Taking the “sword” out of “pas**sword**”

Using new and existing RACF password controls to reduce your risk of breaches

NY/Tampa RACF Users Group
April 16, 2015

Bruce R. Wells
brwells@us.ibm.com

Trademarks

- See url http://www.ibm.com/legal/copytrade.shtml for a list of trademarks.

Agenda

- Passwords are a problem
- The nature of offline password attacks
- What are the elements of an effective password policy?
- What makes a password weak or strong?
- How can I implement my own password rules quickly?

Background

- Despite increasing use of biometrics, digital certificates, tokens, etc, passwords are likely to be around for a long time still.
- Human nature makes us sloppy and lazy. The easier a password is to remember and to type, the more likely we are to use it.
- The easier a password is to remember, the more likely it is that somebody else is going to guess it.
- The harder a password is to remember, the more likely we are to write it down.

When you leave users to their own devices

- They are going to choose bad passwords

When you leave users to their own devices

- They are going to choose bad passwords
- And use them on facebook, LinkedIn, Yahoo mail, etc
- And never change them
- And give them away

https://www.youtube.com/watch?feature=player_embedded&v=opRMrEfAIiIFeb 23, 2015

Level set: what is an offline attack?

- Someone steals a password database, by
  - Exploiting a software vulnerability
  - Exploiting a misconfiguration
  - Stealing/guessing credentials
  - Being, or enlisting, a disgruntled insider
- Then attacks the hashed passwords at home using their own equipment. Access to host machine is not required once the password database is stolen.
- This does NOT go thru the normal login-process. It does not even have to run on the same operating system. There is no revocation mechanism involved.
- Speed is limited to available hardware.

How are password crackers trying to obtain passwords?

- Offline attacks employ the following strategies
  - Dictionary
  - Modified dictionary replacing O with 0, I with 1, s with $, etc
  - Try lists of actual passwords stolen elsewhere
  - Try popular patterns (Upper, 5 lowers, 2 digits and “!”)
  - Target individuals based on info obtained online (family/pet names, sports teams, employer, birthday, etc)
  - Brute force, when all else fails

What resources are available to password crackers?

- Cheap powerful GPUs used in parallel. Advanced hardware is now available on a 'kid-with-summer-job' budget.
- ASICs (application-specific integrated circuit) and FPGAs (field-programmable gate arrays)
- Botnets
- Off the shelf software such as oclHashcat and John the Ripper
  - JtR now has a RACF/DES feature!
- Freely available dictionaries, password lists, and “rainbow tables” (pre-computed dictionary hashes used for fast brute-force attacking)

What can you do as a security admin?

- Educate your users to
  - not share or otherwise divulge passwords
  - not use your RACF password on other sites
  - not use discernible patterns even if they slip by the enforced rules
  - Take a look at that “last access” message when you logon

What can you do as a security admin? ...

- "Help" your users choose better passwords
  - Assign password phrases where possible
  - Implement history and a minimum change interval to prevent password reuse/cycling

    ```
    SETROPTS PASSWORD(HISTORY(8) MINCHANGE(5))
    ```

    - And make sure your reset procedure/software has no loopholes
  - When phrases are not possible, enable mixed case and special characters in passwords
  - Force a mixture of character types
  - Implement the Rexx-based sample new password exit

What can you do as a security admin to thwart attackers?

- Carefully protect the RACF database and its copies and backups!
  - (Almost) nobody needs READ access
  - An insider with READ can perform an offline attack at their leisure
- Limit invalid password attempts to a very small number

  ```
  SETROPTS PASSWORD(REVOKE(3))
  ```

  - At the risk of (possibly self-inflicted) DoS attacks
  - Is your green-screen logon panel available on the internet?
    http://mainframesproject.tumblr.com

What can you do as a security admin to thwart attackers? ...

- Audit/pen-test your RACF database
  - Password cracking tools are for white-hats also
  - Use results to modify your password rules
- Embrace your paranoia: pretend an attacker has your database
  - Enforce a password change interval shorter than the amount of time an attacker can reasonably crack a password

```
SETROPTS PASSWORD(INTERVAL(90))
```

  - Do you know when you've been breached?
- Enable strong password encryption

Strong encryption: KDFAES!

- New algorithm available with OA43999

1) Start with:
   - DES hash for passwords
     - This step maintains upward compatibility in some cases
   - Clear-text password phrase

2) Append random text (salt)
   - This step defeats “rainbow tables”

3) Iteratively hash (SHA256) this text a (large) number of times to derive a 256-bit AES key
   - This step is intentionally slowing down an offline brute-force attack

4) Encrypt the RACF user ID with the AES key
   - This step appeases your auditors because you are using “an approved algorithm”

Enabling KDFAES: Death to ICHDEX01! Long live SETROPTS!

- Enabling is easy (warning...read the planning considerations first!!!)

```
SETROPTS PASSWORD(ALGORITHM(KDFAES))
```

- Seeing if it is enabled is easy!

```
SETROPTS LIST

PASSWORD PROCESSING OPTIONS:
  THE ACTIVE PASSWORD ENCRYPTION ALGORITHM IS KDFAES
  PASSWORD CHANGE INTERVAL IS  60 DAYS.
  PASSWORD MINIMUM CHANGE INTERVAL IS   3 DAYS.
  MIXED CASE PASSWORD SUPPORT IS IN EFFECT
  SPECIAL CHARACTERS ARE ALLOWED.
  … …
```

- And with OA45608, the new RACF_ENCRYPTION_ALGORITHM Health Check reports the active algorithm

When KDFAES is activated

- Existing DES passwords will continue to be evaluated properly
  - There will be no fallback to masking during evaluation
- When next changed, KDFAES will be used
- If you are paranoid that somebody is **about** to steal your database, you can convert passwords and history to KDFAES without requiring any password changes!

```
ALTUSER STU PWCONVERT
```

  - Or, to do it in bulk

```
SEARCH CLASS(USER) CLIST('ALTUSER ' ' PWCONVERT') NOLIST
EX 'prefix.EXEC.RACF.CLIST'
```

  - Note: phrases and phrase history cannot be converted

Administrative password expiration: new with OA43999

- If you are paranoid that somebody may have **just** stolen your database (perhaps a recent DES-based backup), you can force users to change their password/phrase at next logon

```
ALTUSER STU EXPIRED
```

  - Or, to do it in bulk

```
SEARCH CLASS(USER) CLIST('ALTUSER ' ' EXPIRED') NOLIST
EX 'prefix.EXEC.RACF.CLIST'
```

  - And if you were already using phrases, this will get users to change them faster after enabling KDFAES
  - Can also be useful to force changes when your password rules change

Speaking of password rules...

- Strong encryption is not sufficient to protect weak passwords

- Passwords are generally stronger when
  - they are as long as possible/reasonable
  - they allow a large set of characters to be used
    - and those characters are **actually** used
  - they appear to be essentially random

- A number of mechanisms exist to enforce strong “passwords”

Password phrases

- Take another look. More applications support phrases now than when they were introduced in z/OS V1R8, including
  - TSO/E
  - NFS
  - IBM LDAP
  - IBM z/OS UNIX and LE
  - OpenSSH
  - IBM Network Authentication Services (Kerberos)
  - The FTP and TN3270 servers provided with z/OS Communications Server
  - Tivoli NetView for z/OS, Version 5.4
  - IBM Session Manager for z/OS V3.1
  - CICS Transaction Server for z/OS, Version 4 Release 2
  - DB2 V10 for z/OS
  - WebSphere AppServer V6.1 and later
  - IMS V12
  - OMEGAMON e3270 UI
  - IBM zSecure suite
  - Connect:Direct (up to 64 characters)

Password phrases ...

- The idea is that they are easy to remember, **and** they are longer and harder to crack than passwords.
  https://xato.net/passwords/a-line-from-sf-to-ny/
- Examples:
  - You should read Love in the Time of Cholera
  - Touchdown! Gronk spike ball!
  - I really hate my boss
  - You're gonna need a bigger boat
  - and she's buy-uy-ing a stairway to heaven
  - cattle lakeshore stripmall bridge office
- Should also intentionally misspell/reorder some words, add symbolics here and there, etc, to stay ahead of the password crackers
  http://arstechnica.com/security/2013/08/thereisnofatebutwhatwemake-turbo-charged-cracking-comes-to-long-passwords/

Password phrases ...

- With OA43999, users can be assigned a phrase without also requiring a password:

```
ADDUSER JOE NOPASSWORD PHRASE('This is a temp phrase')
ALTUSER BOB NOPASSWORD PHRASE('This is a temp phrase')
```

- Some users cannot always use phrases. For example, console logon does not support it.
- Such users will need a password, and also having a phrase doesn't really add any protection
  - They'll never use it
  - Attackers will go after the password
  - The password can also be used to logon

If you can't use phrases, at least make your passwords stronger

- Upper case letters, digits, and nationals (@#$)
  - 39**8 = 5,352,009,260,481 possible 8-character passwords
- Take another look at mixed case passwords, more applications support them than when they were introduced in z/OS V1R7

  ```
  SETROPTS PASSWORD(MIXEDCASE)
  ```

  - 65**8 = 318,644,812,890,625 possible 8-character passwords
- Consider the new special character support in OA43999

  ```
  SETROPTS PASSWORD(MIXEDCASE SPECIALCHARS)
  ```

  - Adds the following characters: ! % & * - _ = + | : < > . ?
  - 79**8 = 1,517,108,809,906,561 possible 8-character passwords
  - Can force a password to contain one each of upper, lower, digit, and symbol (new characters plus nationals)

    ```
    SETROPTS PASSWORD(RULE1(LENGTH(8) MIXEDALL(1:8)))
    ```

Password quality

If you could try one million  passwords per second:

- NOMIXEDCASE/NOSPECIALCHARS = 62 days max to crack a password
- MIXEDCASE/NOSPECIALCHARS = 3688 days = >10 years
- MIXEDCASE/SPECIALCHARS = 17559 days = >48 years

- How many passwords **can** a cracker guess in a second?
  - It depends

http://arstechnica.com/security/2012/12/25-gpu-cluster-cracks-every-standard-windows-password-in-6-hours/

  - But if your password = “PASSWORD”, it doesn't matter

Password quality ...

- Users can still do the wrong thing, even with syntax rules in place

```
SETROPTS PASSWORD(MIXEDCASE SPECIALCHARS
                  RULE1(LENGTH(8) MIXEDALL(1:8)))
```

- Bruce01!, Bruce02!, Bruce03!, etc
  https://www.youtube.com/watch?v=qR-qRUbeKAo
  - 23:56 - “I've never seen password complexity programs/rules that prevent common password patterns.”

- Native RACF rules can not prevent this
- You need a new password exit (ICHPWX01)
- Thankfully, it does not need to be difficult!
- And consider: if someone steals your RACF database, they also have your password rules to focus their attack. But if your rules are in an exit...

Implement your own password rules

- In Rexx!
- Using the sample provided on the RACF web site
  - http://www-03.ibm.com/systems/z/os/zos/features/racf/downloads/rexxpwexit.html
- Consists of:
  - The traditional assembler part: ICHPWX01
    - Assemble, link edit, IPL – once – to get it active
    - But first:
  - The Rexx part containing the rule logic: IRRPWREX
    - Put it in your System Rexx concatenation
    - Make a change, save the file, change is active! (but test it first!)
  - You can install IRRPWREX without ICHPWX01 being active, but the converse is not true!

Features of the sample

- ICHPWX01
  - System Programmer Override, in case of emergency
    - When the only gal who can fix your problem can't logon because her password is expired, you have a problem
    - Grant READ access to IRR.ICHPWX01.OVERRIDE in FACILITY class to allow logon w/ password change when there is a Sysrexx error (the rules obviously aren't being enforced, but SETROPTS is)
- IRRPWREX
  - Debug mode
  - Pre-provided checks that can be enabled by simply changing a configuration variable
  - A single config variable enables the checks that satisfy the DISA STIG
  - Ability to query the active settings from the console
- Readme file

System Rexx configuration

- IRRPWREX must exist in your REXXLIB concatenation when ICHPWX01 calls it
  - System Rexx default library is SYS1.SAXREXEC
  - Can add others in your AXRxx member in SYS1.PARMLIB

```
REXXLIB ADD DSN(BRWELLS.SAXREXEC) VOL(D94001)
```

  - `AXRUSER(userid)` specifies the identity under which the exec will run

- AXR=xx keyword in IEASYSxx points to the AXRxx member in use
  - Default is AXR00 if AXR= not specified
- Or, 0,SYSP=(AA,BB,CC),AXR=xx during IPL
- See the AXRxx chapter in MVS Initialization and Tuning Reference

Checks provided in IRRPWREX

- Minimum length violation
- Password contains disallowed characters
- Password does not contain at least one character from a specified number of character types (numbers, letters, special)
- Password contains part of user's name
- Password is only trivially different from previous value
- Password does not contain enough character differences, by position, from previous value
- Password contains a word from the restricted word list
- Password contains too many unchanged characters, by position, from previous value
- Password does not contain enough new characters from previous value
- Password does not contain all unique characters
- Password contains "consecutive" characters
- Password contains the user ID, or some subset of the user ID
- Password contains too many repeating characters
- Password starts with a string from the restricted prefix list
- Password uses a restricted pattern

Debug mode

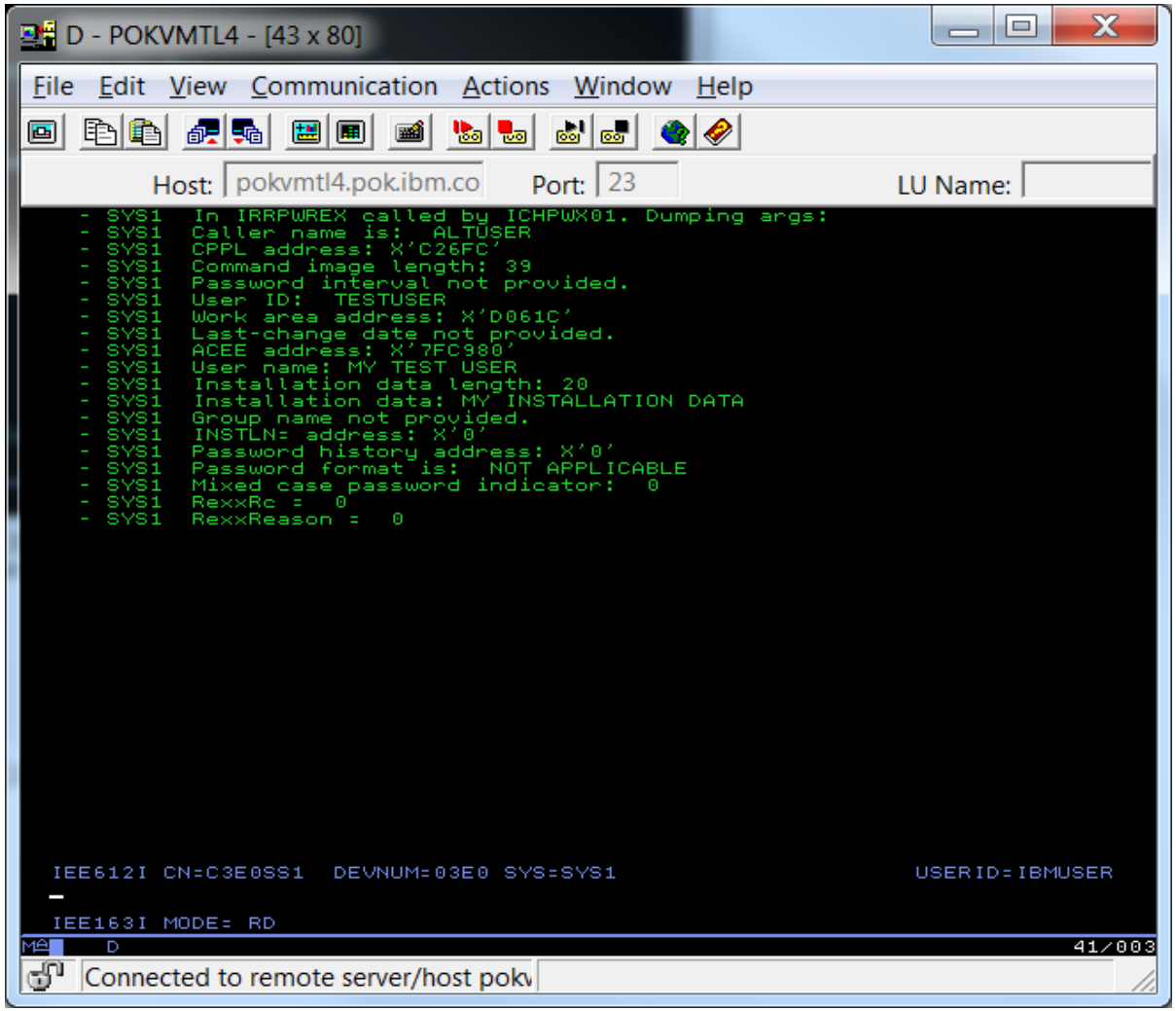
- Helpful when implementing for the first time, to confirm the exit is being called successfully

```
/*-------------------------------------------------------------------*/
/*   Debug mode. If 'on', the input arguments and final return       */
/*   and reason code are dumped to the console using WTO.            */
/*                                                                   */
/*   Note that System REXX provides additional functions from the    */
/*   AXREXX macro which could be useful for debugging. ICHPWX01      */
/*   would need to be modified to exploit these.                     */
/*                                                                   */
debug = 'on'


ALTUSER TESTUSER PASSWORD(ABC) NOEXPIRED
```

results in the following on the console (next slide)....

Debug mode output out of the box



Enabling checks: example

```
/*--------------------------------------------------------------------*/
/*   Minimum unique characters by position. This check prevents       */
/*   a new password which differs by only a few character positions   */
/*   from the old password. For example, changing the password from   */
/*     AFD4TRH                                                        */
/*   to                                                               */
/*     BFD3TRH                                                        */
/*                                                                    */
/*   An associated variable controls whether the passwords are first  */
/*   upper-cased before the check is performed. The check is          */
/*   performed for the length of the smaller string, so even if the   */
/*   new password is longer than the old, this rule could still fail  */
/*   the change if there aren't enough unique characters in the       */
/*   beginning part.                                                  */
/*                                                                    */
/*   This check is only performed for PASSWORD and RACINIT, since     */
/*   ALTUSER does not provide the old password.                       */
/*                                                                    */
/*   This check may be enabled by changing the setting to a non-zero  */
/*   value.                                                           */
/*                                                                    */
Pwd_min_unique = 0
Pwd_min_unique_upper = 'yes'
```

<===Change to 5, for example

Enabling checks: example – What if my policy is the DISA STIG?

```
/*-------------------------------------------------------------------*/
/*   STIG compliance.                                                */
/*                                                                   */
/*   This check automatically enables the other checks that enforce  */
/*   compliance with the United States Defense Information Systems   */
/*   Agency's (DISA) Security Technical Implementation Guide (STIG)  */
/*   V6R21 with regard to RACF password quality rules, to the extent */
/*   possible, taking some liberties on the content of the user ID   */
/*   and user name that are checked.                                 */
/*                                                                   */
/*   Not all the subsequent checks are relevant to the STIG, and     */
/*   they may also be enabled as desired. A STIG-relevant check      */
/*   will be identified with an asterisk to the left of the first    */
/*   line of its description.                                        */
/*                                                                   */
/*   Changing the value of STIG_Compliant to 'yes' will result in    */
/*   the relevant checks being enabled, regardless of any changes    */
/*   made to the explicit checks immediately below.                  */
/*                                                                   */

STIG_Compliant = 'yes'            /* Enforce DISA STIG compliance    */
```

Query setting from the console using the Sysrexx “modify” command

```
F axr,irrpwrex list



The following IRRPWREX password exit rules are in place:
  STIG compliance is explicitly specified
  The minimum password length is 8
  The number of required character types is 4
  The user's name cannot be contained in the password
    Only 3 consecutive characters of the user's name are allowed
    The minimum word length checked is 8
  The user ID cannot be contained in the password
    Only 3 consecutive characters of the user ID are allowed
  All characters in the new password must be unique
  No consecutive characters (e.g. AB or 12) are allowed
    This check is not case sensitive
```

There is also a “robot-friendly” output format suitable for consumption by programs

Case study: Implementing the topology check

```
/*--------------------------------------------------------------------*/
/*   Pattern check. This is sort of like the SETROPTS password        */
/*   rules in reverse, in that we specify patterns (or "topologies")  */
/*   that can *not* be used, as opposed to those that *can* be used.  */
/*   The default patterns specified below are the ones most           */
/*   frequently encountered by password crackers and penetration      */
/*   testers for 8-character passwords.  As such, these patterns      */
/*   are the first ones tried in a brute-force attack.  Of course,    */
/*   these defaults may not be in sync with other of your password    */
/*   policy settings.                                                 */
/*                                                                    */
/*   U = upper, L = lower, N = number, and S = special.               */
/*                                                                    */
/*   Strings defined in this list will be upper-cased prior to the    */
/*   check.                                                           */
/*                                                                    */
/*   To enable this check, add/remove/alter patterns as desired and   */
/*   set Pwd_pattern.0 to the resulting number of restricted          */
/*   patterns.                                                        */
/*                                                                    */
Pwd_pattern.0 = 0
Pwd_pattern.1 = 'ULLLLLLN'
Pwd_pattern.2 = 'ULLLLLLS'
Pwd_pattern.3 = 'ULLLLLNN'
Pwd_pattern.4 = 'ULLLNNNN'
Pwd_pattern.5 = 'ULLLLLNS'
Pwd_pattern.6 = 'ULLNNNNS'
Pwd_pattern.7 = 'ULLSNNNN'
```

```
/*************************************************************/
/* Build the pattern string by assigning the appropriate     */
/* "pattern character" for each position of the password.     */
/*************************************************************/
Do I = 1 to Length(newPwd)
  chk_letter = Substr(newPwd,I,1)
  Select
    When verify(chk_letter,Upper_letters,Match) /= 0 Then
      pattern_string = pattern_string || 'U'
    When verify(chk_letter,Lower_letters,Match) /= 0 Then
      pattern_string = pattern_string || 'L'
    When verify(chk_letter,numbers,Match)       /= 0 Then
      pattern_string = pattern_string || 'N'
    Otherwise
      pattern_string = pattern_string || 'S'
  End /* Select */
End   /* For each password character */

/*************************************************************/
/* Now see if the constructed pattern string matches any of  */
/* the restricted patterns.                                  */
/*************************************************************/
Do I = 1 to Pwd_pattern.0
  UpperPattern = Pwd_pattern.I
  Upper UpperPattern
  If UpperPattern = pattern_string Then
    Do
      RexxReason = 15
      signal pwdexit
    End
End /* For each restricted pattern */
```

So what **IS** a good password? A “random” one.

- Make a sentence
  - Dad ate **6** donuts for breakfast today = d8Sdf%ft
- Enumerate objects in a visual space, such as a house, store, office, etc
  - bsK9ck+d (When I go home, I enter through the **b**asement, see some **s**hoes, my dog greets me! When I go upstairs, I hang my coat in the **c**loset, go into the **k**itchen, and bonus, **d**inner's ready!)
  - Practice it
- Channel your inner misspeller
- Use a password manager
- Use 2-factor when available
- Google the topic; there's no shortage of opinions!
- What's your favorite strategy?

What about phrase rules?

- A Rexx-based ICHPWX11/IRRPHREX is provided in SYS1.SAMPLIB
- ICHPWX11 is shipped link-edited into LINKLIB, so you can skip the assemble and link-edit steps
  - Copy IRRPHREX from SAMPLIB into the REXXLIB concatenation
  - Copy the ICHPWX11 module from LINKLIB to LPA
  - IPL
- It is very similar to ICHPWX01/IRRPWREX, and the password readme can be helpful for understanding it
- It does not have the ability to query its rules from the console

- Why is one in samplib and one on the web site? History.

Checks provided in IRRPHREX

- Minimum length violation
- Maximum length violation
- Phrase contains disallowed characters
- Phrase contains leading blanks
- Phrase contains trailing blanks
- Phrase contains part of user's name
- Phrase is only trivially different from previous value
- Phrase does not contain enough character differences from previous value
- Phrase does not contain enough unique word differences from previous value
- Phrase contains a word from the restricted word list

Coming in z/OS V2R2

- You never need an ICHDEX01 exit unless you are implementing your own password algorithm
- RACF_ENCRYPTION_ALGORITHM Health Check raises an exception if KDFAES is not active
- ADDUSER will not assign a default password

```
ADDUSER STU TSO(...) OMVS(...) NAME('DISCO STU')
ICH01024I User STU    is defined as PROTECTED.
```

  - ALTUSER and PASSWORD cannot be used to **re**set a password to the user's default group
    - It can, of course, be explicitly assigned...if your rules allow it!
- RACLINK DEFINE(*node*.*user*/*pwd*) supports password phrases
- The RACF ISPF panels support the new OA43999 functions

Lessons learned – What can you do?

- Gnash teeth
- Pull out hair
- Curl up in a fetal position
- Drink heavily
- Unplug your server

http://www.theonion.com/articles/after-checking-your-bank-account-remember-to-log-o,32260/

And then ...

- Revisit your password policy
- Double check your RACF database protection (and anywhere else you may be storing passwords/keys)
  - Review your backup policy
- Check out the new OA43999 functions
- Take another look at password phrases and mixed case passwords
- Take a look at ICHPWX01/IRRPWREX
- Re-educate your users (and yourself)
- Stay on top of trends and developments