RACFSNOW User Guide

Version 1.8

Contents

Version

Version 1.8 contains a number of changes. First and most significant is the use of file oclHashcat.pot which is used in exactly the same way as oclHashcat uses it. In other words it is a cache of every password found and is cumulative. Racfsnow also utilises is to maximise efficiency of effort. **Note:** racfsnow honours the password_mask when writing to the screen or any racfsnow.??? file, but ignores it when writing to the olcHashcat.pot file, for obvious reasons. Other changes include amending max dictionary_limit down to 2,400,000 (from 2,500,000), required to make space for the new longer line length of 0200 records following **APAR OA43999 – RACF password security enhancements**. Removed references to sorttable.js in html and no longer try to utilise this code.

Version 1.7 contains bug fixes and major revamp of the output files. The primary output file remains racfsnow.htm which is tailored according to the selected options located in the racfsnow.ini file. However, in addition there are now two pairs of files. One pair of files is in the CRACF / WEAKWORD text format, one showing recovered details, and the other unrecovered details. Then, likewise the second pair show recovered and unrecovered details but it the John the Ripper / oclHashcat text format. Significant improvements have also been made to the demo mode.

Version 1.6 now comes in both 32 and 64 bit versions. The other significant changes are regarding file types, both input and output. In particular, it will accept an input file which is in the format used by John the Ripper and oclHashcat. It also generates output files in John the Ripper and oclHashcat format so they can be used easily in conjunction with racfsnow.

Version 1.5 contains minor changes to the dictionary limit, both how it's determined and the actual limit. The effective limit will be determined by the ini file parameter, up to the maximum supported by the compilation, which for now is 2.5 million (previously 2 million).

Version 1.4 contains fix to correctly handle lowercase passwords. This hadn't been tested previously as no test data had been available. Following a bug report, and the provision of some test data this has now been resolved.

Version 1.3 contains further enhancements to the demo mode, including the generation of a small dictionary containing the base words used in the demo. It also has changes to make it easier to use the racfsnow program to run against a file of hashes. If no input file is specified, then it will look for and use racfsnow.txt if it exists.

Version 1.2 contains enhancements around displaying the version number of the executing software, a minor improvement around the estimated time remaining, and a bug fix relating to the report_mode parameter not working correctly (it had been acting as if 'all' was selected even when 'found' was actually selected). Also added some date related common passwords to passwords.txt plus the base of every password featuring in the demo users (most were already included).

Version 1.1 contains enhancements around the scanning of the binary input file, together with better error handling when encountering unexpected input data. It also contains a new demo_mode making it easier to get started.

Introduction

The whole point of this program is to try and bring home to folks the importance of choosing good passwords. Put simply, because RACF stores passwords securely, a good password will take of the order of 2 years to crack using brute force, but a poor password will take of the order of 2 minutes to crack using a dictionary attack. The choice is yours!

To get started download and run racfsnow.exe If the racfsnow.ini file doesn't already exist it will be created and racfsnow will run in demo mode, using data supplied purely for demo purposes. To run normally, simply set demo_mode to false and supply the name of your own input file (and preferably also an IRRDBU00 unload file if possible).

Description

RACFSNOW, is a highly optimised PC program for performing a dictionary attack against a RACF database, with the option of using a database unload (IRRDBU00) to validate the User IDs to attack. It uses an ini file to control various parameters to enable focusing the attack on certain user IDs and or passwords.

Either the sample racfsnow.ini file can be used and edited, or if the program can't find an existing racfsnow.ini it will create a skeletal one which can then be edited in order to get up and running.

There is only one optional command line parameter, all the rest are supplied via the racfsnow.ini file. The optional command line parameter is a single User ID. This invokes what is referred to as targeted mode and ignores all other scoping parameters, simply focusing on trying to recover a single User ID.

Note: because the initial pass of the binary database file is doing a low level search for any passwords, it may still find several hits for a single User ID.

Parameters

demo_file

This parameter is required and should be either true or false. The default is true and when true the program uses its own demo data for both password hashes and IRRDBU00 input to enable an appreciation of how the program operates. Note: within the demo data which contains 999 sample users, some of these users have elevated privilege attributes, and some are revoked. The idea is to appreciate the ability of not only testing a large number of passwords at a simple level, but also being able to focus resources on particular users as appropriate in order to understand the overall risk of poor passwords.

Valid values for this are:

demo_mode=true

demo_mode=false

input_file

This parameter is required, and should be the filename of a binary copy of the RACF database. It can either be just the filename of a binary copy of the RACF database if located in the same directory, or a full or partial pathname if preferred.

dictionary_file

The dictionary file is a list of passwords to be used to test against the password hashes. This is optional, but if used the file format expected is straight Windows text file with one password per line where every line has a carriage return, line feed at the end. Passwords greater than 8 characters are ignored.

password_mask

The password mask must be 8 characters long. If it's not the default mask of ******** will be used. The purpose of this mask is to determine whether or not to reveal the recovered password. If the position is an asterisk then nothing is revealed, but if the position is a zero then the recovered character is revealed.

e.g. password_mask=0000****

The above example will reveal the first 4 characters of the password, leaving the second 4 characters displayed as ****

irrdbu00_unload

This is an optional parameter specifying the filename or pathname of an ASCII file containing an IRRDBU00 unload. This is used to validate User IDs and also gather additional information from the 0200 records. Use of this is recommended as it reduces the number of password hashes to be processed as there are always password hashes found in the binary database file relating to deleted User IDs.

targeted_list

This is similar to the command line, targeted mode, except instead of just focusing on a single User ID this is the ability to use a file containing a list of targeted User IDs. As with the command line targeted mode it ignores all other scoping rules, and the file format is one User ID per line with every line terminated with a carriage return, line feed.

display_mode

The display mode is used to determine what information to show on the command line when the program is running in relation to any passwords recovered. Valid values for this are:

display_mode=quiet

display_mode=verbose

If the display mode is verbose it will display the User ID and password on the command line, although this is done in accordance with the password mask.

report_mode

The report mode is used to determine what information is written to the racfsnow.htm output file. Valid values for this are:

report_mode=found

report_mode=all

If the report mode is found, then only information for User IDs with recovered passwords are written to the output file.

user_mask

The user mask is used to limit the scope of the User IDs being tested. This must be 8 characters long.

e.g. user_mask=********

The above example will include every User ID within the scope of the search.

e.g. user_mask=RACF****

The above example will limit the scope of the search to only those User IDs which have RACF as the first 4 characters.

dfltgrp_mask

The dfltgrp mask is similar to the user mask, but this time limiting the scope based on the dfltgrp of the User IDs. Again, this mask must be 8 characters long. The dfltgrp mask and the user mask work in conjunction with each other, thus only those User IDs which satisfy both these masks will be included in the scope of the search.

run_limit

The run limit is a cap on the maximum number of User IDs to be included within the scope.

dictionary_limit

The dictionary limit is a cap on the maximum number of dictionary words to be loaded. The maximum possible value for this is 2.5 million. This is intended for making it easy to reduce the size of the dictionary and hence find out the difference a smaller dictionary will make to the run time.

password_length

This is the minimum password length to be tested for.

Note: this needs to be considered in conjunction with the numeric parameter. For example if a numerics parameter of 1 is specified and a password length of 7, then 6 character dictionary words will be tested when appended with a single numeric.

numerics

This parameter is to control the ability to append a dictionary word with either 1 or 2 numeric digits.

Valid values for this are:

numerics=0

numerics=1

numerics=2

Appendix

Understanding the way passwords are securely hashed using DES.

Let’s consider a User ID of AAAAAAAA with a password of AAAAAAAA.

Character	EBCDIC (hex)	binary	Description
A	C1	1100 0001	UserID as DES input data
A	C1	1100 0001	obfuscate (xor 55)
		1001 0100	shift left
	28	0010 1000	Password as DES key
	So in full	DES key	`2828282828282828`
		DES input	`C1C1C1C1C1C1C1C1`
		DES output	`062314297C496E0E`