text{ is odd} \end{cases} mod(10)$$

3.4.3 Waitrose

Discounts for Waitrose are encoded in Code 128 barcodes. Waitrose discount barcodes don't have a check digit at the end, and the resultant discount is always 20 digits long, no matter the parameters for price and barcode:

10 BBBBBBBBBBBB PPPPP

1. All discount barcodes start with 10.

2. Afterwards follows the original barcode of the item $BBBBBBBBBBBB$. This is always 13 characters, and if the original item being discounted has an EAN-8 barcode, then it is zero padded on the left with 5 zeros, to increase the length to 13. The last character B_{13} should be the original check digit.

3. Then is the new price $PPPPP$ encoded as a 5-digit number in pence. This is always zero padded, and allows for prices up to £999.99.

3.4.4 Morrisons

Discounts for Morrisons are encoded in Code 128 barcodes, and are constructed in a very similar fashion to how Tesco's discount barcodes are produced. The resultant discount code is always 24 digits long, no matter the parameters for price and barcode:

92 BBBBBBBBBBBB PPPPP M 000 C

1. All Morrisons discount barcodes start with 92

2. Afterwards follows the original barcode of the item $BBBBBBBBBBBB$. This is always 13 characters, and if the original item being discounted has an EAN-8 barcode, then it is zero padded on the left with 5 zeros, to increase the length to 13. The last character B_{13} should be the original check digit.

3. Following that is the new price $PPPPP$, encoded as a 5-digit number in pence. This is always zero padded, and allows for prices up to £999.99.

4. The next digit is a mystery digit M. This could take any value between 0 and 9, but I was unable to determine what the purpose of this digit is, or if it's a discount reason like Tesco's discount reason code.

5. After the mystery digit is always 000. In all samples I was able to acquire, this appears to be true for all discount barcodes.

6. Finally is the check digit C, calculated with the opposite weightings to Sainsbury's:

$$C = \sum_{i=0}^{n-1} x_i * \begin{cases} 9 \text{ where } i \text{ is even} \\ 7 \text{ where } i \text{ is odd} \end{cases} mod(10)$$

3.4.5 Marks and Spencer

Discounts for M&S are encoded in Code 128 barcodes, and the resultant discount code is always 24 digits long, no matter the parameters for price and barcode.

82 I BBBBBBBBBBBB OOOOO NNNNN

1. All M&S discount barcodes start with 82

2. The next digit I is the discount iteration. Each time the item is reduced, goes unsold, and it gets reduced further, the digit I gets incremented.

3. Afterwards follows the original barcode of the item $BBBBBBBB$. This is always 8 characters, doesn't support EAN-13 barcodes at all. This is due to M&S using their own subsets of EAN-8 barcodes called MS7A, MS7B or MS8A. MS7A and MS7B only show 7 data digits under the barcode, but the data includes a leading zero that must be kept.

4. Following that is the old price $OOOOO$, encoded as a 5-digit number in pence. This is the MSRP the item would sell for had it not been discounted. If the item is being discounted multiple times, this is NOT the price that was on the previous discount, this is the original price. This is always zero padded, and allows for prices up to £999.99.

5. Finally is the new price $NNNNN$, encoded as a 5-digit number in pence. This is always zero padded, and allows for prices up to £999.99.

3.4.6 Co-op Food

Discounts for Co-op are always encoded in Code 128 barcodes, but there are two variants of discounts, which depends on the specific store that is visited. Some Co-operatives, such as the Southern Co-op I visited when testing, have discount codes that are 20 digits long, whereas there are other Co-operatives that have 25-digit discount codes.

3.4.6.1 Co-operative Long Discounts

BBBBBBBBBBBB M 01 PPPP DDDD C

The Co-operatives with 25-digit discount codes include a mystery digit I was unable to derive, and then 4 extra digits that encoded extra information about the discount. By applying some basic OSINT techniques, I gathered various images of discounts from Co-op across a large timespan and discovered three important facts about the longer 25-digit discount codes:

1. Discounts produced on the same day had the same 4 unknown digits.
2. Older discounts had a smaller value in this 4-digit field than newer discounts.
3. Where there were two discounts produced n days apart, the 4-digit value in the newer discount was greater than the value in the old discount.

From these facts I was able to deduce not only that this field encoded information about the date which the discount was produced, but also that this was counting the number of days from an epoch. The epoch date in question was January 1st, 2000, i.e. Y2K.

1. All Co-op discounts begin the original barcode of the item $BBBBBBBBBBBB$. This is always 13 characters, and if the original item being discounted has an EAN-8 barcode, then it is zero padded on the left with 5 zeros, to increase the length to 13. The last character B_{13} should be the original check digit
2. The next digit is a mystery digit M, which either takes the value 0 or 1. I was unable to determine what the purpose of this digit is, or if it's a discount reason like Tesco's discount reason code.
3. After the mystery digit is always 01. In all samples I was able to acquire, this appears to be true for all discount barcodes.
4. Following that is the new price $PPPP$, encoded as a 4-digit number in pence. This is always zero padded, and only allows for prices up to £99.99.
5. Then comes the epoch counter $DDDD$, which is calculated as the number of calendar days between the day of Y2K and the current date the discount is being generated.
6. Finally is the check digit C, calculated with the opposite weightings to Sainsbury's and the short discount:

$$C = \sum_{i=0}^{n-1} x_i * \begin{cases} 9 \; where \; i \; is \; even \\ 7 \; where \; i \; is \; odd \end{cases} \; mod(10)$$

3.4.6.2 Co-operative Short Discounts

The Co-op short discount is a 20-digit Code 128 barcode, and is calculated like so:

BBBBBBBBBBBB 01 PPPP C

1. All Co-op discounts begin the original barcode of the item $BBBBBBBBBBBB$. This is always 13 characters, and if the original item being discounted has an EAN-8 barcode, then it is zero padded on the left with 5 zeros, to increase the length to 13. The last character B_{13} should be the original check digit

2. After the original barcode is always 01.

3. Following that is the new price $PPPP$, encoded as a 4-digit number in pence. This is always zero padded, and only allows for prices up to £99.99.

4. Finally is the check digit C, calculated with the same weightings as Sainsbury's, the opposite to the weightings used in the long discount:

$$C = \sum_{i=0}^{n-1} x_i * \begin{cases} 7 \; where \; i \; is \; even \\ 9 \; where \; i \; is \; odd \end{cases} \; mod(10)$$

3.4.7 Iceland

For Iceland stores that print discount barcodes, the discount takes the form of an EAN-13 barcode:

BBBBBBBB PPPP C

1. The discount code begins original barcode of the item $BBBBBBBB$. This is always 8 characters, and doesn't support EAN-13 barcodes at all. Shorter barcodes are padded with leading zeros to increase the length to 8.

2. Then is the new price $PPPP$ encoded as a 4-digit number in pence. This is always zero padded, and only allows for prices up to £99.99.

7. Finally is the check digit C, calculated with the opposite weightings to Sainsbury's:

$$C = \sum_{i=0}^{n-1} x_i * \begin{cases} 9 \; where \; i \; is \; even \\ 7 \; where \; i \; is \; odd \end{cases} \; mod(10)$$

3.5 Lidl

Lidl doesn't support discount barcodes that allow for arbitrary pricing, but there are two other types of discounts: Fixed discounts as a part of the Waste Not scheme, and a relative 30% discount.

3.5.1.1 Waste Not Discounts

As part of the Waste Not scheme, Lidl provides these fixed price discounts. These discounts for Lidl are not dynamically calculated based on the original item, and all reductions to the same price will have the same barcode, as data of the original item doesn't get encoded in the discount.

New Price	Barcode
20p	0035255
70p	0041461
90p	0035378
£1.50	0035248

Table 3 - Lidl Waste-Not Discounts

3.5.1.2 Lidl 30% Off Discount

The Lidl 30% discount barcode is a very simple 15-digit Code-128 barcode:

`55 BBBBBBBBBBBB`

1. The discount always starts with `55`, which indicates to the POS that this is a 30% discount
2. Following that is the original barcode of the item $BBBBBBBBBBBB$. This is always 13 characters, and if the original item being discounted has an EAN-8 barcode, then it is zero padded on the left with 5 zeros, to increase the length to 13. The last character B_{13} should be the original check digit, not a new check digit.

3.5.2 Can't Implement - Aldi

Aldi doesn't have any discount barcodes at all, only red stickers that have "30% Off" or "50% Off% written on them. Furthermore, there are no Aldi's with self-checkouts.

3.5.3 Can't Implement - Asda

Fortunately, Asda has transitioned their discount system from the one they had back in 2012, to a more secure alternative that doesn't allow the barcode to dictate arbitrary prices for items. Up until at least July 2017 [14], Asda was using the old 205 series discounts that were described by Peter of Greenhac back in 2012 [13]. However, images of the newer, more secure 499-Series discount barcodes can be found on the internet starting from February 2018 [15], so we can assume the new system was being rolled out between these two dates.

3.5.3.1 205-Series Discount Barcodes

205 XX PPPPPP

1. All 205-series discounts began with 205
2. Then came the department number XX. For example, all discounted bread would have the value 25 for these two digits, vegetables would be 94, etc.
3. Finally was the new price in pence $PPPPPP$, zero padded on the left to be 6 digits long

3.5.3.2 499-Series Discount Barcodes

499 MM DDDDDD C

1. All 499-series discounts begin with 499
2. Then there's 2 unknown digits MM. They may serve as a department number just as they did in 205-series discount barcodes, but sometimes they are left as 00.
3. Then is a discount counter $DDDDDD$. Each batch of discounts produced has a different number for these digits, although discounts produced on the same day have nearby numbers for this field.
4. Finally is a check digit C, calculated with the same weights as Sainsbury's

$$C = \sum_{i=0}^{n-1} x_i * \begin{cases} 9 \; where \; i \; is \; even \\ 7 \; where \; i \; is \; odd \end{cases} \; mod(10)$$

When a 499-series discount is purchased, the self-checkout knows exactly what the item is and how much is should weigh, as well as the new discount price. This suggests there exists a database indexed on $DDDDDD$ that provides it with the necessary information, and an attacker cannot generate arbitrary discounts unless they can insert their own discounts into the database.

3.6 Building an Application

To demonstrate the vulnerability, I created an Android application that implements all the reverse engineered algorithms shown up to this point. Development was done with Java in Android Studio, with the help of the ZXing library [4] for scanning and generating barcodes. The application has many features that make it very easy to use.

3.6.1 Clean Interface

The interface only shows the inputs relevant to the target store, includes accessibility considerations such as Text-To-Speech metadata, high-contrast text across light and dark themes, feedback for interactions, and intuitive self-documenting controls. Input fields irrelevant to the target store aren't shown to the user

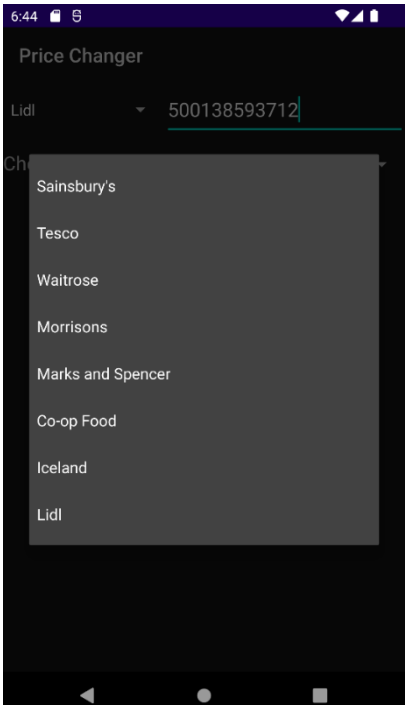

Figure 2 – App Store Selection

Support for several supermarkets

By market share, the app allows the user to set arbitrary prices in 65% of UK supermarkets, as well as relative discounts in Lidl.

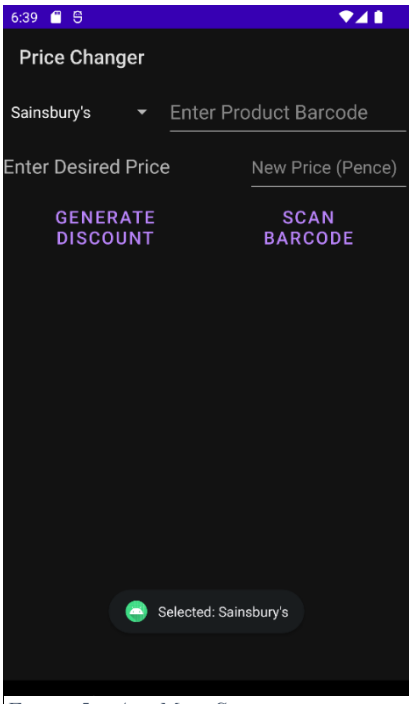

Figure 5 – App Main Screen

Includes Barcode Utilities

Prevents input errors and saves significant time by allowing the user to scan the original barcode for the item.

The generated discount barcode is also shown as large as possible to improve detection rates

Figure 8 - App Barcode scanner

Figure 7 – App Barcode Generation

3.7 Testing

3.7.1 Unit Tests

During development, I setup unit tests using JUnit to make sure that the reverse engineered discounting algorithms for all the supported stores are creating barcodes identical to the legitimate samples I discovered during research. These

3.7.2 Real World Testing

Once the app was fully developed and passing unit tests, I tested it in real stores in order to confirm that the theoretical vulnerabilities actually exist. There was always the chance that despite the simple barcode algorithms, discounts generated by actual employees were in fact stored in a database and only kept valid for short periods of time, so it was important to see that self-checkouts did in fact accept these "counterfeit" discounts. My tests included increasing price, decreasing price slightly, and decreasing price drastically. This was done without checking out and paying for items.

The stores I was able to test were Sainsbury's, Tesco, Waitrose, M&S, Co-op and Lidl. There were no Iceland or Morrisons stores with self-checkouts close enough to the University to complete testing.
Of the stores I was able to test, the app successfully allowed for the setting of arbitrary prices in Sainsbury's, Tesco, Waitrose, M&S and Co-op. When testing for Lidl was carried out, the fixed discounts did not work, and the self-checkout produced an "Unknown item" error. Despite this, the 30% discount at Lidl still worked.

3.8 Impact

At first glance, the impact of this attack may seem comparable to shoplifting, but there are three main reasons why this attack is actually more significant. For starters, some items such as gift cards can't be shoplifted at all, since must be activated at the checkout. By purchasing them at an arbitrary price, the gift card would still be activated for the full amount.

A persistent criminal can exploit this vulnerability for a very long period of time without ever getting caught. As long as the price reductions they generate are reasonable and comparable to genuine discounts (i.e. not discounting items that would never otherwise get reduced, and not purchasing everything for 1p), this behaviour is hard to detect in current inventory systems.

The inventory system also wouldn't identify any discrepancies in stock numbers, since the items are actually being bought and registered as purchased. Only a keen observer watching the checkout process would notice.

The attack can be very discrete if the attacker comes prepared with discount labels instead of having to scan their phone. They can just print their own labels and affix them on top of the original barcode before going to the checkout.

Finally, by actually purchasing the item, even if it is at an arbitrary price of the attackers choice, many security systems would also get bypassed. There are a variety of Electronic Article Surveillance (EAS) in place at most supermarkets to prevent shoplifting. For example, RFID labels are small stickers that are automatically disabled during the checkout process, without the need for intervention from the store staff. Larger security tags that must be manually removed would also likely get removed by employees during the checkout process.

4 Section B – Developing a Solution

The second part of this project is developing a secure alternative to the discounting systems that I went over in the previous sections. Not only must this alternative be cryptographically secure and robust, it must also be easy to integrate into existing inventory and Point of Sale systems.

4.1 Attack Vectors to Consider

The main attack to consider is the lack of authenticity on the discount barcodes. The reason I was able to attack existing systems by reverse engineering the discounting algorithms and producing my own discounts, is that the POS system was unable to differentiate between discount barcodes generated by staff and those generated by an attacker. If the POS system can't verify the authenticity of the discount, this issue will persist.

The second attack vector that exists in the current systems is replay attacks. Even if the user couldn't generate arbitrary discounts of their choice, as is the case with Asda's newer 499-series discount barcodes, an attacker can reuse any previous discounts that they've observed. This could mean buying 10 of the same item at a reduced price when only one should be discounted, or an attacker can take note of a barcode for reuse in days or weeks from it's inception. To address this, discounts must be made so that they expire after their intended lifespan, and become invalid.

There is also the consideration of a removal attack, where an attacker detaches a discount barcode from one item, just to place on a similar item of a different batch. For example, an attacker can take the barcode from a damaged item, and use it to purchase an intact one. If we solve the problem of replay attacks without addressing removal attacks, this could leave the originally discounted item invalid and at full price. However, the problem of tightly coupling a physical object with it's identifier is a problem that can't be solved with a purely digital approach. Addressing removal attacks can only be done with tamper-resistant Auto-ID techniques, such as stickers

The final attack vector still present in the current system is the so called "Banana Trick" [5]. As long as the only way the self-checkout system can differentiate between items is by weight, this issue will persist as there will always be multiple items that have the same weight.

4.2 Possible Approaches for Auto-ID

Automatic identification and data capture (AIDC) make it possible for systems to automatically identify objects and collect data about them. Without Auto-ID, stores would still have cashiers manually ringing up products and calculating costs.

4.2.1 Linear Barcodes

Linear barcodes such as EAN-8, EAN-13 and Code-128 barcodes are what's currently being used in all UK supermarkets. This is thanks to the GS1's standardisation of the Global Trade Item Number (GTIN), which incorporates UPCs, EANs, and more. These linear (or one-dimensional) barcodes can be read both by cameras as well as laser scanners. In most stores the self-checkout uses a camera based optical sensor rather than a laser scanner, despite the fact that a laser scanner is simpler, cheaper, and better at scanning moving barcodes.

The disadvantage of being restricted to linear barcodes is that they're impractical for storing larger amount of information. For example, if we want to encode 72 numeric only characters in a Code-128 barcode, it would look this:

Figure 9 - Code-128 Barcode Encoding 72 Numeric Characters

Figure 10 - PDF417 Barcode Encoding 72 Numeric Characters

We can still utilise the Y axis even if we're stuck with a laser scanner, by utilising a stacked linear barcode such as the PDF417. The lower width makes it much for feasible for affixing to item packaging.

Many laser scanners, such as the Symbol LS4000 series [16], support scanning PDF417 barcodes as well standard linear barcodes. The laser scanner can scan one row at a time, and each row is labelled with a row number in case they get scanned in the wrong order. In terms of data density, a PDF417 barcode is still not as good as a QR code, and in practice ends up with 4 times the footprint of a QR code for the same amount of data being encoded [17].

4.2.2 2D Barcodes

2D barcodes encode data in both the X and Y axis, and as such require an imager to scan. Examples include Aztec codes, QR codes and the DataMatrix. The advantage of 2D barcodes is that with an image scanner, they can be read in any orientation. 2D barcodes such as the QR code have a much higher data capacity and density than linear barcodes, and are not restricted to numeric to alphanumeric content.

Figure 11 - QR Code Encoding 72 Numeric Characters

QR codes come in 40 different sizes, called versions, each of which support 4 levels of error correction. The more of the QR code is dedicated to storing redundant data, the smaller the encoded message can be. We can also use the QR code more efficiently by avoiding mixing data types. Storing only numeric information takes up less space than storing alphanumeric information, and both take up less space than storing arbitrary bytes.

4.2.2.1 Secure QR Codes (SQRCs)

The QR code knows where the data ends as there is a terminator after the last character. This makes it possible to hide information from public view by storing data after the terminator. Most readers won't show data stored after the terminator, and this private area can be used to store extra information. Denso Wave's Secure QR Code (SQRC) works by encrypting the data stored in this hidden part of the QR code, such that only a scanner that is looking for this data and has the appropriate key can find it.

4.2.3 RFID

RFID is already used in supermarkets for EAS, but not so much as a form of Auto-ID. RFID tags come in several classes, where class 1 RFID tags are passively powered Write Once / Read Many (WORM), and class 5 RFID tags are actively powered, support several read/writes, and can communicate with other devices. Of particular interest are Electronic Product Code (EPC) Gen 2 tags [18], which are standardised class 1 RFID tags that contain a 96-bit URI in memory which uniquely identifies each tag.

Such tags can be sourced for as cheap as 5p per piece [19], which is financially viable for tagging items that are more than a few pounds. Another advantage of RFID is that a scanner can actively detect and interact with multiple tags at a time. RFID would also make it possible to prevent the "banana trick" attack, as the POS can just ensure that the tags being paid for are also those which have been placed in the weighed bagging area.

There also exist lightweight authentication protocols for class 1 RFID tags such as KMAP [20], which enables mutual authentication and is resistant to attacks such as traceability attacks, replay attacks and desynchronisation attacks.

4.3 Possible Approaches for Authentication

The POS needs to be confident that the discount was generated by someone authorised to do so, therefore we must be able to verify the authenticity of the discount. For this we can either use symmetric or asymmetric cryptography techniques. We also need to minimise the amount of extra data that we must include in the discount, as larger barcodes are more prone to damage and RFID tags with more memory are pricier.

4.3.1 Asymmetric Cryptography

Using asymmetric cryptography, the device generating the barcodes is in possession of a private key, which is used to generate a digital signature that is appended to the contents of the discounts. If we use RSA 2048, our signature will be 256 bytes, or 512 alphanumeric characters.

If we use Elliptic Curve Digital Signature Algorithm (ECDSA) instead, we can achieve the same strength of security with smaller keys and signatures. ECDSA signatures are 2 times longer than the signer's private key, so with a curve such as secp256k1, the produced signature will be 64 bytes, or 128 alphanumeric characters. A BLS digital signature can get even smaller than ECDSA,

4.3.2 Symmetric Cryptography

By shifting the burden of key exchange to a separate entity, we can use a shared secret key to generate a HMAC instead. Hash-based Message Authentication Codes (HMACs) are cryptographic functions that take a message and a secret cryptographic key, and produces a deterministic result. If the party producing the discount generates a HMAC with their private key, the POS can verify the authenticity of the discounter by producing the same HMAC, assuming the shared secret key is still secret and not compromised.

HMACs can use any hash function, such as MD5, SHA1 or SHA256. A HMAC-SHA1 value is only 20 bytes, which can be expressed as 40 hexadecimal characters, and is plenty secure for our purposes. HMACs also guarantee data integrity as well as authenticity.

We could also use authenticated encryption, which would assure the authenticity and confidentiality of our data at the same time. AES-GCM takes four inputs: A secret key, initialisation vector, the plaintext to be encrypted, and Additional Authentication Data. The nonce and AAD are passed in cleartext, which is useful as we can pass an ID for the secret key in the AAD to tell the POS which discounter generated the discount, so the corresponding for that discounter is used.

4.4 Chosen Approach

The approach I chose to go with is a QR code based discount code, which provides authenticity by including a HMAC generated with a secret shared key. The shared secret key must be established between each discount generator the Point of Sale before the discounter can produce valid discounts. To address replay attacks, discounts are encoded with a nonce as well as an expiration date. Each nonce can only be used once, and must be used before the expiration date.

I decided against using AES-GCM as there was no need for confidentiality of the discount, and I wanted to ensure message integrity instead. All the data encoded is encoded in the QR code normally, without SQRC techniques to hide any private data.

4.5 Implementation

The implementation was produced with Python, is split up into 3 parts, the discounter (Discounter.py), the POS system (Scanner.py), and the central server (API.py)

4.5.1 Discounter.py

The discounter is the entity responsible for generating discounts codes, authenticating them with HMACs, and encoding them in QR codes. The structure of a discount code goes like this:

```
discount = LL BBBBBBBB… PPPPPP TTTTTTTTTT NNNNNNNNNNNNNNNN
```

- *LL* is the length of the products original barcode
- *BBBBBBBB…* is the original barcode, which can be variable in length
- *PPPPPP* is the new price of the item, in pence
- *TTTTTTTTTT* is the time the discount expires, as a POSIX timestamp
- *NNNNNNNNNNNNNNNN* is an 8-byte nonce produced by a CSPRNG, represented as 16 hexadecimal characters

The discounter then uses it's secret shared key to calculate a HMAC for the discount, but since each discounter will have it's own private key and the scanner will need to know which discounter produced the discount, the discounter will also include a Key ID:

```
output = discount + IIIIIIII + SHA1HMAC(discount, key)
```

When the discount is ready, it registers the nonce with the API, and outputs the authenticated discount to a QR code to be scanned by the POS.

4.5.2 API.py

The API made using Flask, and provides a REST API to interact with a database that keeps track of the nonces. I chose to make an API that interacts with an SQL database because that's likely how the real world system is. The API provides three important functions

- submitNonce(), which is used by the discounter to register a unique nonce for each discount produced
- verifyNonce(), which is used by the scanner to check a nonce is still valid without destroying it
- consumeNonce(), which deletes the nonce from the database, permanently invalidating the associated barcode.

The difference is that verifyNonce() should be used by the POS before checkout to check a discount is still valid, whereas consumeNonce() should be called after checkout() to destroy the old discount.

4.5.3 Scanner.py

The scanner uses the computer vision library OpenCV to extract the contents of the discount QR code, and then breaks it down twice. First, like so:

DDDDD…, IIIIIIII, HHHHHHHH… = decode(QR Code Contents)

- *DDDDD*… is the serialised discount code
- *IIIIIIII* is the key ID of the discounter, always 8 hex characters
- *HHHHHHHH…* is the HMAC of the serialised discount code, always 40 hex characters

The scanner will then calculate the HMAC of the serialised discount itself, using the same key that the discounter did, and if the resulting HMAC is the same, it will break down the serialised discount:

LL, BBBBBBBB…, PPPPPP, TTTTTTTTTT, NNNNNNNNNNNNNNNN = breakdown(serialisedDiscount)

Finally, the scanner carries out two more checks, it will confirm that the discount is valid:

- Check that the current time has not exceeded the expiry date included in the discount
- Check the validity of the nonce with the API to make sure the discount hasn't been used before

If those two final checks pass, then the POS knows that the discount is authentic, hasn't expired yet, and hasn't been used before.

4.6 Results

To demonstrate how the three parts come together, I created Demo.py, which runs through the whole process starting from the key creation and ending with the consumption of a discount. To start, make sure the API is up on localhost:5000, and then run Demo.py, which will do the following:

1. A key for the HMAC are generated and placed where both the scanner and discounter can access them.
2. The discounter is called to produce 1 new discount, taking input from the user
3. It shows the QR code that was generated by opening a window
4. The scanner confirms the discount is valid by consulting the API
5. The demo sends a delete request to destroy the discount
6. The scanner checks to see if the discount is still valid

Figure 3 - Generated Secure Discount Code

The API must be running on localhost for the demo to work, otherwise the discounter can't register the discount, the scanner can't verify the discount, and the demo can't destroy the discount.

As the demo runs, it should show the discount QR code that was generated. It should also be saved to a discounts/ folder in the working directory.

5 Conclusion

During this project, I rediscovered a vulnerability that has lying dormant for almost 10 years, hiding in plain sight at every supermarket in the UK. By shining a light on the severity of the current systems as well as producing the alternative system that would be invulnerable to these attacks, I hope to show that things could, and should, be better.

6 Future Enhancements

The solutions demonstrated were not very effective at addressing the physical vulnerabilities of systems. Despite identifying a risk of removal attacks, there was no way to mitigate them without tamper-resistant equipment. By distributing the discount generation process and storing secret key across multiple portable devices, we also run the risk of attackers gaining access to the devices and extracting the keys. This risk should be possible to mitigate through use of a Trusted Platform Module, by storing the key on the TPM with a policy that doesn't allow for it's extractinon.

7 References

[1] "OpenCV Python," [Online]. Available: https://github.com/opencv/opencv-python.

[2] "Flask," [Online]. Available: https://github.com/pallets/flask/.

[3] "Pure Python QR Code generator," [Online]. Available: https://github.com/lincolnloop/python-qrcode.

[4] "ZXing Android Embedded," [Online]. Available: https://github.com/journeyapps/zxing-android-embedded.

[5] "The Banana Trick and Other Acts of Self-Checkout Thievery," 7 February 2018. [Online]. Available: https://www.psychologicalscience.org/news/the-banana-trick-and-other-acts-of-self-checkout-thievery.html.

[6] "GS1 General Specifications," GS1, [Online]. Available: https://www.gs1.org/docs/barcodes/GS1_General_Specifications.pdf.

[7] "GS1 Company Prefix," [Online]. Available: https://www.gs1.org/standards/id-keys/company-prefix.

[8] "Tesco Discount Barcodes, Cracked," [Online]. Available: https://web.archive.org/web/20120723224611/https://mtdevans.com/projects/barcode/.

[9] MartinMcGirk, "Hacker News," 2012 12 24. [Online]. Available: https://news.ycombinator.com/item?id=4284849.

[10] @mtdevans, 2012 July 22. [Online]. Available: https://web.archive.org/web/20160705185205/https://twitter.com/mtdevans.

[11] Anonymous, "@YourAnonNews - Twitter," [Online]. Available: https://twitter.com/YourAnonNews/status/227572237392297984.

[12] M. Evans, "Tesco Discount Barcodes Cracked, Removed," [Online]. Available: https://mtdevans.com/projects/barcode/.

[13] Peter, "Asda Reduction Barcodes," [Online]. Available: https://greenhac.org.uk/2012/10/asda-reduction-barcodes/.

[14] "Last Usage of Asda 205-Series Discount," 20 July 2017. [Online]. Available: https://www.novini.london/%D0%BD%D0%BE%D0%B2%D0%B8%D0%BD%D0%B0/24971-%D0%9A%D0%B0%D0%BA-%D0%B4%D0%B0-%D0%BF%D0%B0%D0%B7%D0%B0%D1%80%D1%83%D0%B2%D0%B0%D0%BC%D0%B5-%D0%BD%D0%B0%D0%B9-%D0%B5%D0%B2%D1%82%D0%B8%D0%BD%D0%BE-%D0%B2-%D0%9B%D0%BE%D0%BD%D0%B4%D0%BE.

[15] D. Bourne, "Braving the whoopsie aisles - where shoppers swoop for reduced supermarket cakes, sausage and quiche," Manchester Evening News, 11 February 2018. [Online]. Available: https://www.manchestereveningnews.co.uk/whats-on/shopping/whoopsie-aisle-supermarket-reductions-asda-14272130.

[16] "LS 4000i Scanner Series," Scanner, [Online]. Available: https://cdn.barcodesinc.com/themes/barcodesinc/pdf/Symbol/ls4000.pdf.

[17] "Using Barcodes in Documents – Best Practices," Accusoft, [Online]. Available: https://web.archive.org/web/20120524085651/http://accusoft.com/whitepapers/barcodes/BarcodesinDocuments-BestPractices.pdf.

[18] "EPC Tag Data Standard," GS1, [Online]. Available: https://www.gs1.org/sites/default/files/docs/epc/GS1_EPC_TDS_i1_11.pdf.

[19] "EPC Gen2 RFID Tags on Alibaba," [Online]. Available: https://www.alibaba.com/product-detail/DTB-Long-Range-Monza4E-B42-epc_1600176777880.html?spm=a2700.galleryofferlist.normal_offer.d_title.26714b8fBYi09V&s=p.

[20] M. N.-u.-I. S. S. Umar Mujahid, "A New Ultralightweight RFID Authentication Protocol for Passive Low Cost Tags: KMAP," [Online]. Available: https://doi.org/10.1007/s11277-016-3647-4.

[21] "Market share of grocery stores in Great Britain from January 2017 to May 2021," Statista, [Online]. Available: https://www.statista.com/statistics/280208/grocery-market-share-in-the-united-kingdom-uk.

[22] "Electronic Article Surveillance (EAS)," [Online]. Available: https://en.wikipedia.org/wiki/Electronic_article_surveillance.

[23] "ISO/IEC 15417:2007 - Code 128 bar code symbology specification," [Online]. Available: https://www.iso.org/standard/43896.html.

[24] "ISO/IEC 15438:2015 - PDF417 bar code symbology specification," [Online]. Available: https://www.iso.org/standard/65502.html.

[25] "Using Barcodes in Documents – Best Practices," [Online]. Available: https://web.archive.org/web/20120524085651/http://accusoft.com/whitepapers/barcodes/BarcodesinDocuments-BestPractices.pdf.

[26] "ISO/IEC 24728:2006 - MicroPDF417 bar code symbology specification," [Online]. Available: https://www.iso.org/standard/38838.html.

[27] W. Z. S. P. Yu-Ju Tu, "Security Threats and Solutions for Two-Dimensional Barcodes: A Comparative Study," [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0167923620302268.

[28] W. Z. S. P. Yu-Ju Tu, "Critical risk considerations in auto-ID security: Barcode vs. RFID," [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0167923620302268.