Blind SQL Injection

Bünyamin Demir, Ağustos 2009, WGT E-Dergi 1. Sayı

Bir çoğumuz SQL cümlecikleri kullanarak veritabanlarımızda listeleme, güncelleme ve ekleme işlemleri yaparız. Fakat bu kullandığımız SQL cümleciklerinin ne tür sıkıntılar doğurabileceğinin üzerinde pek durmayız. Bir yazılımcı için önemli olan tüm kullanıcı listesini çekmek ise "SELECT * FROM users" yazmaktır. Belli bir numaraya sahip bir kullanıcı çekmek için ise "SELECT * FROM users WHERE UserID=101" yazmak yetecektir. Fakat kullandığımız programlama teknikleri bir kullanıcının bilgilerini çekmek için yazılan SQL cümleciğini çok tehlikeli durumlara sokabilir. Bu aslında bazen kullanılan dilin, bazen de yazılımcının tekniğinin sonucudur. Yine de ucu açık bir tartışma konusu diyebiliriz.

SQL Injection için söylenecekler çok fazla olsa da temel de problemin ortaya çıkış nedeni; cümleciğin herhangi bir yerine eklenen kod parçacıklarıyla farkedilir. Örneğin; " ` " gönderip, sonucunun hata dönmesi gibi. Tabi bazen bu hata çıktılarıda işimizi görmeyebilir veya hata dönmeyebilir. Blind hem SQL Injection açığını barındırmasıyla birlikte sonucun her zaman true/false dönmesiyle ilgilenir. Aslında bu açıklığın ciddi bir problem oluşturması temel bir matematik probleminin varlığıyla meydana gelmektedir.

Blind SQL Injection için örnek;

Sorgu1: SELECT Username FROM users WHERE userid=1
Cevap : bunyamin

Sorgu2: SELECT Username FROM users WHERE userid=121
Cevap :

Görüldüğü gibi database de userid=1 de bir kullanıcı var, userid=121 de ise kullanıcı yok. Burda yapılan bir tane "true" değeri yakalayabilmek. Sorgu1 ile bu sağlanmış oldu. Bundan sonra yapılması gereken, görülen cevaba bakılarak sorgu üzerinde oynamalar yapmak. Her değişikliğimizde geri dönen cevap, yaptıklarımızla doğru yolda olup olmadığımızı söyleyecektir. Sorgu1`i biraz değiştirecek olursak;

Sorgu1: SELECT Username FROM users WHERE userid=1 AND ASCII(SUBSTRING(Lastname,1,1))=97
Cevap :

Userid=1 için kullanıcının soyisminin ilk harfi "a" mıdır? Görüldüğü gibi bizim daha önce elde ettiğimiz içerik gelmedi. Biz her sorgumuz da bu içeriğin aynısının gelip gelmediğine bakıyoruz (Yukarıda cevabın true/false dönmesi ile farkedilir dediğimiz kısım).

Sorgu1: SELECT Username FROM users WHERE userid=1 AND ASCII(SUBSTRING(Lastname,1,1))=98
Cevap :

Userid=1 için kullanıcının soyisminin ilk harfi "b" midir?

Sorgu1: SELECT Username FROM users WHERE userid=1 AND ASCII(SUBSTRING(Lastname,1,1))=68
Cevap : bunyamin

Userid=1 için kullanıcının soyisminin ilk harfi "D" olarak karşımıza çıktı. Çünkü lastname`ni ilk karakterinin ASCII değeri 68. Bu da "D" harfine denk gelmektedir. Bu sorgumuzla birlikte daha önce görmüş olduğumuz içeriği (true) elde ederek farkettik.

Sorgu1: SELECT Username FROM users WHERE userid=1 AND ASCII(SUBSTRING(Lastname,2,1))=69
Cevap :

Userid=1 için kullanıcının soyisminin ikinci harfi "E" midir?

Sorgu1: SELECT Username FROM users WHERE userid=1 AND ASCII(SUBSTRING(Lastname,2,1))=101
Cevap : bunyamin

Userid=1 kullanıcısının soyisminin ikinci harfi "e" dir.

Buraya kadar Blind SQL Injection açığının ne olduğunu temel olarak görmüş olduk. Fakat tablo isimleri, tabloların alan adları, içindeki verileri böyle teker teker deneyerek bulmak nekadar zamanımızı alır? Muhtemelen a-z,A-Z ve 0-9 arasına bir de "?,-,+,* v.s" gibi karakterlerin de varlığını hesaba katarsak. Substring ile elde ettiğimiz tek karakteri deneyerek bulmak, bulanık bir zaman dilime gelecekti ki bu da elle imkansıza yakın. Tabi çok büyük verilerin bu şekilde alınmasından bahsediyoruz.

Şimdi ise bu problemi gerçekten kritik bir problem haline getiren matematiksel alt yapıyı ele alalım.

Algoritma Analizi ve Karmaşıklık

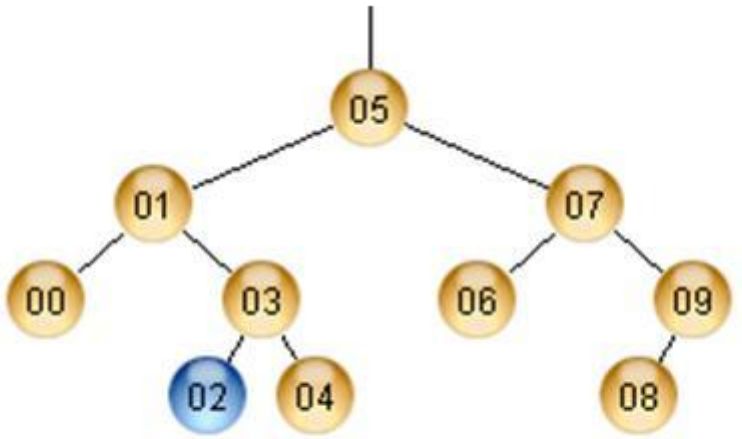
Aslında bir algoritmayı neden analiz etmek isteriz? Performansı ölçmek için, farklı algoritmalarla karşılaştırabilmek için, daha iyisi mümkün mü? Bir algoritmanın özelliklerini analiz ederken de çalışma zamanı ve hafıza da kapladığı alana bakarız. Tabi bu maddeleri çoğaltmak mümkün. Fakat genel kabul bir algoritmanın karmaşıklığı; çalışma hızı ve hafıza da kapladığı alanla belirlenir. Biz şu an "zaman" problemi üzerinde duracağız. Çünkü Blind SQL Injection da gördüğümüz gibi teker teker deneme yoluyla bilgi toplamak gerçekten çok zor. Ancak bu deneme zaman aralığını (deneme sayısını) kısaltırsak bu problemi kullanışlı hale getirebiliriz. Zaman karmaşıklığı Büyük-O notasyonu ile gösterilir.

İkili Arama (Binary Search)

Problem çözümlerinde böl ve yönet prensibine dayanır. Çözüm süresini genelde ½ oranında bölerek zaman aralığını kısaltmayı hedefler. Bu algoritma da dikkat edilmesi gereken bir husus ise elemanların artan veya azalan şekilde sıralı olmasıdır. Arama işlemi ilk bölünmede

sonuçlanmıyorsa tekrar alt kümeler de bölünerek arama işlemi devam eder. İkili arama algoritması için zaman karmaşıklığı O(logN)`dir.

Örnek verecek olursak;



Sayı olduğunu bildiğimiz bir alan da "2" sayısına ulaşabilmek için 0,1,2,3,4,5,6,7,8,9 u sırayla denesek 3. deneme de bulacaktık. Fakat bu tamamen şansımızla alakalıydı. Eğer aynı durum da "7" için bir arama yapıyor olsaydık bu sefer 8. Deneme de bu sayıyı bulacaktık. Fakat ikili arama lagoritması ile önce 5`den büyük veya küçük olduğuna bakılır. Küçükse soldan, büyükse sağdan devam edilir. Eşit ise durulur.

7 için;

7=5 (false)
7>5 (true)
7=7 (true)

3. denemede eşitliği yakaladık.

ASCII karakterler yardımıyla denemelerimizi yapmıştık. ASCII okunabilir karakterler 36-126 arasındadır.

. f(n) = ?(log n)
. f(126-32) = ?(log (126-32))?
. ~7 karşılaştırmada karakterin doğruluğuna erişilir.

Yani bu aradaki tüm karakterler için en fazla 7 deneme yapmamız yeterlidir. Oysa 126-32=94 defa deneme yapmaktan kurtulmuş oluyoruz. Tabi bu her bir karakter için 94 defa bakılacak anlamına gelmez. Şansınız iyiyse bu sayı daha da küçülebilir. Fakat aynı durum ~7 için de geçerlidir.

Bu sayede bir karakter için harcamamız gereken maksimum deneme 7 olarak belirlemiş olduk. Peki deneme yapacağımız karakter sayısı çok fazla ise? Bu sefer deneme yapılan kümeyi biraz daha azaltabiliriz. Yukarıda örnekler de görüldüğü gibi "E" ve "e" harfleri bir birinden farklı olarak algılanmıştı. Eğer A-Z arasındaki büyük harfleri atar ve kontrol etmek istediğimiz karakteri lowercase`den geçirirsek?

Sorgu1: SELECT Username FROM users WHERE userid=1 AND ASCII(SUBSTRING(Lastname,1,1))=68
Cevap : bunyamin

Bu sefer "D" harfi yerine "d" yi kontrol ederek

Sorgu1: SELECT Username FROM users WHERE userid=1 AND ASCII(LOWER(SUBSTRING(Lastname,1,1)))=100
Cevap : bunyamin

Aynı sonuca erişebiliriz. Bu sayede deneme yapacağımız kümeyi ~6 indirebiliriz.

Bu sayede en fazla 94 deneme yerine en fazla 6 deneme de istediimiz bilgiye erişme olanağı ortaya çıkmış oldu.

Özetle; binary search algoritmasının varlığı Blind Sql Injection`u daha tehlikeli hala sokuyor. Eğer bu şekilde bir algoritma olmasaydı bu açıklığın kullanılabilir olması pek mümkün olmayacaktı.

Kaynaklar

1) http://www.owasp.org/index.php/Blind_SQL_Injection
2) http://sqlmap.sourceforge.net/
3) http://www.owasp.org/index.php/Testing_for_SQL_Injection_(OWASP-DV-005)