An attack on MySQL's login protocol

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

January 24, 2002

Abstract

The MySQL challenge–and–response authentication protocol is proved insecure. We show how can an eavesdropper impersonate a valid user after witnessing only a few executions of this protocol. The algorithm of the underlying attack is presented. Finally we comment about implementations and statistical results.

1 Introduction

The use of computer-based user authentication has become a cryptographic tool widely used in these days. Every remote connection (SSH, SSL, et cetera) is initiated with a user authentication. This also holds true for remote access databases as the MySQL Database Engine. Computer-based user authentication amounts to one of different process by which an entity, the user, is able to authenticate himself by way of a cryptographic protocol to another entity, often a server, in such a way that no other person —than the user— can do this. Different standards exist for user authentication such as zero–knowledge cryptography (e.g., Fiat–Shamir [3], Guillon–Quisquater [6] or Schnorr [9, 10] identification protocols), or challenge–and–response protocols (e.g., see Kerberos [7], the Secure Shell authentication protocols [5] see also [13]).

MySQL Database Engine ([2]) is a world known open source database engine enabling remote access 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 scenario is constituted by a *server*, which centralizes all the information in a database to which validated users, called *clients*, have access by logging on the server through an authentication procedure. When a client authenticates himself to the server, he can then start a session and succeedingly obtain any information from the database in the server. This information then, travels through information channels encrypted with a key negotiated between the client and server, and can thus, only be read by this client. Different parameters as to how this is done can and are selected by server administrators. However, all this possible configurations have in common the same authentication procedure.

The authentication protocol is designed by the MySQL team with a twofold purpose, to prevent 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. For these purposes, a challenge–response mechanism for authentication is chosen together with a hash function. There is no mention of this authentication mechanism in the literature as it was designed by the MySQL development team and never published. The authentication procedure we describe here is extracted from the source code[1] implemented on every version of MySQL. Slight variations are to be found between versions prior to `3.20`, and versions after `3.21`.

Regrettably, this authentication mechanism is not cryptographically strong. Firstly we shall see that the second objective is not met, since the only value needed to authenticate a user is stored both in his machine as in the server. But moreover, we shall see that, each time a user underpasses a challenge–and–response execution, information allowing an attacker to recover this user's password is leaked.

In view of these vulnerabilities which we describe in Section 2 we designed an attack —described in Subsection 3— which permits an eavesdropper to authenticate to the database engine impersonating the witnessed valid user after only witnessing a few successful authentications of this user.

We shall prove that all the contents of a MySQL database can be obtained by sniffing a few client authentications. In fact, our algorithmic construct works in such a way that, for every time a client authenticates himself to the server he narrows the key search space almost in an exponential manner. That is, this is done in such a way that starting with a brute–force key search space of 2^{64}, we are reduced to a search space of 300 after only 10 authentications!

Previous vulnerabilities in the MySQL Database Engine where discovered in the Bugtraq advisories [8], [12] and [4]. An advisory authored by the subscribers which briefly describes this attack appeared as a Bugtraq Advisory in [1].

[1]available at `http://www.mysql.com`

2 Technical Description

2.1 The challenge/response mechanism

The authentication protocol of MySQL Database Engine is of the challenge–and–response type, the underlying idea behind this construction is that no passwords are sent between client and server through the connection. The challenge-response mechanism of MySQL does the following (From `mysql-3.22.32/sql/password.c`). MySQL provides users with two primitives used for the authentication protocol:

- a hash function, and
- a (supposedly) one–way function;

both of their own design. The protocol goes as follows. On connection, when a user wants to log in, a random string is generated by the server and sent to the client —this is the challenge. The client, using as input the hash value of the random string he has received and the hash value of his password, calculates a new string using the one–way function —the response— which is sent to the server.

This *checksum* string is sent to the server, where it is compared with a string generated from the stored `hash_value` of the password and the random string. The password is saved (in `user.password`) by using the `PASSWORD( )` function in mysql. If the server calculates the same string as the response, the user is authenticated.

2.2 Problem Description

The hash function provided by MySQL outputs eight-bytes strings, this makes 2^{64} possibilities. Whereas the one–way function outputs eight-bytes strings, but with only 2^{45} possibilities, having a fixed input size of 8 bytes. From this we deduce that more than one hashed password produces the same (expected) response for a given challenge. That is, we shall show that for a given challenge, not only the original password gives the correct response, but a much larger collection of values —standing for different hashed passwords— also do (see Section 4). We wish to remark that only the one–way function shall be analyzed and proved insecure, whereas the hash function shall not be analyzed.

We also point out that the authentication mechanism of MySQL does not require the password for a successful authentication, but the password's hash value. Hence, to impersonate a user only the hash value of this user's password is needed, so that the hash function is of no interest for this account.

To validate our claim, we explain why the hash value of the password can be efficiently calculated using only a few executions of the challenge–and–response mechanism for the same user. More explicitly, in the forth–coming section we exploit this weakness, and deduce an attack much more efficient than brute–force attack can be carried out in only a few hours on a personal computer (see Section 4). Explicitly, after gaining a positive number of pairs of challenge and response, an eavesdropper is able to efficiently calculate the set of values, standing for hashed passwords, that pass the intercepted challenge and response pairs.

To do this, firstly we describe how does the MySQL one–way function work and proceed to analyze the scheme's security, and then describe the attack we devised. The actual algorithm for making this calculations will be described in the Section 3. The algorithm we describe was implemented in Squeak Smalltalk (see [11]) by co–authors Gerardo Richarte and Carlos Sarraute. All the empirical results herein provided are derived from that implementation and the figures 1, 2 and 3 (in subsections 1 and 3) are cut–and–pasted black and white screen images of this implementation.

Let $n := 2^{30} - 1$ (here n is the `max_value` used in `randominit( )` and `old_randoninit( )` respectively). Fix a user $\mathcal{U}$. And initiate a challenge and response. That is, suppose the server has sent a challenge to the user $\mathcal{U}$. The hash value of this user's password is 8 bytes long. Denote by p_1 the first (leftmost) 4 bytes of this hash value and by p_2 the last 4 bytes (rightmost). Likewise, let c_1 denote the first 4 bytes of the challenge's hash value and c_2 the last 4. We describe how to calculate the output of the one–way function and how is this une–way function used.

1. calculate the values $s_1 := p_1 \oplus c_1$ and $s_2 := p_2 \oplus c_2$ (here $\oplus$ denotes the bitwise exclusive or (X–or) function, and s_1 and s_2 are the input to the one–way function),

2. calculate recursively for $1 \leq i \leq 8$:

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

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

3. calculate form the preceding values

$$\begin{aligned} s_1 &= s_1 + 3 \cdot s_2 && \text{modulo } (n) \\ s_2 &= s_1 + s_2 + 33 && \text{modulo } (n) \\ w_9 &= \left\lfloor \frac{31 \cdot s_1}{n} \right\rfloor \end{aligned}$$

4. output the checksum value

$$w = \Big((w_1 \oplus w_9) \,\|\, \ldots\ldots \,\|\, (w_7 \oplus w_9) \,\|\, (w_9 \oplus w_9) \Big)$$

It is this checksum $w \in \{0,1\}^{64}$ that is sent, by $\mathcal{U}$, to the server. The server, that has in store the hash value of $\mathcal{U}$'s password, recalculates the checksum by this same process and succinctly verifies the authenticity of the value it has received. However it is a small collection of these checksums that allows any attacker to obtain p_1 and p_2 (the hash value of the user's password) and hence, enables the attacker to impersonate any user with only the information that travels on the wire between server and client (the user $\mathcal{U}$). Actually, at each step of the algorithm a set of values (p_1, p_2) is calculated such that,

The reason why the process of producing the checksum out of the hash values of both the password and the challenge, is insecure is that this process can be efficiently reversed due to its rich arithmetic properties. More specifically, consider the one–way function described above as a mapping f that takes as input the two values X and Y and produces the checksum value $f(X,Y) = w$ (e.g., in our case $X := p_1 \oplus c_1$ and $Y := p_2 \oplus c_2$). Then we can efficiently calculate all of the values X', Y' which map to the same checksum value than X, Y, i.e. if $f(X,Y) = w$, then we calculate the set of all the values X', Y' such that $f(X', Y') = w$. This set is of negligible size in comparison to the 2^{64} points set of all the possible passwords' hashes in which it is contained. Furthermore, given a collection of challenge and response pairs made between the same user and the server, it is possible to efficiently calculate the set of all (hash values of) passwords passing the given tests.

3 The Algorithm For the Attack

We now give a brief description of the attack we propose. This description shall enable readers to verify our assertion that the MySQL authentication scheme leaks information. This attack has been implemented on Squeak Smalltalk and is now perfectly running. In what follows we shall depict the procedures that constitute our attack. Since the attack is of a geometric nature, we will be able to illustrate these procedures with screen snapshots of the Squeak implementations.

The attack we designed is mainly divided into three stages. In these stages we respectively use one of our three algorithmic tools in various opportunities.

$\boxed{\text{Procedure 1}}$ is an algorithmic process which has as input a checksum w, and outputs a set of convex polygons $\mathcal{P} = \{P\}$ such that

- each $P \in \mathcal{P}$ is defined by its vertices,
- $f^{-1}(\{w\}) = \bigcup_{P \in \mathcal{P}} P$, e.g. the point $(p_1 \oplus c_1, p_2 \oplus c_2)$ of $\mathbb{Z}^2$ belongs to a polygon P in $\mathcal{P}$,
- $\#\mathcal{P} = 36$ or 48 (see Remark 1 on the next subsection), and
- $\#(\mathbb{Z}^2 \cap P) \sim 2^{33}$ (unproven estimate).

$\boxed{\text{Procedure 2}}$ An input is a pair of tuples $(c, \mathcal{P}), (c', \mathcal{P}')$ and an integer k, $1 \leq k \leq 32$, where c and c' are different pairs of challenges, with respective responses w and w' such that $f^{-1}(w) = \cup_{\mathcal{P}} P$ and $f^{-1}(w') = \cup_{\mathcal{P}'} P'$, and such that $\mathcal{P}$ and $\mathcal{P}'$ are compliant with the four conditions stated above. The output is a collection of polygons $\widetilde{\mathcal{P}}$ consisting of all the polygons $\widetilde{P} = \bigcup (P \cap Q)$, the union taken over all the squares Q of side 2^{32-k} and vertices with entries in $\{i \cdot 2^k : 0 \leq i \leq 2^{32-k}\}$, and all the polygons $P \in \mathcal{P}$ in $\mathcal{P}$ for which there exists a $P' \in \mathcal{P}'$ such that $(P \oplus c) \cap (P' \oplus c') \neq \emptyset$.

$\boxed{\text{Procedure 3}}$ Given a set of integer points $\mathcal{Z}$ and a collection of pairs $(c^{(1)}, w^{(1)}), \ldots,$ $(c^{(t)}, w^{(t)})$ as input ($n \in \mathbb{Z}, n \geq 2$), this procedure outputs a new collection of points $\mathcal{Z}'$ such that every point z in $\mathcal{Z}'$ passes the n given challenges (i.e. $f(z, c^{(k)}) = w^{(k)}$ for all $z \in \mathcal{Z}'$).

The rest of this section goes as follows, the three first forth–coming subsections correspond to the three algorithmic tools we just described. On the fourth and last subsection we explain how are these tools used in the attack and prove the attack effective.

3.1 From brute–force to brute forge

In the preceding paragraphs of this section we have stated the input/output of Procedure 1. This subsection is devoted to this procedure we call algorithmic tool number 1, which we follow to describe. As explained, Procedure 1, from a pair of challenge and response (c, w) produces a set of polygons $\mathcal{P}$ containing all the hashed passwords passing the same challenge, i.e. $\mathcal{P}$ is such that $f^{-1}(w) = \cup_{P\in\mathcal{P}}(P \oplus c)$. We explain why are the elements of $\mathcal{P}$ convex polygons and why is it the case that this is a natural way of representing this particular set of points (i.e. $f^{-1}(w)$), subsequently we describe the algorithm underlying these procedure.

For any (hashed) password p, and a (hashed) challenge c, the corresponding response is calculated by the process described in the previous section. To invert this process, we inspect the one–way function f more closely. Suppose with out loss of generality that $w_1, \ldots, w_8)$ are known (e.g. w_9 is known). Later in this section, in Remark 2, we justify this supposition by explaining how is this problem tackled.

From the definition of the $w_1, \ldots w_8$ it follows that $64 \leq w_i < 96$ (and for w_9 it is $0 \leq w_9 < 32$). For the input $X = p_1 \oplus c_1$ and $Y = p_2 \oplus c_2$, it holds that the entries $w_i \in \mathbb{Z}$ verify a certain formula of the form $w_i = \lfloor \frac{31}{n}\big((\alpha_i \cdot X + \beta_i \cdot Y + \gamma_i \cdot 33) \bmod (n)\big)\rfloor + 64$ for some integers $\alpha_i, \beta_i, \gamma_i \in \mathbb{Z}$, where the $\alpha_i, \beta_i, \gamma_i$ can be calculated for once and for all. For example,

$$w_1 = \left\lfloor \frac{31}{n}(3X + Y \bmod (n)) \right\rfloor + 64,$$

$$w_2 = \left\lfloor \frac{31}{n}(12X + 5Y + 33 \bmod (n)) \right\rfloor + 64,$$

$$\vdots$$

$$w_8 = \left\lfloor \frac{31}{n}(322863X + 140206Y + 33 \cdot 42450 \bmod (n)) \right\rfloor + 64,$$

$$w_9 = \left\lfloor \frac{31}{n}(1389207X + 603275Y + 33 \cdot 182656 \bmod (n)) \right\rfloor.$$

Notice that for the floor operation it holds that $\lfloor x \rfloor \leq x < \lfloor x \rfloor + 1$. Then from the value of w_1, we deduce that the inequation

$$\frac{n}{31}(w_1 - 64) \leq 3X + Y \bmod (n) < \frac{n}{31}((w_1 - 64) + 1)$$

holds, i.e. there exists an integer $\delta_1 \in \mathbb{Z}$ such that $\frac{n}{31}(w_1 - 64) + \delta_1 n \leq 3X + Y < \frac{n}{31}((w_1 - 63) + \delta_1 n$. Furthermore, form the fact that $0 \leq X, Y < 2^{32}$ we deduce that $0 \leq \delta_1 \leq \lceil \frac{4 \cdot 2^{32}}{n} \rceil - 1 = 16$.

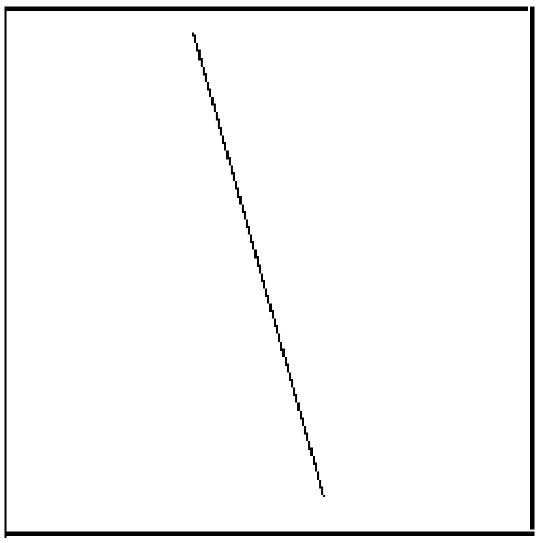

Figure 1: A polygon

For $w_2, \ldots, w_8$ similar equations hold. By a similar process to the one we just applied to the defining equation of w_1, we deduce that

$$\frac{n}{31}(w_i - 64) + \delta_i n \leq \alpha_i \cdot X + \beta_i Y + \gamma_i \cdot 33 < \frac{n}{31}((w_i - 63) + \delta_i n \qquad (1)$$

where here the bounds for the δ_i can be analogously deduced, i.e., it follows that $\delta_i < \left\lceil \frac{2^{32}(\alpha_i+\beta_i)+33\gamma_i}{n} \right\rceil$.

In this way, an attacker, for each choice of $\delta_1, \ldots, \delta_8$ is able to construct the convex polygon

$$P_\delta := \bigcap_{1\leq i\leq 8} \Big\{(x,y) \in \mathbb{R}^2 : \frac{n}{31}(w_1 - 64) + \delta_i n \leq \alpha_i \cdot X + \beta_i Y + \gamma_i \cdot 33 < \\ < \frac{n}{31}((w_1 - 63) + \delta_i n\Big\},$$

seen in Picture 1. We thus see that for every $\delta = (\delta_1, \ldots, \delta_8)$, for every integer point (a,b) in $P_\delta \cap \mathbb{Z}^2$ it holds that $f(c,(a,b)) = w$. In fact it is easy to see that the other inclusion also holds, i.e., for every pair (a,b) which is mapped via f to the checksum s there exist a tuple $\delta = (\delta_1, \ldots, \delta_8)$ such that (a,b) belongs to P_δ.

We shall see that many choices of the δ will define the same polygon. Furthermore, we prove that $\mathcal{P} := \cup P$ (where the union is taken over all the possible tuples $\delta_1, \ldots, \delta_8 \in \mathbb{Z}$), is a collection of 36 or 48 polygons in the next remark.

Remark 1 *Let $\mathcal{P}$ be defined as above. Then, each $P \in \mathcal{P}$ is a traslation of the other, i.e. for every $P, P' \in \mathcal{P}$ there exists $v \in \mathbb{Z}^2$ such that $P = P' + v$. Furthermore, it holds that $\#\mathcal{P}$ is equal to either* 36 *or* 48*.*

Proof: The traslation statement is straight–forward . What the procedure we described does, is constructing polygons by intersecting the convex polygons defined by equations (1) and the square $[0, 2^{32}] \times [0, 2^{32}]$. The different polygons that appear in $\mathcal{P}$ are generated by the different choices of k_i in the equations (1) and nothing else, this choices of k_i shift the different defining lines $k_i n$ upwards. The tangents of the different lines (which define the polygons) respective to $w_1, \ldots, w_8$ are for every choice of challenge and password $3, 2.4, 2.3181, 2.3052, 2.3031, 2.3028, 2.3027, 2.3027$ all rounded to the fourth decimal.

To prove the second statement, let P be a polygon in $\mathcal{P}$ with vertices $(a_1, b_1), \ldots, (a_t, b_t)$ it is easy to see that the polygon with vertices $(a_1 + i\frac{n}{3}, b_1 + jn), \ldots, (a_n + i\frac{n}{3}, b_n + jn)$ can be produced by a different choice of δ, for $i, j \in \mathbb{Z}$. Before proving this, we notice that we are interested only in those polygons which have a nonempty intersection with the square $[0, 2^{32} - 1] \times [0, 2^{32} - 1]$ where the password is located. Since $\lfloor 2^{32}/\frac{n}{3} \rfloor = 12$ and $\lfloor 2^{32}/n \rfloor = 4$, we deduce that $\#P \sim 48$. □

However, for our computational aims, we notice that the δ_i can be more accurately bounded by the formula re-studying the calculation process for the w_i. The best bound for δ_1 is the already calculated $\delta_1 \leq 16$. For $1 \leq i \leq 8$ we write $s_1^{(i)} := 3s_1^{(i-1)} + s_2^{(i-1)} \bmod (n)$, and $s_2^{(i)} := s_1^{(i-1)} + s_2^{(i-1)} + 33 \bmod (n)$ (where $s_1^{(0)} = X, s_2^{(0)} = Y$). For $1 \leq i \leq 8$, denote by ϵ_i and ϵ'_i the integers such that $s_1^{(i)} = 3s_1^{(i-1)} + s_2^{(i-1)} - \epsilon_i n$, and $s_2^{(i)} = s_1^{(i)} + s_2^{(i-1)} + 33 - \epsilon'_i n$. Since $0 \leq s_1^{(i)}, s_2^{(i)} < n$, it follows that $0 \leq \epsilon_i \leq 3$ and $0 \leq \epsilon'_i \leq 2$ and $s_2^{(1)} = s_1^{(1)} + s_2^{(i-1)} + 33 - \epsilon'_i n$.

Fix a user $\mathcal{U}$ with hashed password p. Denote by $F(p, c)$ the function that from a (hashed) password P and a (hashed) challenge c produces the checksum w as explained in the previous section. Suppose that a pair (c, w) of challenge and response corresponding to user $\mathcal{U}$ is known (e.g., to the attacker). Denote, as in the previous section, the (32) more significant bits of p and c by p_1 and c_1, and the (32) least significant bits by p_2, c_2 respectively.

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, since this result is totally dependent on the specific parameters used for this authentication scheme. However, we do

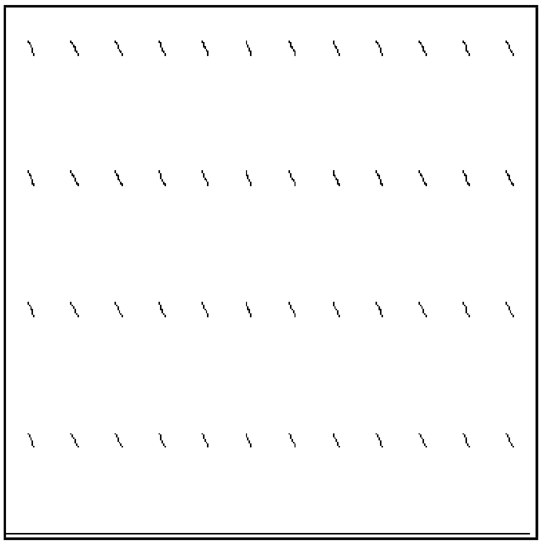
Figure 2: The polygons collection $\mathcal{P}$

give a mild justification. Suppose that we have made our choice for a w_9 candidate, say $\hat{w}_9$, and that it is a wrong choice. Suppose furthermore that the δ_i are already chosen. Then the polygon $\mathcal{P}$ defined by this choices is

$$\mathcal{P} = \bigcap_{1\leq i\leq 8} \Big\{(x,y) \in \mathbb{R}^2 : \frac{n}{31}(w_i \oplus w_9 \oplus \hat{w}_9 - 64) + \delta_i n \leq \alpha_i \cdot X + \beta_i Y + \gamma_i \cdot 33 < \frac{n}{31}((w_i \oplus w_9 \oplus \hat{w}_9 - 63) + \delta_i n\Big\}.$$

Notice that each of the sets defined between brackets appearing in the above intersection is a polygon, furthermore the two underlying lines at each intersection $\frac{n}{31}(w_i \oplus w_9 \oplus \hat{w}_9 - 64) + \delta_i n = \alpha_i \cdot X + \beta_i Y + \gamma_i \cdot 33$ and $\frac{n}{31}(w_i \oplus w_9 \oplus \hat{w}_9 - 63) + \delta_i n = \alpha_i \cdot X + \beta_i Y + \gamma_i \cdot 33$ are at a vertical distance of an integer factor of $\frac{n}{31}$. When the equations come out *incorrectly* the polygon they define is shifted vertically (upwards or downwards) a distance which is a factor of $\frac{n}{31}$. This last fact, together with the fact explained in the previous remark that the tangents of the lines of each of these polygons are quite similar implies our assertion.

In Figure 2 we can see an image of the result of Procedure 1, four *rows* of twelve polygons each. As we explained, this is a typical behavior.

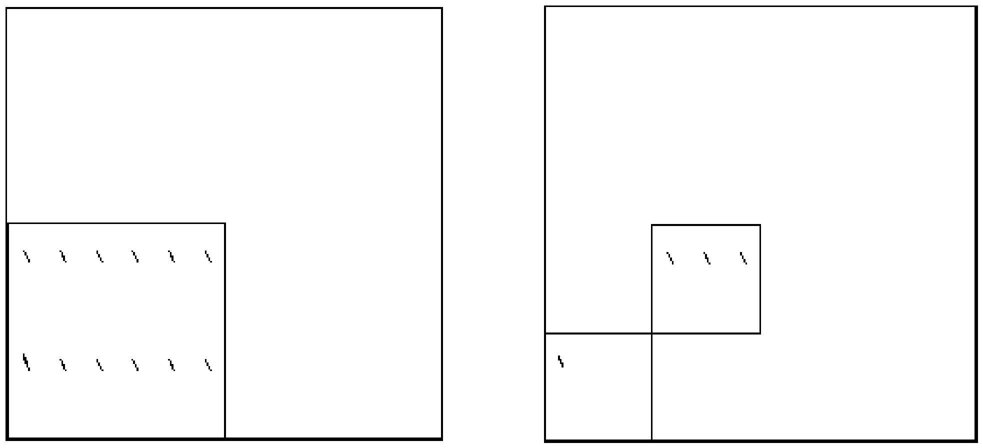

Figure 3: The second and third steps

3.2 Wash out of invalid passwords

Let be given a collection $(c, w, \mathcal{P}), (c', w', \mathcal{P}')$ and an integer k, $1 \le k \le 32$, such that (c, w) and (c', w') are two pairs of challenge and response, and that $\mathcal{P}$ and $\mathcal{P}'$ are the set of polygons respective to the given pairs of challenge and response, and as produced with the Procedure 1.

To describe the output we define some notation. For every $0 \le i, j < 2^k$, let

$$Q \quad := \quad \left[i \cdot 2^{32-k}, (i+1)\cdot^{32-k}\right] \times \left[j \cdot 2^{32-k}, (j+1) \cdot 2^{32-k}\right].$$

And let $\mathcal{Q}$ denote the set of all these squares. Then, let $\mathcal{P}_1$ denote the set $\mathcal{P}_1 := \cup (P \cap Q)$, where the union is taken over all $Q \in \mathcal{Q}, P \in \mathcal{P}$. Then the output is the subset of $\mathcal{P}_1$ of all $P_1 \in \mathcal{P}_1$ for which there exists a $P' \in \mathcal{P}'$ such that

$$(P \oplus c) \cap (P' \oplus c') \neq \emptyset,$$

this means that there exists a point $(a_1 \oplus c, b_1 \oplus c) = (a' \oplus c', b' \oplus c')$ for some $(a, b) \in P$ and $(a', b') \in P'$ (so that $f(a, b) = s$ and $f(a', b') = s'$).

The output is a collection of polygons $\widetilde{\mathcal{P}}$ consisting of all the polygons $\widetilde{P} = \bigcup (P \cap Q)$, the union taken over all the squares Q of side 2^{32-k} and vertices with entries in $\{i \cdot 2^k : 0 \le i \le 2^{32-k}\}$, and all the polygons $P \in \mathcal{P}$ in $\mathcal{P}$ for which there exists a $P' \in \mathcal{P}'$ such that $(P \oplus c) \cap (P' \oplus c') \neq \emptyset$. In Figure 2, we see two steps of this procedure over the chosen example.

Notice that we do not calculate $\cup_{P \in \mathcal{P}, P' \in \mathcal{P}'} P \cap P'$, this would be inefficient, e.g. it would make the size of the output grow and would not make much of a difference in the point size of the output.

3.3 Rising of passwords

Procedure 3 is straight forward and needs not much explanation. Given a set of points and a pair of challenge and response, ones simply browses over every point of this set calculating the checksum corresponding to this point and the given challenge and adds it to the output set only if it produces the given response.

3.4 The complete algorithmic attack

A complete attack to a valid user, who has produced the pairs of challenge and response $(c_1, w_1), \dots, (c_t, w_t)$ is done by repeated application of the procedures we have just described. To start with, we select the number $k \leq t$ of these challenge and response pairs to which we are going to apply to Procedure 1. That is for the pairs $(c_1, w_1), \dots, (c_k, w_k)$ we apply the Procedure 1and output collections $\mathcal{P}_1, \dots, \mathcal{P}_k$. Typically —on our examples— the number k is taken to be 5 or less.

On a second step, after selecting a second integer $1 \leq k' \leq 32$, we apply Procedure 2 recursively to the triplets $(c^{(1)}, w^{(1)}, \mathcal{P}^{(1)}), \dots, (c^{(k)}, w^{(k)}, \mathcal{P}^{(k)})$; that is first we apply Procedure 2 to $(c^{(1)}1, w^{(1)}, \mathcal{P}^{(1)})$ and $(c^{(2)}, w^{(2)}, \mathcal{P}^{(2)})$ using cubes of size $2^{32-k'}$, then we apply Procedure 2 to the previous result and $(c^{(3)}, w^{(3)}, \mathcal{P}^{(3)})$, and continue this recursive application until the k–th tuple is reached. After doing this (the k applications of this procedure) we get a set of polygons $\widetilde{\mathcal{P}}$ having a small number of integer points (compared to the brute–force value 2^{64}).

Finally, we recursively apply Procedure 3 to the set $\widetilde{\mathcal{P}}$ and each of the remaining challenge and response pairs $(c^{(k+1)}, w^{(k+1)}), \dots, (c^{(t)}, w^{(t)})$ as follows. First we extract every integer point of $\widetilde{\mathcal{P}}$ and store them as points. Then we use Procedure 3 with the resulting set as input and $(c^{(k+1)}, w^{(k+1)})$. After we have finished with $(c^{(k+j)}, w^{(k+j)})$, for $j \geq 1$, we continue by applying Procedure 3 to the resulting set and $(c^{(k+j+1)}, w^{(k+j+1)})$. We end when we $(c^{(t)}, w^{(t)})$ is reached and there are no more challenge and response pairs left, or before if the set of remaining points has only one point left. In the latter case we have found the password's hash. Else, we get as output the set of passwords' hashes that pass every one of the challenge and response pairs we have as input. It should be remarked, and shall be furtherly emphasized by empiric data in the next section, that in the case of the output being a set of more than one point, all of these points are not just random points in $[0, 2^{32}] \times [0, 2^{32}]$, but have high probability of passing an additional test.

4 Statistics and Conclusions

We coded the algorithm of the preceding section in Squeak Smalltalk, and analyzed the results. In the examples tested, about 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. It took about 100 pairs of challenge and response to cut the 300 points set to a set containing 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.

In other examples we used only ten pairs of challenge and response, getting thus a set of approximately 300 points in each case. Then we randomly selected 1000 challenges and made the 1000 tests to every one of the 300 points we had. The result was that each of the remaining points passed over 920 of the 1000 tests. That is, the mean (sample–mean) of the probability a point (in the set left after applying our algorithm to ten pairs of challenge and response) is of 92%.

We therefore are able to make a variety of attacks depending on the amount of pairs of challenge and response we get from the user we want to impersonate. The two extremal cases being very few pairs of challenge and response from the same user, and a lot of pairs of challenge and response. The second attack, that of many pairs of challenge and response captured, is straight–forward: apply the algorithm described above until the password is found. The first case, that of only a few pairs of challenge and response captured, is as well easy to carry: simply apply the algorithm we described with all the pairs of challenge and response captured, then use any possible password in the set produced by the application of the algorithm for authenticating yourself as a user (some of these fake passwords will still pass many tests!).

We do not analyze the order of the complexities of our procedures because we are working with an input of fixed size. Since every one of the attacks we made was much alike in the performance/computation–time aspect, we describe a specific example of an attack pointing out the computation time needed at each step in the attack, and the amount of points left (in the 2^{64} to 1 countdown of possible hashed passwords). In a specific example we applied Procedure 1 to a pair of challenge and response calculating 48 polygons, each a traslation of the other, of area 2^{33}, which makes a set of area approximately 2^{38}. The whole procedure application lasted no more than half an hour in a Pentium 3 64Mb RAM personal computer. We applied Procedure 1 to four other pairs of challenge and response obtaining similar results. This constituted the first two hours and a half of the attack. The four applications of Procedure 2 lasted half an hour each, and resulted in a collection of 32 polygons of area approximately 2^{16} each, i.e. a set of size 2^{21}. Finally we applied Procedure 3 getting the announced 300 possible hashed passwords in about six hours and using only 10 pairs of challenge and response. The complete attack lasted approximately twelve hours, in this case we had available some 300 more challenge and response pairs and were able to recover the hashed password in five more minutes.

Credits: This vulnerabilities were found and researched by Agustin Azubel, Emiliano Kargieman, Gerardo Richarte, Carlos Sarraute and Ariel Waissbein of CORE Security Technologies, Buenos Aires, Argentina.

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].

References

[1] Agustin Azubel, Emiliano Kargieman, Gerardo Richarte, Carlos Sarraute, and Ariel Waissbein. MySQL authentication vulnerability. Bugtraq advisory (BID 1826).

[2] MySQL Database Engine. See `http://www.mysql.com`

[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. Bugtraq advisory (BID 926).

[5] Internet Engineering Task Force (IETF). SSH authentication protocol. Internet draft.

[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. The 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. Bugtraq advisory (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. Refer to `http://www.squeak.org`

[12] Robert van der Meulen. MySQL unauthenticated remote access vulnerability. Bugtraq advisory (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.