ZNEUŽITÍ WWW APLIKACE K ÚTOKU NA DATABÁZI

Autor článku: **Martin Mačok – DCIT, s.r.o.**
http://www.dcit.cz

Článek zveřejněn v časopise **Data Security Management** 2/2004
http://www.dsm.tate.cz

Provozujete WWW aplikaci? Pracuje tato aplikace s databází? Jste si jisti, že znáte všechna rizika a databáze je v bezpečí? Pokud jste ještě nikdy neslyšeli o útocích typu SQL injection, existuje reálné riziko, že právě vaše databáze je ohrožena. Pro tuto domněnku hovoří i skutečnost, s jakou frekvencí se s tímto závažným problémem setkáváme během praxe.

Úvod

O nutnosti zabezpečení systémů a aplikací vystavených na Internetu asi nebude nikdo z čtenářů pochybovat. Stejně tak je asi zbytečné dlouze diskutovat o existenci hackerů a hrozbách, které představují – vždyť s téměř železnou pravidelností zaznamenáváme více či méně medializované bezpečnostní incidenty, které ve skutečnosti tvoří pouze zlomek uskutečněných průniků. Databáze chyb softwarových produktů renomovaných komerčních i nezávislých producentů se rostoucím tempem plní novými a novými publikovanými bezpečnostními slabinami, které nemilosrdně postihují téměř všechny systémy bez ohledu na renomé výrobce či cenu produktu.

Tento článek je však zaměřen na jeden velmi specifický okruh bezpečnostních slabin, které se poněkud vymykají výše uvedenému konceptu. Z našich poměrně bohatých praktických zkušeností považujeme za jedny z nejzáludnějších a nejzákeřnějších problémů chyby ve WWW aplikacích či jiných systémech vyvíjených na zakázku.

V čem jsou chyby ve WWW aplikacích zajímavé?

- Jelikož jsou WWW aplikace ve většině případů softwarová díla na zakázku, obsahují chyby, které jsou s trochou ironie rovněž neopakovatelné a „na zakázku“;
- vesměs se jedná se o jedinečné chyby programátora (vlastní zaměstnanec či pracovník dodavatele), které se pochopitelně neobjeví v žádné z uznávaných databází publikovaných bezpečnostních slabin;
- naprosto logicky tyto problémy neodhalí žádný automatický vulnerability scanner obsahující sebevětší databázi všech globálně publikovaných slabin, naopak výsledek prověření takovým scannerem často pouze vytvoří jakousi falešnou iluzi bezpečí (i když existují nástroje, které vyhledávání podobných slabin částečně automatizují a usnadňují);
- z podobných důvodů se nelze spoléhat ani na IDS (Intrusion Detection System), který je navíc v typické situaci předřazen WWW serveru a tudíž často ani nevidí obsah přenášených dat uvnitř šifrovaného protokolu HTTPS a nemůže jej tedy kontrolovat.

Pro potenciálního útočníka ovšem nebude ani tak zajímavá samotná WWW aplikace, jako spíše databáze (resp. data v ní uložená), která se za WWW aplikací skrývá. Pro řadu lidí to bývá překvapení, ale k průniku do databáze stačí v některých případech pouhý jeden jediný otevřený port – např. šifrovaný přístup k WWW aplikaci (443/tcp) – zde vám ani sebelepší firewall nepomůže, neboť komunikace s tímto portem povolena být musí.

Technik zneužívajících chyby ve WWW aplikacích existuje celá řada a vesměs jsou založeny na manipulaci s parametry dynamických stránek, cookies či jinými vstupy. V dalších kapitolách bude podrobněji popsána metoda útoku „SQL injection“ a některé její variace.

Na čem je založen SQL injection útok?

Nejprve popišme typickou situaci, v níž se problém může vyskytovat. Předpokládejme, že provozujeme WWW aplikaci – WWW server s dynamickým obsahem, který přímo závisí na datech uložených v databázi. Uživateli z Internetu je (v ideálním případě) dostupný pouze frontend aplikace – WWW server, s kterým uživatel komunikuje protokolem HTTP(S) prostřednictvím svého WWW prohlížeče. Pokud WWW aplikace obdrží žádost od uživatele, sestaví z jejích relevantních částí SQL dotaz na databázový server – backend aplikace a následnou odpověď od databázového serveru aplikace zpracuje do podoby HTML stránky, kterou odešle zpět uživateli.

Pro další pochopení problému je vhodné jej demonstrovat na příkladu. Předpokládejme, že vstup do WWW aplikace je chráněn uživatelským jménem a heslem, které musí uživatel vyplnit a odeslat, aby mu byl umožněn další přístup do aplikace. Uživatel ve svém WWW prohlížeči vyplní přihlašovací formulář a odešle jej na WWW server. Ten z uživatelského jména a hesla sestaví SQL dotaz a pošle jej databázovému serveru, aby ověřil, zda-li se daná kombinace uživatelského jména a hesla v databázi vyskytuje a zda-li tedy má uživateli umožnit další přístup do aplikace. SQL dotaz bude mít zhruba následující podobu:

```
SELECT * FROM users WHERE login = '$user' AND password = '$password'
```

(pozn. barevné zvýraznění proměnných je podstatné pro dále uvedené příklady)

Aplikace nahradí řetězce $user a $password obsahy příslušných proměnných z formuláře, připojí se na SQL server, dotaz odešle a zařídí se dle přijaté odpovědi. Zdá se, že v tomto případě je vše v pořádku – pokud je kombinace neplatná, databázový server vrátí prázdnou odpověď a uživatel je odmítnut. Bohužel, opak je pravdou. Útočník může i v této situaci zvolit takovou kombinaci uživatelského jména a hesla, která se v databázi nevyskytuje a přesto se s ní do aplikace úspěšně přihlásí!

Problém spočívá ve způsobu vytváření SQL dotazu. Pokud útočník do proměnné $user vloží znaky se speciálním významem, může změnit podobu SQL dotazu způsobem, který autor aplikace nepředpokládal. V našem typickém případě je tímto kouzelným znakem apostrof.

Vyplní-li útočník například:

→ jako uživatelské jméno ..... `admin';#`

→ jako heslo ..... `cokoliv`

výsledný dotaz potom bude mít podobu:

```
SELECT * FROM users WHERE login = 'admin';#' AND password = 'cokoliv'
```

a je ekvivalentní dotazu:

```
SELECT * FROM users WHERE login = 'admin'
```

Pokud tento uživatel existuje, odpoví databáze stejným způsobem, jako by bylo zadáno správné heslo – útočník se tak může snadno přihlásit k aplikaci jako libovolný uživatel,

aniž by znal jeho heslo. Pokud se útočníkovi nepodaří uhodnout ani uživatelské jméno, může zkusit jiný trik a vyplnit:

→ jako uživatelské jméno ..... **ahoj' OR 1=1;#**

→ jako heslo ..... **x**

výsledný dotaz potom odpovídá výrazu:

SELECT * FROM users WHERE login = 'ahoj' OR 1=1;#' AND password = 'x'

tento je ekvivalentní:

SELECT * FROM users WHERE login = 'ahoj' OR 1=1

Databáze v tomto případě vrátí aplikaci celou tabulku uživatelů, což aplikace v typickém případě pochopí jako úspěšné ověření uživatele (odpověď byla neprázdná) a uživatele přihlásí jako prvního z tabulky. Tato skutečnost může být pro útočníka o to zajímavější, že první uživatel v tabulce může být uživatel s vyššími privilegii, kterého vytvořil administrátor pro ověření funkčnosti aplikace před nasazením do ostrého provozu a nebo který sloužil jako testovací uživatel při vývoji aplikace. Z tohoto příkladu je i zřejmý název tohoto typu útoku – útočník do dat vkládá část SQL kódu, který bude součástí SQL dotazu kladeného aplikací databázovému serveru, odtud SQL injection.

Obvykle získává útočník možnost upravovat klauzuli WHERE v SQL dotazu. Nejčastějšími triky jsou

- vložení OR 1=1 resp. AND 1=0 způsobí, že je podmínka nebo její část vždy pravdivá resp. nepravdivá;
- vložení --, /* nebo ;# předčasně ukončí příkaz;
- použití UNION umožní pokládat další libovolné dotazy;
- vkládání poddotazu.

Amputace podmínky je vhodná především pro testování přítomnosti této chyby v aplikaci, ale jak bylo vidět na předešlém příkladě, může umožnit obejití autentizace či vylistování většího počtu záznamů z databáze.

Vložením UNION lze dotaz proměnit ve sjednocení dvou dotazů – původního a de facto libovolného dalšího. Jedná však o poměrně náročnou techniku, neboť je ale třeba zajistit správné počty a datové typy sloupců v původním i „novém“ dotazu připojovaném klauzulí UNION – útočník toho obvykle dosáhne metodou pokus-omyl.

Pro „nekalou“ činnost je vhodné znát schéma databáze a další informace (tzv. metadata). Pokud to není veřejné tajemství (obecně známá aplikace nebo její klon), pak je třeba je zjistit, k čemuž mohou pomoci následující triky:

- V Oracle databázi existují systémové tabulky ALL_TABLES a spol., případně specialitka ALL_SOURCE se zdrojovými kódy;
- v MS SQL jsou to tabulky sysobjects apod.;
- některé DB servery jako například MySQL podobné tabulky obecně neobsahují, potom nezbývá než hádat.

V mnoha případech však může WWW frontend útočníkovi značně ušetřit práci tím, že uživateli vrátí podrobný popis chyby, pokud tato nastane v průběhu zpracování SQL dotazu. Tyto podrobné výpisy mnohdy obsahují jména tabulek a sloupců, umožňují odhalit typy dat v příslušných sloupcích a mnohdy i podobu SQL dotazů, do kterých útočník vkládá svůj vstup. V MS SQL lze pomocí dynamicky generovaných chybových hlášení o chybách při typové konverzi (z textu na číslo) získávat data, protože hlášení obsahuje doslovný opis problematické hodnoty.

Někdy je informace o chybě při zpracování velmi strohá, např. číslo řádku, na němž nastala chyba. I to však může vést k fatálním následkům – setkali jsme se v praxi

s případem, kdy různé vstupy vedly k chybám na dvou různých řádcích v kódu aplikace. V tomto případě se podařilo zjistit, že chyba na řádku 16 znamenala nesplnění určité části dotazu a chyba na řádku 17 opak.

I tato nevinně vyhlížející skutečnost v některých případech umožní útočníkovi vytvořit tzv. „orákulum“ schopné vyhodnotit v databázi libovolný logický výraz dávající výsledek „ano“ (= chyba na řádky 17) – „ne“ (= chyba na řádku 16).

Takto lze, i když poněkud nepohodlně a pomalu (hrubou silou), zjistit prakticky libovolná data z databáze, a to některým z následujících způsobů:

- Opakovaným použitím relací „je větší“ a „je menší“ v dotazech a metodou půlení intervalu tak bylo možné postupně získat libovolnou položku (řádek) z databáze.
- Získání většího množství dat lze např. masovou sérií postupných dotazů typu: „existuje OSOBA u které atribut PRIJMENI začíná: a, b, …, aa, ab, ac, … , aaa, aab, …?“

Tyto a jiné podobné metody tak lze prakticky zneužít k „vyloupení“ obsahu celé databáze.

Vyhodnocování výrazu v podmínce může mít při použití vhodných vestavěných či knihovních funkcí i zajímavé vedlejší účinky. Mnoho běžně používaných databází totiž disponuje funkcemi umožňujícími např. zjišťovat informace o verzi software, aktuálního uživatele atd. Některé funkce mohou obsahovat i vlastní slabiny ve zpracování parametrů (např. buffer overflow): OPENDATASOURCE v MS SQL a jiné. Nebezpečné jsou i funkce umožňující spouštět příkazy OS (např. xp_cmdshell v MS SQL a utl_file, utl_smtp v Oracle) nebo ukládat výsledky dotazů do souborů na disku. Útočník by mohl například vložit dávkový (případně i binární) soubor do položky databáze (může si pro tento účel případně i vytvořit novou tabulku), obsah příslušného pole uložit do souboru na disk na serveru a knihovní funkcí dávku či program spustit.

Tímto způsobem by mohl získat kontrolu nad celým serverem, na němž je SQL server provozován. Tato trofej může být pro útočníka o to cennější, že na rozdíl od WWW frontendu aplikace provozované na WWW serveru v DMZ (demilitarizované zóně) bývá databázový server často umístěn hluboko v intranetu a v těsné blízkosti jiných atraktivních serverů.

V některých případech je však zneužití chyby velmi komplikované, nikoliv však nemožné. Uživatelské vstupy mohou být vkládány do složitě uzávorkovaných výrazů, potom může být vytvoření smysluplného SQL dotazu netriviální. Jindy je vynuceno použití UNION ALL SELECT nebo jsou problémy s kombinacemi UNION a ORDER BY. Některé dotazy není možno uměle ukončit komentářem (např. víceřádkové dotazy v Oracle) nebo obsahují podivné datové typy (LONG v Oracle). Jindy je vstup vkládán do více různých příkazů a je tak omezen prostor pro jeho smysluplnou (a syntakticky správnou) deformaci. V některých případech je potlačen chybový výstup z databáze a útočník je nucen provádět útok tzv. naslepo. Všechny tyto komplikace mohou útočníkovi značně ztížit práci a některé ho mohou i odradit, pokud ale není vstup dostatečně ošetřen a je možné upravovat SQL dotaz, téměř vždy to může vést k závažným následkům.

Kde všude lze SQL injection použít?

Problém se může vyskytovat na všech místech v aplikaci, ve kterých je k sestavení SQL dotazu používán text z příchozích dat od uživatele z Internetu – tedy dat, nad nimiž nemáme kontrolu a je nutno je považovat za nedůvěryhodná.

V nejjednodušších případech, které byly demonstrovány v předchozí části, jsou tímto kritickým místem právě položky formulářů. Často se problém vyskytuje již v přihlašovacím dialogu, kde je zřejmě nejvíce nebezpečný – útočník jej může snadno zneužít díky typicky jednoduchému SQL dotazu (jehož podobu může snadno odhadnout), nemusí ani znát přístupové heslo k aplikaci a útok může provést pomocí libovolného běžně používaného WWW prohlížeče. Dalším běžně se vyskytujícím formulářovým vstupem bývá fulltextový vyhledávač.

Je však nutno si všímat nejenom přímo viditelných položek formulářů, které uživatele explicitně vybízejí k vyplnění textu. Formulář může obsahovat i různé skryté parametry, které může útočník objevit inspekcí zdrojového (HTML) kódu formuláře. Navíc mohou existovat i jiné skryté parametry, které nejsou standardně formulářem odesílány, přesto je však může příslušná aplikace přijmout a zpracovat do SQL dotazu. Takové parametry typicky náhodný útočník neobjeví, ale nelze na to spoléhat – jednak může mít přístup ke zdrojovému kódu (pokud se například jedná o klon volně dostupné aplikace) anebo může mít štěstí a jméno proměnné uhodnout (například slovníkovým útokem). Chyby v těchto parametrech typicky nelze zneužít pomocí obyčejného prohlížeče. Pro zkušeného útočníka to však nepředstavuje komplikaci, neboť existují různá speciální rozšíření prohlížečů, specializované (často velmi jednoduché) programy nebo nástroje typu HTTP proxy, které snadno umožňují libovolné modifikace odchozích žádostí případně i příchozích odpovědí.

Často se opomíná ošetřovat vstup z parametrů, které uživatel přímo nevyplňuje, ale pouze vybírá z několika nabízených možností. Pokud je v SQL dotazu přímo použit výsledný text, i zde může být osudové, pokud aplikace tiše předpokládá, že přijatá hodnota parametru bude obsahovat pouze text z nabídnutých možností.

Mimo formulářů tímto problémem často trpí i parametry dynamicky generovaných stránek, které závisí na hodnotách parametrů HTTP dotazu. Příkladem může být třeba číslo požadovaného dokumentu – například *http://server/clanek.php?id=123* – zde bude obsah parametru id jistě použit v SQL dotazu a jeho šikovnou modifikací může útočník dotaz upravit. V případě očekávaných číselných hodnot často ani není hodnota uzavírána do apostrofů či uvozovek a ke zneužití postačí vložit obyčejnou mezeru. Dynamicky generované stránky často obsahují velké množství parametrů – barva pozadí, téma/skin vzhledu, znaková sada apod. (často se ani celé URL nevejde na jeden řádek obrazovky) a stejně jako u formulářů, i zde mohou být další skryté parametry, které nejsou v běžné situaci vidět, ale aplikace na serveru je přijme a zpracuje.

Zcela samostatnou kapitolou jsou možnosti manipulace s URL v případě některých aplikačních middleware, které sice vstupní parametry hlídají velmi důkladně nicméně často využívají složitě konstruovaná URL typu:
https://server.cz/web/x/y/package.proc/go?a=1&b=2 přičemž:

- V některých případech (např. Oracle WebDB) je část URL přímo jménem volané uložené procedury (v našem příkladě „package.proc“).
- Pokud útočník tyto nečekané možnosti „prohlédne“ nabízí se mu obvykle velmi pestrý sortiment standardních procedur typu „util.dsql“ (dynamické SQL – možnost libovolného dotazu příp. dokonce spuštění vlastního PL/SQL kódu) nebo „utl_file“ (přístup na filesystem DB serveru či „utl_tcp“ (možnost vytvářet TCP spojení z DB serveru do hlubin vnitřní sítě).
- Pochopitelně přístupové právo EXECUTE pro PUBLIC u těchto nebezpečných procedur není problémem samotného aplikačního serveru, nýbrž správce resp. dodavatele, který aplikační server konfiguroval. Samozřejmě můžeme polemizovat, proč jsou práva takto nevhodně nastavena „by default“.

Dalším často opomíjeným uživatelským vstupem jsou hodnoty cookies. Aplikace do nich často ukládá různé stavové informace (např. takzvané session ID), které jsou rovněž ukládány do databáze a později vyhledávány. Vzhledem k typickému použití cookies tvůrce aplikace nepředpokládá, že by uživatel cookies měnil, neboť by z toho patrně neměl žádný užitek, spíše naopak – nemohl by se do aplikace přihlásit.

Jak se útokům typu „SQL injection“ bránit

K odstranění slabiny je nutné identifikovat a odstranit její skutečnou příčinu. Tou je v tomto případě přílišná důvěra a z toho vyplývající nedostatečná kontrola uživatelských vstupů, které jsou následně vkládány do SQL dotazů. Chyba je tedy ve frontendu aplikace (nejčastěji ASP nebo PHP skriptech), který nedostatečně chrání (filtruje) přístup k backendu aplikace – databázi. Přímočarým řešením je explicitní zakázání všech potenciálně nebezpečných znaků ve všech uživatelských vstupech – těmito znaky jsou především apostrof, uvozovky, mezera, středník, zpětné lomítko, dopředné lomítko, hvězdička aj. Ze strategického hlediska však doporučujeme volit spíše opačný postup – explicitní povolení bezpečných znaků a odmítnutí všech ostatních. Tímto způsobem se dá vyhnout opomenutí nějakého speciálního znaku a navíc jednou ranou omezit i možnost mnoha jiných typů útoků.

Číselné vstupy by tak měly obsahovat pouze numerické znaky, uživatelská jména alfanumerické znaky (případně tečku apod.). U položek typu heslo je vhodné (opatrně) povolit i jiné další znaky, aby aplikace neodmítala kvalitnější hesla a nevynucovala pouze jednoduchá. Všechny běžně používané programovací jazyky již obsahují vestavěné (knihovní) funkce použitelné k těmto kontrolám a není tedy nutné je znovu programovat.

Kontrolu vstupu je vhodné provádět co nejdříve, nejlépe v okamžiku převzetí vstupu aplikací. Naopak zcela předčasná je kontrola vstupu javascriptem, neboť ten je prováděn ještě na straně uživatele, kterému ale nemůžeme důvěřovat – naopak, musíme od něj čekat to nejhorší. Útočník si javascript vypne nebo použije vlastní implementaci javascriptu, případně data snadno upraví během přenosu.

Pokud vstupní data nevyhovují podmínkám a obsahují i jiné než povolené znaky, nejbezpečnější reakcí je ve většině případů okamžité ukončení výpočtu a vrácení stručné (ne příliš konkrétní) chybové odpovědi (včetně kontaktu na zodpovědnou osobu, aby si případný uživatel mohl stěžovat, pokud byl odmítnut i legitimní vstup). Poněkud riskantnějším řešením je ošetření výskytu těchto znaků pokusem o jejich zneškodnění např. escape sekvencí (zpětným lomítkem apod.), což se ovšem ne vždy ukazuje jako zcela spolehlivé, a proto jednoznačně doporučujeme, pokud je to možné, závadný vstup bez milosti odmítnout.

Jiným poměrně elegantním řešením je odstínění aplikace od databáze pomocí uložených procedur. Výhodou je přenesení zodpovědnosti za chyby z programátorů frontendu na programátory databáze a především typicky výrazně menší citlivosti tohoto řešení na zlomyslně poškozený vstup od uživatele.

Dále je vhodné, aby v případě, kdy při zpracování nastane chyba v aplikaci nebo databázi, nebyly navenek viditelné detaily a příčiny těchto chyb. Chybové hlášky často obsahují příliš mnoho citlivých informací a výrazně usnadňují útočníkovi práci. Podrobnosti chyb by měly být zůstat uloženy na lokálních systémech a neměly by být exponovány zpět do Internetu. Uživatel by měl být stručně informován o blíže neurčené chybě a odkázán na technickou podporu.

Univerzálním doporučením je rovněž provoz databáze pod pokud možno nejméně privilegovaným uživatelem, odstranění všech potenciálně citlivých dat ze serveru a z databáze, která nejsou přímo nutná k provozu aplikace a nakonec i umístění databázového serveru pro Internetové služby do DMZ.

Závěr

Jak je vidět, pomyslná stěna mezi vašimi daty a „divokým“ Internetem může být v některých případech nečekaně tenká. Z naší zkušenosti víme, že v případě problematické WWW aplikace bývá pro motivovaného útočníka s dostatečnou technickou znalostí zneužití slabiny smysluplným způsobem otázkou jednotek hodin či dní.

Navíc tak, jak tyto specifické („na zakázku“) chyby WWW aplikací nebývají obsaženy v databázích celosvětově známých bezpečnostních slabin, nejsou ani přesné vzorky těchto útoků obsaženy v databázích IDS či podobných systémů – třebaže existují jisté charakteristiky, podle kterých lze útoky typu SQL injection v mnoha případech identifikovat.

Ačkoliv nemůžeme být zcela konkrétní v našich praktických zkušenostech, věřte, že problémy, které jsou v tomto článku diskutovány se nevyhýbají ani vysoce renomovaným institucím a navíc postihují i řešení velmi zavedených dodavatelů.

Literatura:

[1] Jody Melbourne, David Jorm: *Penetration Testing for Web Applications*, *http://www.securityfocus.com/infocus/1709*

[2] Mgr. Pavel Kaňkovský: prezentace *SQL injection & spol.*, Prosinec 2003

[3] Ofer Maor, Amichai Shulman: *Blindfolded SQL injection*, WebCohort Inc., 2003

[4] Cesar Cerrudo: *Manipulating Microsoft SQL Server Using SQL injection*, Application Security, Inc., 2002