



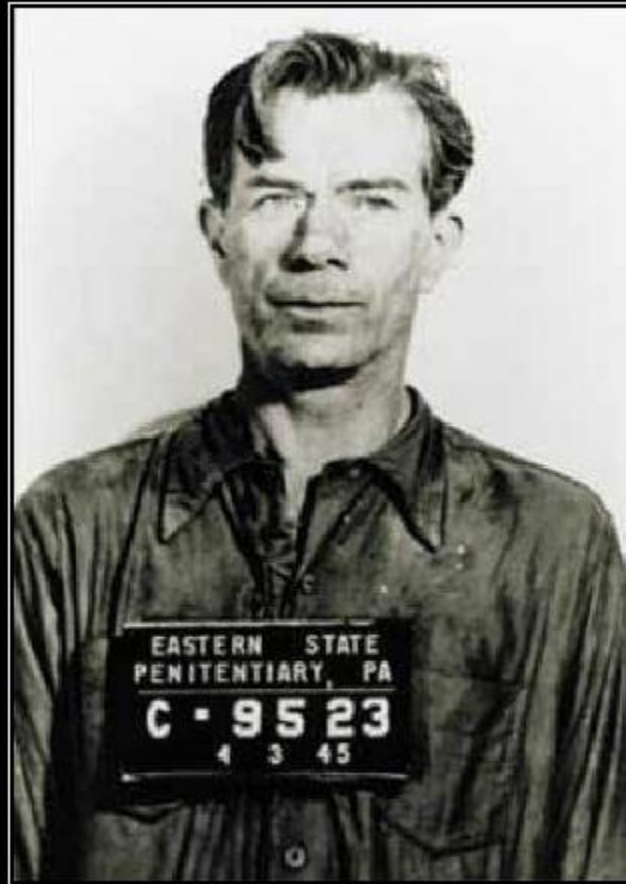
# How to Rob an Online Bank (and get away with it)

**Mitja Kolsek**  
ACROS Security

Session ID: HT-108

Session Classification: Intermediate

**RSACONFERENCE**  
EUROPE 2012



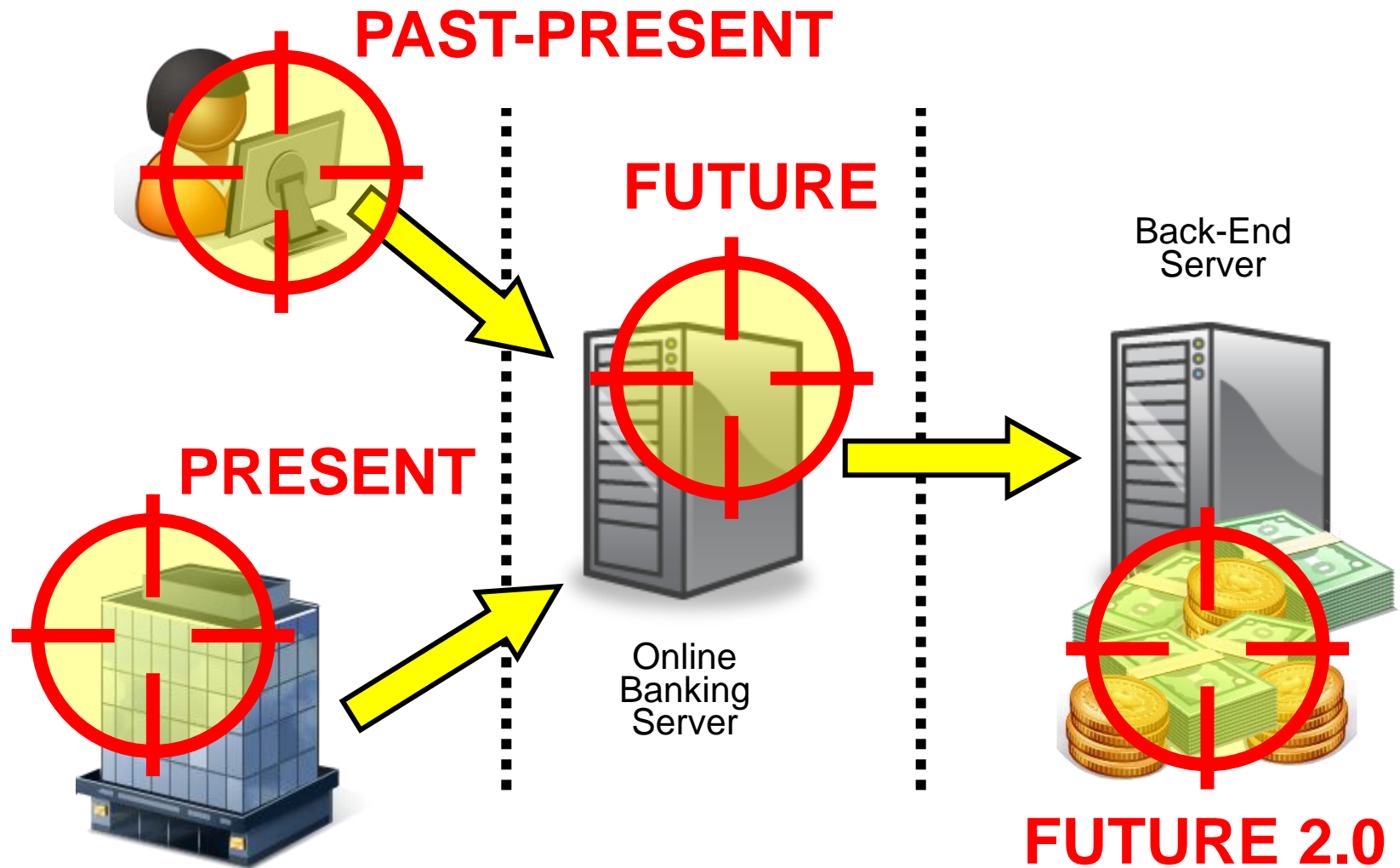
## WILLIE SUTTON (1901-80)

When asked why he robbed so many banks, he replied,  
" 'Cause that's where all the money is "





# Evolution Of E-banking Attacks



# 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

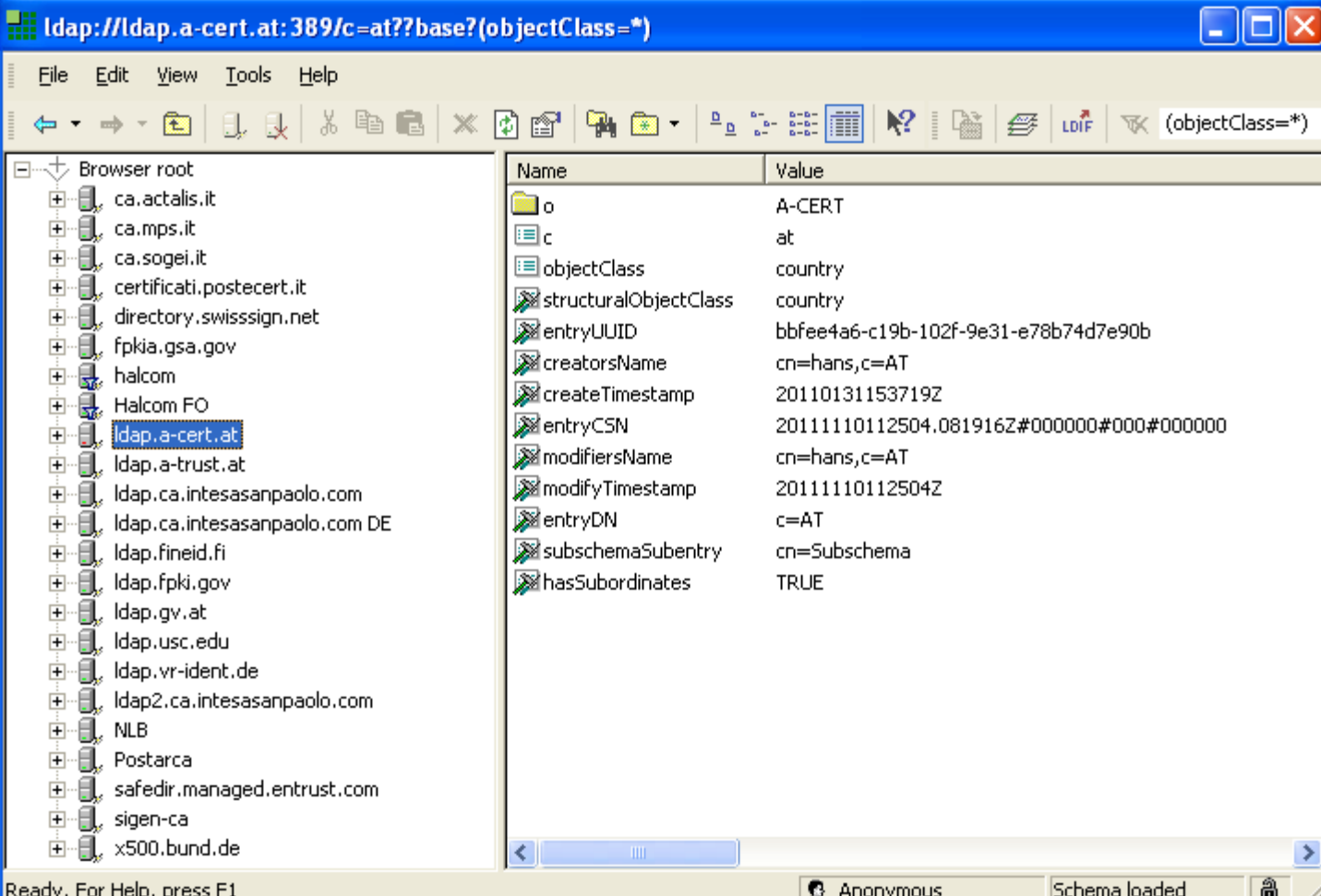
- Same as with individual users

### Advantages

- More money
- Large transactions not unusual
- Targets can be found in public certificate directories



# LDAP Explorer - Online Bank Robber's Google



LDAP Explorer window showing the directory tree on the left and the entry details on the right. The entry details are as follows:

Name	Value
o	A-CERT
c	at
objectClass	country
structuralObjectClass	country
entryUUID	bbfee4a6-c19b-102f-9e31-e78b74d7e90b
creatorsName	cn=hans,c=AT
createTimestamp	20110131153719Z
entryCSN	20111110112504.081916Z#000000#000#000000
modifiersName	cn=hans,c=AT
modifyTimestamp	20111110112504Z
entryDN	c=AT
subschemaSubentry	cn=Subschema
hasSubordinates	TRUE



# Example: Published Corporate Certificate

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

E-Mail Address  
Personal Name  
Company Name

The screenshot shows a Windows dialog box titled "Certificate(s) [eidCertificateSerialNumber=382631,cn=Halcom ...]". The "Certificate" tab is active, displaying "X.509 Certificate Information". A table lists the certificate's fields and values:

Field	Value
Serial number	3826 31(0 X5D6 A7)
Signature	sha1WithRSAEncryption
Issuer	CN=Halcom CA PO 20=HalcomC=SI
Valid from	Apr 22 09:45:49 2010 GMT
Valid until	Apr 22 09:45:49 2013 GMT
Subject	1.3.6.1.4.1.5939.2.2 = #13083939...
Public key	rsaEncryption (1024 bits)

Below the table, the certificate's subject details are shown in a text area:

```
1.3.6.1.4.1.5939.2.2 = #13083939343435323830  
emailAddress = matjaz.cadez@halcom.si  
1.3.6.1.4.1.5939.2.3 = #1308343333353313236  
GN = Matjaz  
SN = Cadez  
CN = Matjaz Cadez  
O = HALCOM D.D.  
C = SI
```

Red boxes highlight the email address, personal name, and company name fields in the subject details. Red lines connect these fields to the labels on the left. The dialog box includes "Save Info", "OK", "Cancel", and "Help" buttons.





# Attacks Against Online Banking Servers



Online  
Banking  
Server

## 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)



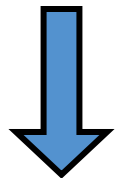
# Direct Resource Access - URL Base64 encoding

`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 += "&salt=nobodywillguessthis"
```

```
params += "&rand=" + random()
```

```
enc_params = AES_encrypt(params, key)
```

```
path = "/balance?" + base64(enc_params)
```



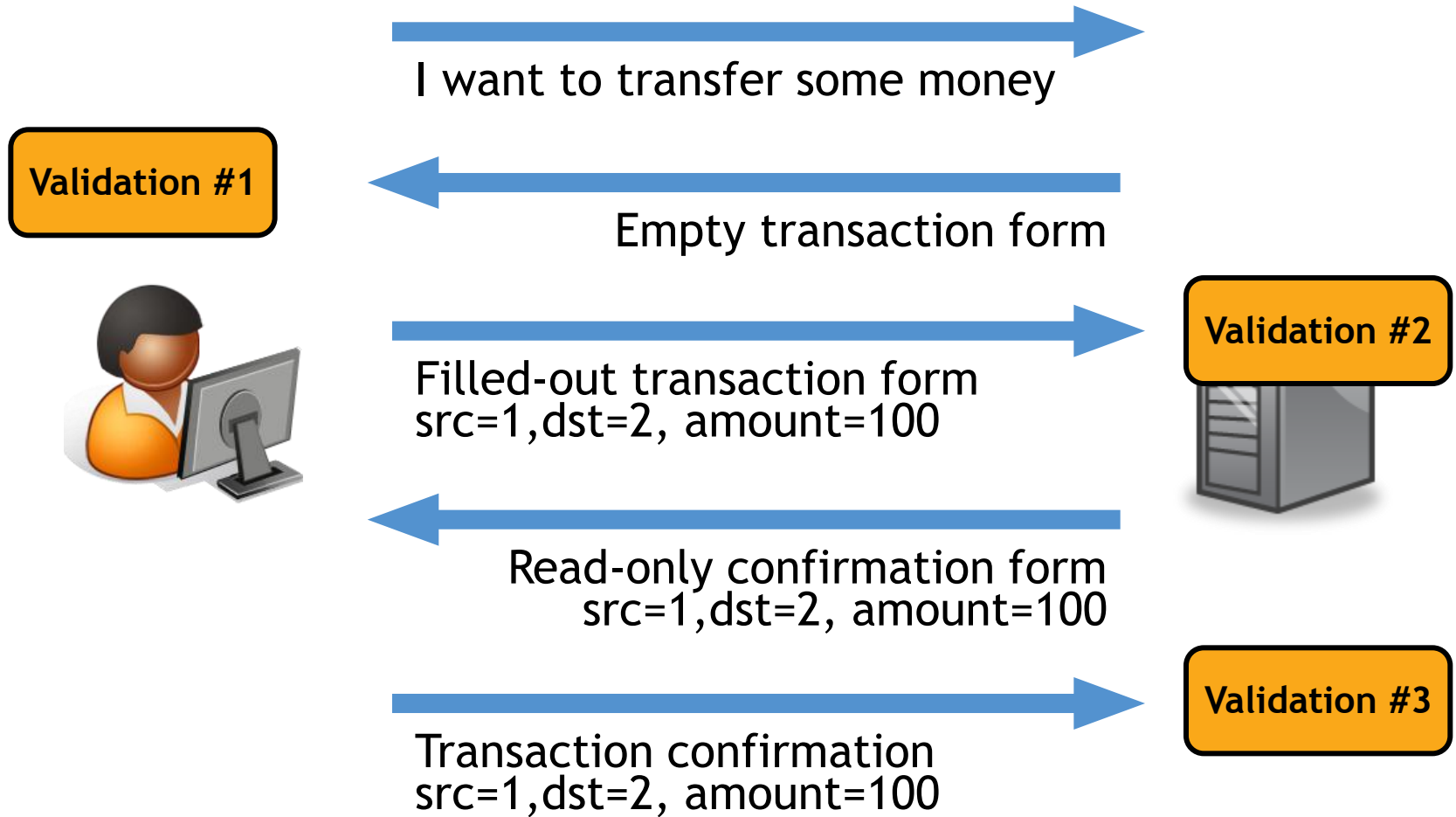
# Taking Money From Other People's Accounts

/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



# Negative Numbers





# Negative Numbers - A Devastating Oversight

```
IF RequestedAmount > DisposableAmount  
THEN ERROR();
```

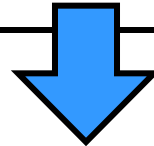
```
IF 3,000 > 2,000  
THEN ERROR(); // Error - Insufficient Funds
```

```
IF -100 > 2,000  
THEN ERROR(); // No Error Here
```

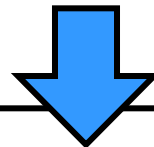


# “Here’s minus hundred bucks for you”

Attacker:	0 \$
Victim:	100 \$



(Transfer -100 \$ to Victim)

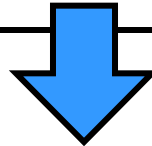


Attacker:	100 \$
Victim:	0 \$

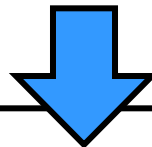


# Creating Money Out Of Thin Air

Normal Account:	0 \$
Savings Account:	0 \$



(Transfer -100 \$ to Savings Account)



Normal Account:	100 \$
Savings Account:	0 \$

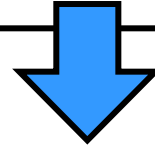


# Bypassing Limit Checks



# Normal Overdraft

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



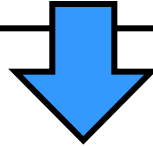
(Transfer 1,000 \$ from #1 to #2)



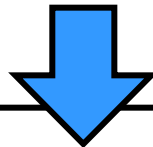
Account #1:	-900 \$
Account #2:	1,000 \$

# “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

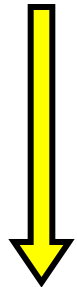


POST /transfer  
source=1 & dest=2 & amount=100

JSP



```
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

PHP



```
source = $_POST["source"] // 1  
dest = $_POST["dest"] // 2  
amount = $_POST["amount"] // 100
```







# HTTP Parameter Pollution

(Source Account)

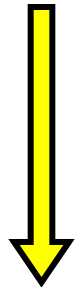


POST /transfer  
source=1 & dest=2 & amount=100

JSP



```
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

PHP



```
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

JSP



```
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

PHP



```
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 = '". $acctid2. "'"
```

```
"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”

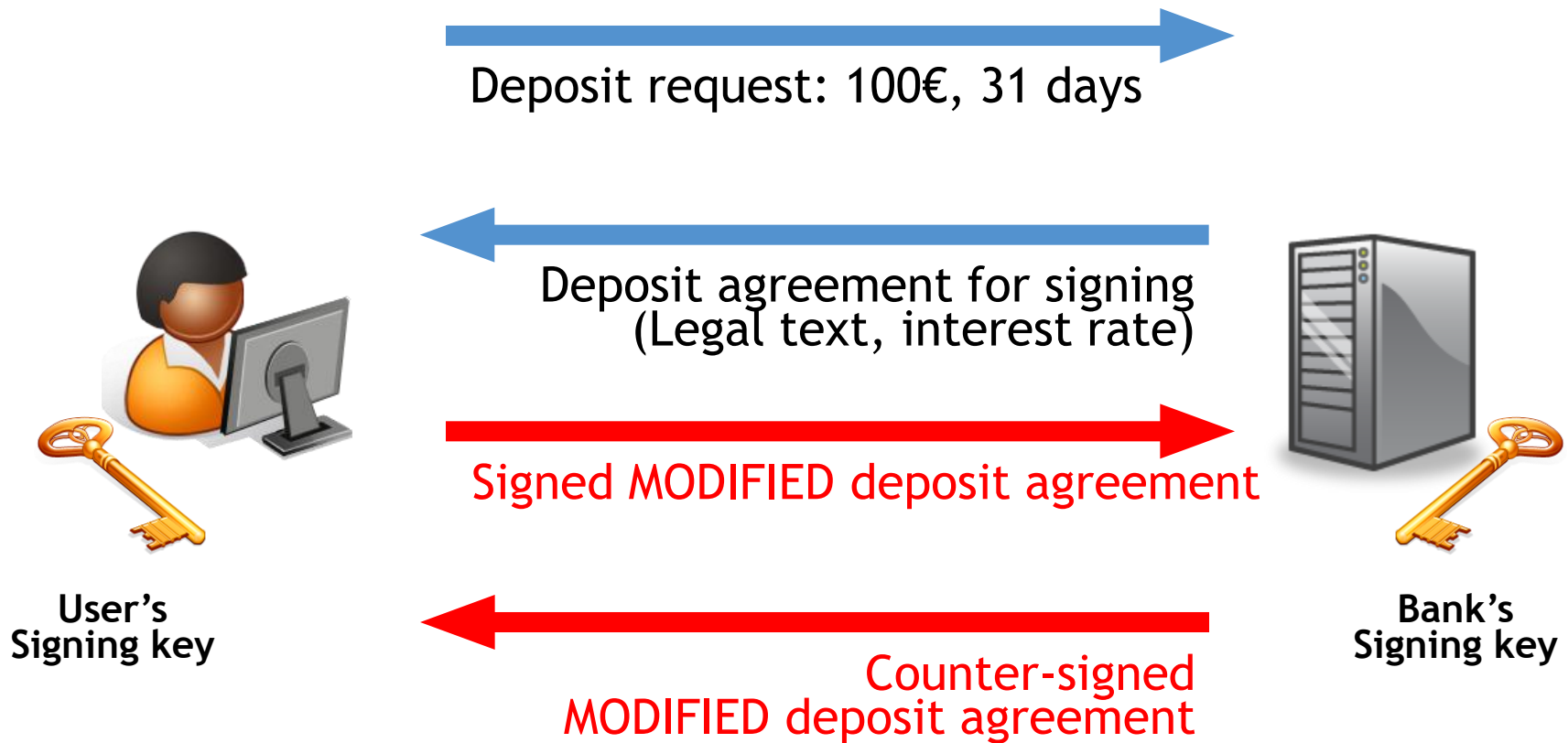


# Forging Bank's Digital Signatures





# Automated Signing Of Deposit Agreement





# 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



# Rounding And Currency Exchange

1 € ↔ 1,364 \$

0,01 € → 0,01 \$

Loss :  $-0,00364 \$ = -27\%$

0,01 € ← 0,01 \$

Profit :  $+0,00266 € = +36\%$



# Asymmetric Currency Rounding

David M'Raihi<sup>1</sup>, David Naccache<sup>2</sup>, and Michael Tunstall<sup>3</sup>  
<sup>1</sup> Gemplus  
3 Lagoon Drive, Suite 300, Redwood City, CA 94065, USA  
david.mraihi@gemplus.com

<sup>2</sup> Gemplus  
34 rue Guynemer, Issy-les-Moulineaux, F-92447, France  
naccache@gemplus.com

<sup>3</sup> Gemplus, Card Security Group  
B.P. 100, Côtémas, F-13851, France  
tunstall@gemplus.com

**Abstract.** The euro was introduced on the first of January 1999 as a common currency in fourteen European nations. EC regulations are fundamentally different from usual banking practices for they forbid fees when converting national currencies to euros (fees would otherwise deter users from adopting the euro); this creates a unique fraud context where money can be made by taking advantage of the EC's official rounding rules. This paper proposes a public-key-based protection against such attacks. In our scheme, the parties conducting a transaction can not predict whether the rounding will cause loss or gain while the expected statistical difference between an amount and its euro-equivalent decreases exponentially as the number of transactions increases.

## 1 Introduction

Economic and Monetary Union (in short EMU) is a further step in the ongoing process of European integration. EMU will create an area whose economic potential will sustain comparison to that of the United States. Given the size of the euro area, the euro is expected to play an important role as an international currency. As a trade invoicing currency, the euro will also extend its role beyond direct trade relations. Issues related to euro conversion were therefore precisely defined in the general framework of the European financial market and issued [1]. The main objective of the conversion rules for currencies inside the euro area is to ensure that conversions with a comprehensive set of rules are paid while...

Known  
at least  
since  
**2001**

Asymmetric Currency Rounding  
by M'Raihi, Naccache and Tunstall

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

...related to currency  
...the constraint imposed by the  
... (a political requirement) opens





# Currency Rounding Attack: Algorithm

- 1: Convert 100€ to \$  
// We have 136,40\$
- 2: for i = 1 to 13640  
    Convert 0,01\$ to 0,01€  
    // 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](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](http://www.acrossecurity.com)

[mitja.kolsek@acrossecurity.com](mailto: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

