Tobias Nickl
Benjamin Weigl
Prof. Dr. Andreas Aßmuth

Bedrohungen durch Scheduling-basierende Angriffsszenarien im Automotive-Bereich

Zusammenfassung

Durch die stetige Vernetzung aller Systeme und Dinge (engl. Internet of Things, IoT) sowie die Digitalisierung physischer Systeme (engl. cyber-physical system) im Automobil-Bereich soll dem Passagier ein intuitives Fahrerlebnis ermöglicht werden. Sensoren, Aktoren und mobile Endgeräte sind hierzu miteinander vernetzt. Während bisher zahlreiche Angriffe, vom unautorisierten Öffnen des Fahrzeugs bis hin zur unbefugten Nutzung von Bedienelementen (z. B. Scheibenwaschanlage), durch Schwachstellen im System demonstriert wurden, zeigt diese Arbeit eine neue Art von Angriffen auf. Die immer größer werdende Zahl an Steuerungsaufgaben bringt Single-Core-Systeme an ihre Grenzen. Multi- bzw. Many-Core-Systeme sollen Abhilfe schaffen und beim Hersteller Kosten einsparen. Ein besonderes Gefahrenpotenzial birgt dabei das notwendige Scheduling, das u. a. Safety- und zukünftig auch Security-Funktionen Rechenzeit zuweist. Angriffe auf das Scheduling können im Automobilbereich das Echtzeitverhalten beeinträchtigen und somit Safety-Mechanismen wie das Auslösen des Airbags oder der Bremse verzögern. Im folgenden Aufsatz werden Scheduling-basierende Angriffsszenarien vorgestellt sowie deren Funktionsweise erläutert.

Abstract

To provide an intuitive experience for the driver, connecting objects and devices (Internet of Things IoT) as well as the digitalization of physical systems (cyber-physical system) are becoming prevalent in the automotive industry. Therefore, actuators, sensors and devices in the car are interconnected with the infotainment system. In recent years, cyber security researchers showed the vulnerabilities of cars for cyber attacks. All these attacks have in common that they can perform only actions which are provided by or implemented in the system, e.g. running the windshield wipers. In this paper we describe a new way of potential attacks based on the scheduling system of a control unit. Due to the ever-increasing amount of control tasks in a vehicle single-core systems are approaching their limits. For this purpose, multi- and many-core systems will be used in future. The scheduling system which distributes the computing time, e.g. for safety and security functions, is a potential target for future cyber attacks. These type of attacks can affect the real-time behaviour of a system and thereby delay or block the signal to trigger the airbag or the vehicle‘s brakes. In this paper, scheduling-based attack scenarios and their general operation are discussed.

1 Einleitung

Der durchschnittliche Amerikaner verbringt ca. 250 Stunden pro Jahr in seinem Fahrzeug. Ein modernes „connected car“ erzeugt pro Stunde ca. 25 GB an Daten, darunter Sensor- und Aktuatordaten, Steuerungsinformationen für die Fahrzeugassistenz oder auch mediale Inhalte des Infotainmentsystems [1]. Bestand die Aufgabe einer zentralen Fahrzeugsteuerung (engl. electronic control unit, ECU) in der Vergangenheit darin, einfache Signale und Prozesse der Motorsteuerung zu verarbeiten, müssen diese heute ca. 1,3 TB an Daten erzeugen, verwalten und kommunizieren. Um dabei möglichst energieeffizient und somit wirtschaftlich zu sein, rüstet die Industrie nach. Der klare Vorteil von Multi- bzw. Many-Core-Systemen, mit geringerer Taktzahl und dadurch mit geringerer Energieaufnahme bzw. Wärmeabgabe eine größere Menge an Daten effizient bewältigen zu können, soll nun auch in der Fahrzeugsteuerung für mehr Leistung sorgen.

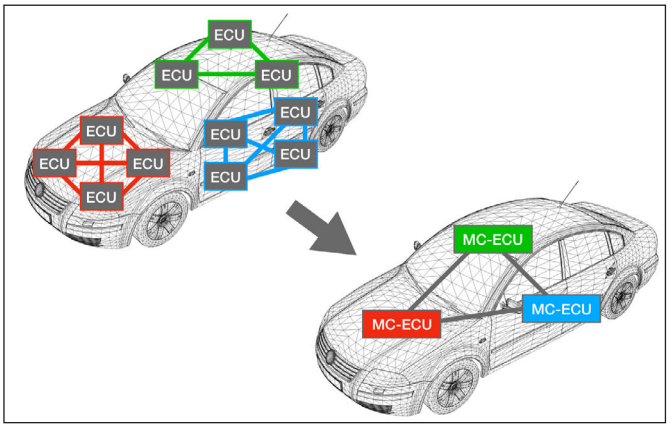

Abbildung 1: Entwicklung der Steuergeräte von Single-Core- zu Multi-Core-Systemen

1.1 FORMUS³IC

Das seit 2015 durch die Bayerische Forschungsstiftung geförderte Projekt „Forschungsverbund Multi-Core Safe and Software-intensive Systems Improvement Community“ (kurz FORMUS³IC) hat es sich zur Aufgabe gemacht, einen ganzheitlichen Ansatz für die Herausforderungen zu finden, die durch den neuen Einsatz heterogener Multi- bzw. Many-Core Architekturen entstehen, und den Automotive- bzw. Avionik-Sektor bei der Einführung dieser Systeme zu unterstützen. Von der formalen Verifikation über die Modellierung des Software-Entwurfs bis hin zur Virtualisierung von Hardware bestreiten die einzelnen Teilprojekte Aufgabenstellungen mit dem Ziel, Scheduling, Energieeffizienz und Sicherheit in Automotive- und Avionik-Applikationen zu verbessern. Im Folgenden soll das Teilprojekt 2, das dieser Arbeit zugrunde liegt, kurz beschrieben werden [2].

1.2 Teilprojekt 2

Im Rahmen des Forschungsprojekts FORMUS³IC ist im Teilprojekt 2 insbesondere der Einsatz von Scheduling-Verfahren in Steuergeräten aus dem Bereich Automotive zentraler Gegenstand der Forschung.
Im Zentrum steht dabei die Architekturbeschreibungssprache EAST-ADL, deren Konzepte in den international etablierten Automotive Open System Architecture (AUTOSAR) Standard einfließen. EAST-ADL sieht jedoch bisher noch keine Möglichkeit vor, Multi-Core-Eigenschaften zu berücksichtigen. Ziel ist es, diese domänenspezifische Architekturbeschreibungssprache um spezielle Multi-Core-Attribute zu erweitern.

1.3 Security-Beitrag zu Teilprojekt 2

Der Security-Beitrag aus Amberg behandelt die Analyse von Angriffen auf das Scheduling-Verfahren für stark zeitkritische Komponenten. Dabei sollen die Gefahren und Konsequenzen durch die Kompromittierung des Schedulers im Automotive-Umfeld untersucht werden. Ferner sollen geeignete Gegenmaßnahmen zur Vermeidung solcher Angriffe aufgezeigt werden.

2 Grundlagen

Um ein grundlegendes Verständnis für die Sicherheitslage und Angriffsszenarien zu erhalten, werden in diesem Kapitel die Grundlagen erklärt, um anschließend darauf aufbauend Angriffe auf das Scheduling zu erläutern.

2.1 Safety

Die Sicherheit im Sinne der Unfallvermeidung (engl. safety) ist der Schutz vor Systemausfällen, z. B. der Technik im Fahrzeug. Dem Fahrer werden hierzu Hilfsmittel bereitgestellt, die ihn in gefährlichen Situationen warnen und ggf. unterstützen. Durch diese Maßnahmen sollen Unfälle verhindert bzw. deren Auswirkungen verringert werden [3].

2.2 Security

Die Sicherheit im Sinne der Kriminalprävention (engl. security) wird als Schutz vor Angriffen auf das technische System bezeichnet. Die Security schützt dabei nicht den Menschen vor der Maschine, sondern die Maschine wird vor böswilligen Eingriffen wie z. B. dem nicht vorgesehenen Abschalten von Safety-Funktionen geschützt [3].

2.3 Safety und Security im Automotive-Bereich

Bisher lag der Fokus im Automotive-Bereich verstärkt auf der funktionalen Sicherheit (safety) des Systems. So sind Kommunikationsnetzwerke und Steuergeräte umfassend gegen technische Ausfälle oder Umwelteinflüsse geschützt, insbesondere bezüglich hoher Verfügbarkeit, Redundanz und Fehlertoleranz kritischer Komponenten [4] [5]. Die Security hingegen spielte speziell bei Steuergeräten im Zusammenhang mit eingebetteten Systemen bislang keine große Rolle. Dies begründet sich damit, dass die Systeme aus individuell konfigurierten Komponenten bestanden, physikalisch voneinander getrennt und unabhängig waren.

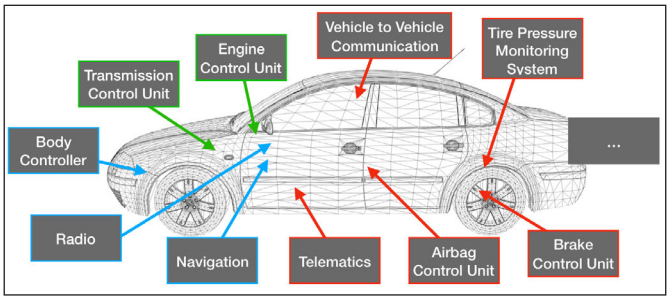

Abbildung 2: Beispiele verschiedener Steuerungsaufgaben in einem modernen Kraftfahrzeug

Durch den Einzug des IoT in das moderne Fahrzeug, das Bedürfnis zur Fernüberwachung sowie neue Steuerungsmöglichkeiten, z. B. eine Handy-App zur Fernsteuerung von Fahrzeugfunktionen, wurden unbeabsichtigt auch neue Schwachstellen und Risiken geschaffen. Darüber hinaus basieren heutzutage die einzelnen Systeme meist

auf sog. Commercial-off-the-shelf- (COTS-)Komponenten, die miteinander vernetzt sind. COTS sind hingegen keine speziellen individuellen Komponenten, sondern serienmäßig gefertigte Produkte, die in großer Stückzahl produziert werden. Der Vorteil vom Einsatz solcher COTS liegt hauptsächlich in der Kostenreduzierung. Der Nachteil hingegen liegt in integrierten Funktionen und Schnittstellen, die nicht immer benötigt werden und serienmäßig schlecht bis gar nicht abgesichert sind. Durch den großflächigen Einsatz solcher Komponenten nimmt auch das Interesse an Angriffen zu. Zudem entwickeln sich auch die Angriffstechniken weiter und werden immer ausgeklügelter. So zeigten u. a. der Computerwurm Stuxnet [6] sowie die Malware Brutal Kangaroo [7] auf, wie physikalisch voneinander getrennte Systeme, die als besonders sicher galten, infiziert werden können. Bisherige Angriffe im Automotive-Bereich umgingen oder kompromittierten die unzureichenden Schutzvorkehrungen, um Funktionen auszuführen, die auch der Fahrer selbst steuern kann. Weitaus schlimmer wäre es, wenn die Safety- sowie die Security-Mechanismen eines Fahrzeugs durch einen Angreifer direkt manipuliert werden könnten.

2.4 Von eingebetteten Single- zu Multi-Core-Systemen

Bei eingebetteten Systemen handelt es sich um spezielle Computersysteme, die in eine technische Umgebung wie z. B. das Automotive-Umfeld eingebunden sind. Die Hauptaufgabe ist dabei die Überwachung und Steuerung von physikalischen Prozessen [8]. Infolge der zunehmenden Leistungsfähigkeit der Hardware übernehmen diese mehr Steuerungsaufgaben, wodurch die Anzahl an zu verarbeitenden Prozessen steigt. Einzelne Prozessoren stoßen hier an ihre Grenzen und werden in der Praxis zunehmend von Multi-Core-Systemen abgelöst (Abbildung 1). Durch die parallele Ausführung von Softwarekomponenten auf mehreren Prozessoren kann der Durchsatz und somit die Leistungsfähigkeit erhöht werden [9]. Jedoch muss bereits bei der Entwurfsphase Rücksicht auf die Verteilung der Softwarekomponenten (siehe Unterabschnitt 2.5) genommen werden. Funktionale sowie nicht-funktionale Anforderungen einer Komponente, z. B. Arbeitsspeicher, Kommunikationsnetzwerk sowie Echtzeit- und Zuverlässigkeitsanforderungen, müssen dabei erfüllt werden.

2.5 Besonderheiten der Softwarearchitektur im Automotive-Bereich

Weitverbreitete Betriebssysteme für eingebettete Systeme sind im Automotive-Bereich u. a. OSEK- und AUTOSAR. Die europäische Organisation OSEK/VDX unterstützt die Vereinheitlichung der Softwarearchitektur für ECUs im Fahrzeugbau. Der AUTOSAR-Standard, entwickelt von einer internationalen Arbeitsgemeinschaft verschiedener Firmen, baut auf den Ansätzen von OSEK auf und stellt die Referenzgrundlage für die Entwicklung von Softwarefunktionen dar. Dabei wird eine Systemfunktion gemäß dem Standard durch AUTOSAR Software Components realisiert, die in der Entwicklungsphase einzelnen ECUs zugeteilt werden können (Abbildung 3). Jede Softwarekomponente besteht wiederum aus einer Menge von

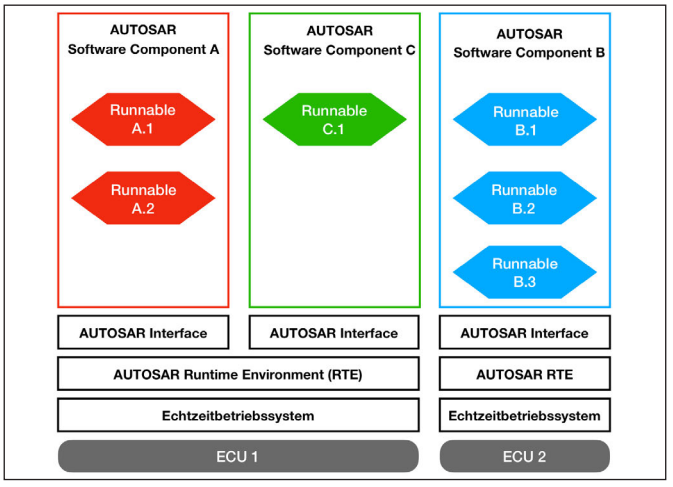

Abbildung 3: Platzierung von AUTOSAR-Softwarekomponenten auf verschiedene ECUs [10]

Runnable Entities, kurz als Runnable bezeichnet. Sie stellt in der AUTOSAR-Architektur die kleinste funktionale Code-Einheit dar, die unabhängig von anderen Runnables ausgeführt und vom Scheduler eingeplant werden kann [11]. Dadurch ist es möglich, Softwarekomponenten verschiedener Hersteller auf unterschiedliche ECUs zu verteilen. Beispielsweise könnte Software Component A zum Auslesen von Sensordaten zuständig sein, die von Software Component C verarbeitet werden (Abbildung 3). Die Software kann dabei von unterschiedlichen Herstellern bereitgestellt werden.

2.6 Prozess-Scheduling in eingebetteten Systemen

Bei der Verteilung einer Softwarekomponente wird zwischen der räumlichen (engl. mapping) und der zeitlichen Dimension (engl. scheduling) unterschieden. Beim Mapping wird die Zuteilung auf eine physikalische Ressource vorgenommen, hingegen wird beim Scheduling die Koordination der konkurrierenden Zugriffe auf die gemeinsam genutzten Ressourcen einer ECU gemanagt.

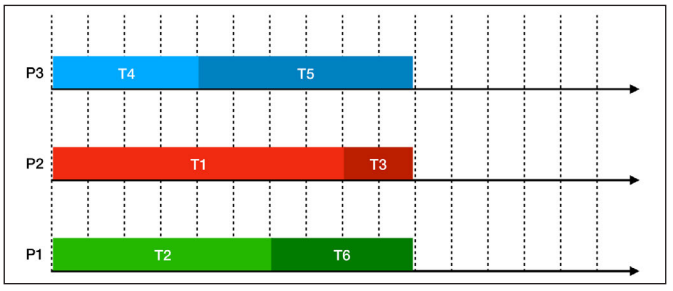

Abbildung 4: Mapping und Scheduling verschiedener Tasks [10]

Der Scheduler in einem System übernimmt die Aufgabe, an einzelne Tasks (Task wird in dieser Arbeit als Synonym für Prozess verwendet) Rechenzeit zu vergeben und deren Zustände (aktiv, blockiert, bereit) zu verwalten [12]. Je

nach Scheduling-Strategie zum Lösen eines Scheduling-Problems wird die Verteilung hinsichtlich verschiedener Gesichtspunkte optimiert, z. B. Fairness, Minimierung der Ausführungsdauer oder Echtzeitverhalten. Des Weiteren wird zwischen unterbrechenden (präemptiv) und nicht-unterbrechenden (kooperativ) Scheduling-Verfahren unterschieden. Präemptiv bedeutet, dass der Scheduler einen Task niedriger Priorität unterbrechen kann und durch einen Task höherer Priorität ablöst. Hingegen kann beim kooperativen Scheduling ein Task nicht unterbrochen werden, bis dieser vollständig abgearbeitet wurde [13].

2.7 Scheduling in Automotive-Umfeld

Im Automotive-Umfeld nimmt die Bedeutung von Fahrerassistenzsystemen (Kollisionsvermeidung, Geschwindigkeitsregelung etc.) signifikant zu. Das Verhalten des Systems muss dabei zu jedem Zeitpunkt der Laufzeit genau bekannt sein, was mithilfe von Echtzeitsystemen realisierbar ist [14]. Echtzeit bedeutet, dass die Aufgabe eines Tasks zu einer bestimmten Deadline fertiggestellt werden muss. Um diese Eigenschaft zu garantieren, kommen spezielle Echtzeitscheduling-Algorithmen zum Einsatz. Zu unterscheiden gilt es hierbei zwischen „harten“ (engl. hard real-time, HRT), „weichen“ (engl. soft real-time, SRT) und „festen“ (engl. firm real-time, FRT) Echtzeitsystemen (Abbildung 5).

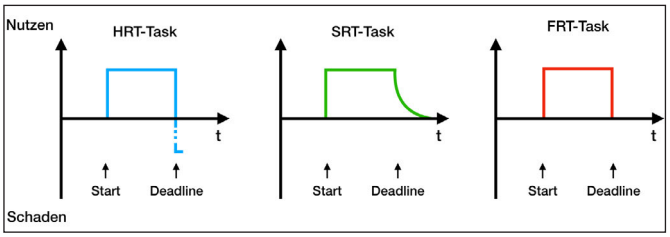

Abbildung 5: Arten von Echtzeitystemen [10]

In „harten“ Echtzeitsystemen kann ein Fehler im Werte- und Zeitbereich zu katastrophalen Folgen führen, z. B. zur Zerstörung des Systems oder zur Gefährdung von Menschenleben. „Fest“ hingegen bedeutet, dass das Verstreichen eines Termins keinen unmittelbaren Schaden erzeugt, jedoch ist das Ergebnis wertlos. In „weichen“ Echtzeitsystemen sinkt mit verstrichener Deadline der Wert des ermittelten Ergebnisses, z. B. bei einem verzögerten Sprachanruf über das Internet [15]. In Echtzeitsystemen bzw. Echtzeitbetriebssystemen werden in der Regel präemptive Verfahren eingesetzt, die entweder offline mit statischen Prioritäten vor der Laufzeit oder online mit dynamischen Prioritäten zur Laufzeit arbeiten. Dabei wird immer der Task mit der höchsten Priorität vom Scheduler bevorzugt. Leistungsfähige Assistenzsysteme erfordern oftmals, dass Sensordaten in Echtzeit wiederholt abgefragt werden. Dies führt zwangsläufig zu prioritätsgesteuerten Tasks, die fortlaufend und periodisch verarbeitet werden müssen. Ein periodischer Task $\tau_i = (C_1, T_i)$ ist vereinfacht $(T_i = D_i)$ durch seine maximale Ausführungszeit (engl. worst case execution time, WCET) C_i, seine Periode T_i und die Deadline D_i definiert. Liu und Layland [16] zeigten 1973 mit den Scheduling-Verfahren Rate Monotonic (RM) und Earliest Deadline First (EDF) auf, dass eine Menge an unabhängigen periodischen Tasks $\Gamma = (\tau_1, ..., \tau_n)$ für eine natürliche Zahl n unter „harten“ Echtzeitbedingungen realisierbar ist, falls die notwendige Bedingung

$$u = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$$

erfüllt wird. Falls der Auslastungsfaktor $u > 1$ ist, gibt es für den Scheduler keinen zeitlich zulässigen Ablaufplan. Für $u < 1$ treten sog. Leerlauftasks (engl. idle tasks) auf, in denen der Prozessor unbeschäftigt ist. Die hinreichende Bedingung ist wiederum vom gewählten Scheduling-Verfahren abhängig. Falls alle Tasks ausgeführt werden können, ohne die vorgegeben zeitlichen Bedingungen zu verletzen, ist die Taskmenge vom Scheduler zeitlich planbar.

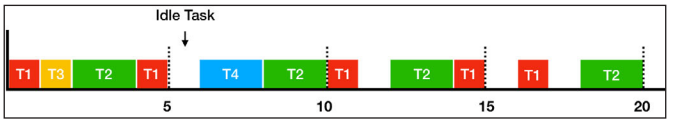

Abbildung 6: Echtzeit-Scheduling von periodischen Tasks [10]

2.8 Memory Management in Multi-Core-Systemen

Um die Performance in modernen Multi-Core-Systemen zu steigern, werden Caches in verschiedenen Hierarchiestufen eingesetzt. Dabei erhöhen sich mit zunehmender Ebene der Speicherhierarchie die Cache-Zugriffszeiten. Abhängig von der Architektur besitzt jeder Kern eines Prozessors fest zugewiesenen Cache-Speicher (Level-1, opt. Level-2), der nur exklusiv vom Kern selbst verwendet werden kann. Um jedoch zwischen den einzelnen Kernen kommunizieren zu können, teilen sich diese den Last-Level-Cache (Abbildung 7).

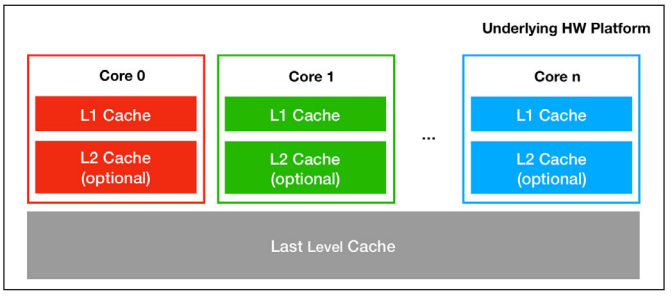

Abbildung 7: Cache-Hierarchie in Multi-Core-Systemen

Im Cache werden Kopien von Teilen der im Hauptspeicher abgelegten Daten verwaltet. Diese Kopien werden vom Cache-Controller in Form von Cache-Paketen (engl. cache line) fester Länge aus dem Hauptspeicher in den Cache geladen. Stellt die CPU eine Anfrage zu einem Eintrag, überprüft der Cache-Controller, ob dieser im Cache gespeichert ist. Kann der Wert aus dem Cache geladen werden, so spricht man von einem Cache-Treffer (engl. cache hit). Muss der Cache-Controller hingegen den

Wert erst aus dem Hauptspeicher in den Cache laden, so spricht man von einem Cache-Fehlzugriff (engl. cache miss). Die relativ hohe Zugriffszeit auf den Hauptspeicher führt dabei zu Wartezeiten [17].

3 Scheduling-basierende Angriffsszenarien

Im Folgenden werden Scheduling-basierende Angriffe auf Multi-Core-Systeme aufgezeigt. Die Angriffe zielen auf die Sicherheit im Sinne der Security (siehe Unterabschnitt 2.2) eines Systems ab und kompromittieren die aus der Informationssicherheit bekannten klassischen Schutzziele Vertraulichkeit, Verfügbarkeit und Integrität. Eingriffe dieser Art auf die Security können u. a. Einfluss auf die Safety eines Systems haben.

3.1 Angriffe auf das Schutzziel Verfügbarkeit

Der Einsatz von Multi-Core-Prozessoren im Automotive-Bereich birgt neue Gefahren durch das Scheduling und die gemeinsam genutzten Ressourcen (vgl. Unterabschnitt 2.8). Abhängig vom Zielsystem und den Absichten des Angreifers können z. B. Nachrichten im System verändert oder die Verarbeitung von Sensordaten sowie die Steuerung von Aktoren manipuliert bzw. gestört werden. Durch die deterministische Eigenschaft von „harten“ Echtzeitsystemen mit statischen Prioritäten (siehe Unterabschnitt 2.7) ist es für den Angreifer möglich, das Scheduling eines Systems zu rekonstruieren [18]. Anhand des rekonstruierten Scheduling-Plans kann der Angreifer gezielt Tasks einschleusen, um das Scheduling zu beeinflussen. Abbildung 8 zeigt das Scheduling zweier Echtzeit-Tasks, die beide ihre Deadline-Vorgaben erfüllen.

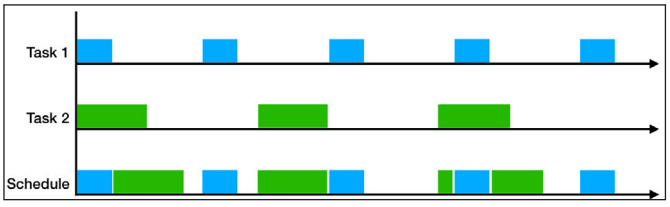

Abbildung 8: Ablaufplan eines präemptiven Echtzeit-Schedulings

Durch das Einbringen eines weiteren periodischen Tasks durch einen Angreifer (gekennzeichnet als „Attacker“ in Abbildung 9) ist es möglich, dass Deadline-Vorgaben anderer Tasks verfehlt werden. Somit könnten gezielt Deadlines von bestimmten Tasks manipuliert oder das ganze System blockiert (engl. denial of service, DoS) werden. Angriffe dieser Art können zur Gefährdung der Betriebssicherheit im System führen. Im Automotive-Bereich könnte dies bedeuten, dass sich z. B. Fahrerassistenzsysteme fehlerhaft verhalten oder im Fahrzeug moderne Bremssysteme mit Brake-by-Wire-Technologie, bei denen der Bremsimpuls nicht über hydraulischen Leitungsdruck, sondern elektronisch durch ein Steuersignal übertragen wird, nicht mehr funktionieren.

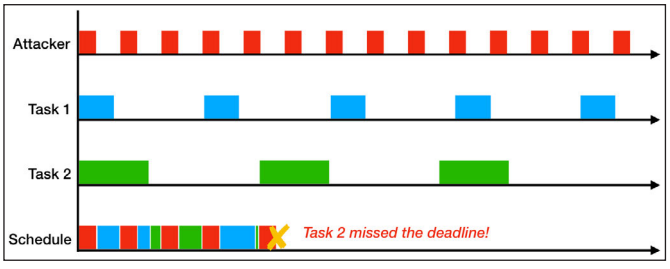

Abbildung 9: Beeinträchtigung des präemptiven Echtzeit-Schedulings durch einen Angreifer

3.2 Angriffe auf das Schutzziel Integrität

Statt die Betriebsfähigkeit eines Systems zu stören, kann ein Angreifer die Absicht verfolgen, Daten und Steuersignale zu manipulieren. Dabei muss zuerst das System verdeckt ausspioniert werden, um Informationen für den weiteren Angriff zu sammeln. Dieses Verhalten konnte u. a. auch beim Computerwurm Stuxnet beobachtet werden, der mehrere Monate verdeckt im Hintergrund agierte [6]. Um während eines Angriffs unerkannt zu bleiben, ist es wichtig, dass sich der gewohnte Ablauf des Systems nicht verändert. Dabei besteht die Möglichkeit, die beim Scheduling verwendeten Idle Tasks (siehe Unterabschnitt 2.7) zu manipulieren [18] und somit durch Seitenkanalangriffe wichtige Informationen zu gewinnen [19].

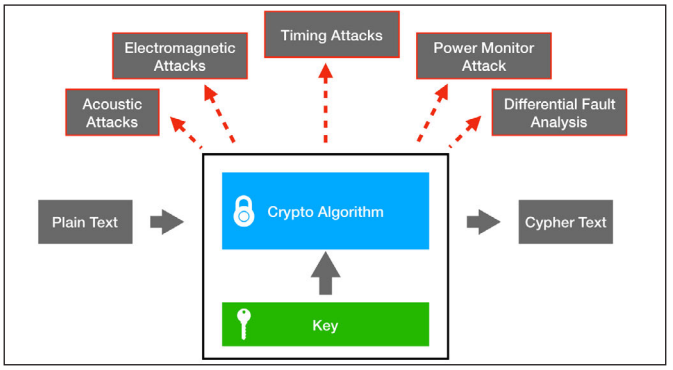

Abbildung 10: Mögliche Seitenkanalangriffe auf ein System

Durch Seitenkanalangriffe wie z. B. Cache-Timing-Angriffe ist es möglich, Informationen zur weiteren Kompromittierung des Systems abzugreifen [20]. So ist es aufgrund gemeinsam genutzter Ressourcen wie z. B. dem Last-Level-Cache (vgl. Unterabschnitt 2.8) möglich, Zeitunterschiede zwischen einem Cache-Hit und einem Cache-Miss zu messen. Folgendes Beispiel soll den Ablauf eines Cache-Timing-Angriffs darstellen. Dabei wird angenommen, dass sich ein böswilliger Task A und ein Task B mit vertraulichen Daten denselben Last-Level-Cache teilen.

1. Task A belegt den gesamten Cache des Systems mit eigenen Daten.
2. Task B führt eine Funktion mit vertraulichen Daten aus, wodurch die benötigten Daten in den Cache geladen werden müssen.
3. Task A fordert die zuvor in den Cache geschriebenen Daten wieder vom System an und misst die einzelnen Ladezeiten.

Durch diesen Angriff lernt Task A, welche Cache-Adressen von Task B verwendet wurden. Ideal wäre es für den Angreifer, wenn der Prozessor den böswilligen Task A abwechselnd mit dem Task B ausführt. Mithilfe dieser Angriffsmethode wurde in [21], [22] aufgezeigt, wie AES-Verschlüsselungsverfahren kompromittiert werden können. Speziell durch die gemessenen Zugriffszeiten auf einzelne Cache-Lines kann geschlussfolgert werden, welche Tabellenfragmente der AES-Verschlüsselung zu welchem Zeitpunkt in den Cache geladen wurden. Dadurch ist es möglich, den verwendeten AES-Schlüssel zu rekonstruieren und die Verschlüsselung zu brechen. Im Automotive-Bereich könnten somit verschlüsselte Nachrichten auf einem Kommunikationskanal entschlüsselt werden, um diese zu manipulieren. Dabei könnte es sich bspw. auch um eine Nachricht mit personenbezogenen Daten des Fahrers handeln, für die besondere datenschutzrechtliche Sicherheitsmaßnahmen (z. B. Verschlüsselung) notwendig sind.

3.3 Angriffe auf das Schutzziel Vertraulichkeit

Des Weiteren kann ein Angreifer das Ziel verfolgen, Daten vom System verdeckt abzugreifen. Durch diese Methode können z. B. vertrauliche Daten von einem Task zu einem anderen Task übertragen werden, obwohl diese keine direkte Möglichkeit zur Kommunikation besitzen. Anhand des Scheduling-Plans kann der Angreifer gezielt Tasks einschleusen, um verdeckt zu kommunizieren. Bei prioritätsbasierenden Scheduling-Algorithmen wie z. B. Rate Monotonic wird die Abhängigkeit zwischen Priorität und Periode eines Tasks ausgenutzt, um diese gezielt für Angriffe zu koordinieren [23]. Die Kommunikation zweier Tasks findet dabei über einen sog. verdeckten Kommunikationskanal (engl. covert channel) statt (siehe Abbildung 11). Mithilfe dieser Methode ist es für den Angreifer möglich, Nachrichten bzw. Daten im System unbemerkt zwischen verschiedenen Tasks zu senden, die sich Ressourcen wie z. B. den Cache-Speicher teilen. Diese Art der Kommunikation wurde erstmals 1973 von Lampson beschrieben [24].

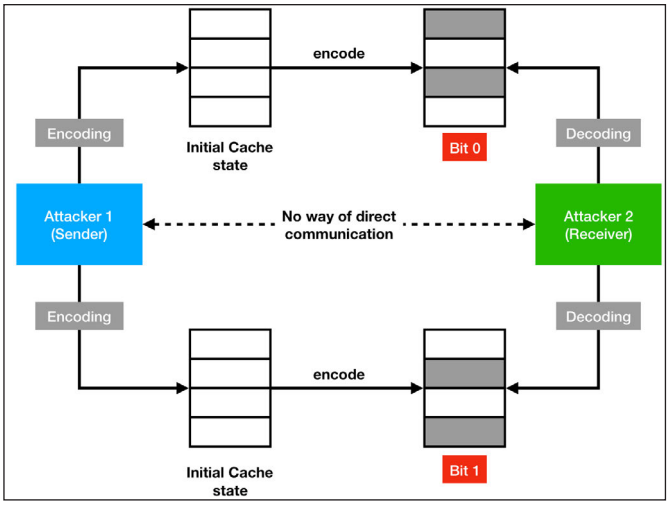

Abbildung 11: Kommunikation zweier Tasks über einen „covert channel“

Bei Covert-Channel-Angriffen gilt es zwischen Ressourcenkanälen (engl. ressource channels) und Zeitkanälen (engl. timing channels) zu unterscheiden. Bei Ressourcenkanälen können Tasks, die sich gemeinsame Ressourcen wie z. B. Festplatten, Hauptspeicher etc. teilen, über den Zugriff auf die Ressourcen miteinander kommunizieren [25]. Beispielsweise kann eine Kommunikation stattfinden, indem ein Task eine Ressource exklusiv beansprucht. Ein zweiter Task kann somit feststellen, ob dieselbe Ressource zur Verfügung steht oder blockiert wird. Diese beiden Zustände können als die Bit-Zustände 0 oder 1 interpretiert werden. Wird diese Methode in zeitlichen Abständen wiederholt, können die einzelnen Bit-Zustände zusammengesetzt Nachrichten bilden (Abbildung 12).

Zeitpunkt	Ressource vorhanden?	Empfangenes Bit
1	Ja	1
2	Ja	1
3	Nein	0
4	Ja	1
5	Nein	0
...	...	...

Abbildung 12: Ressource Channel

Bei Zeitkanälen müssen die Tasks hingegen in der Lage sein, die zeitlichen Abstände zwischen manipulierten Ereignissen zu messen. Beide Tasks müssen dabei Zugriff auf einen Zeitgeber, z. B. die Uhr des Systems, besitzen [25]. Die zeitlichen Abstände werden vom Empfänger wieder als Bit-Zustände interpretiert, die wiederum die Nachricht bilden. Beispielsweise können so Übertragungsprotokolle ausgenutzt werden, um die Reihenfolge von Netzwerkpaketen bzw. die Zeitdeltas zwischen Paketsendungen zu verändern (Abbildung 13) [26].

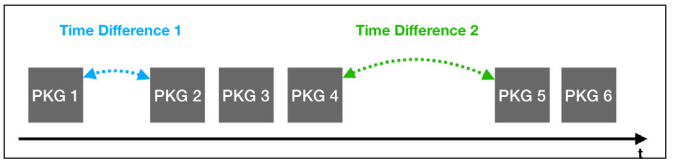

Abbildung 13: Timing Channel

Mithilfe von Zeitkanälen ist der Angreifer nicht mehr an Tasks gebunden, die sich eine ECU bzw. deren Ressourcen teilen. Durch einen gemeinsamen Übertragungskanal wie z. B. den CAN-Bus oder FlexRay wäre es möglich, zwischen Tasks im ganzen System verdeckt zu kommunizieren.

4 Zusammenfassung

Während im letzten Forschungsbericht Angriffe auf klassische fahrzeuginterne Kommunikationsnetzwerke und Bussysteme dargestellt wurden [27], sind in dieser Arbeit neue Angriffsmöglichkeiten basierend auf dem Scheduling-Verfahren aufgezeigt worden. Der Einzug von Multi- bzw. Many-Core-Systeme in den Automotive-Bereich birgt neue Gefahren, bedingt durch das Scheduling der Tasks sowie die Nutzung gemeinsamer Ressourcen. Dabei kann

sowohl die Security als auch die Safety eines Fahrzeugs beeinträchtigt werden. Durch das in Unterabschnitt 2.5 vorgestellte Modell könnte ein Hersteller versuchen, das System mit böswilligen Tasks zu infiltrieren, um an Informationen anderer Hersteller zu gelangen. Die Motive eines solchen Angriffs reichen vom Verkauf vertraulicher, durch Spionage gewonnener Daten bis hin zur Zerstörung von Systemen durch Sabotage, um gegenüber anderen Herstellern Vorteile auf dem Markt zu gewinnen. Denkbar wären auch Angriffe auf das Scheduling von außen durch veraltete Software oder ungepatchte Systeme.

Wie das Bundeslagebild 2016 zeigt, nahm die Anzahl an Cybercrime-Straftaten in Deutschland im Vergleich zum Vorjahr um knapp 81 % zu [28]. Angriffe auf das Automobil sind dabei längst keine Seltenheit mehr, da moderne Fahrzeuge ständig mit dem Internet verbunden sind und immer mehr zu fahrenden Computern werden. Deshalb müssen in Zukunft auch Security-Aspekte im Automotive-Bereich berücksichtigt werden. Ziel der weiteren Arbeit von Teilprojekt 2 im Forschungsprojekt FORMUS^{3}IC ist es, die vorgestellten Angriffe im Automotive-Umfeld weiter zu analysieren, um z. B. Präventionsmaßnahmen in der Softwarearchitektur oder ein Intrusion Detection System für Echtzeitsysteme zu entwickeln.

Referenzen:

[1] Hitachi Data Systems, „The Internet on Wheels and Hitachi, Ltd.“ 2015. [Online] Available: https://www.hds.com/en-us/pdf /white-paper/hitachi-white-paper-internet-on-wheels.pdf (Accessed 2017-08-29).

[2] J. Mottok und D. Fey, „Gesamtvorhabensbeschreibung FORMUS^{3}IC“, 2016.

[3] C. Öxleand M. Kuhlmey, Praxishandbuch Security, Richard Boorberg Verlag, 2015.

[4] International Organization for Standardization, „ISO 26262-1-Road vehicles : Functional Safety“ 2011. [Online] Available: https://www.iso.org/standard/43464.html (Accessed 2017-08-10).

[5] International Electrotechnical Commission, „IEC Functional Safety and IEC 61508“ 1998, [Online] Available: http://www.iec.ch/functionalsafety/ (Accessed 2017-08-13).

[6] T. M. Chen und S. Abu-Nimeh, „Lessons from Stuxnet“, 2011.

[7] WikiLeaks, „Brutal Kangaroo User Guide“ 2017, [Online] Available: https://wikileaks.org/vault7/document/BrutalKangaroo-DriftingDeadline-V2-UserGuide/ (Accessed 2017-09-04).

[8] E. A. Lee und S. A. Seshia, Introduction to Embedded Systems: A Cyber-Physical Systems Approach, MIT Press, 2016.

[9] S. B. Chien, Andrew A., „The Future of Microprocessors“ 2011. [Online] Available: https://cacm.acm.org/magazines /2011/5/107702-the-future-of-microprocessors/fulltext (Accessed 2017-08-11).

[10] H. Robert, Platzierung von Softwarekomponenten auf Mehrkernprozessoren, Springer-Verlag, 2015.

[11] F. Schäfer, Steuergeräte-Entwicklung mit AUTOSAR: Evaluierung der Entwicklungsumgebung Arctic Studio, 2014.

[12] J. Wietzke und M. T. Tran, Automotive Embedded Systeme, Springer-Verlag, 2006.

[13] W. Lange und M. Bogdan, Entwurf und Synthese von Eingebetteten Systemen: Ein Lehrbuch, Walter de Gruyter, 2013.

[14] R. Hilbrich, J. R. van Kampenhout und H.-J. Goltz, „Modellbasierte Generierung statischer Schedules für sicherheitskritische, eingebettete Systeme mit Multicore-Prozessoren und harten Echtzeitanforderungen“, Springer-Verlag, 2012.

[15] J. A. Stankovic und K. Ramamritham, „What is Predictability for Real-Time Systems?“, 1993.

[16] C. L. Liu und J. W. Layland, „Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment“, 1973.

[17] T. Rauber und G. Rünger, Parallele Programmierung, Springer-Verlag, 2012.

[18] C.-Y. Chen, R. B. Bobba und S. Mohan, „Schedule-Based Side Channel Attack in Fixed-Priority Real-time Systems“, 2015.

[19] Dan Boneh, Richard A. De Millo, und Richard J. Lipton, „On the Importance of Eliminating Errors in Cryptographic Computations“, 1997.

[20] D. Page, „Theoretical Use of Cache Memory as a Cryptanalytic Side-Channel“, 2002.

[21] Daniel J. Bernstein, „Cache-timing attacks on AES“, 2005.

[22] D. A. Osvik, A. Shamir und E. Tromer, „Cache Attacks and Countermeasures: The Case of AES“, 2006.
[23] J. Son und J. Alves-Foss, „Covert Timing Channel Analysis of Rate Monotonic Real-Time Scheduling Algorithm in MLS systems“, 2006.
[24] Butler W. Lampson und J. Alves-Foss, „A Note on the Confinement Problem“, 2006.
[25] Matt Bishop, Introduction to Computer Security. AddisonWesley, 2005.
[26] S. Wendzel, Tunnel und verdeckte Kanäle im Netz: Grundlagen, Protokolle, Sicherheit und Methoden, Springer-Verlag, 2012.
[27] B. Weigl und A. Aßmuth, „Bedrohungslage fahrzeuginterner Kommunikationsnetzwerke und Bus-Systeme“, 2017.
[28] Bundeskriminalamt, „Cybercrime Bundeslagebild 2016“, 2016.

Fördergeber:
Die Autoren danken der Bayerischen Forschungsstiftung (BFS) und dem Forschungsverbund FORMUS³IC „Multi-Core Safe and Software-intensive Systems Improvement Community“ für die finanzielle Unterstützung.

Bayerische Forschungsstiftung

Kontakt:

Tobias Nickl

Ostbayerische Technische
Hochschule (OTH) Amberg-Weiden
Fakultät Elektrotechnik, Medien
und Informatik
Laboratory for Safe and
Secure Systems (LaS³)
Kaiser-Wilhelm-Ring 23
92224 Amberg

t.nickl@oth-aw.de

Benjamin Weigl

Ostbayerische Technische
Hochschule (OTH) Amberg-Weiden
Fakultät Elektrotechnik, Medien
und Informatik
Laboratory for Safe and
Secure Systems (LaS³)
Kaiser-Wilhelm-Ring 23
92224 Amberg

b.weigl@oth-aw.de

Prof. Dr. Andreas Aßmuth

Ostbayerische Technische
Hochschule (OTH) Amberg-Weiden
Fakultät Elektrotechnik, Medien
und Informatik
Laboratory for Safe and
Secure Systems (LaS³)
Kaiser-Wilhelm-Ring 23
92224 Amberg

a.assmuth@oth-aw.de