Ethical and Professional Concerns About Online Banking Fraud: A Whitehat Hacker's Perspective on Social Responsibility and Pointing Out the Stupidity and Intentional Ignorance of Others

Anonymous

Purpose and Intent of this Article

The genesis of this article was aimed for local distribution as a whitepaper in the Florida area to provide unwitting banking customers with information about the mis–handling of their sensitive financial data with respect to Bank of America's policy of dealing with accusations of fraudulent activity on local accounts. In the process of writing out the concerning details of my in–person interactions with so–termed customer support representatives at my local branch, I have included a number of examples of charges from my own and family member's financial statements that are too important to leave out from a discussion of mobile and online banking fraud carried out on a larger scale in the United States. It is hard to believe that Bank of America is unaware of these potential abuses built in to use of their proprietary banking systems, and they are apparently claiming intentional ignorance of the misuses of customer financial data by policy. Moreover, the support staff that implement the financial recovery process for fraudulent transactions guaranteed on all transactions with this bank are under–manned, insensitive, and mostly ignorant in handling information of this type.

The original framework for distributing the information contained within this article was to provide a service for duplicating suspect fraudulent charges noticed on paying customer accounts up to the unique ID numbers associated with the original transactions at a rate of $12 per two or three good examples obtained by expert consultation, also the current going monthly service rate charged by Bank of America since the fraudulent debit activity on my account caused a dip its the balance last year. Since this was considered too greedy approach by some in the local community and greater Florida area, I have decided to provide the relevant details in this format to educate users of this potential threat to their account transactions with Bank of America and with other national online banking institutions.

Know Your Rights

Timelines for Effective Action Mandated by Federal Law

Compared to Bank of America's grossly and underwhelming mis-advertised timelines for their fiscal accountability with your sensitive financial data, underspecified at 10 to 60 days depending on who you ask, when you ask, and the phonetree worker routed through Mobile, AL from your cell phone, U.S. federal law provides an unlimited time to recover funds removed from your checking and/or savings accounts by fraudulent activity. This is functionally a statement mandated by U.S. (inter)national law providing unwitting consumers an indefinite postponement of the process under any abnormal circumstance to protect the rights of customers and victims of identity theft regardless of social status or the financial means to hire teams of statisticians and lawyers to protect their assets and hard earned money.

In other words, the Bank of America corporation is unconditionally accountable to their customers independent of the convenient policies stated on their website and even in their print mass mailing notices of what they think is an appropriate timeline appropriate for them to act on your behalf. Moreover, Bank of America and other similar financial institutions are accountable to their customers who have not filed explicit claims of fraudulent activity and should have their paid expert technical analysts proactively finding other small errors executed on a larger scale without having to be asked to do so. This is a part of the implied social contract formed with these financial institutions for your business and their rights in managing the sensitive personal information required for the day–to–day operations of these large–scale corporations.

Legal Rights for "Hacking" Your Own Bank Account to Detect and Discover Fraudulent Activity

According to jurisdiction under federal law, the consumer is granted protections to post online banking transactions of any nature to their own financial accounts. Again, to summarize and be clear, if someone employed by your bank tells you that "hacking" is a crime and you are to be arrested in the state of Florida for replicating suspect charges on your own account, please do proceed to issue them an appropriate however understated middle finger gesture, do it any way, and then call the police on them for criminal negligence and obstruction of justice. These laws and the U.S. constitution provide consumers a bare minimum of these legal rights – even if you manage to "piss off" airheads and stupid people endowed with a title by Bank of America (or otherwise) in due legal process.

In particular, knowledge is power, educated users should be aware of their rights, and simply processing the technical know–how to exploit a vulnerability in the the flawed banking systems used to manage your own sensitive personal data does not make you an evil counterpart to the perpetrators of these criminal activities. Additionally, federal and Florida state laws (along with CA, CO, IL, MO, SD, and WA depending on jurisdiction and the sources of the on–shore electronic wire transfers involved in routing these transactions) require a minimum of ten plaintiffs to file a class action lawsuit under sufficiently competent legal counsel for wrongdoings of this nature, so be prepared as an individual to substantiate your claims well by clear–cut demonstration–of–point example and to necessarily motivate the designated customer service representatives at Bank of America to investigate the fraudulent charges to your account.

My experience consulting with Bank of America employees on staff for an on–site scheduled appointment was to have a print–out of their newly mailed policies (which I was mysteriously made unaware of up to that point) on correcting fraudulent charges printed, highlighted, and then handed to me by their most experienced statistician of financial engineering available to me for my visit at that time. This response, incidentally, may or may not be representative of their corporate policies in larger state–wide or national practice. When we met for my scheduled visit this time, I was told by my designated financial representative (Figure 1) to check whether it is actually worth it and in my best fiscal interest to request alternate copies of the prohibitively expensive transaction history associated with my account (now only available to me upon serious inquiry at the rate of $5 per statement per month for the last 5–6 years) if, in fact, their in–house fraud department would be willing to help me at all.

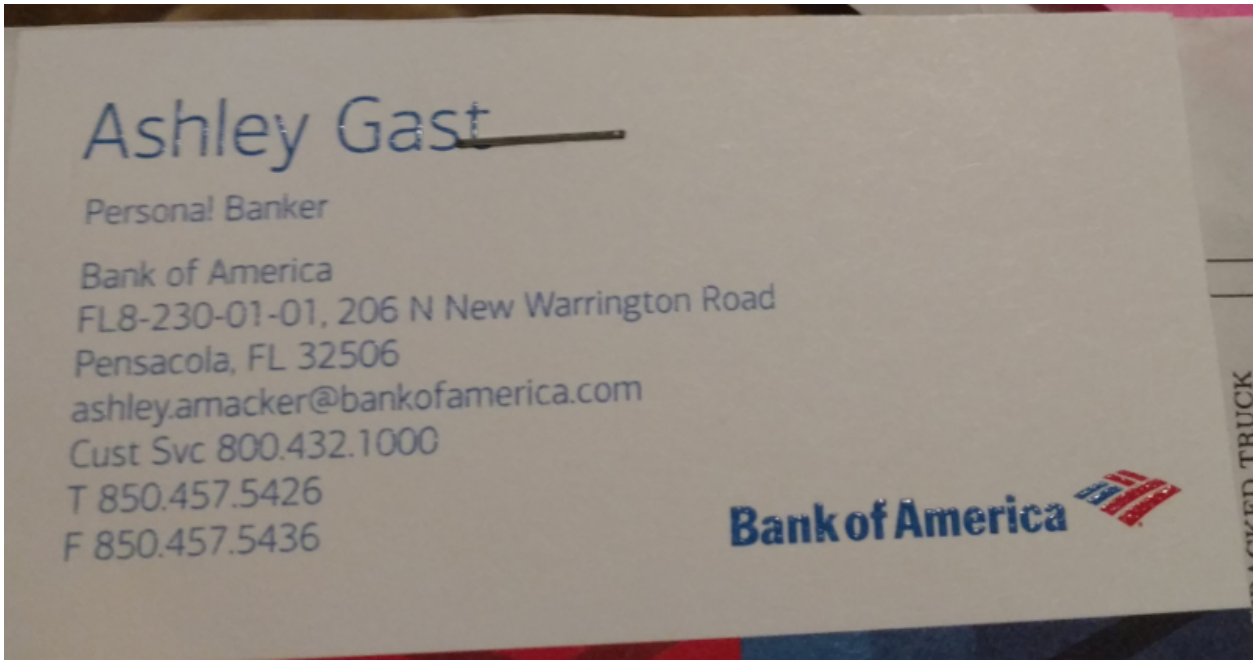


Figure 1: Contact Information for Bank of America's Favorite Local Technology Expert and Financial Engineering Major at Large

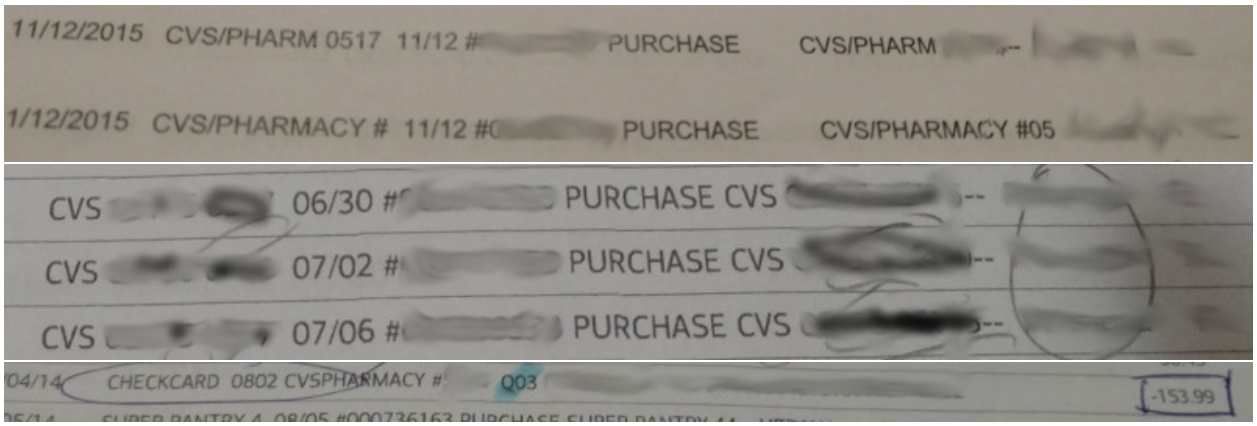


Figure 2: CVS Pharmacy Transactions and Separate Payment Charges in Florida

Pseudocode for the USAePay Mobile Banking Websites

Disclaimer

Before we proceed with the next information contained in the next pseudocode source listings and with the opinionology a priory to be attributed strictly to the author, please note that I do not regularly, and have barely ever, been able to write functional source code in Visual Basic (or VB Script). My disgust with stumbling onto the Vericheck and USAePay websites in my search for information about current technology laws has led to our forays into the open source freeware code on the next pages to be written in the ugly, partially functional, however still disturbingly accurate, demonstrations given below. Any of the languages supported by a modern installation of Microsoft's Visual Studio IDE should also suffice to make this point (C# / Java, XML, or just plain old javascript will do in the provided API framework examples on these sites).

Explanation and Implicit Documentation of this Light API

What this API (a.k.a. Application Programming Interface) effectively lets you do is enable a mildly experienced novice "*script kiddie*" with a "*would do, but never ever should do*" mentality (and more to the understated point), rather than more mastered technical expertise in the form of seamless "*can do*" hacking magic, to use an internet–enabled Windows computer to easily manipulate the sensitive personal financial data of millions in the greater Florida area.

The source code provided in the listing below is semi–functional and needs significant overhauls to demonstrate the under–documented API functionality specified through thinly veiled examples documented on the USAePay and Vericheck WIKI websites that accompany the freely available sandbox usage for testing non–production output before subscribing to their for–pay online services. I expect that the bullet points in the next section can be re–implemented easily by sufficiently motivated readers and I encourage improvements to this codebase for the intended user education purposes of this article.

Undocumented API Functionality

Other undocumented functionality in the USAePay API specification given on the websites referenced in the accompanying VB source code examples includes the functionality listed in the following examples:

1. Brute–force discovery of semi–random customer data points. See the verbose descriptions of the error codes returned by the API documented on the "ACH Return Codes" and "NOC Codes" on the Vericheck pages, and by the Wikipedia documentation for Address Verification System (AVS) return code entries.

2. Repeat or recurring transactions differing by small amounts. There are at least a couple of references available online to thinly veiled documentation of small charges and service upgrade fees charged to implement transactions of this type.

3. Updating routing numbers of failed scanned check (RDC) transactions using the ACH return codes from above. Since the bank routing number information is all that is required to access a customer's unique sensitive financial data, we observe that this loophole effectively provides an attacker with

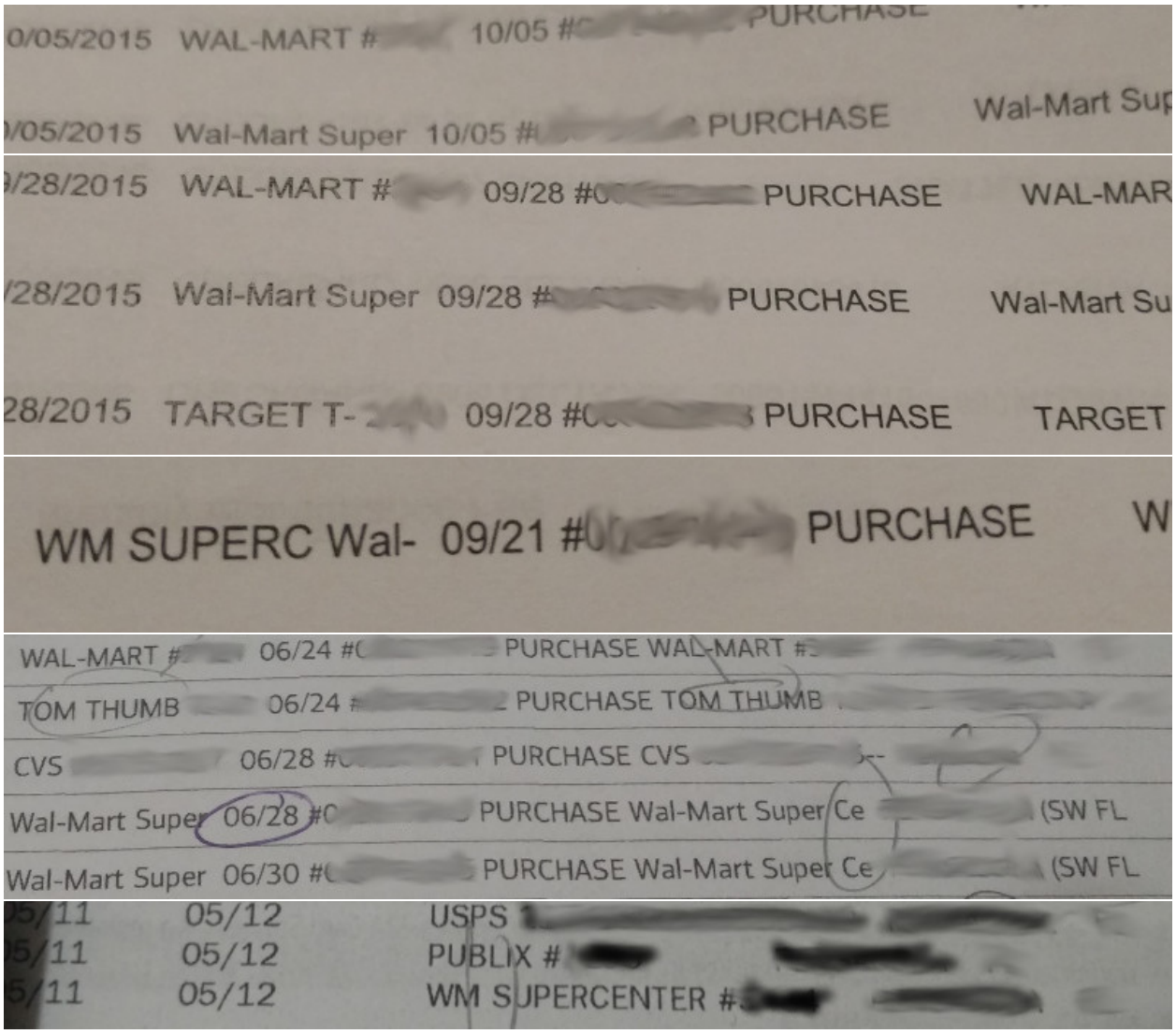


Figure 3: Examples of At Least Four Distinct Walmart Transaction Headers (and One Assumed At Least Partially Legitimate CC Bill Entry) Originating From the Same Local Store and Same Two Cash Registers in Florida

perpetual access to your funds with Bank of America's flawed proprietary systems as they are currently implemented in practice.

4. Other untested noteworthy API functionality for handling debit / credit card transactions, bulk transaction uploads, documentation of the transaction object fields, and the processing of credit card batches. See the online API WIKI pages for the methods used to processs Bulk Transaction Uploads, see the API Functions to submit multiple transactions by single same-day-bulk uploads to avoid higher overhead processing fees.

5. Ability to lookup previous transaction records in full by names, unrelated unique identifiers or other customers, and by transaction description strings.

6. There is sandbox access available to testers before posting transaction strings to live accounts for processing.

More Information and Links to Online Reading Material

Explanations of Suspicious Charges By Example

See Figures 2 – 4 referenced on the next and previous pages.

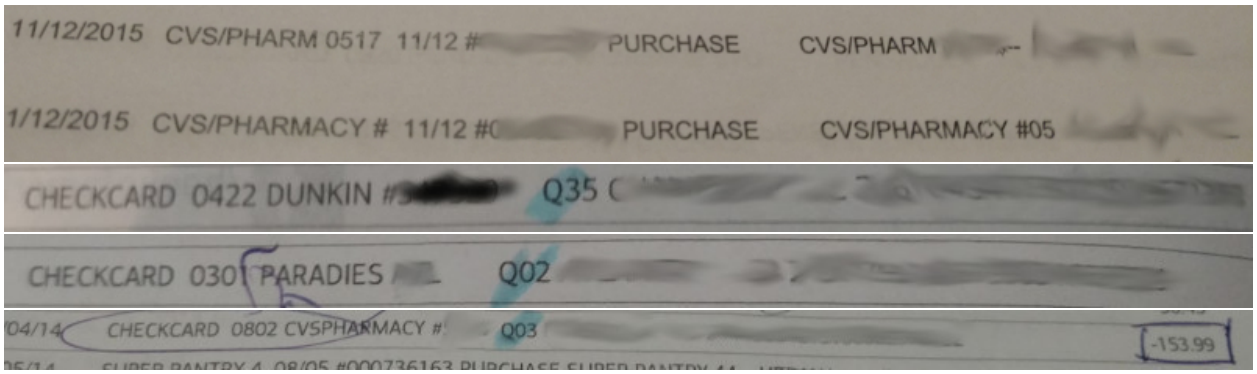


Figure 4: Dunkin Donuts in Champaign, IL and a Few Other Examples of Mysteriously Concatenated Magic Number Strings in Suspicious Transaction Headers

Homograph Attacks

Some examples of well documented homograph attacks detailed in the whitepapers online in the context of banking transaction headers include Unicode varieties of the '-' (dash) characters, misuses of print capital versus lowercase letters, and hash marks, among other notable violations. It is typically considered good programming practice to be cautious with transaction record strings containing multiple similar or at least visually indistinguishable markings with Unicode character sets, for example, to avoid unnecessary bank fraud (refer to Figure 5). This is one argument, in addition to non–standardized font printout possibilities in the summary versions provided to customers by the monthly statements issued by your bank, for why your printed Bank of America transaction logs alone are inadequate to detect all possibilities of fraudulent transactions. It is also important to check transaction headers and records for hash marks, italicized print characters, the abuses of 0x0999 in numeric portions of transaction headers (refer to Figure 6), and fixed width font variations to identify all possibilities for documentable fraudulent activity on sensitive financial accounts.

Other Examples of Suspicious Transaction Record Patterns

Transaction entries for hard–wired, non–checkcard records not posted on the same day should throw an immediate red flag in auditing suspicious activity on an account (see the tables provided on the ACH Funding at a Glance page in Figure 7). Other suspicious activities to look out for on your account include ARC, BOC, and NSF transaction header oddities in the listings typically contained in separate listings on Bank of America's monthly statement printouts (see Figure 8).

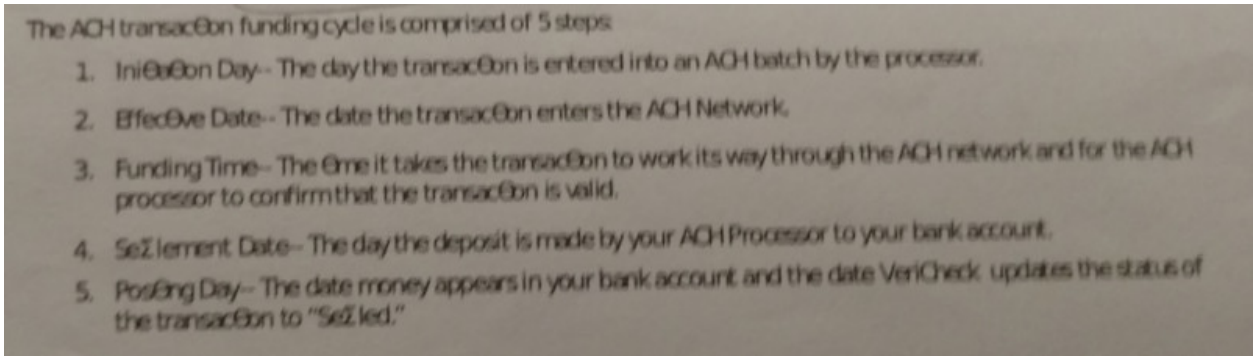
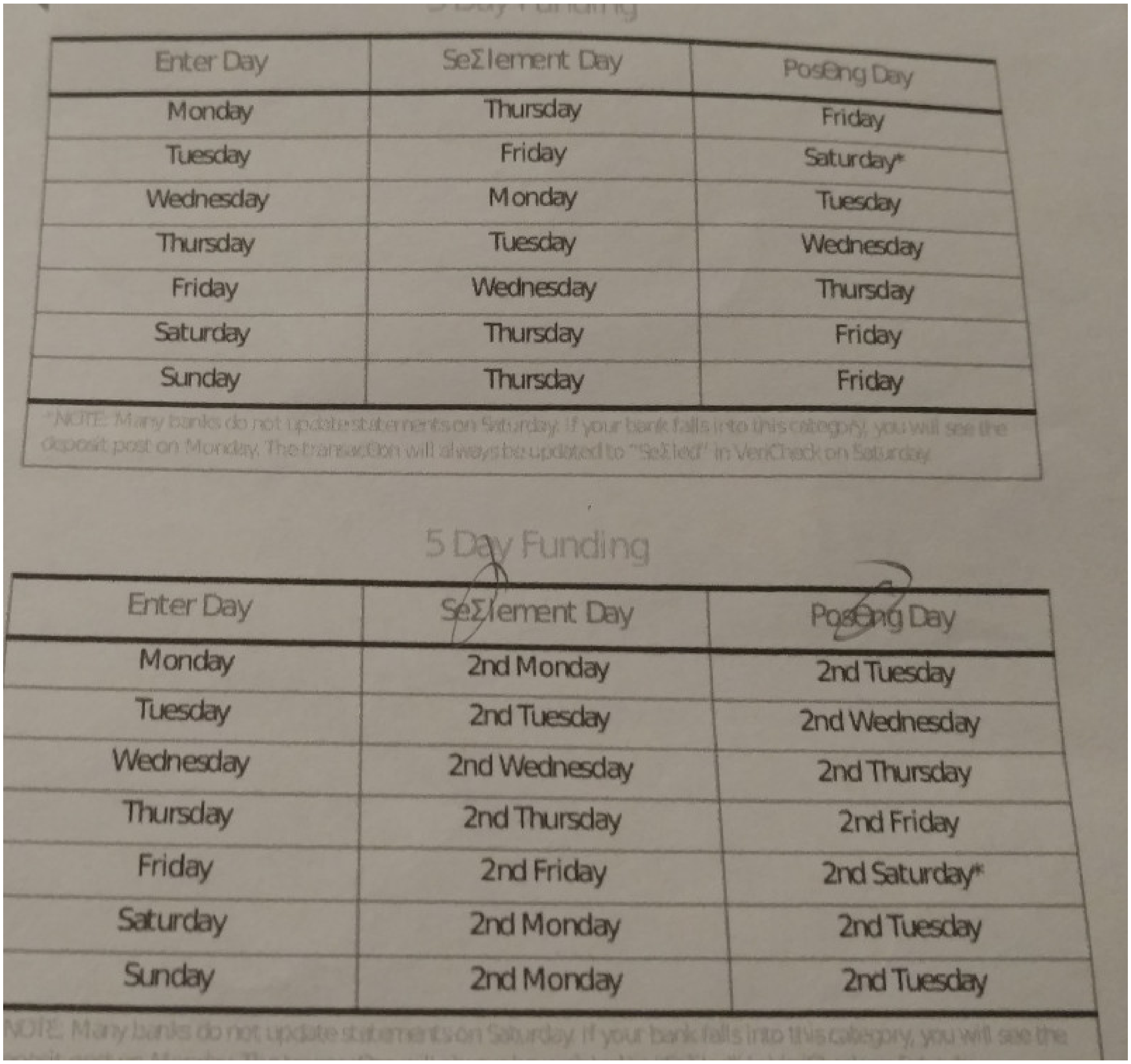


Figure 5: Printouts of PDF to Text Translations Illustrating Our Findings With Regard to the Homograph Attack Framework

Figure 6: Another Example of a Credit Card Transaction With Bank of America



Enter Day	SeΣlement Day	PosΣng Day
Monday	Thursday	Friday
Tuesday	Friday	Saturday*
Wednesday	Monday	Tuesday
Thursday	Tuesday	Wednesday
Friday	Wednesday	Thursday
Saturday	Thursday	Friday
Sunday	Thursday	Friday

*NOTE: Many banks do not update statements on Saturday. If your bank falls into this category, you will see the deposit post on Monday. The transaction will always be updated to "SeΣled" in VeriCheck on Saturday.

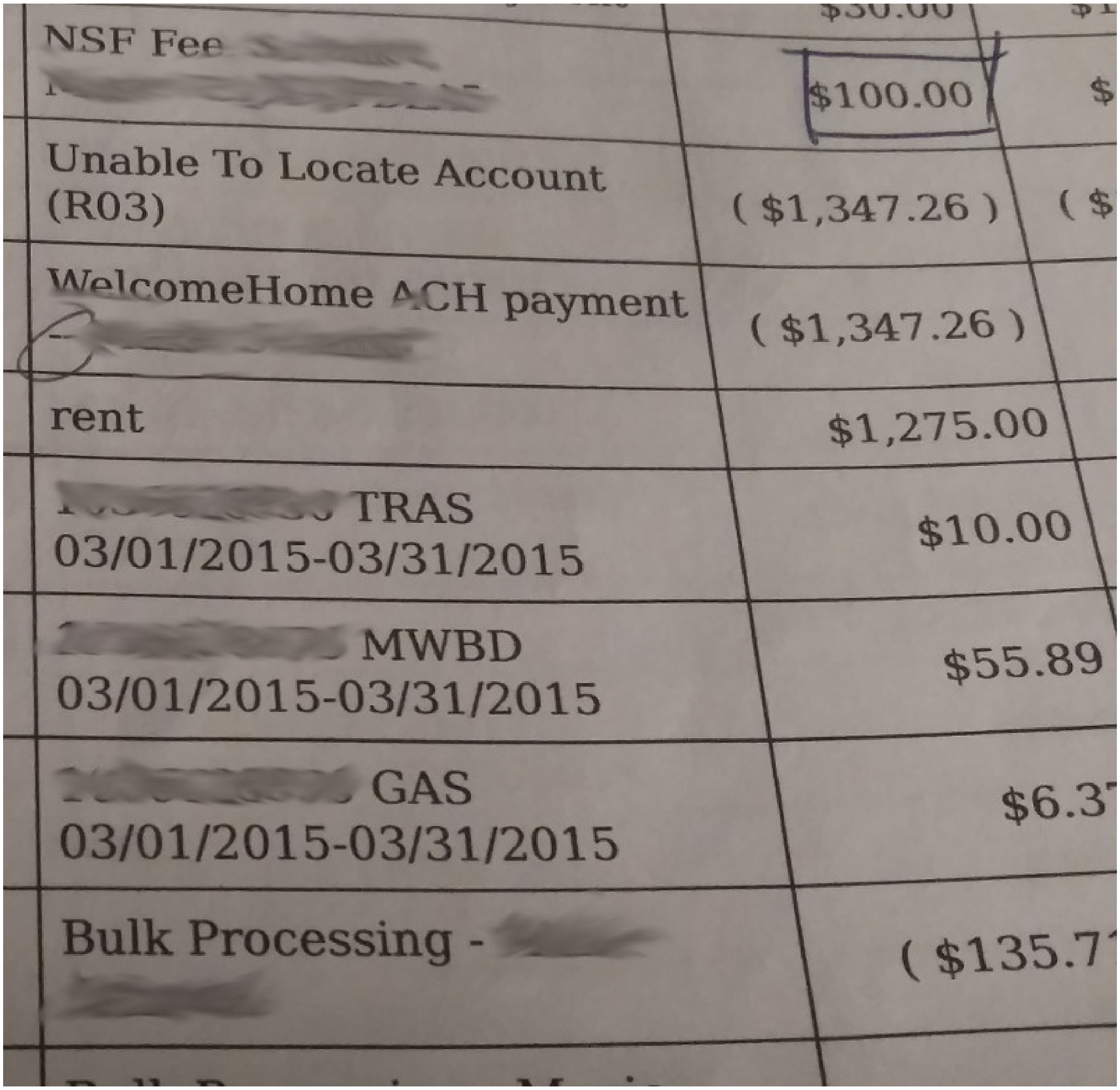
5 Day Funding

Enter Day	SeΣlement Day	PosΣng Day
Monday	2nd Monday	2nd Tuesday
Tuesday	2nd Tuesday	2nd Wednesday
Wednesday	2nd Wednesday	2nd Thursday
Thursday	2nd Thursday	2nd Friday
Friday	2nd Friday	2nd Saturday*
Saturday	2nd Monday	2nd Tuesday
Sunday	2nd Monday	2nd Tuesday

NOTE: Many banks do not update statements on Saturday. If your bank falls into this category, you will see the

Figure 7: Tables of Bulk Processing Timelines

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'' Source Code References Constructed From Consulting the
'' USAePay API and Online Verisign Documentation on 11/20/2015:
''      (1) http://docs.vericheck.com/v1.0/docs/obtaining-credentials
''      (2) http://wiki.usaepay.com/developer/statuscodes
''      (3) http://www.vericheck.com/ach-return-codes/
''      (4) http://wiki.usaepay.com/developer/soap-1.4/methods
''      (5) Supporting documentation from the online WIKI guide references
''          "What is Vericheck?" and "ACH Funding at a Glance"
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'' The following provides a listing of pseudocode, or more loosely speaking,
'' informal untested functional technical documentation prepared to educate the
'' reader. This sourcecode, and moreover the information prepared in this entire
'' document without the contained contributing author information referenced on the
'' first page, is informally released under a tentative open source creative-commons-
     like
'' license for non-comercial use and greater-good-type public education purposes.
'' What this means in non-technical speak is that you may blatently copy the
'' next source code and freely distribute the information contained in this
'' whitepaper writeup to educate yourself and to prevent this fradulent misuse of
```



	$30.00	$1
NSF Fee [redacted]	$100.00	$
Unable To Locate Account (R03)	($1,347.26)	($
WelcomeHome ACH payment [redacted]	($1,347.26)	
rent	$1,275.00	
[redacted] TRAS 03/01/2015-03/31/2015	$10.00	
[redacted] MWBD 03/01/2015-03/31/2015	$55.89	
[redacted] GAS 03/01/2015-03/31/2015	$6.3	
Bulk Processing - [redacted]	($135.7	

Figure 8: On–Shore NSF Fees Incurred From Bank of America

```
'' bank transactions from happening again to your neighbor's checking and savings account.

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'' Definitions of Constants and Other Magic Numbers in the Source Code:
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
On Error GoTo ThrowUserError;
Dim DEFAULT_ERROR_CODE As Unsigned Integer = 0xdeadbeef;
Dim OurLocalBranchExperience As Integer = 11 << 6 Xor 19 << 4 Xor 2015;
Dim uniqueSessionIDStr = "ABCDEFG-%{RemoteOSSessionIDStr}-%{strftime}" % \
   ("1234", "LinuxMintIsWindows9ImprovedManPages");

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'' Declare Variables to Fill the Requisite Transaction Fields in the API:
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Dim merchantID, transSummatoryAmount, transDesc, transType, transSEClass As String;
merchantID = "IWouldntButSomeoneElseWouldAndMightHave_sdkf02322512";
transSummatoryAmount = "153.99";
   ' From my favorite local pharmacy apparently submited to pay for a
   ' stock, recurring, and otherwise routine prescription to be filled last summer
```

```
39 %transDesc = "Something Routine, Overly Simplistic, and Most-Plausible-Probable-Cause-
       Type String Handling";
40 %transType = "debit";
41 %transSEClass = "POP";
42    ' Also: TEL, CCD, BOC, or WEB
43    ' See "Types of Vericheck Transactions and Funding" at the online WIKI site
44 % TODO: Implement full setup of the transaction request object for use immediately below
45 '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
46 '' Transaction Objects Defined by the API:
47 '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
48 Dim malwareAPIClientInstance As usaepay.usaepayService;
49 %malwareAPIClientInstance = New usaepay.usaepayService;
50
51 %Dim localUniqueTransactionToken As ueSecurityToken;
52 %localUniqueTransactionToken = API.CreateToken(uniqueSessionIDStr, 0xcafebabe);
53
54 %Dim ourProofOfConceptTransaction As usaepay.TransactionRequestObject;
55 %ourProofOfConceptTransaction = New usaepay.TransactionObject;
56 %ourProofOfConceptTransaction.Details = New usaepay.TransactionDetail;
57 %ourProofOfConceptTransaction.Details.Amount = transSummatoryAmount;
58 %ourProofOfConceptTransaction.Details.AmountSpecified = true;
59 %ourProofOfConceptTransaction.Details.Invoice = "232431";
60 %ourProofOfConceptTransaction.AuthCode = "ReturnedAPICall";
61 %ourProofOfConceptTransaction.RefNum = "AnotherReturnedAPICall";
62
63 '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
64 '' Submit and Process Our Untested Replicate Transaction:
65 '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
66 %Dim processedTransactionResponse As usaepay.TransactionResponse;
67 %processedTransactionResponse = New usaepay.TransactionResponse;
68 %processedTransactionResponse = malwareAPIClientInstance.captureTransaction(
       localUniqueTransactionToken, \
69 %  ourProofOfConceptTransaction.RefNum, ourProofOfConceptTransaction.Details.Amount);
70 %statusMessage = processedTransactionResponse.ResultCode == "A" ? \
71 %  "Approved, Reference Number is " & processedTransactionResponse.RefNum : \
72 %  "Declined for " & processedTransactionResponse.Error;
73 %System.StrError.printf("Status Code %c: %s\n", statusMessage,
       processedTransactionResponse.ResultCode);
74 %'' And so on
75
76 '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
77 '' Some Good Programming Practices for Default Error Handling,
78 '' Both Stock and Implied by Descriptive Suggestion:
79 '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
80 ThrowUserError:
81   GoTo BankOfAmerica
82 End Sub
83
84 BankOfAmerica:
85    ' Default descriptive expectation of privacy and accountability
86   On Error GoTo Hell
87   Err.raise OurLocalBranchExperience + DEFAULT_ERROR_CODE, _, \
88     "Error Class (None): Don't Shoot the Programmer, Blame Someone Else's System", \
89     "Error Message: For default expectations of privacy and accountability " + \
90     "in Hell, consult the next lines"
91 End Sub
92
93 Hell:
94   Debug.print
95     "Infinite Loop: " + \
96     "Rinse, Lather, Repeat (Install Linux and Kill Process to be Sure)"
97   GoTo ThrowUserError
98 End Sub
```