BLAZE
INFORMATION SECURITY

Common security pitfalls of banking and financial applications

April 28th 2016 • Julio Cesar Fort • Version 1.0
prepared for Rotterdam University of Applied Sciences

$ whoami

- Co-founder and security engineer at Blaze Information Security
- Been around in the security community and industry for a few years now
- Likes punk rock and ska – off to Groezrock festival to see Rancid tomorrow!
- Currently contracting for one of world's largest financial institutions

Agenda

Presentation roadmap

What will be discussed today and
what to expect from this presentation

Agenda

- A brief overview of banking systems
- Common vulnerabilities in financial applications
- Real life application security failures
- Basic security checklist for the project
- Other applications

Banking systems

A brief overview of banking systems

A word or two about core banking systems and how security never seem to come first

Banking systems

- Many banks implement off-the-shelf core banking solutions
- Think of them as a “bank in a box”
- Popular core banking systems include Oracle Flexcube, Infosys Finacle, Tata TCS BaNCs, Temenos
- In many cases banks customize the solution and add its own UI + website on top of it, but the back-end is still the same
- Very few public security research and scrutiny from the IT security community

References:

http://www.sfchronicle.com/business/article/Friendly-hackers-ignored-when-pointing-out-bugs-6647673.php
https://www.sec-consult.com/fxdata/seccons/prod/downloads/sec_consult_capgemini_study_application_security_for_cbs_201210_v101.pdf

Where it breaks

Common vulnerabilities in financial applications

Financial applications have their own specific threat profile.
Context matters

Where it breaks – Race conditions

- Race conditions are hard to spot and to exploit in practice
- Race conditions in remote web apps is even trickier due to network jitter, etc.
- A few successful hacks have been documented against Bitcoin exchanges and Starbucks gift cards

```
transfer(src_account, dst_account, amount) {
    available_balance = read_balance(src_account)

    if (amount <= available_balance) {
        new_balance = available_balance - amount
        update_balance(new_balance)
        execute_transaction(src_account, dst_account, amount)
    }
}
```

References:

http://hackingdistributed.com/2014/04/06/another-one-bites-the-dust-flexcoin/

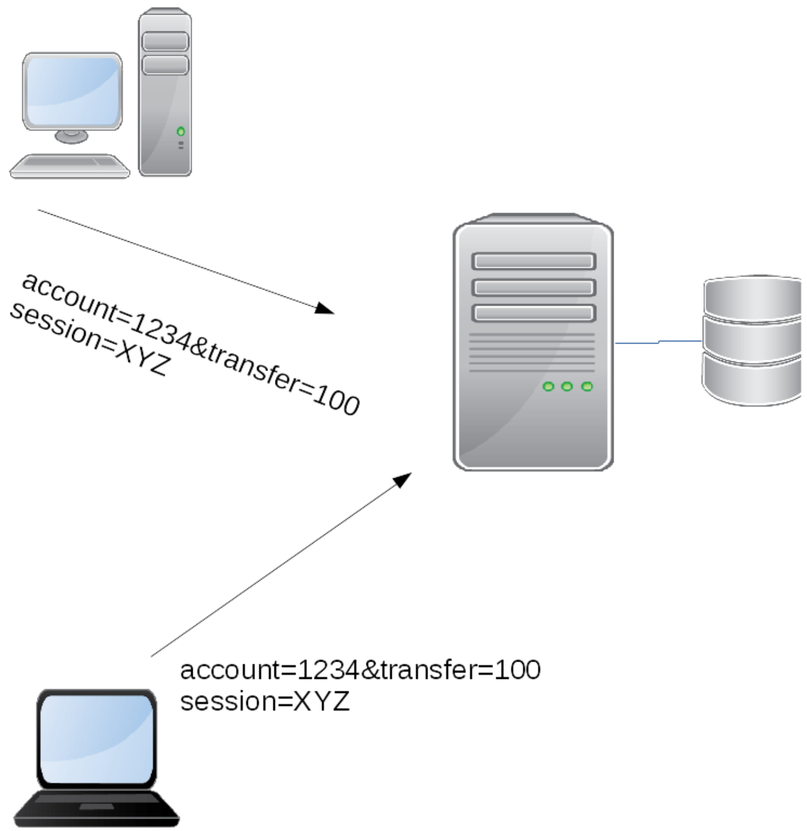
https://www.reddit.com/r/Bitcoin/comments/1wtbiu/how_i_stole_roughly_100_btc_from_an_exchange_and/

http://sakurity.com/blog/2015/05/21/starbucks.html

https://bitcointalk.org/index.php?topic=499580

https://defuse.ca/race-conditions-in-web-applications.htm

Where it breaks –
Race conditions



Where it breaks –
Insufficient authorization

- Insufficient authorization checks can lead to data exposure and other problems
- This is not that hard to get right, but easy to forget once in a while
- One can often find issues of privilege escalation, both horizontal and vertical

Where it breaks – Rounding bugs

- Different exchange rates for buying and selling foreign currency
- Money in real world operates down to two decimal digits, whereas exchange rates go down to a lot more digits
- After calculation, the final result is rounded to the nearest number up or down (i.e., $9.856 gets rounded to $9.86)

References:
http://blog.acrossecurity.com/2012/01/is-your-online-bank-vulnerable-to.html

Where it breaks – Rounding bugs

Polish Zloty	Euro	Rounded	Resulting conversion rate PLN-EUR
4.38	1	1	0.22831 (official)
0.01	0.0022	0.00	WE LOSE
0.02	0.0045	0.00	WE LOSE
0.03	0.0068	0.01	**0.33333**
0.04	0.0091	0.01	0.2500
…	…	…	…
0.08	0.0182	0.02	0.28571
0.09	0.0205	0.02	0.28571

Where it breaks –
Logic issues

- Transfers using negative or extreme values
- Bypassing a required step – i.e., jumping over the two factor authentication check

Where it breaks – Logic issues

- Tampering with parameters to change the source account where money is transferred from
- Other parameter-related issues: remove a parameter altogether, send a parameter without a value, send the parameter twice in the same request with the same values or different ones, add new parameters, repeat parameters from a previous step into a new one, etc.

Where it breaks – Denial of service

- Unusual user-supplied input may cause the application throw unhandled exceptions
- Not limited to the application: widespread account lockouts can cause business denial of service
- Certain subnormal numbers can cause troubles

References:
http://www.exploringbinary.com/php-hangs-on-numeric-value-2-2250738585072011e-308/
http://www.exploringbinary.com/java-hangs-when-converting-2-2250738585072012e-308/

Where it breaks – Denial of service

PHP Hangs On Numeric Value 2.2250738585072011e-308

By Rick Regan (Published January 3rd, 2011)

I stumbled upon a very strange bug in PHP; this statement sends it into an infinite loop:

```
<?php $d = 2.2250738585072011e-308; ?>
```

(The same thing happens if you write the number without scientific notation — 324 decimal places.)

I hit this bug in the two places I tested for it: on Windows (PHP 5.3.1 under XAMPP 1.7.3), and on Linux (PHP Version 5.3.2-1ubuntu4.5) — both on an Intel Core Duo processor. I've written a bug report.

What's Going On?

As a string, 2.2250738585072011e-308 causes no problems; it's when it's treated as a numeric value that the bug hits.

Where it breaks –
Attacks against ATMs

- Copy the disk with ‘dd’, mount the partition in a virtual machine – usually it does not check if it is running in the right hardware, so an adversary can set up a fake ATM, copy the software for reversing purposes, etc.
- Check if BIOS is password protected, full disk encryption is in place (and the key not in an unencrypted partition), secure boot is enabled

Where it breaks –
Attacks against ATMs

- Usually it has plenty of room inside the cabinet for an adversary to place its own malicious hardware (i.e., a Raspberry Pi)
- Sometimes all it takes is a registry key change to modify the money cartridges order and withdraw more money than the account is charged

War stories

Real life application security failures

A few examples of real-world vulnerabilities we have seen before in financial applications

War stories

- Non-cryptographically signed parameters make it easier to tamper with
- All account numbers were numeric and rather easy to predict
- Requests were similar to the one below:

```
POST /accounts/view_balance HTTP/1.1
Host: www.bank.com
Cookies: JSESSIONID=D258081D2030693CB1BD52992FFE6CFD; HttpOnly; secure
Content-Length: 20
Connection: keep-alive

accountNumber=123456
```

- Insecure object reference leading to the exposure of statements of ALL account holders
- Bank secrecy breach can lead to very high fines and loss of customer confidence
- It happened with one of the famous core banking systems we introduced earlier. In production.

War stories

- Fail #1: Again, all account numbers were numeric and predictable
- Fail #2: Side channels - attempting to logging in with a valid account ID but invalid password displayed a revealing message
- Fail #3: Strict account lock out policy of three failed passwords
- Fail 1 + 2 + 3 = recipe for disaster. Internet banking-wide denial of service
- It happened in real life ¯_(ツ)_/¯

War stories

- Input-validation looked pretty robust, but there was still hope…
- It was necessary to perform a transaction to someone, $0.01 would do the job
- On the description of the transaction, write a XSS payload
- The beneficiary is happy he received money, goes to check transaction details and…
- … script executed on client-side
- Friendly reminder: a stored XSS can read anti-CSRF tokens from the DOM
- Good return on investment: with $0.01 spent you could probably steal much more

War stories

- Foreign currency transfers
- Sometimes numbers are rounded up or down, depending on the result
- In theory, it was possible to gain one cent for each transaction; doing it multiple times and one could gain a lot more effectively having a 1 to 1 exchange rate.

Your project

Basic security check list for your university project

Tips on how to approach the “break it” part of the project

Your project –
Basic approach

- Break down the project in smaller components
- First do a security review of each component separately
- Second, see how they relate security-wise when integrated: in many cases, the magic happens in the boundaries

Your project – Internet banking

- Surprisingly, this seems to have a fairly reduced attack surface in the project
- Review the authentication mechanism for common issues like account enumeration, password brute force prevention (or lack there of), login bypasses, etc.
- If all you can do is to verify your balance, authorization checks have to work flawlessly
- Hardening the server and application: review for outdated software, weak password for databases, firewall rules, etc.

Your project –
Transfering funds

- Review the solution for logic issues
- A few things consider when reviewing:

✓ Is it possible to transfer a negative amount?

✓ Is it possible to change the source account where the transfer will come from?

✓ Is it possible to withdraw more than the funds available or go below the overdraft limit?

✓ How does the system handles concurrency/race condition situations?

- The choice of the database system plays a role in defending from race conditions (ACID property)

Your project –
RFID cards

- RFID have well known security weaknesses
- Can the tags be cloned easily?
- Are the tags easily rewritten?
- What sort of information is stored in the tags?
- Can a user replace the account number or another important value stored in the tag for another one?

Your project –
PIN pad

- Can the PIN be brute force-able? Does the PIN pad detect (and prevent) guessing attempts?
- PIN bypass attacks are feasible in EMV cards, how does it relate to your architecture?
- Is the PIN pad software/firmware vulnerable to memory corruption and other bugs exploitable from a specially-crafted RFID card?

References:

https://www.lightbluetouchpaper.org/2010/02/11/chip-and-pin-is-broken/

https://www.youtube.com/watch?v=Bw6Ah8RXcLg

Your project – Communications

- Man in the middle attacks are not uncommon in such setups
- If an adversary can tamper with data in transit, it better be encrypted and authenticated
- If an adversary can manipulate the memory of the program responsible for dispensing cash, for example, jackpotting may happen

Your project – Communications

- Ensure all communication links are encrypted and use protocols that support authenticated encryption and performs integrity checks
- Having an anti-debugging mechanisms may seem like an overkill but is not a bad idea

Your project –
ATM's physical security

- Enable secure boot
- Password protect the BIOS
- Full disk encryption, but have the key stored elsewhere like in a pendrive or password-derived

Other applications

Financial applications go beyond retail banking

Investment banking has its own software stack, protocols – and potential security problems, too.

Other applications

- Many trading platforms rely on the FIX protocol
- The standard document is huge, but barely touches security
- Speed is key, so trading systems do not enable encryption or integrity checks
- Very little published security research on the topic; here's an opportunity to be a pioneer

Thank you!

Julio Cesar Fort • julio@blazeinfosec.com

HTTPS://WWW.BLAZEINFOSEC.COM