CVE-2012-5611, EXPLOITATION MÉMOIRE DU SGDB MYSQL

Samir MEGUEDDEM (@ipv_) – Synacktiv – samir.megueddem@synacktiv.com

mots-clés : CVE-2012-5611 / KINGCOPE / RCE / STACK BUFFER OVERFLOW / GRANT STATEMENT

Le 1er décembre 2012, un hacker allemand du nom de Kingcope fait de nouveau parler de lui en diffusant une série de vulnérabilités [DISCLO] sur le système de gestion de base de données MySQL. D'après Kingcope, ces vulnérabilités permettent potentiellement d'exécuter du code à distance, à condition de disposer d'un compte utilisateur non privilégié.

1 Introduction

MySQL est le moteur de base de données *open source* le plus déployé **[MARKET]**. Son succès repose sur son accessibilité (gratuit et documenté), sa facilité de déploiement (multi-plateforme), sa flexibilité (petites ou grandes bases de données), et ses nombreuses fonctionnalités (richesse de l'API MySQL, sous-requêtes, procédures stockées, etc.).

Du point de vue de l'attaquant, MySQL est une cible bien connue pour plusieurs raisons :

- la popularité du produit dans les environnements web ;
- une API riche, qui facilite l'évasion des filtres sécurité lors des injections SQL ;
- une configuration souvent perfectible (ACL utilisateur, processus avec les droits administrateurs, etc.).

Avec le rachat de MySQL par Oracle, certains développeurs ont décidé de créer MariaDB, un *fork* du projet MySQL. Une vulnérabilité sur MySQL aura de fortes chances d'être également présente sur MariaDB (et vice-versa).

2 Analyse du CVE-2012-5611

2.1 Lancement de la preuve de concept

L'analyse du CVE-2012-5611 démarre par la preuve de concept fournie par Kingcope. Puisque la vulnérabilité nécessite une authentification au serveur MySQL, l'attaquant doit disposer d'au moins un compte non privilégié sur le SGDB. Pour un développeur d'*exploit*, la première étape est de reproduire le *bug* :

```
my $dbh = DBI->connect("DBI:mysql:database=test;host=192.168.2.3;","user","
secret", {'RaiseError' => 1});
$a ="A" x 100000;
my $sth = $dbh->prepare("grant file on $a.* to 'user'\@'%' identified by
'secret';");
```

Une analyse rapide permet de mettre en cause le nom de la base de données. Dans ce code, la commande **GRANT** attribue la permission **FILE** sur une longue chaîne représentant le nom de la base de données. Cette permission s'applique uniquement à l'utilisateur **user**, identifié par le mot de passe **secret**.

Aussi surprenant que cela puisse paraître, cela déclenche le plantage immédiat du SGBD MySQL :

```
Program terminated with signal 11, Segmentation fault.
  #0  0x41414141 in ?? ()
```

Dans le cas d'un binaire compilé avec le SSP (*Stack Smashing Protector*), le débordement de tampon est détecté :

```
**** stack smashing detected ****
```

La recompilation en mode *debug* et l'approche « pas à pas » permettent d'incriminer la fonction **acl_get** :

```
gdb$ bt
#0 0x08260b1c acl_get (host=0x0, ip=0x892aff0 "192.168.130.1", user=0x8940d58
"test_mysql",      db=0x895e808 'a'<repeats 200 times>..., db_is_pattern=0x1)
at sql_acl.cc:1194
#1 0x081d7301 in check_access (thd=0x89277d0, want_access=0x1e73c3f,
db=0x895e808 'a' <repeats    200 times>..., save_priv=0xb4f6d3dc, dont_check_
global_grants=0x1,       no_errors=0x0, schema_db=0x0) at sql_parse.cc:5756
#2 0x081d3ab2 in mysql_execute_command (thd=0x89277d0) at sql_parse.cc:4441
```

```
#3 0x081d87e7 in mysql_parse (thd=0x89277d0, rawbuf=0x895b7d8 "grant ALL on ",
'a' <repeats 187   times>..., length=0x1b90, found_semicolon=0xb4f6e198) at
sql_parse.cc:6497
#4 0x081cd4db in dispatch_command (command=COM_QUERY, thd=0x89277d0,
packet=0x89577a9    "grant ALL on ", 'a' <repeats 187 times>..., packet_
length=0x1b91) at sql_parse.cc:1989
#5 0x081ccabd in do_command (thd=0x89277d0) at sql_parse.cc:1657
#6 0x081cbdaa in handle_one_connection (arg=0x89277d0) at sql_parse.cc:1242
#7 0xb7f9396e in start_thread () from /lib/tls/i686/cmov/libpthread.so.0
#8 0xb7d7ca4e in clone () from /lib/tls/i686/cmov/libc.so.6
```

Dans cette *backtrace*, nous pouvons identifier la fonction **clone** qui est appelée à chaque nouvelle connexion d'un client. Cette routine crée un nouveau *thread* dont le pointeur d'instruction est initialisé à l'adresse spécifiée en paramètre. Contrairement à un **fork**, plusieurs parties de la mémoire sont partagées avec le parent, cependant il dispose de son propre contexte d'exécution et de sa propre pile. Les fonctions **handle_one_connection** et **do_command** vont initialiser des paramètres globaux et analyser les informations transmises par le client. À cette étape, ces fonctions traitent les données liées au protocole d'échange MySQL (envoyées de manière transparente par le client MySQL).

La fonction **dispatch_command** se chargera d'appeler la bonne routine en fonction de la commande envoyée par le client. L'analyse syntaxique de la requête démarre à partir de la fonction **mysql_parse**. Tout comme **dispatch_command**, la fonction **mysql_execute_command** se charge d'appeler le composant principal de la requête (ici **GRANT**, une commande affiliée à la gestion des comptes de MySQL).

Enfin, avant d'arriver dans notre fonction vulnérable, la fonction **check_access** est sollicitée pour vérifier la légitimité de l'utilisateur qui appelle la commande **GRANT** (système d'ACL).

L'analyse et le traçage du flux d'exécution nous permettent de mieux cerner le contexte d'exécution dans lequel se trouve MySQL au moment d'atteindre la fonction vulnérable **acl_get**.

À ce stade, nous constatons un premier problème qui pourrait être gênant pour réaliser une exécution de code arbitraire. Lorsque nous inspectons notre nom de base de données, il se compose uniquement de caractères alphanumériques. Tous les autres caractères déclenchent une erreur de syntaxe. Nous en prenons bonne note et reviendrons dessus plus tard.

2.2 Analyse de la fonction vulnérable

Le fichier **sql_acl.cc** contient la fonction **acl_get** qui semble être au cœur de la vulnérabilité. Cette fonction démarre par le code suivant :

```
ulong acl_get(const char *host, const char *ip,
              const char *user, const char *db, my_bool db_is_pattern)
{
  ulong host_access= ~(ulong)0, db_access= 0;
  uint i,key_length;
  char key[ACL_KEY_LENGTH],*tmp_db,*end;
  acl_entry * entry;
  DBUG_ENTER("acl_get");

  VOID(pthread_mutex_lock(&acl_cache->lock));
  end=strmov((tmp_db=strmov(strmov(key, ip ? ip : "")+1,user)+1),db) ;
[...]
```

Ce code (dont vous remarquerez le peu d'esthétisme) déclare des variables locales, puis imbrique plusieurs appels à la directive **strmov**. Pour plus de brièveté, les développeurs ont jugé bon d'utiliser une condition ternaire.

La macro **strmov** est déclarée de la manière suivante :

```
#define strmov(A,B) stpcpy((A),(B))
```

Après analyse, le code peut être représenté par les lignes suivantes :

```
p_key = strcpy(key, ip ? ip : "");
p_db = strcpy(p_key + 1, user);
p_end = strcpy(p_db + 1, db);
```

Le code crée une clé de cache pour optimiser l'appel à cette fonction. La clé est la concaténation de l'adresse IP, du nom d'utilisateur, et du nom de la base de données.

La taille du tampon est définie par **ACL_KEY_LENGTH** :

```
#define ACL_KEY_LENGTH (IP_ADDR_STRLEN+1+NAME_LEN+1+USERNAME_LENGTH+1)
#define IP_ADDR_STRLEN (3+1+3+1+3+1+3)
[...]
#define NAME_LEN        64             /* Field/table name length */
#define USERNAME_LENGTH 16
```

Une allocation de 98 octets est réalisée sur la pile pour le tampon de destination nommé **key**. Aucune vérification n'est réalisée sur la taille de la source, et la *stack-frame* de la fonction **acl_get** peut être réécrite, ce qui impacterait le flux d'exécution du programme.

À ce stade, nous pouvons tirer plusieurs observations :

- La vulnérabilité nécessite un compte utilisateur (non privilégié).
- Le nom de la base de données doit a priori être composé uniquement de caractères alphanumériques.
- La fonction **acl_get** réalise des appels à des sous-fonctions qui pourraient perturber le flux d'exécution après réécriture de la stack-frame.
- Certaines distributions (par exemple Ubuntu) compilent le projet MySQL avec SSP (Stack Smashing Protector), ce qui stoppe l'exécution du programme lors de la sortie de la fonction **acl_get** en cas de débordement de tampon.

En disposant de ces informations, nous pouvons faire planter le processus MySQL en spécifiant une chaîne d'environ 128 octets. Cette valeur varie en fonction de la version et des paramètres de compilation utilisés. Cependant, l'attaquant dispose d'un compte, il peut donc identifier la version de MySQL utilisée afin de fiabiliser son exploit :

```
mysql> SELECT @@version,@@version_comment,@@version_compile_machine,@@version_
compile_os;
+-------------------------------+-------------------------+---------------
---------------------+--------------------------+
| @@version                   | @@version_comment | @@version_compile_machine|@@
version_compile_os |
+-------------------------------+-------------------------+---------------
---------------------+--------------------------+
| 5.5.29-0ubuntu0.12.10.1 | (Ubuntu)                | x86_64
| debian-linux-gnu         |
+-------------------------------+-------------------------+---------------
---------------------+--------------------------+
```

Il est également possible de récupérer le contenu de la variable `@@version` via le `Server Greeting` émis par le service MySQL lors de la réception d'une connexion d'un client.

2.3 Exploitation

2.3.1 Contournement du jeu de caractères via UTF-8

Comme indiqué dans la documentation de MySQL **[IDENTIFIER]**, le nom des objets tels qu'une base de données, une table ou une colonne, est nommé « identifiant ». En interne, un identifiant est représenté par une chaîne Unicode encodée en UTF-8 et celui-ci est soumis à des restrictions particulières. Le tableau suivant illustre ces restrictions :

Sans guillemet	Avec guillemet
a-z A-Z 0-9 $ _ U+0080 - U+FFFF	U+0001 - U+007F U+0080 - U+FFFF

Figure 1 : Caractères autorisés dans les identifiants

Les identifiants utilisent donc uniquement le jeu de caractères Unicode encodé par UTF-8 qui se traduit grossièrement par le schéma suivant :

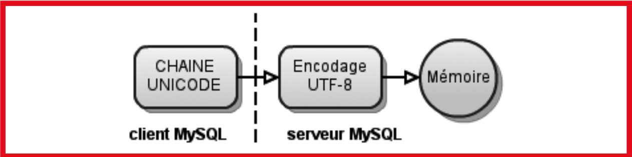

Figure 2 : Schéma d'encodage

L'encodage UTF-8 **[UTF8]** peut être simplifié en deux parties :

- l'encodage de caractères sur un octet (table US-ASCII, U+0000->U+007F) ;
- l'encodage de caractères sur plusieurs octets (nommé *multi-bytes sequence*).

Chaque encodage UTF-8 transpose un simple caractère en un ou plusieurs octets. L'encodage UTF-8 d'un seul symbole peut atteindre jusqu'à six octets (séquence multi-octets). Néanmoins, un caractère d'identifiant doit être dans l'espace U+0001 jusque U+10000. Ce qui signifie que l'encodage UTF-8 de MySQL est d'au moins un octet et peut atteindre au maximum trois octets.

Pour tirer profit de l'encodage, il est nécessaire de modifier les outils de ROP pour inclure des conditions sur les gadgets trouvés. Pour valider l'adresse d'un gadget vis-à-vis de l'encodage UTF-8, il est possible d'intégrer un vérificateur dans le générateur ROP.

Voici deux exemples de règle :

- Si chaque octet de l'adresse est plus petit que 0x80, l'adresse est valide.
- Si les octets LSB (*Least Significant Byte*) de l'adresse sont plus grands que 0x80 et identiques aux octets LSB d'une séquence multi-octets, alors ils peuvent servir à construire une adresse valide.

MySQL 5.0.96 Ubuntu 10.04		
Chaine python	*Encodé en UTF-8*	*Gadget*
"\x23\x7d\x06\x08"	\x23\x7d\x06\x08	0x08067D23 = push esp; ret;
"AAA"+unichr(2844)+"\x06\x08" unichr(2844) = [illegible]	AAA\xe8\xac\x9c\x06\x08	0x08069CAC = sysenter

Figure 3 : Exemples d'adresses ROP valides en UTF-8

Il est donc possible de créer une table de correspondance UTF-8, de l'intégrer dans un générateur ROP et d'appliquer les règles pour vérifier la validité de l'adresse d'un gadget.

2.3.2 Contournement SSP via le TCB

Deux notions s'opposent lorsqu'il s'agit de traiter des ressources systèmes. Le *Forking Model* (utilisé par exemple par VSFTPd ou Apache2-prefork) et le *Threading Model* (utilisé par exemple par MySQL ou Apache2-threaded). MySQL utilise un système de *thread* pour gérer ses clients, cela au travers de la bibliothèque `pthread` qui s'appuie sur l'appel système `clone`.

Plusieurs dénominations existent pour l'acronyme TCB dans la gestion des processus. Il définit ici le *Thread Control Block* **[TCB]**, une structure créée à chaque nouveau thread, et dont voici les champs les plus importants :

- Un pointeur vers le *Dynamic Thread Vector* (structure pointant vers les blocs TLS (*Thread Local Storage*) des *Dynamic Shared Objects*).
- Un pointeur nommé `sysinfo` qui pointe vers les routines noyau exportées dans la section VDSO. Celle-ci contient l'instruction `sysenter`.

```
0xb7fe2420 <__kernel_vsyscall>: push   ecx
0xb7fe2421 <__kernel_vsyscall+1>: push   edx
0xb7fe2422 <__kernel_vsyscall+2>: push   ebp
0xb7fe2423 <__kernel_vsyscall+3>: mov    ebp,esp
0xb7fe2425 <__kernel_vsyscall+5>: sysenter
```

- Le *cookie* SSP (nommé `stack_guard`) comparé lors de l'épilogue à celui sauvegardé dans la pile (après le `saved ebp`) durant le prologue.

Par défaut, le TCB est dans une zone mémoire *mmaped* bien distincte (en lecture-écriture et non contiguë à la pile principale). Néanmoins, les nouveaux threads sont créés dans une région mmaped dédiée aux clients contenant à la fois le TCB et les variables locales de toutes les fonctions du backtrace précédent.

Lorsqu'un client se connecte (fonction `run_server_loop` du fichier `tools/mysql_manager.c`), le serveur MySQL lui attribue un nouveau thread qui continuera l'exécution dans la fonction `process_connection` :

```
pthread_create(&th,&thr_attr,process_connection,(void*)thd)
```

À cette occasion, une nouvelle pile mémoire dédiée au thread est créée. Pour accéder au TCB du thread courant, le code *userland* utilise le sélecteur `gs`.

Pour contourner le SSP, nous allons réécrire le pointeur `sysinfo` utilisé par toutes les fonctions. Puisque le SSP lui-même fera appel à `printf` pour afficher son message d'erreur, nous contrôlerons le flux d'exécution (cet article **[TCBLOCAL]** a également présenté l'astuce sur un binaire en local).

NOTE

Rappel du fonctionnement de la PLT/GOT et du handler VSYSCALL

La Figure 4 présente le déroulement de l'appel de la fonction :

1. **La fonction my_funccontient un appel à la fonction printf.**
2. **La Procedure Linkage Table (PLT) va préparer l'appel à printf en cherchant son adresse dans la Global Offset Table (GOT).**
3. **Si l'adresse de printf n'est pas initialisée dans la GOT (lazy binding), alors la fonction dl_runtime_resolve de la bibliothèque « ld » (le loader) est sollicitée.**
4. **Le loader résout l'appel et l'inscrit dans l'entrée printf de la GOT. Les étapes 3 et 4 ne seront plus réalisées lors des prochains appels à printf.**
5. **La fonction printf est appelée.**
6. **Chaque appel système provenant d'une fonction de la libc est réalisé par les étapes suivantes :**
   - **préparation des registres (numéro du syscall dans eax) ;**
   - **appel du handler VSYSCALL à l'aide de gs ; call DWORD PTR gs:0x10 (l'offset 0x10 correspond au champ sysinfo).**
7. **Le code exécute l'instruction x86 sysenter qui sera traitée par le noyau.**

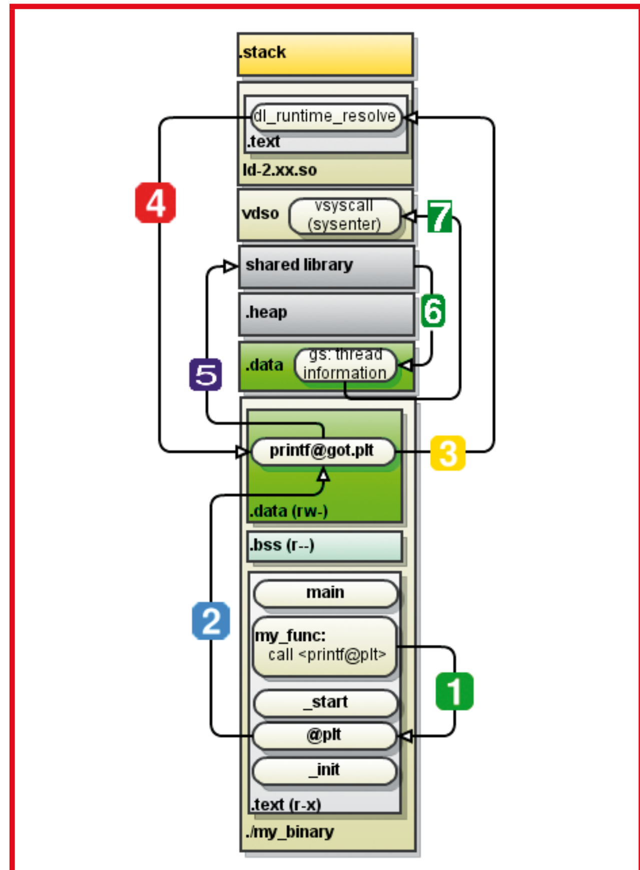

Figure 4 : Rappel du fonctionnement de la PLT/GOT et du handler VSYSCALL

2.3.3 Contournement ASLR via les zones non aléatoires

Pour contourner les zones touchées par l'ASLR (l'ensemble des bibliothèques est soumis à l'aléa), nous profitons des zones mémoire non impactées, `.text` et `.data`, du binaire.

En effet, le projet n'a pas été compilé avec les paramètres `-fpie -fPIE`, de ce fait il est possible de retomber dans des gadgets de la section contenant le code exécutable. À ce titre, l'élaboration de la table d'adresses ROP se fera sur les critères suivants :

- Lorsque nous contrôlons le flux d'exécution, quels registres maîtrisons-nous ?
- Possédons-nous des gadgets pour lire, écrire et exécuter des adresses mémoire ?
- Devons-nous adapter les registres avant l'usage de ces gadgets ?
- Devons-nous faire usage de gadgets temporaires (cas pratique de l'utilisation de `xchg`, `mov`, etc.) ?
- Quelle est la fiabilité de ces gadgets ? Sont-ils soumis à l'ASLR ?

2.3.4 Contournement du bit NX via un ROP-Mprotect

L'approche classique pour contourner le bit NX (pour « No eXecute ») est de réaliser un `ret2libc` **[RET2LIBC]** vers des fonctions utiles (`system`, `execve`, `mprotect`, le couple `mmap`/`read`, etc.). Cette technique requiert un ajustement minutieux des arguments sur la pile qui ne peut pas être appliquée ici.

Il est également possible de réécrire une entrée de la GOT **[GOTCTF]** et de déclencher la fonctionnalité adéquate du programme (contenant des paramètres que l'on maîtrise). Cependant, l'aléa de la libc, ou une éventuelle option de compilation (`full-relro` règle la GOT en lecture seule), ne nous le permettra pas.

Une technique d'exploitation a été présentée en 2009 **[GOTDEREF]** : le *GOT-Dereferencing*. Puisque la GOT est située dans la section `.data` qui n'est pas soumise à l'aléa, il suffit de trouver des gadgets pour extraire un pointeur de fonction de la libc déjà appelé, de calculer la distance vers la fonction recherchée, et enfin de sauter dessus (regarder pour cela l'exploit Wireshark **[IPV]**). De même, en fonction des versions de GCC et de la libc, d'autres approches peuvent être envisagées (réécriture de certains pointeurs sensibles : `.dynamic` **[DYN]**, `.dtors` **[DTORS]**, etc.).

Dans notre cas, nous utiliserons l'appel système `sys_mprotect` après avoir réalisé un ET logique sur une adresse de la pile.

Dans la version Red Hat (sans SSP), nous pouvons extraire le `sysinfo` dans un registre pour l'utiliser comme passerelle vers `sysenter`.

Dans la version Ubuntu, MySQL met aimablement l'instruction `sysenter` à disposition dans la section `.text`. Il suffit donc de mettre un pivot vers notre `ROP-chain` dans le `sysinfo`, et de déclencher l'appel à la routine `_stack_check_fail` de SSP **[EXPLOIT]**.

2.4 Autres voies d'exploitation ?

L'exemple du format string dans **sudo** (CVE-2012-0809), publié début 2012, est assez parlant. Certaines personnes ont fait preuve d'entrain pour son exploitation (exemple de l'excellent article de longld **[LONGLD]**), même après les améliorations de plusieurs fonctionnalités de grsecurity qui se sont avérées être inutiles (pauvre Brad...). Ce qu'il ne faut pas négliger, c'est que l'exploitation d'une vulnérabilité peut être facilitée si des structures de données sensibles sont présentes dans la mémoire. **sudo** en est l'exemple (astuce non présente dans l'article de longld) ;).

Dans le cas de MySQL, il peut être intéressant d'investiguer la réécriture de valeurs sensibles dans la pile qui permettrait d'obtenir les droits administrateur ou la permission `FILE` du SGDB. En enchaînant avec la fonctionnalité d'UDF (*User-Defined Function*), un attaquant arrivera au même résultat : l'obtention d'un *shell*.

Conclusion

L'analyse du CVE-2012-5611 a permis de découvrir et mettre en pratique des techniques atypiques. En adaptant la recherche de gadget, il a été possible de contourner la restriction des caractères afin de placer notre charge utile en mémoire. De même, le modèle de *Threading* nous a fait découvrir une nouvelle technique d'exploitation pour outrepasser SSP. De nos jours, de nombreuses protections userland sur Linux sont présentes, mais malgré cela, cette vulnérabilité nous montre que le développeur d'exploit a encore une chance d'arriver à ses fins. ■

■ REMERCIEMENTS

Je tiens à remercier l'équipe Synacktiv et André Moulu pour leur relecture.

■ RÉFÉRENCES

[DISCLO] http://seclists.org/fulldisclosure/2012/Dec/4

[MARKET] https://www.mysql.com/why-mysql/marketshare/

[IDENTIFIER] http://dev.mysql.com/doc/refman/5.0/en/identifiers.html

[UTF-8] https://en.wikipedia.org/wiki/UTF-8#Description

[TCB] http://dev.gentoo.org/~dberkholz/articles/toolchain/tls.pdf

[TCBLOCAL] http://bases-hacking.org/tcb-overwrite.html

[RET2LIBC] http://www.phrack.org/issues.html?issue=58&id=4

[GOTCTF] http://leetmore.ctf.su/wp/codegate-2012-quals-vuln500-write-up/

[GOTDEREF] http://security.dsi.unimi.it/~roberto/pubs/acsac09.pdf

[IPV] http://blog.ring0.me/2012/01/wireshark-14x-145-cve-2011-1591.html

[DYN] http://leetmore.ctf.su/wp/ifsf-ctf-2012-9-x97/

[DTORS] http://synnergy.net/downloads/papers/dtors.txt

[LONGLD] http://www.vnsecurity.net/2012/02/exploiting-sudo-format-string-vunerability/

[EXPLOIT] http://ring0.me/exploits/mysql-bof_CVE-2012-5611/mysql-bof_CVE_2012_5611.py