Digital Rights Management Mechanisms in the SCO Product Line

Anonymous

ABSTRACT

This document attempts to explain the primary mechanisms with which SCO has attempted to protect their intellectual property as well as how they fell short of doing so.

In particular, the serial number and activation key (SNAK) combination, the license data and the registration key shall be covered.

7 February 2018

Digital Rights Management Mechanisms in the SCO Product Line

Anonymous

Overview

The product line of SCO Group, Inc. (SCO) consists of the OpenServer and UnixWare operating systems as well as associated add-on software, such as user licenses or development kits. After SCO had gone bankrupt, UnXis, Inc. bought their UNIX business and continued to offer it. UnXis, Inc. later renamed to Xinuos, Inc. The new OpenServer 10 operating system offered by Xinuos, Inc. is *not* covered by this document; OpenServer 10 eschews digital rights management entirely. Because of this, ‘‘SCO products’’ is still used, even though the products are now maintained and sold by a different entity.

SCO’s digital rights management consists of a number of different parts:

1. the serial number (henceforth **serno**) and activation key, (**actkey**), forming the ‘‘**SNAK**’’,
2. the license data indicating additional information about a product, and
3. the registration procedure.

The purchase of an SCO product provides the buyer with a SNAK and license data if required. For continued use, a product usually also needs to be *registered*. When the system prompts for registration, a registration lock (**reglock**) is displayed. The reglock is then supposed to be entered together with personally identifying information on a Xinuos, Inc. web portal.

Serial Number and Activation Key

The pair of serial number and activation key, internally also called ‘‘SNAK’’, consists of a nine-character serno and an eight-character actkey. The actkey encodes the product ID, product version and whether license data is required as well as a checksum over the serno and the data contained in the actkey. The first three characters of the actkey denote the product ID. The second triplet denotes the product version and whether license data is required. The final two characters are the checksum. While the serno is effectively an arbitrary string, the actkey is an all-lowercase string. For example, a valid SNAK would be the serno ‘‘SCO524572’’ and the actkey ‘‘mnridxjo’’. There are two layers of protection: obscurity and a checksum.

The **actkey** is encrypted. Presumably because of cryptography export restrictions in the United States of America, the encryption algorithm is weak. The ciphertext (i.e. the actkey) is processed in reverse, starting with the NUL. Each character is *rotated* in the alphabet (a-z) by the previous character’s offset in the alphabet. The first character to be decrypted, which is the last character in the string, will always be unchanged by this algorithm. See *incfrp*() (decryption) and *decfrp*() (encryption) functions for the implementation.

The second layer of protection is a simple checksum over the serno and the first six characters of the decrypted actkey. See the*mnsnc*() function for the implementation.†

After decrypting the actkey and validating the checksum, the decrypted actkey is parsed. Both of the data triplets in the actkey are encoded in base 26 with alphabet a-z; the most significant digit is written first. For the example above, the decrypted actkey is ‘‘ahvneoco’’. ‘‘ahv’’ decodes to 0xCB (203), which is the product ID. ‘‘neo’’ decodes to 0x22CA (8906), which contains the flag if license data is required and the product version. The version is contained in the lower three nibbles: 0x22CA & 0xFFF equals 714. The last decimal digit (4) denotes the minor version, the other ones denote the major version (71). ‘‘co’’ is the checksum value.

† ‘‘incfrp’’ and ‘‘mnsnc’’ are the names used by SCO. It is unclear what they stand for. ‘‘decfrp’’ has been named that way because the opposite of increment is decrement, assuming ‘‘inc’’ in ‘‘incfrp’’ stands for ‘‘increment’’.

License Data

License data is required if bits 11–15 of the second triplet in the actkey equal 3. In the given example, this wasn't the case; the relevant bits equal 2 instead. Values other than 2 and 3 are invalid. License data is used to supply additional information about a license and enforce restrictions. If license data is required, it is printed alongside the SNAK on Certificates of License and Authenticity. For example, license data could look like this: "c4;k0;mjqs8r7".

The core of the algorithm is an MD5 hash over a secret value, the serno, the actkey and the rest of the license data. Then, this hash is translated to a custom base 32 alphabet. It is important to note that the license data is ordered by the flag characters, except for the m flag always being at the end. See the file for the implementation.

This algorithm is actually susceptible to length-extension attacks because the secret is merely prepended and MD5 has no mitigations for length-extension attacks. However, because license data is fairly strict in validating the input, it is not actually practical.

The following fields are known:

c number of CPUs;

d expiration of license in days; this is used for evaluation licenses;

k if no argument or a non-zero argument is given, registration is required, but more than one user can access the system; if the argument is zero, no registration is required, but only one user can access the system and possibly the network stack is gimped;

u number of users that may access the system at the same time;

m MD5 hash over the license data secret, the serno, the actkey and a canonicalized version of the string until this flag.

More flags exist. At least b, g, q have been observed but their format and effects are unknown.

Registration Key

Some products must be registered, else they expire after a set amount of time. This involves a per-installation identifier, the host ID (also referred to as node ID). The host ID and serial number are then combined to generate the registration key. Because the host ID is different for each installation of a product, storing registration keys is meaningless. Most likely, SCO realized the deficiencies of the simple SNAK scheme and added another layer of protection. Customers are meant to use an online portal to register their products, which also checks for double registrations of the same product, allowing the identification of serial numbers that have been shared. *\/etc/br and -k serno* or *scoadmin* can be used to generate the registration lock. Some of the data there is superfluous for registration code generation; it is surmised that they are collected for statistical purposes. This also implies that SCO has customers, which may be a stretch in the first place, especially considering the USD 2,500 price tag.

The core of the algorithm is an MD5 hash over a secret value, the host ID and the serno. The secret value differs from the one used for the license data. After that, the first four bytes of the hash are swapped and then encoded in base16 with a custom alphabet and a two-character checksum. Because this scheme is symmetric, no actual interaction with SCO is required. Furthermore, no measures were taken to obscure the secret – it is stored in the */etc/brand* binary with no obfuscation whatsoever. The secret value used to sign a registration key is the same as the one used to create the m flag for the registration lock. There is something truly, profoundly wrong with this; I believe I need not spell it out. See the *reg.c* file for the implementation.

A registration lock may look like this: "d180120;e380106;i203/71.4;oSCO310807;uorxrrwjwxz;mg7fuxu". Like the license data, it is a flag string. The following flags are known:

i the product ID and version; the product ID is separated by a slash, the product major and minor versions are separated by a dot;

o the serno;

u the host ID, encoded in some variation of base16 with a custom alphabet (kbwtacorhzgsejqx) with an appended checksum;

m MD5 hash over the registration secret, and a canonicalized version of the string until this flag; the canonicalization algorithm is the same as for license data.

Other flags exist, namely d and e, but their purpose is unknown.

Conclusions

SCO has done everything wrong that could possibly be done wrong, while also making matters much more complicated for themselves than necessary. There are a total of four checksums: one in the SNAK, a different one in the flag string for the license data, a different one in the flag string for the registration lock and a different one for the registration key. Furthermore, there are two secrets: the one for the flag string in the license data and the one used for both sides of the registration process. And then there are three different encoding schemes: the encryption of the activation key, the encoding of the activation key (base 26), the base 32 encoding in the m flag for license data and registration lock and the base 16 encoding for the registration key. It would not have been necessary to keep this many separate encodings and algorithms around.

Due to poor operational security, a mostly complete code dump of SCO UnixWare leaked on the Internet. They realized that keeping the */etc/brand* utility in the main tree would be dangerous, so it was checked in only as a binary file. However, the binary was neither optimized nor stripped, making reverse engineering effectively trivial.

Because all secrets in this DRM mechanism are known to both SCO and */etc/brand*, reverse engineering is all that is required to break every layer of protection; there is no cryptographic layer of protection, such as asymmetric signatures over the registration key. Elliptic curve signatures in a base64 encoding would likely have been tolerable for users. Alternatively, a truncated RSA signature could have been used – a full signature would be too long for users to type into the terminal. The short Schnorr signatures would have been another option, used by Microsoft in the Windows XP era.

None of this has a real-world impact and the estimated amount of lost sales tends towards zero. The only reasons to buy an SCO product are either legacy applications or the support contract that comes with it. Legacy applications generally do not generate new sales. Breaking the DRM mechanisms does not cause a support contract to come into existence out of thin air.