Second Order Sql Injection

Second Order Sql Injection kullanıcıdan gelen kötü amaçlı kodun ilk yerleştiği sorguda zafiyet teşkil etmediği ancak daha sonra kullanıldığı sorgularda zafiyet teşkil ettiği durumlara denir. Daha detaylı ifade edecek olursak second order sql injection sql kodlarının sisteme gönderilmesi sonrası ilk sorguyu geçip veritabanına olduğu gibi kaydedilmesi, ardından kaydedilen verinin uygulamaya ait bir başka sayfa tarafından çağırılması sonucu ortaya çıkan zafiyete denir.

Second Order Sql Injection saldırısı adım adım şu şekilde gerçekleşir. Kötü amaçlı sql kodları uygulamaya gönderilir. Gönderilen sql kodları sql ifadesine güvenlik fonksiyonları ile güvenli bir şekilde (string'leştirilerek) yerleştirilir. Fakat gönderilen sql kodlarına filtreleme (ayıklama yapma) yerine sadece string'leştirme işlemi yapıldığından sql kodları sorguya string değeri olarak yerleşir ve olduğu gibi veritabanına kaydolur. Ardından uygulamanın bir başka sayfası veritabanına kaydedilen kaydı bir sorgu ile çeker ve çekilen kaydın bir bilgisini ikinci bir sql sorgusundaki Where clause'a dahil eder. İkinci sql sorgusunun Where clause'una eklenen sql kodları ne filtrelemeye ne de string'leştirmeye tabi tutulmamışsa bu durumda sorguda enjeksiyon meydana gelecektir. Sonuç olarak gidiş güvenli olsa bile geliş güvenli olmadığından sql injection saldırısı gerçekleşecektir. Bu kademeli sql injection saldırısına second order sql injection saldırısı adı verilir.

Wikipedia Tanımı:
Second Order SQL injection kötü amaçlı kodlar içeren değerlerin gönderilir gönderilmez yürütülmeyip bir süre tutulduğu zaman oluşur. Uygulama SQL ifadesini güvenlik foksiyonları ile doğru şekilde encode edip geçerli SQL ifadesi olarak depo edebilir. Sonrasında SQL injectona karşı denetimsiz olan uygulamanın başka bir kısmında depolanan SQL ifadesi çalıştırılır. Böylece sql injection saldırısı gerçekleşmiş olur. Bu saldırıyı gerçeklemek için saldırganın gönderilen değerlerin daha sonra nasıl kullanıldığına dair daha fazla bilgiye sahip olması gerekir. Otomatik web uygulaması güvenlik tarayıcıları bu tür bir SQL injection'ları kolaylıkla algılayamaz. Dolayısıyla kötü niyetli yazılımların kodun handi kısmında olduğu manuel olarak kontrol edilmelidir.

Uygulama Açıklaması

Second Order Sql İnjection zafiyetini anlatmak adına kullanılacak uygulama

/var/www/Second Order SQL Injection Uygulaması

dizininde mevcuttur. Bu başlıkta önce bahsedilecek uygulamanın nasıl işlediği anlatılacaktır. Sonraki başlıkta ise uygulamalı second order sql injection saldırısı gerçekleştirilecektir.

Öncelikle uygulamamızı tanıyalım. Uygulamamızda iki web sayfası mevcuttur. Birincisi veritabanına insert eden sayfa, ikincisi veritabanından veri çekip ekrana basan sayfa. Insert eden sayfanın kaynak kodu aşağıdaki gibidir.

insertPage.php

```
…

$sql_statement = "INSERT into datastore(nickname,age,firstName,lastName)
values('" . mysql_real_escape_string($_REQUEST["nickname"]) . "'," .
intval($_REQUEST["age"]) . ",'" .
mysql_real_escape_string($_REQUEST["firstName"]) . "','" .
mysql_real_escape_string($_REQUEST["lastName"]) . "')";

$query = mysql_query($sql_statement, $conn) || die(mysql_error());

…
```

Görüldüğü üzere insert eden sayfaya girilen her input denetime tabi tutulmuş. Örneğin parametrelerden birine sql injection kodu girildi diyelim. Bu durumda girdiğimiz bu sql injection kodu mysql_real_escape() fonksiyonu sayesinde sorguya güvenli bir şekilde (string'leştirilerek) yerleştirilecektir. Fakat yerleşen bu input filtrelemeye tabi tutulmadığından (yani ayıklanmadığından) string olarak olduğu gibi veritabanına kaydolacaktır.

Veritabanındaki kayıtları ekrana basan sayfa ise aşağıdaki gibi olsun.

showPage.php

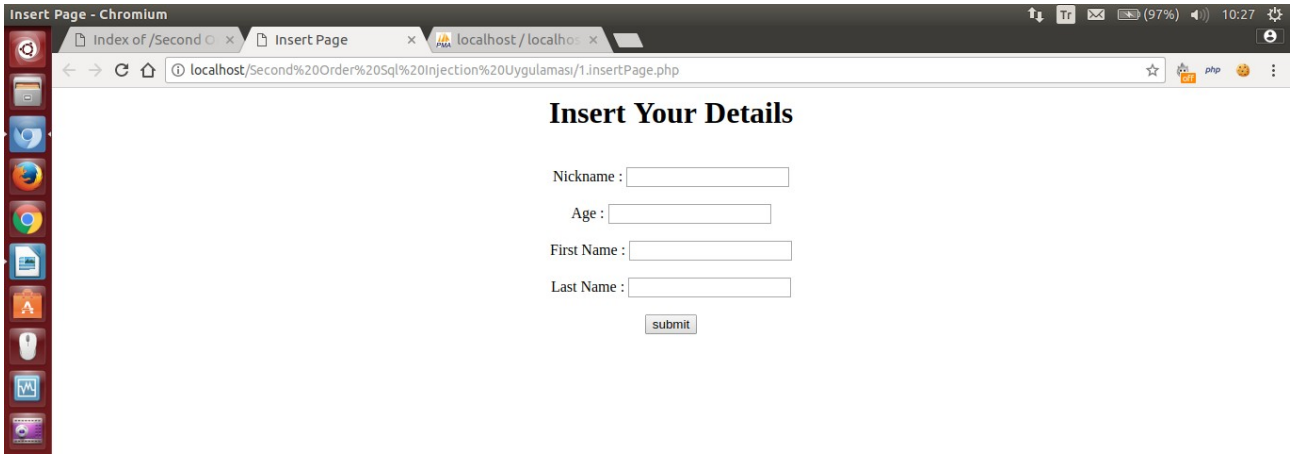
```
…

// İlk Sorgu (Nickname'e göre veri çeken sorgu)
$sql_statement = "Select * from datastore where nickname='" .
mysql_real_escape($_POST["nickname"]) . "'";

$result = mysql_query($sql_statement, $conn);
$row = mysql_fetch_row($result);

// İkinci Sorgu (Problemli Sorgu)
$sql_statement2 = "Select * from datastore where firstName='" . $row[2] . "'";

$result2 = mysql_query($sql_statement2, $conn);
$row2 = mysql_fetch_row($result2);

echo "<br><b><center>Database output</center></b></<br><br>";
echo "<b>$row2[0]</b><br>Age: $row2[1]<br>First Name:$row2[2]<br>Last
Name: $row2[3]<br><hr><br>";

…
```

Görüldüğü üzere ekrana basan sayfadaki ilk sorgu dışarıdan gelen nickname'e göre veritabanından bir satır çekiyor. İkinci sorgu (problemli sorgu) ise çekilen satırdaki bir bilgiyi (kolonu) kullanarak bir başka sorguda bulunuyor. Dolayısıyla ikinci sorguda sorguya eklenen veri veritabanında tutulan sql injection kodu olduğundan sql injection zafiyeti meydana geliyor. Eğer ikinci sorguda mysql_real_escape() fonksiyonu kullanılsaydı ilk sorguda veritabanından çekilen sql injection kodları ikinci sorguya string'leştirilerek eklenecekti ve böylece ikinci sorguda sql injection meydana gelmeyecekti. Fakat ikinci sorguda mysql_real_escape() denetimi kullanılmadığından birinci sorguda veritabanından çekilen sql injection kodları ikinci sorguya faal olarak eklenecektir ve ikinci sorguda sql injection saldırısı gerçekleşecektir.

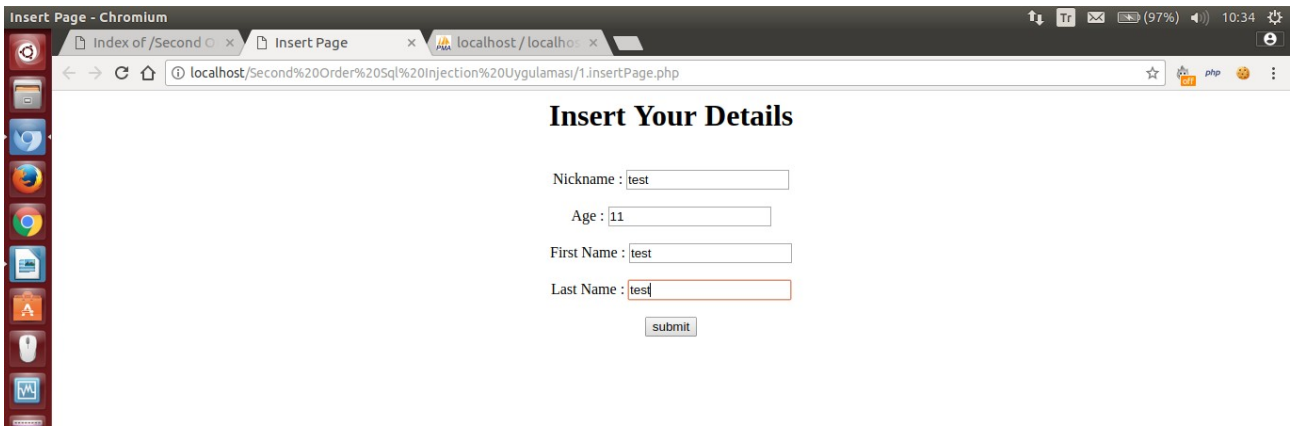
Sonuç olarak sql injection kodu insert sayfasında girilir ve veritabanına kaydolur. Sqli kodların yerleştiği satırdaki ilgili kolon uygulamanın bir başka sayfasındaki sorgu ile çekilir ve kolonun tuttuğu değer aynı sayfadaki ikinci sorgunun Where clause'una yerleştirilir. Eğer where clause'da mysql_real_escape() fonksiyonu kullanılmamışsa bu durumda sql injection kodları sorguya faal olarak eklenir ve ekrana sql injection saldırısının çıktısı yansır. Birinci sorgu ile veritabanından veriyi çekip ardından çekilen değeri ikinci sorguya koyma sonucu ikinci sorguda doğan zafiyete second order sql injection zafiyet adı verilir.

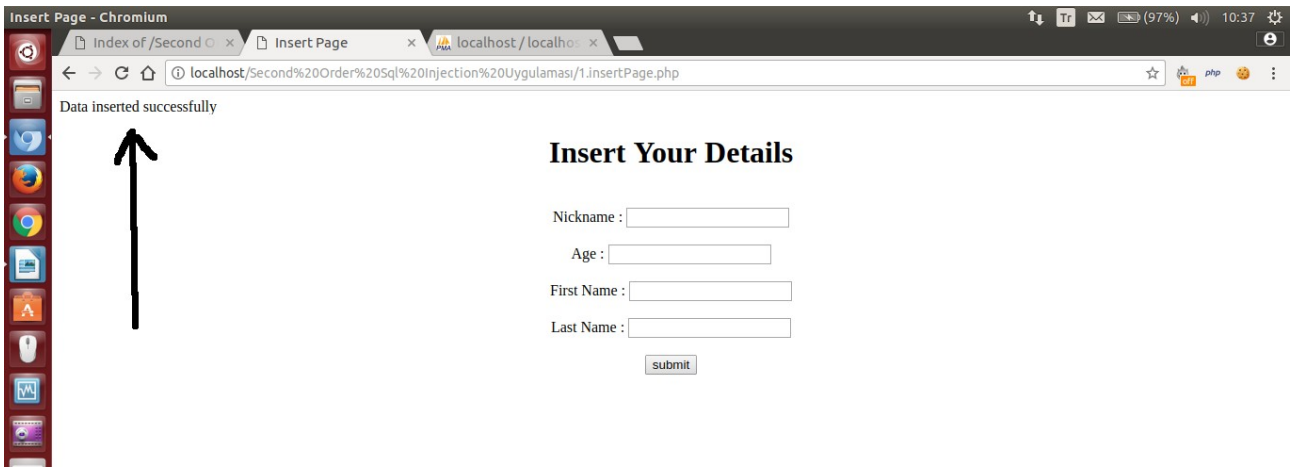
Uygulama (Second Order Sql Injection Saldırısı Örneği) // Birebir denenmiştir ve
// başarıyla uygulanmıştır.

Öncelikle veritabanına veri ekleyen insertPage.php'yi bir görelim.

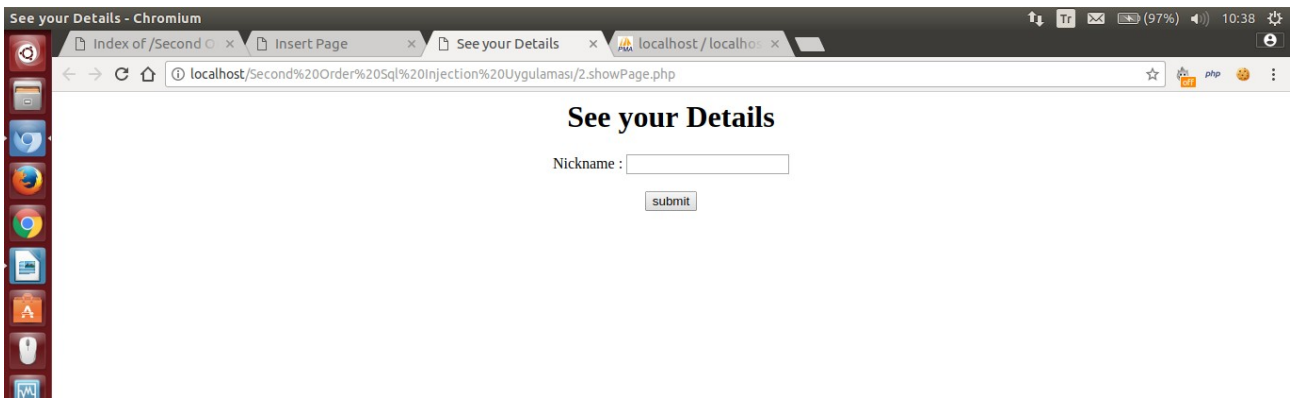


Bu sayfaya girdiğimiz veriler veritabanına kaydolmaktadır. Normalde bu sayfada sql injection'a karşı koruma vardır. Fakat sayfa kullanıcıdan gelen input'ları sorguya güvenli şekilde yerleştirse bile input'ları olduğu gibi veritabanına kaydetmektedir. Bu nedenle veritabanına koyulan verinin çıktığı nokta denetim altında alınmazsa second order sql injection yapılmış olacaktır. Şimdi ilk olarak sayfaya normal veriler girelim.

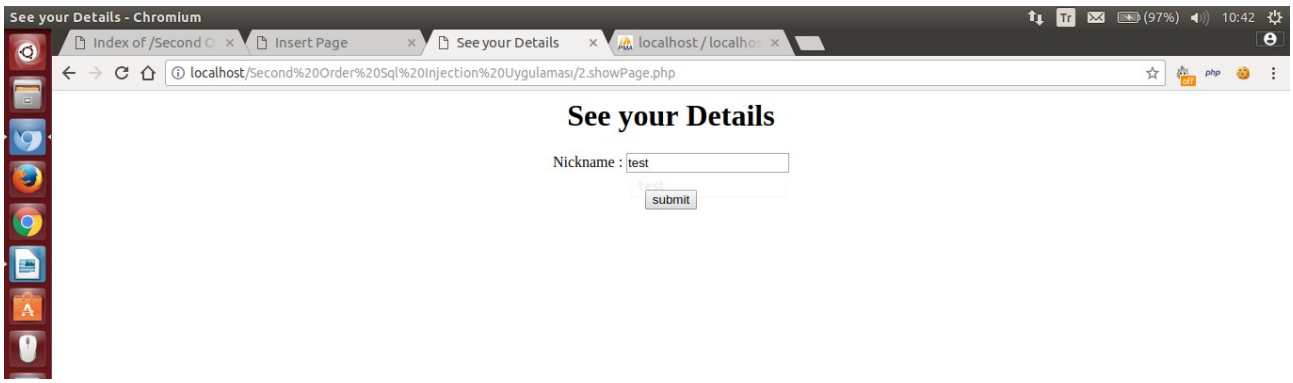


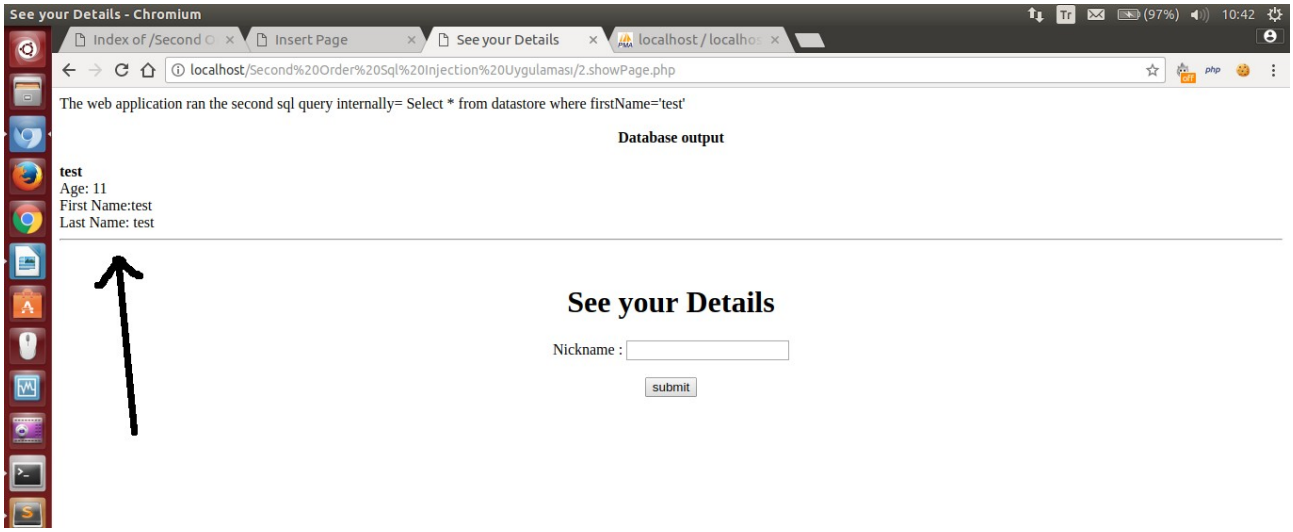


Ardından girdiğimiz veriyi görüntüleyebileceğimiz sayfaya (showpage.php'ye) geçelim.

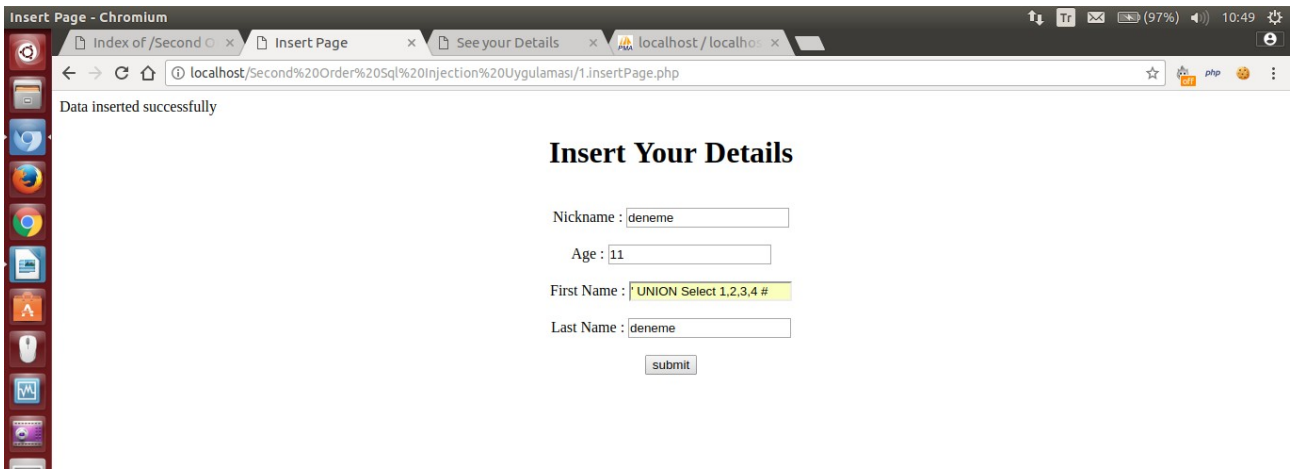


Bu sayfaya nickname'mimizi (test verisini) girdiğimizde girdiğimiz bilgiler ekrana gelecektir.

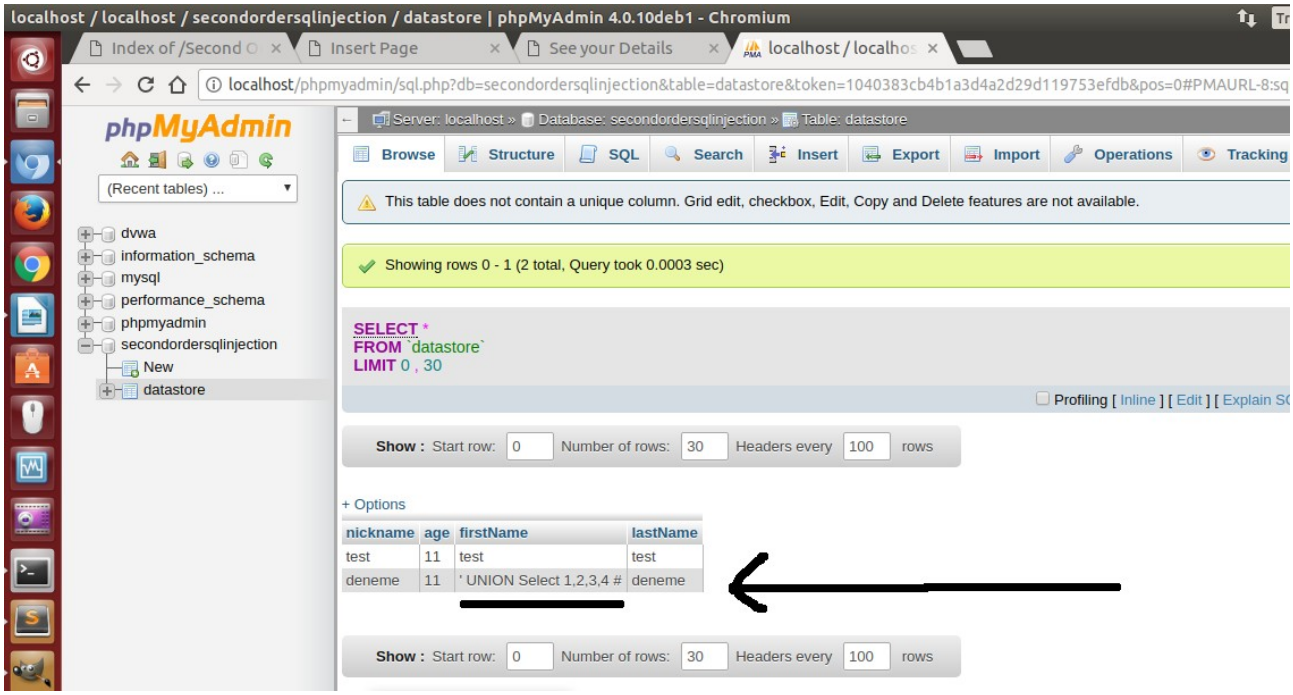




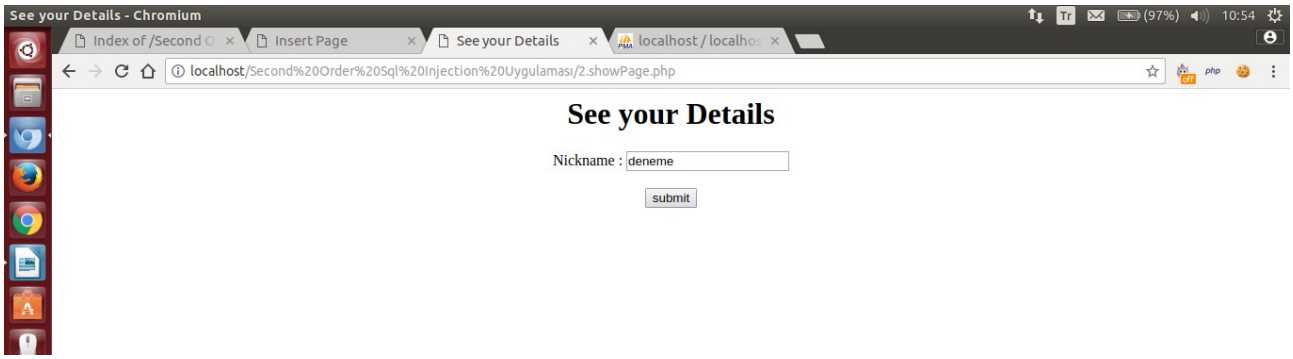
Şimdi ilk sayfaya sql injection ifadesi girelim.

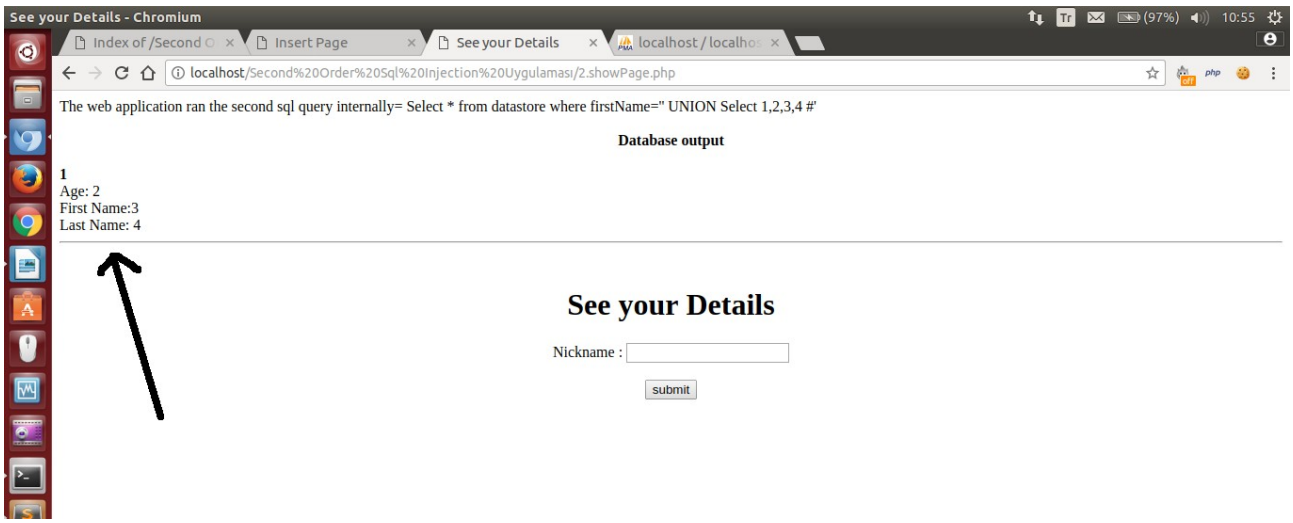


Sql injection kodu güvenli şekilde sorguya yerleştirilir, fakat olduğu gibi veritabanına kaydedilir.



Ardından girdiğimiz bilgileri görüntüleme sayfasına gelelim ve son girdiğimiz kayıtı getirelim.





Görüldüğü üzere metin kutusuna girdiğimiz ' UNION Select 1,2,3,4 # sorgusunun çıktısı ekrana yansımıştır. Normalde bu sayfanın arkaplanında ' UNION Select 1,2,3,4 # verisi veritabanından ilk sorgu ile çekilmiştir ve ikinci sorgunun where clause'una dahil edilmiştir. İkinci sorguda denetim olmadığından where clause'da bir enjeksiyon olmuştur ve ekrana UNION'ın çıktısı yansımıştır. Böylece insert sayfasına ' UNION Select 1,2,3,4 yerine information_schema 'dan bilgi toplayacak select sorguları koyarak show sayfasında çıktıları görüntüleyebilir ve sql injection saldırılarımızı böylelikle devam ettirebiliriz.

Not:

Second order sql injection zafiyetinin normal sql injection'dan farkı deneme yanılma ile gireceğimiz sql kodlarının çıktılarını uygulamanın bir başka sayfasında görüntülüyor oluşumuzdur.

Yararlanılan Kaynaklar

https://www.esecforte.com/second-order-sql-injection/

https://tr.wikipedia.org/wiki/SQL_Injection#.C4.B0kincil_SQL_injection.28Second_order_SQL_injection.29