Smartcard-Reader von Kobil geknackt

Der zertifizierte Klasse 3 Smartcard-Reader Kaan Tribank von Kobil wurde geknackt.

17.04.2010 (Version 1.0) aktuelle Version auf http://colibri.de.ms/ => Smartcard-Reader-Hack

Colibri <colibri_dvb@lycos.com>

Inhalt

Smartcard-Reader von Kobil geknackt

Einleitung

Beim zertifizierten Klasse 3 (hat eigene Tastatur und Display) Smartcard-Reader Kaan Tribank von Kobil kann man über die mitgelieferte Dockingstation auch vom Computer aus mit dem Reader arbeiten. Aus Sicherheitsgründen muss die PIN über die eigene Reader-Tastatur eingegeben werden damit evtl. vorhandene Trojaner auf dem PC keine Change haben die PIN zu ermitteln. Auch ein Firmwareupdate kann man vom PC aus durchführen. Um zu verhindern das Trojaner eine unauthorisierte Firmware (die z.B. die eingegebene PIN zum PC schickt) auf dem Reader aufspielt ist das Firmwareupdate signiert (Hashalgorithmus: SHA-1 / Signaturalgorithmus: asymmetrisches ECDSA bei einer Bitlänge vom 192).

In der Anleitung die dem Reader beiliegt behauptet Kobil dass nur authentische Firmware Versionen von KOBIL vom Gerät akzeptiert werden. Da ganz allgemein Hersteller natürlich vieles behaupten können lege ich persönlich wert darauf, dass Sicherheitsfunktionen auch von unabhängigen Parteien bestätigt wurden. Dieser Leser wurde vom Bundesamt für Sicherheit in der Informationstechnik (BSI) und von T-Systems zertifiziert [1][2]. Die schreiben in Ihrer Bestätigung u.a. folgendes:

BSI:

„Die Chipkartenterminal-Produktvarianten […] KAAN TriB@nk wurden erfolgreich nach den Common Criteria (CC) mit der Prüfstufe EAL3+ (EAL3 mit Zusatz AVA_VLA.4 (gegen ein hohes Angriffspotential), AVA_MSU.3 (eine vollständige Missbrauchsanalyse), ADV_IMP.1, ADV_LLD.1 und ALC_TAT.1, ADO_DEL.2 (Erkennung von Manipulation)) evaluiert. Die eingesetzten Sicherheitsfunktionen erreichen die Stärke hoch.“

T-Systems:

„Die Chipkartenterminals […] KAAN TriB@nk“ gemäß Abschnitt 1.1 wurden erfolgreich nach der Common Criteria Prüfstufe EAL3+ (mit Zusatz in Übereinstimmung mit Anlage 1, Abschnitt I, Nr. 1.2 SigV) re-evaluiert. Die eingesetzten Sicherheitsmechanismen erreichen die Stärke "hoch".“

Um ganz sicher zu gehen habe ich den Leser selber analysiert und mit Entsetzen festgestellt das sich der Leser sehr wohl hacken hat lassen, also eine von mir manipulierte Firmware annimmt und durch einen Fehler in der Implementierung der Signaturprüfung auch ausführen lässt.

Folgendes Kapitel beschreibt die Details zu dem Hack auch ein Tool mit dem man den Hack selbst ausprobieren kann ist verfügbar [3]. Nach der Beschreibung des Tools folgen Empfehlungen für Anwender und Hersteller.

Hackdetails

Um herauszufinden ob die Signaturprüfung sicher ist habe ich mir erst mal eine unverschlüsselte Firmware besorgen müssen um die Signaturprüffunktion auf Fehler untersuchen zu können. Dazu war es nötig den Reader zu zerlegen um die Firmware auszulesen und letztendlich ein Tool zu schreiben das den Fehler demonstrieren kann in dem es eine unauthorisierte Firmware aufspielen und starten kann. Für die Phase der Informationsbeschaffung war also alles erlaubt. **Um vom einen Hack sprechen zu können ist es wichtig zu beachten dass das Demonstrationstool an einen zweiten ungeöffneten noch versiegelten Reader auch funktioniert - was es auch tut.**

Informationsbeschaffung

Ich hatte meinen Reader bereits vor längeren gekauft. Ich hatte mir damals die Firmwareupdate_KAAN_TriBank_V55.16_Docking_Station_V0.32.exe runtergeladen und den Reader upgedatet. Ich habe den Reader und die EXE jetzt näher analysiert.

Startet man die EXE wird sofort die Version geprüft, ggf. der Update durchgeführt und die EXE wieder beendet.

Ich habe gesehen das im Temp-Verzeichnis“ C:\Dokumente und Einstellungen\<username>\Lokale Einstellungen\Temp\“ Unterverzeichnisse mit Dateien angelegt werden, aber beim Beenden der EXE sofort wieder gelöscht werden. Ich deshalb nach dem Starten der EXE sofort die Unterverzeichnisse kopiert um sie in Ruhe analysieren zu können.

Darin war u.a. die Datei Online_V0.32.dfu enthalten. DFU-Files sind für die Dockingstation und für den Hack nicht relevant.

Auch die Firmwaredateien für den Reader selbst waren vorhanden:

- eTAN_HHD_V55.16.upd
- BootloaderUpdate_V16.upd

Beim Öffnen im Hexeditor kann man sehen das die .upd-Files einen 100h langen Vorspann (TLV-Struktur) haben. Ist aber uninteressant da hier z.B. nur nochmal der Dateiname und die Version stehen. Der Rest der Datei ist verschlüsselt da keinerlei Texte sichtbar sind. Man kann aber sehen dass vor jedem 200h Byte langen verschlüsselten Block ein unverschlüsselter Header steht:

- 80 FF 00 30 00 02 00
- 80 FF 00 31 00 02 00
- 80 FF 00 32 00 02 00
- …
- 80 FF 00 7F 00 02 00
- 80 FF 00 80 00 00 32

Der letzte Block hat nur 32h Byte Daten.

Byte 6 und 7 (0200h und 0032h) geben also die Länge der folgenden Daten an.

Byte 4 (30h..80h) könnte was mit der Adresse zu tun haben an der die entschlüsselten Daten gespeichert werden.

Untersucht man den Teil des Programms der die Firmware zum Reader schickt sieht man dass da nichts entschlüsselt wird. Es wird also erst im Reader entschlüsselt. Also kommt man um das Zerlegen des Readers und Auslesen der Firmware nicht herum.

Ich habe den Reader zerlegt und hinter dem Display den relevanten Chip gefunden.

Es ist leider keine Beschriftung zu sehen nur ein schwarzer Brocken Harz.

Um den Chip zu ermitteln habe ich die Webseiten der gängigen Hersteller abgeklappert.

Als Anhaltspunkt hatte ich die Anzahl der Pins (64).

Neben der Schnittstelle in Richtung Dockingstation gibt es auch eine zusätzliche Schnittstelle JP1 (über die kann man die Firmware auslesen). Ich habe die beschrifteten Leitungen der Schnittstelle zum Prozessor zurückverfolgt um zu sehen welche PIN Nr. welche Funktion hat, außerdem habe ich noch die Leitungen von beiden Oszillatoren zurückverfolgt. Ich hatte jetzt genügend Anhaltspunkte um den Chip anhand der Datasheets zu ermitteln.

Es stellte sich letztendlich heraus dass es der Toshiba Chip TMP86FS49 ist. Das Datasheet ist unter [4] zu finden.

Interessant ist „20. Serial PROM Mode“ im Datasheet. Über die Pins der zuvor erwähnten Schnittstelle JP1 kann die Firmware bei der Produktion von Kobil aufgespielt werden. Ich habe die Schnittstelle über einen Pegelkonverter mit der seriellen Schnittstelle von meinem PC verbunden.

Die Schnittstelle unterstützt diverse Kommandos u.a. Flash löschen, beschreiben, Prüfsumme bilden, Produkt ID ausgeben, Programme ins RAM schreiben und starten (RAM loader), Sicherheitseinstellungen setzen und abfragen.

Da es keinen direkten Befehl gibt um das Flash auszulesen musste ich den Befehl „RAM loader“ verwenden. Ich habe ein kleines ASM-Program geschrieben das das Flashmemory ausliest und Byte für Byte über die serielle Schnittstelle ausgibt. Das Program muss dem RAM loader als Intel Hex Format (Binary) übergeben werden, also konkret:

```
:100100004C950F1883F4784CFF0F18C0F4784C9777
:100110000F4B0090490070E1950F5ADBFA0DF4780F
:100120003339D9F34CFF0F18C8F4784C970F4B00B4
:1001300080490080E1950F5ADBFA0DF4783339D904
:03014000F3FCFECF
:00000001FF
```

Die Empfangenen Bytes entsprechen der unverschlüsselten Firmware.

Eine kleine Schwierigkeit gab es aber noch bevor der RAM loader Befehl ausgeführt werden konnte.

Der Chip unterstützt zwei Sicherheitsfeatures.

Bei einem leeren Flash ist keines aktiv und der Chip kann problemlos beschrieben werden.

Im beschriebenen Zustand, so wie der mir vorlag, ist jedoch ein Passwortschutz zwangsweise aktiv.

Der Passwortschutz bewirkt dass man kritische Befehle jetzt nur noch mit Passwort ausführen darf, dazu gehört auch der RAM loader.

Der zweite optionale Schutz verhindert alle kritischen Befehle wie den RAM Loader permanent (auch wenn man das Passwort kennt). Der Schutz kann nur noch durch den Chip erase Befehl aufgehoben werden. Wäre der optionale Schutz durch Kobil aktiviert worden, dann hätte ich die Firmware nicht auslesen können und ich hätte hier aufgeben müssen. Da Kobil den Schutz nicht aktiviert hatte ging es weiter und es war nur noch der Passwortschutz im Weg.

Vom Chip wird ein Passwort erzwungen das min. 8 Stellen lang ist und nicht aus zu vielen gleichen Bytes bestehen darf. Das Passwort ist auch nicht auf ASCII-Zeichen beschränkt sondern darf alle 256 Werte eines Bytes nutzen. Gäbe es nur ein Passwort wäre ein Knackversuch also ziemlich hoffnungslos (bei min. 256 * 256 * 256 * 256 * 256 * 256 * 256 * 256 Möglichkeiten).

Toshiba ist aber im Gegensatz zum optionalen permanenten Schutz beim Passwortschutz aber nicht so streng. Wie im Datasheet beschrieben gibt es nicht ein einziges Referenzpasswort im Chip mit dem das Übermittelte verglichen wird. Stattdessen kann man sich bei der Authentifizierung eine Adresse in der Firmware aussuchen mit der die min. 8 Bytes verglichen werden. Man gibt dabei auch

eine zweite Adresse in der Firmware an von der das Längenbyte für den Vergleich der Passwörter verwendet werden soll geladen wird.

Die Default Einstellung die Toshiba bei seinem Flashtool verwendet ist:

- Adresse der Passwort Länge: FE80h
- Adresse des Passworts: FE81h
- Password string: 01 02 03 04 05 06 07 08 (diesen hat Kobil aber geändert)

Da das default Toshiba Passwort nicht funktionierte, brauchte ich einen min. 8 Stellen langen Teil aus der Firmware um die komplette Firmware auszulesen.

Mir ist dann eingefallen dass ich ja Teile aus der Firmware kannte, die Meldungen im Display des Readers müssen ja auch in der Firmware gespeichert sein und diese können ja als Referenzpasswort angegeben werden. Schiebt man eine Smartcard mit der falschen Seite in den Reader kommt die Meldung „Kartenfehler“. Ich habe mich entschlossen diesen Text als Passwort zu übermitteln.

Jetzt brauchte ich nur noch die Adresse des Referenzstrings übermitteln mit der verglichen werden soll. Da blieb dann nur das Durchprobieren aller Flashadressen (1000h-FFFFh) übrig.

Das letzte Problem war dann noch die Adresse die als Passwortlänge verwendet werden soll. Das Byte an der Adresse musste eine Zahl zwischen 8 und 12 haben, da „Kartenfehler“ 12 Stellen hat und min. 8 Stellen Voraussetzung ist. Nach zwei Tagen automatisierten Durchprobieren hatte ich es geschafft. Ich habe dann die Firmware empfangen und am PC abgespeichert.

Zuerst hat mich interessiert (obwohl ich es jetzt nicht mehr brauchte) welches Passwort denn Kobil an der von Toshiba vorgesehenen default Adresse FE81h gespeichert hat. Ich war enttäuscht von Kobil:

```
0000fe60h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ; ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0000fe70h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ; ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0000fe80h: 08 47 65 68 65 69 6D 3F 21 56 35 35 2E 31 36 00 ; .Geheim?!V55.16.
0000fe90h: FF 7F BF 3F DF 5F 9F 1F EF 6F AF 2F CF 4F 8F 0F ; ÿ□¿?ß_Ÿ.ïo¯/ÏO□.
0000fea0h: F7 77 B7 37 D7 57 97 17 E7 67 A7 27 C7 47 87 07 ; ÷w·7×W—.çg§'ÇG‡.
```

Ich habe dann die Firmware disassembled und mir die Signaturprüfung angesehen.

Fehler in der Signaturprüfung

Zuerst mal grob die Info wie die das Firmwareupdate vom Reader verarbeitet wird.

Der Flashspeicher im Reader geht von 1000h bis FFFFh. Er ist logisch in zwei Bereiche unterteilt:

- Bootloader: 1000h bis 5FFFh
- Anwendung und Resetvektor: 6000h bis FFFFh

Der Bootloader enthält die Funktionen für das Firmwareupdate. Er bietet auch ein paar Basisfunktionen (EEPROM auslesen und beschreiben, Text am Display ausgeben, Datenaustausch mit der Dockingstation, Ver- und Entschlüsselung von Daten, …) die die Anwendung aufrufen kann.

Die Version der Anwendung (z.B. V55.16) muss deshalb mit der Version des Bootloaders (z.B. V16) zusammenpassen. Konkret muss die Versionszahl in der Anwendung nach dem Punkt also xx.16 mit der Bootloader Version V16 übereinstimmen.

Nach einem Reset wird normalerweise die Anwendung ausgeführt.

Wird ein Firmwareupdate übertragen dann wird das Flash so umprogrammiert dass nach einem Reset der Bootloader aufgerufen wird. Der Bootloader nimmt jetzt das Firmwareupdate entgegen und schreibt es entschlüsselt in den Bereich der Anwendung.

Jetzt nochmal kurz zurück zu den Headern der oben erwähnten .upd-Files:

- 80 FF 00 **30** 00 02 00
- 80 FF 00 **31** 00 02 00
- 80 FF 00 **32** 00 02 00
- …
- 80 FF 00 **7F** 00 02 00
- 80 FF 00 **80** 00 00 32

- 80 FF 00 <**Adressinfobyte**> 00 <Datenlänge>

Die Adresse an die der entschlüsselte 200h Byte lange Block geschrieben wird errechnet sich wie folgt:

Flashadresse = Adressinfobyte * 200h

Bei 30h wird der Block also ins Flash ab Adresse 6000h (Anfang der Anwendung) geschrieben.

Aus Sicherheitsgründen wird der Resetvektor nicht überschrieben, er zeigt immer auf den Bootloader. Es ist also keine Kunst eigenen Code ins Flash zu schreiben, damit hat Kobil auch gerechnet, das Problem ist aber den Reader dazu zu bringen den Resetvektor wieder vom Bootloader auf die Anwendung umzustellen, damit der eigene Code auch Ausgeführt wird.

Der einzige Weg ist das Kommando mit dem Adressinfobyte 80h und den 32h Byte langen Datenteil zu schicken. Der Datenteil entspricht der Signatur. Während vorher die entschlüsselten Daten ins Flash geschrieben worden sind wurde auch ein SHA-1 Hashwert über alle entschlüsselten Daten gebildet. Das Adressinfobyte 80h bewirkt das die Signatur geprüft wird. Wenn die Signatur passt, dann wird der Resetvektor vom Bootloader auf die Anwendung umgestellt. Falls die Signatur nicht passt bleibt der Bootloader aktiv und er wartet auf ein erneutes Firmwareupdate. Eine eigene Signatur für seine manipulierte Firmware zu berechnen geht nicht da der Algorithmus selbst sicher ist. Jedoch gibt es einen gravierenden Implementierungsfehler im Reader der ausgenutzt werden kann um trotzdem eigenen Code auszuführen.

Der Fehler ist das der SHA-1 Hashwerts nur über die 200h Byte langen Datenblöcke gebildet wird **aber nicht über das Adressinfobyte**. Das heißt man bei einem original Firmwarefile die ganzen Adressinfobytes ändern kann und das Firmwareupdate trotzdem aktiviert wird.

Eigenen Code kann man dadurch wie folgt ausführen lassen:

Erst wird in Phase 1 der eigene Code zum Reader übertragen. Dieser wird auch im Flash gespeichert, jedoch wegen der falschen Signatur nicht ausgeführt Resetvektor zeigt immer noch auf den Bootloader.

Dann wird in Phase 2 ein original Kobil Image zum Reader geschickt. Nur die Adressinfobytes werden auf einen unbenutzen Block umgeleitet. Unser in Phase 1 geschriebener Code wird durch die Umleitung nicht überschrieben. Aber da die Signatur beim original Kobil Image richtig war wird der Resetvektor vom Bootloader auf unsere manipulierte Anwendung umgestellt.

Eine kleine Einschränkung gibt es durch den Trick aber noch. Für den manipulierten Code steht nur der Bereich 6000h bis EFFFh (statt bis FFFFh) zur Verfügung. Hintergrund ist folgender:

Ein Flashsektor (ist 1000h Bytes lang) muss vor dem Neuprogrammieren gelöscht (Sector erase) werden bevor er mit neuen Daten beschrieben werden kann.

Empfängt der Reader während einer Updatesession das erste Mal einen Block der sich innerhalb des Bereichs 6000h – EFFFh befindet, dann löscht der Reader gleich die ganzen Sektoren 6000h – EFFFh.

Empfängt der Reader während einer Updatesession das erste Mal einen Block der sich innerhalb des Bereichs F000h – FFFFh befindet, dann löscht der Reader gleich den ganzen Sektor F000h – FFFFh.

Ich habe bei dem Trick die original Firmware in den unbenutzten Bereich FC00 – FDFF umgeleitet. Der Bereich 6000h – EFFFh mit der manipulierten Firmware wurde dadurch nicht gelöscht. Aber durch die Umleitung wurde der Bereich F000h – FFFFh gelöscht (bis auf Reset Vektor und ein paar andere Bytes). Darum kann man nur bis zur Adresse EFFFh eigene Firmware nutzen.

Aber auch diese Beschränkung lässt sich aufheben. Dazu muss man nur die Signaturprüfung im Bootloader ganz abschalten. Dann muss man beim Testen verschiedener eigener manipulierter Anwendungen nicht immer die original Firmware hinterherschicken um die langwierige Signaturprüfungen zu bestehen.

Schauen wir uns dazu an wie ein Bootloader update im Vergleich zu einem Anwendungsupdate abläuft. Ein neuer Bootloader wird vom Hersteller mit einer Kopier- und Programmierfunktion ausgestattet außerdem ist der eigentliche Bootloader in Blöcke mit Adressinfos strukturiert damit die Programmierfunktion weis wo es die Blöcke ins Flash schreiben muss. Das Bootloader update wird wie eine Anwendung erst in den Bereich ab 6000h ins Flash geschrieben. Nach dem Aktivieren kopiert sich der Bootloader in den unteren Bereich 1000h bis 5FFFh. Die ursprüngliche Anwendung geht bei einem Bootloader update also immer verloren und muss anschließend aufgespielt werden.

Um eigene Firmware verschlüsseln zu können was es notwendig folgende Funktion zu finden:

```
2BC3h: LD   (SP-),0     #Flag 0=Entschlüsseln 1=Verschlüsseln

2BC6h: LD   WA,0x5A     #RAM Adresse an der die berechneten
2BC9h: PUSH WA          #Rundenschlüssel abgelegt werden sollen

2BCAh: LD   WA,0x11CF   #An Adresse 11CFh befindet sich der
2BCDh: PUSH WA          #8 Byte lange DES-Key zum Entschlüsseln
                        #der Firmware

2BCEh: CALL 0x4342      #SetDesKey(DesKey, RoundKeys, Flag)
```

Zum Entfernen der Signaturprüfung war es nur notwendig drei Bytes (48 E9 11 -> FE 7F 33) zu am Anfang der Signaturprüfung zu ändern damit ein Jump Befehl entsteht der in den Programzweig springt der normalerweise nur bei einer erfolgreichen Prüfung ausgeführt wird.

Mit dem Demo-Tool kann man sich die beiliegenden Demo-Firmwareupdates entschlüsseln lassen. Dann kann man sie vergleichen um die kleinen Unterschiede zu sehen.

Ist der manipulierte Bootloader im Reader, dann kann man auch die manipulierte Demo-Anwendung flashen. Mit einem HEX-Editor sieht man dass die 32h Bytes am Ende falsch sind (alle Signaturbytes sind AAh). Trotzdem wird nach dem Update ab sofort immer wenn man den Reader über die Dockingstation mit dem PC verbindet ein anderer Text im Display angezeigt.

Vorher:

Nachher:

Vorher:

```
BA01h: LD    A,0xC       #Byte für Clear Screen laden
BA03h: CALLV 1           #und Byte zum Display schicken

BA04h: LD    WA,0xF66D   #0xF66D enthält „KAAN TriB@nk“
BA07h: CALLV 2           #und Text zum Display schicken

BA08h: LD    A,0xA       #Byte für Zeilenumbruch laden
BA0Ah: CALLV 1           #und Byte zum Display schicken

BA0Bh: LD    WA,0xFE89   #0xFE89 enthält „V55.16“
BA0Eh: CALLV 2           #und Text zum Display schicken

BA0Fh: LD    WA,0xF6CA   #0xF6CA enthält „ Secoder 1“
BA12h: CALLV 2           #und Text zum Display schicken

BA13h: NOP               #
BA14h: RET               #
```

Nachher:

```
BA01h: LD    WA,0xF7B0   #0xF7B0 enthält
                         #0x0C, “*HACKED* version”,
```

```
                              #0x0A, "colibri.de.ms"
BA03h: CALLV 2                #und Text zum Display schicken
BA05h: NOP                    #
…
BA13h: NOP                    #
BA14h: RET                    #
```

Benutzen des Demo-Tools

Unter [3] gibt ein ReaderHack.zip zum Runterladen. Das Zip enthält folgende Dateien:

- ReaderHack.exe
- BootloaderUpdate_V16_org.upd
- eTAN_HHD_V55.16_org.upd
- BootloaderUpdate_V16_hack_prepare.upd
- BootloaderUpdate_V16_hack_activate.upd
- eTAN_HHD_V55.16_hack.upd

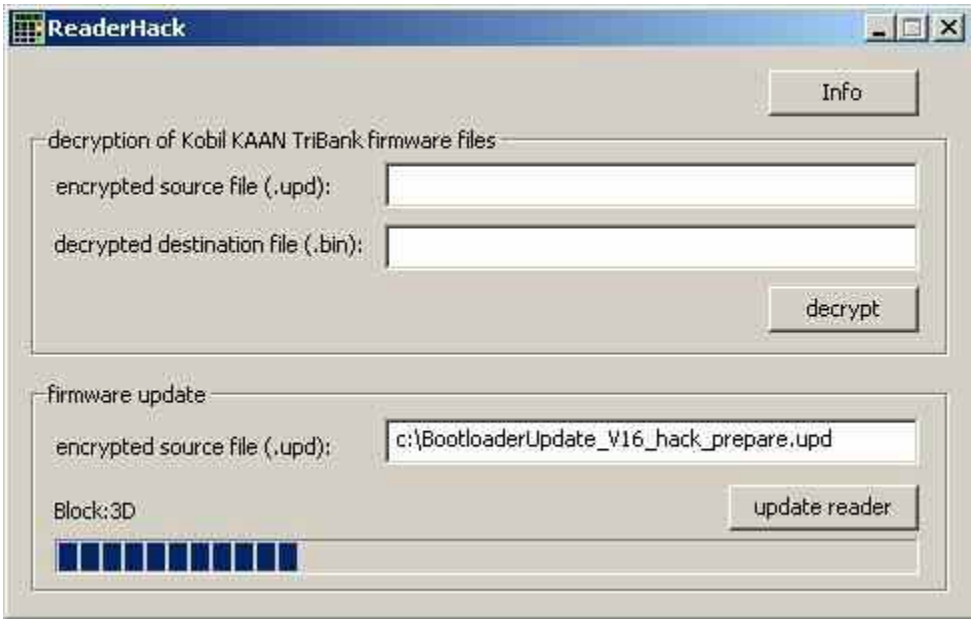
Ziel dieser Demo ist es zu zeigen dass es möglich ist eine manipulierte Firmware in den Reader zu bringen und ausführen zu lassen. Nach der Prozedur ist die Signaturprüfung im Bootloader deaktiviert und die Anwendung zeigt beim Verbinden mit dem PC eine geänderte Meldung im Display.

Das Tool wurde mit einem versiegelten Reader der die neueste Firmware (Firmwareupdate_KAAN_TriBank_79.22_Docking_Station_0.39.exe) hatte erfolgreich getestet. Bei noch neueren Versionen in der Kobil den Bug gefixt hat funktioniert das Tool wahrscheinlich nicht mehr.

Reihenfolge der Schritte:

1. ReaderHack.exe starten
2. Sicherstellen das der Reader über die Dockingstation mit dem PC verbunden ist
3. Bei „encrypted source file (.upd)“ den Pfad und den Namen der Firmware “BootloaderUpdate_V16_hack_prepare.upd” eingeben
4. „update reader“ anklicken
5. Es erscheint einmalig während der Sitzung die Sicherheitsabfrage „Wollen Sie wirklich eine neue Firmware in den Reader schreiben?“ die entsprechend mit „Ja“ oder „Nein“ beantwortet werden kann. (Manchmal wird der Anzeige im Tool nicht aktualisiert – im Hintergrund läuft die Übertragung aber weiter - Der Abschluss wird aber immer mit „done“

bestätigt).



6. Der Update läuft jetzt und nach dem Übertragen der Daten gibt es eine längere Phase der Signaturprüfung durch den Reader „Verify…“. Da das nur der erste Teil der Manipulation ist muss die Signaturprüfung mit der Meldung „Update FAILED Consult Manual“ am Reader scheitern.
7. Bei „encrypted source file (.upd)“ den Pfad und den Namen der Firmware „BootloaderUpdate_V16_hack_activate.upd” eingeben.
8. „update reader“ anklicken
9. Jetzt wird hauptsächlich Block 7E upgedatet und es kommt keine Meldung „Update FAILED Consult Manual“ mehr. Jetzt ist der manipulierte Bootloader im Reader aktiviert worden.
10. Bei „encrypted source file (.upd)“ den Pfad und den Namen der Firmware „eTAN_HHD_V55.16_hack.upd” eingeben.
11. „update reader“ anklicken
12. Nach Übertragung des letzten Blocks erscheint diesmal kein „Verify…“ mehr, sondern die Anwendung wir sofort ungeprüft ausgeführt und die folgende Text wird angezeigt:

Um wieder auf die Originale Firmware zurückzustellen. Kann man entweder erst „BootloaderUpdate_V16_org.upd“ und dann „eTAN_HHD_V55.16_org.upd“ einspielen.
Noch besser ist es aber gleich die neueste Firmware von der Kobil Webseite [5] einzuspielen.

Empfehlungen

Empfehlung für die Anwender:

- Den Reader bis nicht mehr mit dem PC verbinden (sondern nur noch offline Nutzen), bis der Hersteller auf der Webseite [5] einen Bugfix für den Reader bereitgestellt hat. Gibt es nur eine neue Version ohne Kommentar, dann könnte der Bug noch existieren. Steht aber bei der neuen Version dabei das der Signaturfehler behoben wurde dann kann man davon ausgehen dass das auch stimmt.

Empfehlungen für den Hersteller:

- Überprüfen ob die anderen Klasse 3 Reader auch betroffen sind. In der Regel werden wird der Quellcode für die Signaturprüfung nicht immer wieder für jedes Model komplett neu geschrieben. Dadurch könnte es theoretisch sein das der Bug auch bei den anderen Modellen existiert und auch gefixt werden muss.
- Für alle betroffenen Reader einen Hinweis auf der Webseite geben damit die Benutzer informiert sind.
- Dem BSI und der T-Systems die die fehlerhafte Version zertifiziert [1][2] haben, bescheid geben das die die Zertifizierung für die fehlerhaften Versionen wiederrufen.
- Nach dem Bereitstellen einer fehlerfreien Version, diese wieder zertifizieren lassen.

Literaturverzeichnis

	Beschreibung
[1]	Bestätigung BSI.02096.TE.12.2008 https://www.bsi-fuer-buerger.de/cae/servlet/contentblob/485368/publicationFile/29542/02096_pdf.pdf
[2]	T-Systems.02219.TU.04.2009 Nachtrag Nr. 1 zur Bestätigung BSI.02096.TE.12.2008 vom 19.12.2008 http://www.t-systems-zert.de/pdf/ein_02_sig_pro/zf_02219_d.pdf
[3]	http://colibri.de.ms/ => Smartcard-Reader-Hack
[4]	http://www.semicon.toshiba.co.jp/eng/product/micro/selection/870family/870c/selection/index.html Die Datasheets für TMP86FS49**BFG** oder TMP86FS49**BUG** unterscheiden sich anscheinend nur in der Bauform QFP bzw. LQFP. Also einfach eins der Beiden verwenden.
[5]	Downloadseite von Kobil http://www.kobil.com/index.php?id=44