

An attack on the MySQL login protocol

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

January 24, 2002

Abstract

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

1 Introduction

The use of computer-based user authentication has become a cryptographic tool widely used these days. Every remote connection (SSH, SSL, et cetera) is initiated with a user authentication. This 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 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 protocols (e.g., Fiat–Shamir [4], Guillon–Quisquater [7] or Schnorr [9, 10] identification protocols), or challenge–response protocols (e.g., see Kerberos [8], the Secure Shell authentication protocols [6] see also [12]).

`MYSQL DATABASE ENGINE` package ([1]) is a popular open source database engine capable of 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 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 an authentication procedure. When a client authenticates themselves to the server and starts a session, he

can obtain 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 the authenticated client. Different parameters as to how this is done can be selected by server administrators. However all these possible configurations have the same authentication procedure in common.

The authentication protocol was 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 was 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. The authentication procedure we describe here is extracted from the source code¹ implemented in 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. First 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 and in the server. But moreover we shall see that each time a user underpasses a challenge–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 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 in an almost exponential manner. 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 witnessing only 10 authentications!

Previous vulnerabilities in the MySQL DATABASE ENGINE were disclosed in the Bugtraq advisories [11] and [5]. An advisory authored by the authors which briefly describes this attack appeared as a Bugtraq Advisory in [3].

¹available at <http://www.mysql.com>

2 Technical Description

2.1 The challenge–response mechanism

The authentication protocol of MySQL DATABASE ENGINE package is of the challenge–response family, 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 package provides users with two primitives used for the authentication protocol:

- a hash function
- a (supposedly) one–way function

both of which are 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 uses the hash value of the random string he has received and the hash value of his password as input to the one–way function and calculates a new string —the response— which is sent to the server.

This *response* 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 successfully.

2.2 Problem Description

The hash function provided by MySQL outputs an eight-bytes string, making 2^{64} possibilities for the hashed password. Whereas the one–way function outputs eight-byte strings, but has by construction a range set of only 2^{45} possible outputs (with a fixed input size of 8 bytes). From this fact, we deduce that more than one hashed password produces the same (expected) response for a given challenge, e.g. there are collisions. 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 should also point out that the authentication mechanism of MySQL does not require the password for a successful authentication, but the password's hash value. To impersonate a user only the hash value of this user's password is needed, so the hash function is of no interest for this account.

To validate our claim, we explain how an eavesdropper can impersonate a user using only a few executions of the challenge–response mechanism for the same user. More explicitly, in the forth coming section we exploit this weakness, and construct an attack much more efficient than brute–force, which can be carried out in a few hours on a personal computer (see Section 4). After gaining a positive number of pairs of challenge–response, an eavesdropper is able to efficiently calculate the set of values, standing for hashed passwords, that produce the correct responses for the intercepted challenges.

To do this, we first describe how the MySQL one–way function works, and proceed to analyze the scheme's security, and then describe the attack we have 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 [2]) 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 3.1 and 3.2) are cut–and–pasted black and white screen images of this implementation.

Let $n := 2^{30} - 1$. Fix a user \mathcal{U} and initiate a challenge–response run, 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 one–way function used. Let \oplus denote the bitwise exclusive or (X–or) operation.

1. Let $s_1^{(0)} := p_1 \oplus c_1$ and $s_2^{(0)} := p_2 \oplus c_2$. The values $s_1^{(0)}$ and $s_2^{(0)}$ are then used as input for the one–way function,

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

$$s_1^{(i)} := s_1^{(i-1)} + 3 \cdot s_2^{(i-1)} \quad \text{modulo } (n)$$

$$s_2^{(i)} := s_1^{(i)} + s_2^{(i-1)} + 33 \quad \text{modulo } (n)$$

$$w_i := \left\lfloor \frac{31 \cdot s_1^{(i)}}{n} \right\rfloor + 64$$

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

3. Finally let

$$s_1^{(9)} := s_1^{(8)} + 3 \cdot s_2^{(8)} \quad \text{modulo } (n)$$

$$w_9 := \left\lfloor \frac{31 \cdot s_1^{(9)}}{n} \right\rfloor$$

4. output the response value

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

It is this response $w \in \{0, 1\}^{64}$ that is sent by the user \mathcal{U} to the server. The server, that has stored the hash value of \mathcal{U} 's password, parallelly calculates this response by this same process and verifies if equality holds with the value it has received. Our attack statement is that eavesdropping a small collection of these challenges and responses that travel on the wire between server and client enables the attacker to impersonate the user.

The reason why the process of producing the response 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 response value $f(X, Y) = w$ (e.g., in our case $X := p_1 \oplus c_1$ and $Y := p_2 \oplus c_2$). We can then efficiently calculate all of the values X', Y' which map to the same response 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-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. 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 divided into three main stages. In these stages we respectively use one of our three algorithmic tools in various opportunities.

Procedure 1 An algorithmic process which has as input a response w , and outputs a set of convex polygons $\mathcal{P} = \{P\}$ such that

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

Procedure 2 The input is a pair of tuples $(c, \mathcal{P}), (c', \mathcal{P}')$, where c and c' are different pairs of challenges, with respective responses w and w' such that $f^{-1}(w) = \bigcup_{P \in \mathcal{P}} P$ and $f^{-1}(w') = \bigcup_{P' \in \mathcal{P}'} P'$, and such that \mathcal{P} and \mathcal{P}' are compliant with the four conditions stated above. The output is a collection of smaller polygons $\tilde{P}_{i,j} = (P \cap Q_{i,j})$ for which there exists $P' \in \mathcal{P}'$ such that $(Q_{i,j} \oplus c \oplus c') \cap P' \neq \emptyset$.

Procedure 3 Given a set of integer points \mathcal{Z} and a collection of pairs $(c^{(1)}, w^{(1)}), \dots, (c^{(n)}, w^{(n)})$ 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. $f(z_1 \oplus c_1^{(k)}, z_2 \oplus c_2^{(k)}) = w^{(k)}$ for all $z \in \mathcal{Z}', 1 \leq k \leq n$).

The rest of this section goes as follows: the three first forthcoming subsections correspond to the three algorithmic tools we just described. On the fourth and last subsection we show how to use these tools in the attack, we prove the attack effective, and briefly analyze attack strategies.

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–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 $f^{-1}(w) = \bigcup_{P \in \mathcal{P}} P$. We show that the elements of \mathcal{P} convex polygons and deduce that this is a natural way of representing this particular set of points (i.e. $f^{-1}(w)$), subsequently we describe the algorithm underlying this 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, \dots, 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.

By construction of the w_1, \dots, w_8 it follows that $64 \leq w_i < 96$ (and w_9 verifies $0 \leq w_9 < 32$). For the input (X, Y) , where $X = p_1 \oplus c_1$ and $Y = p_2 \oplus c_2$, it holds that the entries $w_i \in \mathbb{Z}$ of w verify a formula of the form $w_i = \lfloor \frac{31}{n} ((\alpha_i X + \beta_i Y + \gamma_i 33) \bmod (n)) \rfloor + 64$ for some integers $\alpha_i, \beta_i, \gamma_i \in \mathbb{Z}$, where the $\alpha_i, \beta_i, \gamma_i$ can be calculated once and for all. Rewriting the recursive formulas for the w_i in terms of X, Y we get the equalities

$$\begin{aligned} w_1 &= \left\lfloor \frac{31}{n} (3X + Y \bmod (n)) \right\rfloor + 64, \\ w_2 &= \left\lfloor \frac{31}{n} (12X + 5Y + 33 \bmod (n)) \right\rfloor + 64, \\ &\vdots \\ w_8 &= \left\lfloor \frac{31}{n} (322863X + 140206Y + 33 \cdot 42450 \bmod (n)) \right\rfloor + 64, \\ w_9 &= \left\lfloor \frac{31}{n} (1389207X + 603275Y + 33 \cdot 182656 \bmod (n)) \right\rfloor. \end{aligned}$$

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 $X, Y \in \mathbb{Z}^2$ belongs to the following semi-algebraic set

$$\left\{ (x, y) : 0 \leq x, y < 2^{32}, \frac{n}{31}(w_1 - 64) \leq 3x + y \bmod (n) < \frac{n}{31}((w_1 - 64) + 1) \right\}$$

must hold, 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, from the fact that $0 \leq X, Y < 2^{32}$ we deduce that $0 \leq \delta_1 \leq \lceil \frac{4 \cdot 2^{32}}{n} \rceil - 1 = 16$.

For w_2, \dots, w_8 similar equations hold. By an analogous process to the one we have just applied to the defining equation of w_1 , we deduce for any i , with $1 \leq i \leq 8$, that the point (X, Y) belongs to each of the semi-algebraic sets

$$\begin{aligned} \left\{ (x, y) : 0 \leq x, y < 2^{32}, \frac{n}{31}(w_i - 64) + \delta_i n \leq \alpha_i x + \beta_i y + 33 \gamma_i < \right. \\ \left. < \frac{n}{31}(w_i - 63) + \delta_i n \right\} \quad (1) \end{aligned}$$

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

Figure 1: A polygon

In this way, an attacker, for each choice of $\bar{\delta} := (\delta_1, \dots, \delta_8)$ is able to construct the convex polygon

$$P_{\bar{\delta}} := \bigcap_{1 \leq i \leq 8} \left\{ (X, Y) \in \mathbb{R}^2 \cap ([0, 2^{32}] \times [0, 2^{32}]) : \frac{n}{31}(w_i - 64) + \delta_i n \leq \leq \alpha_i X + \beta_i Y + \gamma_i 33 < \frac{n}{31}(w_i - 63) + \delta_i n \right\},$$

seen in Figure 1. We thus see that for every $\bar{\delta} \in \mathbb{Z}^8$, for every integer point (a, b) in $P_{\bar{\delta}} \cap \mathbb{Z}^2$ it holds that $f(a, b) = w$. 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 response w there exist a tuple $\bar{\delta}$ such that (a, b) belongs to $P_{\bar{\delta}}$.

Many choices for the $\bar{\delta}$ actually define the same polygon in \mathbb{Q}^2 . We prove in the next remark that the subset $\mathcal{P} := \cup P_{\bar{\delta}}$ of $[0, 2^{32} - 1] \times [0, 2^{32} - 1]$ (where the union is taken over all the possible tuples $\bar{\delta} \in \mathbb{Z}^8$), is a collection of 48 or 36 polygons.

Remark 1 *Let the above notations and assumptions hold. Let $\mathcal{P} = \cup_{\bar{\delta}} P_{\bar{\delta}}$. Then the following conditions hold*

- *each $P \in \mathcal{P}$ is a traslation of the other, i.e. for every $P_{\bar{\delta}}, P_{\bar{\delta}'} \in \mathcal{P}$ there exists $v \in \mathbb{Z}^2$ such that $P = P' + v$.*
- *the number pf polygons in \mathcal{P} is either 48 or 36, with each polygon $P_{\bar{\delta}} \in \mathcal{P}$ containing at most $\#(\mathbb{Z}^2 \cap P_{\bar{\delta}}) \leq 2^{36}$ integer points.*

Proof: Each polygon P is, by construction, an intersection of the convex polygons defined by equations (1) and the square $[0, 2^{32} - 1] \times [0, 2^{32} - 1] \subset$.

To simplify notations, let us consider the objects in $(\mathbb{R}/2^{32}\mathbb{Z}) \times \mathbb{R}$ instead of $\mathbb{R} \times \mathbb{R}$.

To construct the polygons, we first construct strips according to equations $S_{\delta_1}^{(1)} := \{(X, Y) : \frac{n}{31}(w_1 - 64) + \delta_1 n \leq 3X + Y < \frac{n}{31}(w_1 - 63) + \delta_1 n\}$ where δ_1 is variable. Now to determine how many of these strips we will obtain, it suffices to see how many of them intersect the X-axis between 0 and $2^{32} - 1$. Since $\lfloor (2^{32} - 1)/\frac{n}{3} \rfloor = 12$, there are exactly 12 strips.

Then we construct strips according to $S_{\delta_2}^{(2)} := \{(X, Y) : -33 + \frac{n}{31}(w_2 - 64) + \delta_2 n \leq 12X + 5Y < -33 + \frac{n}{31}(w_2 - 63) + \delta_2 n\}$ where δ_2 is variable. Examining again the intersections with the X-axis, we find $\lfloor (2^{32} - 1)/\frac{n}{12} \rfloor = 48$ strips.

The left boundaries of the first strips are lines L_{δ_1} of slope -3 which intersect the X-axis at distances of $n/3$. The left boundaries of the second strips are lines L'_{δ_2} of slope $-12/5$. It is easy to see that each line L'_{δ_2} intersects exactly one line L_{δ_1} . So at this stage, by intersecting the first and second strips, we obtain 48 parallelograms. It is clear that they are all translations of each other.

Now writing more succinctly the equations of our lines as: $3X + Y = k\frac{n}{31}$; $3X + Y = (k + 1)\frac{n}{31}$; $12X + 5Y = l\frac{n}{31} - 33$; $12X + 5Y = (l + 1)\frac{n}{31} - 33$; we obtain expressions for the four vertices of the parallelograms:

$$\begin{aligned} X &= (5k - l) \cdot n / (3 \cdot 31) + 11 & Y &= (l - 4k) \cdot n / 31 - 33 \\ X &= (5k - 1 - l) \cdot n / (3 \cdot 31) + 11 & Y &= (l + 1 - 4k) \cdot n / 31 - 33 \\ X &= (5k + 5 - l) \cdot n / (3 \cdot 31) + 11 & Y &= (l - 4 - 4k) \cdot n / 31 - 33 \\ X &= (5k + 4 - l) \cdot n / (3 \cdot 31) + 11 & Y &= (l - 3 - 4k) \cdot n / 31 - 33 \end{aligned}$$

Using these expressions, we can see that each parallelogram can be intersected by only one strip $S_{\delta_3}^{(3)} := \{(X, Y) : -6 \cdot 33 + \frac{n}{31}(w_3 - 64) + \delta_3 n \leq 51X + 22Y < -6 \cdot 33 + \frac{n}{31}(w_3 - 63) + \delta_3 n\}$, so that by intersecting the parallelograms with the third strips we obtain 48 polygons. In a similar way, the result of intersecting the resulting polygons with the strips $S_{\delta_4}^{(4)}, \dots, S_{\delta_8}^{(8)}$ will be to successively refine the polygons. (See the table for all the equations.) We finally obtain at most 48 polygons, translations of each other.

To evaluate the area of the polygons, notice that each polygon is included in the parallelogram determined by the lines $3X + Y = k\frac{n}{31}$; $3X + Y = (k + 1)\frac{n}{31}$; $75036X + 32585Y = l\frac{n}{31} - 33$; $75036X + 32585Y = (l + 1)\frac{n}{31} - 33$; the area of these parallelograms is approximatively $5.2806 \cdot 10^{10} \approx 2^{35.6}$ and thus $\#(\mathbb{Z}^2 \cap P) \leq 2^{36}$. \square

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

i	α_i	β_i	γ_i	α_i/β_i
1	3	1	0	3.0
2	12	5	1	2.4
3	51	22	6	2.31818181818182
4	219	95	28	2.30526315789473
5	942	409	123	2.30317848410758
6	4053	1760	532	2.30284090909091
7	17439	7573	2292	2.30278621418196
8	75036	32585	9865	2.30277735154212
9	322863	140206	42450	2.30277591543871
10	1389207	603275	182656	2.30277568273175

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 $s_2^{(i)} = s_1^{(i-1)} + s_2^{(i-1)} + 33 - \epsilon'_i n$. 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 $s_2^{(1)} = s_1^{(1)} + s_2^{(i-1)} + 33 - \epsilon'_i n$.

Remark 2 *By applying the algorithmic procedure just described to $w[k] = (w_1 \oplus k \parallel \dots \parallel w_8 \oplus k)$ 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; however, we do 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

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

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

Figure 2: The polygons collection \mathcal{P}

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.

3.2 Wash out of invalid passwords

Let be given a collection $(c, w, \mathcal{P}), (c', w', \mathcal{P}')$ and an integer $k, 1 \leq k \leq 32$, 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 $0 \leq i, j < 2^k$, let

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

And let \mathcal{Q} denote the set of all these squares. Then, let $\tilde{\mathcal{P}}$ denote the set $\tilde{\mathcal{P}} := \cup (P \cap Q_{i,j})$, where the union is taken over all $Q_{i,j} \in \mathcal{Q}, P \in \mathcal{P}$.

If the (hash of the) password is $p = (p_1, p_2)$, we know that there exist $P \in \mathcal{P}$ and $Q_{i,j} \in \mathcal{Q}$ such that $(x, y) = (p_1 \oplus c_1, p_2 \oplus c_2) \in P \cap Q_{i,j}$. There also exist $P' \in \mathcal{P}'$ such that $(x', y') = (p_1 \oplus c'_1, p_2 \oplus c'_2) \in P'$. The relation between (x, y) and (x', y') is:

$$\begin{aligned} x' &= x \oplus c_1 \oplus c'_1 \\ y' &= y \oplus c_2 \oplus c'_2 \end{aligned}$$

Figure 3: The second and third steps

So if we xor the points of the square $Q_{i,j}$ with the first k bits of $(c_1 \oplus c'_1, c_2 \oplus c'_2)$, we will obtain a square of the same size that must intersect P' .

This is the motivation of procedure 2: given $P \in \mathcal{P}$, for each $\tilde{P}_{i,j} = P \cap Q_{i,j} \in \tilde{\mathcal{P}}$, we determine if the xor of the square $Q_{i,j}$ with the first k bits of $(c_1 \oplus c'_1, c_2 \oplus c'_2)$ intersect one of the polygons $P' \in \mathcal{P}'$. If not, then we reject $\tilde{P}_{i,j}$.

The output is a subset of $\tilde{\mathcal{P}}$ consisting of all the smaller polygons $\tilde{P}_{i,j} = P \cap Q_{i,j}$ which verify the xor condition. In Figure 2, we see two steps of this procedure over the chosen example.

Notice that we can tune the choice of k , since a value of k too small will make the number of output polygons be too big (and result in storage problems). Whereas a value of k too big simply won't produce any refining of our polygon collection. When we apply successively procedure 2 to different pairs of challenge–response, we start with $k = 12$ then gradually reduce it.

3.3 Rising of passwords

Procedure 3 is straight forward and needs not much explanation. Given a set of integer points \mathcal{Z} and a collection of pairs $(c^{(1)}, w^{(1)}), \dots, (c^{(n)}, w^{(n)})$ as input, ones simply browses over every point of \mathcal{Z} calculating the responses corresponding to this point and the challenges $c^{(k)}$ and adds it to the output set only if it produces the given response $w^{(k)}$.

3.4 The complete algorithmic attack

A complete attack to a valid user, who has produced the pairs of challenge and response $(c^{(1)}, w^{(1)}), \dots, (c^{(n)}, w^{(n)})$ 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–response pairs to which we are going to apply to Procedure 1. That is for the pairs $(c^{(1)}, w^{(1)}), \dots, (c^{(p)}, w^{(p)})$ we apply the Procedure 1 and output collections $\mathcal{P}^{(1)}, \dots, \mathcal{P}^{(p)}$. Typically —on our examples— the number p is taken to be 5 or less.

On a second step, after selecting integers $32 > k_1 \geq k_2 \geq \dots \geq k_{p-1} > 1$, we apply Procedure 2 recursively to the triplets $(c^{(1)}, w^{(1)}, \mathcal{P}^{(1)}), \dots, (c^{(q)}, w^{(q)}, \mathcal{P}^{(q)})$; that is first we apply Procedure 2 to $(c^{(1)}, w^{(1)}, \mathcal{P}^{(1)})$ and $(c^{(2)}, w^{(2)}, \mathcal{P}^{(2)})$ using cubes of size 2^{32-k_1} , then we apply Procedure 2 to the previous result and $(c^{(3)}, w^{(3)}, \mathcal{P}^{(3)})$ using cubes of size 2^{32-k_2} , and continue this recursive application until the p -th tuple is reached. After doing this (the k applications of this procedure) we get a set of polygons $\tilde{\mathcal{P}}$ having a small number of integer points (compared to the brute–force value 2^{64}).

Finally, we extract every integer point of $\tilde{\mathcal{P}}$ and store them as a set of points \mathcal{Z} . Then we apply Procedure 3 to the set \mathcal{Z} with the remaining challenge–response pairs $(c^{(p+1)}, w^{(p+1)}), \dots, (c^{(n)}, w^{(n)})$. We end when $(c^{(n)}, w^{(n)})$ 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, get-

ting 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) of passing another challenge-response is of %92.

We are therefore 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 extreme cases being very few pairs of challenge and response from the same user, and numerous pairs of challenge and response. The second case is straight-forward: apply the algorithm described above until the password is found. The first case 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 (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 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 on a Pentium 3 with 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 Agustín Azubel, Emiliano Kargieman, Gerardo Richarte, Carlos Sarraute and Ariel Weissbein of CORE Security Technologies, Buenos Aires, Argentina. (URL: <http://www.corest.com>).

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 [3].

References

- [1] Mysql database engine. At <http://www.mysql.org/>, last checked 2002.
- [2] Squeak smalltalk. At <http://www.squeak.org/>.
- [3] Agustín Azubel, Emiliano Kargieman, Gerardo Richarte, Carlos Sarraute, and Ariel Waissbein. Mysql authentication algorithm vulnerability. Bugtraq advisory (BID 1826), 2000.
- [4] 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.
- [5] Viktor Fougstedt. Mysql grant global password changing vulnerability. Bugtraq advisory (BID 926).
- [6] Internet Engineering Task Force. Ssh authentication protocol. Internet draft.
- [7] 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, 1988. Springer-Verlag.
- [8] RFC1510. The kerberos network authentication service (v5). Internet Request For Comments (RFC) 1510, J. Kohl and C. Neumann, September 1993.
- [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] Robert van der Meulen. Mysql unauthenticated remote access vulnerability. Bugtraq advisory (BID 975).
- [12] Thomas Wu. The secure remote password protocol. In *Proceedings of the 1998 Internet Society Network and Distributed System Security Symposium*, pages 97–111, 1998.