Problems and Solutions to a Large Scale Passwords Reset

SHARIL TUMIN
University of Bergen
IT-Department
P. O. Box 7800, 5020 Bergen
NORWAY
edpst@it.uib.no

SYLVIA ENCHEVA
Stord/Haugesund University College
Faculty of Technology, Business and Maritime Sciences
Bjørnsonsg. 45, 5528 Haugesund
NORWAY
sbe@hsh.no

Abstract: IT systems are not 100 percent protected from internal and external threats. User credentials can be compromised and that will effect enterprise wide Users-System trust. A Web-based system especially designed for passwords reset is needed that should be able to tackle massive password changing activities in term of security, scalability, efficiency and usability. It is of utmost importance that the users’ confidence to the system must be maintained.


1 Introduction

Sooner or later your IT system will be hacked and users’ credential will be compromised by the attackers. It is not necessary for you to panic even if the responsibility of IT security of the enterprise belongs to you. The best way is to be ready for such eventualities. When such an unlucky happening happens to your system, you need to keep a cool head to evaluate risks and provide counter measure. Anyhow, you do not know to which extents the damage has been done. Have you made safe copies of important system programs and libraries?

The only way to bring back users trust is to reset all users’ credential as quickly and efficiently as possible. However, keep in mind that the process of resetting thousands of users’ passwords is by no mean an easy task. In this paper we discuss a way to handle the situation properly, in another word not to be reactive but be rational. We also propose a Web-based system that the users can use to change their passwords which needs more then just a simple username and password authentication.

Each user will receive a system message with a personal URL via email. The URL is a link to a special Web application. The URL will have an encoded parameter containing personal data that only the user knows. There will be a limit on how many times a user can use the provided URL to prevent misuse of the facility. There is also a time period in which the URL is valid.

The user needs to provide valid data that correspond to the parameters of the URL in order to change her password. By using Web-based application, users can change their passwords by their own means without the need of helps from call center or users support.

To implement such a system, we assumed that the enterprise has an IdM (Identity Management System) in which all personal data to all users and all users’ credentials are stored.

2 Background

An enterprise IT system, bases much of its security mechanisms and resources protection, on knowing who a user of the system is at any one time. Even though much research has been done concerning authentication methods using smart-cards, biometric, radio devices and other smart techniques [4] and complicated protocols [5], the basic username and password pair authentication method still persists. This basic authentication method is simple, practical, cheap and universally available.

A user-accounts database containing all user credentials, normally username and password pairs, together with authorization and permissions database are necessary parts of a multi-user system, see Figure 1. Authentication data is very important to system integrity. System security can be improved by proactively checking for weak passwords [6]. However, some of these credentials can be compromised by cyber criminals. Even if the passwords are usually stored in their hash values, the lost of list of users with associated hashed passwords is enough to suspect a serious security issue. Attackers can use dictionary attack technique to obtain users’ clear text passwords.

Let $A$ and $B$ be a collection of authentication and authorization data respectively. $A$ contains
Figure 1: System Authentication and Authorization

all user credentials and $B$ contains all definitions for resources access control. Their definitions and relationships are shown below.

$$A = \{(u_i, p_i), i = [1, n]\}$$

where $(u_i, p_i)$ is the username and password pair for user $i$.

$$u_i \in U \text{ where } U \text{ contains all enterprise users.}$$

$$B = \{(R_j, g_j, P_j, C_j), j = [1, m]\}$$

where $(R_j, g_j, P_j, C_j)$ is a resource, group, permissions and constrains tuple for enterprise resource $j$.

$$R_j \in R \text{ where } R \text{ contains all resources,}$$

$$g_j \in G \text{ where } G \text{ contains all defined groups,}$$

$$P_j \in P \text{ where } P \text{ contains all defined permissions,}$$

$$C_j \in C \text{ where } C \text{ contains all defined constrains,}$$

then,

$$u_i \in g_j, \bigcup_{j=1}^{m} g_j \subseteq U$$

Accepting Murphy’s law that says “Anything that can go wrong will go wrong” [1] as a practical wisdom, it is prudent to assume that any system will be hacked one day. To rest in the believe that “my” IT system is totally secure is naive to say the least.

If $A$ is compromised then $B$ will lose much of its security functions. Once user credentials are compromised, the authorization of the system falls apart especially if the stolen credentials belong to power users. Be prepared before it happens.

It is impossible to make an enterprise IT system completely secure. By definition a system is defined as a group of interacting, interrelated, or interdependent elements forming a complex whole [2]. An enterprise IT system is being threatened both locally and globally for fun, financial gains, commercial and political reasons, and recently cyber terrorism.

Taking a discovery of a successful attack as an incident, we can then divide the sequence of handling of the event into three separate periods:

1. pre-incident,
2. immediate reaction,
3. post-incident.

The best immediate reaction will depend very much on what has already been done in the pre-incident period. All preventive tasks like firewalling, security backup and security patching belong to pre-incident. Safe copies of important system programs and libraries, for example (assuming Unix/Linux system) passwd, ls, find, ps and libc are of special importance in mitigating the ill effect of such security incident.

Post-event tasks depend on what type of damages the attack had successfully inflicted on the system. The worse scenario is when some parts or the whole system need to be reinstalled.

The best immediate reaction will be to find all eventual fraudulent programs installed by the attacker(s) and to delete or replace them. The next thing to be done is to change passwords to all locally defined users especially root user (assuming Unix/Linux system).

In this paper, our concerns are not with these important issues mentioned in passing above. Our modest contribution is to model, design and implement a system for password reset for all users as an immediate reaction to an event when users’ credentials are suspected to be compromised by the attackers. When such things do happen, normal authentication methods using a username and password pair is no more trustworthy. To build back the trust after such an incident, uses must be asked to change their passwords, without using their current passwords as credentials.

The proposed solution should be able to handle a short term notice for massive password changing activities. The main considerations in decreasing order of importance will be:
1. security,
2. scalability,
3. efficiency
4. usability.

The system availability should not be unduly interrupted. It is of utmost importance that the users’ confidence to the system must be maintained.

3 System

When a successful attack on an enterprise is discovered and depending on the security risk evaluations of the current threat, system administrators can choose to:

1. disable all enterprise users’ passwords,
2. disable user’s password on some high risk systems,
3. not to disable users’ passwords.

If alternative 1) is put into action then users can not be warned using email. The only possible way to communicate to users is by using Web. However, with their enterprise passwords disabled users might not be able to logon into their office or lab workstations and PCs.

Since in practice we can not disable users’ enterprise passwords, alternative 2) provides us with a nice compromise. We will then be able to warn and communicate with users with enterprise email addresses. Users will be given a limited time period of three days to change their passwords. The accounts that do not conform to this will be disabled.

The proposed system depends on the following for it security mechanism:

1. encoded URL parameter that contain personal data,
2. personal data,
3. valid time period,
4. current password,
5. valid client IP range.

For practical reasons the fifth constrain can be dropped as it is not practical for an organization that has mobile users as f. ex. in an educational organization.

We assume that the current user password was stolen, so in fact the authentication will need an added complexity based on the encoded parameter and the user’s personal data working together in combination during authentication. The personal data are data that only the person knows, for example: social security number, date of birth, employee number or PIN to key-card for accessing buildings. For the sake of simplicity in the discussion, we say that the system will use the username and PIN combination for the encoded parameter. An extra piece of control data can be added without affecting the general operating principle of the proposed system.

Listing 1 Python SHA1 example.

```python
>>> import sha
>>> ctn = 'UXPIN190809'
>>> id = 'sbe'
>>> PIN = '9236'
>>> s = sha.sha('%s:%s:%s' % (ctn, id, PIN))
>>> s.hexdigest()
'c6d4cf665faa838312704000a4d2e075fd24a485'

>>> u_id = 'sbe'
>>> u_PIN = '9236'
>>> u_s = sha.sha('%s:%s:%s' % (ctn, u_id, u_PIN))
>>> u_s.hexdigest()
'c6d4cf665faa838312704000a4d2e075fd24a485'

>>> u_PIN = '9326'
>>> u_s = sha.sha('%s:%s:%s' % (ctn, u_id, u_PIN))
>>> u_s.hexdigest()
'd622b48e69e0aee3ef21c8dea41471398c0d5c5'
```

The system will encode the relevant control variables using SHA1 hash function, [8]. A hash function can take an arbitrary data block called the message and returns a fix-sized bit string called the message digest. Any change in the message either accidentally or intentionally will also change the message digest.

Listing 1 shows how SHA1 Python library works. The control variable ctn is only known to the system and acts as an incident identifier. The control variables id and PIN will be compared with the user input variables. The SHA1 hash calculated by the s.hexdigest() will be used as the URL parameter to be given to a particular user indicated by the id variable. Now, the user input variables related to the parameter variables are u_id and u_PIN. The SHA1 hash calculated by u_s.hexdigest() will be compared with the URL parameter and the user is authenticated if the two values are the same. If the user inputs variables are not the same, even in a single byte, the SHA1 hash produced by these two different instances will be different.

The URL produced by the system’s application will be sent to the user by email. For this particular example the URL will then be:

https://siam.uib.no/chgpwd?cp=c6d4cf665faa838312704000a4d2e075fd24a485

To be correctly authenticated at the entry point of the system, this particular user needs to supply username equal to ‘sbe’ and PIN equal to ‘9236’. Upon successful authentication, the system will then present the user with a Web form where the user can change her password. The secret control variable ctn will be
linked to some internal constrains, related to this control variable. In this case variable name ‘cp’ will use the control value ‘UXPIN190809’, which indicates that the application will check user’s PIN as well as date constrain. It necessities that ‘cp’ and value of the control variable $ctn$ are directly related. In addition, the URL is only valid within some prescribed constrains, for example client location and time period constrains.

Figure 2 shows a simple conceptual process flow diagram of the proposed system.

The $IdM$ will only change the user password if all the controls variables check out with the data reside in the $IdM$ database.

The proposed system shown in Figure 3 will be implemented as a multi-tiers Web-based application, built using free and open source software. The users using their Web browsers will interact with an Apache [7] Web server using HTTPS (Hypertext Transfer Protocol Secure) communication protocol. The programmable environment and the middlewares are written in Python [10]. The back-end database is implemented using PostgreSQL [9].

The Python programming environment is used as integrating middleware between the Web server front-end and the database back-end. Some of the important Python packages used to implement the system are 1) mod_python - live-programmable module to Apache, 2) xmlrpclib - XML (Extensible Markup Language) RPC (Remote Procedure Call), 3) tlsite - SSL v3 (Secure Sockets Layer) and TLS v1 (Transport Layer Security) libraries, 4) pyPgSQL - API (application programming interface) to PostgreSQL and 5) standard Python libraries for examples - base64 and binascii.

The $Prompter$ and the $PasswordUpdater$ are the two main applications supported by the system. The purpose of the $Prompter$ is 1) to gather all personal data of enterprise users needed by the system from the $IdM$ and for each user 2) calculates SHA1 hash, and 3) email the URL to the user. The main job of the $PasswordUpdater$ is to 1) check constrains, 2) check the entry point authentication from the parameter and the user’s inputs, 3) administer bad attempts, 4) get user’s current password and new password, 5) check for password validity, and 6) submit new user password to $IdM$ together with all the control variables.
4 Security analysis

The secret control string \texttt{cdn} will protect the application from accepting parameters not originally created by the \textit{Prompter} application. Anyone can produce a SHA1 hash. In order for attackers to fool the system to accept bogus parameter they should know the secret control string and the format in which the control variables are arranged by the \textit{Prompter}. Still, if all these are lost to the attackers, the last defense is that the IdM database was not compromised.

The maximum number of unsuccessful authentication attempts is three, where the URL will be marked as invalid. This will reduce the risk that another person who is not the owner of the URL will be able to guess the control variables encoded in the parameter. A legitimate user blocked out by the system in this way needs to meet up at the call center personally.

The system will also set a maximum number of unsuccessful password changes on the IdM to three. This will reduce the risk of guessing current users’ passwords. A blocked user needs to meet up at the call center personally.

The logon to the application using the URL with a result of successful password change can only occur once. Suppose the attackers also know the values of all control variables of a user and manage to change the user password. When the real user then tries to change her password at a later time, she will be blocked by the application. The legitimate user will then know someone has changed her password illegally and will report this fact to the system administrators.

All accounts that have not received new passwords within a limited time period of three days will be disabled automatically by the IdM. Users with disable accounts need to take contact with a call center personally.

Time period and even location constrains can be easily implemented should the need arises to increase the security control.

The \textit{PasswordUpdater} is implemented separated from the IdM, see Figure 3. Access to the IdM is controlled by a firewall and communication between them is done using XML-RPC over TLS. Moreover the RPC port on the IdM accepts only connection from PasswordUpdater with specially formatted requests.

After the process of changing passwords is completed, the database to the system that implements PasswordUpdater is cleared before the whole server is taken down. However, the server must be ever ready to be initialized and ready again for an eventual security incident at a moment notice.

5 Conclusions

It will take around 28 hours for all password changes of 20000 users to be effected, if each password change takes 5 seconds to complete starting from a user changed her password on the Web application until the change is synchronized to all others enterprise IT systems. To cater for some parallelism in servicing the users, several \textit{PasswordUpdater} servers can be installed.

Password changing process can be stressful to certain users. They prefer to use their old password that they have had for the past ten years or more. Providing information on simple risks and benefits pertinent to the process are usually sufficient to persuade the hardiest of protesters.

However well the application was designed and implemented it will never satisfy 100 percent of all the users. For example, an external user that never received a key-card will not have a PIN to use. The solution installed in the system should be agile enough to accommodate special cases. If a user does not have the preferred piece of control data and alternative control data should be used. The URL parameter name can be used to signal the Web application on what control variable to is be used, for example ‘cp’ for PIN, ‘cs’ for social security number, ‘cb’ for date of birth, and ‘ce’ for employee number.

So, how the proposed system will measure up against the 1) security, 2) scalability, 3) efficiency and 4) usability criteria mention in Section 2?

By using SHA1 hash of control variables as parameter to personalized URL the fist line security implementation is satisfied. The second line security implementation is done at the IdM server by checking for inconsistencies in the data provided by the users via the \textit{PasswordUpdater}. Several \textit{PasswordUpdater} Web applications can be installed to increase the capacity. It is clear that using Web applications for massive password change activities is more efficient then using the call center and other temporary user service points. A Web application can be designed to be useful and highly usable with very little extra effort.

In the fight against cyber crimes especially in relation to attacks that disrupt user-system trust, the best strategy is to be proactive, keep panic to a minimum, and be prepared. No systems on the internet are 100 percent protected from internal and external threats, sooner or later it is your turn.

References:

[1] Murphy’s law
http://en.wikipedia.org/wiki/Murphy’s_law
2009 (last accessed)
http://www.thefreedictionary.com/
2009 (last accessed)

2009 (last accessed)


http://www.apache.org/
2009 (last accessed)


http://www.postgresql.org/
2009 (last accessed)

[10] Python Programming Language.
http://www.python.org/
2009 (last accessed)