An attack on the MySQL authentication protocol

Ivan F.F. Arce Agustín Azubel Emiliano Kargieman
Gerardo Richarte Carlos Sarraute Ariel Waissbein

CORE SECURITY TECHNOLOGIES

January 24, 2002

Abstract

The MySQL challenge and response authentication protocol is proven insecure. Sensitive information is shown to be leaked during each execution of this protocol. We present an algorithm exploiting this vulnerability that enables a passive attacker to impersonate a valid user after witnessing a small number of protocol executions. The paper concludes with statistical information and some effciency and effectiveness estimates.

1 Introduction

Computer-based user authentication has become a tool widely used in our networked society. Usually computer-based authentication is done by way of cryptographic protocols. Remote connections, such as SSH and SSL, are initiated with a user authentication. This also holds true for remote access databases such as the *MySQL Database Engine*.

Computer-based user authentication amounts to one of several different processes by which an entity, the user, is able to ensure his identity by way of a protocol specified *a proiori* to another entity, often a server. Different standards exist for user authentication such as (weak) password authentication and (strong) zero–knowledge/challenge–and–response protocols (e.g., [3], [6], [7], [5] and [13]).

MySQL Database Engine package ([2]) is a popular open source engine enabling remote access to databases through secured channels. This package is widely used in many applications such as world-wide web portals and intranet services and has become a standard in its category.

The MySQL package scenario is constituted of a *server*, which centralizes all the information in a database to which validated users, called *clients*, have access by logging into the server through a cryptographic authentication procedure. When a client is authenticated themself to the server it is allowed to querry the datatabase for information. This information then travels through information channels, encrypted (by a standard encryption algorithm) with a key negotiated between the client and server, and can thus be read by the authenticated client alone. Different parameters as to how this is done are selected by server administrators and shall not be treated in this paper. However all these possible configurations have the same authentication procedure in common.

This authentication procedure was completely designed by the MySQL Development Team to serve a twofold purpose: to prevent both the flow of plaintext passwords over the network and the storage of them in plaintext format on the server's and user's respective terminals. A challenge and response type authentication method was chosen. Regrettably, the authentication mechanism designed is not cryptographically secure. Regarding the latter requirement, we shall see that the only value needed to authenticate a client, which is not the password but it's hash value, is stored both in the client's and the server's memory. But moreover, regarding the former requirement, in the sequel we shall prove that each time a user underpasses a challenge and response execution, information allowing an attacker to impersonate this user is leaked. Hence the security requirements, though achieved, remain unavailing and security can all the same be violated.

In view of these vulnerabilities, described in detail in Section 2, we designed an algorithmic attack, which we describe in Subsection 3, permiting an eavesdropper to authenticate to the database–engine's server impersonating a valid user after witnessing only a few successful authentications of this user. In fact, our algorithmic construct works in such a way that, for every time a client authenticates themself to the server it narrows the key—search space in an almost exponential manner. As a result, a brute–force key–search space of 2^{64} is reduced to a key–search space of 300 after witnessing only 10 authentications! The password is usually recovered after some 300 witnessed runs of the protocol.

Previous vulnerabilities in the MySQL Database Engine were disclosed in the Bugtraq advisories [8], [12] and [4]. A communication by the authors briefly describing this attack appeared as a Bugtraq Advisory in [1].

2 The authentication mechanism

MySQL package provides users with two primitives used for the authentication protocol: a hash function, and a (supposedly) cryptographic one–way function. Both of which are their own design.

A protocol execution is initiatied by a client with a login request and responded by the server with a random string —the challenge— generated by the server. The client runs the response algorithm which calculates the xor of the hash value of the random string he has received with the hash value of it's password (both of the same bit size) and subsequently maps the result by the one-way function to a new string, the response, which is sent to the server. This response string is compared with a string generated by the response algorithm, this time ran on the server. Only if the server calculates the same response string he received from the client, the challenge is passed the client is succesfully authenticated.

As a first remark, we notice that the one–way function provided by MySQL outputs eight-byte strings having —by construction— a range set of only 2^{45} possible outputs (and a fixed input size of 8 bytes). This is evident from the function's evaluation algorithm we show in the next section. This last fact implies several conclusions, firstly notice that 2^{45} is within today's computational range and a brute-force approach to obtain the password (preimage) could be done with a higher computational cost. Moreover, by a birthday paradox reasoning we see that only $2^{22.5}$ values need to be examined to find collisions. Even more, the pigeon–hole principle implies that there is at least one output value with $2^{64-45} = 2^{19}$ collisions (preimages). Empirical evidence show that the situation is even worse.

The one–way function shall be introduced and proved insecure in the next subsection. However, the hash function $h : \{0,1\}^* \rightarrow \{0,1\}^{64}$ need not and shall not be analyzed since the authentication mechanism of MySQL does not require the password for a successful authentication, but only the password's hash value.

2.1 This is (not) another one-way function

For the remaining of the paper let the following notation and assumptions hold. Let $n := 2^{30} - 1$. Fix a client $\mathcal{C}$. On initiating a challenge and response protocol execution the client logs in and is challenged by a random string. Denote this string's hash value by $c \in \{0,1\}^{64}$, with first 4 leftmost bytes $c_x \in \{0,1\}^{32}$ and last 4 rightmost bytes $c_y \in \{0,1\}^{32}$. Likewise let p_x and p_y denote the 4 leftmost and 4 rightmost bytes of the password's hash value $p \in \{0,1\}^{64}$. The response w is calculated by the one-way function $f : \{0,1\}^{32} \times \{0,1\}^{32} \rightarrow \{0,1\}^{64}$ as $w := f(p_x \oplus c_x, p_y \oplus c_y)$, where $\oplus$ denotes the xor (bitwise exclusive or) operation, following the algorithm we now outline. (This validates our claim that only the password's hash value is required for a succesful authentication.)

In what follows numbers will be intercheangably be treated as integers in decimal representation or as the bit strings arising form their binary representation. It should always be clear from the context which representation of the element we are refering to, e.g. the xor of two values will always refer to the bitwise xor, whereas $3 \cdot x$ will refer to the result of multiplying the number 3 by the number x both in $\mathbb{Z}$ (actually in $\mathbb{Z}/n\mathbb{Z}$).

Response algorithm:

1. Let $s_1^{(0)} := p_x \oplus c_x$ and $s_2^{(0)} := p_y \oplus c_y$. The values $s_1^{(0)}$ and $s_2^{(0)}$ are then used as input for the one–way function f,

2. For $1 \leq i \leq 8$ let

$$\begin{aligned} s_1^{(i)} &:= s_1^{(i-1)} + 3 \cdot s_2^{(i-1)} \quad \text{modulo } (n) \\ s_2^{(i)} &:= s_1^{(i)} + s_2^{(i-1)} + 33 \quad \text{modulo } (n) \\ w_i &:= \left\lfloor \frac{31 \cdot s_1^{(i)}}{n} \right\rfloor + 64 \end{aligned}$$

(here $\lfloor x \rfloor := \max\{k \in \mathbb{Z} : k \leq x\}$ is the floor function)

3. Finally let

$$\begin{aligned} s_1^{(9)} &:= s_1^{(8)} + 3 \cdot s_2^{(8)} \quad \text{modulo } (n) \\ w_9 &:= \left\lfloor \frac{31 \cdot s_1^{(9)}}{n} \right\rfloor \end{aligned}$$

4. output the response value

$$w := f(s_1^{(0}, s_2^{(0)}) := \Big((w_1 \oplus w_9) \parallel \ldots\ldots \parallel (w_7 \oplus w_9) \parallel (w_8 \oplus w_9)\Big)$$

It is this response $w \in \{0,1\}^{64}$ that is sent by the client $\mathcal{C}$ to the server. The server, which has in store the hash value of $\mathcal{C}$'s password, parallely calculates this response by this same algorithm verifying if equality with the value it has received holds. In which case the challenge is passed. This ends the authentication procedure.

Notice that $64 \leq w_i < 96$, for $1 \leq i \leq 8$, and $0 \leq w_9 < 32$ hold true, which guarantees that the range set consists of at most $32^9 = 2^{45}$ points. Preimages $f^{-1}(f(x,y))$ of the map f can be efficiently calculated due to this map's rich arithmetic properties deeming it not a one way function. Moreover, this set of preimages is of negligible size in comparison to the 2^{64} points set of all the possible passwords' hashes in which it is contained, and can furthermore be efficiently calculated in a low-storage highly malleable representation. See Lemma 1.

3 The Algorithm For the Attack

The algorithmic attack we designed is composed of two procedures which are repetedly used during the attack plus a minor additional procedure. We follow to outline the two main procedures which are detailed in the forthcoming subsections.

The first procedure is the one inverting f. More precisely, we design an algorithmic process by which one is able to, on input a response $w \in \{0,1\}^{64}$, calculate a suitable representation of the preimage $f^{-1}(w)$ of $w = f(p_x \oplus c_x, p_y \oplus c_y)$ by f. This preimage is given by a collection of (non-repeating) convex polygons $\mathcal{P} = \{P\}$, and satisfying the following properties:

- every $P \in \mathcal{P}$ is given by its vertices in $\mathbb{Q}$ (at most 15 vertices),
- $f^{-1}(\{w\}) = \bigcup_{P\in\mathcal{P}} P$, in particular the point $(p_x \oplus c_x, p_y \oplus c_y)$ of $\mathbb{Z}^2$ belongs to a polygon P in $\mathcal{P}$,
- the collection $\mathcal{P}$ contains at most 48 polygons, and
- each $P \in \mathcal{P}$ contains at most 2^{35} points.

Details will be given in the next subsection. Figure 1 shows a collection $\mathcal{P}$ arising from an example, and a zoom in one of the members P of $\mathcal{P}$.

Procedure 2 relies not in the weaknesses of the one-way function f, but on the precalculation process $c \mapsto c \oplus p$ and the high malleability of the collections $\mathcal{P}$. Let us take some consideration into how can we efficiently intersect sets $c \oplus f^{-1}(w)$ and $c' \oplus f^{-1}(w')$ arising from two different authentications in order to get the set of passwords' hash values passing these challenges. A simpleminded approach is to partition each of these sets into a collection of their integer points keeping only repetitions, i.e. keeping $\mathbb{Z}^2 \cap (c \oplus f^{-1}(w)) \cap (c' \oplus f^{-1}(w')$. But this would *untangle* the efficient representation we have over the preimages $f^{-1}(w), f^{-1}(w')$ (each having approximately 2^{35} points) inevitably yielding an intractable procedure. A less drastic measure towards that direction is pursued: we use a divide–and–conquer approach with much coarser partitions.

For an input of triplets $(c, w, \mathcal{P}), (c', w'\mathcal{P}')$ where c, w and c', w' are challenge and response pairs and $\mathcal{P}, \mathcal{P}'$ are polygons collections with $\cup_{P\in\mathcal{P}} P \subseteq f^{-1}(w)$ and $f^{-1}(w') = \cup_{P'\in\mathcal{P}'} P'$, and for a precision parameter $k \in \mathbb{Z}, 0 \leq k \leq 32$, we calculate a refinement of $\mathcal{P}$ consisting of the polygons

$$P_{i,j,k} := P \cap \left([i \cdot 2^{32-k}; (i+1) \cdot 2^{32-k}) \times [j \cdot 2^{32-k}; (j+1) \cdot 2^{32-k})\right)^{1}$$

satisfying an intersection property. More precisely, for each nonempty $P_{i,j,k}$ in the output collection it holds that:

[1]both intervals are closed on the left and open on the right.

– There exists a polygon $P' \in \mathcal{P}'$ and integer points $(x, y) = (x_1, \ldots, x_{32}; y_1, \ldots, y_{32}) \in P_{i,j,k} \cap \mathbb{Z}^2$ and $(x', y') = (x'_1, \ldots, x'_{32}; y'_1, \ldots, y'_{32}) \in P' \cap \mathbb{Z}^2$ such that the first k bits of $x' \oplus c'_x$ and $x \oplus c_x$ agree, and the first k bits of $y' \oplus c'_y$ and $y \oplus c_y$ agree, i.e.,

$$\begin{aligned} x_t \oplus c_{x,t} &= x'_t \oplus c'_{x,t} \qquad \text{for } 1 \leq t \leq k \\ y_t \oplus c_{y,t} &= y'_t \oplus c'_{y,t} \qquad \text{for } 1 \leq t \leq k, \end{aligned}$$

where $(c'_x, c'_y) = (c'_{x,1}, \ldots, c'_{x,32}; c'_{y,1}, \ldots, c'_{y,32})$ and $(c_x, c_y) = (c_{x,1}, \ldots, c_{x,32}; c_{y,1}, \ldots, c_{y,32})$ denote the bit representation of c' and c.

– $P_{i,j,k}$ is represented by it's vertices in $\mathbb{Q}$.

Notice that, since the first k bits of $X \oplus c_x = (p_x \oplus c_x) \oplus c_x = p_x$ and $X' \oplus c'_x = (p_x \oplus c'_x) \oplus c'_x = p_x$ agree, as well as $Y \oplus c_y = (p_y \oplus c_y) \oplus c_y = p_y$ and $Y' \oplus c'_y = (p_y \oplus c'_y) \oplus c'_y = p_y$, this condition holds in particular for the polygon $P_{i,j,k}$ containing the point (X, Y). For a more accurate description see Subsection 3.2.

3.1 From brute–force to brute forge

Procedure 1 produces, from a pair of challenge and response (c, w), a collection of polygons $\mathcal{P}$ containing all the xor values of the challenge c with the hashed passwords passing the same challenge, i.e. the collection $\mathcal{P}$ is such that $f^{-1}(w) = \cup_{P \in \mathcal{P}} (P \cap \mathbb{Z}^2)$. Implicitly, we also deduce that the output of Procedure 1 is a natural way of representing the preimage $f^{-1}(w)$ by the function f of the response $w = (w_1 \oplus w_9, \ldots, w_8 \oplus w_9)$.

Suppose without loss of generality that $w_1, \ldots, w_8$ are known, e.g. w_9 is known. In the Remark 2 ahead, we justify this supposition and explain how this problem is tackled.

The point $(X, Y) = (p_x \oplus c_x, p_y \oplus c_y)$ can be expressed in terms the entries $w_1, \ldots, w_8$ and some universal constants $\alpha_i, \beta_i, \gamma_i \in \mathbb{Z}$ by formulas

$$w_i = \lfloor \frac{31}{n} \big((\alpha_i \, X + \beta_i \, Y + \gamma_i \, 33) \bmod (n) \big) \rfloor + 64$$

which are deduced by rewriting the recursive formulas defining the w_i (Response algorithm). For example we have $w_1 = \left\lfloor \frac{31}{n} (3X + Y \bmod (n)) \right\rfloor + 64, w_2 = \left\lfloor \frac{31}{n} (12X + 5Y + 33 \bmod (n)) \right\rfloor + 64, \ldots$. Here the $\alpha_i, \beta_i, \gamma_i$ can be calculated once and for all.

Since for any $z \in \mathbb{Q}$ it holds that $\lfloor z \rfloor \leq z < \lfloor z \rfloor + 1$ we deduce that the point $(X, Y) \in \mathbb{Z}^2$ belongs to the following semi–algebraic sets

$$\Big\{ (x, y) \in R : \frac{n}{31}(w_i - 64) \leq \alpha_i \, x + \beta_i \, y + 33 \, \gamma_i \bmod (n) \; < \frac{n}{31}(w_i - 63) \Big\}$$

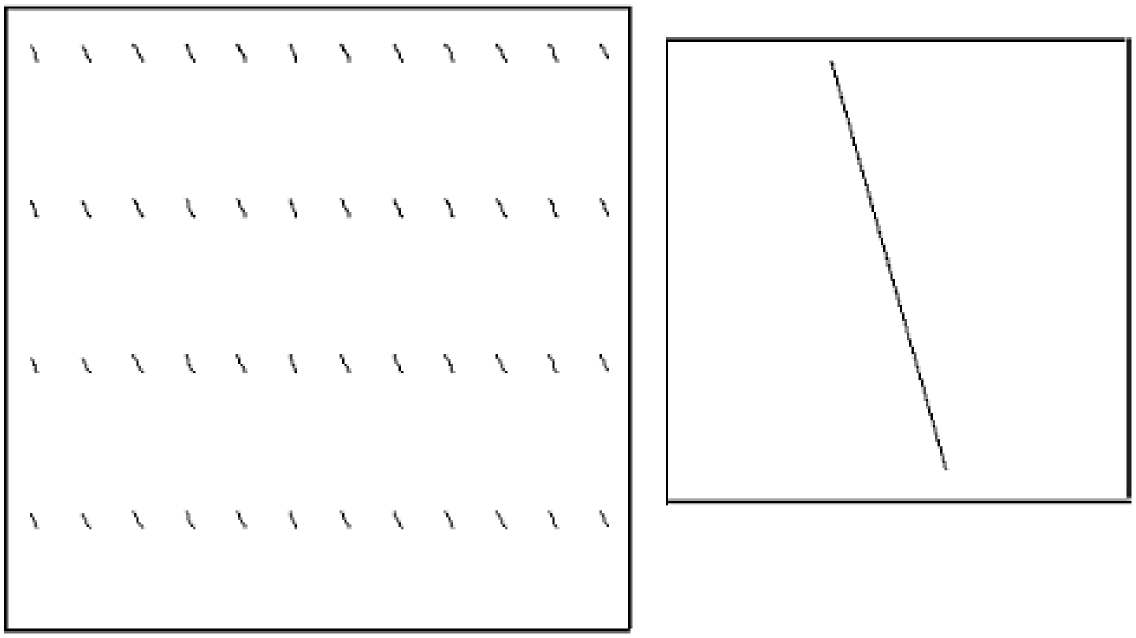

Figure 1: $\mathcal{P}$ on the left and a zoom in P on the right

of $\mathbb{Q}^2$, where R is the square $R := \{0 \le x, y < 2^{32}\}$. That is, for $1 \le i \le 8$, there exist an integer δ_i such that the point (X, Y) is in the semi-algebraic set $\big\{(x, y) \in R : \frac{n}{31}(w_i - 64) \le \alpha_i\, x + \beta_i\, y + 33\,\gamma_i - \delta_i\, n < \frac{n}{31}(w_i - 63)\big\}$. Furthermore, since these entries $X, Y \in \mathbb{Z}$ verify $0 \le X, Y < 2^{32}$, we deduce that the δ_i are non-negative integers bounded from above by $\delta_i < \left\lceil \frac{2^{32}(\alpha_i+\beta_i)+33\gamma_i}{n} \right\rceil$.

For each choice of $\overline{\delta} := (\delta_1, \ldots, \delta_8) \in \mathbb{Z}^8$ consider the semi–algebraic set

$$P_{\overline{\delta}} := \bigcap_{1 \le i \le 8} \Big\{(x, y) \in R : \frac{n}{31}(w_i - 64) + \delta_i\, n \le \\ \le \alpha_i\, x + \beta_i\, y + \gamma_i\, 33 < \frac{n}{31}(w_i - 63) + \delta_i\, n\Big\},$$

which is trivially a convex polygon. It follows that, for every $\overline{\delta} \in \mathbb{Z}^8$ and for every integer point (a, b) in $P_{\overline{\delta}} \cap \mathbb{Z}^2$ it holds that $f(a, b) = w$. In fact, it is easy to see that the reverse inclusion also holds, i.e., for every pair (a, b) which is mapped by f to the response w there exist a tuple $\overline{\delta} \in \mathbb{Z}^8$ such that (a, b) belongs to $P_{\overline{\delta}}$. And $f^{-1}(w) = \cup_{P \in \mathcal{P}} P$ as announced.

Let $\mathcal{P}$ denote the collection of all $P := P_{\overline{\delta}}$ with $\overline{\delta} \in \mathbb{Z}^8$. Next lemma states some useful bounds and properties over the collection $\mathcal{P}$.

Lemma 1 *Let the above notations and assumptions hold. Recall the notations $R := [0, 2^{32} - 1] \times [0, 2^{32} - 1] \subset \mathbb{Q}^2$ and $\mathcal{P} = \cup_{P \in \mathcal{P}} P$. The following conditions hold*

- *each $P \in \mathcal{P}$ is a traslate of the other, i.e. for every $P, P' \in \mathcal{P}$ there exists $v \in \mathbb{Q}^2$ such that $P = P' + v$.*
- *the number of polygons $\#\mathcal{P}$ in $\mathcal{P}$ is bounded from above by 48, with each polygon $P \in \mathcal{P}$ containing at most $\#(\mathbb{Z}^2 \cap P) \leq 2^{36}$ integer points.*

Proof: For $i \in \mathbb{Z}$, with $1 \leq i \leq 8$, let $\delta_i \in \mathbb{Z}$ denote an integer parameter. Let us fix some additional notation. Let $S^{(i)}_{\delta_i}$ be the subset $S^{(i)}_{\delta_i} :=$

$$\{(x, y) \in R : \frac{n}{31}(w_i - 64) + \leq \alpha_i\, x + \beta_i\, y + \gamma_i\, 33 - \delta_i\, n < \frac{n}{31}(w_i - 63)\}$$

of $\mathbb{Q}^2$. And let $S_{\delta_1,\ldots,\delta_k} := S^{(1)}_{\delta_1} \cap \cdots \cap S^{(k)}_{\delta_k}$ for $k \in \mathbb{Z}, 1 < k \leq 8$. Notice that $S_{\delta_1,\ldots,\delta_8} = P_{(\delta_1,\ldots,\delta_8)}$.

We follow the calculation process for each of the P in $\mathcal{P}$. Firstly, we show that the cardinality $\#\{S_{\delta_1,\delta_2} : \delta_1, \delta_2 \in \mathbb{Z}\}$ is bounded by 48. To see this, we calculate the intersecion of the straight lines defining the boundries of the $S^{(1)}_{\delta_1}$ and $S^{(2)}_{\delta_2}$ for any $(\delta_1, \delta_2) \in \mathbb{Z}^2$. Say, the lines defined by the equations $3x + y = \frac{n}{31}(w_1 - 64) - \delta_1 n$ and $12x + 5y + 33 = \frac{n}{31}(w_2 - 64) - \delta_2 n$ intersect for

$$x = \frac{1}{3}\left\{\frac{n}{31}\left[5(w_1 - 64) - (w_2 - 64)\right] + n(\delta_2 - 5\delta_1) - 33\right\}$$

which ranges between 48 different values of $x \in \mathbb{Q}$, constrained to $0 \leq x \leq 2^{32} - 1$ and $\delta_1, \delta_2 \in \mathbb{Z}$.

The intersections of other lines defining the boundries of $S^{(1)}_{\delta_1}$ and $S^{(2)}_{\delta_2}$ can be simillarly calculated and the same conclusions are verified. Namely, at stage $k = 2$ the collection $\{S_{\delta_1,\delta_2} : \delta_1, \delta_2 \in \mathbb{Z}\}$ contains at most 48 polygons whose vertices' entries can be expressed by equations like the preceding. Moreover, by examining this defining equations we deduce that: a) every polygon S_{δ_1,δ_2} has x coordinates bounded between $\frac{1}{3}\{\frac{n}{31}[5(w_1 - 63) - (w_2 - 64)] + n(\delta_2 - 5\delta_1) - 33\}$ and $\frac{1}{3}\{\frac{n}{31}[5(w_1 - 64) - (w_2 - 63)] + n(\delta_2 - 5\delta_1) - 33\}$ for different δ_1, δ_2. Which are $\frac{2}{31}n$ units appart. And furthermore, b) each of these polygons is a traslate of the other.

The result of covering stages $k \in \{3, \ldots, 8\}$ will be to successively refine the polygons calculated at the stages $k - 1$. More specifically, for a given δ_k we are able to deduce that $S^{(k)}_{\delta_k}$ is such that: a) the lines defined by each $S^{(k)}_{\delta_k}$ are at distances in the x coordinate greater than $\frac{2}{31}n$, and hence intersect no more than one of the S_{δ_1,δ_2}; and b) the intersections are at even distances resulting again in a set of at most 48 polygons each a translate of the other.

Finally, to estimate the area of the polygons, we notice that for each $P \in \mathcal{P}$ there exists integers $\delta_1, \delta_8 \in \mathbb{Z}$ such that the parallelogram $S^{(1)}_{\delta_1} \cap S^{(8)}_{\delta_8}$ contains P. So that the area of P is bounded by the area this parallelogram:

which we estimate by $5.2806 \cdot 10^{10} \approx 2^{35.6}$. Hence $\#(\mathbb{Z}^2 \cap P) \leq 2^{36}$ as announced. $\square$

The algorithm for this procedure should be now clear. Supposing w_9 known the algorithm selects values of $\overline{\delta}$ within the preestablished range $(0 \leq \delta_i \leq \lceil 2^{32}(\alpha_i + \beta_i)/n \rceil)$ until a nonempty polygon $P = P_{\overline{\delta}}$ is calculated. Once this polygon is calculated it is easy to calculate the coordinates where every other polygon of $\mathcal{P}$ is placed. This is done by examining the equations defining P. In fact, this examination shows that $\mathcal{P}$ will consist of either 48 or 36 polygons. These polygons are stored in the output collection as a collection of their vertices in $\mathbb{Q}^2$. We point out that on the implementations, the use of IEEE standard floating point arithmetic tends to speed the algorithm and approximations have no negative effect.

Let us go back to our assertion that w_9 is known. Suppose it is not, and suppose that we have made our choice for w_9 candidate, say $\hat{w}_9$. Suppose furthermore that the δ_i are fixed. Then the polygon P defined by this choices is

$$P = \bigcap_{1 \leq i \leq 8} \Big\{ (x, y) \in R : \frac{n}{31}(w_i \oplus w_9 \oplus \hat{w}_9 - 64) \leq \\ \alpha_i X + \beta_i Y + \gamma_i 33 - \delta_i n < \frac{n}{31}((w_i \oplus w_9 \oplus \hat{w}_9 - 63) \Big\}.$$

We have shown in the proof of Lemma 1 that the x coordinates of the points inside S_{δ_1,δ_2} differ in at most $\frac{2}{31}n$. For a wrong choice $\hat{w}_9$ of w_9, we see that one of the further intersections is bound to be empty. This is stated in the next remark.

Remark 2 *By applying the algorithmic procedure just described to* $w[k] = (w_1 \oplus k \parallel \ldots \parallel w_8 \oplus k)$ *for* $0 \leq k < 32$ *only one value of* k *produces a nonempty output, hence that value of* k *is precisely* w_9*, i.e. we have* $w_9 = k$.

We will not prove this remark. Notice however that the validity of this remark is not needed for the attack to hold, but only helps to speed the attack by a factor of 32.

3.2 Wash out of invalid passwords

Let be given triplets $(c, w, \mathcal{P}), (c', w', \mathcal{P}')$ and a precision parameter $k \in \mathbb{Z}$, with $1 \leq k \leq 32$, such that (c, w) and (c', w') are challenge and response pairs, and that $\mathcal{P}$ and $\mathcal{P}'$ are polygons collections satisfying $\cup_{P \in \mathcal{P}} P \subseteq f^{-1}(w)$ and $\cup_{P' \in \mathcal{P}'} P' = f^{-1}(w')$.

We use the following notation. For any subset $Q \subseteq R = [0; 2^{32} - 1] \times [0; 2^{32} - 1]$ and integers $r, i, j \in \mathbb{Z}$ with $1 \leq r \leq 32$ and $0 \leq i, j \leq 2^r - 1$ denote by $Q_{i,j,r}$ the set

$$Q_{i,j,r} := Q \cap \big([i \cdot 2^{32-r}; (i+1) \cdot 2^{32-r}) \times [j \cdot 2^{32-r} \leq y < (j+1) \cdot 2^{32-r})\big).$$

resulting of the intersection of Q with the 2^{32-r}–side square in the prescribed coordinates. For a point v in $\{0,1\}^{32} \times \{0,1\}^{32}$ we write $v = (v_{x,1}, \ldots, v_{x,32}; v_{y,1}, \ldots, v_{y,32})$.

Given two sets Q and Q' of $\mathbb{Q}^2$, and an integer r as above, we say that Q and Q' satisfy the *r–property* if and only if there exist points $(x,y) \in Q \cap \mathbb{Z}^2$ and $(x',y') \in Q' \cap \mathbb{Z}^2$ such that

$$\begin{aligned} x_t \oplus c_{x,t} &= x'_t \oplus c'_{x,t} \qquad \text{for } 1 \le t \le r \\ y_t \oplus c_{y,t} &= y'_t \oplus c'_{y,t} \qquad \text{for } 1 \le t \le r. \end{aligned}$$

The algorithm for Procedure 2 calculates the subset of polygons of $\{P_{i,j,k} : P \in \mathcal{P}; 0 \le i,j \le 2^k - 1\}$ such that there exists a polygon P' and the polygons $P_{i,j,k}$ and P' satisfy the *k–property.* Working in a divide–and–conquer manner it calls a recursive procedure for values of r ranging from $r = 1$ and increasing at steps of 1 to $r = k$ always keeping those $\{P_{i,j,r}\}$ such that there exists a polygon P' and the polygons $P_{i,j,r}$ and P' satisfy the *r–property.*

Let $r \in \mathbb{Z}, 1 \le r \le k$ be fixed. We outline the recursive procedure. The procedure starts with two input collections $\mathcal{P}$ and $\mathcal{P}'$, a temporary storage collection for intermediate steps, and an output collection. We first calculate $\{P_{i,j,r} : 0 \le i,j \le 2^r - 1, P \in \mathcal{P}\}$ and store it as $\mathcal{P}$. Then, we iterate over the polygons in the collection $P' \in \mathcal{P}'$ partitioning each P' as $P' = \cup_{0 \le \ell,\kappa \le 2^r - 1} P_{\ell,\kappa,r}$ and storing the result in the temporary collection.

Notice that for every $0 \le \ell, \kappa \le 2^r - 1$ it holds that $P'_{\ell,\kappa,r}$ is a polygon in $R \subset \mathbb{Q}^2$ whose integer points (x,y) have coordinates verifying $\ell \cdot 2^{32-r} \le x < (\ell+1) \cdot 2^{32-r}$ and $\kappa \cdot 2^{32-r} \le y < (\kappa+1) \cdot 2^{32-r}$ and hence their respective first r bits are constant (and respectively equal to the first r bits of $\ell \cdot 2^{32-r}$ and $\kappa \cdot 2^{32-r}$). An analogous property holds for all the $P_{i,j,r}$. In particular, to decide whether a pair of polygons $P_{i,j,r}$ and $P'_{\ell,\kappa,r}$ verify the *r–property* one need only check it for any pair of points $(x,y) \in P_{i,j,r}$ and $(x',y') \in P'_{\ell,\kappa,r}$. We use any pair of vertices, since they are at store. Explicitly, for each $P'_{\ell,\kappa,r}$ we iterate over the collection $\mathcal{P} = \{P_{i,j,r}\}$ extracting —and storing in the output collection— those polygons $P_{i,j,r}$ verifying the *r–property* (with $P'_{\ell,\kappa,r}$).

After the iteration over all the $P_{\ell,\kappa,r}$ is finished the output collection holds all the polygons $P_{i,j,r}$ such that there exists a $P'_{\ell,\kappa,r}$ with the pair verifying the *r–property.* When r is increased and the recursive function is called again, the output collection is previously stored in place of $\mathcal{P}$ rewriting it's previous contents.

In Figure 2, we see the result of calling our recursive function (for a single fixed P' and all of the P) for the values of $r = 1$ and $r = 2$ over the chosen example. For this example the first bits of $c \oplus c'$ are $(0,1,\ldots;0,1,\ldots)$. Polygon

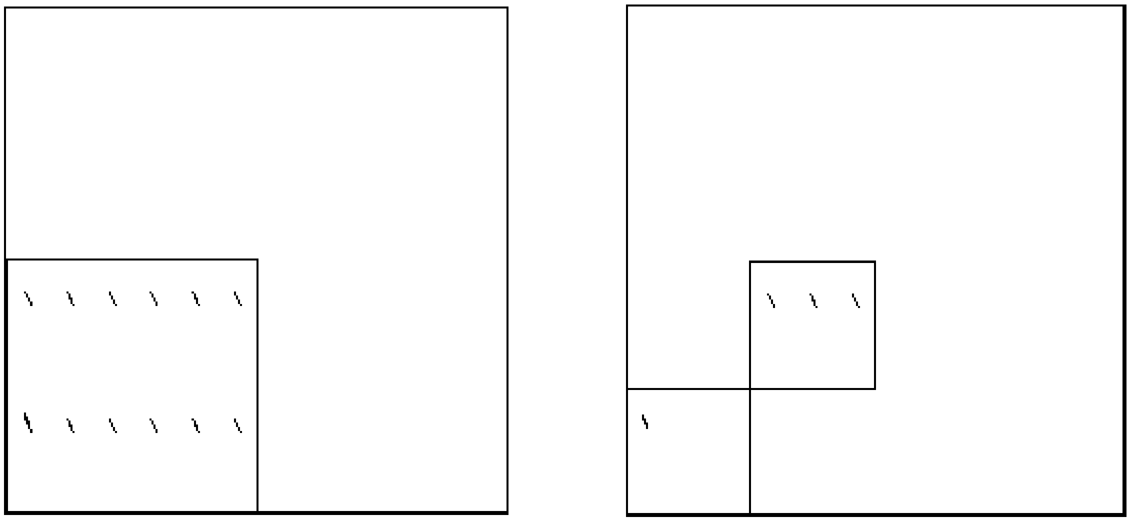

Figure 2: The recursive function's result for $r = 31$ and $r = 30$.

P' has all of its points with coordinates starting as $(0, 0, \ldots; 1, 1, \ldots)$. The figure on the left shows that only 12 polygons in $\{P_{i,j,1}\}$ verify the 1–property (it's coordinates are written as $(0 \oplus 0, \ldots; 1 \oplus 1, \ldots) = (0, \ldots; 0, \ldots)$). For $r = 2$ it follows that only 3 polygons $\{P_{i,j,2}\}$ verify the 2–*property*, those with coordinates starting as $(0, 1, \ldots; 1, 0, \ldots)$.

3.3 The complete algorithmic attack

A complete attack against a valid client which has produced pairs of challenge and responses $(c^{(1)}, w^{(1)}), \ldots, (c^{(n)}, w^{(n)})$ is done by repeated application of the procedures we have just described. To start with, we select the number $p \leq n$ of these challenge and response pairs to which we are going to apply to Procedure 1. That is for the pairs $(c^{(1)}, w^{(1)}), \ldots, (c^{(p)}, w^{(p)})$ we apply the Procedure 1 and output collections $\mathcal{P}^{(1)}, \ldots, \mathcal{P}^{(p)}$. Typically —on our examples— the number p is taken to be 5 or less.

On a second step, after selecting integers $1 \leq k_1 \leq k_2 \leq \ldots \leq k_{p-1} \leq 32$, we apply Procedure 2 recursively to the triplets $(c^{(1)}, w^{(1)}, \mathcal{P}^{(1)}), \ldots, (c^{(q)}, w^{(q)}, \mathcal{P}^{(q)})$; that is first we apply Procedure 2 to $(c^{(1)}, w^{(1)}, \mathcal{P}^{(1)})$ and $(c^{(2)}, w^{(2)}, \mathcal{P}^{(2)})$ using precision $k := k_1$ and save the result, then we apply Procedure 2 to the previous saved result and $(c^{(3)}, w^{(3)}, \mathcal{P}^{(3)})$ using precision $k := k_2$ saving the result overwritting the previous one, and continue this recursive application until the p–th tuple is reached. After the p–fold application of this procedure we get a set of polygons $\widetilde{\mathcal{P}}$ having a relatively small number of integer points.

We notice that the attack can be optimized by tuning the choices of k: bigger values of k will result in an increase in the number of polygons in the output (and might result in storage problems) and smaller values of k simply will not produce any refining of our polygon collection (but spend computing time).

Finally, every integer point in $\widetilde{\mathcal{P}}$ is extracted and challenged with the whole collection of tests $(c^{(1)}, w^{(1)}), \ldots, (c^{(n)}, w^{(n)})$ and saved in the output collection only if it passes every one of these tests. The algorithmic attack ends either when all the $(c^{(t)}, w^{(t)})$ have been passed through and there are no more challenge and response pairs left, or before, if the set of remaining points has only one point left —the password's hash.

4 Statistics and Conclusions

In the tested examples an average of 300 possible passwords were left with the use of only 10 pairs of challenge and response. Notice that in a plain brute–force attack about $2^{64} - 300 = 18,446,744,073,709,551,316$ would remain as possible passwords. Using an average of 100 pairs of challenge and responses, this set was reduced to 2 possible passwords (i.e., a fake passwords and the password indeed). Finally it took about 300 pairs of challenge and response to get the password.

Implementations on Squeak Smalltalk and a Python module performing this attack can be downloaded from `http://www.corest.com/corelabs/`

A result worth mentioning is the following experiment. Using only ten pairs of challenge and responses, getting thus an average of 300 possible passwords in each case, we randomly selected 1000 challenges and asked whether the selected password would pass these challenges. With this test we estimated the probability each of these 300 passwords has of passing a test. Our samples had an average of 0.92 probability of success. Hence, not having enough challenge and response pairs to calculate the exact password might not be a problem. We can simply apply the attack algorithm to all the pairs of challenge and response captured, then use any possible password in the possible passwords collection remaining after the last application of Procedure 3. Fake passwords will still pass many tests!.

The whole procedure application lasted no more than an hour running on a 700Hz Pentium 3 personal computer with 64Mb RAM on every example we tested.

No evident workaround for this authentication procedure seems plausible, since variation of the constants in the response algorithm only result in a loss of efficiency with no security improvement.

* * *

Thanks: A prior notification of these results appeared signed by researchers and co–author Ivan Arce as "An advisory on MySQL's login protocol" in Securityfocus' Bugtraq newsgroup [1]. We would like to thank the Vulnerability Help Team at Securityfocus for their help in the drafting of the advisory.

References

[1] Agustin Azubel, Emiliano Kargieman, Gerardo Richarte, Carlos Sarraute, and Ariel Waissbein. Mysql authentication vulnerability. Securityfocus bugtraq advisory `http://www.securityfocus.com/bid/1826` or `http://www.corest.com`.

[2] MySQL Database Engine. Visit `http://www.mysql.com` and for the Section Security `http://www.mysql.com/documentation/mysql/commented/manual.php?section=Security` for more information.

[3] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology — Crypto '86*, pages 186–194, New York, 1987. Springer-Verlag.

[4] Viktor Fougstedt. Mysql grant global password changing vulnerability. Securityfocus bugtraq advisory `http://www.securityfocus.com/bid/926`.

[5] Internet Engineering Task Force (ITEF). draft-ietf-secsh-userauth. Visit `http://www.ietf.org` for the latest version.

[6] Jean-Jacques Quisquater and Louis Guillou. A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory. In *Advances in Cryptology — Eurocrypt '88*, pages 123–128, New York, 1998. Springer-Verlag.

[7] RFC1510. Ther kerberos network authentication service (v5). Internet Request For Comments (RFC) 1510, J. Kohl and C. Neumann, September 1993.

[8] Jesse Schachter. 'ic radius buffer overflow vulnerability. Securityfocus bugtraq advisory `http://www.securityfocus.com/bid/1147`.

[9] C. Schnorr. Efficient identification and signatures for smart cards. In *Advances in Cryptology — Crypto '89*, pages 239–252, New York, 1989. Springer-Verlag.

[10] C.-P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 4(3):161–174, 1991.

[11] Squeak Smalltalk. See `http://www.squeak.org`.

[12] Robert van der Meulen. Mysql unauthenticated remote access vulnerability. Securityfocus bugtraq advisory `http://www.securityfocus.com/bid/975`.

[13] Thomas Wu. The secure remote password protocol. In *Proceedings of the 1998 Internet Society Network and Distributed System Security Symposium*, pages 97–111, San Diego, CA, March 1998.