Firefox decrypt passwords

This python script extracts the passwords stored in Firefox's password manager.

firefox-decrypt.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.

# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <http://www.gnu.org/licenses/>.

# Based on original work from: www.dumpzilla.org

import argparse
import csv
import ctypes as ct
import json
import logging
import os
import select
import sqlite3
import sys
from base64 import b64decode
from getpass import getpass
from subprocess import PIPE, Popen

try:
    # Python 3
    from subprocess import DEVNULL
except ImportError:
    # Python 2
    DEVNULL = open(os.devnull, 'w')

try:
    # Python 3
    from urllib.parse import urlparse
except ImportError:
    # Python 2
```

```
    from urlparse import urlparse

try:
    # Python 3
    from configparser import ConfigParser
    raw_input = input
except ImportError:
    # Python 2
    from ConfigParser import ConfigParser

PY3 = sys.version_info.major > 2
LOG = None
VERBOSE = False
SYS64 = sys.maxsize > 2**32

if not PY3 and os.name == "nt":
    sys.stderr.write("WARNING: You are using Python 2 on Windows. If
your "
                     "passwords include non-alphanumeric characters you
"
                     "will run into problems.\n")
    sys.stderr.write("WARNING: Python 2 + Windows is no longer
supported. "
                     "Please use Python 3 instead\n")

# Windows uses a mixture of different codecs for different components
# ANSI CP1252 for system messages, while NSS uses UTF-8
# To further complicate things, with python 2.7 the default
stdout/stdin codec
# isn't UTF-8 but language dependent (tested on Windows 7)

if os.name == "nt":
    SYS_ENCODING = "cp1252"
    LIB_ENCODING = "utf8"
else:
    SYS_ENCODING = "utf8"
    LIB_ENCODING = "utf8"

# When using pipes stdin/stdout encoding may be None
USR_ENCODING = sys.stdin.encoding or sys.stdout.encoding or "utf8"


def py2_decode(_bytes, encoding=USR_ENCODING):
    if PY3:
        return _bytes
    else:
        return _bytes.decode(encoding)


def py2_encode(_unicode, encoding=USR_ENCODING):
    if PY3:
```

```
        return _unicode
    else:
        return _unicode.encode(encoding)


def type_decode(encoding):
    return lambda x: py2_decode(x, encoding)


def get_version():
    """Obtain version information from git if available otherwise use
    the internal version number
    """
    def internal_version():
        return '.'.join(map(str, __version_info__[:3])) +
''.join(__version_info__[3:])

    try:
        p = Popen(["git", "describe", "--tags"], stdout=PIPE,
stderr=DEVNULL)
    except OSError:
        return internal_version()

    stdout, stderr = p.communicate()

    if p.returncode:
        return internal_version()
    else:
        # Both py2 and py3 return bytes here
        return stdout.decode(USR_ENCODING).strip()


__version_info__ = (0, 8, 0, "+git")
__version__ = get_version()


class NotFoundError(Exception):
    """Exception to handle situations where a credentials file is not
found
    """
    pass


class Exit(Exception):
    """Exception to allow a clean exit from any point in execution
    """
    ERROR = 1
    MISSING_PROFILEINI = 2
    MISSING_SECRETS = 3
    BAD_PROFILEINI = 4
    LOCATION_NO_DIRECTORY = 5
```

```
    BAD_SECRETS = 6

    FAIL_LOCATE_NSS = 10
    FAIL_LOAD_NSS = 11
    FAIL_INIT_NSS = 12
    FAIL_NSS_KEYSLOT = 13
    FAIL_SHUTDOWN_NSS = 14
    BAD_MASTER_PASSWORD = 15
    NEED_MASTER_PASSWORD = 16

    PASSSTORE_NOT_INIT = 20
    PASSSTORE_MISSING = 21
    PASSSTORE_ERROR = 22

    READ_GOT_EOF = 30
    MISSING_CHOICE = 31
    NO_SUCH_PROFILE = 32

    UNKNOWN_ERROR = 100
    KEYBOARD_INTERRUPT = 102

    def __init__(self, exitcode):
        self.exitcode = exitcode

    def __unicode__(self):
        return "Premature program exit with exit code
{0}".format(self.exitcode)


class Credentials(object):
    """Base credentials backend manager
    """
    def __init__(self, db):
        self.db = db

        LOG.debug("Database location: %s", self.db)
        if not os.path.isfile(db):
            raise NotFoundError("ERROR - {0} database not
found\n".format(db))

        LOG.info("Using %s for credentials.", db)

    def __iter__(self):
        pass

    def done(self):
        """Override this method if the credentials subclass needs to do
any
        action after interaction
        """
        pass
```

```
class SqliteCredentials(Credentials):
    """SQLite credentials backend manager
    """
    def __init__(self, profile):
        db = os.path.join(profile, "signons.sqlite")

        super(SqliteCredentials, self).__init__(db)

        self.conn = sqlite3.connect(db)
        self.c = self.conn.cursor()

    def __iter__(self):
        LOG.debug("Reading password database in SQLite format")
        self.c.execute("SELECT hostname, encryptedUsername,
encryptedPassword, encType "
                       "FROM moz_logins")
        for i in self.c:
            # yields hostname, encryptedUsername, encryptedPassword,
encType
            yield i

    def done(self):
        """Close the sqlite cursor and database connection
        """
        super(SqliteCredentials, self).done()

        self.c.close()
        self.conn.close()


class JsonCredentials(Credentials):
    """JSON credentials backend manager
    """
    def __init__(self, profile):
        db = os.path.join(profile, "logins.json")

        super(JsonCredentials, self).__init__(db)

    def __iter__(self):
        with open(self.db) as fh:
            LOG.debug("Reading password database in JSON format")
            data = json.load(fh)

            try:
                logins = data["logins"]
            except Exception:
                LOG.error("Unrecognized format in {0}".format(self.db))
                raise Exit(Exit.BAD_SECRETS)

            for i in logins:
                yield (i["hostname"], i["encryptedUsername"],
```

```
                        i["encryptedPassword"], i["encType"])


class NSSDecoder(object):
    class SECItem(ct.Structure):
        """struct needed to interact with libnss
        """
        _fields_ = [
            ('type', ct.c_uint),
            ('data', ct.c_char_p),  # actually: unsigned char *
            ('len', ct.c_uint),
        ]

    class PK11SlotInfo(ct.Structure):
        """opaque structure representing a logical PKCS slot
        """

    def __init__(self):
        # Locate libnss and try loading it
        self.NSS = None
        self.load_libnss()

        SlotInfoPtr = ct.POINTER(self.PK11SlotInfo)
        SECItemPtr = ct.POINTER(self.SECItem)

        self._set_ctypes(ct.c_int, "NSS_Init", ct.c_char_p)
        self._set_ctypes(ct.c_int, "NSS_Shutdown")
        self._set_ctypes(SlotInfoPtr, "PK11_GetInternalKeySlot")
        self._set_ctypes(None, "PK11_FreeSlot", SlotInfoPtr)
        self._set_ctypes(ct.c_int, "PK11_CheckUserPassword",
SlotInfoPtr, ct.c_char_p)
        self._set_ctypes(ct.c_int, "PK11SDR_Decrypt", SECItemPtr,
SECItemPtr, ct.c_void_p)
        self._set_ctypes(None, "SECITEM_ZfreeItem", SECItemPtr,
ct.c_int)

        # for error handling
        self._set_ctypes(ct.c_int, "PORT_GetError")
        self._set_ctypes(ct.c_char_p, "PR_ErrorToName", ct.c_int)
        self._set_ctypes(ct.c_char_p, "PR_ErrorToString", ct.c_int,
ct.c_uint32)

    def _set_ctypes(self, restype, name, *argtypes):
        """Set input/output types on libnss C functions for automatic
type casting
        """
        res = getattr(self.NSS, name)
        res.restype = restype
        res.argtypes = argtypes
        setattr(self, "_" + name, res)
```

```
    @staticmethod
    def find_nss(locations, nssname):
        """Locate nss is one of the many possible locations
        """
        fail_errors = []

        for loc in locations:
            nsslib = os.path.join(loc, nssname)
            LOG.debug("Loading NSS library from %s", nsslib)

            if os.name == "nt":
                # On windows in order to find DLLs referenced by
nss3.dll
                # we need to have those locations on PATH
                os.environ["PATH"] = ';'.join([loc,
os.environ["PATH"]])
                LOG.debug("PATH is now %s", os.environ["PATH"])
                # However this doesn't seem to work on all setups and
needs to be
                # set before starting python so as a workaround we
chdir to
                # Firefox's nss3.dll location
                if loc:
                    if not os.path.isdir(loc):
                        # No point in trying to load from paths that
don't exist
                        continue

                    workdir = os.getcwd()
                    os.chdir(loc)

            try:
                nss = ct.CDLL(nsslib)
            except OSError as e:
                fail_errors.append((nsslib, str(e)))
            else:
                LOG.debug("Loaded NSS library from %s", nsslib)
                return nss
            finally:
                if os.name == "nt" and loc:
                    # Restore workdir changed above
                    os.chdir(workdir)

        else:
            LOG.error("Couldn't find or load '%s'. This library is
essential "
                      "to interact with your Mozilla profile.",
nssname)
            LOG.error("If you are seeing this error please perform a
system-wide "
                      "search for '%s' and file a bug report indicating
```

```
any "
                      "location found. Thanks!", nssname)
            LOG.error("Alternatively you can try launching
firefox_decrypt "
                      "from the location where you found '%s'. "
                      "That is 'cd' or 'chdir' to that location and run
"
                      "firefox_decrypt from there.", nssname)

            LOG.error("Please also include the following on any bug
report. "
                      "Errors seen while searching/loading NSS:")

            for target, error in fail_errors:
                LOG.error("Error when loading %s was %s", target,
py2_decode(str(error), SYS_ENCODING))

            raise Exit(Exit.FAIL_LOCATE_NSS)

    def load_libnss(self):
        """Load libnss into python using the CDLL interface
        """
        if os.name == "nt":
            nssname = "nss3.dll"
            if SYS64:
                locations = (
                    "",  # Current directory or system lib finder
                    r"C:\Program Files\Mozilla Firefox",
                    r"C:\Program Files\Mozilla Thunderbird",
                    r"C:\Program Files\Nightly",
                )
            else:
                locations = (
                    "",  # Current directory or system lib finder
                    r"C:\Program Files (x86)\Mozilla Firefox",
                    r"C:\Program Files (x86)\Mozilla Thunderbird",
                    r"C:\Program Files (x86)\Nightly",
                    # On windows 32bit these folders can also be 32bit
                    r"C:\Program Files\Mozilla Firefox",
                    r"C:\Program Files\Mozilla Thunderbird",
                    r"C:\Program Files\Nightly",
                )

            # FIXME this was present in the past adding the location
where NSS was found to PATH
            # I'm not sure why this would be necessary. We don't need
to run Firefox...
            # TODO Test on a Windows machine and see if this works
without the PATH change
            # os.environ["PATH"] = ';'.join([os.environ["PATH"],
firefox])
```

```
            # LOG.debug("PATH is now %s", os.environ["PATH"])

        elif os.uname()[0] == "Darwin":
            nssname = "libnss3.dylib"
            locations = (
                "",  # Current directory or system lib finder
                "/usr/local/lib/nss",
                "/usr/local/lib",
                "/opt/local/lib/nss",
                "/sw/lib/firefox",
                "/sw/lib/mozilla",
                "/usr/local/opt/nss/lib",  # nss installed with Brew on
Darwin
                "/opt/pkg/lib/nss",  # installed via pkgsrc
            )

        else:
            nssname = "libnss3.so"
            if SYS64:
                locations = (
                    "",  # Current directory or system lib finder
                    "/usr/lib64",
                    "/usr/lib64/nss",
                    "/usr/lib",
                    "/usr/lib/nss",
                    "/usr/local/lib",
                    "/usr/local/lib/nss",
                    "/opt/local/lib",
                    "/opt/local/lib/nss",
                    os.path.expanduser("~/.nix-profile/lib"),
                )
            else:
                locations = (
                    "",  # Current directory or system lib finder
                    "/usr/lib",
                    "/usr/lib/nss",
                    "/usr/lib32",
                    "/usr/lib32/nss",
                    "/usr/lib64",
                    "/usr/lib64/nss",
                    "/usr/local/lib",
                    "/usr/local/lib/nss",
                    "/opt/local/lib",
                    "/opt/local/lib/nss",
                    os.path.expanduser("~/.nix-profile/lib"),
                )

        # If this succeeds libnss was loaded
        self.NSS = self.find_nss(locations, nssname)

    def handle_error(self):
```

```
        """If an error happens in libnss, handle it and print some
debug information
        """
        LOG.debug("Error during a call to NSS library, trying to obtain
error info")

        code = self._PORT_GetError()
        name = self._PR_ErrorToName(code)
        name = "NULL" if name is None else name.decode(SYS_ENCODING)
        # 0 is the default language (localization related)
        text = self._PR_ErrorToString(code, 0)
        text = text.decode(SYS_ENCODING)

        LOG.debug("%s: %s", name, text)

    def decode(self, data64):
        data = b64decode(data64)
        inp = self.SECItem(0, data, len(data))
        out = self.SECItem(0, None, 0)

        e = self._PK11SDR_Decrypt(inp, out, None)
        LOG.debug("Decryption of data returned %s", e)
        try:
            if e == -1:
                LOG.error("Password decryption failed. Passwords
protected by a Master Password!")
                self.handle_error()
                raise Exit(Exit.NEED_MASTER_PASSWORD)

            res = ct.string_at(out.data, out.len).decode(LIB_ENCODING)
        finally:
            # Avoid leaking SECItem
            self._SECITEM_ZfreeItem(out, 0)

        return res


class NSSInteraction(object):
    """
    Interact with lib NSS
    """
    def __init__(self):
        self.profile = None
        self.NSS = NSSDecoder()

    def load_profile(self, profile):
        """Initialize the NSS library and profile
        """
        LOG.debug("Initializing NSS with profile path '%s'", profile)
        self.profile = profile
```

```
        profile = profile.encode(LIB_ENCODING)

        e = self.NSS._NSS_Init(b"sql:" + profile)
        LOG.debug("Initializing NSS returned %s", e)

        if e != 0:
            LOG.error("Couldn't initialize NSS, maybe '%s' is not a
valid profile?", self.profile)
            self.NSS.handle_error()
            raise Exit(Exit.FAIL_INIT_NSS)

    def authenticate(self, interactive):
        """Check if the current profile is protected by a master
password,
        prompt the user and unlock the profile.
        """
        LOG.debug("Retrieving internal key slot")
        keyslot = self.NSS._PK11_GetInternalKeySlot()

        LOG.debug("Internal key slot %s", keyslot)
        if not keyslot:
            LOG.error("Failed to retrieve internal KeySlot")
            self.NSS.handle_error()
            raise Exit(Exit.FAIL_NSS_KEYSLOT)

        try:
            # NOTE It would be great to be able to check if the profile
is
            # protected by a master password. In C++ one would do:
            #   if (keyslot->needLogin):
            # however accessing instance methods is not supported by
ctypes.
            # More on this topic: http://stackoverflow.com/a/19636310
            # A possibility would be to define such function using
cython but
            # this adds an unecessary runtime dependency
            password = ask_password(self.profile, interactive)

            if password:
                LOG.debug("Authenticating with password '%s'",
password)
                e = self.NSS._PK11_CheckUserPassword(keyslot,
password.encode(LIB_ENCODING))

                LOG.debug("Checking user password returned %s", e)

                if e != 0:
                    LOG.error("Master password is not correct")

                    self.NSS.handle_error()
                    raise Exit(Exit.BAD_MASTER_PASSWORD)
```

```
            else:
                LOG.warning("Attempting decryption with no Master
Password")
        finally:
            # Avoid leaking PK11KeySlot
            self.NSS._PK11_FreeSlot(keyslot)

    def unload_profile(self):
        """Shutdown NSS and deactive current profile
        """
        e = self.NSS._NSS_Shutdown()

        if e != 0:
            LOG.error("Couldn't shutdown current NSS profile")

            self.NSS.handle_error()
            raise Exit(Exit.FAIL_SHUTDOWN_NSS)

    def decode_entry(self, user64, passw64):
        """Decrypt one entry in the database
        """
        LOG.debug("Decrypting username data '%s'", user64)
        user = self.NSS.decode(user64)

        LOG.debug("Decrypting password data '%s'", passw64)
        passw = self.NSS.decode(passw64)

        return user, passw

    def decrypt_passwords(self, export, output_format="human",
csv_delimiter=";", csv_quotechar="|"):
        """
        Decrypt requested profile using the provided password and print
out all
        stored passwords.
        """
        # Any password in this profile store at all?
        got_password = False
        header = True

        credentials = obtain_credentials(self.profile)

        LOG.info("Decrypting credentials")
        to_export = {}
        outputs = []

        if output_format == "csv":
            csv_writer = csv.DictWriter(
                sys.stdout, fieldnames=["url", "user", "password"],
                lineterminator="\n", delimiter=csv_delimiter,
                quotechar=csv_quotechar, quoting=csv.QUOTE_ALL,
```

```
            )
            if header:
                csv_writer.writeheader()

        for url, user, passw, enctype in credentials:
            got_password = True

            # enctype informs if passwords are encrypted and protected
by
            # a master password
            if enctype:
                user, passw = self.decode_entry(user, passw)

            LOG.debug("Decoding username '%s' and password '%s' for
website '%s'", user, passw, url)
            LOG.debug("Decoding username '%s' and password '%s' for
website '%s'", type(user), type(passw), type(url))

            if export:
                # Keep track of web-address, username and passwords
                # If more than one username exists for the same web-
address
                # the username will be used as name of the file
                address = urlparse(url)

                if address.netloc not in to_export:
                    to_export[address.netloc] = {user: passw}

                else:
                    to_export[address.netloc][user] = passw

            if output_format == "csv":
                output = {"url": url, "user": user, "password": passw}
                if PY3:
                    csv_writer.writerow(output)
                else:
                    csv_writer.writerow({k: v.encode(USR_ENCODING) for
k, v in output.items()})
            elif output_format == "json":
                output = {"url": url, "user": user, "password": passw}
                outputs.append(output)

            else:
                output = (
                    u"\nWebsite:   {0}\n".format(url),
                    u"Username: '{0}'\n".format(user),
                    u"Password: '{0}'\n".format(passw),
                )
                for line in output:
                    sys.stdout.write(py2_encode(line, USR_ENCODING))
            if output_format == "json":
```

```
                print(json.dumps(outputs))


        credentials.done()

        if not got_password:
            LOG.warning("No passwords found in selected profile")

        if export:
            return to_export


def test_password_store(export, pass_cmd):
    """Check if pass from passwordstore.org is installed
    If it is installed but not initialized, initialize it
    """
    # Nothing to do here if exporting wasn't requested
    if not export:
        LOG.debug("Skipping password store test, not exporting")
        return

    LOG.debug("Testing if password store is installed and configured")

    try:
        p = Popen([pass_cmd], stdout=PIPE, stderr=PIPE)
    except OSError as e:
        if e.errno == 2:
            LOG.error("Password store is not installed and exporting
was requested")
            raise Exit(Exit.PASSSTORE_MISSING)
        else:
            LOG.error("Unknown error happened.")
            LOG.error("Error was %s", e)
            raise Exit(Exit.UNKNOWN_ERROR)

    out, err = p.communicate()
    LOG.debug("pass returned: %s %s", out, err)

    if p.returncode != 0:
        if 'Try "pass init"' in err:
            LOG.error("Password store was not initialized.")
            LOG.error("Initialize the password store manually by using
'pass init'")
            raise Exit(Exit.PASSSTORE_NOT_INIT)
        else:
            LOG.error("Unknown error happened when running 'pass'.")
            LOG.error("Stdout/Stderr was '%s' '%s'", out, err)
            raise Exit(Exit.UNKNOWN_ERROR)


def obtain_credentials(profile):
```

```
    """Figure out which of the 2 possible backend credential engines is
available
    """
    try:
        credentials = JsonCredentials(profile)
    except NotFoundError:
        try:
            credentials = SqliteCredentials(profile)
        except NotFoundError:
            LOG.error("Couldn't find credentials file (logins.json or
signons.sqlite).")
            raise Exit(Exit.MISSING_SECRETS)

    return credentials


def export_pass(to_export, pass_cmd, prefix, username_prefix):
    """Export given passwords to password store

    Format of "to_export" should be:
        {"address": {"login": "password", ...}, ...}
    """
    LOG.info("Exporting credentials to password store")
    if prefix:
        prefix = u"{0}/".format(prefix)

    LOG.debug("Using pass prefix '%s'", prefix)

    for address in to_export:
        for user, passw in to_export[address].items():
            # When more than one account exist for the same address,
add
            # the login to the password identifier
            if len(to_export[address]) > 1:
                passname = u"{0}{1}/{2}".format(prefix, address, user)

            else:
                passname = u"{0}{1}".format(prefix, address)

            LOG.debug("Exporting credentials for '%s'", passname)

            data = u"{0}\n{1}{2}\n".format(passw, username_prefix,
user)

            LOG.debug("Inserting pass '%s' '%s'", passname, data)

            # NOTE --force is used. Existing passwords will be
overwritten
            cmd = [pass_cmd, "insert", "--force", "--multiline",
passname]
```

```
            LOG.debug("Running command '%s' with stdin '%s'", cmd,
data)

            p = Popen(cmd, stdout=PIPE, stderr=PIPE, stdin=PIPE)
            out, err = p.communicate(data.encode(SYS_ENCODING))

            if p.returncode != 0:
                LOG.error("ERROR: passwordstore exited with non-zero:
%s", p.returncode)
                LOG.error("Stdout/Stderr was '%s' '%s'", out, err)
                raise Exit(Exit.PASSSTORE_ERROR)

            LOG.debug("Successfully exported '%s'", passname)


def get_sections(profiles):
    """
    Returns hash of profile numbers and profile names.
    """
    sections = {}
    i = 1
    for section in profiles.sections():
        if section.startswith("Profile"):
            sections[str(i)] = profiles.get(section, "Path")
            i += 1
        else:
            continue
    return sections


def print_sections(sections, textIOWrapper=sys.stderr):
    """
    Prints all available sections to an textIOWrapper (defaults to
sys.stderr)
    """
    for i in sorted(sections):
        textIOWrapper.write("{0} -> {1}\n".format(i, sections[i]))
    textIOWrapper.flush()


def ask_section(profiles, choice_arg):
    """
    Prompt the user which profile should be used for decryption
    """
    sections = get_sections(profiles)

    # Do not ask for choice if user already gave one
    if choice_arg and len(choice_arg) == 1:
        choice = choice_arg[0]
    else:
        # If only one menu entry exists, use it without prompting
```

```
        if len(sections) == 1:
            choice = "1"

        else:
            choice = None
            while choice not in sections:
                sys.stderr.write("Select the Firefox profile you wish
to decrypt\n")
                print_sections(sections)
                try:
                    choice = raw_input()
                except EOFError:
                    LOG.error("Could not read Choice, got EOF")
                    raise Exit(Exit.READ_GOT_EOF)

    try:
        final_choice = sections[choice]
    except KeyError:
        LOG.error("Profile No. %s does not exist!", choice)
        raise Exit(Exit.NO_SUCH_PROFILE)

    LOG.debug("Profile selection matched %s", final_choice)

    return final_choice


def ask_password(profile, interactive):
    """
    Prompt for profile password
    """
    if not PY3:
        profile = profile.encode(SYS_ENCODING)

    passmsg = "\nMaster Password for profile {0}: ".format(profile)

    if sys.stdin.isatty() and interactive:
        passwd = getpass(passmsg)

    else:
        # Ability to read the password from stdin (echo "pass" |
./firefox_...)
        if sys.stdin in select.select([sys.stdin], [], [], 0)[0]:
            passwd = sys.stdin.readline().rstrip("\n")
        else:
            LOG.warning("Master Password not provided, continuing with
blank password")
            passwd = ""

    return py2_decode(passwd)
```

```
def read_profiles(basepath, list_profiles):
    """
    Parse Firefox profiles in provided location.
    If list_profiles is true, will exit after listing available
profiles.
    """
    profileini = os.path.join(basepath, "profiles.ini")

    LOG.debug("Reading profiles from %s", profileini)

    if not os.path.isfile(profileini):
        LOG.warning("profile.ini not found in %s", basepath)
        raise Exit(Exit.MISSING_PROFILEINI)

    # Read profiles from Firefox profile folder
    profiles = ConfigParser()
    profiles.read(profileini)

    LOG.debug("Read profiles %s", profiles.sections())

    if list_profiles:
        LOG.debug("Listing available profiles...")
        print_sections(get_sections(profiles), sys.stdout)
        raise Exit(0)

    return profiles


def get_profile(basepath, interactive, choice, list_profiles):
    """
    Select profile to use by either reading profiles.ini or assuming
given
    path is already a profile
    If interactive is false, will not try to ask which profile to
decrypt.
    choice contains the choice the user gave us as an CLI arg.
    If list_profiles is true will exits after listing all available
profiles.
    """
    try:
        profiles = read_profiles(basepath, list_profiles)
    except Exit as e:
        if e.exitcode == Exit.MISSING_PROFILEINI:
            LOG.warning("Continuing and assuming '%s' is a profile
location", basepath)
            profile = basepath

            if list_profiles:
                LOG.error("Listing single profiles not permitted.")
                raise
```

```
            if not os.path.isdir(profile):
                LOG.error("Profile location '%s' is not a directory",
profile)
                raise
        else:
            raise
    else:
        if not interactive:
            sections = get_sections(profiles)

            if choice and len(choice) == 1:
                try:
                    section = sections[(choice[0])]
                except KeyError:
                    LOG.error("Profile No. %s does not exist!",
choice[0])
                    raise Exit(Exit.NO_SUCH_PROFILE)

            elif len(sections) == 1:
                section = sections['1']

            else:
                LOG.error("Don't know which profile to decrypt. We are
in non-interactive mode and -c/--choice is missing.")
                raise Exit(Exit.MISSING_CHOICE)
        else:
            # Ask user which profile to open
            section = ask_section(profiles, choice)

        section = py2_decode(section, LIB_ENCODING)
        profile = os.path.join(basepath, section)

        if not os.path.isdir(profile):
            LOG.error("Profile location '%s' is not a directory. Has
profiles.ini been tampered with?", profile)
            raise Exit(Exit.BAD_PROFILEINI)

    return profile


def parse_sys_args():
    """Parse command line arguments
    """

    if os.name == "nt":
        profile_path = os.path.join(os.environ['APPDATA'], "Mozilla",
"Firefox")
    elif os.uname()[0] == "Darwin":
        profile_path = "~/Library/Application Support/Firefox"
    else:
        profile_path = "~/.mozilla/firefox"
```

```
    parser = argparse.ArgumentParser(
        description="Access Firefox/Thunderbird profiles and decrypt
existing passwords"
    )
    parser.add_argument("profile", nargs="?", default=profile_path,
                        type=type_decode(SYS_ENCODING),
                        help="Path to profile folder (default:
{0})".format(profile_path))
    parser.add_argument("-e", "--export-pass", action="store_true",
                        help="Export URL, username and password to pass
from passwordstore.org")
    parser.add_argument("--pass-compat", action="store",
                        choices={"default", "browserpass", "username"},
                        default="default",
                        help="Export username as is (default), or with
one of the compatiblity modes")
    parser.add_argument("-p", "--pass-prefix", action="store",
default=u"web",
                        help="Prefix for export to pass from
passwordstore.org (default: %(default)s)")
    parser.add_argument("-m", "--pass-cmd", action="store",
default=u"pass",
                        help="Command/path to use when exporting to
pass (default: %(default)s)")
    parser.add_argument("-f", "--format", action="store",
choices={"csv", "human", "json"},
                        default="human", help="Format for the output.")
    parser.add_argument("-d", "--delimiter", action="store",
default=";",
                        help="The delimiter for csv output")
    parser.add_argument("-q", "--quotechar", action="store",
default='"',
                        help="The quote char for csv output")
    parser.add_argument("-t", "--tabular", action="store_true",
help=argparse.SUPPRESS)
    parser.add_argument("-n", "--no-interactive", dest="interactive",
                        default=True, action="store_false",
                        help="Disable interactivity.")
    parser.add_argument("-c", "--choice", nargs=1,
                        help="The profile to use (starts with 1). If
only one profile, defaults to that.")
    parser.add_argument("-l", "--list", action="store_true",
                        help="List profiles and exit.")
    parser.add_argument("-v", "--verbose", action="count", default=0,
                        help="Verbosity level. Warning on -vv (highest
level) user input will be printed on screen")
    parser.add_argument("--version", action="version",
version=__version__,
                        help="Display version of firefox_decrypt and
exit")
```

```
    args = parser.parse_args()

    # replace character you can't enter as argument
    if args.delimiter == "\\t":
        args.delimiter = "\t"

    if args.tabular:
        args.format = "csv"
        args.delimiter = "\t"
        args.quotechar = "'"

    return args


def setup_logging(args):
    """Setup the logging level and configure the basic logger
    """
    if args.verbose == 1:
        level = logging.INFO
    elif args.verbose >= 2:
        level = logging.DEBUG
    else:
        level = logging.WARN

    logging.basicConfig(
        format="%(asctime)s - %(levelname)s - %(message)s",
        level=level,
    )

    global LOG
    LOG = logging.getLogger(__name__)


def main():
    """Main entry point
    """
    args = parse_sys_args()

    setup_logging(args)

    if args.tabular:
        LOG.warning("--tabular is deprecated. Use `--format csv --
delimiter \\t` instead")

    LOG.info("Running firefox_decrypt version: %s", __version__)
    LOG.debug("Parsed commandline arguments: %s", args)
    LOG.debug("Running with encodings: USR: %s, SYS: %s, LIB: %s",
USR_ENCODING, SYS_ENCODING, LIB_ENCODING)

    # Check whether pass from passwordstore.org is installed
    test_password_store(args.export_pass, args.pass_cmd)
```

```
    # Initialize nss before asking the user for input
    nss = NSSInteraction()

    basepath = os.path.expanduser(args.profile)

    # Read profiles from profiles.ini in profile folder
    profile = get_profile(basepath, args.interactive, args.choice,
args.list)

    # Start NSS for selected profile
    nss.load_profile(profile)
    # Check if profile is password protected and prompt for a password
    nss.authenticate(args.interactive)
    # Decode all passwords
    to_export = nss.decrypt_passwords(
        export=args.export_pass,
        output_format=args.format,
        csv_delimiter=args.delimiter,
        csv_quotechar=args.quotechar,
    )

    if args.export_pass:
        # List of compatibility modes for username prefixes
        compat = {
            "username": "username: ",
            "browserpass": "login: ",
        }

        username_prefix = compat.get(args.pass_compat, "")
        export_pass(to_export, args.pass_cmd, args.pass_prefix,
username_prefix)

    # And shutdown NSS
    nss.unload_profile()


if __name__ == "__main__":
    try:
        main()
    except KeyboardInterrupt as e:
        print("Quit.")
        sys.exit(Exit.KEYBOARD_INTERRUPT)
    except Exit as e:
        sys.exit(e.exitcode)
```