crx_unpack Documentation

Release 0.1.4

Mike Mabey

Jun 19, 2017

Contents

CHAPTER 1

crx_unpack Package

The purpose of this module is to mimic how Google Chrome unpacks CRX files as closely as possible. Involved in this is the need to remove the CRX headers (see the structure details of CRXs on the *Home* page), separate the underlying ZIP file, extract the contents of the ZIP file, among other things.

For end users, the only function you should need to call is *unpack*, which will handle each of the steps mentioned above.

crx_unpack.**unpack**(*crx_file*, *ext_dir=None*, ***, *overwrite_if_exists=False*, *img_tallies=None*, *test_contents=True*, *passwd=None*, *skip_img_formats=None*, *unpack_in_subprocess=False*, *convert_in_subprocess=True*, *do_convert=False*, *zip_dir=None*)

Unpack the CRX and extract it in the directory at ext_dir.

Return the absolute, normalized path to the extraction directory (useful if it wasn't given as a parameter).

As part of the unpacking process, this function will create a duplicate of the CRX but with the headers removed. This is technically a temporary file and will not persist past a reboot of the machine. However, because this ZIP file may be of interest to users, it is not deleted after the unpacking process is complete. To discover the path to this file, you'll need to either (1) set the *zip_dir* parameter yourself, or (2) set the logging level to *DEBUG*.

Parameters

- **crx_file** (*str*) – Path to the CRX file.
- **ext_dir** (*str*) – Directory where to extract the contents.
- **zip_dir** (*str*) – Directory where to store the ZIP file after removing the Chrome headers. Defaults to *ext_dir/../*.
- **overwrite_if_exists** (*bool*) – When extracting to a directory that already exists, unpack will normally fail. Setting this to True will delete the contents of the destination directory before unzipping.
- **img_tallies** (*dict*) – A dictionary for storing the number of each type of image file converted during the unpacking process.
- **test_contents** (*bool*) – When unpacking the CRX, use the zipfile module's test feature to test the validity of the embedded zip file before extraction.

- **passwd** (*str*) – Optional password to use when extracting the CRX. If the CRX was obtained from Google's Chrome Web Store, you should *not* need this. If you provide a password here, it will be passed on to the *extract_zip* function.
- **skip_img_formats** (*list or tuple*) – The image formats to skip when attempting to convert them to PNG. This will typically include the strings ICO, PNG, and WEBP.
- **unpack_in_subprocess** (*bool*) – Flag indicating if the job of unpacking the CRX should be done in a subprocess rather than calling the function directly. Usually this shouldn't need to be set as it will only hinder performance.
- **convert_in_subprocess** (*bool*) – Flag indicating if the job of converting the images in the CRX should be done in a subprocess rather than calling the function directly. Usually this SHOULD be set, since converting images can sometimes cause a segmentation fault, which kills the whole process.
- **do_convert** (*bool*) – Flag indicating whether images should be converted during the unpacking process (intended to mimic Chrome's unpacking process more closely).

Returns Directory where the archive was extracted.

Return type str

crx_unpack.**extract_zip**(*zip_file*, *extract_dir*, *pwd=None*, *test_contents=True*, *reraise_errors=True*)

Simple wrapper around the Python zipfile.ZipFile class.

Typically, it is not necessary to call this function directly from anywhere other than the *unpack* function.

Parameters

- **zip_file** (*str*) – Path to the zip file to be extracted.
- **extract_dir** (*str*) – Directory where the contents will be extracted.
- **pwd** (*str*) – Password for the zip file.
- **test_contents** (*bool*) – Whether to use the library's testzip() function on the archive before extracting. Tests if the CRC and header of each file in the archive are valid.
- **reraise_errors** (*bool*) – Set to False when the unpack script is run with the *xo* (extract only) command, in which case the function will return a non-zero value when an error occurs. The default, False, indicates that any errors that come up should just be re-raised.

Return type None

exception crx_unpack.**BadCrxHeader**

Bases: Exception

Raised when a CRX's header length or values aren't valid.

CHAPTER 2

Encrypted Temp Directory

The *EncryptedTempDirectory* class is designed for when you need a directory whose contents are encrypted by eCryptfs, but you would also like to take advantage of the features of in-memory-only directories, such as increased access speed and automatic deletion (e.g. when you're *unpacking a CRX*).

To understand this class fully, please also read the documentation on the `TemporaryDirectory` class.

class `crx_unpack.encrypted_dir.`**`EncryptedTempDirectory`**`(*, upper_dir, **kwargs)`

Bases: `tempfile.TemporaryDirectory`

Create and return an encrypted temporary directory.

This behaves similarly to `TemporaryDirectory`, except for the following:

- It requires that an "upper directory" be specified, which will be the mount point used by eCryptfs to mount the encrypted directory to the filesystem.
- It creates two files in *~/.ecryptfs* required to mount the directory (both of which are deleted when this object is):
  - *ALIAS.sig* - Contains the signatures for the FEK and FNEK encryption keys.
  - *ALIAS.conf* - Contains *fstab*-style information for which directory eCryptfs should mount and where.

In the above notes, *ALIAS* (which is a term used in the eCryptfs documentation, see links below) will be the name of the created temp directory, accessible as the basename of `self.name`.

To use an *EncryptedTempDirectory* object, it's best to use it with a `with` clause, like so:

```
with EncryptedTempDirectory(upper_dir=upper) as lower:
    ...
```

Better yet, use an instance of `TemporaryDirectory` as the upper directory, like this:

```
with TemporaryDirectory() as upper, \
        EncryptedTempDirectory(upper_dir=upper) as lower:
    ...
```

Note: In the above example, both temporary directories are deleted as soon as the `__exit__()` method is called (triggered by the close of the `with` clause). So make sure that anything you need to do with these objects, you do before leaving the `with` clause.

Note: This class depends on eCryptfs, so it will need to be installed on the system to work properly. Similarly, this class depends on the following Unix tools/devices:

- `head`
- `ecryptfs-add-passphrase`
- `mount`
- `keyctl`
- `/dev/urandom`

On Debian/Ubuntu-based systems, you can install these with:

```
sudo apt-get install coreutils mount keyutils ecryptfs-utils
```

For more information, see the following resources:

- http://manpages.ubuntu.com/manpages/zesty/en/man1/mount.ecryptfs_private.1.html
- http://manpages.ubuntu.com/manpages/zesty/en/man1/ecryptfs-add-passphrase.1.html
- https://askubuntu.com/questions/574110/how-to-use-ecryptfs-with-a-random-directory/574425#574425

Parameters

- **upper_dir** (*str*) – Path where the encrypted directory will be mounted, and where the unencrypted version of the files will be accessible.
- **kwargs** – Additional parameters to pass to the constructor of the `TemporaryDirectory` class.

This module contains several utilities for working with Google Chrome extension files (CRXs), which have the `*.crx` file extension. The goal of this project is to mimic as closely as possible the functionality of Google Chrome when these extensions are unpacked and installed.

The first module is *crx_unpack*, which handles the headers and structure of the CRX itself (see below for more details on this).

The second module is *Encrypted Temp Directory*, which gives a way to use eCryptfs to encrypt a directory that only ever exists in memory by inheriting from the `TemporaryDirectory` class and hooking into some eCryptfs tools to handle the encryption of the file contents and file names (handled by different keys).

CHAPTER 3

Installation

Since `crx_unpack` is available on PyPI, you can install it using pip:

```
$ pip install crx_unpack
```

CHAPTER 4

Structure of CRX Package Format

The information in this section introduces the structure and contents of CRX files.

As explained at https://developer.chrome.com/extensions/crx

Package Header

The header contains the author's public key and the extension's signature. The signature is generated from the ZIP file using SHA-1 with the author's private key. The header requires a little-endian byte ordering with 4-byte alignment. The following table describes the fields of the `.crx` header in order:

Field	Type	Length	Value	Description
magic number	char[]	32 bits	Cr24	Chrome requires this constant at the beginning of every `.crx` package.
version	unsigned int	32 bits	2	The version of the `*.crx` file format used (currently 2).
public key length	unsigned int	32 bits	*pubkey.length*	The length of the RSA public key in *bytes*.
signature length	unsigned int	32 bits	*sig.length*	The length of the signature in *bytes*.
public key	byte[]	*pubkey.length*	*pubkey.contents*	The contents of the author's RSA public key formatted as an X509 SubjectPublicKeyInfo block.
signature	byte[]	*sig.length*	*sig.contents*	The signature of the ZIP content using the author's private key. The signature is created using the RSA algorithm with the SHA-1 hash function.

Extension Contents

The extension's ZIP file is appended to the `*.crx` package after the header. This should be the same ZIP file that the signature in the header was generated from.

Python Module Index

C

Index