Security Practical Answers

Dr Chris G. Willcocks

Last Modified: October 2, 2019

Practical 1

Prevent dictionary attack

This solution uses heat to prevent a dictionary attack. While you can do it any way you wish, this approach is efficient as it requires no data structures or loop iterations. It keeps an event log (and prevent it from growing too large) and records the heat per IP address. Note it could also be stored per user or both, with a different set of implications. Also, the heat does not need to be decremented in a loop as you can simply keep track of when the last heat was added. There are many different ways to implement this, each with its own implications, and this serves as a basic example:

Listing 1: Python

```python
from http.server import BaseHTTPRequestHandler, HTTPServer
from collections import OrderedDict
import base64
import time

usernames = ["admin", "chris", "greg", "john", "test"]
passwords = ["12345qwert", "ncc1701d", "zxcvbn", "1qaz2wsx", "ncc1701d"]

event_log = OrderedDict()

...

def verify(self, data):

    # ensure event log never gets too big (run out of memory)
    while len(event_log) > 1000000:
        event_log.popitem(last=False)

    # fetch heat from event logs (this is simply stored per IP address)
    heat_key = self.client_address[0]+'heat'
    last_heat_key = self.client_address[0]+'last_heat'
    if heat_key not in event_log:
        event_log[heat_key] = 0.0
        event_log[last_heat_key] = 0.0

    # heat logic (could be improved)
    event_log[heat_key] -= time.time()-event_log[last_heat_key]
    event_log[heat_key]  = min(max(event_log[heat_key],0.0), 1000.0)
    event_log[heat_key] += 1.0

    print(event_log)

    # only allow successful login if heat is small enough
    if event_log[heat_key] < 5.0:
        for i in range(len(usernames)):
            if data == 'Basic '+base64.b64encode(bytes(usernames[i]+':'+passwords[i], 'UTF-8')).decode("utf-8"):
                print(usernames[i]+' has logged in!')
                return True

    # there was a failed login attempt, so accumulate heat
    event_log[last_heat_key] = time.time()

    return False
```

Hashed passwords

You can either store hashes in bytes or as encoded strings. Here's a simple example:

Listing 2: Python

```python
from http.server import BaseHTTPRequestHandler, HTTPServer
import base64
import hashlib

usernames = ["admin", "chris", "greg", "john", "test"]

passwords = [
    '4f58b8c57d068b133c6c7308248dbb8dcf76405311fbd3591243c3840bba906a',
    '43c405a9b48588fde67c2a0d89439c0f7b013efa6f2c8aa3e705b07633a09b5c',
    '2e0e630297236bab0cb85333aab77e2d4f85a58566aaff03e7e2e42ca0b4bba1',
    '287e9b1c43b8d963a70a1956887fab8126c829b4ef76ab49b2bb1b0db02a0957',
    '43c405a9b48588fde67c2a0d89439c0f7b013efa6f2c8aa3e705b07633a09b5c'
]

...

def verify(self, data):
    raw_data = base64.b64decode(data[6:]).decode('UTF-8')
    username = raw_data.split(':')[0]
    password = raw_data.split(':')[1]
    hashed_password = hashlib.sha3_256(password.encode('UTF-8')).hexdigest()

    for i in range(len(usernames)):
        if usernames[i] == username and passwords[i] == hashed_password:
            print(usernames[i]+' has logged in!')
            return True
    return False

...
```

Notice how the hashes for ‘chris’ and ‘test’ users are the same.

Salted passwords

This example applies salts manually using cryptographically strong random numbers with the secrets module.

Listing 3: Python

```python
from http.server import BaseHTTPRequestHandler, HTTPServer
import base64
import hashlib
import secrets

usernames = ["admin", "chris", "greg", "john", "test"]

# generated with sha3_256(password+salt)
salted_passwords = [
    '2fd5cee9ac10fa04062a9591291a58afc12813b695a884e20127632e0a385f77',
    '740dd50b45a45a8d966da5e7379c5aa488c907b5d9a900f2ec8ba217fd8e0746', # <- ensure different
    '1c1f4e9ba0b663d3b0156bb43d9719f69d205884fb33eb3f26ecc64cd8061e3f',
    'a033657625c982a18cc707a919407ac885f0ffb36df2fc00a00e144b85b1210e',
    'f63f611bc5ff974220d48d166884427cb02748044f3ef36d9d966080ed2fbaff'  # <- ensure different
]

# generated with 'secrets.token_hex(8)'
salts = [
    '1af0c96ad2cc7672',
    '4f4ba4445148cdce',
    '619702861b47e8a9',
    '31cf86b247c4b175',
    '986e14e7134b735a'
]

...

def verify(self, data):
    raw_data = base64.b64decode(data[6:]).decode('UTF-8')
    username = raw_data.split(':')[0]
    password = raw_data.split(':')[1]

    for i in range(len(usernames)):
        salted_password = hashlib.sha3_256((password+salts[i]).encode('UTF-8')).hexdigest()
        if usernames[i] == username and salted_passwords[i] == salted_password:
            print(usernames[i]+' has logged in!')
            return True
    return False

...
```

Notice how the hashes for ‘chris’ and ‘test’ users are now different, even though they have the same password.