© jay | adobe.stock

FUZZ-TESTING IN UDS ÜBER CAN

Verdeckte Schwachstellen erkennen und ausmerzen

Unified Diagnostic Services ist ein Standard, der zur Übertragung von Daten zu und von Steuergeräten zum Einsatz kommt. Der OBD-II-Port ist zu Diagnosezwecken mit einem oder mehreren Kommunikationsbussen verbunden. Der leichte Zugriff auf den OBD-Port ermöglicht es, böswillige UDS-Pakete zu versenden, um die Schwachstellen in Steuergeräten auszunutzen. Mit einem Fuzzing-Konzept sollen ungültige Daten oder verdeckte Schwachstellen erkannt werden.

Heutzutage integriert die Automobilindustrie zahlreiche neue Funktionen in vernetzte Fahrzeuge, und es gibt eine zunehmende Interaktion zwischen verbundenen ECUs über bordeigene Kommunikationsprotokolle. Dieser enorme Datenfluss muss im Sicherheitskontext validiert werden, um zu überprüfen, ob irgendwelche potenziellen Schwachstellen zu Beeinträchtigungen bei Kernfunktionen führen können. Um diese Echtzeit-Schwachstellen über das CAN-UDS-Protokoll zu finden, wird ein Black-Box-Fuzz-Testing-Ansatz genutzt. Die zentrale Idee beim Fuzzing ist die Entwicklung eines Fuzzers. Er besteht aus einer positiven Testsuite sowie veränderten Kriterien und Fuzzing-Kriterien. Zudem generiert und liefert er ungültige Eingaben an das zu testende System (SUT). Das Verhalten des SUT wird überwacht und Anomalien werden gemeldet. Mit Fuzzing lassen sich Schwachstellen an verschiedenen Schnittstellen und Steuergeräten (Electronic Control Units, ECUs) auffinden. Der Prozess umfasst das Abrufen der IDs und Sitzungen vom SUT, um das Fuzzing durchzuführen. Anschließend wird eine positive Testsuite mit gültigen CAN-Nachrichten erstellt. Die gültigen Eingabeaufforderungen werden weiter verändert, dem SUT werden veränderte Eingaben vorgelegt, und seine Reaktionen werden auf Abstürze oder Unterbrechungen

analysiert. Durch ungültige Eingaben an das System wird das Verhalten der ECU beobachtet. Auf diese Weise werden fehlerhafte Antworten der ECU gefunden und gemeldet. Diese Anfragen und Antworten werden in einer Datei gespeichert und analysiert.

UDS – ISO 14229

Der Standard Unified Diagnostic Services (UDS), auch als ISO 14229 [1] bekannt, ist eine in Straßenfahrzeugen verwendete Anwendungsprotokollschnittstelle für die Diagnose, Fehlerbehebung und Konfiguration von ECUs. UDS definiert, wie Nachrichten zu formatieren sind, nicht jedoch, wie sie zu implementieren sind – obwohl der Standard einige bewährte Verfahren vorschlägt. Daher handelt es sich um eine Schnittstelle. Bild 1 zeigt die Struktur und die Arten von Diagnosemeldungen. Der UDS-Client ist im Allgemeinen ein an einen Fahrzeugdiagnose-Port anzuschließendes Prüfgerät. Der UDS-Server ist im Allgemeinen ein Gerät oder eine elektronische „Name of ECU"-Steuereinheit (ECU) im Fahrzeug, die mit dem CAN-Bus verbunden ist – zugänglich über den Diagnose-Port.

Jede Funktion des Standards ist innerhalb des Konzepts eines Dienstes gruppiert. Ein Dienst ist eine Anfrageart, die einige Parameter aufweist. Hier einige Beispiele, wie Funktionen in Service-Kennungen gruppiert sind:

- ECU Reset – 0x11
- Diagnostic Session Control – 0x10
- Tester Present – 0x3E
- Read Data by identifier – 0x22

Diese Services können spezifische Unterfunktionen aufweisen. So hat zum Beispiel der Dienst „ECU Reset" eine Unterfunktion, die die Art der Rücksetzung beschreibt, die der Client durchführen möchte. Das kann ein harter Reset (Einschaltzyklus) oder ein softer Reset (Neustart der Firmware) sein. Die Unterfunktion ist das erste Byte der Dienstnutzlast, und ihr Wert wird durch UDS definiert. Wie der Server eine Rücksetzung auslöst, wird nicht von UDS definiert und bleibt dem Server-Programmierer überlassen.

Beim Herstellen einer Verbindung mit einem Server verfügt der Client über eine Sitzung und eine Sicherheitsstufe. In der Voreinstellung weist der Server den Client der „Standardsitzung" zu, in der nur einige wenige spezifische Dienste zugänglich sind, wie etwa das Auslesen der Diagnosefehler-Codes (DTCs). ISO 14229 definiert vier Sitzungstypen und definiert die Liste der verfügbaren Dienste – aber nur für die Standardsitzung. Mit anderen Worten, der ECU-Hersteller entscheidet, welche Dienste für jede Sitzung verfügbar sind, mit Ausnahme der Standardsitzung. Ein Client kann mit dem Dienst DiagnosticSessionControl ohne Einschränkungen auf jede Sitzung wechseln. Der ECU-Hersteller kann 32 zusätzliche Sitzungen definieren.

Die Sicherheitsstufe ist ein Status, den der Client durch Freischalten von Funktionen im Server mittels Sicherheitsschlüssel erreicht. Die UDS-Auslegung erlaubt bis zu 64 Sicherheitsstufen, die letztendlich im Server gesetzte Boolesche Flags sind. Diese Sicherheitsstufen und was sie freischalten, wird nicht von UDS definiert, sondern vom Hersteller der ECU. Eine Sicherheitsstufe kann einen kompletten Dienst, eine Unterfunktion oder den Zugriff auf einen spezifischen Wert freischalten. So setzt zum Beispiel das Schreiben der Fahrgestellnummer (VIN) möglicherweise eine spezifische Sicherheitsstufe voraus, die sich von dem unterscheidet, was zum Schreiben der Höchstgeschwindigkeit oder zum Überschreiben der Fahrzeug-IOs erforderlich ist.

In der Standardsitzung ist das Freischalten von Sicherheitsstufen nicht zulässig. Um einige Privilegien zu gewinnen, muss der Client zunächst auf eine Nicht-Standardsitzung schalten, die den SecurityAccess-Dienst aktiviert. Nur dann kann der Client den Handshake ausführen, der die gewünschte Funktion

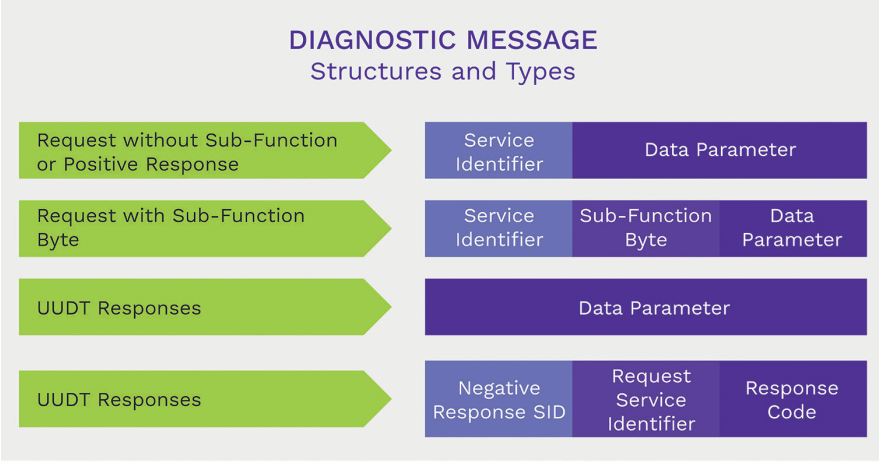

Bild 1: Diagnosemeldungen – Strukturen und Typen © KPIT Technologies

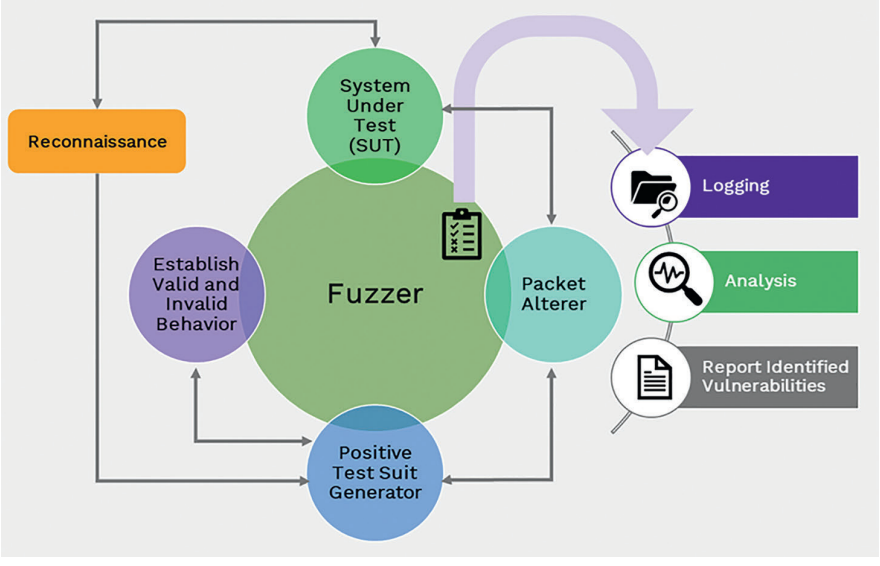

Bild 2: Fuzzing-Prozess © KPIT Technologies

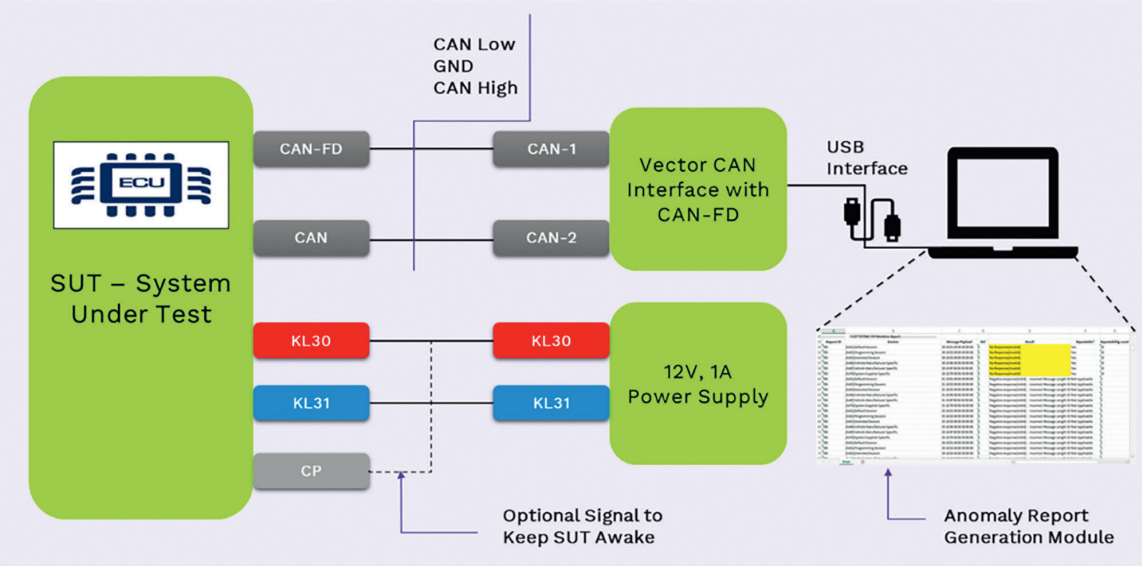

Bild 3: Versuchsaufbau © KPIT Technologies

	A	B	C	D	E	F	G
52	----------	*FUZZ TESTING PCI Mutation Report*----------					
53	*Request ID*	*Session*	*Message Payload*	*DLC*	*Result*	*Repeatable?*	*Repeatability count*
54	700	(0x01)Default Session	00 10 01 00 00 00 00 00	8	No Response(Invalid)	Yes	10
55	700	(0x02)Programming Session	00 10 02 00 00 00 00 00	8	No Response(Invalid)	Yes	10
56	700	(0x03)Extended Session	00 10 03 00 00 00 00 00	8	No Response(Invalid)	Yes	10
57	700	(0x40)Vehicle Manufacturer Specific	00 10 40 00 00 00 00 00	8	No Response(Invalid)	Yes	10
58	700	(0x4f)Vehicle Manufacturer Specific	00 10 4f 00 00 00 00 00	8	No Response(Invalid)	Yes	10
59	700	(0x70)System Supplier Specific	00 10 70 00 00 00 00 00	8	No Response(Invalid)	Yes	10
60	700	(0x01)Default Session	01 10 01 00 00 00 00 00	8	Negative response(Valid) - Incorrect Message Length Or Not Applicable		0
61	700	(0x02)Programming Session	01 10 02 00 00 00 00 00	8	Negative response(Valid) - Incorrect Message Length Or Not Applicable		0
62	700	(0x03)Extended Session	01 10 03 00 00 00 00 00	8	Negative response(Valid) - Incorrect Message Length Or Not Applicable		0
63	700	(0x40)Vehicle Manufacturer Specific	01 10 40 00 00 00 00 00	8	Negative response(Valid) - Incorrect Message Length Or Not Applicable		0
64	700	(0x4f)Vehicle Manufacturer Specific	01 10 4f 00 00 00 00 00	8	Negative response(Valid) - Incorrect Message Length Or Not Applicable		0
65	700	(0x70)System Supplier Specific	01 10 70 00 00 00 00 00	8	Negative response(Valid) - Incorrect Message Length Or Not Applicable		0
66	700	(0x01)Default Session	03 10 01 00 00 00 00 00	8	Negative response(Valid) - Incorrect Message Length Or Not Applicable		0
67	700	(0x02)Programming Session	03 10 02 00 00 00 00 00	8	Negative response(Valid) - Incorrect Message Length Or Not Applicable		0
68	700	(0x03)Extended Session	03 10 03 00 00 00 00 00	8	Negative response(Valid) - Incorrect Message Length Or Not Applicable		0
69	700	(0x40)Vehicle Manufacturer Specific	03 10 40 00 00 00 00 00	8	Negative response(Valid) - Incorrect Message Length Or Not Applicable		0
70	700	(0x4f)Vehicle Manufacturer Specific	03 10 4f 00 00 00 00 00	8	Negative response(Valid) - Incorrect Message Length Or Not Applicable		0
71	700	(0x70)System Supplier Specific	03 10 70 00 00 00 00 00	8	Negative response(Valid) - Incorrect Message Length Or Not Applicable		0
72	700	(0x01)Default Session	04 10 01 00 00 00 00 00	8	Negative response(Valid) - Incorrect Message Length Or Not Applicable		0
73	700	(0x02)Programming Session	04 10 02 00 00 00 00 00	8	Negative response(Valid) - Incorrect Message Length Or Not Applicable		0
74	700	(0x03)Extended Session	04 10 03 00 00 00 00 00	8	Negative response(Valid) - Incorrect Message Length Or Not Applicable		0

Sheet

Bild 4: Durch Fuzzing festgestellte Anomalien © KPIT Technologies

freischaltet. Im Allgemeinen laufen erlangte Privilegien nach einem kurzen vom ECU-Hersteller definierten Zeitraum ab. Eine Keep-Alive-Nachricht hält die Sicherheitsstufe freigeschaltet und die Sitzung aktiv, wobei diese Keep-Alive-Nachrichten mit dem Dienst TesterPresent versendet werden.

Die zum Freischalten einer Sicherheitsstufe zu sendende Nutzlast ist nicht von UDS definiert: UDS definiert die Vorgehensweise beim Schlüsselaustausch. Dieser Prozess basiert auf zwei Anfrage/Antwort-Austauschvorgängen zwischen Client und Server. Er geht wie folgt vonstatten:

- Zunächst fragt der Client ein Seed an, um eine spezifische Sicherheitsstufe, die durch eine Nummer gekennzeichnet ist, freizuschalten. Dieses Seed ist im Allgemeinen ein Zufallswert, den der Client verwenden muss, um den Schlüssel zu berechnen. Damit soll verhindert werden, dass jemand den Nachrichtenaustausch des CAN-Busses aufzeichnet und sich dann Privilegien durch unbesehenes Zusenden der Aufzeichnungen verschafft. Kryptografisch gesehen ist das Seed eine Nonce zur Vermeidung von Replay-Angriffen.
- Sobald der Client das Seed erhält, muss er einen Schlüssel berechnen, indem er einen Algorithmus verwendet, der vom ECU-Hersteller definiert wurde und dem Server bekannt ist.
- Der Client sendet den Schlüssel dann an den Server, der Server überprüft ihn und wenn er zum Server-Wert passt, wird die Sicherheitsstufe freigeschaltet, und der Client erhält eine positive Nachricht.

Der Sicherheitsalgorithmus kann ein beliebiger Algorithmus sein. Der Mangel an Algorithmusdefinition im UDS-Standard lässt einigen Raum für gute Sicherheitskonzepte, aber auch für schlechte – es hängt ganz vom Hersteller ab. In der Tat nutzen einige Hersteller das Konzept „Sicherheit durch Verschleierung", während sich andere für ein robusteres Pre-Shared-Key-Schema entscheiden.

Prozessablauf

Ziel dieses Experiments war die Verallgemeinerung des Fuzzer-Algorithmus, sodass er für alle ECUs zum Einsatz kommen kann, die UDS über CAN gemäß UDS-Protokoll nutzen [1]. Weil ein Black-Box-Fuzzing durchgeführt wird, bekommt man ein Grundverständnis des Systems. In der Erkundungsphase hat sich KPIT mit dem Systemverständnis beschäftigt. Darauf folgte die Erstellung einer positiven Testsuite auf der Grundlage der Ergebnisse der Erkundungsphase. Es wurde eine Änderung umgesetzt, und die Ausgaben wurden auf der Basis eines Satzes von Kriterien klassifiziert. Des Weiteren wird jede Phase des Fuzzings im Detail erläutert. Bild 2 stellt einen Fuzzing-Prozess dar.

Zu testendes System

Der Prozess begann mit der Identifizierung des UDS über CAN-Ports/OBD-II-Ports, gefolgt von einer Hardware-Konfiguration. Für diese wurde die korrekte Spannung bereitgestellt und das SUT unter Verwendung eines USB-2-CAN-Protokollkonverters mit dem Laptop verbunden. Um Kriterien, Überwachung, Veränderung und Analyse der Pakete festzulegen, wurden CAPL-Skripts entwickelt.

Erkundung

Für eine erfolgreiche Durchführung des Fuzzings (Bild 2) ist es sehr wichtig, die diagnostische Arbitrierungs-ID zu finden. Die Feststellung der Arbitrierungs-

ID ist ein wichtiger Schritt zur Kommunikation mit der ECU per Diagnose über CAN. Dazu wurde ein CAPL-Skript zur Identifizierung der Arbitrierungs-ID entwickelt. Das CANoe-CAPL-Skript spielt mit dem 8-Byte-Datenfeld des CAN-Frames. Das Arbitrierungs-ID-Feld der Reihe wird nach auf bis zu 11 Bit für den Standard-CAN-Frame geändert – dabei bleiben andere Datenfelder konstant. Auf ähnliche Weise werden die in den ECUs implementierten Dienste, die Dienst-IDs und die Unterfunktions-IDs festgestellt.

Positive Testfallgenerierung und Definition von Fuzzing-Kriterien

Zu diesem Schritt gehört die Erstellung gültiger UDS-Anfragen. Zu diesem Zweck wurde der UDS-Protokollstandard [1] und den UDS über CAN-Standard [2] untersucht. Damit werden Unified-Diagnostic-Services-Anforderungen für die Anwendungsschicht definiert. Diese ISO wurde untersucht und man hat eine positive Testsuite erstellt. Die positive Testsuite wird automatisch vom Fuzzer generiert. Der Algorithmus generiert automatisch eine positive Testsuite, mit der das Verhalten der ECU beobachtet und Fuzzing-Kriterien definiert werden.

Systematische Veränderung in CAN-Paketen

Mutationsbasiertes Fuzzing ist ein effektiver Ansatz zur Verbesserung der Sicherheit und Zuverlässigkeit von Protokollimplementierungen. Der Änderungsbasierte Fuzzer arbeitet mit gültigen Daten und wendet zufällige Veränderungen darauf an. Mit dieser Technik lässt sich tief in die Nachrichtenverarbeitung am Ziel eindringen. Die gültige Eingabeaufforderung wird verändert. Veränderte Eingaben werden dem SUT vorgelegt und die Reaktionen des SUT werden auf Abstürze oder Unterbrechungen analysiert. Durch ungültige Eingaben an das System wird das Verhalten der ECU beobachtet. Auf diese Weise werden fehlerhafte Antworten der ECU gefunden und gemeldet. Diese Anfragen und Antworten werden in einer Datei gespeichert und analysiert. Während der Analyse wird eine generierte Liste anfälliger Nachrichten hervorgehoben. Ein Nachrichtenlängen-Byte, ein Unterfunktions-Byte mit und ohne gesetztem SPR-Bit und ein Datenlängencode (DLC) wurden verändert. Ebenso wurden Änderungen an Diensten ohne Unterfunktion durchgeführt. Dieser Prozess wird auf Dienste sowohl in Standard- als auch in Nicht-Standardsitzungen angewendet. Der Grund dafür ist, dass einige Dienste in Standardsitzungen und einige in Nicht-Standardsitzungen implementiert sind.

Versuchsaufbau

Die Einrichtung besteht aus dem zu testenden System (SUT). Bild 3 zeigt einen Versuchsaufbau für das Fuzzing. Damit wurde ein Fuzzing an mehreren ECUs von Drittanbietern durchgeführt. Das SUT ist mit der Vector-CAN-Schnittstelle verbunden. Diese ist an den PC angeschlossen, der als Prüfgerät fungiert. Die Fuzzer-Skripts werden mit CANoe-CAPL auf dem PC ausgeführt.

Datenanalyse-basierte Algorithmen

Am Ende des Fuzzing-Prozesses wird eine Liste von Sicherheitslücken erstellt, bei denen es sich um Antworten handelt, die weder negativ noch positiv gemäß ISO 14229–1 sind. Das wird durch Entwicklung Datenanalyse-basierter Algorithmen für das Berichtsmodul erreicht. Der Algorithmus wendet Datenanalysetechniken auf die positive Testsuite und ECU-Antwort an, um Anomalien festzustellen, die dann in einer Excel-Tabelle berichtet werden. Alle gültigen Antworten werden ebenfalls protokolliert, und die Anomalien werden hervorgehoben. Bild 4 zeigt die gelb hervorgehobenen Anomalien. Der Bericht besteht aus Arbitrierungs-ID, Sitzungen, der mutierten Nutzlast und dem DLC sowie dem Ergebnis. ■ (eck)

www.kpit.com

Quellenverzeichnis

[1] Straßenfahrzeug – Einheitliche Diagnosedienste (UDS) ISO 14229–1, ISO 14229–1, 2013.

[2] Straßenfahrzeuge – Diagnosekommunikation über Controller Area Network (DoCAN) – ISO 15765 Teil 2, 2004.

Ajey Gotkhindikar arbeitet als Subject Matter Expert für Automotive Cybersecurity bei KPIT Technologies.

Diagnosetester mit Remote-Funktion

Vector Informatik bietet mit der Version 8 des Fahrzeugdiagnosetesters Indigo neue Möglichkeiten zur Individualisierung und eine optimierte Remote-Diagnose. Mit dem neuen Plugin-Installer-Konzept und dem Software Development Kit (SDK) passt der Anwender die Funktionalität individuell an. Zudem bietet die Remote-Diagnose mit Indigo 8 ein umfassendes Sicherheitskonzept ohne Einschränkungen am Bedienkomfort. Der intuitiv zu bedienende Fahrzeugdiagnosetester, lässt sich ohne Diagnoseprotokoll-Kenntnisse nach Bedarf nutzerspezifisch anpassen und erweitern. Die Anwender profitieren bei der Version 8 von einem Plugin-Installer-Konzept, das die Installation von individuellen Erweiterungen erleichtert. Dabei wird zusätzlich zur Indigo Installation ein Anwender-Plugin ausgeliefert, das ohne Administratorenrechte installiert werden kann und das Tool funktional erweitert. Diese Anpassungen an Anwenderbedürfnisse werden von Vector auf Wunsch umgesetzt. Erweiterungen der Funktionalität im kleineren Umfang kann der Nutzer selbst mit dem Indigo SDK vornehmen. Die Implementierung eigener Use-Case-Diagnosefenster sowie der Zugriff auf den vollen Funktionsumfang der Vector Diagnostic Scripting Bibliotheken ist dabei einfach möglich.

www.vector.com

Vector Informatik stellt die Version 8 des Fahrzeugdiagnosetesters Indigo zur Verfügung © Vector Informatik