18.12.2023

Курс:

Практическая работа к уроку № Lesson_4

--
Message Authentication Code

Задание_1:

Напишите небольшое веб-приложение, используя готовый фреймворк (web.py, Sinatra, …). В этом приложении будет один эндпойнт, который принимает на вход имя файла и подпись.
`http://localhost:9000/test?file=foo&signature=46b4ec586117154dacd49d664e5d63fdc88efb51`

Сервер при получении подобного запроса будет брать HMAC (функцию HMAC реализуйте самостоятельно) от файла `foo` и сверять полученное значение со значением `signature`. Если значения совпадут, то сервер покажет файл, иначе сервер вернет ошибку.

Теперь напишите функцию `insecure_compare`, которая побайтово сравнивает строки и реализует “ранний выход” (т.е. возвращает False на первой паре не совпавших байт). В цикле `insecure_compare` добавьте искусственную задержку в 50 мс (например, функцией time.sleep(0.05) в Python).

Для проверки подписи используйте `insecure_compare` вместо обычной функции сравнения.

Используя факт задержек и раннего выхода, напишите программу, которая будет подбирать валидную подпись для любого файла (без знания ключа!).

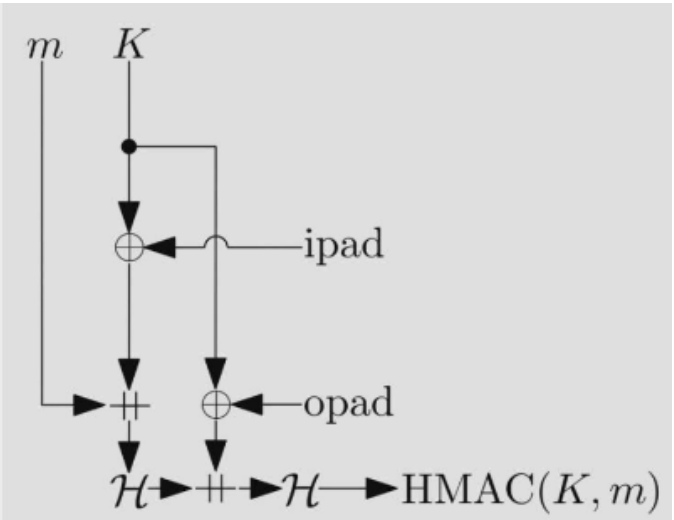
В реальном мире подобные уязвимости тяжело эксплуатировать из-за задержек в сети. Однако это хороший пример атаки по сторонним каналам и похожие уязвимости время от времени встречаются в CTF-ах. Например, в 50m-ctf (https://ajxchapman.github.io/security/2019/03/26/h1-702-ctf-2019.html) от HackerOne была очень похожая уязвимость.
http://cryptopals.com/sets/4/challenges/31

MAC — аббревиатура message authentication code (код аутентификации сообщения), не стоит путать с MAC-адресом, это абсолютно разные вещи. Он предоставляет возможность подтвердить, что сообщение не было изменено, то есть, по сути, обладает свойством хеш-функции. Также он позволяет быть уверенным, что сообщение получено от лица, которое знает секретный ключ. Это дополнительное свойство, которое хеш-функция гарантировать не может, то есть MAC — это уже криптосхема, а не примитив:

HMAC

HMAC =
H ((k ^ opad) || H ((k ^ ipad) || m)

k - ключ
m - сообщение
opad, ipad - паддинги



Настройки web-serv:

```
┌──(kali㉿kali)-[~]
└─$ pip3 install web.py -U

# https://webpy.org
# pip3 install web.py
# ... python3 app.py

import web

urls = (
    '/(.*)', 'hello'
)
app = web.application(urls, globals())

class hello:
    def GET(self, name):
        if not name:
            name = 'World'
        return 'Hello, ' + name + '!'

if __name__ == "__main__":
    app.run()
```

```
┌──(kali㉿kali)-[~/tmp/task4]
└─$ cat server.py
import web
import hmac
import time
import binascii

def insecure_compare(s1, s2):
        b1 = bytearray(binascii.unhexlify(s1))
        b2 = bytearray(binascii.unhexlify(s2))

        for i in xrange(len(min(b1,b2))):
                if b1[i] != b2[i]:
                        return False
```

```
                time.sleep(0.05)

        return True

KEY = 'YELLOW SUBMARINE'

urls = (
    '/(.*)', 'hello'
)
app = web.application(urls, globals())

class hello:
        def GET(self):
                params = web.input(_method='get')

                fil = params['file'] if 'file' in params else None
                signature = params['signature'] if 'signature' in params else None

                if fil and signature and len(signature) == 40:
                        with open(fil, 'r') as f:
                                file_content = f.read()

                        file_hmac = hmac.hmac(bytearray(file_content),
bytearray(KEY))

                        if insecure_compare(file_hmac, signature):
                                return file_content

                return 'Access Denied...'

if __name__ == "__main__":
    app.run()


┌──(kali㉿kali)-[~/tmp/task4]
└─$ cat server2.py
import socketserver
import time
from prob31 import myhmac
from prob28 import sha1_from_github
from prob1 import hexToRaw

RESPONSE_500 = b'HTTP/1.1 500 Internal Server Error\n'
RESPONSE_200 = b'HTTP/1.1 200 OK\n'
COMPARE_DELAY = .050

HMAC_KEY = b'YELLOW SUBMARINE'

class MyWebServer(socketserver.StreamRequestHandler):
    def handle(self):
        while True:
            # typical url: http://localhost:9000/test?
file=foo&signature=46b4ec586117154dacd49d664e5d63fdc88efb51
            line = self.rfile.readline().strip();
            if (line == None):
                break;
```

```
            file_index = line.find(b'file=')
            signature_index = line.find(b'&signature=')
            if ((file_index == -1 ) or (signature_index == -1)):
                continue;
            file_value = line[file_index + len(b'file='):signature_index]
            signature_hex = line[signature_index+len(b'&signature='):]
            computed_signature_hex = bytes(myhmac(sha1_from_github, file_value,
HMAC_KEY), 'UTF-8');

            if (insecure_equals(hexToRaw(signature_hex),
hexToRaw(computed_signature_hex))):
                self.wfile.write(RESPONSE_200);
            else:
                self.wfile.write(RESPONSE_500);
        self.wfile.write(self.data.upper())

def insecure_equals(this, that):
    if (len(this) != len(that)):
        return False;
    for i in range(len(this)):
        if (this[i] != that[i]):
            return False;
        time.sleep(COMPARE_DELAY);
    return True;

def start_server(delay):
    HOST, PORT = "localhost", 9000
    global COMPARE_DELAY
    COMPARE_DELAY=delay;
    # Create the server, binding to localhost on port 9999
    server = socketserver.TCPServer((HOST, PORT), MyWebServer)
    server.allow_reuse_address = True;

    # Activate the server; this will keep running until you
    # interrupt the program with Ctrl-C
    server.serve_forever()

┌──(kali㉿kali)-[~/tmp/task4]
└─$ python server.py 31337
```

Задание 2

Найдите реализацию SHA-1 на вашем языке программирования (например, можно использовать https://github.com/ajalt/python-sha1 для Python). Примечание: это задание является подготовкой к атаке Hash Length Extension, поэтому нужна именно чистая реализация SHA-1, а не библиотечная.
Напишите функцию, которая будет реализовывать MAC вида `SHA1(key || message)`, где `||` - конкатенация.
Убедитесь, что вы не можете подделать сообщение, не изменив при этом MAC.
http://cryptopals.com/sets/4/challenges/28

```
┌──(kali㉿kali)-[~/tmp/task4]
└─$ ls
secret.txt  server2.py  server.py  sha1.py  test.py
```

```
┌──(kali㉿kali)-[~/tmp/task4]
└─$ echo hello | python sha1.py

...
# ПРИМЕР
import hashlib
import hmac
import base64

def make_digest(message, key):
    key = bytes(key, 'UTF-8')
    message = bytes(message, 'UTF-8')

    digester = hmac.new(key, message, hashlib.sha1)
    signature1 = digester.hexdigest()

    signature2 = base64.urlsafe_b64encode(signature1)
    return str(signature2, 'UTF-8')


result = make_digest('123', 'secret')
print(result)
```

Задание 3 (*)

В файле leak_db.txt находятся хеши паролей из скомпрометированной базы данных. Вам необходимо восстановить все пароли из базы данных. Известно, что пароли словарные.

Напишите функцию, которая производит MD паддинг для произвольного сообщения. Проверьте, что написанная вами функция генерирует такой же паддинг как и функция во взятой вами реализации SHA-1.

Возьмите MAC сообщения, которое вы хотите подделать (это просто SHA-1 хеш). Разбейте хеш на 32-битные регистры SHA-1 (a, b, c, d, e).

Измените реализацию SHA-1 так, чтобы при вызове можно было передать новый значения для регистров a, b, c, d, e (обычно они заполнены магическими числами).

Реализуйте атаку Hash Length Extension - сгенерируйте MAC для строки `comment1=cooking%20MCs;userdata=foo;comment2=%20like%20a%20pound%20of%20bacon` со случайным ключом (ключ можно взять из /usr/share/dict/words; используйте его только для генерации исходного MAC). Подделайте сообщение, чтобы оно заканчивалось на `;admin=true`.
http://cryptopals.com/sets/4/challenges/29

Выводы:

MAC — алгоритм, который принимает на вход сообщение и ключ, а на выходе производит так называемую имитовставку (это слово используется только в русском языке, на английском — tag или просто MAC), которая позволяет гарантировать целостность сообщения и подтверждать отправителя, владеющего таким же секретным ключом, как и мы.
HMAC — MAC на основе хешей.
Hash length extension attack — тип атаки на хеш-функцию, заключающейся в добавлении новой информации в конец исходного сообщения. При этом новое значение хэша может быть вычислено,

даже если содержимое исходного сообщения неизвестно. При использовании хеш-функции в качестве имитовставки новое значение будет действительным кодом аутентификации для нового сообщения.
Структура Меркла-Дамгора — метод построения криптографических хеш-функций, предусматривающий разбиение входных сообщений произвольной длины на блоки фиксированной длины и работающий с ними по очереди с помощью функции сжатия, каждый раз принимая входной блок с выходным от предыдущего прохода.
Side-channel attack — класс атак, направленных на уязвимости в практической реализации криптосистемы, использует информацию о физических процессах в устройстве, которые не рассматриваются в теоретическом описании криптографического алгоритма.

> Вся информация в данной работе представлена исключительно в ознакомительных целях! Любое использование на практике без согласования тестирования подпадает под действие УК РФ.
> - https://gb.ru

*Выполнил: ==AndreiM