How to Rob an Online Bank (and get away with it)

Mitja Kolsek
ACROS Security
WILLIE SUTTON (1901-80)

When asked why he robbed so many banks, he replied,
"Cause that's where all the money is!"
Evolution Of E-banking Attacks

PAST-PRESENT

PRESENT

FUTURE

FUTURE 2.0

Back-End Server

Online Banking Server
Attacks Against Individual Users

Goal: Identity Theft

Methods
- Phishing, Fake security alerts
- XSS, CSRF
- Malware (man in the browser, extraction of certs and private keys)

Problems
- User awareness
- 2-factor authentication
- OOB transaction confirmations
- Additional passwords/PINs
- “Known good” target accounts
Attacks Against Corporate Users

Goal: Identity Theft

Methods
- Same as with individual users

Problems
- More money
- Large transactions not unusual
- Targets can be found in public certificate directories
LDAP Explorer - Online Bank Robber’s Google
Example: Published Corporate Certificate

ldap://ldap.halcom.si:389/eidCertificateSerialNumber=382631

- E-Mail Address
- Personal Name
- Company Name
Attacks Against Online Banking Servers

Goal: Exploiting Applications

Methods
- Hacking

Problems
- Getting noticed while looking for flaws

Advantages
- Unlimited amount of money
- No user interaction (social engineering)
- Possible creation of new money
Direct Resource Access
Direct Resource Access - URL Cleartext ID

https://bank/balance?uid=7728356
(my account balance data)

https://bank/balance?uid=7728355
(another user’s account balance data)
Direct Resource Access - URL Base64 encoding

https://bank/balance?dWlkPTc3MjgzNTY=
(my account balance data)

Base64decode("dWlkPTc3MjgzNTY=")

"uid=7728356"

Base64encode("uid=7728355")

https://bank/balance?dWlkPTc3MjgzNTU=
(another user’s account balance data)
Direct Resource Access - URL Encryption

/balance?Ko7hIGJJ2GqfhSZ9... (Base64)
/balance?AF86B301008AEF5... (Hex)

params = "uid=7728356"
params += AES_encrypt(params, key)
params += "&salt=nobodywillguessthis"
params += "&rand=" + random()
params = "/balance?" + base64(params)
Taking Money From Other People’s Accounts

/transfer? src=1 & dest=2 & amount=100
(from my account)

/transfer? src=42 & dest=2 & amount=100
(from another user’s account)
Transaction Creation Process

1. I want to transfer some money

2. Empty transaction form

3. Filled-out transaction form
   src=1, dst=2, amount=100

4. Read-only confirmation form
   src=1, dst=2, amount=100

5. Transaction confirmation
   src=1, dst=2, amount=100
Negative Numbers
Negative Numbers - A Devastating Oversight

IF RequestedAmount > DisposableAmount THEN ERROR();

IF 3,000 > 2,000 THEN ERROR(); // Error - Insufficient Funds

IF -100 > 2,000 THEN ERROR(); // No Error Here
“Here’s minus hundred bucks for you”

Attacker: 0 $
Victim: 100 $

(Transfer -100 $ to Victim)

Attacker: 100 $
Victim: 0 $
Creating Money Out Of Thin Air

Normal Account: 0 $
Savings Account: 0 $

(Transfer -100 $ to Savings Account)

Normal Account: 100 $
Savings Account: 0 $
Bypassing Limit Checks
Normal Overdraft

Account #1: 100 $  
Account #2: 0 $  

(Transfer 1,000 $ from #1 to #2)  

Account #1: -900 $  
Account #2: 1,000 $
“Over-Overdraft”

Account #1: 100 $
Account #2: 0 $

(Transfer 1,000,000 $ from #1 to #2)

Account #1: -999,900 $
Account #2: 1,000,000 $
HTTP Parameter Pollution
Request Flow

POST /transfer
source=1 & dest=2 & amount=100

source = request.getParameter("source") // 1
amount = request.getParameter("amount") // 100
IF NOT user_authorized_for(source) THEN ERROR()
IF disposable(source) < amount THEN ERROR()
Call BackEndTransaction(request)

POST /BackEndTransaction
source=1 & dest=2 & amount=100

source = $_POST["source"] // 1
dest = $_POST["dest"] // 2
amount = $_POST["amount"] // 100
HTTP Parameter Pollution
(Source Account)

POST /transfer
source=1 & dest=2 & amount=100

source = request.getParameter("source")  // 1
amount = request.getParameter("amount")  // 100
IF NOT user_authorized_for(source) THEN ERROR()
IF disposable(source) < amount THEN ERROR()
Call BackEndTransaction(request)

POST /BackEndTransaction
source=1 & dest=2 & amount=100

source = $_POST["source"]  // 42
dest = $_POST["dest"]  // 2
amount = $_POST["amount"]  // 100
IF NOT user_authorized_for(source) THEN ERROR()
HTTP Parameter Pollution
(Transfer Amount)

POST /transfer
source=1 & dest=2 & amount=100 & amount=100000

source = request.getParameter("source") // 1
amount = request.getParameter("amount") // 100
IF NOT user_authorized_for(source) THEN ERROR()
IF disposable(source) < amount THEN ERROR()
Call BackEndTransaction(request)

POST /BackEndTransaction
source=1 & dest=2 & amount=100 & amount=100000

source = $_POST["source"] // 1
dest = $_POST["dest"] // 2
amount = $_POST["amount"] // 100000
IF NOT user_authorized_for(source) THEN ERROR()
SQL Injection
SQL Injection - Data Theft

"SELECT rate FROM exch_rates
WHERE currency = '" . $currency . ""

"SELECT rate FROM exch_rates
WHERE currency = " UNION
SELECT balance FROM accounts
WHERE account_id = '887296'"
SQL Injection - Messing With Transactions

”BEGIN TRANSACTION”

”UPDATE accounts SET balance = 0 WHERE account_id = ’’.\$acctid1.’’”

”UPDATE accounts SET balance = 100 WHERE account_id = ’’.\$acctid1.’’”

”COMMIT TRANSACTION”
SQL Injection - Messing With Transactions

”BEGIN TRANSACTION”

”UPDATE accounts SET balance = 0 WHERE account_id = ’123’”

”UPDATE accounts SET balance = 100 WHERE account_id = ’456’”

”COMMIT TRANSACTION”
SQL Injection - Messing With Transactions

”BEGIN TRANSACTION”

”UPDATE accounts SET balance = 0 WHERE account_id = ’123’”

”UPDATE accounts SET balance = 100 WHERE account_id = ’456’ OR account_id = ’123’”

”COMMIT TRANSACTION”
Forging Bank’s Digital Signatures
Automated Signing Of Deposit Agreement

Deposit request: 100€, 31 days

Deposit agreement for signing (Legal text, interest rate)

Signed MODIFIED deposit agreement

Counter-signed MODIFIED deposit agreement
Server-Side Code Execution
Server-Side Code Execution

Examples

- Java code injection (JBoss bug in 2010)
- PHP code injection (eval, system, includes...)
- Shell argument injection (command1&command2)
- Buffer overflows

Impact

- Change e-banking application code
- Obtain database/WS credentials, issue direct requests to DB or back-end WS
The List Goes On...
Other Attacks

- Session Puzzling
- Insecure Mass Assignment
- Numerical Magic: Overflows, Underflows, Exponential Notation, Reserved words (Corsaire whitepaper)
- “Stale” Currency Exchange
- Race Conditions
- ...

- New functionalities: automated deposits, loans, investment portfolio management, ...
Getting Rich Without Breaking The Law

http://blog.acrossecurity.com/2012/01/is-your-online-bank-vulnerable-to.html
### Rounding And Currency Exchange

<table>
<thead>
<tr>
<th>1 €</th>
<th>↔</th>
<th>1,364 $</th>
</tr>
</thead>
</table>

- **Loss:** $-0.00364 = -27\%
- **Profit:** €$+0.00266 = +36\%
Asymmetric Currency Rounding

by M’Raïhi, Naccache and Tunstall

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.91.8055&rep=rep1&type=pdf

Known at least since 2001
Currency Rounding Attack: Algorithm

1: Convert 100€ to $
   \text{// We have 136,40\$}$

2: for i = 1 to 13640
   Convert 0,01$ to 0,01€
   \text{// Now we have 136,40€}$

3: goto 1
Currency Rounding Attacks

The Speed Of Getting Rich

Assume: 10 exchanges / second
1 day = 86,400 seconds
Daily profit: 2,300 €
Monthly profit: ~ 70,000 €

Improvements

Optimal exchange rate (getting as close to 0,005 as possible)
Corporate banking: packets (1000s of exchanges in one packet)

Does it really work?

My personal e-banking: YES
My company’s corporate e-banking: YES

Countermeasures

Limit minimum amount to 1 whole unit, exchange fee
Phases Of Online Bank Robbery
4 Phases Of Online Bank Robbery

Phase 1  Vulnerability Finding
Source code or black box, “user-in-the-middle”

Phase 2  Vulnerability Exploitation
“Creation” of money on attacker’s account, “user-in-the-middle”

Phase 3  Buying Time
Passing the money across borders, borrowing corporate accounts

Phase 4  Extraction
Classic methods, also borrowing user accounts for transfers
Apply
Apply: Bankers - Short Term

- Currency conversion attacks
  - Set minimum amount or fee

- “Phase 1”: Disallow vulnerability tests
  - Terms of service: suspend user account if vuln-testing
Apply: Bankers - Long Term

- "Phase 1": Detect vulnerability tests
  - OWASP AppSensor*
  - Terminate user session if testing is detected
  - Many server-side errors caused by single user
- "Phase 2": Find and fix vulnerabilities
  - Review your code (tools don’t find logical bugs)
  - Get help from those who break into banks
  - Use external pentest for designing and testing "Phase 1" detection

* https://www.owasp.org/index.php/OWASP_AppSensor_Project
Apply: Penetration Testers

- **Staging != Production**
  - Authentication, back-end processing, scheduled jobs
  - Find bugs in staging, verify in production
  - Successful tests in production may be hard to undo (therefore test with small amounts)

- **Vulnerabilities**
  - Use HTTP proxy* to “eliminate” client user interface
  - Be really thorough with negative numbers
  - Base64 encoding is often mere obfuscation
  - Banks are focused on authentication, not authorization

* [https://www.owasp.org/index.php/Phoenix/Tools#HTTP_proxying_.2F_editing](https://www.owasp.org/index.php/Phoenix/Tools#HTTP_proxying_.2F_editing)
Mitja Kolsek
ACROS d.o.o.
www.acrossecurity.com
mitja.kolsek@acrossecurity.com
Twitter: @acrossecurity, @mkolsek

Thanks: Mikko H. Hypponen, René Pfeiffer, Claudio Criscione, Stefan Ortloff and Candi Carrera for help with gathering information on national digital certificate usage