TWiki > DICE Web > Project168PWstrength (28 Oct 2016, TobyBlake)

Password strength checks

Project history

This project was previously stalled and has now been awoken. All previous documentation can be found here:

http://www.dice.inf.ed.ac.uk/units/infrastructure/Projects/168-PWstrength/

Update 2015-11-24 - Summary of current policy/policies

First, a word on cracklib...

Cracklib

Many of the places where password restrictions are applied use the system cracklib library. This checks against a configured (and configurable) dictionary. In the descriptions below, "Cracklib check" refers to calling the "FascistCheck" cracklib library function. This does the following (from its README):

```
4) it's MIND-NUMBINGLY THOROUGH!

(is this beginning to read like a B-movie flyer, or what?)

CrackLib makes literally hundreds of tests to determine whether you've
chosen a bad password.

* It tries to generate words from your username and gecos entry to tries
to match them against what you've chosen.

* It checks for simplistic patterns.

* It then tries to reverse-engineer your password into a dictionary
word, and searches for it in your dictionary.

- after all that, it's PROBABLY a safe(-ish) password.  8-)
```

To give an indication of the checks made, these returned errors are from a crude grep of the cracklib code:

```
("it is based on your username");
("it is based upon your password entry");
("it is derived from your password entry");
("it's derived from your password entry");
("it is derivable from your password entry");
("it's derivable from your password entry");
("you are not registered in the password file");
("it is WAY too short");
("it is too short");
("it does not contain enough DIFFERENT characters");
("it is all whitespace");
("it is too simplistic/systematic");
("it looks like a National Insurance number.");
("it is based on a dictionary word");
("it is based on a (reversed) dictionary word");
("error loading dictionary");
```

Current policies

We currently have four different places where we enforce a password policy. The specific details of each policy are described below.

Password portal

- (for new students only)
- Cracklib check
- Minimum of 8 characters

- Must contain 4 different character classes
  - digit, lower case, upper case, non alpha-numeric symbol (custom definition)

Prometheus remctl tools

- (currently only called via password portal)
- Cracklib check

DICE PAM stack (pam_cracklib on SL6, pam_pwquality on SL7)

- (applied when a user changes password on a DICE machine, using 'passwd' command)
- Cracklib check
- Minimum of 8 characters
- Minimum of 3 different character classes (out of 4)
  - digits, upper and lower letters and other characters
- reject_username - checks for username in password (in order or reversed)
- Performs additional checks - palindrome, case change only, similarity, simplicity, rotation of previous password

KDC kadmind

- (applies to **all** password changes)
- kadmin policy 'infuser' applies to all person accounts (not machine or other). Policies can be applied on a per-principal basis.
- Minimum of 8 characters
- Minimum of 3 different character classes (out of 5)
  - lower case, upper case, numbers, punctuation, and whitespace/unprintable characters
- Must not be the same as previous two passwords

In summary

We have four different places in which password policies are applied. They are all different from each other, both in terms of the restrictions they apply and the rules they follow (e.g. what constitutes a given character class). Depending how a user sets their password, different policies will be applied, e.g.

- new student user: password portal->prometheus->kadmind
- existing user using DICE 'passwd': pam_cracklib->kadmind
- self-managed/DICE user using 'kpasswd': kadmind

The only policy we can be sure is being applied is that enforced by kadmind on the KDC.

krb5-strength

The krb5-strength module is implemented as a password quality plugin module for kadmind. It would allow us to implement the following policies at the server side (in the KDC's

kadmind):

- Cracklib check
- Optional additional dictionary configurations:
  - CDB database - "When checking against a CDB database, the password, the password with the first character removed, the last character removed, the first and last characters removed, the first two characters removed, and the last two characters removed will all be checked against the dictionary."
  - SQLite database - "When checking a SQLite database, the password will be rejected if it is within edit distance one of any word in the dictionary, meaning that the database word can be formed from the password by deleting, adding, or changing a single character."
- minimum_different (at least this number of unique characters)
- minimum_length
- require_ascii_printable
- require_classes - this is quite a flexible restriction in that you can require different character classes for a password depending on its length, e.g. something like "8-19:upper,lower 8-15:digit 8-11:symbol" - the rules are cumulative so an 8 character password requires all 4 classes, a 20 character password requires none. It might be nice to patch the module so we can enforce number of separate character classes for a given password length, not just which ones.
- require_non_letter - require a character that's not just upper and lower case characters and space

Recommendations/questions

- We should decide on what we want our password policy to be, given what's available to us (CEG?)
  - Do we want to insist on [number of] character classes?
  - If so, do we want added flexibility so that a password of a certain length can bypass character class restrictions?
  - Should we augment the default cracklib dictionary?
- We should, as far as possible, implement password policies/restrictions at the server side, not client side.
  - This means rely on KDC policies/kadmind/krb5-strength and make sure they are strong/flexible enough
  - If we do the above, do we want **any** client-side checks performed? Only cracklib?
- We should implement additional measures to protect us against brute-force attacks, e.g. use fail2ban on KDCs, weblogin, etc.

Update 2015-12-02

We discussed this project at the development meeting.

We agreed the following:

- We should only have server side checks and will remove all client side checks - this means removing cracklib checks from everywhere but the KDC, including pam stack (carefully), to avoid confusion.
- We want a more flexible policy, e.g. passwords more than 20 characters only need one character class - however no actual policy was defined at the meeting.
- We should use the krb5_strength module on the KDC to enforce our chosen policy.
- We should patch krb5_strength so that named character class constraints could be replaced by >N character class constraints.
- We should use fail2ban on KDCs AND weblogin to guard against brute-force attacks, but this will require some thought to avoid potentially blocking our own services.
    - eg we don't want multiple password failures on ssh.inf to result in the KDC denying ssh.inf (via fail2ban) for some period of time, and thus affecting all users of ssh.inf.
- We should allow COs to try different passwords against a test KDC running krb5_strength to help determine a reasonable policy.
- We should augment the default cracklib dictionary with others, including different languages. These need to be sourced.

The following points discussed but without definitive decision:

- We may want one blanket user password change to ensure passwords all comply with new policy.
- We may wish to do bulk brute force offline checks against our KDC database and advise users of failures. We will investigate this.
- We should consider nagios monitoring kdc logs against a "number of failed attempts" threshold and/or regularly scan logs for deviation from the norm - a useful first step for the latter would be to write a script to gather statistics for authentication requests.
- We should consider patching krb5_strength for any other requirements we deem are not being met - e.g. repetition of a string >N characters.

Update 2016-01-22

The krb5-strength module has been patched to allow >N character class constraints, specifically to allow us to set something like the following:

```
require_classes = 8-12:3 13-19:2
```

... this specifies that passwords of 8-12 characters must contain 3 character classes, passwords of 13-19 characters must have 2 classes and passwords of 20 characters or more have no character class restrictions.

The patch for this should be tidied up (mainly to document the change) and passed

upstream.

We have also patched krb5-strength so that it returns a dictionary error on failing cracklib tests, rather than the generic one. Whether this is correct is debatable, but the generic error "Unspecified password quality failure while changing password for... " is not particularly helpful for the user.

We have enlisted a number of volunteers for testing passwords against our test realm.

Our test KDC is using a cracklib dictionary which consists of the dictionaries shipped in RedHat's cracklib-2.9.6 RPM, with the addition of the free openwall word list (see http://www.openwall.com/wordlists/). This contains words from other languages, in addition to English.

The next stages in the project are:

- Continue testing and finalise desired krb5-strength password policy
- Test fail2ban for use with KDC
- Have a look at performing some offline brute-force attacks against our KDC db (the value of this is questionable, however, as we intend using fail2ban to protect against any such brute-forcing.

Update 2016-01-28

The fail2ban work mentioned above has been shuffled sideways into a more appropriate home - the KDC software and configuration upgrade project.

Update 2016-05-17

A summary of the current situation:

We have a test KDC running the krb5-strength module. This has been patched locally to allow the specification of the number of required character classes for passwords of different lengths. This patch has been sent upstream.

We have been using a wordlist which comprises the cracklib wordlist, as shipped by Red Hat, with the addition of the free openwall wordlist (which contains words from 20+ languages).

This password policy has been tested by 3-4 COs, specifically the following settings:

```
miniumum length = 8
require_classes = 8-12:3 13-19:2
```

Although not part of this project, our KDCs are now protected from brute force attacks by fail2ban.

Work remaining:

Changes to lcfg-kerberos templates/resources to add support for krb5-strength

It would be nice to do some more testing to determine how much information is returned to the user when their password is rejected. We have only been able to test the direct use of 'kinit' and, short of setting up an entire infrastructure around the TEST.INF realm, we are very limited in what we can do. Our only sensible approach is to test in place. We have some concerns over the level of information returned to the user.

Implementation plan. The next stage in this project is to come up with a plan for implementing our password policy. In short, and in no particular order (although order will be important), it involves:

- implement krb5-strength on master KDC
- remove/disable cracklib from password portal
- remove/disable cracklib from prometheus tools
- ~~remove cracklib from pam stack~~
- change existing kadmin policy to remove char constraints
- change policy documentation
- publicise

We want this project to be completed in time for the new intake of students.

Update 2016-07-08

A post from the author of the krb5-strength module to the Heimal mailing list raised some concerns about the code quality of CrackLib.

Following this, it seemed prudent to revise our use of cracklib within krb5-strength. We are now, as per the suggestion in the above post, testing krb5-strength using the SQLite database approach ... from krb5-strength's README:

```
When checking a SQLite database, the password will be rejected if it
is within edit distance one of any word in the dictionary, meaning
that the database word can be formed from the password by deleting,
adding, or changing a single character.
```

We are using a wordlist that combines the one distributed in RedHat's cracklib-2.9.6 RPM with the free openwall wordlist (see http://www.openwall.com/wordlists/). In total, this contains nearly 5.8 million words.

It is worth stating what we consider the threats to any individual user's passwords to be and how to protect against these:

1. Brute-force dictionary attack against our systems
    a. from outside the University
    b. from within the University
2. Password re-use (i.e. DICE password used elsewhere) and password compromised on another system
3. Password compromised externally (poor password management, phishing, etc.)

Use of the krb5-strength module with a good wordlist protects against *1.* Our use of fail2ban

protects us further against *1a*. (this currently only applies to attacks from outside the University networks - we should perhaps consider extending this to all IPs from inside the University, to protect against *1b*.)

2. and *3.* can only really be guarded against through user education and awareness. Education is important to enable users to, e.g. spot phishing attempts. If a password has been compromised, it is important that we spot this as soon as possible - e.g. through our monthly mailing to each user detailing their successfull authentications. More details on this and other, related, considerations here.

In addition to the above, all sysadmins receive a weekly email with details of authentications (successful and unsuccessful) against their principal. We also gather and process statistics on all attempted authentications against our KDCs on a weekly basis.

This update has meandered off-topic somewhat, in an attempt to provide a summary of where this particular project fits into our security infrastructure.

We should not become too preoccupied with the specifics of the dictionary checking provided by krb5-strength - we can always revise the details at a later date - the module's biggest advantage is perhaps the flexibility it gives us in setting our password policy.

We should also decide what, if anything, we want to do about existing passwords - e.g. do we want to force a one-off change, do we want to attempt to crack our existing user passwords to identify weak ones?

Work remaining on this project is to implement the plan identified in the previous update.

Update 2016-08-12

A quick update on the issue of using cracklib in our pam stack on DICE machine... on SL7 we use the pwquality module with the *local_users_only* option. From pam_pwquality(8):

```
local_users_only
    The  module  will  not test the password quality for users that are
    not present in the /etc/passwd file. The module still asks for  the
    password  so  the  following  modules  in  the  stack  can  use the
    use_authtok option.  This option is off by default.
```

This means we rely on the KDC to do password checks.

Update 2016-08-19

The use of the krb5-strength module raises issues about the feedback of password change errors to users, in particular the messages returned make assumptions that the kadmin built-in password policies are being used exclusively.

krb5-strength is a plugin module and can only return a limited range of error codes, as defined by the KADM5 API:

https://k5wiki.kerberos.org/wiki/Projects/Password_quality_pluggable_interface

These codes are then interpreted by the kadm5 library and error strings are returned to the client. Here's an example (for `KADM5_PASS_Q_CLASS`):

```
New password does not have enough character classes.
The character classes are:
   - lower-case letters,
   - upper-case letters,
   - digits,
   - punctuation, and
   - all other characters (e.g., control characters).
Please choose a password with at least 1 character classes.
```

Note the last line - it is incorrect - it refers to the kadmin password policy which is applied to the principal, not the (entirely separate) policy enforced by the krb5-strength module. In this case, the required number of character classes is determined by the length of the new password and, for the 9-character password used in this example, is actually 3.

The character classes in the message above are also incorrect - kadmin defines five character classes, whereas krb5-strength defines four (upper, lower, digit, symbol).

If there was no kadmin password policy applied to this principal, we wouldn't see this misleading information. However we do want to continue to use policies as they offer some functionality which is unavailable in krb5-strength (primarily keeping a configurable history of previous keys for a principal to stop users changing the password to what it was last time (or last time plus one)).

We can run a patched krb5 on the server side (KDC master) to return more accurate information, e.g.

```
New password does not have enough character classes.
The character classes are:
   - lower-case letters,
   - upper-case letters,
   - digits,
   - symbols (all non-alphanumeric, including space).
```

Programmatically determining the number of classes required at password change time does not appear to be trivial.

It's not ideal, but we could hard-code our policy into the message - this would provide users with accurate information, but with the obvious drawbacks such hard-coding entails.

Similarly, the error message for a password which is found in the krb5-strength dictionary check is:

```
New password was found in a dictionary of possible passwords and
therefore may be easily guessed. Please choose another password.
See the kpasswd man page for help in choosing a good password.
```

Note that `kpasswd(1)` contains no such help.

We could change this text so that it points towards (a revised):

http://computing.help.inf.ed.ac.uk/password-policy

This boils down to a trade off between the goal of providing useful and informative feedback to our users and the unpleasantness of hard-coding changable policy information in software.

Update 2016-08-20

Proposed new text for http://computing.help.inf.ed.ac.uk/password-policy ...

DICE passwords **must** currently:

- be at least 8 characters long, and
- be composed of a minimum of:
  - 3 character classes (for a password length of 8-12 characters)
  - 2 character classes (for a password length of 13-19 characters)

Passwords which are 20 characters or more have no character class constraints.

The four recognised character classes are:

- lowercase letters
- uppercase letters
- digits
- symbols (all non-alphanumeric characters, including space)

In addition, any new password must differ from the previous two passwords.

Any new password will also be checked against a large dictionary of known password/passphrases. Any password that matches an entry in this dictionary will be rejected.

These standards are enforced at password creation/update time by the School's Kerberos Key Distribution Centre (KDC).

Choosing a good password:

A good password is something which is easy for you to remember, but difficult for someone else to guess. You should avoid commonly known words or phrases.

You should not write your password down or tell it to anyone.

You **must not** use your DICE password anywhere else.

See also Information Services' guidance on information security including advice on passwords.

Update 2016-10-28

The password policy page has now been updated:

http://computing.help.inf.ed.ac.uk/password-policy

A systems blog article was written:

http://blog.inf.ed.ac.uk/systems/2016/09/27/changes-to-dice-password-policy/

This project is now complete, pending final report and sign-off.

-- TobyBlake - 28 Oct 2016

Topic revision: r16 - 28 Oct 2016 - 13:03:39 - TobyBlake