SQL Injection

Vložil/a pog [1], 24 Leden, 2005 - 04:54

- Hacking [2]
- Hacking method [3]
- SQL [4]

Pro mnoho z vás "SQL injection" znamená denní zábavu, pro někoho cizí pojem. Zajímá vás, co to je? a hlavně: jak na to?

Rozsáhlejší a podrobnější článek můžete nalézt zde: SQL Injextion (Full paper) [5]

O co jde?

pod pojmem SQL injection se skrývá podvržení vstupních dat (hodnot proměnných odesílaných serveru) tak, aby byl nějakým způsobem pozměněn výsledek SQL dotazu. Pokud útočník zná strukturu tabulky (nejlépe i SQL dotazů), které svými proměnnými ovlivní, má vše daleko jednodušší, než když musí odhadovat, jaké sloupce jsou použity, jaké mají asi datové typy a jak se jmenují. Proto JE NESMÍRNĚ DŮLEŽITÉ, aby vaše skripty podávaly co nejméně informací o struktuře vašich databází a tabulek v případě, že se vyskytne nějaká chyba. Osvědčený způsob je volat vždy v případě neúspěchu svou funkci, která zjistí, jestli může zobrazit informaci o chybě (například když má klient IP adresu 127.0.0.1, což znamená, že spouštíte aplikaci na svém počítači) a nebo ne.

Co hrozí?

útočník může udělat 4 základní věci

- může získat přístup k citlivým datům (například uživatelská hesla, skryté emaily atd)
- může získat přístup k jakémukoliv, například administrátorskému účtu na webu (je-li nějaký)
- může získat přístup k všem účtům naráz
- může vám samozřejmě smazat všechna data co máte v tabulkách (proto doporučuji velmi často zálohovat obsah vašich databází)

Co nehrozí (v MySQL ve spojení s PHP)?

- PHP nepovoluje vykonávání několika SQL dotazů v jednom volání funkce mysql_query(), takže tím odpadá problém s řetězcem typu '; truncate tabulka; - provádí se vše, co se nachází před prvním středníkem mimo hodnoty sloupců.
- V MySQL neexistuje nic, co by mohlo zavolat externí aplikaci jako je tomu například u Microsoftí alternativy MSSQL, takže odpadá spousta dalších potenciálně nebezpečných dotazů.
- zobrazení vašich zdrojových kódů - pokud nemáte v databázi uložený kód, který se pak pomocí eval() provádí
- zjištění hesla k vašemu FTP (pokud nemáte ve všech účtech použité stejné heslo)
- jak už jsem zmínil, pokud ve svých databázích neuchováváte PHP kód který následně pomocí funkce eval() vykonáváte, nehrozí ani provedení cizího kódu na vašem serveru

-

a ještě pár slov než začneme:

možná se Vám budou simulované situace zdát absurdní, budete si říkat "takhle hloupě se ten kód snad ani nedá napsat?!" nebo "nikdy mě ani nenapadlo, že by se to dalo takhle napsat" - v tom případě je vše v pořádku. V opačných případech, pokud Vám zde uvedený kód není cizí a používáte ho někde na svých webech, měli byste při čtení těchto řádek dostávat nefalšovanou panickou hrůzu, zvlášť jestli Vaše dílo běží na internetu již delší dobu.

Následující příklady by fungovaly s předpokladem, že na serveru jsou nastavené register_globals na ON a že jsou gpc_magic_quotes na OFF (nesmějte se, i s touto konfigurací jsem se setkal a musím říct, že takto nastavený server si přímo říkal o "drobnou změnu v obsahu")

Nyní budou následovat konkrétní příklady využití (a zneužití) SQL injection a budu je aplikovat na následující tabulce:

```
Název sloupce  Datový typ
id             INT
nick           VARCHAR(32)
email          VARCHAR(255)
```

Příklad #1 - zobrazení celého obsahu tabulky poprvé

původní dotaz:
```
SELECT nick, email FROM uziv WHERE id = {$id}
```
použité proměnné:
$id = integer získaný od uživatele, například po kliknutí na "ukaž detaily o uživateli"
mnou vložená hodnota:
```
9 OR 1=1 --
```
výsledný dotaz:
```
SELECT nick, email FROM uziv WHERE id = 9 OR 1=1 --
```

Popis:

principem všech operací se "všemi daty" je to, že v dotazu musí být vždy platná podmínka a tudíž dotaz (select, update, delete, to je jedno) ovlivní celý obsah tabulky. Jediné omezení při SQL injection je to, že vzniklý dotaz musí být syntakticky správný - musíte nejdřív ukončit původní podmínku a pak teprv můžete pokračovat se svým kódem. V našem případě je ukončení původního dotazu pouze první znak (9), zbytek už je naše "vždy platná podmínka" - 1 se vždy rovná 1. MySQL má komentáře ve stylu "--" a to se nám hodí pro případ, že v originálním dotazu je ještě něco za částí, kterou zneužíváme (klauzule order by, limit a jiné)POZOR: TENTO DOTAZ BUDE ZE STRANY SERVERU FUNGOVAT VŽDY, i když budete mít magic_quotes na ON - vložený řetězec neobsahuje žádné escapovatelné znaky (integery se nemusí dávat to apostrofů), je tedy jen na Vás, abyste si ohlídali vstupní data, jestli obsahují pouze povolené znaky (např. Málokdy potřebujete vložit ?=? nebo ?-?.

kdy to bude fungovat:

přes nastavení serveru to projde vždy, je tedy jen na vás a na vašich skriptech, jak si s tím poradíte.

co s tím:

pokud víte, že potřebujete vypsat právě jednu (nebo víc, záleží na Vás) hodnotu a nikdy jinak, kontrolujte počet řádků ve výsledku, když číslo nebude jakkoliv souhlasit, zalogujte si to.když máte předem daný počet řádků které očekáváte, nepoužívejte while($x = mysql_fetch_*()) ale obyčejnou funkci mysql_result() kde přesně stanovíte, kolik řádků výsledku chcete zobrazovat (omezený počet iterací cyklu for nebo konkrétně vámi definovaná čísla řádků, které se mají vypsat).

Příklad #2: - zobrazení celého obsahu tabulky podruhé

původní dotaz:
```
SELECT nick, email FROM uziv WHERE nick = '{$nick}'
```
použité proměnné:
$nick = řetězec získaný od uživatele třeba stejnou cestou jak v příkladě #1
mnou vložená hodnota proměnné $nick:
```
' or 'a' = 'a ' or 1=1 --
```
výsledný dotaz:
```
SELECT nick, email FROM uziv WHERE nick = '' OR 'a' = 'a'SELECT nick, email FROM uziv WHERE nick = '' OR 1=1 --'
```

Popis:

toto je velmi podobný příklad jako #1 s tím rozdílem, že pokud chcete pracovat s řetězci, musíte je uzavřít do apostrofů, a ty už by neprošly bez újmy skriptem, který escapuje nebezpečné proměnné. Ten by ' přeměnil na \' a výsledek by byl (ve většině případů) prázdný.

co s tím:

jak jsem se již předtím zmínil, pro zamezení tohoto typu SQL injection stačí escapovat vstupní hodnoty - nastavením direktivy magic_quotes_gpc na on zautomatizujete kontrolu vstupních hodnot, přesto ale doporučuji nespoléhat se na cizí opatření.

Příklad #3 - jak vyzrát na skriptem zajištěné zobrazení jediného řádku výsledku

Původní dotaz:
```
SELECT nick, email FROM uziv WHERE nick = '{$nick}' LIMIT 1
```
ve skriptu navíc implementováno vypsání pouze prvního řádku výsledku
Použité proměnné:
$nick = řetězec získaný od uživatele třeba podobnou cestou jak v příkladě #1
Mnou vložená hodnota proměnné $nick:
```
' OR id = [cislo] ORDER BY [sloupec][DESC/ASC] --' OR id = 2 ORDER BY id DESC --
```
Výsledný dotaz:
```
SELECT nick, email FROM uziv WHERE nick = '' OR id = 2 ORDER BY id DESC --'
```

Popis:

zde jde o to dostat námi požadovaný záznam na první místo, nic víc. Problém je v tom že takto zobrazíte pouze jediný záznam a navíc musíte znát jednu z hodnot, která je v požadovaném záznamu, pak musíte zvolit jednu vyšší (nebo nižší) a následně seřadit výsledek buď sestupně nebo vzestupně v závislosti na zvolené hodnotě

Co s tím:

odpověd na otázku ?co s tím? je jednoduchá: téměř nic. Pokud už povolíte uživateli, aby vkládal do vašich SQL dotazů apostrofy a uvozovky, nezmůže s tím skript vůbec nic, pouze zpracuje data vrácená SQL dotazem, pokud útočník upraví skript tak, aby vracel na prvním řádku data která se snaží získat, máte smůlu.Jediná obrana proti tomuto typu útoku je vždy kontrolovat počet řádků výsledku (když znáte předem počet řádků, které má dotaz standardně vracet.

Příklad #4 - použití klauzule UNION pro získaní dat z jiných

tabulek

Použití UNION má (naštěstí) jisté omezení, takže napřed troška teorie

UNION slouží pro spojení výsledků 2 různých dotazů a jejich vrácení skriptu jako 1 výsledek. Z toho už rovnou vyplývá, že nelze míchat datové typy sloupců a sloupce druhého výsledku musí mít stejný počet a stejné názvy, a ?stejný? datový typ. To znamená, že pokud první výsledek (SELECT cislo FROM ?) budete chtít spojit s druhým (SELECT retezec FROM ?), tak to nepůjde.

Proč to nepůjde? Předpokládejme že oba 2 sloupce mají jiný typ i název ? datové typy míchat nemůžete ? bere se vždy datový typ sloupce prvního výsledku a do integeru nemůžete ukládat řetězec, takže z řetězce bude 0. Dále pokud chcete vytáhnout jiný sloupec, tak mu musíte dát ještě jiný alias, aby ho skript mohl vypsat (přece v PHP píšete přesně které sloupce chcete kde vypsat), pokud by se Vám podařilo vytáhnout jiná data, skript by je prostě nevypsal.

POZOR: aktualizace po zkoušení MySQL 4.1: výše zmíněná omezení datového typu v MySQL 4.1 NEEXISTUJÍ, můžete sloupce míchat "dle libosti"

Původní dotaz:
```
SELECT nick, email FROM uziv WHERE uid = '{$uid}' LIMIT 1
```
Použité proměnné:
$uid = řetězec získaný od uživatele, použitelný kdekoliv, kde se pracuje s uživateli
Mnou vložená hodnota proměnné $nick:
```
1' UNION SELECT heslo AS nick, nick AS email FROM uziv --
```
Výsledný dotaz:
```
SELECT nick, email FROM uziv WHERE uid = '1' UNION SELECT heslo AS nick, nick AS email FROM uziv --' LIMIT 1
```

Popis:

Tímto SQL dotazem se vybere jeden uživatel a jeho jméno a druhým dotazem se vyberou všichni uživatelé a jejich hesla. Klauzule UNION toto všechno spojí do jediného výsledku, takže pokud toto útočník vloží například do seznamu uživatelů (kde zpravidla bývá cyklový výpis výsledku), tak skript poslušně vypíše podle mě velice citlivá data. Samozřejmě můžete kombinovat jakékoliv sloupce a jakékoliv tabulky

Co s tím?

Nesmíte dovolit útočníkovi vložit řetězec podobný mému příkladu. Pokud se mu to podaří, nedá se s tím zpravidla dělat vůbec nic. Konkrétně pro případ uchovávání hesel doporučuji hashovat hesla například pomocí PHP funkce sha1(), pokud už dojde k ukradení těchto údajů, nesmírně tím znepříjemníte útočníkovi jejich lámání.

To je snad všechno, co byste mohli chtít pro začátek vědět. Zbytek se dá najít v manuálu, chytrých knihách a nebo se nebojte pouzit google ;) ... a nebo zde přiložený diskuzní thread

URL článku: https://security-portal.cz/clanky/sql-injection

Odkazy:
[1] https://security-portal.cz/users/pog
[2] https://security-portal.cz/category/tagy/hacking
[3] https://security-portal.cz/category/tagy/hacking-method
[4] https://security-portal.cz/category/tagy/sql
[5] https://security-portal.cz/clanky/sql-injection-full-paper