FALMOUTH
UNIVERSITY

Lecture 11: Cryptography

COMP260: Distributed Systems
BSc (Hons) Computing for Games

- Today's session:
  - Linux server workshop review
  - Why use encryption
  - Simple Encryption
  - Hard Encryption
  - Assignment 2 Review
  - Workshop

- Linux server workshop review
  - Are you all comfortable with remote command line linux development and operation (devops)?

- Linux server workshop review
  - Are you all comfortable with remote command line linux development and operation (devops)?
    - Git client is installed on server
      - Git pull instead of ftp client (filezilla)
    - Can run httpserver from your accounts with a different port to your socket server
      - See httpserver from COMP130 last year
    - Can host web content if you want
      - Symlink from your home/<user> folders

- Why use encryption

- Why use encryption
  - As we saw in week 5
    - Unencrypted data enables passive hacking attacks
      - 3rd parties can read your data as a byte steam
      - Even using packet formats, it is still effectively a byte stream

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	00	01	00	00	00	FF	FF	FF	FF	01	00	00	00	00	00	00	.....ÿÿÿÿ.....
00000010	00	0C	02	00	00	00	46	41	73	73	65	6D	62	6C	79	2D	......FAssembly-
00000020	43	53	68	61	72	70	2C	20	56	65	72	73	69	6F	6E	3D	CSharp, Version=
00000030	30	2E	30	2E	30	2E	30	2C	20	43	75	6C	74	75	72	65	0.0.0.0, Culture
00000040	3D	6E	65	75	74	72	61	6C	2C	20	50	75	62	6C	69	63	=neutral, Public
00000050	4B	65	79	54	6F	6B	65	6E	3D	6E	75	6C	6C	07	01	00	KeyToken=null...
00000060	00	00	00	01	00	00	00	45	00	00	00	04	09	44	61	74	.......E.....Dat
00000070	61	45	6E	74	72	79	02	00	00	00	0D	27	09	03	00	00	aEntry.....'....
00000080	00	09	04	00	00	00	09	05	00	00	00	09	06	00	00	00	....
00000090	09	07	00	00	00	09	08	00	00	00	09	09	00	00	00	09	....
000000A0	0A	00	00	00	09	0B	00	00	00	09	0C	00	00	00	09	0D	....
000000B0	00	00	00	09	0E	00	00	00	09	0F	00	00	00	09	10	00	....
000000C0	00	00	09	11	00	00	00	09	12	00	00	00	09	13	00	00	....

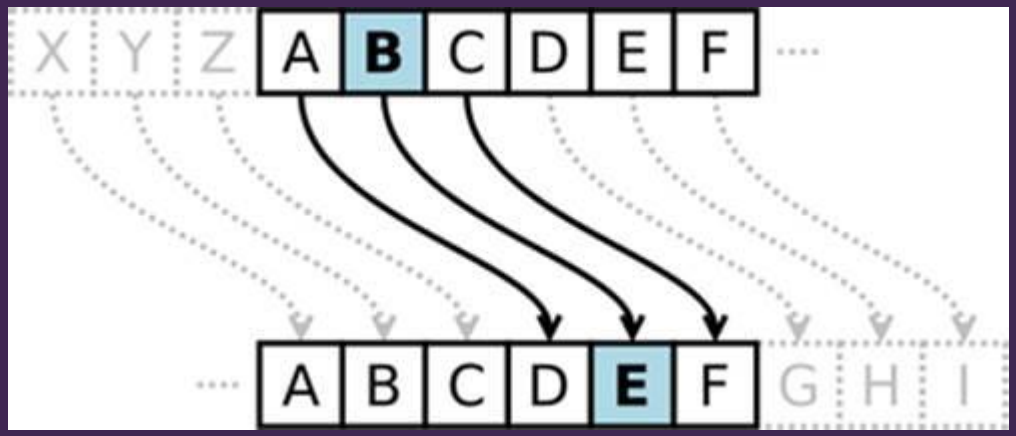
- Why use encryption
  - As a service provider, we don't want people to read our secrets
    - Regardless of how trivial they are
  - Black boxes provide a lot of security
  - Any security is better than none
    - Often hackers will look for better *low-hanging fruit*

- Simple Encryption

- Simple Encryption
  - At its simplest level, we want to have an approach to encryption that will move data away from being in a plain-text format

- utf-8
  - 8-bit Unicode format
  - 1-4 bytes of data per character to encode all known alphabets
  - 1-byte for English text (no accents)
  - 2-4 bytes for everything else
  - Extends original ASCII format (1 byte) left

- Simple Encryption
  - Traditional replacement and swizzling approaches
  - Caesar's cypher (static offset)
    - https://inventwithpython.com/chapter14.html
    - Often quoted as a simple approach that is cryptographically valueless
    - Will turn text into garbage to stop casual inspection



- Simple Encryption
  - Traditional replacement and swizzling approaches
  - Caesar's cypher (static offset)
    - Very easy to brute-force text with a dictionary
    - Can solve with visual inspection, given ordering of letter and letter frequency
    - Can be made more interesting with dynamic offsets

- Simple Encryption
  - Traditional replacement and swizzling approaches
  - XOR approaches
    - https://www.geeksforgeeks.org/xor-cipher/
    - Bit-swizzling / twiddling with the XOR or ^ operator

```
// The same function is used to encrypt and
// decrypt
void encryptDecrypt(char inpString[])
{
    // Define XOR key
    // Any character value will work
    char xorKey = 'P';

    // calculate length of input string
    int len = strlen(inpString);

    // perform XOR operation of key
    // with every caracter in string
    for (int i = 0; i < len; i++)
    {
        inpString[i] = inpString[i] ^ xorKey;
        printf("%c",inpString[i]);
    }
}
```

Output:

```
Encrypted String: 55;#6?"55;#
Decrypted String: GeeksforGeeks
```

- Simple Encryption
  - Traditional replacement and swizzling approaches
  - XOR approaches
    - Again, extremely easy to brute-force due to static look up from A->B

- Simple Encryption
  - Traditional replacement and swizzling approaches
  - Data compression
    - https://docs.python.org/3/library/zlib.html
    - A side effect of compressing data is that it makes it hard to read
    - Helps if data payload is quite large for this, often compressors will dump plain data if compression is larger than raw data
      - Due to lookup table overheads

- Simple Encryption
  - Traditional replacement and swizzling approaches
  - Data compression
    - Often compressed data will have clear footprints (headers / sections) making it obvious that the data is compressed
      - Can XOR to get round this ;)
      - Or strip some header parts
      - Or move data around in payload :)

- Simple Encryption
  - Traditional replacement and swizzling approaches
  - Although simple approaches are very bad on their own, they can create value when combined
  - Often, data that cannot be easily decrypted is left alone for data that can
    - Unless the data has potential value

- Hard Encryption

- Hard Encryption
  - Python has lots of cryptography libraries and functionality
    - From its widespread us as a backend service
    - And the wide availability of lots of cryptography technologies

- Hard Encryption
  - `Import cryptography.Fernet`
    - Fernet is a key-based symmetric encryption method
    - Part of cryptography package

- Hard Encryption
  - 01.fernet
    - Uses a shared key between encrypt and decrypt

```
import cryptography

from cryptography.fernet import Fernet
key = Fernet.generate_key()
cipher_suite = Fernet(key)
cipher_text = cipher_suite.encrypt(b"A really secret message. Not for prying eyes.")
plain_text = cipher_suite.decrypt(cipher_text)

print(plain_text)
```

    - Can use password salt as the key
      - As it's sent to the client anyway
      - Unique per client so fair secure
      - Fernet.generatekey() creates a longer key than the md5 examples the other week

- Hard Encryption
  - 02.cryptodome
    - Cryptodome is a rework of an unsupported crypto package
      - Import into project
    - AES cipher is more complex than Fernet
      - ‘Advanced Encryption Standard’
        - » Our Rijndael-based approach for assignment
      - But still relies on shared keys

- Hard Encryption
  - 02.cryptodome

```
import json
import Crypto
from base64 import b64encode
from base64 import b64decode

from Crypto.Cipher import AES
from Crypto.Util.Padding import pad
from Crypto.Util.Padding import unpad
from Crypto.Random import get_random_bytes


data = b"secret"
key = get_random_bytes(16)
cipher = AES.new(key, AES.MODE_CBC)
ct_bytes = cipher.encrypt(pad(data, AES.block_size))
iv = b64encode(cipher.iv).decode('utf-8')
ct = b64encode(ct_bytes).decode('utf-8')
result = json.dumps({'iv':iv, 'ciphertext':ct})
print(result)

# We assume that the key was securely shared beforehand
try:
    b64 = json.loads(result)
    iv = b64decode(b64['iv'])
    ct = b64decode(b64['ciphertext'])
    cipher = AES.new(key, AES.MODE_CBC, iv)
    pt = unpad(cipher.decrypt(ct), AES.block_size)
    print("The message was: ", pt)
except Exception:
    print("Incorrect decryption")
```

Sending side

- Key is a shared 16 byte random byte array (from [password salt)
- Is used to generate ciphers
- Send the cipher encrypted data and initialisation vector (IV) to the recipient as json

- Hard Encryption
  - 02.cryptodome

```
import json
import Crypto
from base64 import b64encode
from base64 import b64decode

from Crypto.Cipher import AES
from Crypto.Util.Padding import pad
from Crypto.Util.Padding import unpad
from Crypto.Random import get_random_bytes


data = b"secret"
key = get_random_bytes(16)
cipher = AES.new(key, AES.MODE_CBC)
ct_bytes = cipher.encrypt(pad(data, AES.block_size))
iv = b64encode(cipher.iv).decode('utf-8')
ct = b64encode(ct_bytes).decode('utf-8')
result = json.dumps({'iv':iv, 'ciphertext':ct})
print(result)

# We assume that the key was securely shared beforehand
try:
    b64 = json.loads(result)
    iv = b64decode(b64['iv'])
    ct = b64decode(b64['ciphertext'])
    cipher = AES.new(key, AES.MODE_CBC, iv)
    pt = unpad(cipher.decrypt(ct), AES.block_size)
    print("The message was: ", pt)
except Exception:
    print("Incorrect decryption")
```

Receive side

- Key is a shared 16 byte random byte array (salt)
- Generate a new cipher using the shared key and IV
- Decrypt the data to reveal its true meaning

- Hard Encryption
  - 02.cryptodome

```
import json
import Crypto
from base64 import b64encode
from base64 import b64decode

from Crypto.Cipher import AES
from Crypto.Util.Padding import pad
from Crypto.Util.Padding import unpad
from Crypto.Random import get_random_bytes


data = b"secret"
key = get_random_bytes(16)
cipher = AES.new(key, AES.MODE_CBC)
ct_bytes = cipher.encrypt(pad(data, AES.block_size))
iv = b64encode(cipher.iv).decode('utf-8')
ct = b64encode(ct_bytes).decode('utf-8')
result = json.dumps({'iv':iv, 'ciphertext':ct})
print(result)

# We assume that the key was securely shared beforehand
try:
     b64 = json.loads(result)
     iv = b64decode(b64['iv'])
     ct = b64decode(b64['ciphertext'])
     cipher = AES.new(key, AES.MODE_CBC, iv)
     pt = unpad(cipher.decrypt(ct), AES.block_size)
     print("The message was: ", pt)
except Exception:
     print("Incorrect decryption")
```

Receive side

- Key is a shared 16 byte random byte array (salt)
  - Key is not sent with data (obviously)
  - Apart from initial log on
    - To locally hash user password before sending to server for validation

- Hard Encryption
  - 02.cryptodome
    - Have more data to play with
      - Can use simple encryption techniques on top of this
      - And store data in interleaved formats to make it harder to unpack

- Hard Encryption
  - Packet management
    - Given we've looked at formats like:
      - <ID><LENGTH><DATA>
      - Makes it fairly easy to see where the data is
    - An approach of
      - <DATA.FIXED>
      - Makes this far harder to unpack
      - (and to write / test)

- Hard Encryption
  - Packet management
    - <DATA.FIXED>
      - To decrypt as cryptodome we need:
        - » IV
        - » Data
        - » Datasize.2
      - Also need to know it's 'our' data
        - » ID.4

- Hard Encryption
  - Packet management
    - Decompress <DATA.FIXED> into
      - <ID.4><cryptsize.2><IV><encrypted data>
        - » cryptsize = size of IV + data
    - Don’t need to store data in such an obvious format
      - Anything that makes it hard to unpack will scare hackers away
      - <ID.1><cryptsize.2><ID.1><IV><ID.1>< encrypted data ><ID.1>
      - Or split IV and encrypted data so they aren’t contiguous

FALMOUTH UNIVERSITY

• Assignment 2 Review

Marking Rubric

Criterion	Weight	Refer for Resubmission	Basic Proficiency	Novice Competency	Novice Proficiency	Professional Competency	Professional Proficiency
Threshold	40%	At least one part is missing or is unsatisfactory.	Submission is timely. Enough work is available to hold a meaningful discussion. Provided a meaningful review of a peer's work. Clear evidence of programming knowledge and communication skills. Clear evidence of use of appropriate version control techniques, including regular commits and some use of branching. No breaches of academic integrity. Server has been developed in Python				
Remote Service	10%	Client and server cannot communicate Server is not hosted remotely on Digital Ocean droplet	Server is hosted on DO droplet and requires multiple restarts during the viva	Server is hosted on DO droplet and requires a single restart during the viva	Server is successfully hosted on DO droplet and runs without the need to reset it during the viva	Server is successfully hosted on DO droplet and runs without the need to reset it during the viva Server can support multiple clients from different IP addresses	Server is successfully hosted on DO droplet and runs without the need to reset it during the viva Server can support multiple clients from different domains
User Security	10%	Application has no user / account security Username + password is not required to play MUD	Users can log on during viva regardless of credentials	Users can successfully log in during viva Users cannot log in with incorrect credentials	Users can successfully log in during viva Users cannot log in with incorrect credentials A user cannot log on multiple times	Users can successfully log in during viva Users cannot log in with incorrect credentials A user cannot log on multiple times New accounts can be created from client	Users can successfully log in during viva Users cannot log in with incorrect credentials A user cannot log on multiple times New accounts can be created from client & validated
Communication Security	10%	Client server communication are implemented as unencrypted byte streams that can be cast to strings and read	MUD service uses a trivial approach to encrypt packet data	MUD service uses a weak approach to communication security	MUD service uses a strong approach to communication security, such as AES/ Rijndael	MUD service uses a strong approach to communication security, such as AES/ Rijndael, and adds packet sequencing to stop packet replay hacking	MUD service uses multiple strong approaches from security wiki that go beyond typical industry standards
Data Persistence	30%	Server does not use relational database to manage persistent game data or client data On returning to the game, players restart at the 'beginning' of the dungeon rather than where they last were	User data is stored in server-side SQL database Player data is stored in server-side SQL database	Implementation of developer-defined persistent features that work badly e.g.: -last log-on details -last attempted log-on -returning player information / MotD -message service for off-line players -ownership/transference of game objects	Implementation of developer-defined persistent features that work reasonably well	Implementation of developer-defined persistent features that work well	Implementation of developer-defined persistent features that work very well

- Workshop
  - This week:
    - Assignment support :)

- Questions