15.3. `secrets` — Generate secure random numbers for managing secrets

New in version 3.6.

Source code: Lib/secrets.py

The `secrets` module is used for generating cryptographically strong random numbers suitable for managing data such as passwords, account authentication, security tokens, and related secrets.

In particularly, `secrets` should be used in preference to the default pseudo-random number generator in the `random` module, which is designed for modelling and simulation, not security or cryptography.

See also: **PEP 506**

15.3.1. Random numbers

The `secrets` module provides access to the most secure source of randomness that your operating system provides.

class `secrets.`**`SystemRandom`**
A class for generating random numbers using the highest-quality sources provided by the operating system. See `random.SystemRandom` for additional details.

`secrets.`**`choice`**(*sequence*)
Return a randomly-chosen element from a non-empty sequence.

`secrets.`**`randbelow`**(*n*)
Return a random int in the range [0, *n*).

`secrets.`**`randbits`**(*k*)
Return an int with *k* random bits.

15.3.2. Generating tokens

The `secrets` module provides functions for generating secure tokens, suitable for applications such as password resets, hard-to-guess URLs, and similar.

`secrets.`**`token_bytes`**([*nbytes=None*])

Return a random byte string containing *nbytes* number of bytes. If *nbytes* is `None` or not supplied, a reasonable default is used.

```
>>> token_bytes(16)
b'\xebr\x17D*t\xae\xd4\xe3S\xb6\xe2\xebP1\x8b'
```

`secrets.`**`token_hex`**([*nbytes=None*])

Return a random text string, in hexadecimal. The string has *nbytes* random bytes, each byte converted to two hex digits. If *nbytes* is `None` or not supplied, a reasonable default is used.

```
>>> token_hex(16)
'f9bf78b9a18ce6d46a0cd2b0b86df9da'
```

`secrets.`**`token_urlsafe`**([*nbytes=None*])

Return a random URL-safe text string, containing *nbytes* random bytes. The text is Base64 encoded, so on average each byte results in approximately 1.3 characters. If *nbytes* is `None` or not supplied, a reasonable default is used.

```
>>> token_urlsafe(16)
'Drmhze6EPcv0fN_81Bj-nA'
```

15.3.2.1. How many bytes should tokens use?

To be secure against brute-force attacks, tokens need to have sufficient randomness. Unfortunately, what is considered sufficient will necessarily increase as computers get more powerful and able to make more guesses in a shorter period. As of 2015, it is believed that 32 bytes (256 bits) of randomness is sufficient for the typical use-case expected for the `secrets` module.

For those who want to manage their own token length, you can explicitly specify how much randomness is used for tokens by giving an `int` argument to the various `token_*` functions. That argument is taken as the number of bytes of randomness to use.

Otherwise, if no argument is provided, or if the argument is `None`, the `token_*` functions will use a reasonable default instead.

Note: That default is subject to change at any time, including during maintenance releases.

15.3.3. Other functions

`secrets.`**`compare_digest`**(*a*, *b*)

Return `True` if strings *a* and *b* are equal, otherwise `False`, in such a way as to reduce the risk of timing attacks. See `hmac.compare_digest()` for additional details.

15.3.4. Recipes and best practices

This section shows recipes and best practices for using `secrets` to manage a basic level of security.

Generate an eight-character alphanumeric password:

```
import string
alphabet = string.ascii_letters + string.digits
password = ''.join(choice(alphabet) for i in range(8))
```

Note: Applications should not store passwords in a recoverable format, whether plain text or encrypted. They should be salted and hashed using a cryptographically-strong one-way (irreversible) hash function.

Generate a ten-character alphanumeric password with at least one lowercase character, at least one uppercase character, and at least three digits:

```
import string
alphabet = string.ascii_letters + string.digits
while True:
    password = ''.join(choice(alphabet) for i in range(10))
    if (any(c.islower() for c in password)
            and any(c.isupper() for c in password)
            and sum(c.isdigit() for c in password) >= 3):
        break
```

Generate an XKCD-style passphrase:

```
# On standard Linux systems, use a convenient dictionary file.
# Other platforms may need to provide their own word-list.
with open('/usr/share/dict/words') as f:
    words = [word.strip() for word in f]
    password = ' '.join(choice(words) for i in range(4))
```

Generate a hard-to-guess temporary URL containing a security token suitable for password recovery applications:

```
url = 'https://mydomain.com/reset=' + token_urlsafe()
```