6G7Z1009 Introduction to Computer Forensics and Security week 10 - Lab

This lab aims to learn secure hashes and message digests

Preparation

1.1 Reboot your machine to Linux environment

1. Hash encryption using DuckGo Search Engine

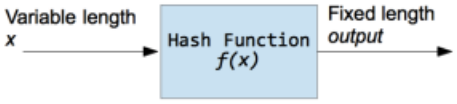
1. Read this article on cryptography hacks:
   http://thehackernews.com/2014/01/cryptography-hacks-hash-encryption.html

2. Testing Hash functions on PracticalCryptography.com

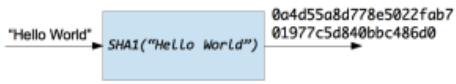
   Visit http://practicalcryptography.com/hashes/ and read about the six hashing functions described there. Test each function by running the given python code if available.

3. Hashing Strings with Python (http://www.pythoncentral.io/hashing--strings--with--python/)

A hash function is a function that takes input of a variable length sequence of bytes and converts it to a fixed length sequence. It is a one way function. This means if *f* is the hashing function, calculating *f(x)* is pretty fast and simple, but trying to obtain *x* again will take years. The value returned by a hash function is often called a hash, message digest, hash value, or checksum. Most of the time a hash function will produce unique output for a given input. However depending on the algorithm, there is a possibility to find a collision due to the mathematical theory behind these functions.



Now suppose you want to hash the string "Hello Word" with the SHA1 Function, the result is `0a4d55a8d778e5022fab701977c5d840bbc486d0`.



Hash functions are used inside some cryptographic algorithms, in digital signatures, message authentication codes, manipulation detection, fingerprints, checksums (message integrity check), hash tables, password storage and much more. As a Python programmer you may need these functions to check for duplicate data or files, to check data integrity when you transmit information over a network, to securely store passwords in databases, or maybe some work related to cryptography.

I want to make clear that hash functions are not a cryptographic protocol, they do not encrypt or decrypt information, but they are a fundamental part of many cryptographic protocols and tools.

Some of the most used hash functions are:

- **MD5**: Message digest algorithm producing a 128 bit hash value. This is widely used to check data integrity. It is not suitable for use in other fields due to the security vulnerabilities of MD5.
- **SHA:** Group of algorithms designed by the U.S's NSA that are part of the U.S Federal Information processing standard. These algorithms are used widely in several cryptographic applications. The message length ranges from 160 bits to 512 bits.

The `hashlib`[2] module, included in The Python Standard library is a module containing an interface to the most popular hashing algorithms. `hashlib` implements some of the algorithms, however if you have OpenSSL installed, `hashlib` is able to use this algorithms as well.

This code is made to work in Python 3.2 and above. If you want to run this examples in Python 2.x, just remove the `algorithms_available` and `algorithms_guaranteed` calls.

First, import the `hashlib` module:

```
1 import hashlib
```

Now we use `algorithms_available` or `algorithms_guaranteed` to list the algorithms available.

```
1 print(hashlib.algorithms_available)
2 print(hashlib.algorithms_guaranteed)
```

The `algorithms_available` method lists all the algorithms available in the system, including the ones available trough OpenSSl. In this case you may see duplicate names in the list. `algorithms_guaranteed` only lists the algorithms present in the module. `md5`, `sha1`, `sha224`, `sha256`, `sha384`, `sha512` are always present.

MD5

```
1 import hashlib
2 hash_object = hashlib.md5(b'Hello World')
3 print(hash_object.hexdigest())
```

The code above takes the "Hello World" string and prints the HEX digest of that string. `hexdigest` returns a HEX string representing the hash, in case you need the sequence of bytes you should use digest instead. It is important to note the "b" preceding the string literal, this converts the string to bytes, because the hashing function only takes a sequence of bytes as a parameter. In previous versions of the library, it used to take a string literal. So, if you need to take some input from the console, and hash this input, do not forget to encode the string in a sequence of bytes:

```
1 import hashlib
2 mystring = input('Enter String to hash: ')
3 # Assumes the default UTF-8
4 hash_object = hashlib.md5(mystring.encode())
5 print(hash_object.hexdigest())
```

SHA1

```
1 import hashlib
2 hash_object = hashlib.sha1(b'Hello World')
3 hex_dig = hash_object.hexdigest()
4 print(hex_dig)
```

SHA224

```
1 import hashlib
2 hash_object = hashlib.sha224(b'Hello World')
3 hex_dig = hash_object.hexdigest()
4 print(hex_dig)
```

SHA256

```
1 import hashlib
2 hash_object = hashlib.sha256(b'Hello World')
3 hex_dig = hash_object.hexdigest()
4 print(hex_dig)
```

SHA384

```
1 import hashlib
2 hash_object = hashlib.sha384(b'Hello World')
3 hex_dig = hash_object.hexdigest()
4 print(hex_dig)
```

SHA512

```
1 import hashlib
2 hash_object = hashlib.sha512(b'Hello World')
3 hex_dig = hash_object.hexdigest()
4 print(hex_dig)
```

Using OpenSSL Algorithms

Now suppose you need an algorithm provided by OpenSSL. Using `algorithms_available`, we can find the name of the algorithm you want to use. In this case, "DSA" is available on my computer. You can then use the `new` and `update` methods:

```
1 import hashlib
2 hash_object = hashlib.new('DSA')
3 hash_object.update(b'Hello World')
4 print(hash_object.hexdigest())
```

Practical example: hashing passwords

In the following example we are hashing a password in order to store it in a database. In this example we are using a salt. A salt is a random sequence added to the password string before using the hash function. The salt is used in order to prevent dictionary attacks and rainbow tables attacks. However, if you are making real world applications and working with users' passwords, make sure to be updated about the latest vulnerabilities in this field.

Python 2.x

```
import uuid
import hashlib

def hash_password(password):
    # uuid is used to generate a random number
    salt = uuid.uuid4().hex
    return hashlib.sha256(salt.encode() + password.encode()).hexdigest() + ':' + salt

def check_password(hashed_password, user_password):
    password, salt = hashed_password.split(':')
    return password == hashlib.sha256(salt.encode() + user_password.encode()).hexdigest()

new_pass = input('Please enter a password: ')
hashed_password = hash_password(new_pass)
print('The string to store in the db is: ' + hashed_password)
old_pass = input('Now please enter the password again to check: ')
if check_password(hashed_password, old_pass):
    print('You entered the right password')
else:
    print('I am sorry but the password does not match')
```