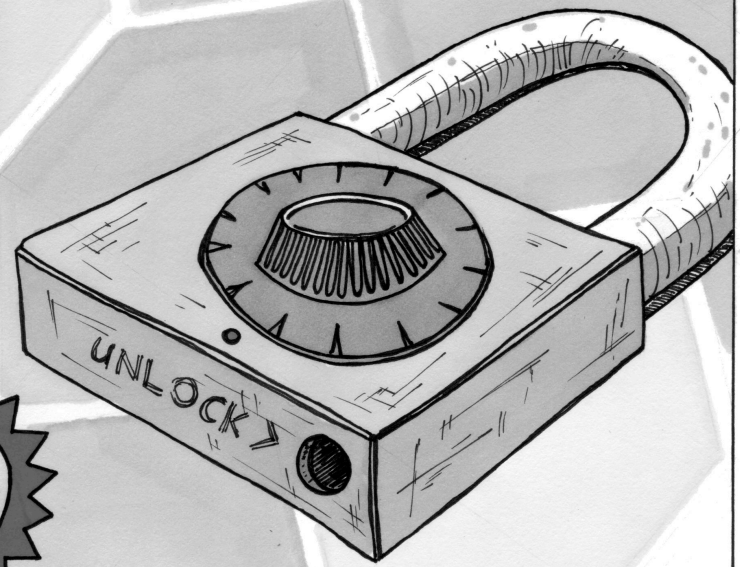
Tess Sluijter, 2019

Things I've Learned

#1

Oct 2019
T. Sluijter

The tale of the DUBIOUS CRYPTO



#TIL

You'd store this in a file like "myclass.class" and compile it with "javac myclass.class". The result is a file "myclass", which you run as "java myclass".

For the exploit we borrowed code and the four secrets from the vendor. We adjusted some parts to work with Java 11, added error handling and argument parsing.

The end result is a small Java app which can decrypt any password stored in any install of the target app, globally.

At least until the vendor fixes their errors...

And until their customers apply the patch...

:)

Thank you for reading!
You'll find more of my stuff at github.com/tsluyter.

Hi there!

I'm Tess, an InfoSec enthusiast from the Netherlands. In daily life I'm a consultant, but I also love teaching about things like PKI, pen-testing and security.

Blog: https://www.kilala.nl
Twit: @Tess Sluijter

@sailorhg and @bØrk inspired me to try something new: sharing things I'd learned through a "zine".

So here we are!

You're reading my very first zine, in what I hope will become the series *"Things I've Learned"*.

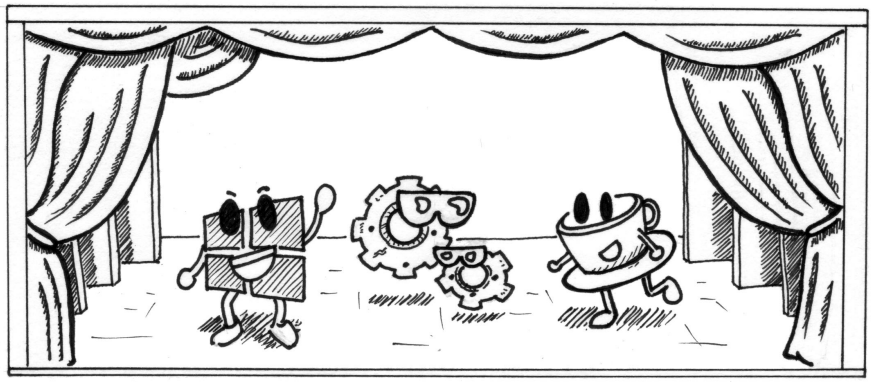
In *"The tale of the dubious crypto"* I will share some cool stuff I learned during a pentest, including Java decompilation and understanding other people's code.

Let's set the stage

We were asked to investigate an app which will not be named (CVEs still pending).

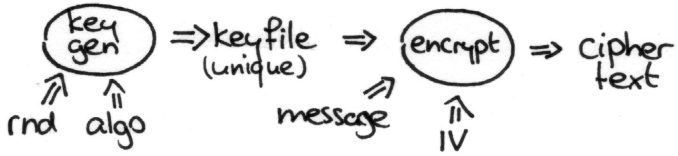
Things we figured out quickly:

- The app runs on Windows Server.
- It installs multiple Windows services.
- It's built as a standalone Java app.
- It has at least one admin web interface.
- It stores plenty of passwords!

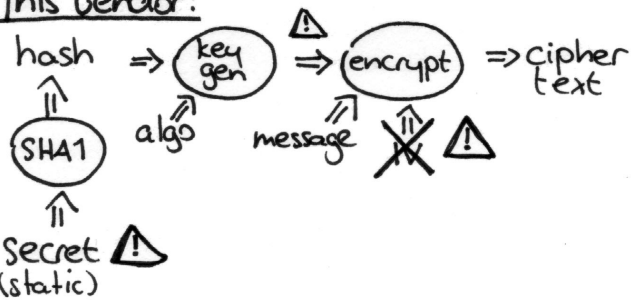


Final conclusion? We found four hard-coded "secret" strings used as key material input.

Proper crypto:



This vendor:



As a final exercise I wanted to build an exploit for this vulnerability. I've never written Java before, so Armen helped me out again. He taught me the bare minimum of code needed for a Java app.

```
class myClass
{
  public static void main(String[] arguments)
  {System.out.println("Hello world");}
}
```

The encryption key is loaded with SecretKeySpec(). By passing a key ("keyBytes") and an algorithm (from "gAlgo.getAlgo()). The "keyBytes" contents are made from the hash of a secret string ("gKey", passed as "secret"). We traced the source for these two to "product.server.c" and ".g".

```
if (defSection equals("admin"))
{ int ind = s.indexOf(',');
  if (ind > -1) {
  userHb.pub(s.substring(0, ind), cryptAdmin.decrypt
       (s.substring(ind+1, s.length()));}
```

Upon login the username and password are taken from "user.ini" (into "s"). The password (after the comma), is passed to "CryptAdmin".

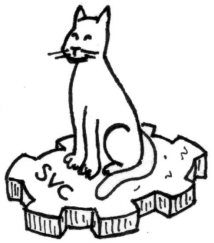
```
c CryptAdmin = new d().aM(com.vendor.product.
    server.c.hD[0]).aN("triple-DES").aG();
```

We just found the algorithm (triple DES) and a reference to the secret string. We looked for the definition of array "hD[]".

```
public static final String hD = {"Aa...c4", "E4...7D", "Fe...6D"};
```

It turns out that "hD" contains the secret for three types of passwords: admin, user and group.

One of the things to check are Windows services, to see if they're configured securely. So-called "*unquoted servicepaths*" are undesired! You can check each service's properties using "services.msc".



That's dreary work, so let's turn to Powershell!

```
ForEach ($Service in (Get-WmiObject -Class Win32_Service))
{ $ServiceEXE = $Service.pathname
  if ( $ServiceEXE -eq $null ) { continue }

  $IndexOfSpace = $ServiceEXE.indexOf(" ")
  $IndexOfExe = $ServiceEXE.ToLower().indexOf(".exe")
  $IndexOfQuote = $ServiceEXE.indexOf("`"")

  if ( ($IndexOfSpace -ne -1) -AND
       ($IndexOfSpace -lt $IndexOfExe) -AND
       ($IndexOfQuote -ne 0 ))
    { Write-Host "WARNING: $Service.Name" }
  }
```

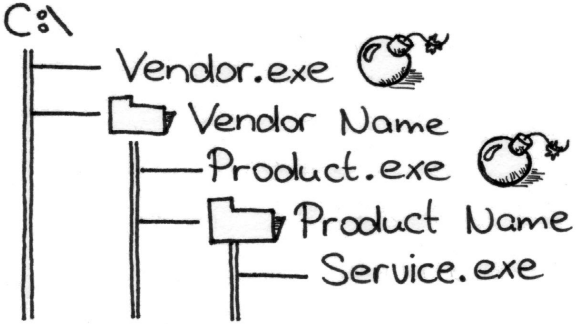
This told me that the application's services start a wrapper script from the following spot:

⇒ C:\Vendor Name\Product Name\web

It also showed that this path was NOT enclosed in quotes!

So why is this a bad thing?

Let's say that a malicious actor wants to replace the service with a binary of their own. If they can't change the service's definition, nor replace the target binary, they can try to hijack part of the path.



Without the quotes, Windows will gladly accept part of the path before a space as the target binary. Anything after the space will be seen as arguments.

If you can install either of the binaries shown above, you can make the computer do your bidding once the service restarts.

Always enclose service paths in quotes.

We struck paydirt! I'm only showing parts of the code from "product.encryption.impl.a". The "decrypt" method is the inverse of "encrypt".

```
public a(String secret, com.vendor.product.encryption.a algo)
 { this.gKey = secret;
   this.gAlgo = algo; }

public String encrypt(String message)
{ MessageDigest md = MessageDigest.getInstance("SHA-1");
  byte[] mdPass = md.digest(this.gKey.getBytes("utf-8");
  byte[] keyBytes = Arrays.copyOf(mdPass, 24);

  SecretKey key = new SecretKeySpec(keyBytes, this.gAlgo.getAlgo());
  Cipher cipher = Cipher.getInstance(this.gAlgo.getAlgo());
  cipher.init(1, key);

  byte[]. plaintext = message.getBytes("utf-8");
  byte[] buf = cipher.doFinal(plaintext);
  return DataTypeConverter.printBase64Binary(buf);
}
```

I checked Java's docu for "cipher.init" and learned that you usually pass more than two parameters. Not just the key ("key") and the algorithm (gAlgo.getAlgo()), but also some IV to introduce randomness.

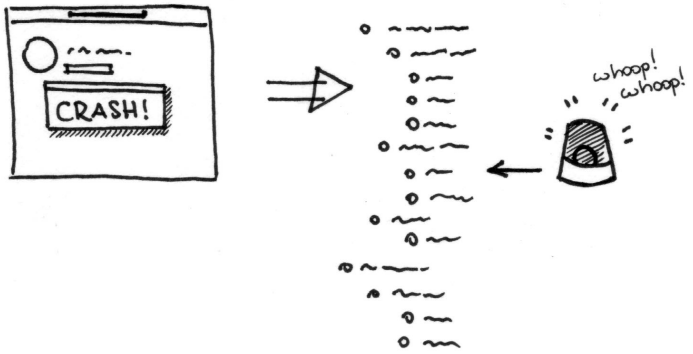
The vendor skipped the IV and also did nasty things with the key! The app does not have a key! They just remake the same one...

The admin web interface spans a few dozen files, most of which had obfuscated names like "a.class", "b.class", etc. To find out where I had to start my search I had to make the application crash.

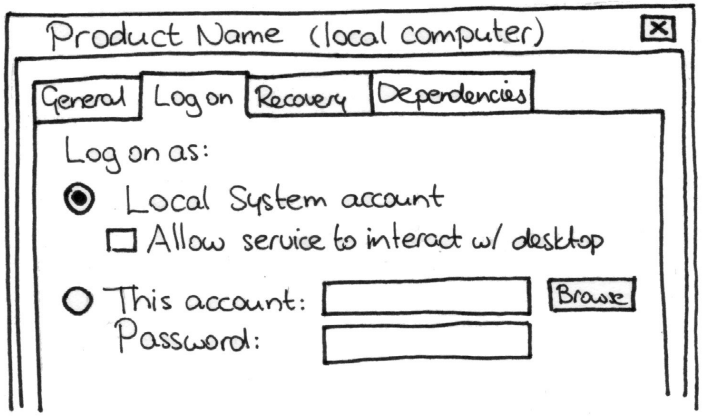
I edited "user.ini" and made an invalid password string by removing one of the trailing "=" characters. When I tried logging into the web interface, this was denied. A Java stack trace showed up in "logs\wrapper.log".

This showed the following consecutive calls:

```
product.server.c.l
 product.server.utils.g.b
  product.encryption.impl.a.decrypt
   product.encryption.b
    javax.xml.bin.dataTypeConverter.parseBase64Binary
```



Another thing that I noticed was that the application's services run as "SYSTEM". You can tell by looking at another Win32_Service property called "StartName". You can also see this on the "Log on" tab of "services.msc".

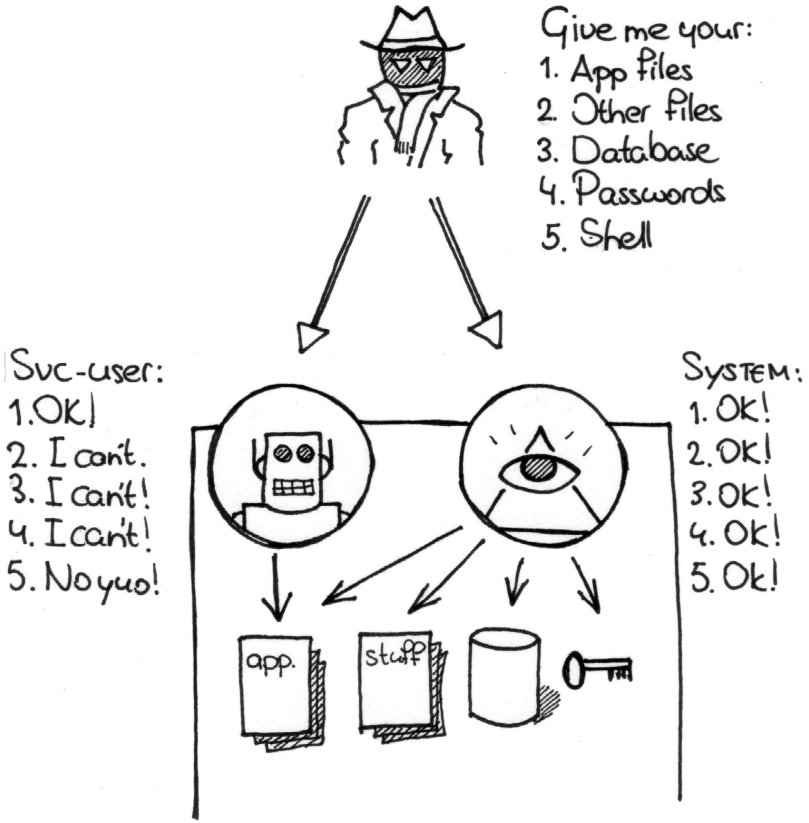


You could say that "SYSTEM" is Windows' equivalent of "root" on Unix: the local God-like account which can access any resource on the computer.

Running your services as "SYSTEM" is not best practice. It is usually recommended that you create a specific service account, which can only access its intended files.

Again, what's the big deal? Well, someone could find a nasty bug in the service. Maybe it's a buffer overflow, maybe it's an RCE or some file inclusion. Either way it could allow the malicious actor to tell the computer to do things outside of the services' normal tasks.

And if that software runs as "SYSTEM" the whole computer and its contents are forfeit!



There's only so much I could do "black boxing" the encryption. I needed to get a look at the insides of the program. I grabbed a copy of the whole web interface and put it on my Linux workstation. Now I could prod at it with "jd-gui" (github.com/java-decompiler).

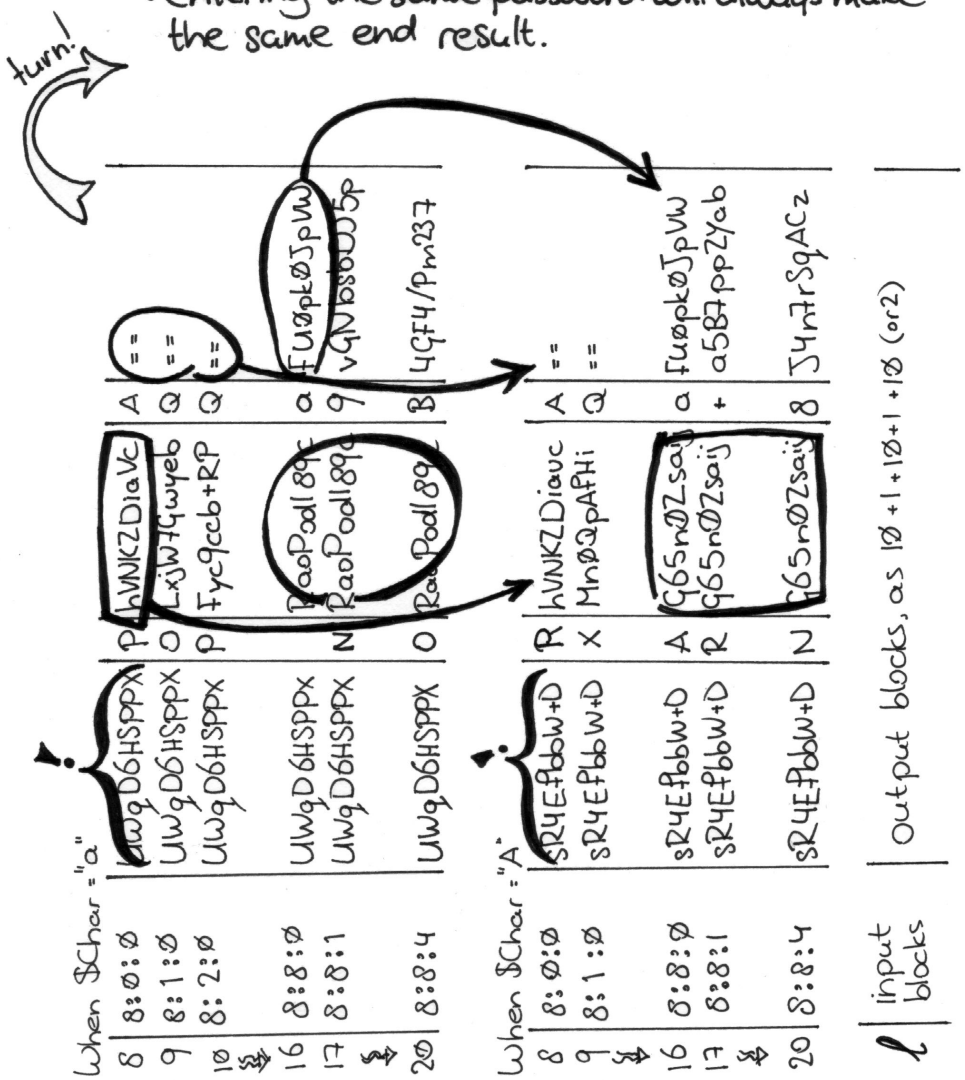
A Java decompiler will take a Java app and turn it inside out, showing you the source code. It's magical! There's a wide variety to chose from, even online ones! (Better not use those on confidential stuff though!)

Unfortunately I know just about fsck-all about Java! I found promising parts of the source code, but quickly got lost in Java's style of referencing variables.

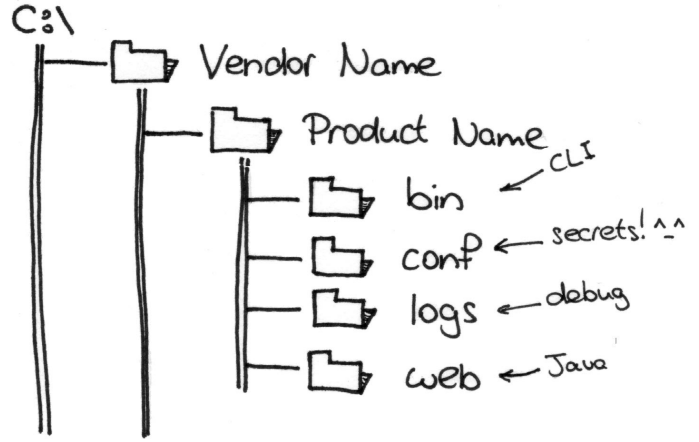
What a lucky break that my colleague Armen knows his way around Java!

The outcome showed recurring patterns!

- Passwords are chopped into blocks, 8:8:4 chars.
- Each block is processed seperately.
- The outcome consists of max. 10+1+10+1+10 chars.
- The same block will generate the same output.
- Entering the same password will always make the same end result.



Speaking of files and resources, let's take a look at our web interface! We'll dive into the path we found in the service definition.



The "logs\wrapper.log" gives a nice, running commentary on everything the web interface does. The "web" dir was chockfull of JSP, WAR and CLASS files. And "bin" has goodies we will get to later.

Paydirt was struck in "\conf\users.ini".

Notepad - users.ini

```
[admin]
admin,d5KKKqMRuautgXxbm7j9QB==
[users]
```

That admin entry looks promising! Smells like Base64! So I tried decoding it:

```
$ echo "d5KKKgMRuautgXxbm7j9QB==" | base64 -D
w???*?????!L???H
```

The output was something binary, so either:
A) This was not Base64, but some odd hash,
B) It's binary data,
C) It's encrypted data, or
D) It's something else. ^_^

Time to start playing around!

In the "bin" directory I found a batch script, "consoleuser.bat", that lets you change any user's password. It passes your input to a call to the Java app. I added a new admin user called "tester". The batch script would change the password, but the login page would only accept the initial "admin" user.

```
C:\%Bindir% > .\consoleuser.bat -reset \
                tester testing123
Password has been reset successfully!
```

Playing with the reset tool taught me that valid passwords were between 8 and 20 chars long. Curious to see the changes in "user.ini" I wrote a Powershell script to iterate passwords.

```
$ResetCmd = "$ProductDir\bin\consoleuser.bat"
$ConfFile = "$ProductDir\conf\user.ini"
$Testchars = "a","A","z","Z"

ForEach ($Char in $Testchars)
{ $Length = 8
  While ($Length -lt 21)
{ $Password = "$Char" * $Length
  Start-Process -Wait "$ResetCmd" "-reset tester $Password"

  $ConfContent=(Select-String -Pattern "tester" -Path `
                $ConfFile).Line.Split(",")[1])

  Write-Output ( $Password + "," +
                 $ConfContent.Substring(0,10) + "," +
                 $ConfContent.Substring(10,1) +"," +
                 $ConfContent.Substring(11,10) +"," +
                 $ConfContent.Substring(21,1) +"," +
                 $ConfContent.Substring(22))
}
  $Length +=1
}
```