



Ceci est un extrait électronique d'une publication de  
Diamond Editions :

<http://www.ed-diamond.com>

Ce fichier ne peut être distribué que sur le CDROM offert  
accompagnant le numéro 100 de **GNU/Linux Magazine France**.

La reproduction totale ou partielle des articles publiés dans Linux  
Magazine France et présents sur ce CDROM est interdite sans accord  
écrit de la société Diamond Editions.

Retrouvez sur le site tous les anciens numéros en vente par  
correspondance ainsi que les tarifs d'abonnement.

Pour vous tenir au courant de l'actualité du magazine, visitez :

<http://www.gnulinuxmag.com>

Ainsi que :

<http://www.linux-pratique.com>

et

<http://www.miscmag.com>



Par :: Philippe PRADOS :: [presse@philippe.prados.name](mailto:presse@philippe.prados.name) ::  
:: [site@philippe.prados.name](http://site@philippe.prados.name) ::

## Clavier Biométrique

Le rythme d'utilisation du clavier est une information biométrique caractérisant un individu. Nous proposons une approche biométrique économique, permettant de renforcer la sécurité d'un mot de passe.



Le mot de passe est un élément fragile de l'authentification. Il permet d'identifier un utilisateur en partant de l'hypothèse qu'il est le seul à connaître un secret. En regardant un utilisateur entrer son mot de passe, il est possible de le découvrir. Cette technique est utilisée par les voleurs de cartes bancaires. Après vous avoir observé lors de l'introduction de votre code, ils vous volent votre carte et peuvent l'utiliser sans difficulté, vidant ainsi votre compte en banque.

Pour des raisons de facilité, les utilisateurs notent leurs mots de passe sur des post-it ou les confient à des personnes dignes de confiance avant de partir en vacances. Ces usages ne sont pas recommandés, car ils brisent l'hypothèse initiale : l'individu est identifié car il est le seul à connaître le secret.

Comment limiter ces utilisations malheureuses ? Il faut pouvoir identifier réellement l'individu, et non la connaissance d'un secret que celui-ci possède. Le secret peut être volé ou communiqué à des tiers. Comme dit la maxime : « Un secret est une information que l'on transmet à une seule personne à la fois ! » Pour corriger ces dérives, les badges magnétiques possèdent généralement une photographie. Un gardien peut alors vérifier qu'il appartient bien à celui qui le présente. Le gouvernement envisage de procéder de même pour la carte vitale. Cette approche oblige à faire intervenir un humain dans le processus d'authentification, le seul capable de

vérifier la conformité de la photographie avec l'individu qui la présente.

Pour automatiser l'identification d'un individu, différentes techniques de biométrie ont été inventées. Elles permettent de mesurer une caractéristique physique, censée l'identifier de manière unique. Cela peut être une mesure de l'empreinte digitale, de l'empreinte rétinienne, du timbre de la voix, la forme du visage, etc. Ainsi, un mécanisme automatique est capable d'identifier une personne. Pour éviter les faux négatifs lors de la reconnaissance, des marges de tolérance sont pratiquées. Il existe alors plusieurs individus ayant les mêmes caractéristiques physiques. Des vrais jumeaux seront difficilement discriminables par une photographie.

Pour améliorer l'authentification, deux approches sont utilisées simultanément. Toutes les combinaisons d'authentification sont possibles parmi : je connais (mot de passe), je possède (carte magnétique), je sais faire (signature) et je suis (biométrie). Par exemple, l'authentification est renforcée par la combinaison de la lecture d'une carte magnétique et de la reconnaissance digitale.

Les technologies de biométrie sont généralement coûteuses et complexes à mettre en place. Comment proposer une authentification forte, sans technologie complexe et à moindre coût ?

Pour résoudre le « je connais », les applications exploitent généralement

un mot de passe. Pour résoudre le « je possède », il est impératif d'avoir une technologie permettant de tester la possession d'un support (carte USB, lecteur de carte magnétique, clef Wifi, etc.). Ces technologies imposent un investissement sur les postes des clients afin de pouvoir vérifier la présence du support, et obligent d'en acquérir un pour chaque client.

Pour résoudre le « je sais faire », il est possible d'utiliser des périphériques de reconnaissance de signature, mais il est plus économique de proposer un challenge à l'utilisateur. Cela est utilisé lors de l'enregistrement à des listes de diffusions. Une image brouillée est présentée à l'utilisateur. L'être humain est capable d'y extraire un ou plusieurs mots, prouvant ainsi qu'il ne s'agit pas d'une machine. Cela n'est pas suffisant pour contrer le vol du mot de passe par une autre personne.

Pour résoudre le « je suis », il est envisageable de détecter le rythme utilisé par un individu pour saisir son mot de passe. Le temps entre chaque appui de touche est caractéristique de l'individu. Bien entendu, deux personnes peuvent avoir le même rythme pour saisir le même mot, mais cela réduit considérablement les risques, et cela augmente la complexité du mot de passe. Cette approche permet de marier simultanément trois techniques : je connais, je sais faire et je suis. La seule contrainte physique est la présence d'un clavier.

Le temps nécessaire à l'introduction de chaque touche peut être mémorisé, avec le mot de passe. Pour parfaire la reconnaissance, plusieurs saisies peuvent être effectuées. Les valeurs minimums et maximales sont alors indiquées pour chaque lettre. Pendant la saisie, ces informations sont capturées et envoyées vers le serveur en même temps que le mot de passe. Après avoir vérifié la validité du mot de passe, le serveur vérifie que le temps mis par l'utilisateur entre chaque touche est conforme au modèle mémorisé. Comme pour toute technologie biométrique, une tolérance peut être ajoutée pour accepter une saisie légèrement plus rapide ou plus lente.

J'ai voulu vérifier s'il était possible de proposer une vérification biométrique

avec quelques lignes de code javascript. Il était évident que seul le navigateur pouvait identifier le temps entre chaque touche. J'ai écrit un javascript s'occupant de mémoriser le rythme lors de la saisie d'un mot de passe.

Le code est inhabituelle, car, amoureux de la technologie objet, j'ai rédigé une classe javascript et non des fonctions. Commençons par expliquer les techniques javascript utilisées par le programme. Il est possible de construire et d'initialiser un objet javascript lors de sa déclaration. Pour cela, il faut utiliser des accolades lors de l'initialisation d'une variable.

```
var monobject={
  nom: «prados»,
  prenom: «philippe»
};
```

Cette écriture permet de créer un objet possédant deux attributs, nom et prénom, et de les valoriser. Ce code est équivalent à celui-ci :


```
var monobject=new Object();
monobject.nom=«prados»;
monobject.prenom=«philippe»;
```

L'écriture compacte est plus élégante. Pour ajouter une méthode à un objet javascript, il faut déclarer une fonction et l'associer à l'objet.

```
function monobject_maMethode()
{
  // Le corps de ma fonction
}
monobject.maMethode=monobject_maMethode;
```

Il est aussi possible de construire une instance d'une fonction, directement lors de la valorisation d'une variable.

```
monobject.maMethode=function()
{
  // Le corps de ma méthode
};
```

 **Notez le point-virgule après l'accolade fermante. Cette écriture consiste à créer une fonction anonyme, et à l'associer à la méthode `maMethode` de `monobject`. Il est alors possible de l'invoquer à l'aide de `monobject.maMethode()`.**

En combinant l'écriture compacte de création d'un objet et la génération de fonction anonyme, il est possible de créer une classe javascript, aussi facilement qu'une classe java, C++ ou C#.

```
var monobject={
  nom: «prados»,
  prenom: «philippe»,
  maMethode: function()
  {
    // Le corps de ma méthode
    alert(this.nom);
  }
};
```

Cette approche permet de réduire les collisions de noms lorsque plusieurs scripts sont utilisés dans une page. C'est une exploitation du *pattern singleton* en javascript. Les variables `nom` et `prenom`, ainsi que la fonction `maMethode()` ne sont accessibles que par l'intermédiaire de la variable `monobject`. J'ai utilisé cette approche pour déclarer le code nécessaire à la capture du temps entre chaque caractère lors de la saisie d'un mot de passe. Le javascript source 1 déclare le singleton `KeyboardBiometry` qui propose les fonctions nécessaires à la capture du temps, lors des appuis sur les touches du clavier. L'attribut `lastTimes` possède un tableau avec les ticks horloges enregistrés lors de l'appui sur les touches, indexés par la taille courante du champ saisi. La méthode `onKey()` capture les événements clavier pour mémoriser les ticks horloges de chaque touche. Quelques caractères ont un comportement particulier. Par exemple, le retour chariot ou la tabulation ne sont pas pris en compte. La touche `delete` efface tout le champ et réinitialise les mémorisations temporelles. En effet, l'utilisateur n'a pas le droit à l'erreur lors de la saisie. Il doit taper le mot de passe d'une traite, afin d'identifier le rythme de la saisie. S'il corrige un seul caractère, il doit tout recommencer. La méthode `getTime()` retourne un tableau avec le temps mis entre chaque touche. C'est cette information qui caractérise le rythme. Pour un mot de `n` caractères, il y a donc `n-1` valeurs de rythme. Pour utiliser cet objet, il faut invoquer la méthode `onKey()` sur les événements `onkeydown()` et `onkeyup()`. Avant la soumission du formulaire, il faut récupérer le rythme de la saisie et le placer dans un champ caché.

```
<form action="SampleKeyboardBiometry" method="post"
  onsubmit="rythme.value=KeyboardBiometry.
getTime(password)">
  <p>
  Utilisateur : <input type="text" name="user"/><br/>
  Mot de passe : <input type="password" name="password" value=""
    onkeydown="KeyboardBiometry.onKey(event,this)"
    onkeyup="KeyboardBiometry.onKey(event,this)"
    /><br/>
  <input type="hidden" name="rythme"/>
  <input type="submit" value="OK"/>
  </p>
</form>
```

Avant de pouvoir utiliser ce code, il faut effectuer un apprentissage biométrique de l'utilisateur. Pour cela, un autre objet javascript se charge de calculer les paramètres caractéristiques du rythme de la saisie (Voir le javascript source 2).

Les attributs `keyTimeMin` et `keyTimeMax` mémorisent les temps minimum et maximum entre deux touches. L'attribut `password` mémorise le mot de passe utilisé lors de l'analyse. Cela permet d'identifier les inconsistances lors des saisies d'apprentissage. La méthode `reset()` annule l'apprentissage précédent. La méthode `record()` enregistre le dernier rythme. La méthode `merge()` ajoute le dernier rythme au rythme actuel. Cela permet d'adapter la reconnaissance avec plusieurs saisies. Enfin, la méthode `check()` permet d'effectuer une vérification locale de la reconnaissance. Le rythme actuel est comparé avec le modèle précédemment calculé. Un coefficient de tolérance est utilisable. La valeur minimum du rythme est multipliée par ce coefficient, et la valeur maximale est multipliée par 1 plus le coefficient. Par exemple, avec une valeur de 0.3, cela permet de tolérer plus ou moins 30% de la valeur apprise.

La page <http://www.philippe.prados.name/Secureite/KeyboardBiometry/CheckKeyboardBiometry.html> est un exemple local d'apprentissage. Aucune information n'est envoyée vers le serveur. L'utilisateur doit entrer cinq fois le même mot de passe. Cela permet au programme d'apprendre le rythme de saisie de l'utilisateur. Puis, le nom et le mot de passe sont demandés. Une alerte indique si la reconnaissance est valide ou non. Cela vous permet de tester cette approche très facilement. Vous pouvez par exemple mémoriser un mot de passe, vérifier l'apprentissage et demander à une ou un ami de saisir le même mot. Si les rythmes sont différents, l'identification échoue.

Si vous rencontrez des difficultés, contactez-moi. Cela me permettra d'améliorer le programme. Il a été testé avec Internet Explorer version 6.0 et Firefox 0.8 de Mozilla sous Windows.

Pour utiliser ce code avec un serveur, il faut effectuer quelques modifications. Il faut réécrire l'algorithme de qualification

du rythme avec le langage utilisé (Java, PHP, Perl, C#, etc.). Il faut également mémoriser l'apprentissage du rythme. Le champ caché `pattern` possède deux listes d'entiers, séparées par le caractère pipe. Il suffit d'analyser ce paramètre sur le serveur et de le mémoriser avec les informations de l'utilisateur.

Vous n'êtes pas sans savoir qu'il ne faut jamais sauver un mot de passe directement dans la base de données. Sinon, la moindre compromission de celle-ci permet de révéler tous les mots de passe des utilisateurs. Ayant une mémoire réduite, les internautes utilisent généralement le même mot de passe pour plusieurs applications. Connaître le mot de passe d'une application permet d'avoir un passe pour utiliser d'autres applications.

Le DBA devient un suspect possible d'une attaque interne ou une cible privilégiée pour un cheval de Troie (un cheval de Troie est un programme anodin permettant à un pirate de prendre le contrôle d'un poste). Si le poste de l'administrateur est attaqué, le pirate peut consulter la base de données et obtenir tous les mots de passe.

D'après Pascal Courtin, chef de la Brigade d'Enquête sur les Fraudes aux Technologies de l'Information, "La plupart des agressions dont sont victimes les entreprises viennent de l'intérieur. Les différentes études estiment ce chiffre aux alentours de 80%".

Pour augmenter la sécurité, déresponsabiliser le DBA, supprimer un maillon faible, il est préférable de chiffrer les mots de passe avec un algorithme à sens unique de qualité (SHA1 pour les informations ASCII, MD5 pour les binaires), de telle sorte qu'il ne soit pas possible de déduire le mot de passe à partir des informations de la base de données. Pour utiliser cet algorithme, il faut enrichir le mot de passe d'une clef spécifique à l'application avant d'utiliser le MD5. Sinon, il est possible d'utiliser des dictionnaires de mots de passe pré-calculés pour casser les accès de certains utilisateurs.

```
MD5(se1+password)
```

`se1` est un nombre aléatoire, généré lors de l'installation de l'application et présent sur le serveur accueillant l'application. Le pirate doit alors connaître deux secrets pour obtenir de quoi utiliser un dictionnaire de mots de passe : le secret de l'application et un accès à la base de données. L'administrateur de la base ne doit pas avoir accès au secret de l'application.

Pour vérifier un mot de passe, il suffit alors de calculer la valeur MD5 à partir des informations de l'utilisateur et de comparer le résultat avec le hash mémorisé. Cela fonctionne car il n'est pas nécessaire de décrypter le mot de passe pour effectuer la vérification.

Mais, pour sauver les caractéristiques de rythme de l'utilisateur, il est indispensable de pouvoir les décrypter. Le serveur doit pouvoir les avoir en clair, afin d'appliquer l'algorithme d'identification. Le rythme est une information aussi confidentielle que le mot de passe. Comment le protéger ? Comme cette étape arrive après la qualification du mot de passe, il est possible d'exploiter celui-ci en clair. Il est possible de générer une suite pseudo aléatoire de bits, à partir du mot de passe et d'un `se1` différent de celui ayant permis le codage du mot de passe. Cette suite de bits permet d'appliquer un ou exclusif avec les informations de rythme.

```
RythmeCrypté=RC4(se1rythme,password) xor Rythme
```

Le résultat de ce calcul est sauvé dans la base de données. Sans le mot de passe, il n'est pas possible de retrouver le rythme. Avec le mot de passe, il suffit d'appliquer pratiquement la même formule pour retrouver le rythme.

```
Rythme=RC4(se1rythme,password) xor RythmeCrypté
```

Ainsi, nous avons sécurisé l'information, sans compromettre la sécurité. Le rythme ne peut être vérifié qu'avec la connaissance du mot de passe. Malgré l'obligation d'avoir des informations confidentielles en clair, nous avons trouvé une approche sécurisée, sans compromis.

Avec nos petits moyens, nous avons réussi à proposer une authentification forte, économique. Il n'est pas nécessaire d'acquérir des périphériques d'acquisition, le clavier fait l'affaire. L'usage est très simple car il suffit d'entrer le mot de passe comme d'habitude. Le seul écart

par rapport à l'approche habituelle est une phase d'apprentissage lors du changement du mot de passe. Celle-ci est simplifiée à l'extrême.

Pour intégrer cette technologie plus discrètement à votre application, vous pouvez, dans un premier temps, intégrer cela sans en faire un élément d'authentification. Vous ajoutez des paramètres aux informations des utilisateurs pour connaître l'étape de l'apprentissage, et les paramètres de rythme.

Au début, vous récupérez les informations de rythme pour effectuer l'apprentissage au fur et à mesure. Après cinq entrées de l'utilisateur, vous passez à la deuxième phase : la vérification du rythme de l'authentification. Au début, il est possible de ne faire que des statistiques sur la reconnaissance, pour tester la technologie. Après la phase d'apprentissage, le rythme est testé et le résultat est agrégé dans des statistiques, sans pour autant refuser l'authentification.

Ensuite, il faut rejeter une authentification si le rythme est trop différent de l'apprentissage. Le nombre d'échecs avant le blocage du compte peut être augmenté, pour tenir compte de la nouvelle technologie.

Cette démarche incrémentale peut être reprise dès le changement d'un mot de passe. Ainsi, l'utilisateur n'a pas à suivre la procédure d'apprentissage. Elle s'effectue à son insu, lors des cinq premières authentifications.

Ensuite, la détection du rythme se met en place toute seule. Cette approche permet de ne strictement rien changer dans l'ergonomie de l'application. Par contre, le rythme n'est pas testé lors des cinq premières authentifications, créant ainsi une petite faiblesse.

Vous avez tous les outils côté client, il vous reste à coder la partie serveur avec votre langage préféré, en suivant les recommandations indiquées.

J'ai testé cela avec du code java et j'ai pu vérifier qu'un apprentissage mémorisé sous IE fonctionne avec Firefox.

*A vous de jouer !*

## Sources 2

```

/**
 * KeyboardBiometry.js
 *
 * @version 1.0
 * @author Philippe Prados
 **/
var KeyboardBiometry={
  lastTimes:[],
  /**
   * Record the time between two char.
   **/
  onKey:function(event,field)
  {
    if (event.keyCode==13) return;
    if (event.keyCode==9) return;
    if (event.keyCode==8)
    {
      field.value="";
      event.returnValue=false;
    }
    var value=field.value;
    if (value.length<=1)
    {
      this.lastTimes=new Array();
      this.lastTimes[0]=new Date().getTime();
    }
    else
    {
      this.lastTimes[value.length-1]=new Date().getTime();
    }
  },
  /**
   * Return an array with the delta time for each char.
   **/
  getTime:function(field)
  {
    var value=field.value;
    var deltaTime=[]
    for (var i=1;i<value.length;++i)
    {
      deltaTime[i-1]=this.lastTimes[i]-this.lastTimes[i-1];
    }
    return deltaTime;
  }
};
<<<Fin Encadré>>
Source 1
<<<Encadré>>
/**
 * LearnKeyboardBiometry.js
 *
 * @version 1.0
 * @author Philippe Prados
 **/
var LearnKeyboardBiometry={
  keyTimeMin:[],
  keyTimeMax:[],
  password:"",
  /**
   * Reset the current model.
   **/
  reset:function()
  {
    this.keyTimeMin=[];
    this.keyTimeMax=[];
    this.password="";
  },
  /**
   * Record the current model.
   */
  record:function(field)
  {
    this.keyTimeMin=KeyboardBiometry.getTime(field);
    this.keyTimeMax=KeyboardBiometry.getTime(field);
    this.password=field.value;
  },
  /**
   * Update the model with the current.
   */
  merge:function(field)
  {
    if (this.keyTimeMin.length==0)
    {
      this.record(field);
      return;
    }
    if (field.value!=this.password)
    {
      alert(«Ce n'est pas le même mot de passe»);
      return false;
    }
    var keyTime=KeyboardBiometry.getTime(field);
    for (var i=0;i<keyTime.length;++i)
    {
      if (keyTime[i]<this.keyTimeMin[i])
        this.keyTimeMin[i]=keyTime[i];
      if (keyTime[i]>this.keyTimeMax[i])
        this.keyTimeMax[i]=keyTime[i];
    }
    return true;
  },
  /**
   * Check the input with the model.
   */
  check:function(field,percent)
  {
    if (this.password.toString()!=field.value.toString())
    {
      alert(«Ce n'est pas le même mot de passe»);
      return true;
    }
    var currentKeyTime=KeyboardBiometry.getTime(field);
    var l=(this.keyTimeMin.length<currentKeyTime.length)
      ? this.keyTimeMin.length : currentKeyTime.length;
    var baduser=false;
    for (var i=0;i<l;++i)
    {
      if ((currentKeyTime[i]>this.keyTimeMax[i]*(1+percent)) ||
        (currentKeyTime[i]<this.keyTimeMin[i]*(1-percent)))
      {
        baduser=true;
        break;
      }
    }
    return baduser;
  }
};

```