Das Nürnberger Anti-Phishing-Device: Beweisbare Korrektheit durch Einfachheit

Peter Trommler

9. Mai 2016

Zusammenfassung

Angriffe gegen das Online-Banking richten sich meist gegen den Rechner des Benutzers, sodass externe Geräte zum Schutz von Transaktionen verwendet werden. In diesem Beitrag wird das Nürnberger Anti-Phishing Device vorgestellt, das eine sichere Umgebung für die Erstellung digital unterschriebender Transaktionszusammenfassungen umsetzt. Durch die Beschränkung der Funktionalität auf des minimal Notwendige, ist eine formale Verifikation der Software möglich.

1 Einleitung

Online-Banking ist ein beliebtes Ziel für Kriminelle im Internet. Der Kriminelle versucht dabei zumeist den Rechner (oder Tablet, Smartphone) des Kunden derart zu manipulieren, dass er Transaktionen des Kunden auf ein Konto unter seiner Kontrolle umlenken oder ohne Wissen des Kunden selbst Transaktionen veranlassen kann.

Zur Verhinderung solcher Angriffe werden derzeit verschiedene Lösungen mit externen Geräten von den Banken angeboten. Dabei werden unter anderem Geräte mit einer unidirektionalen Verbindung vom Rechner zum Gerät angeboten (z. B. "Flimmern" am Bildschirm). Der Rückkanal wird durch den Kunden realisiert, der Daten von einem Display des Geräts abliest und am Rechner eingibt, was die Verwendung einer kompletten digitalen Unterschrift als "Transaktionsnummer" aus Gründen der Benutzbarkeit ausschließt. Die Anbieter begründen das Fehlen eines "automatischen" Rückkanals mit dem Schutz des Geräts vor Angriffen, da es nicht mit dem Rechner "verbunden" ist.

Im Gegensatz dazu ist ein Klasse-3 Chipkartenterminal (mit Display und Tastatur) direkt mit dem Rechner verbunden und kann somit eine digitale Unterschrift an den Rechner übertragen. Die Flexibilität, die von einem Klasse-3 Chipkartenterminal zu fordern ist, dass verschiedene Anwendungsbereiche unterstützt werden können, führt zu einer hohen Komplexität und dementsprechend zu einem hohen Preis dieser Geräte.

In diesem Beitrag zeigen wir, wie durch eine Reduktion der Funktionalität eines Chipkartenterminals auf das minimal Notwendige die Implementierung klein gehalten und die Sicherheit gegen Manipulationen aus dem Internet formal verifiziert werden kann.

2 Nürnberger Anti-Phishing-Device

Mit dem Nürnberger Anti-Phishing-Device wurde ein Chipkartenterminal und ein Verfahren zur Erstellung von digitalen Unterschriften vorgestellt, das sich durch Einfachheit auszeichnet [1].

Die wirtschaftlich sichere, digitale Unterschrift besteht aus zwei Teilen: einer Transaktionszusammenfassung, die die wirtschaftlich relevanten Daten wie Betrag und Zielkonto in einem kurzen Text ohne Formatierungen enthält und einer digitalen Unterschrift über dieser Transaktionszusammenfassung. Neben der Prüfung der digitalen Unterschrift wird nun zusätzlich geprüft, ob die Transaktionszusammenfassung zu der gesendeten Transaktion passt. Eine Integration dieses Verfahrens als *User Defined Signature* in den FinTS Standard ist realisierbar [1].

Das Nürnberger Anti-Phishing-Device wurde mit einem Arduino Micro Pro Mikrocontroller, einem Nokia 5110 Display, einer PIN-Tastatur mit 12 Tasten und einer Chipkarten-Kontaktiereinheit auch für den Einsatz in Mobiltelefonen entwickelt [2].

Eine Transaktion läuft mit dem Nürnberger Anti-Phishing-Device wie folgt ab: Der Rechner sendet eine Protokollnachricht mit einer Transaktionszusammenfassung an das Chipkartenterminal, das diese auf Wohlgeformtheit prüft. Im negativen Fall wird eine Antwort mit einem Fehlercode an den Rechner zurückgegeben. Im positiven Fall wird die Transaktionszusammenfassung im Display angezeigt und der Kunde gibt seine Entscheidung über Richtigkeit der Transaktionszusammenfassung über die PIN-Tastatur ein. Im negativen Fall wird ein Fehlercode an den Rechner gemeldet. Im positiven Fall wird der PIN-Code abgefragt und an die Chipkarte zur Überprüfung gesendet. Bei Fehleingabe wird zur erneuten Eingabe des PIN-Code aufgefordert, alle anderen Fehler werden an den Rechner weitergeleitet. Bei erfolgreicher PIN-Eingabe wird die Transaktionszusammenfassung an die Chipkarte zur Signatur gesendet. Die Antwort der Chipkarte wird direkt an den Rechner zurück geschickt.

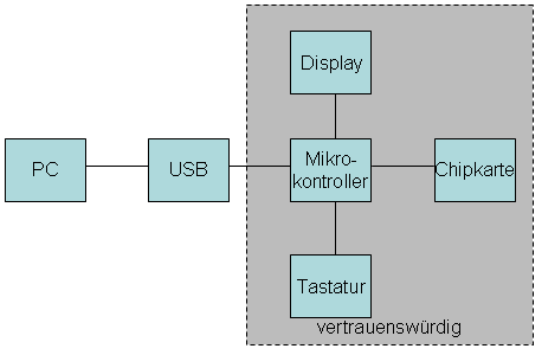

Abbildung 1: Blockdiagramm des Chipkartenterminals

3 Verifikation

3.1 Abgrenzung

Die Verifikation beschränkt sich auf die Software, die auf dem Mikrocontroller läuft. Die USB-Implementierung des Arduino wird nicht in die Trusted Computing Base einbezogen sondern auch als potentiell kompromittiert betrachtet (siehe Abbildung 1. Der Mikrocontroller kommuniziert mit der USB-Implementierung über eine Eingabeleitung und nur Daten, die über diese Leitung transportiert werden, werden betrachtet.

Der Arduino enthält einen Bootloader, der nach dem Starten bzw. Rücksetzen des Mikrocontroller kurz wartet, ob ein neuer Programmcode installiert werden soll. Da der Bootloader vor der installierten Software ausgeführt wird, muss dieser entfernt werden, sodass ein Angreifer die Software auf dem Mikrocontroller nicht ersetzen kann. Die Programmierung des Arduino muss also über die andere Programmierschnittstelle bewerkstelligt werden. Diese Schnittstelle ist im produktiven Betrieb nicht mit dem PC verbunden. Ob der Bootloader tatsächlich entfernt wurde, wird im Rahmen dieser Untersuchung nicht nachgewiesen.

Die Sicherheit der digitalen Unterschrift, die durch die Chipkarte erzeugt wird ist nicht Gegenstand dieser Untersuchung. Zudem kann das Verfahren zur Erstellung der digitalen Unterschrift durch den Einsatz einer anderen oder aktualisierten Chipkarte geändert werden. Wir gehen also im Folgenden davon aus, dass das verwendete digitale Unterschriftsverfahren nach dem Stand der Technik als ausreichend sicher einzustufen ist.

3.2 Sicherheitsziele

Durch die minimale Funktionalität des Chipkartenterminals sind die Manipulationsmöglichkeiten eines Angreifers eingeschränkt. Ein Angreifer kann lediglich Nachrichten an das Chipkartenterminal senden. Diese Nachrichten können beliebig lang sein, beliebigen Inhalt haben und zu jeder beliebigen Zeit gesendet werden. Der Beweis der Sicherheit des Chipkartenterminals besteht also darin, zu zeigen, dass es einem solchen Angreifer nicht gelingt durch das Senden unzulässiger Nachrichten, eine Transaktion signieren zu lassen, die der Benutzer nicht (nach vorheriger Prüfung) freigegeben hat.

Zur weiteren Analyse wird die Klassifikation unbeabsichtigter Schwachstellen in der Softwareentwicklung nach Pfleeger und Lawrence Pfleeger [3] verwendet. Drei Klassen unbeabsichtigter Schwachstellen werden unterschieden:

1. Buffer Overflow: Eingabedaten sind größer als der reservierte Speicher.
2. Incomplete Mediation: Eingabedaten werden interpretiert und dabei werden Verstöße gegen das Format der Daten nicht oder nicht adäquat behandelt.
3. Serialization Flaw: Das Ergebnis der Berechnung hängt von der Reihenfolge ab, in der bestimmte Aktionen ausgeführt werden.

Die folgenden Eigenschaften der Implementierung des Chipkartenterminals sind also für die Klassen von Schwachstellen zu beweisen:

- Buffer Overflow: Die Länge der Protokollnachricht vom Rechner an das Chipkartenterminal muss beschränkt sein. Dies ist zum einen durch das Protokoll nach ISO7816 und zum anderen durch die Maximallänge der Transaktionszusammenfassungen, die nach Hehn et. al. [1] weniger als 200 Zeichen beträgt, gewährleistet. Somit sind alle Nachrichten, die die Maximallänge überschreiten, als ungültig abzuweisen. Zur Speicherung und Verarbeitung gültiger Nachrichten genügt also ein statisch allozierter Puffer. Es muss nachgewiesen werden, dass niemals über das Ende dieses Puffers hinaus geschrieben wird.
- Incomplete Mediation:
  - Nur druckbare Zeichen werden an den Displaytreiber gesendet. Enthält die Protokollnachricht ungültige Zeichen, so wird sie abgewiesen.
  - Nur Protokollnachrichten mit korrektem Header werden verarbeitet, ungültige werden abgewiesen. Es ist nur eine einzige Operation zugelassen, das Signieren einer Nachricht.
- Serialization Flaw: Reihenfolge der Aktionen ist zu garantieren (Anzeige, Freigabe, Signieren). Insbesondere muss sichergestellt sein, dass ein Angreifer nicht, während der Kunde die Transaktionszusammenfassung am Display überprüft, eine zweite, manipulierte Transaktionszusammenfassung vor dem Signieren senden und damit die vom Kunden geprüfte ersetzen kann.

3.3 Arduino API in Frama-C

Frama-C ist eine Plattform zur Verifikation von C-Programmen im Source-Code. Der Source-Code wird mit Annotationen in der Sprache ACSL angereichert. ACSL ist eine Sprache zur formalen Spezifikation des Verhaltens einer Funktion (behavioral interface specification language). Da ACSL Spezifikationen in C-Kommentaren liegen, können annotierte Dateien weiterhin mit einem Standard C-Compiler verarbeitet werden. [4]

Die Bibliotheken des Arduino stehen in der Regel in C++ zur Verfügung und die Sketches des Arduino werden von der IDE um wenige Codezeilen erweitert an den C++-Cross-Compiler weitergegeben. Inspiriert von Burghardt et. al. [5], die Algorithmen der Standard Template Library in C implementiert und mit Frama-C verifiziert haben, wurde für die benötigten Arduino-Bibliotheken eine C-Schnittstelle definiert. Dieses Vorgehen wird zusätzlich dadurch gerechtfertigt, dass in der Implementierung des Programms für den Mikrocontroller keine objektorientierten Mechanismen benötigt werden.

Die Schnittstelle einer Arduino-Bibliothek enthält zum einen die ACSL-Spezifikation der C-Schnittstelle und zum anderen Makro-Ersetzungen, die den entsprechenden C++ Code erzeugen. Für jede Methode wird eine C-Funktion erzeugt, die als ersten Parameter einen Zeiger auf den Zustand des Objekts enthält. Für jeden Konstruktor gibt es eine C-Funktion, die eine neue Instanz erzeugt und in C++ dann durch `new` und den Konstruktor und seine eventuell vorhandenen Argumente ersetzt wird.

3.4 Einhaltung der Puffergröße

Beim Empfang einer Protokollnachricht vom Rechner muss sichergestellt werden, dass diese nicht größer als der Puffer ist. Sendet der Rechner eine zu große Protokollnachricht, dann muss ein Fehlercode an den Rechner zurückgegeben werden.

Das Verhalten der Empfangsfunktion wird zum einen durch die Vorbedingungen (preconditions) und die Nebenwirkungen (side effects) und zum anderen durch den Rückgabewert spezifiziert. Die Vorbedingungen sind, dass der Puffer gültig ist, also aus alloziertem Speicher besteht und, dass die Anzahl der einzulesenden Zeichen nicht größer als die Anzahl der Plätze im Puffer ist. Als Nebenwirkung wird dann genau der entsprechenden Anzahl der Plätze im Puffer die Eingabe vom Rechner zugewiesen. Im Fall, dass die Anzahl der einzulesenden Zeichen zu groß ist, wird ein Fehler zurückgegeben und keine Zuweisung im Puffer vorgenommen.

Ebenso wird eine Nachricht, die nicht dem geforderten Nachrichtenformat entspricht oder ungültige Zeichen enthält mit einer Fehlermeldung an den Rechner zurückgewiesen.

Mit der in Frama-C eingebauten automatischen Entscheidungsprozedur oder *alt-ergo* können alle erforderlichen Beweise automatisch geliefert werden.

3.5 Korrektheit der Anzeige

Das im Prototypen verwendete Display wird pixelweise angesprochen. Daher wird die Ausgabe der Zeichen durch das Nachschlagen in einer Tabelle und das Senden der entsprechenden Bitmuster umgesetzt. Der Code des Display-Treibers prüft die Gültigkeit der Zeichen nicht, sondern greift dann auf andere Bereich des Speichers des Mikrocontrollers zu. Als Vorbedingung für die Schreibfunktion des Display wird daher gefordert, dass sich alle Zeichen in der auszugebenden Zeichenkette im Wertebereich der Zeichentabelle befinden.

Im Verlauf der Prüfung der eingelesenen Protokollnachricht wurde bereits sichergestellt, dass die zu signierende Zeichenkette keine ungültigen Zeichen enthält. Folglich wird als ein Teil der Vorbedingung der Anzeigefunktion die Gültigkeit der in der Zeichenkette enthaltenen Zeichen sein.

Die gültigen Zeichen sind alle auf dem Display darstellbar, jedoch müssen die Zeichencodes der Umlaute umgerechnet werden. Dafür wird eine Umrechnungsfunktion eingeführt. Die Korrektheit dieser Umrechnungsfunktion zusammen mit der Ausgabefunktion des Display muss visuell auf dem Display durch eine Testzeichenkette, die alle zulässigen Zeichen enthält, nachgeprüft werden.

Dann verbleibt noch zu zeigen, dass aus der Prüfung der eingelesenen Protokollnachricht folgt (Vorbedingung), dass die mit der Umrechnungsfunktion umgerechnete Zeichenkette nur Zeichen im Wertebereich der Tabelle des Display-Treiber enthält. Auch dieser Nachweis gelingt mit den automatischen Entscheidungsprozeduren.

3.6 Beweis der Einhaltung der Reihenfolge

Zum Nachweis des zeitlichen Verhaltens eines C-Programms liefert Frama-C das Aoraï Plugin mit. Aoraï bietet drei Möglichkeiten der Spezifikation [6] des Verhaltens an: Büchi-Automat, LTL oder PROMELA.

Für dieses Projekt wurde die Spezifikation als Büchi-Automat in der Aoraï eigenen Sprache YA gewählt. Jeder Zustand des Chipkartenterminals kann durch einen Zustand im Büchi-Automaten in Abbildung 2 repräsentiert werden. Der Start und der akzeptierende Zustand ist dabei der Zustand “R”, der die Empfangsbereitschaft darstellt. Der Zustand “D” bedeutet Anzeigen, der Zustand “P” PIN-Eingabe und der Zustand “S” Signieren.

Zustandsübergänge finden durch Funktionsaufrufe statt. Das bedeutet, dass eine Änderung des Zustandes innerhalb einer Funktion nur stattfinden kann, wenn aus der Funktion heraus eine andere Funktion aufgerufen wird. Im betrachteten Chipkartenterminal wurde der Programmcode von Anfang an in einzelne

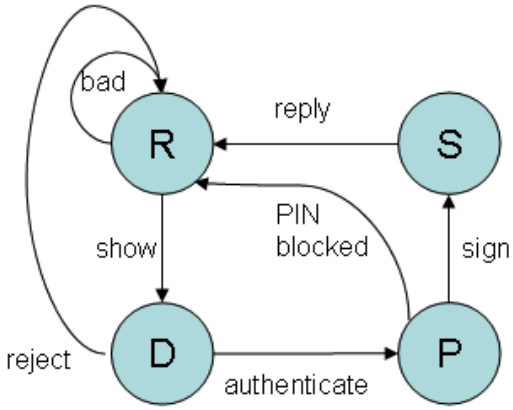

Abbildung 2: Büchi-Automat des Mikrocontroller-Code

Funktionen strukturiert, sodass diese Einschränkung keine Bedeutung hatte.

Bei der Spezifikation des Büchi-Automaten werden die Funktionen, die einen Zustandsübergang bewirken, angegeben. Die übrigen Funktionen werden nicht in den Beweis des korrekten Verhaltens mit einbezogen. Das bedeutet, dass die Funktionen, die zum Empfangen, Anzeigen, PIN-Eingeben und Signieren dienen und nur in bestimmten Zuständen aufgerufen werden dürfen, in der Spezifikation genannt werden müssen. Wir nennen diese Funktionen *kritische* Funktionen.

Bei genauerer Betrachtung zeigt sich, dass es nicht genügt, nur die Zustände aus Abbildung 2 zu verwenden, wenn gewährleistet sein soll, dass die nächste Funktion erst nach dem Abschluss der vorangehenden Funktion aufgerufen werden darf. Z. B. darf erst nach dem Abschluss des Empfangs der Nachricht vom PC diese Nachricht im Display angezeigt werden. Daher werden für jeder Zustand x aus Abbildung 2 zwei Zustände x_{run} und x_{end} definiert. Zu Beginn der Funktion betritt der Büchi-Automat den Zustand x_{run} aus dem es nur einen Zustandsübergang zum Zustand x_{end} gibt, der beim Verlassen der Funktion ausgelöst wird. Damit ist sichergestellt, dass innerhalb einer kritischen Funktion keine andere kritische Funktion ausgeführt werden kann. Jeder Zustandsübergang von x nach y aus dem Büchi-Automaten wird dann durch einen Zustandsübergang von x_{end} nach y_{run} spezifiziert.

Da die Auswertung der Ergebnisse der Funktionen bereits Teil der Verhaltensspezifikation sind, wurde darauf verzichtet, diese nochmals im Büchi-Automaten zu modellieren und als Guards in die Zustandsübergänge einzubeziehen. Da der Büchi-Automat in einer separaten Datei erstellt wird, erschien es sinnvoller, die Ablaufsteuerung in der C-Datei zu belassen und mit Aoraï lediglich zu zeigen, dass die geforderte Reihenfolge der kritischen Funktionen eingehalten wird.

Anders als bei der Verhaltensspezifikation wird die Spezifikation des Automaten als separate Datei erstellt. Dann wird das C-Programm und die Spezifikation des Automaten durch Aoraï in eine neue C-Datei mit erweiterten ACSL Annotationen überführt, die dann mit den Frama-C-Werkzeugen und den Entscheidungsprozeduren verifiziert wird. Da die Verifikation sich auf die Prüfung der korrekten Zustandsübergänge beschränkt und keine weiteren Guards auszuwerten sind, können alle Beweise von der eingebauten Entscheidungsprozedur erbracht werden.

4 Zusammenfassung und Ausblick

In diesem Beitrag wurde gezeigt, wie durch die Einführung der wirtschaftlich sicheren, digitalen Unterschrift die Anforderungen an ein Chipkartenterminal mit Display und Tastatur (Klasse-3) sehr vereinfacht werden können. Durch das Ausschließen der Kommunikationsverbindung (hier USB) mit dem Rechner aus der Trusted Computing Base, wurde das zu verifizierende System weiter verkleinert. Schließlich wurde die Beobachtung, dass der Code keine speziellen objektorientierten Eigenschaften der Sprache C++ genutzt, die Verifikation in der deutlich einfacheren Sprache C durchführbar. Das Projekt zeigt, dass Vereinfachung der Funktionalität formale Beweise von Sicherheitseigenschaften ermöglicht.

Das vorgestellte Nürnberger Anti-Phishing Device ist neben dem Online-Banking auch in anderen Bereichen einsetzbar. Einzige Voraussetzung ist, dass sich die zu schützende Information in einer Nachricht von ca. 200 Zeichen zusammenfassen lässt. Dabei kann untersucht werden, wie sich electronic Commerce oder electronic Government Transaktionen schützen lassen. Eine weitere Möglichkeit ist, die Chipkartenkontaktiereinheit durch eine RFID Antenne zu ersetzen und damit den neuen Personalausweis zur digitalen Unterschrift zu verwenden.

Literatur

[1] Sabine Hehn, Mario Brittig, Ramon Haluf, Markus Harrer, Felix Schmidt, Christian Weyermann, Peter Trommler. Einfache, wirtschaftlich sichere Signaturen. In Patrick Horster, Peter Schartner (Hrsg.) DACH Security 2009, syssec Verlag, 2009.

[2] Peter Trommler. Protecting Online Banking on a Smartphone with Signed Transaction Summaries. In: Ajith P. Abraham, Antonio Palma dos Reis, Jörg Roth (Hrsg.), Proceedings of the International Conference on Theory and Practice in Modern Computing, Las Palmas de Gran Canaria, Spanien, 2015.

[3] Charles P. Pfleeger, Shari Lawrence Pfleeger. Security in Computing. 4. Auflage, Prentice Hall, Uppser Saddle River, NJ, U.S.A., 2007.

[4] Loïc Correnson, Pascal Cuoq, Florent Kirchner, Virgile Prevosto, Armand Puccetti, Julien Signoles, Boris Yakobowski. Frama-C User Manual. Release Magnesium-20151002, Oktober 2015. URL: http://frama-c.com/download/user-manual-Magnesium-20151002.pdf

[5] Burghardt, J. et. al. 2015. ACSL By Example, Towards a Verified C Standard Library. Fraunhofer FOKUS Research Report.

[6] Nicolas Stouls, Virgile Prevosto. Aoraï Plugin Tutorial. Oktober 2015. URL: http://frama-c.com/download/aorai-manual-Magnesium-20151002.pdf