
LIVE

Języki programowania

VS

Security!

Gynvael Coldwind

MEGA Sekurak Hacking Party, Kraków, 2019

Jeśli przeglądasz te slajdy, to rzuć okiem na "speakers notes" - jest tam czasem kilka dodatkowych informacji.

whoami

Tech Lead / Manager @ Google ISE-RIP

All opinions expressed during this presentation are mine and mine alone, and not those of my barber, my accountant or my employer.

W poprzednim odcinku

Slajdy / Video
https://gynvael.coldwind.pl/?id=693

Jadłospis

- Wstęp i secure by default
- Anty-przykłady (nie tylko języki programowania)
- Koniec

Podziękowania dla wielu mądrych ludzi z którymi miałem okazję porozmawiać przez lata :)

Full disclosure: Podczas tej prelekcji korzystam z wybranych slajdów w innych moich prelekcji

Dwa problemy

w bezpieczeństwie aplikacji

Dwa problemy w bezpieczeństwie aplikacji

Istniejące aplikacje

Nowo tworzone aplikacje

Nawet najlepsi programiści to tylko ludzie

2017-08-14: FAQ: How to find vulnerabilities?

faq

Obligatory FAQ note: Sometimes I get asked questions, e.g. on IRC, via e-mail or during my livestreams. And sometimes I get asked the same question repeatedly. To save myself some time (*cough* and be able to give the same answer instead of conflicting ones *cough*) I decided to write up selected question and answer pairs in separate blog posts. Please remember that these answers are by no means authoritative - they are limited by my experience, my knowledge and my opinions on things. Do look in the comment section as well - a lot of smart people read my blog and might have a different, and likely better, answer to the same question. If you disagree or just have something to add - by all means, please do comment.

Q: How does one find vulnerabilities?
A: I'll start by noting that this question is quite high-level - e.g. it doesn't reveal the technology of interest. More importantly, it's not clear whether we're discussing a system vulnerability (i.e. a configuration weakness or a known-but-unpatched bug in an installed service) that one usually looks for during a regular network-wide pentest, or if it's about discovering a previously unknown vulnerability in a an application, service, driver / kernel module, operating system, firmware, etc. Given that I'm more into vulnerability research than penetration testing I'll assume it's the latter. And also, the answer will be as high-level as the question, but should give one a general idea.

My personal pet theory is that there are three* main groups of methods (I'll go in more details below):
* If I missed anything, please let me know in the comments; as said, it's just a pet theory (or actually a pet hypothesis).

1. Code review (this also includes code that had to be reverse-engineered).
2. Black box (this includes using automated tools like scanners, fuzzers, etc).
3. Documentation research.

All of the above methods have a set of requirements and limitations, and are good at one thing or the other. There is no "best method" that always works - it's more target specific I would say. Usually a combination of the above methods is used during a review of a target anyway.

Return to dashboard ⇧

Sections
lang: |
RSS: |

About me
Tools

→ YT YouTube (EN)
→ T Twitter
→ GH GitHub

Links / Blogs
→ dragonsector.pl
→ vexillium.org
Security/Hacking:
j00ru's blog
lcamtuf's blog
invisible things (new)

Nawet najlepsi programiści to tylko ludzie

Q: How does one find vulnerabilities?

...

1. Code review (this also includes code that had to be reverse-engineered).
2. Black box (this includes using automated tools like scanners, fuzzers, etc).
3. **Documentation research.**

Nawet najlepsi programiści to tylko ludzie

3. Documentation research

...

The general idea is to go through the documentation / specification / reference manual and **pinpoint places where the programmer implementing the target might understand something in an incorrect way**.

Nawet najlepsi programiści to tylko ludzie

Programiści popełniają błędy

Nawet najlepsi programiści to tylko ludzie

Programiści popełniają błędy

Dlaczego?

Nawet najlepsi programiści to tylko ludzie

Programiści popełniają błędy

↓

Dlaczego?

- Zapomnieli/nie wiedzieli:
  - że trzeba coś sprawdzić (walidacja).
  - że trzeba coś skonwertować na inną formę (sanitize/escape).
  - że dana operacja może w skrajnych wypadkach mieć efekty uboczne.
- Mieli słabszy dzień / byli zmęczeni / deadline.

Nawet najlepsi programiści to tylko ludzie

Programiści muszą pamiętać / wiedzieć o tych detalach.

Tj. muszą wykonać dodatkową pracę, aby stworzyć bezpieczny kod.

Nawet najlepsi programiści to tylko ludzie

W świecie idealnie kulistych kurczaków
należy wykonać dodatkową pracę,
aby popełnić błąd.

Czyli secure by default w oprogramowaniu.

Kilka losowych efektów

Ekonomia języka:
Krótsza forma jest preferowana względem dłużej formy.

"I have a request" → "I have an ask"
"I don't know" → "idk"
"That's funny" → "lol xD"

Ekonomia języka w programowaniu:

```
html = secure_escape_mb_string_(user_input)
```

vs

```
html = user_input
```

Kilka losowych efektów

Pierwsze Prawo Murphiego:
Jeżeli coś może się nie udać – nie uda się na pewno.

W programowaniu:
Jeżeli do wyboru są dwa interfejsy - bezpieczny i niebezpieczny, na pewno zostanie wybrany ten drugi.
(OK, może "na pewno", ale... zasada najsłabszego ogniwa)

```
secret = rand()
```
vs
```
secret = csprng()
```

Kilka losowych efektów

POLA - Principle of Least Astonishment

Zachowanie danego interfejsu nie może być zaskakujące dla użytkownika.

W programowaniu:

API powinno robić to, czego programista się spodziewa.

BMP

Anty-przykłady

(Tj. będę narzekać na losowe rzeczy)

Operatory porównania

Anty-przykłady

(Tj. będę narzekać na losowe rzeczy)

== w PHP, czyli pseudo-losowe porównanie

== w PHP, czyli pseudo-losowe porównanie

```
"asdf" == "xyz"
"+1" == "0.1e1"
"-0e10" == "0"
"1000" == "0x3e8"
"123" == "\t\r\n 123"
```

== w PHP, czyli pseudo-losowe porównanie

```
"asdf" == "xyz"                    false
"+1" == "0.1e1"
"-0e10" == "0"
"1000" == "0x3e8"
"123" == "\t\r\n 123"
```

== w PHP, czyli pseudo-losowe porównanie

```
"asdf" == "xyz"          false
"+1" == "0.1e1"          true
"-0e10" == "0"
"1000" == "0x3e8"
"123" == "\t\r\n 123"
```

== w PHP, czyli pseudo-losowe porównanie

```
"asdf" == "xyz"          false
"+1" == "0.1e1"          true
"-0e10" == "0"           true
"1000" == "0x3e8"
"123" == "\t\r\n 123"
```

== w PHP, czyli pseudo-losowe porównanie

```
"asdf" == "xyz"              false
"+1" == "0.1e1"              true
"-0e10" == "0"               true
"1000" == "0x3e8"            true
"123" == "\t\r\n 123"
```

== w PHP, czyli pseudo-losowe porównanie

```
"asdf" == "xyz"          false
"+1" == "0.1e1"          true
"-0e10" == "0"           true
"1000" == "0x3e8"        true
"123" == "\t\r\n 123"    true
```

== w PHP i wszystkich ulubiony trick z hashami (aka magic hashes)

```
md5('240610708') == md5('QNKCDZO')      true
```

== w PHP i wszystkich ulubiony trick z hashami (aka magic hashes)

```
md5('240610708') == md5('QNKCDZO')      true
```

```
md5('240610708')
0e462097431906509019562988736854
```

```
md5('QNKCDZO')
0e830400451993494058024219903391
```

== w PHP, czyli pseudo-losowe porównanie

```
"12345123451234512345123455"
==
"12345123451234512345123456"
```

== w PHP, czyli pseudo-losowe porównanie

```
"12345123451234512345123455"
==                                  false
"12345123451234512345123456"
```

== w PHP (stare wersje - 5.4.X)

```
"12345123451234512345123455"
==                                  true
"12345123451234512345123456"
```

== w PHP, czyli pseudo-losowe porównanie

```
0 == "PHP jest najlepsze!"
1337 == "1337 to fajna liczba"

9223372036854775807
==
"9223372036854775808"
```

== w PHP, czyli pseudo-losowe porównanie

```
0 == "PHP jest najlepsze!"          true
1337 == "1337 to fajna liczba"

9223372036854775807
==
"9223372036854775808"
```

== w PHP, czyli pseudo-losowe porównanie

```
0 == "PHP jest najlepsze!"          true
1337 == "1337 to fajna liczba"      true

9223372036854775807
==
"9223372036854775808"
```

== w PHP, czyli pseudo-losowe porównanie

```
0 == "PHP jest najlepsze!"          true

1337 == "1337 to fajna liczba"      true

9223372036854775807
==                                  true
"9223372036854775808"
```

== w PHP, czyli pseudo-losowe porównanie

```
$r = new \PDO("sqlite::memory:");
$r = $r->query(
    "SELECT sqlite_version()");
$r = $r->fetch(\PDO::FETCH_LAZY);
// PDORow Object

$r == "1"
```

== w PHP, czyli pseudo-losowe porównanie

```
$r = new \PDO("sqlite::memory:");
$r = $r->query(
    "SELECT sqlite_version()");
$r = $r->fetch(\PDO::FETCH_LAZY);
// PDORow Object

$r == "1"                                true
```

Przechodniość w PHP

```
"00" == "0"
"0" == false

"00" == false
```

Przechodniość w PHP

```
"00" == "0"      true
"0" == false

"00" == false
```

Przechodniość w PHP

```
"00" == "0"      true
"0" == false     true

"00" == false
```

Przechodniość w PHP

```
"00" == "0"      true
"0" == false     true

"00" == false    false
```

Przechodniość w PHP

```
NULL == false
false == "0"

NULL == "0"
```

Przechodniość w PHP

```
NULL == false    true
false == "0"

NULL == "0"
```

Przechodniość w PHP

```
NULL == false    true
false == "0"     true

NULL == "0"
```

Przechodniość w PHP

```
NULL == false    true
false == "0"     true

NULL == "0"      false
```

Implicit vs explicit cast w PHP ==

```
1234 == "1234 asdf"

(string)1234 == "1234 asdf"
```

Implicit vs explicit cast w PHP ==

```
1234 == "1234 asdf"                    true

(string)1234 == "1234 asdf"
```

Implicit vs explicit cast w PHP ==

```
1234 == "1234 asdf"              true

(string)1234 == "1234 asdf"    false
```

Implicit vs explicit cast w PHP ==

```
[] == "Array"

(string)[] == "Array"
```

Implicit vs explicit cast w PHP ==

```
[] == "Array"                    false

(string)[] == "Array"
```

Implicit vs explicit cast w PHP ==

```
[] == "Array"                    false

(string)[] == "Array"            true
```

Zrozumieć == w PHP

Źródło: https://gynvael.coldwind.pl/?id=492

Rejoice! W JavaScript też mamy == i ===!

Rejoice! W JavaScript też mamy == i ===!

```
9223372036854775807
==
"92233720368547758 08"
```

W JavaScript wszystko* jest floatem. BASIC called...

* Pomijam new Uint8Array etc

Rejoice! W JavaScript też mamy == i ===!

```
9223372036854775807
==                                    true
"92233720368547758 08"
```

W JavaScript wszystko* jest floatem. BASIC called...

* Pomijam new Uint8Array etc

Rejoice! W JavaScript też mamy == i ===!

```
[] == ""
[1,3,3,7] == "1,3,3,7"
[1,2] == [1,2]
```

Rejoice! W JavaScript też mamy == i ===!

```
[] == ""                    true
[1,3,3,7] == "1,3,3,7"
[1,2] == [1,2]
```

Rejoice! W JavaScript też mamy == i ===!

```
[] == ""                    true
[1,3,3,7] == "1,3,3,7"      true
[1,2] == [1,2]
```

Rejoice! W JavaScript też mamy == i ===!

```
[] == ""                    true
[1,3,3,7] == "1,3,3,7"      true
[1,2] == [1,2]              false
```

Podsumowując "miękkie porównanie" ==

W zasadzie powinno się zawsze używać ===.

Ekonomia języka:

Można napisać jedno = mniej!!11one

0 i null w JavaScript (podziękowania dla Nasty)

```
0 == null
0 > null
0 < null
0 >= null
0 <= null
```

0 i null w JavaScript (podziękowania dla Nasty)

```
0 == null                    false
0 > null
0 < null
0 >= null
0 <= null
```

0 i null w JavaScript (podziękowania dla Nasty)

```
0 == null                    false
0 > null                     false
0 < null
0 >= null
0 <= null
```

0 i null w JavaScript (podziękowania dla Nasty)

```
0 == null          false
0 > null           false
0 < null           false
0 >= null
0 <= null
```

0 i null w JavaScript (podziękowania dla Nasty)

```
0 == null          false
0 > null           false
0 < null           false
0 >= null          true
0 <= null
```

0 i null w JavaScript (podziękowania dla Nasty)

```
0 == null     false
0 > null      false
0 < null      false
0 >= null     true
0 <= null     true
```

Więcej o == i === w JS

Ten wątek na Stack Overflow:

https://stackoverflow.com/questions/359494/which-equals-operator-vs-should-be-used-in-javascript-comparisons

I koniecznie:

https://www.slideshare.net/kkotowicz/biting-into-the-forbidden-fruit-lessons-from-trusting-javascript-crypto

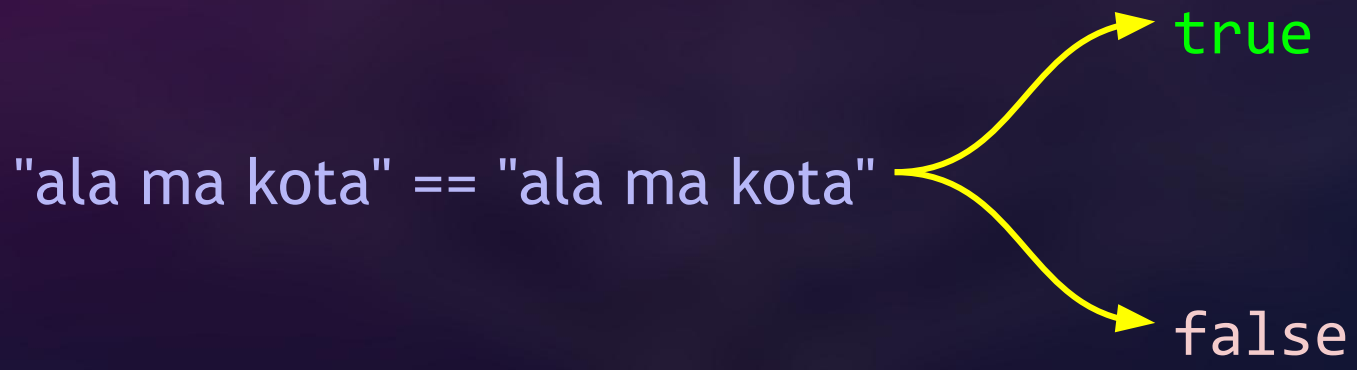
(Krzysztof Kotowicz)

Dobrze, że == w C i C++ są idealne

"ala ma kota" == "ala ma kota"

Dobrze, że == w C i C++ są idealne

Porównanie pointerów



Dobrze, że == w C i C++ są idealne

w C trzeba użyć `strcmp`

```
strcmp("asdf", "asdf")
```

Dobrze, że == w C i C++ są idealne

w C trzeba użyć `strcmp`

```
strcmp("asdf", "asdf")          0 (false)
```

Dobrze, że == w C i C++ są idealne

w C trzeba użyć `strcmp`

(ma to niby swoje uzasadnienie, ale...)

```
!strcmp("asdf", "asdf")          1 (true)
```

Dobrze, że == w C i C++ są idealne

w "nowym" C++ można tak

(ekonomia języka)

```
"ala ma kota"s == "ala ma kota"s                true
```

Dobrze, że == w C i C++ są idealne

true / false?

```
int main(void) {
  char x = 0xff;

  if(x == 0xff)
    puts("true");
  else
    puts("false");
}
```

Źródło: https://gynvael.coldwind.pl/n/char_comparison_riddle

Dobrze, że == w C i C++ są idealne

```
int main(void) {
  char x = 0xff;

  if(x == 0xff)
    puts("true");
  else
    puts("false");
}
```

true / false

char → signed char

x = (int)255;
x → (char)-1
(int)(char)x == (int)255
-1 == 255

Źródło: https://gynvael.coldwind.pl/n/char_comparison_riddle

Dobrze, że == w C i C++ są idealne

```c
int main(void) {
  char x = 0xff;

  if(x == 0xff)
    puts("true");
  else
    puts("false");
}
```

true / false

char → unsigned char

x = (int)255;

x → (char)255

(int)(char)x == (int)255

255 == 255

Źródło: https://gynvael.coldwind.pl/n/char_comparison_riddle

Protokoły sieciowe

Anty-przykłady

(Tj. będę narzekać na losowe rzeczy)

TCP i sumy kontrolne

TCP i sumy kontrolne

TCP provides reliable, ordered, **and error-checked delivery of a stream of octets (bytes)** between applications running on hosts communicating via an IP network.

Źródło: https://en.wikipedia.org/wiki/Transmission_Control_Protocol

TCP i sumy kontrolne

Losowa historia (4p):

Byłem sobie w LV i próbowałem ściągnąć aplikację.

TCP i sumy kontrolne

Losowa historia (4p):

Byłem sobie w LV i próbowałem ściągnąć aplikację.

- 300 MB
- 1 godzina ściągania
- bardzo duży retransmission rate

TCP i sumy kontrolne

Losowa historia (4p):

Byłem sobie w LV i próbowałem ściągnąć aplikację.

- 300 MB
- 1 godzina ściągania
- bardzo duży retransmission rate

Dla porównania na domowym serwerze:

- 12 sekund ściągania

TCP i sumy kontrolne

Losowa historia (4p):

Sprawdzenie podpisu cyfrowego po ściągnięciu applikacji:

FAIL

O nie! ktoś mnie próbuje shaczyć!

TCP i sumy kontrolne

Losowa historia (4p):

Po ściągnięciu na zaufanym serwerze i porównaniu* bloków po 4KB:

- 400 bloków uszkodzonych
- wszystkie w sekcji .rsrc
  ("ktoś Ci chciał ikonkę podmienić :)")

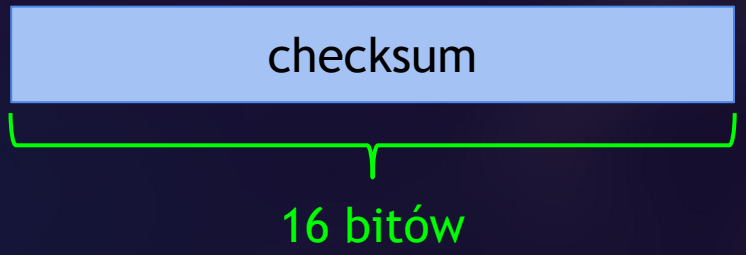
TCP i sumy kontrolne

Losowa historia (4p):

TCP i sumy kontrolne

Losowa historia (4p):

P = 1 / 65536 = 0.00001526 ← prawdopodobieństwo kolizji



TCP i sumy kontrolne

Losowa historia (4p):

P = 1 / 65536 = 0.00001526 ← prawdopodobieństwo kolizji

Przy 300MB danych, ściąganych przez 1h przy dużych zakłóceniach, nie ma wyjścia - Kolizje Muszą Się Zdarzyć.

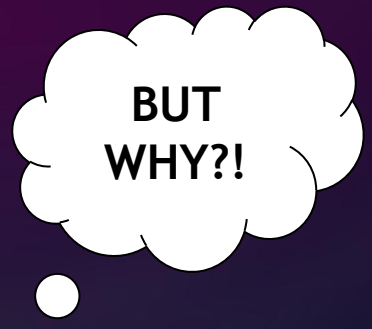
TCP i sumy kontrolne

Losowa historia (4p):
Ostatecznie zastąpiłem uszkodzone bloki poprawnymi.
(tym razem przesłane po SSH; min. 96-bitów MAC)

A gdyby tak TCP miał jakiś odpowiednio większy checksum...
Np. 256-bitów? (2019 vs 1974)

TCP i strumień bajtów

TCP i strumień bajtów (pseudo net stack)



Aplikacje: **Pakiety**

TCP: **Strumienie**

Niskopoziomowy TCP: **Pakiety**

IP: **Pakiety**

Ethernet: **Pakiety**

TCP i strumień bajtów

TCP: **Strumienie**

Wolny internet

- Progressive JPEG
- Adam7 w PNG

Play-when-loading

- WAV może? (bo MP3 nie)

Krok 1

Krok 2

Krok 3

Krok 4

Krok 5

Krok 6

Krok 7

Pakiety

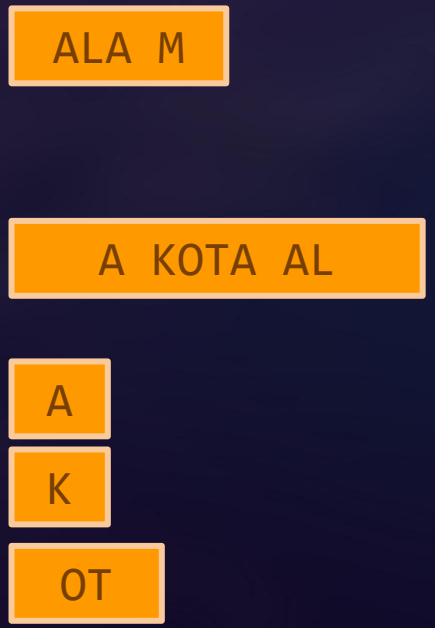
Co jest wysyłane:

Co aplikacja otrzymuje:

Strumienie

Co jest wysyłane:

Co aplikacja otrzymuje:



Strumienie (jeśli mamy pecha podczas testów)

Co jest wysyłane:

Co aplikacja otrzymuje:

Strumienie (albo)

Co jest wysyłane:

Co aplikacja otrzymuje:

ALA MA KOTA ALA KOT

Strumienie a fuzzing

Oprócz fuzzowania danych,
można też fuzzować
timing / rozbijanie / łączenie "pakietów" TCP

TCP i strumienie

Zdecydowana większość aplikacji i tak implementuje pakiety na TCP:

- Albo w formie <długość> <dane>
- Albo delimiterów (np. Linia 1\r\nLinia 2\r\n)

Przydałby się taki pakietowy TCP...

(Teraz i tak bez znaczenia, bo wszyscy poprawnie używają SSL)

Ucieczka z Nowego Jorku (kinda)

Anty-przykłady

(Tj. będę narzekać na losowe rzeczy)

Dopasuj funkcje do kontekstu (PHP)

SQL
SELECT ... WHERE
x='input'

HTML
<tag>input</tag>

HTML
<tag attrib="input">

HTML CSS
<style>
p { color: input }
</style>

HTML JS
<script>
const X='input';
</script>

JSON
{ "name": "input" }

Dopasuj funkcje do kontekstu (PHP)

`addslashes`

SQL
SELECT ... WHERE
x='input'

HTML
<tag>input</tag>

HTML
<tag attrib="input">

HTML CSS
<style>
p { color: input }
</style>

HTML JS
<script>
const X='input';
</script>

JSON
{ "name": "input" }

Dopasuj funkcje do kontekstu (PHP)

SQL
SELECT ... WHERE
x='input'

HTML
<tag>input</tag>

HTML
<tag attrib="input">

HTML CSS
<style>
p { color: input }
</style>

HTML JS
<script>
const X='input';
</script>

JSON
{ "name": "input" }

addslashes

Dopasuj funkcje do kontekstu (PHP)

`mysql_real_escape_string`

SQL
SELECT ... WHERE
x='input'

HTML
<tag>input</tag>

HTML
<tag attrib="input">

HTML CSS
<style>
p { color: input }
</style>

HTML JS
<script>
const X='input';
</script>

JSON
{ "name": "input" }

addslashes

Dopasuj funkcje do kontekstu (PHP)

SQL
SELECT ... WHERE
x='input'

HTML
<tag>input</tag>

mysql_real_escape_string

HTML
<tag attrib="input">

HTML CSS
<style>
p { color: input }
</style>

HTML JS
<script>
const X='input';
</script>

JSON
{ "name": "input" }

addslashes

Dopasuj funkcje do kontekstu (PHP)

htmlentities

SQL
SELECT ... WHERE
x='input'

mysql_real_escape_string

HTML
<tag>input</tag>

HTML
<tag attrib="input">

HTML CSS
<style>
p { color: input }
</style>

HTML JS
<script>
const X='input';
</script>

JSON
{ "name": "input" }

addslashes

Dopasuj funkcje do kontekstu (PHP)

SQL
SELECT … WHERE
x='input'

htmlentities

mysql_real_escape_string

HTML
<tag>input</tag>

HTML
<tag attrib="input">

HTML CSS
<style>
p { color: input }
</style>

HTML JS
<script>
const X='input';
</script>

JSON
{ "name": "input" }

addslashes

Dopasuj funkcje do kontekstu (PHP)

htmlspecialchars

htmlentities

mysql_real_escape_string

addslashes

SQL
SELECT ... WHERE
x='input'

HTML
<tag>input</tag>

HTML
<tag attrib="input">

HTML CSS
<style>
p { color: input }
</style>

HTML JS
<script>
const X='input';
</script>

JSON
{ "name": "input" }

Dopasuj funkcje do kontekstu (PHP)

SQL
SELECT ... WHERE x='input'

htmlentities

mysql_real_escape_string

HTML
<tag>input</tag>

HTML
<tag attrib="input">

HTML CSS
<style>
p { color: input }
</style>

htmlspecialchars

HTML JS
<script>
const X='input';
</script>

JSON
{ "name": "input" }

addslashes

Dopasuj funkcje do kontekstu (PHP)

strip_tags

htmlentities

mysql_real_escape_string

htmlspecialchars

addslashes

SQL
SELECT ... WHERE
x='input'

HTML
<tag>input</tag>

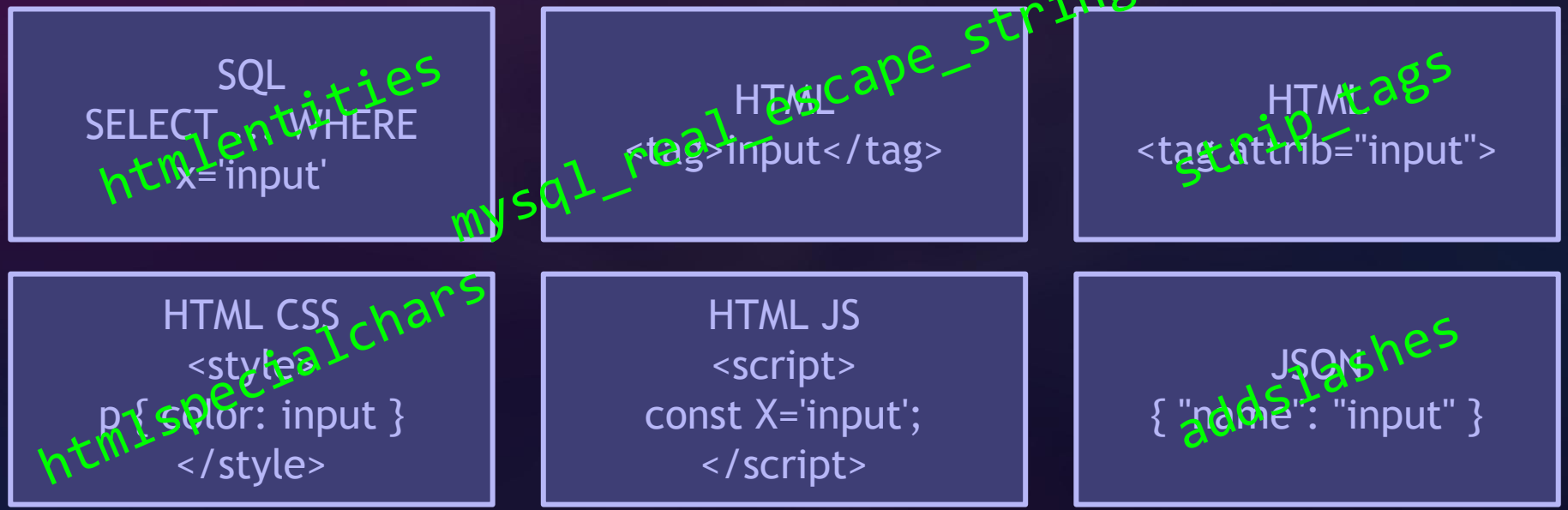
HTML
<tag attrib="input">

HTML CSS
<style>
p { color: input }
</style>

HTML JS
<script>
const X='input';
</script>

JSON
{ "name": "input" }

Dopasuj funkcje do kontekstu (PHP)



Dopasuj funkcje do kontekstu (PHP)

addcslashes

SQL
SELECT … WHERE
x='input'

htmlentities

mysql_real_escape_string

HTML
<tag>input</tag>

HTML
<tag attrib="input">

strip_tags

HTML CSS
<style>
p { color: input }
</style>

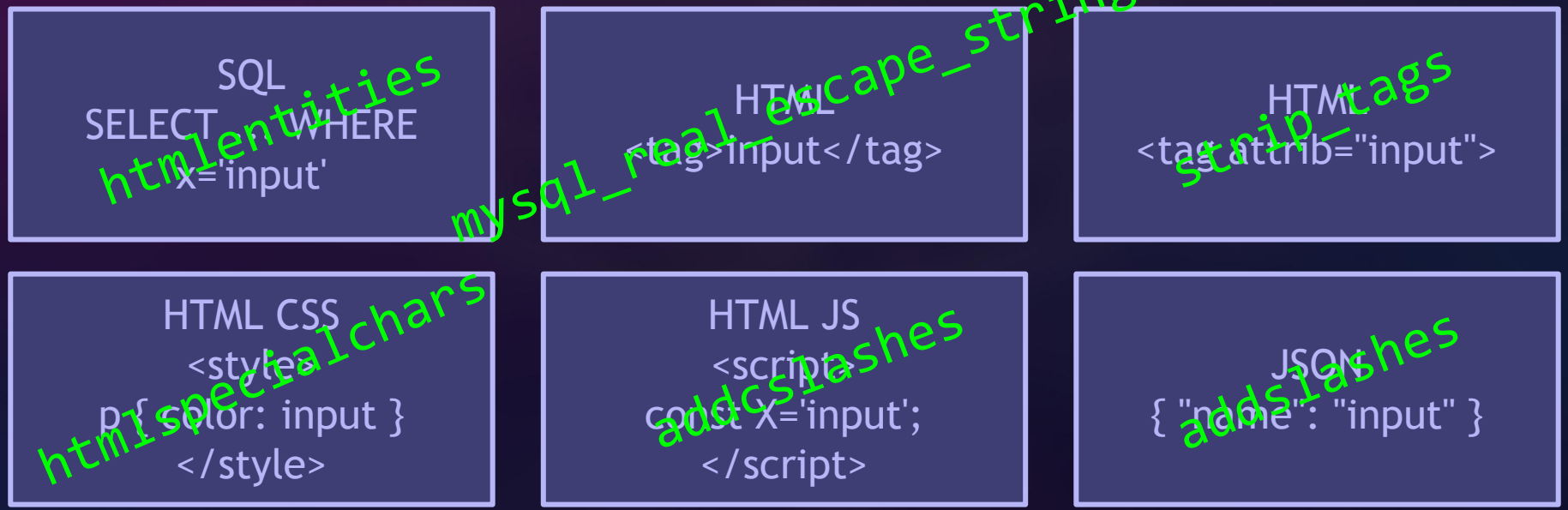
htmlspecialchars

HTML JS
<script>
const X='input';
</script>

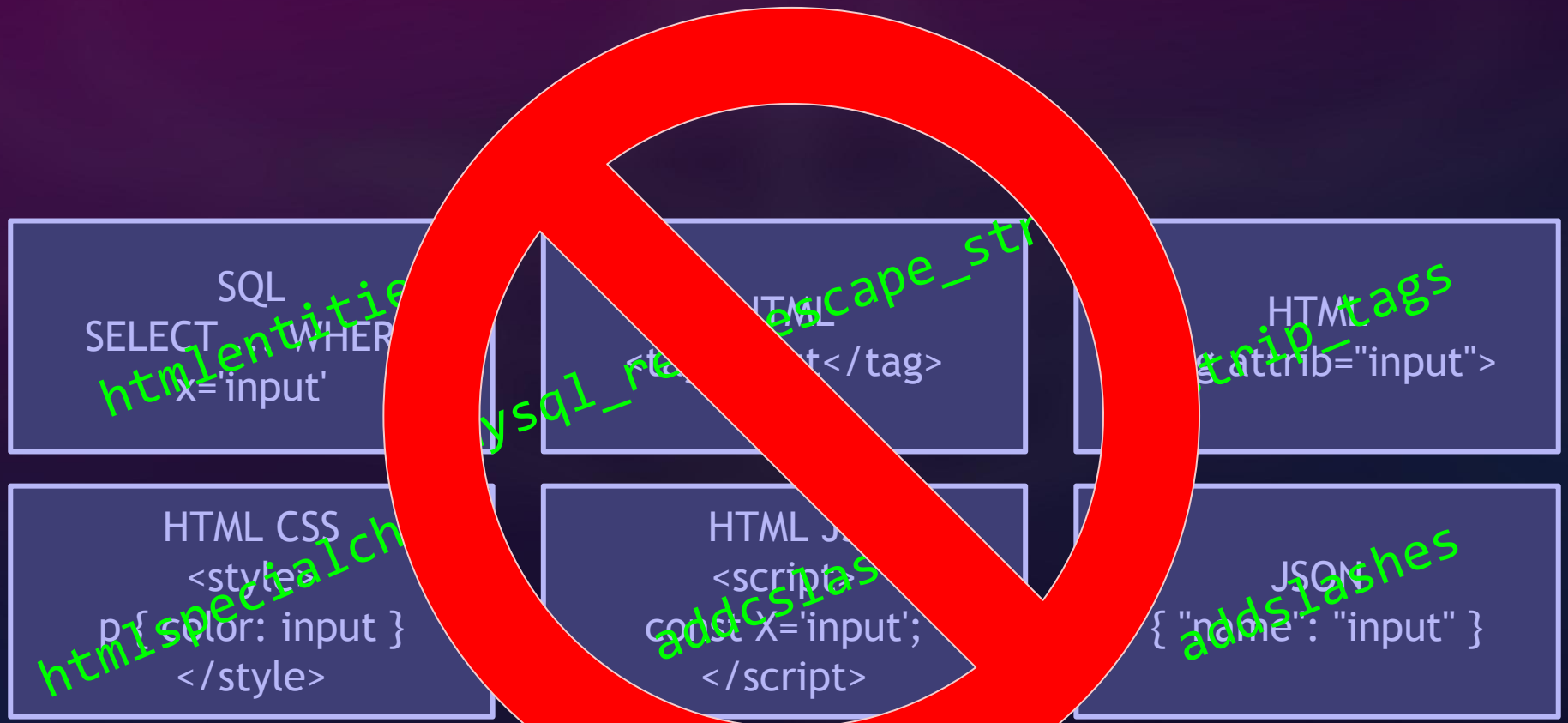
JSON
{ "name": "input" }

addslashes

Dopasuj funkcje do kontekstu (PHP)



Dopasuj funkcje do kontekstu (PHP)



Dopasuj funkcje do kontekstu (PHP)

Czy WSZYSCY programiści ZAWSZE to ogarną?

SQL
SELECT ... WHERE x='input'

HTML
<tag>input</tag>

HTML
<tag attrib="input">

HTML CSS
<style>
p{ color: input }
</style>

HTML JS
<script>
const X='input';
</script>

JSON
{ "name": "input" }

htmlentities

mysql_real_escape_string

strip_tags

htmlspecialchars

addcslashes

addslashes

Prepared statements vs SQLI (PHP)

Na szczęście mamy prepared statements!

Prepared statements vs SQLI (PHP)

Na szczęście mamy prepared statements!

Uwaga: SQL Injection

```
mysql_query("SELECT * FROM asdf WHERE x='$input'")
```

Prepared statements vs SQLI (PHP)

Na szczęście mamy prepared statements!

Uwaga: SQL Injection

```
mysql_query("SELECT * FROM asdf WHERE x='$input'")
```

Prepared statements!

```
$conn->prepare(
          "SELECT * FROM asdf WHERE x=:input")
     ->bindParam(':input', $input);
     ->execute();
```

Prepared statements vs SQLI (PHP)

Na szczęście mamy prepared statements!

Uwaga: SQL Injection

```
mysql_query("SELECT * FROM asdf WHERE x='$input'")
```

Prepared statements!

```
$conn->prepare(
        "SELECT * FROM asdf WHERE x='$input'")
    ->execute();
```

A może by tak...

Idealnie by było, żeby NIE DAŁO SIĘ konkatenować kwerendy SQL ze "zwykłymi" stringami.

A może by tak...

Idealnie by było, żeby NIE DAŁO SIĘ konkatenować kwerendy SQL ze "zwykłymi" stringami.

```
$stmt->select(EVERYTHING, "asdf")
     ->where(
       eq->('x', $input')
     )
     ->execute();
```

A może by tak...

Idealnie by było, żeby NIE DAŁO SIĘ konkatenować kwerendy SQL ze "zwykłymi" stringami.

```
$stmt->select(EVERYTHING, "asdf")
     ->where(
       eq->('x', $input')
     )
     ->execute();
```

Za długie + tracimy zalety SQL.

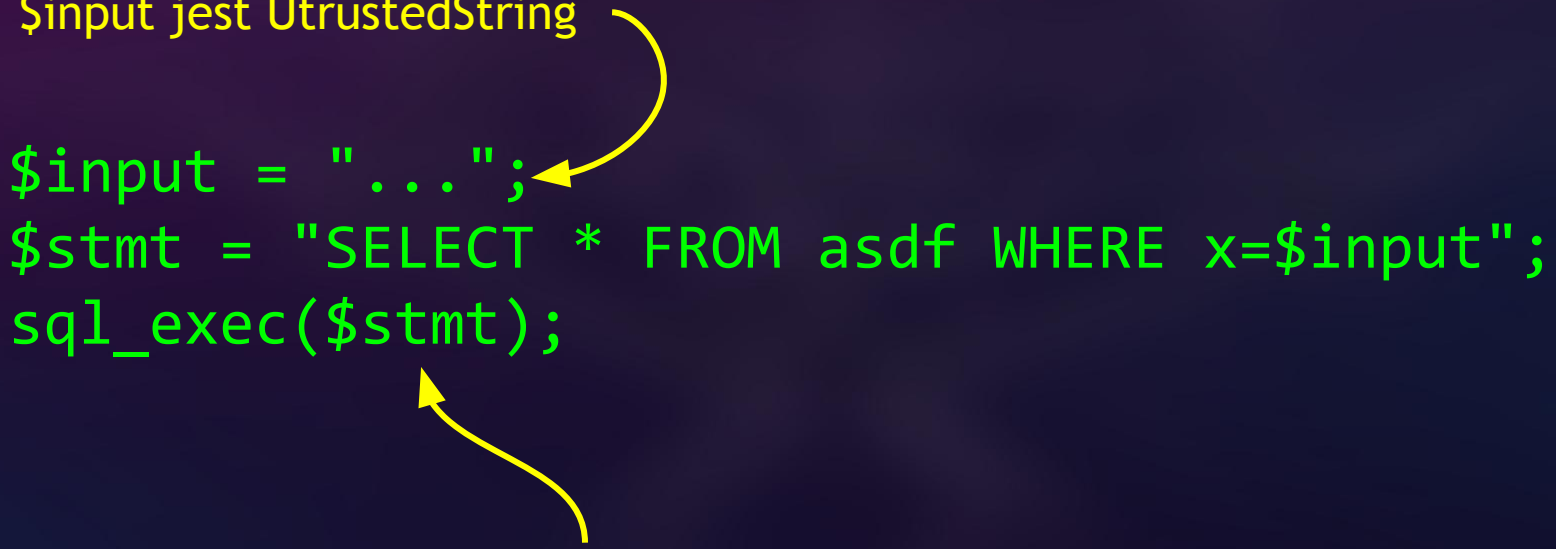
A może by tak...

W idealnym świecie język wie co jest "zwykłym" stringiem, a co np. kwerendą SQL.

```
$input = "...";
$stmt = "SELECT * FROM asdf WHERE x=$input";
sql_exec($stmt);
```

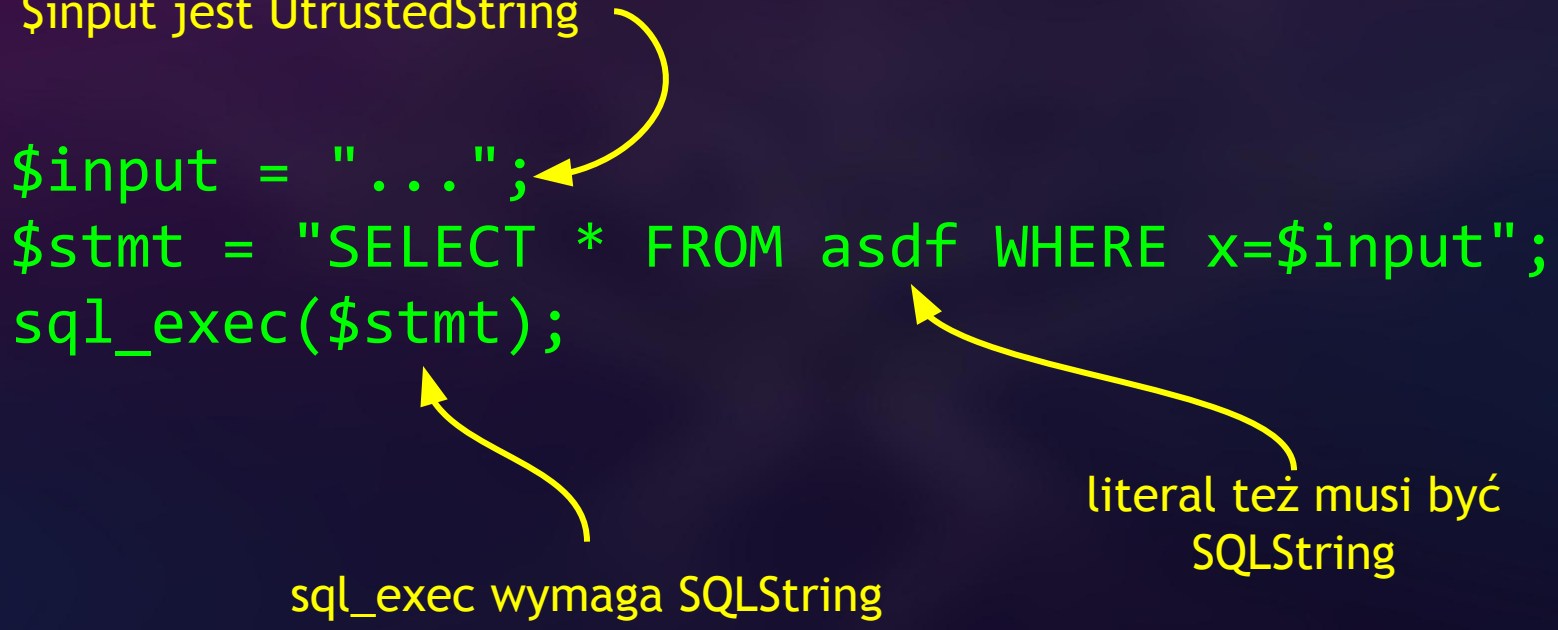
A może by tak...

W idealnym świecie język wie co jest "zwykłym" stringiem, a co np. kwerendą SQL.



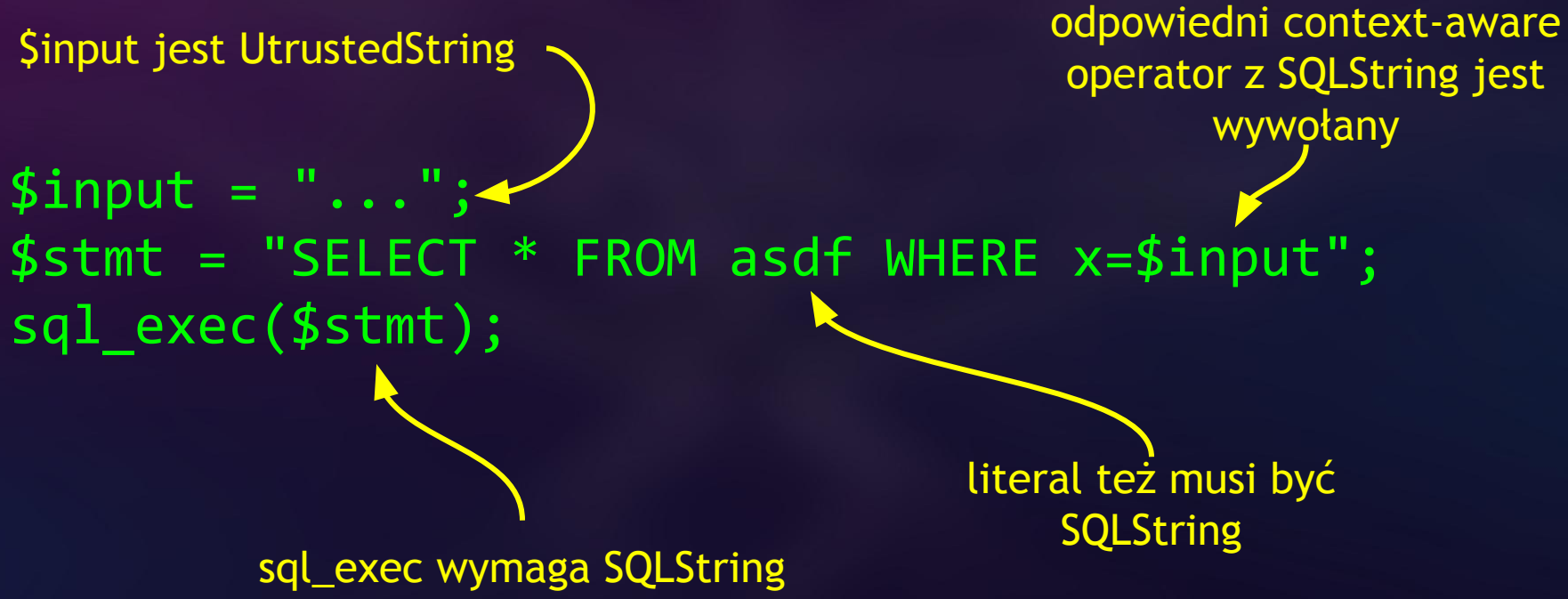
A może by tak...

W idealnym świecie język wie co jest "zwykłym" stringiem, a co np. kwerendą SQL.



A może by tak...

W idealnym świecie język wie co jest "zwykłym" stringiem, a co np. kwerendą SQL.



A może by tak...

Kapkę trudne techniczne...
Nie może być błędów.

```
$input = "...";
$stmt = "SELECT * FROM asdf WHERE x=$input";
sql_exec($stmt);  // Wymaga SQLString
echo($stmt);  // Wymaga HTMLString
```

What now?

Podobna sprawa z szablonami HTML

Podobna sprawa z szablonami HTML

W HTMLu już pogodziliśmy się z szablonami

(Starałem się dobrać nazwy funkcji zgodnie z metodyką PHP)

```
<html>
  <title>[[[title|escape_tag]]]</title>
  <meta charset='[[[charset|attibSecure]]]'>
  <script>
    const TITLE='[[[title|safejs]]]';
  ...
```

Podobna sprawa z szablonami HTML

Można lepiej!

```
[[[context_aware_auto_escape=on]]]
<html>
  <title>[[[title]]]</title>
  <meta charset='[[[charset]]]'>
  <script>
    const TITLE='[[[title]]]';
  ...
```

Podobna sprawa z szablonami HTML

Idealnie - secure by default!

(ale pewnie zadziała tylko dla szablonu HTML)

```
<html>
  <title>[[[title]]]</title>
  <meta charset='[[[charset]]]'>
  <script>
    const TITLE='[[[title]]]';
  ...
```

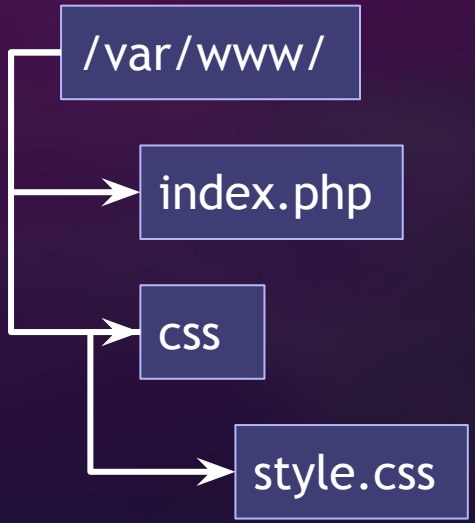
PHP i mapowanie FS

Anty-przykłady

(Tj. będę narzekać na losowe rzeczy)

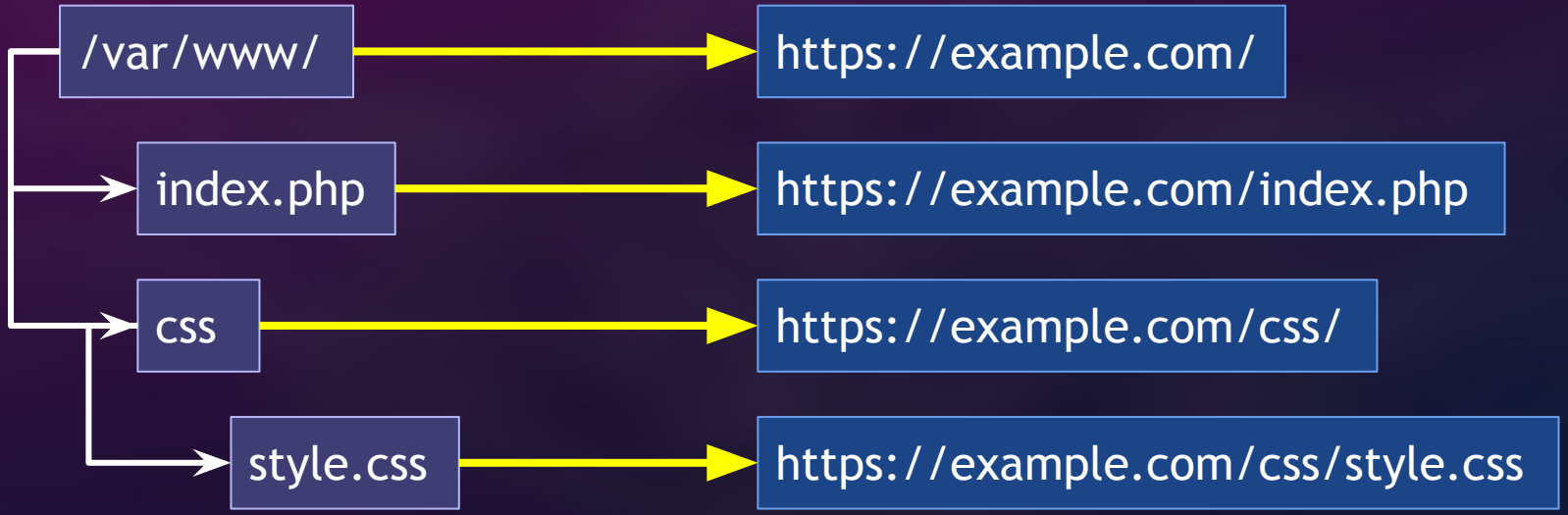
PHP i mapowanie FS do HTTP

Bardzo wygodny ficzer!



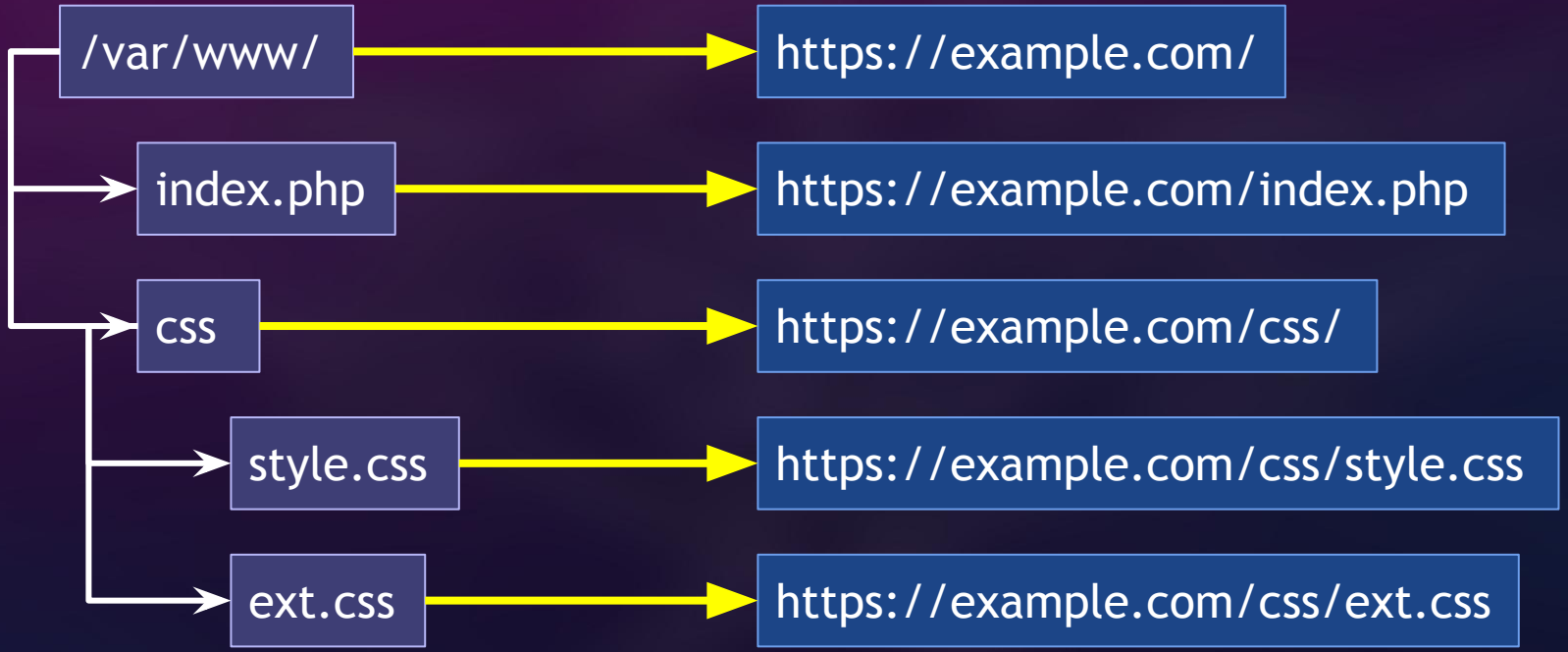
PHP i mapowanie FS do HTTP

Bardzo wygodny ficzer!



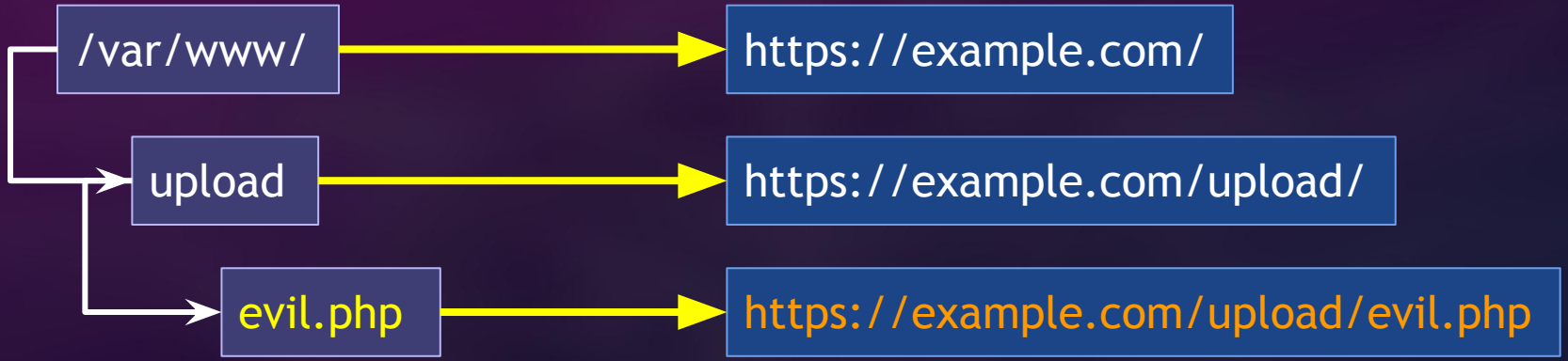
PHP i mapowanie FS do HTTP

Bardzo wygodny ficzer!



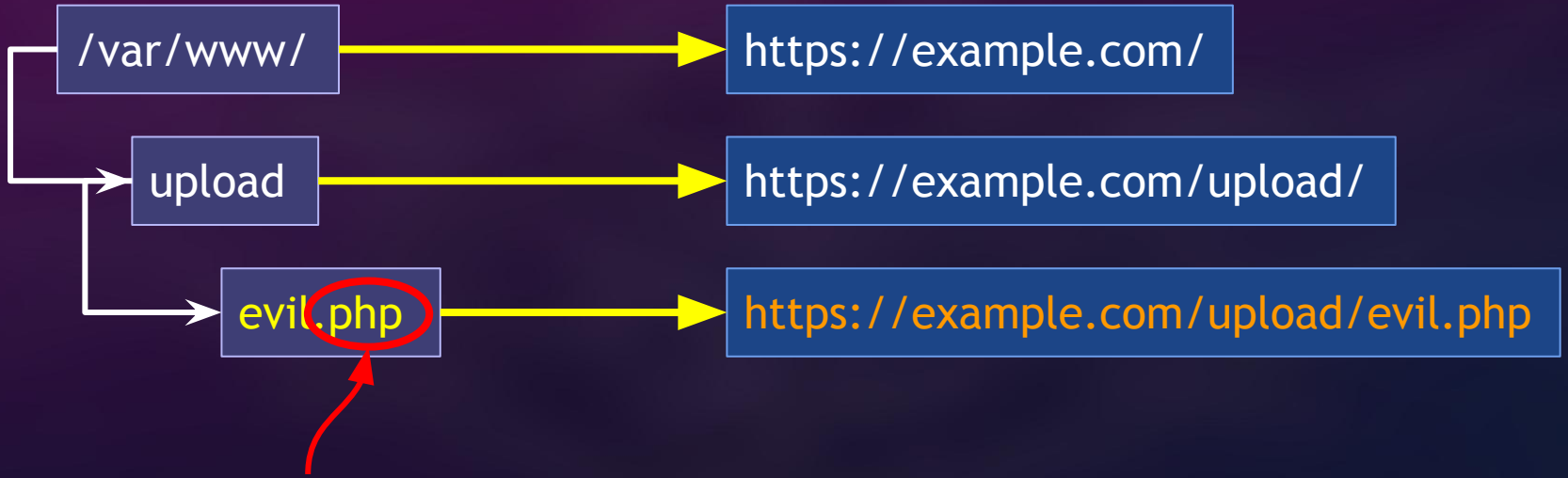
PHP i mapowanie FS do HTTP

Bardzo wygodny ficzer!



PHP i mapowanie FS do HTTP

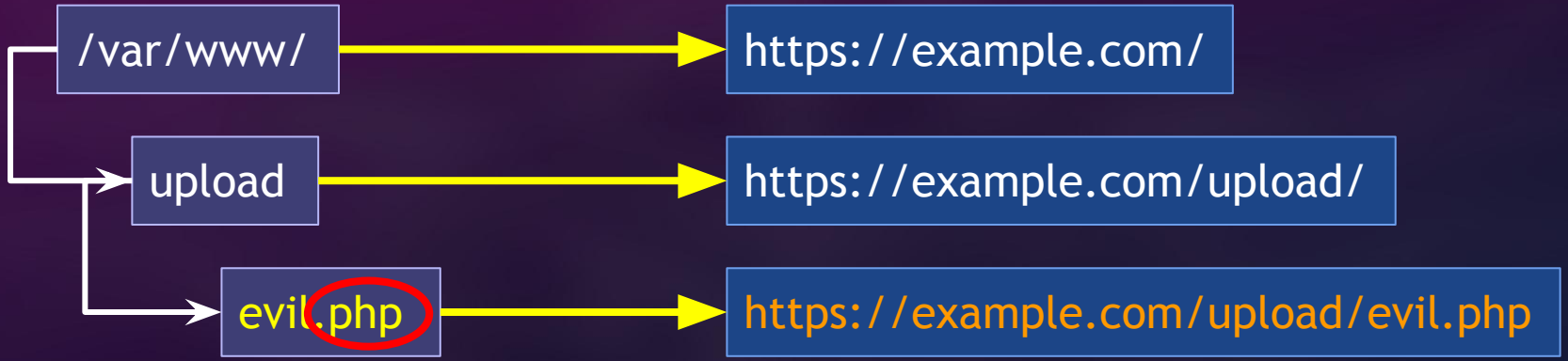
Bardzo wygodny ficzer!



Można wyfiltrować .php!

PHP i mapowanie FS do HTTP

Bardzo wygodny ficzer!

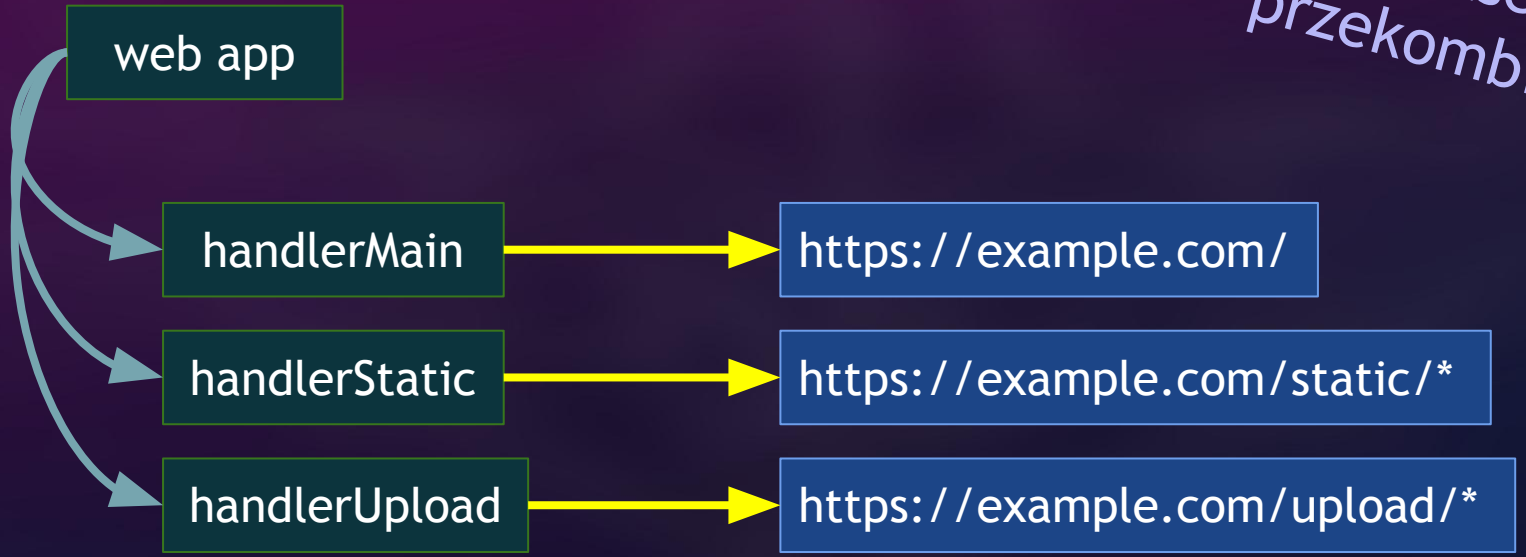


(Arch, all versions) php$: .php
(CentOS with PHP 5.4) php$: .php
(Fedora with PHP 7.0, 7.1) php$: .php
(Fedora with PHP 7.2) (php|phar)$: .php, .phar
(Ubuntu/Debian with PHP 7.1, 7.2) ph(ar|p|tml): .phar, .php, .phtml
(Ubuntu/Debian with PHP 7.0) ph(p[3457]?|t|tml): .php, .php3, .php4, .php5, .php7, .pht, .phtml
(Debian with PHP 5.6) ph(p[345]?|t|tml): .php, .php3, .php4, .php5, .pht, .phtml
(Ubuntu with PHP 5.5) ph(p[345]?|t|tml): .php, .php3, .php4, .php5, .pht, .phtml
(Debian with PHP 5.4) ph(p[345]?|t|tml): .php, .php3, .php4, .php5, .pht, .phtml
(Ubuntu with PHP 5.3) ph(p3?|tml): .php, .php3, .phtml
(Ubuntu with PHP 5.?): .phtm (source)

.htaccess ;)

Frameworki / inne języki umieją bez FS

też czasem można przekombinować ;)



(btw, zuploadowane pliki i tak powinny być serwowane z innej domeny)

Inne

Anty-przykłady
(Tj. będę narzekać na losowe rzeczy)

Losowa lista rzeczy

- Nazwy funkcji vs spójność
- Kolejność argumentów (ln, fgets vs fwrite)
- Wyniki zwracane zgodnie z przewidywaniami
- Możliwość obsługi błędów (atoi)
- Składnia spójna między językami (default value, ^)
- Wbudowane thread-safty (na ile się da)

Podsumowanie

Podsumowanie

Podsumowanie

Podsumowanie

Podsumowanie

Jeśli uczestniczą Państwo w tworzeniu języków, bibliotek, frameworków, etc, zachęcam do zwrócenia szczególnej uwagi na
ich design vs koncept secure by default.

(a bug hunterom / testerom / QA przypominam o możliwości poszukania w dokumentacji tych problematycznych miejsc)

Warto też rzucić okiem na...

https://wiki.theory.org/index.php/YourLanguageSucks

YourLanguageSucks

Contents [hide]

W sumie można by też poprawić tutoriale...

Dzisiaj nauczymy się tworzyć strony internetowe przy użyciu PHP!

Zacznijmy od najważniejszego:

```
<?php
include "page/" . $_GET['page'];
```

KONIEC

Czy są jakieś proste pytania?

E-mail: gynvael@coldwind.pl Twitter: @gynvael YT: GynvaelColdwind / GynvaelEN
Blog: http://gynvael.coldwind.pl/ (Soon also: http://gynvael.live)

Dodatkowe slajdy

Niedokończone lub niewykorzystane

Rozdzielczość w funkcjach mierzących czas

`clock()` w glibc/Linux

`CLOCKS_PER_SEC` → 1 000 000

Czyli jednostki to mikrosekundy (1e-6 sec)!

`clock()` w glibc/Linux

`CLOCKS_PER_SEC` → 1 000 000

Czyli jednostki to mikrosekundy (1e-6 sec)!

```
a = clock();
usleep(1 /*microsec*/);
b = clock();
b - a == ?
```