Computer Security
Assignment 1

University of Sussex
Candidate Number: 164574

Contents

1 Principles, User authentication, Malware **3**
1.1 Task 1 3
1.2 Task 2 4

2 Digital Forensics **6**
2.1 Task 3 6

3 Encryption **9**
3.1 Task 4 9
3.2 Task 5 14

4 Challenge **16**
4.1 Task 6 16
4.2 Task 7 18
4.3 Task 8 27

1 Principles, User authentication, Malware

1.1 Task 1

1. cr@zyp@ss

   - The password cr@zyp@ss is an unsuitable password for computer security. The suitability of the password is that it includes nine characters in total and two of these characters are special characters. The unsuitability of the password is that it contains no upper case characters and no numbers. The special characters are a common replacement for the character 'a' and when replaced leaves two plain text dictionary words. The special characters provide minimal security and is prone to dictionary attacks and brute force.

2. qwerty

   - The password qwerty is an unsuitable password for computer security. The password is extremely short only being 5 characters long without any upper case characters, special characters or numbers. The word 'qwerty' is a word in the dictionary and would be highly likely to be used in a dictionary attack and will be included in common passwords dictionaries and blacklist passwords. Due to its simplicity and frequency used as a password, it would be highly vulnerable to brute force attacks, common password attacks and dictionary attacks, therefore making it an extremely unsuitable password.

3. *laptop_admin#

   - The password *laptop_admin# is an unsuitable password for computer security. The password is a good length at 14 characters long and has three unique special characters. The password however includes two plain text dictionary words and the words laptop_admin will likely be included in common passwords. This password would be vulnerable to sophisticated attacks which include multiple attacks as the complexity of the password with multiple special characters would be secure against brute force attacks but would be more susceptible to dictionary and common password attacks.

4. KVK919

   (a) The password KVK919 is an unsuitable password for computer security. The password is intermediate in length as it is only six characters long however the password does not include and special characters. The password includes a mixture of upper case letters and numbers however it only includes 4 unique characters as the letter 'K' and the number '9' are repeated. This makes the password vulnerable to brute force attacks due to its length, repeated characters and lack of complexity.

1.2 Task 2

- signup.aspx.cs
  - string NewPassword;
    - Passwords should never be stored as a string in C# and the majority of Object Oriented programming languages. This is because an instance of the System.String class is both immutable and, when no longer needed, cannot be programmatically scheduled for garbage collection. This means it is not possible to predict when the instance will be deleted from computer memory. There is a risk the password could be revealed after it is used because the application cannot delete the data from computer memory [4].
      - A method to make the password policy more secure for the C# application is to use the SecureString Class. This represents text that should be kept confidential by deleting it from computer memory when it is no longer needed. This is more secure than String however SecureString may still be exposed to any process or operation that has access to raw memory [4]. The alternative is to use an opaque handle or third party services with strong encryption to handle authentication and authorisation.
  - GeneratePassword() and GetWord()
    - The two functions are the largest security flaw and biggest issue in regards to the password policy. The GeneratePassword() function concatenates two strings returned from GetWord(). GetWord() returns a random word from the passwords.txt file. This is an extremely weak password policy as the majority of words in the passwords.txt are English dictionary words, do not contain any uppercase or special characters and the shortest password is 1 character long. This is a weak password policy as all passwords are vulnerable to brute force attacks due to the lack of complexity of the passwords. If an attacker gained access to the passwords.txt file, every account would be compromised as it could be used in a dictionary attack.
      - A solution to increasing the strength of the password policy is to allow users to create their own passwords and ensure that these are more complex passwords. This includes a minimum requirement of length, having at least one upper case letter and at least one special character to make the password more secure. Another solution to increasing the strength of the password policy is to require users to change their passwords after a set duration of time.
      - A solution to encrypting these passwords is to use a one-way function to transform the password and a unique salt into

a hash. Specialised hashing algorithms should be used to prevent bad practices and vulnerabilities in the algorithms.

- changepassword.aspx.cs
  - ChangePassword(string EMail, string NewPW)
    - The function retrieves the user from the database only from their email passed through the parameter of the function through an SQL statement. The password is then updated if the user is found regardless of other credentials such as username and the previous password. If an attacker injected code into this method, they would have the ability to change any users password with only the knowledge of their email.
      - To make the password policy more secure for this issue, only authenticated users should have the ability to change their own password. Authenticated users should be prompted to re-enter their current password and this should be checked this matches with the password in the database before updating the new password.
- resend.aspx.cs
  - GetPassword(string EMail)
    - This function returns a password from the database based on the email string passed through the parameter. If an attacker was able to exploit the function, they would be able to retrieve the passwords for all the known emails.
      - To make the password policy more secure, the passwords that are retrieved should be hashed and not in plain text. The passwords should also not be stored as a string.

2 Digital Forensics

2.1 Task 3

For this task the coursework files used are surnames L to Z. To get the coursework files onto the e2c Kali instance, the GUI was used to download the files onto the remote system. For this task a step by step process is given to demonstrate the understanding of the solution to the problem. The output file for the password dictionary is included. A shell script 'commands.sh' is also included that contains all of the commands to complete the task. The script can be executed after connecting to the Kali instance, downloading the coursework files and logging in as the root user. To give the shell the permissions to execute the script, the following command is needed:

```
$ chmod 755 commands.sh
```

To execute the script, the following command is needed:

```
$ ./commands.sh
```

The question states that the PDF is protected by three English alphabet characters and my assumption is that the password includes both lowercase and uppercase alphabet characters as this is not specified in the brief.

The first step for completing the task is to connect to the e2c instance using an SSH client. As the task was completed on Ubuntu 18.04, the secure shell protocol was used in the terminal to securely log on to the remote e2c instance. The following command is a generic command taken from AWS documentation to SSH to the Linux instance:

```
$ ssh -i /path/my-key-pair.pem ec2-user@
ec2-198-51-100-1.compute-1.amazonaws.com
```

The next step for completing this task is to login to root user. This is achieved with the 'su -' command and this moves to the /root/ directory. For this project the task was completed in /root/Documents and the command to move to this directory is 'cd Documents/'.

The next step for completing this task is to generate a password dictionary. The program to generate the password dictionary is 'Crunch', which is a built in tool for Kali Linux. The following command generates the password dictionary using Crunch:

```
$ crunch 3 3 -f /usr/share/
rainbowcrack/charset.txt mixalpha
-o password-dict-mixalpha.txt
```

This command generates a password dictionary for all possible combinations of three English alphabet passwords. This is specified by the '3 3' as this is <min-length> and <max-length>. The -f flag specifies a character set from

charset.txt. The character set used is mixalpha which contains all lower case and uppercase English alphabet characters. The -o flag specifies the path to write the output to.

The next stage of this task is to install JohnTheRipper. This is achieved by cloning the GIT repository:

```
$ git clone git://github.com/magnumripper/
JohnTheRipper
```

After the repository has cloned it is necessary to build JohnTheRipper. This is achieved by changing to the src directory through the command 'cd JohnTheRipper/src/' and running the command:

```
$ ./configure && make -s clean && make -sj4
```

The next step for completing the task is to execute the file pdf2john.pl to extract the hash from the PDF. It is required to change to the run directory through the command 'cd ../run/' and executing the command to extract the hash:

```
$ ./pdf2john.pl /root/Documents/
'Group - SurnameStartsFromLtoZ'/
'PasswordProtectedFile[L to Z].pdf'
> /root/Documents/pdf.txt
```

This command extracts the hash from the PDF and writes the output to pdf.txt.

The next stage is to run the generated password dictionary password-dict-mixalpha.txt against pdf.txt. This is achieved by changing directory to Documents through the command 'cd ../../' and executing the command:

```
$ john --wordlist=/root/Documents/
password-dict-mixalpha.txt
/root/Documents/pdf.txt
```

If ran successfully and cracks the password, it displays the output:

```
Using default input encoding:
UTF-8
Loaded 1 password hash
(PDF [MD5 SHA2 RC4/AES 32/64])
No password hashes left to crack (see FAQ)
```

The final step to complete the task is to display the password. This is achieved through the command:

```
$ john --show --format=PDF pdf.txt
```

This will display the following output:

```
/root/Documents/Group -
SurnameStartsFromLtoZ/
PasswordProtectedFile[L to Z].pdf:zyx
1 password hash cracked, 0 left
```

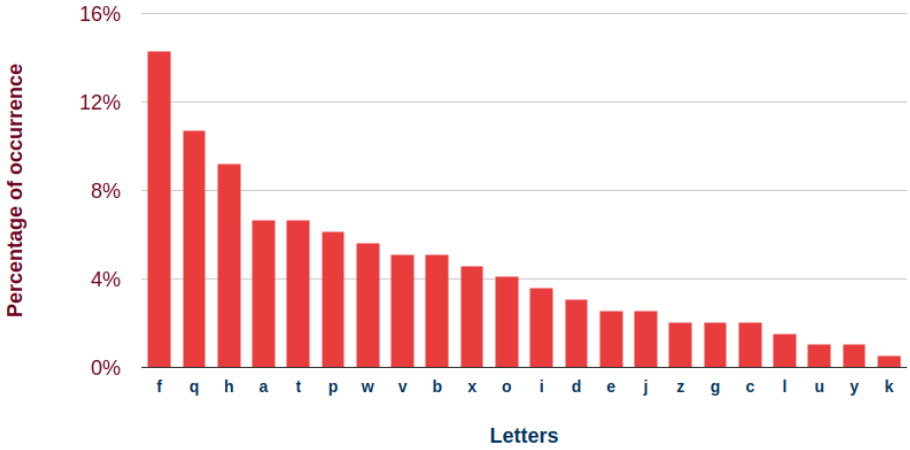
The password for PasswordProtectedFile[L to Z].pdf is zyx.

3 Encryption

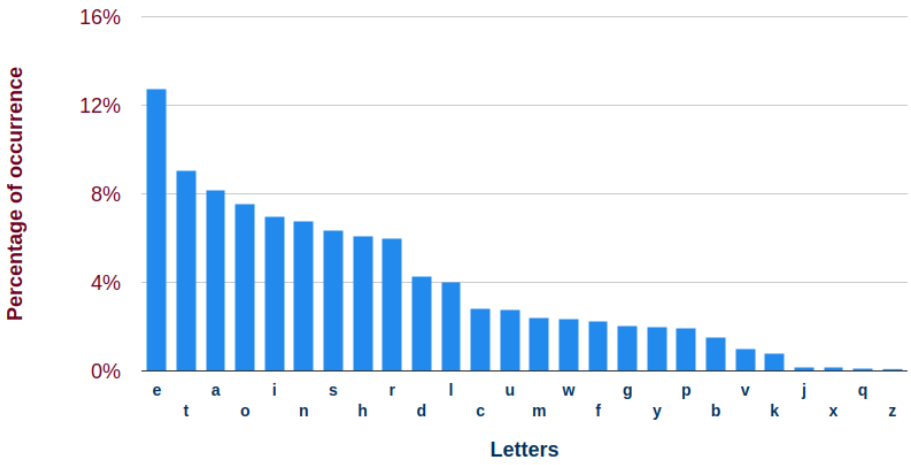
3.1 Task 4

The first step towards deciphering the cipher-text was to perform frequency analysis on the characters in the given cipher-text. This method makes use of the characteristic of any given stretch of written language where certain letters or combinations of letters occur with varying frequency [6]. For instance, letters A,E,I,S,T and R occur more frequently in a sufficiently long English text compared to letters J, X and Z. Likewise, TH, ER, ON, and AN are the most common pairs of letters (termed bigrams or digraphs), and SS, EE, TT, and FF are the most common repeats [1].

Trial and error is needed for frequency analysis, as the frequency distribution of letters in a ciphered message do not always follow the standard for English alphabet frequencies completely, especially for a short message. As the cipher-text is fairly short, the frequency values could be skewed.



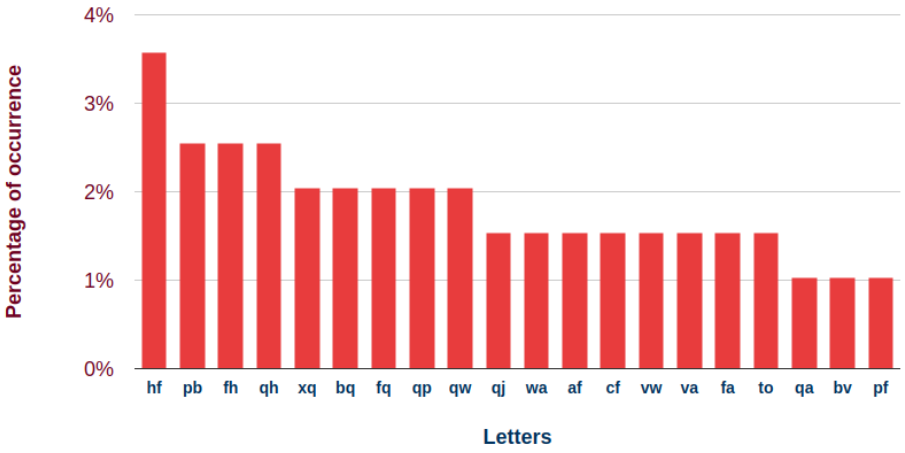
Frequency analysis of the letters in the cipher-text



Frequency analysis of the English Alphabet

When analysing the frequency analysis of the cipher-text compared to the frequency analysis of the English alphabet, it was evident that the letter 'F' in cipher-text matched the letter 'e' in the English alphabet. For convention, cipher-text letters will be represented as a capital letter denoted as 'A' and English alphabet letters will be represented as lower case letters denoted as 'a'.

The next stage to performing frequency analysis is to perform digraphs and trigraphs analysis on the text.



Frequency analysis of the digraphs in the cipher-text

The most common digraphs in the English language are:
TH, HE, AN, IN, ER, ON, RE, ED, ND, HA, AT, EN.

The most common digraphs in the cipher-text are:
HF, QH, FH, PB, BQ, QW, QP, FQ, XQ, CF, FA, QJ, TO.

The most common trigraphs in the English language are:
THE, AND, THA, ENT, ION, TIO, FOR, NDE, HAS, NCE, TIS, OFT, MEN.

The most common trigraphs in the cipher-text are:
PBQ, HFQ, PFH, HPB, PBF, BQW, BQP, TOG, OGH, GHF, FQA, XQV, QVA.

The current assumption is that the cipher-text letter 'F' is the English letter 'e'. When looking at the most common trigraphs in the cipher-text and the most common trigraphs in the English language, the triagraph 'THE' should be easy to solve with the assumption 'F' is equal to 'e'. This leaves the two trigraphs PBF and GHF potentially being the trigraph 'THE' as both cipher-text trigraphs end in the letter 'F'. When looking at the common diagraphs for the cipher text, 'PB' appears and the diagraph 'TH' is common in the English Language. The assumption is that 'PBF' stands for the word 'THE'.

With the assumption that 'PBF' stands for the word 'THE', the next common trigraph to solve with the majority of letters solved is 'PBQ'. A common trigraph in the English language is 'THA' and the assumption is 'PB' is 'TH' so 'PBQ' is likely to be 'THA'.

With the letters 'FQPB' assuming to be solved, the following trigrams that are missing only one character are 'FQA', 'HFQ', 'HPB', 'PFH', 'BQW'. As the letter 'H' appears twice with the trigraphs 'HFQ' and 'HPB', these were the obvious two trigraphs to solve. Two of the most digraphs in the cipher-text are 'HF' and 'FH' and when looking at the most common digraphs in the English language, 'ER' and 'RE' appear. Since 'e' is assumed to be 'F' in the cipher-text, the assumption can be made that cipher-text 'H' is equal to English alphabet 'r'.

With the letters 'FQHPB' assuming to be solved, the following trigraphs that are missing one character are 'BQW', 'GHF', 'FQA'. From looking at the most common digraphs in the cipher-text, QW appears. From the assumptions, 'Q' in cipher-text is 'a' in plain-text. The two common most digraphs in the English language that match 'QA' are 'AN' and 'AT'. As 'AN' is the more common digraph in the English language, the assumption is that cipher-text 'W' is English alphabet 'n'.

With the letters 'FQHPWB' assuming to be solved, the common digraph cipher-text 'FH' has been solved to be common English alphabet digraph 'ER'. As 'FA' is a common cipher-text digraph , the next common English alphabet digraph is 'ED'. As the letter 'A' has a high frequency in the cipher-text, it can be assumed that the cipher-text letter 'A' is equal to English alphabet character 'd'.

With the letters 'FQHAPWB' assuming to be solved, enough letters have been solved to try to analyse the partially solved cipher-text. The partially solved text 'Vndeed' appears which matches to 'VWAFFA' in the cipher-text. The partially solved text looks like the English word 'indeed' therefore the assumption cipher-text letter 'V' is equal to English letter 'i'.

With the letters 'FQHAPWBV' assuming to be solved, the only remaining high frequency cipher-text to not be solved is the letter 'T'. When looking at the high frequencies of the English Language, the letter 'o' has not yet been matched. If the cipher-text letter T was matched to English alphabet letter 'o', then TW would make the word 'on', which is one of the most common digraphs in the English Language. Based on the high frequencies of the digraph 'on' in the English language and the frequency of English letter 'o' and cipher-text 'T', the assumption is that cipher-text 'T' is equal to English alphabet 'o'.

With the letters 'FQHATPWBV' assuming to be solved, the top 9 most frequent cipher-text characters have now been solved. When looking at the partially solved cipher-text, the phrase 'thanKed' appears. This is very close to the English word 'thanked', so it is assumed that the cipher-text letter 'K' is the English alphabet character 'k'.

With the letters 'FQHATPWBVK' assuming to be solved, the partially solved cipher-text appears to be close to displaying many words. The first part of the partially solved cipher-text displays 'ZarCentertheIthanked'. It would appear this can be split and the most obvious split is 'theIthanked'. The cipher-text character 'I' can be substituted for the English alphabet character 'y' to display the plain text words 'they thanked'. When this is assumption is made, it leaves the partially solved text to display 'ZarCenter they thanked' as the word boundaries are known. This leaves the partially solved cipher-text to display the work or phrase 'ZarCenter'. An intuitive guess would be that this partially solved cipher-text is the English word 'carpenter' where cipher-text 'Z' is 'c' and cipher-text 'C' is 'p'.

With the letters 'FQHATPWBVIPCK' assuming to be solved, one of the most common trigraphs is 'TOG' and one of the most common digraphs is 'TO' and cipher-text letter 'O' is common. An assumption could be made that the cipher-text 'O' is equivalent to English alphabet letter 'f'. This is because the digraph 'TO' would become the English word 'of' which isn't a common digraph, but is a common word.

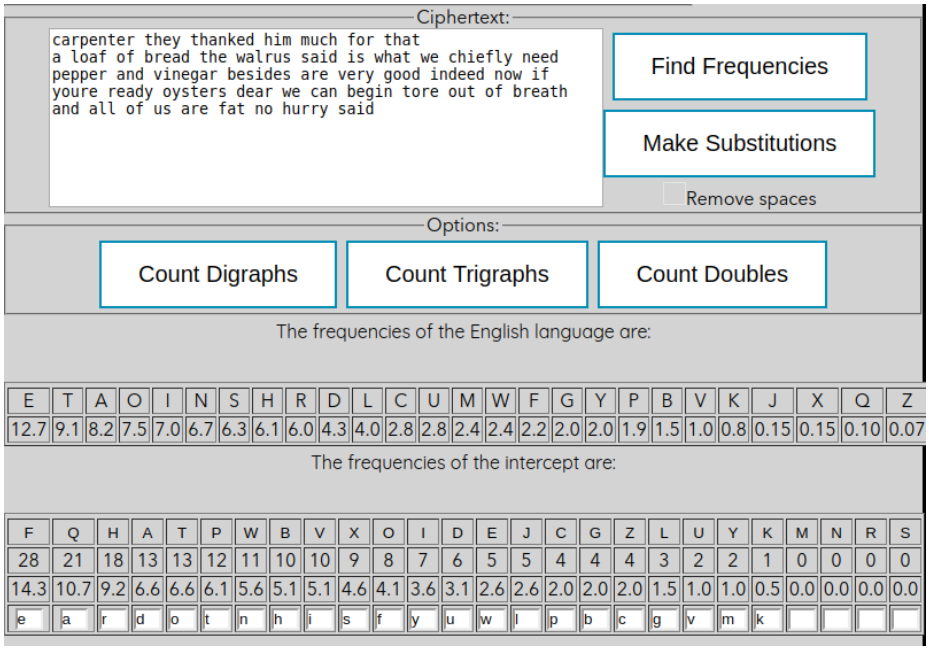
With the letters 'FQHATPWBVOICZK' assuming to be solved, the first line of the partially solved cipher-text displays 'carpenter they thanked hiYYDch for that'. As the word boundaries are known, this leaves the cipher-text 'hiYYDch' to be solved. When reading the sentence, it makes intuitive sense for the sentence to read 'carpenter they thanked him much for that'. This would make sense as the cipher-text letter 'Y' would equal 'm' which works with the sen-

tence. Based under this assumption, cipher-text letter 'Y' is equal to 'm' and cipher-text letter 'D' is equal to 'u'.

With the letters 'FQHATPWBVOIDCZYK' assuming to be solved, many words are starting to appear in the partially solved cipher-text. The phrase 'aJoafof-Gread' appears which looks like the sentence 'a load of bread' which would be the correct syntax in English. This would mean cipher-text 'J' is equal to letter 'l' and cipher-text 'G' would be equal to the letter 'b'. The phrase 'allofuXare-fat' appears towards the end of the partially solved cipher-text and this appears to look like the phrase 'all of us are fat'. The assumption is that cipher-text letter 'X' is the letter 's'.

With the letters 'FQHATPWBVXOIDCGZYK' assuming to be solved, the partially solved cipher-text is now solvable. The phrase 'EecanbeLin' can be assumed to be the sentence 'we can begin' which would mean cipher-text 'E' is 'w' and 'L' is 'g'. The only unsolved text is 'Uinegar', which appears to be 'vinegar'. The assumption is that cipher-text 'U' is 'v'.

The cipher-text has been decrypted and reads:
carpenter they thanked him much for that a loaf of bread the walrus said is what we chiefly need pepper and vinegar besides are very good indeed now if youre ready oysters dear we can begin tore out of breath and all of us are fat no hurry said



Online tool to display frequencies and manually add substitutions

3.2 Task 5

- Choose two distinct primes p and q of approximately equal size so that their product $n = pq$ is of the required bit length.

$$\begin{aligned} p &= 13 \\ q &= 31 \\ n &= 403 \end{aligned}$$

- Compute $\phi(n) = (p-1)(q-1)$.

$$\begin{aligned} p - 1 &= 12 \\ q - 1 &= 30 \\ \phi(n) &= 360 \end{aligned}$$

- Choose a public exponent $e, 1 < e < \phi(n)$, which is co-prime to $\phi(n)$, that is, $gcd(e, \phi(n)) = 1$.

$$gcd(e, \phi(n)) = gcd(19, 360) = 1$$

- Compute a private exponent d that satisfies the congruence $ed \equiv 1 \ (mod \ \phi(n))$.
  - Suitable candidate for $1 \ (mod \ \phi(n)) = 361 = 19 \cdot 19$

$$\begin{aligned} d &= 19 \\ e &= 19 \\ n &= 403 \\ \phi(n) &= 360 \\ e \cdot d &= 361 \\ e \cdot d \ mod \ \phi(n) &= 1 \end{aligned}$$

  - e and $\phi(n)$ are relatively prime
  - d and $\phi(n)$ are relatively prime
- Make the public key (n, e) available to others. Keep the private values d, p, q, and $\phi(n)$ secret.
- Encryption: cipher text, $c = RsaPublic\,(m) = m^e \ mod \ n$, where $1 < m < n - 1$.

$$\begin{aligned} m &= 2 \\ 2^{19} \ mod \ 403 &= 388 \\ c &= 388 \end{aligned}$$

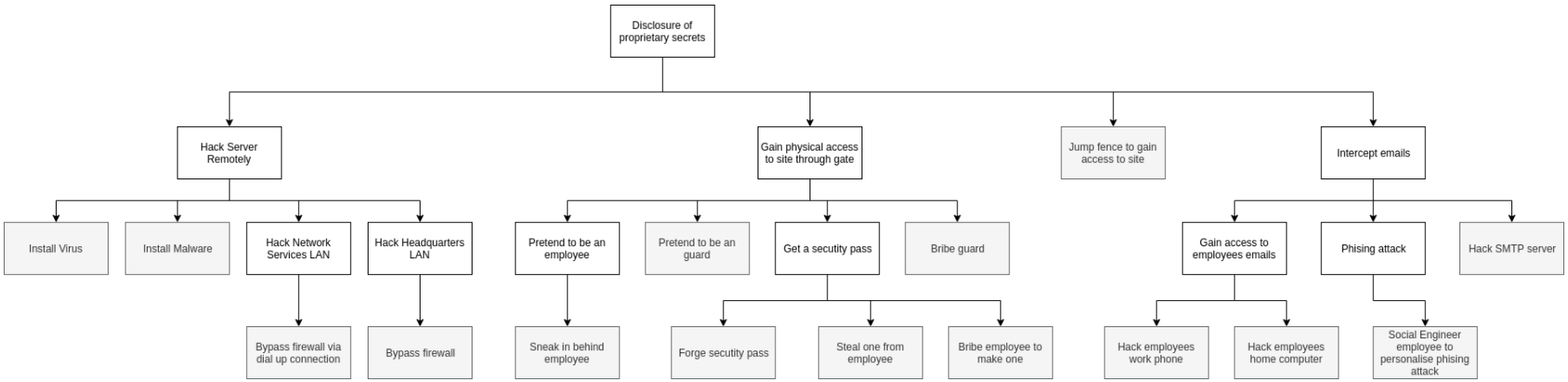
- Decryption: plain text, $m = RsaPrivate\,(c) = c^d \; mod \; n$.

$$\begin{aligned} c &= 388 \\ 388^{19} \; mod \; 403 &= 2 \\ m &= 2 \end{aligned}$$

4 Challenge

4.1 Task 6

Attack tree is displayed on the next page.


Disclosure of proprietary secrets
Hack Server Remotely
Gain physical access to site through gate
Jump fence to gain access to site
Intercept emails
Install Virus
Install Malware
Hack Network Services LAN
Hack Headquarters LAN
Pretend to be an employee
Pretend to be an guard
Get a secutity pass
Bribe guard
Gain access to employees emails
Phising attack
Hack SMTP server
Bypass firewall via dial up connection
Bypass firewall
Sneak in behind employee
Forge secutity pass
Steal one from employee
Bribe employee to make one
Hack employees work phone
Hack employees home computer
Social Engineer employee to personalise phising attack

4.2 Task 7

The project was completed in PHP using XAMPP installation package for Linux. The folder submitted is the htdocs directory with all of the source code. The XAMPP directory and configuration has not been submitted due to it exceeding the university submission file size when zipped. In this report, screenshots of the three features will be provided to demonstrate its functionality. The implementation of security features to ensure the system is safe and can't be exploited will be included with snippets of code to demonstrate the understanding of the task. The uploads directory has not been submitted as it is not needed. The PHP files included are to provide evidence of completion of the task and not to provide a working web application.

Assumptions:

- Users do not need to change their password as it is not specified in the brief.
- Users do not need the functionality to retrieve the file they uploaded as it is not specified in the brief.
- Users do not need the functionality to delete the file they uploaded as it is not specified in the brief.
- Users can only submit one PDF or MS Word format CV and this must be safe and not exploited.
- Two different users can upload the same file or file name.

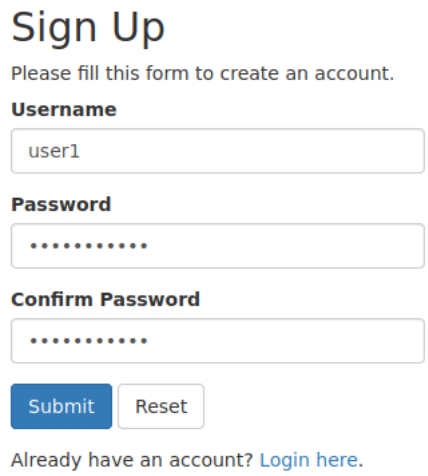
SQL Table Set Up

The following query is executed in the MySQL database to generate the table users.

```
CREATE TABLE users (
    id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(50) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    file_uploaded TINYINT DEFAULT 0,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

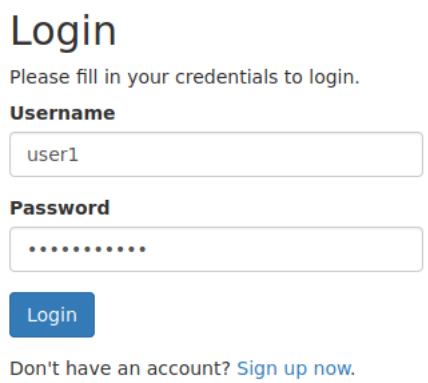
Screenshots of functionality

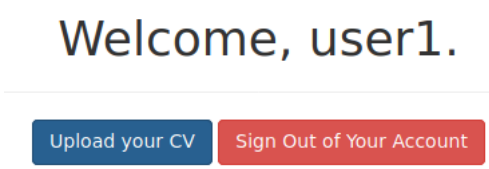
User Registration: The user registration includes functionality of a minimum password length of 6 characters and has a confirm password option to ensure the user inserted their password correctly. The screenshots provide evidence that the user registration works with the entry in the database.



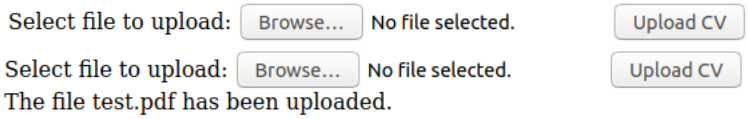
id	username	password	file_uploaded	created_at
1	user1	$2y$10$OzwzMIvTYVynK/RUSsQCLeCQ0b3Wvx4xsuHA6PV2M2i...	0	2019-12-11 17:02:35

User Login: The user login allows a user to login with their username and password. Once the user has logged in, the user is taken to a welcome page where they are given the option to upload their CV or logout.





CV Submission: The user can upload their CV on the upload page. The user can only upload one document which has to be either a pdf or word document. Once it is uploaded, it is stored in a uploads/ directory in the project directory in htdocs.




```
samuel@samuel-GE62-2QF:/opt/lampp/htdocs/computer-security-task-7/uploads$ ls 1t
est.pdf
1test.pdf
```

Security Exploits and Implementations of the web application

Uploaded files represent a significant risk to web applications as it is a method for attackers to get code to the system it wishes to attack. The consequences of unrestricted file upload can vary, including complete system takeover, an overloaded file system or database, forwarding attacks to back-end systems, client-side attacks, or simple defacement [3].

The following are security features that have been implemented into the web application to make it safe to use and not be exploited:

- Only authenticated users can access the web application.
  - It is important to only allow authenticated users to access the web application. This is to prevent unauthorised users from exploiting the web application.

```
// Check if the user is logged in, if not then redirect
    the user to login page
if(!isset($_SESSION["loggedin"]) || $_SESSION["loggedin"]
    !== true){
    header("location: login.php");
    exit;
}
```

  - This code snippet is used for all pages which require user authentication. It will only allow the user to access the page if they are logged in.
  - The security of this feature is storing session ID and session variables which are not included in the URL. When the user is authenticated, the session variable is set to true when the user has been verified in the login.php file.
  - Attackers will attempt try to gain access to user credentials to authenticate themselves onto the system and other ways to make this more secure would be:
    - Multi-factor authentication for when the user logs in
    - Weak-password checks for when the user registers
    - Password rotation policy
    - Failed login attempts limit
    - Log all failures and alert administrators

- A strong password policy for the web application.
  - A password policy is a set of rules designed to enhance computer security by encouraging users to employ strong passwords and use them properly.

```
if(strlen(trim($_POST["password"])) < 6){
    $password_err = "Password must have atleast 6 characters.";
}
```

  - This code snippet forces users to have a password of at least 6 characters. This is an example of a password policy to ensure that users do not enter in empty passwords or have short passwords.
  - A stronger password policy would include:
    * the use of both upper-case and lower-case letters (case sensitivity)
    * inclusion of one or more numerical digits
    * inclusion of special characters, such as @, #, $
    * prohibition of words found in a password blacklist
    * prohibition of words found in the user's personal information
    * prohibition of use of company name or an abbreviation
    * prohibition of passwords that match the format of calendar dates, license plate numbers, telephone numbers, or other common numbers

- Passwords must be hashed and secure in the database.
  - It is vital to not leave passwords in plain text because passwords could be intercepted through transmission or could be easily read by attackers if the database was breached. Hashing a password allows the password to be stored in the database, and when a unique salt is added to the hash, no two passwords' hash can be the same.

```
// Creates a password hash
$param_password = password_hash($password, PASSWORD_DEFAULT);
```

  - This code snippet from register.php uses the password_hash() function. The function creates a new password hash using a strong one-way hashing algorithm.
  - The first parameter is the password set by the user and the second parameter is PASSWORD_DEFAULT which uses the bcrypt algorithm.
  - The function password_verify (string $password , string $hash) is used when a user wants to login to the system. This function verifies if a password matches a hash.
  - These functions allow the user to have their data stored securely in the database and allow user verification when logging in without any exploits.

- Safe from SQL injections.
  - SQL injection is a type of attack that can give an attacker complete control over the web application database by inserting arbitrary SQL code into a database query. The most common injection is injecting SQL through user input [5].

```
$sql = "SELECT file_uploaded FROM users WHERE id = ?";
if($stmt = mysqli_prepare($link, $sql)){
    // Bind variables to the prepared statement as
        parameters
    mysqli_stmt_bind_param($stmt, "i", intval($idStripped
        ));
    // Attempt to execute the prepared statement
    if(mysqli_stmt_execute($stmt)){
        // Store result
        mysqli_stmt_store_result($stmt);
        // Check if id exists
        if(mysqli_stmt_num_rows($stmt) == 1){
            // Bind result variables
            mysqli_stmt_bind_result($stmt, $file_uploaded
                );
            if(mysqli_stmt_fetch($stmt)) {
                /* Update session variable file_uploaded
                    with
                  value from database */
                $_SESSION["file_uploaded"] =
                    $file_uploaded;
            }
        }
        // Close statement
        mysqli_stmt_close($stmt);
    }
}
```

  - This code snippet uses prepared statements and parameterised queries and this is prevalent throughout the entire source code for the web application. The prepared statement execution consists of two stages: prepare and execute. At the prepare stage a statement template is sent to the database server. The server performs a syntax check and initializes server internal resources for later use. The MySQL server supports using anonymous, positional placeholder with ?. Prepare is followed by execute. During execute the client binds parameter values and sends them to the server. The server creates a statement from the statement template and the bound values to execute it using the previously created internal resources [2].

- File must be less than a certain size.
  - Limiting the file size is a security measure to ensure that the system can't exploited. Unlimited file size can lead to DDoS attacks, as it will consume disk space on the server, consume network bandwidth and consume TCP sockets. This would allow the server to not handle any requests from other users if it is exploited.

```
// Check file size
if ($_FILES["fileToUpload"]["size"] > 500000) {
    echo "Sorry, your file is too large. <br>";
    $uploadOk = 0;
}
```

  - This code snippet makes sure the size is under 5mb to prevent the system from being exploited.

- File must be of the correct type.
  - To adhere to the brief, files must be uploaded of the correct type. Allowing any file type to be uploaded could lead to security vulnerabilities as the file could potentially be a virus, malware or try to gain admin privileges to the server.
  - Limiting the file type denies users from being able to exploit the system with dangerous files.

```
// Allow .pdf, .doc and .docx file formats
if($imageFileType !== "pdf" && $imageFileType !== "doc"
    && $imageFileType !== "docx") {
     echo "Sorry, only pdf, doc & docx file formats are
         allowed. <br>";
     $uploadOk = 0;
}
```

  - This code snippet ensures that the user uploads a file of the type pdf, doc and docx.

4.3 Task 8

Task 8 has not been attempted.

References

[1] The black chamber - hints and tips.

[2] Prepared statements - manual.

[3] Unrestricted file upload.

[4] Dotnet-Bot. Securestring class (system.security).

[5] William G Halfond, Jeremy Viegas, Alessandro Orso, et al. A classification of sql-injection attacks and countermeasures. In *Proceedings of the IEEE International Symposium on Secure Software Engineering*, volume 1, pages 13–15. IEEE, 2006.

[6] Douglas R Stinson. *Cryptography: theory and practice*. Chapman and Hall/CRC, 2005.