Security Testing & Analysis

Overview

- Quality vs. Security
- Overview of a common security problem - buffer overflow
- Introduction to testing for security
  - Penetration testing, fuzzing
- Introduction to static analysis for security
- Case study: Cybersecurity in Connected Autonomous Vehicles (CAVs)

Quality vs. Security

Does high quality software = secure software?

Buffer Overflow

Definition

- Cause: writing data to a fixed sized buffer (e.g., array, stack) that is longer than the fixed length
- Where can this occur? In any *unsafe* programming language (i.e., not memory or type safe) when the programmer does not *explicitly* protect against it
  - An example of an unsafe programming language is C++
- Consequences: unintended behaviour, program termination,
- an exception.

Testing for Security

Traditional Testing Techniques

- We have already learned about many kinds of software testing – both black box and white box
- Unfortunately, these traditional testing techniques do not always ensure that a software is secure
- **Why?** Let's consider black box testing…
  - Black box testing is based on software requirements
  - Security errors are often caused by *"unintended"* program behaviour that does not violate these requirements
  - This means that black box tests can not usually find these security errors unless there are explicit security requirements – not often the case!

Testing for Security

Penetration Testing (or Pen Testing)

- An assessment of a system’s security using simulated or mock attacks
- It can be used to test the ease to which a system can be accessed by first discovering and then exploiting vulnerabilities
- Pen tests can be black box or white box depending on if the simulated attack is created with internal knowledge of the system under test
- Common black box pen tests can start with social engineering or phishing

Testing for Security

Fuzzing (or Fuzz Testing)

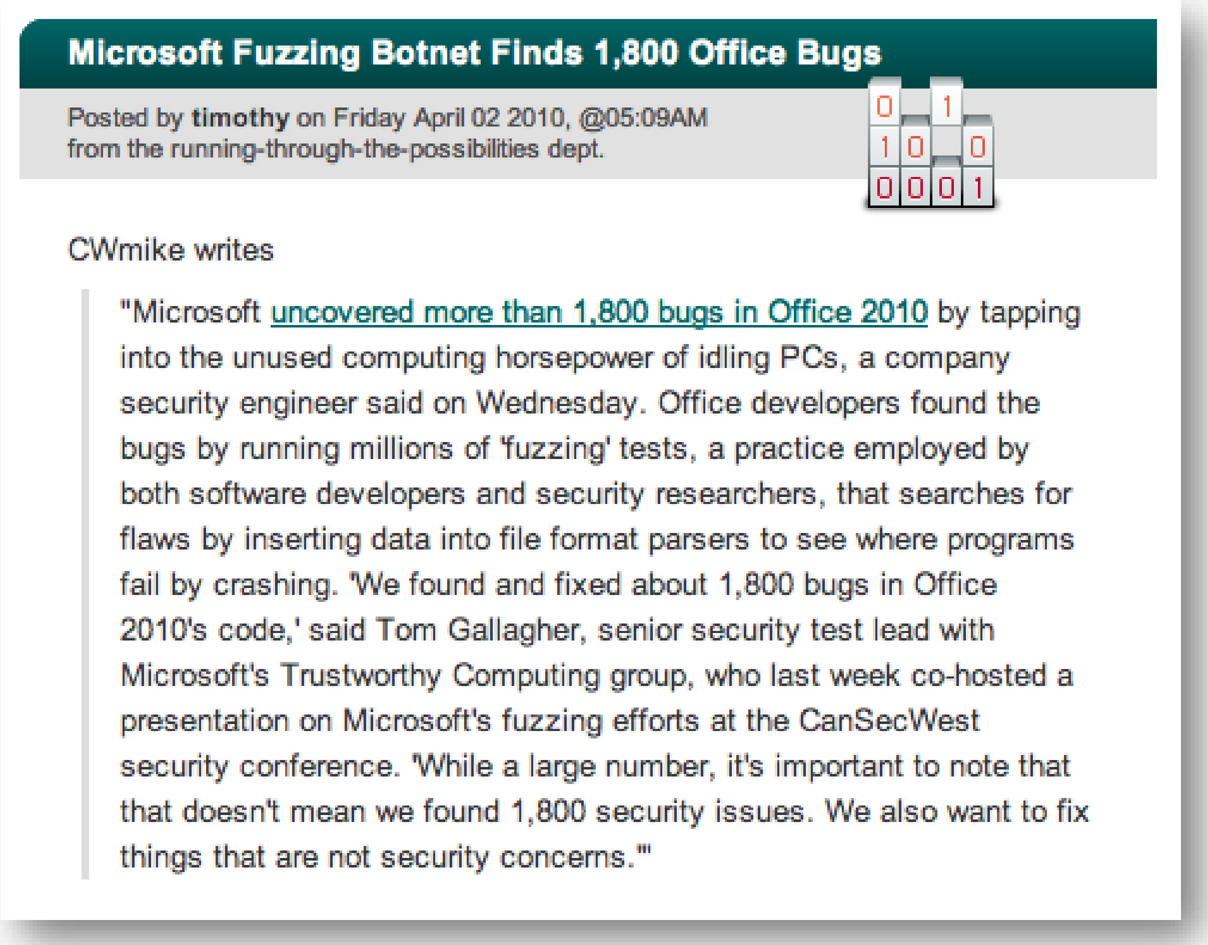
- Based on the observation that unintended or unexpected behaviour can lead to security problems
- Fuzzing attempts to cause unintended/unexpected behavior by testing the program with invalid or semi-valid input data

Testing for Security

Fuzzing (or Fuzz Testing)

- Three kinds of fuzzing:
  1. **Blackbox random fuzzing:** inputs are generated randomly (similar to shotgun testing)
  2. **Grammar-based fuzzing**: inputs are created using mutation and are based on knowledge of valid data format (i.e. input grammar)
  3. **Whitebox fuzzing:** input constraints are identified using symbolic execution and then a constraint solver is used to systematically explore the program execution paths.

Fuzzing at Microsoft (2010)



Microsoft Fuzzing Botnet Finds 1,800 Office Bugs

Posted by **timothy** on Friday April 02 2010, @05:09AM
from the running-through-the-possibilities dept.

CWmike writes

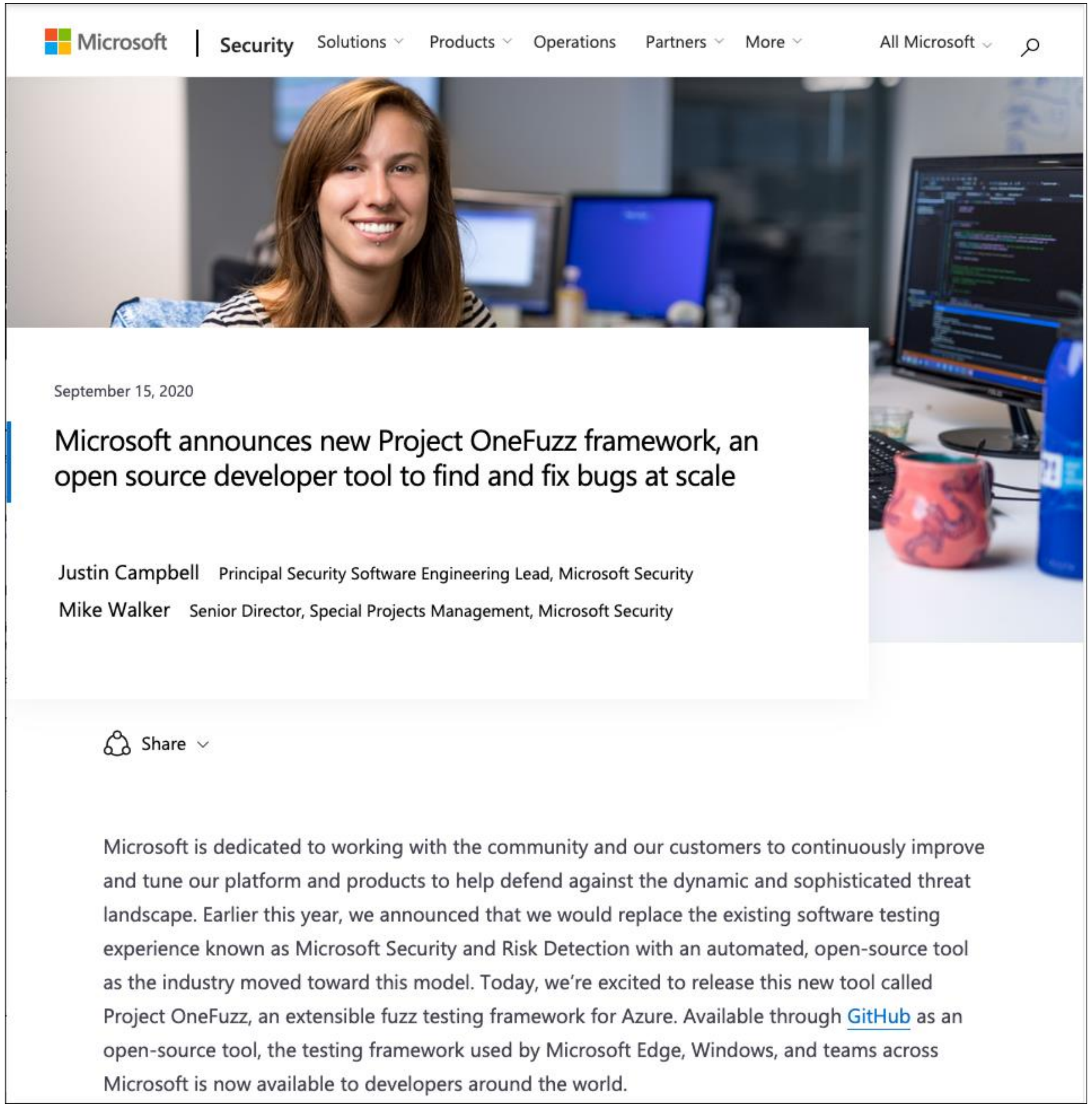
> "Microsoft uncovered more than 1,800 bugs in Office 2010 by tapping into the unused computing horsepower of idling PCs, a company security engineer said on Wednesday. Office developers found the bugs by running millions of 'fuzzing' tests, a practice employed by both software developers and security researchers, that searches for flaws by inserting data into file format parsers to see where programs fail by crashing. 'We found and fixed about 1,800 bugs in Office 2010's code,' said Tom Gallagher, senior security test lead with Microsoft's Trustworthy Computing group, who last week co-hosted a presentation on Microsoft's fuzzing efforts at the CanSecWest security conference. 'While a large number, it's important to note that that doesn't mean we found 1,800 security issues. We also want to fix things that are not security concerns.'"

Source: http://developers.slashdot.org/story/10/04/02/0733221/Microsoft-Fuzzing-Botnet-Finds-1800-Office-Bugs**?**

© J.S. Bradbury

Fuzzing Tools at Microsoft

- Google has open sourced **OneFuzz**
  - https://github.com/microsoft/onefuzz
- OneFuzz is an Azure testing framework



September 15, 2020

Microsoft announces new Project OneFuzz framework, an open source developer tool to find and fix bugs at scale

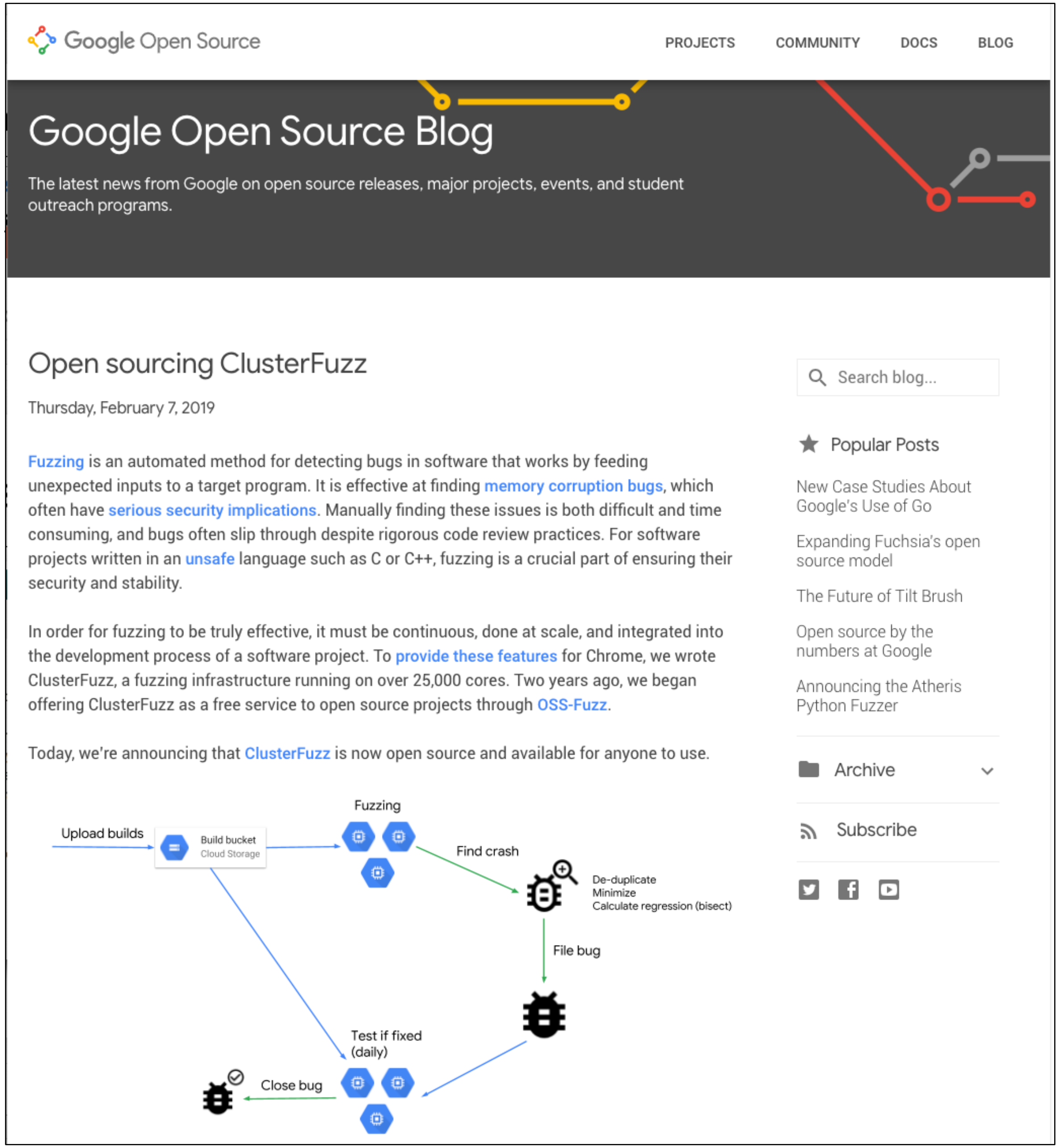
Justin Campbell Principal Security Software Engineering Lead, Microsoft Security

Mike Walker Senior Director, Special Projects Management, Microsoft Security

Microsoft is dedicated to working with the community and our customers to continuously improve and tune our platform and products to help defend against the dynamic and sophisticated threat landscape. Earlier this year, we announced that we would replace the existing software testing experience known as Microsoft Security and Risk Detection with an automated, open-source tool as the industry moved toward this model. Today, we're excited to release this new tool called Project OneFuzz, an extensible fuzz testing framework for Azure. Available through GitHub as an open-source tool, the testing framework used by Microsoft Edge, Windows, and teams across Microsoft is now available to developers around the world.

Fuzzing Tools at Google

- Google has open sourced **ClusterFuzz**
  - https://github.com/google/clusterfuzz
- It has been used to find:
  - 16,000 chrome bugs
  - 11,000+ open source project bugs



Google Open Source

PROJECTS COMMUNITY DOCS BLOG

Google Open Source Blog

The latest news from Google on open source releases, major projects, events, and student outreach programs.

Open sourcing ClusterFuzz

Thursday, February 7, 2019

Fuzzing is an automated method for detecting bugs in software that works by feeding unexpected inputs to a target program. It is effective at finding memory corruption bugs, which often have serious security implications. Manually finding these issues is both difficult and time consuming, and bugs often slip through despite rigorous code review practices. For software projects written in an unsafe language such as C or C++, fuzzing is a crucial part of ensuring their security and stability.

In order for fuzzing to be truly effective, it must be continuous, done at scale, and integrated into the development process of a software project. To provide these features for Chrome, we wrote ClusterFuzz, a fuzzing infrastructure running on over 25,000 cores. Two years ago, we began offering ClusterFuzz as a free service to open source projects through OSS-Fuzz.

Today, we're announcing that ClusterFuzz is now open source and available for anyone to use.

Search blog...

Popular Posts

New Case Studies About Google's Use of Go

Expanding Fuchsia's open source model

The Future of Tilt Brush

Open source by the numbers at Google

Announcing the Atheris Python Fuzzer

Archive

Subscribe

Static Analysis for Security

SpotBugs (aka FindBugs)

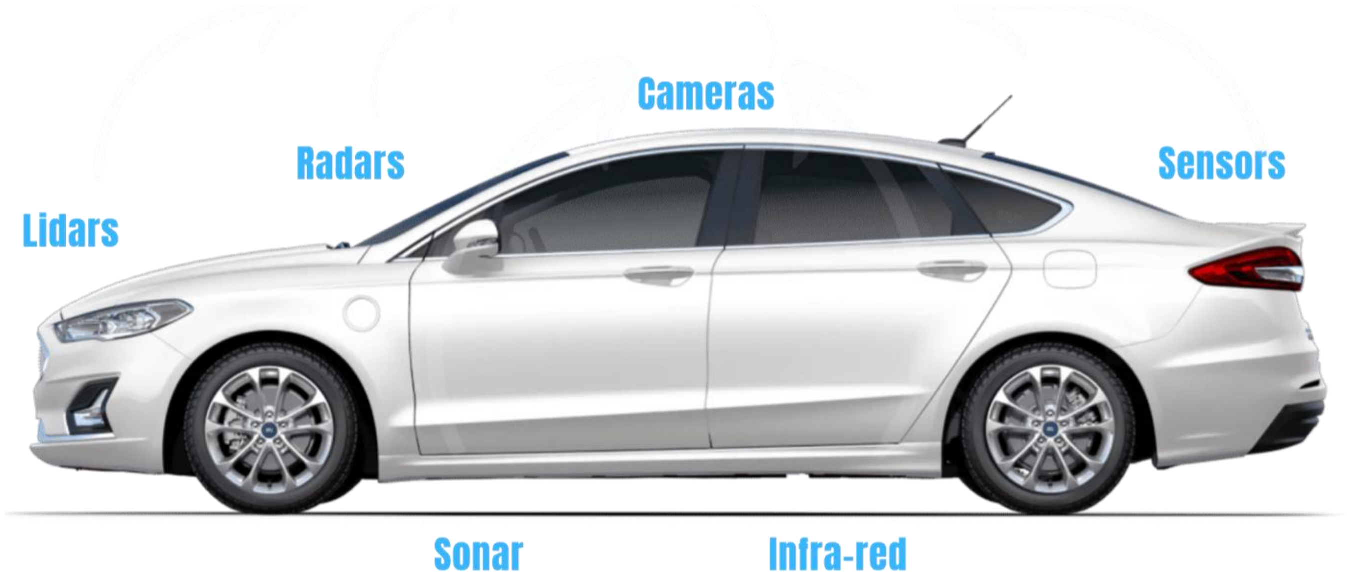
- https://spotbugs.github.io/
- We have already seen that some general bug detection tools include bug patterns related to security

Coverity

- https://www.synopsys.com/software-integrity/security-testing/static-analysis-sast.html
- Assesses both security and quality

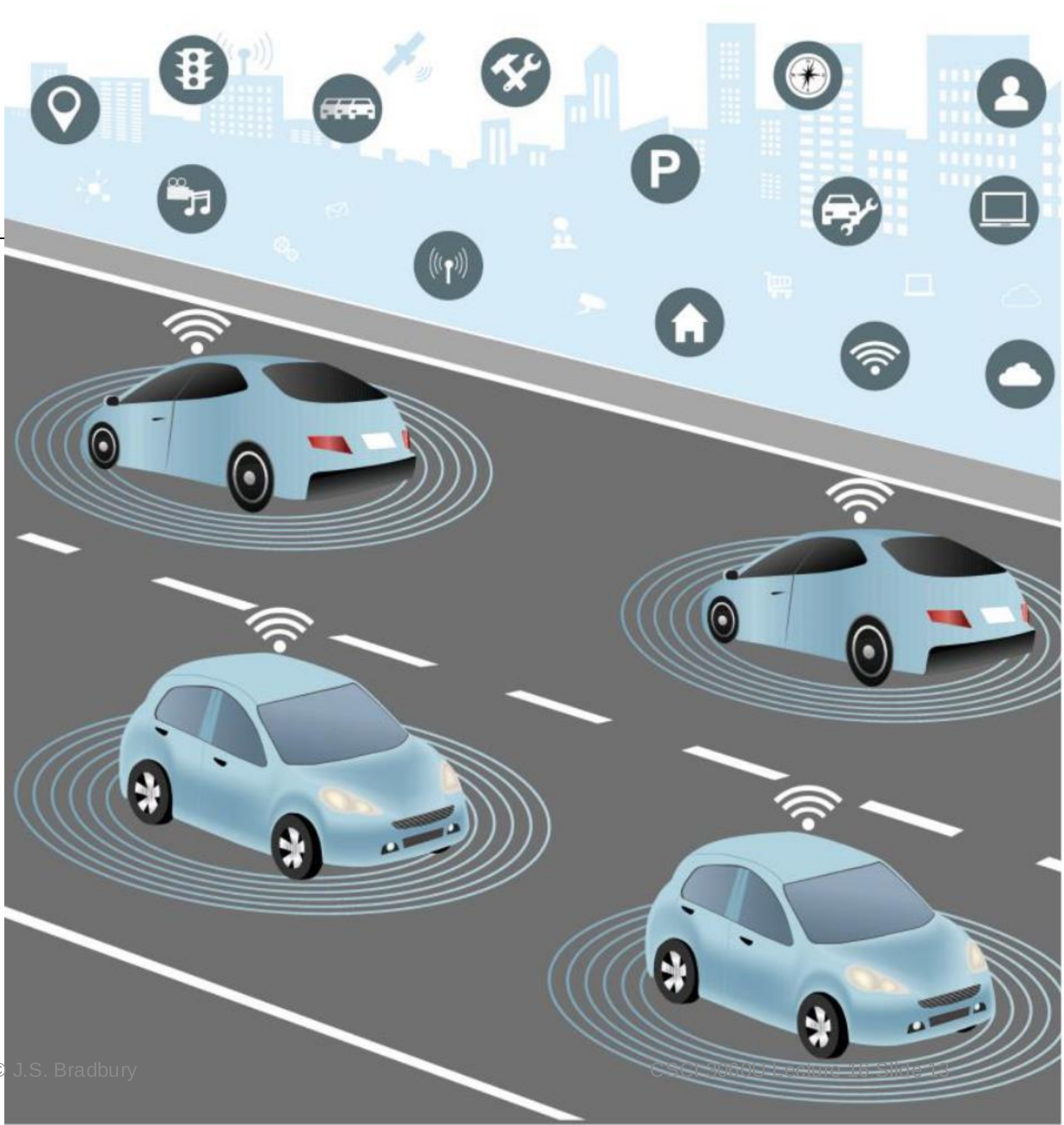
Case Study

Cybersecurity in Connected Autonomous Vehicles (CAVs)



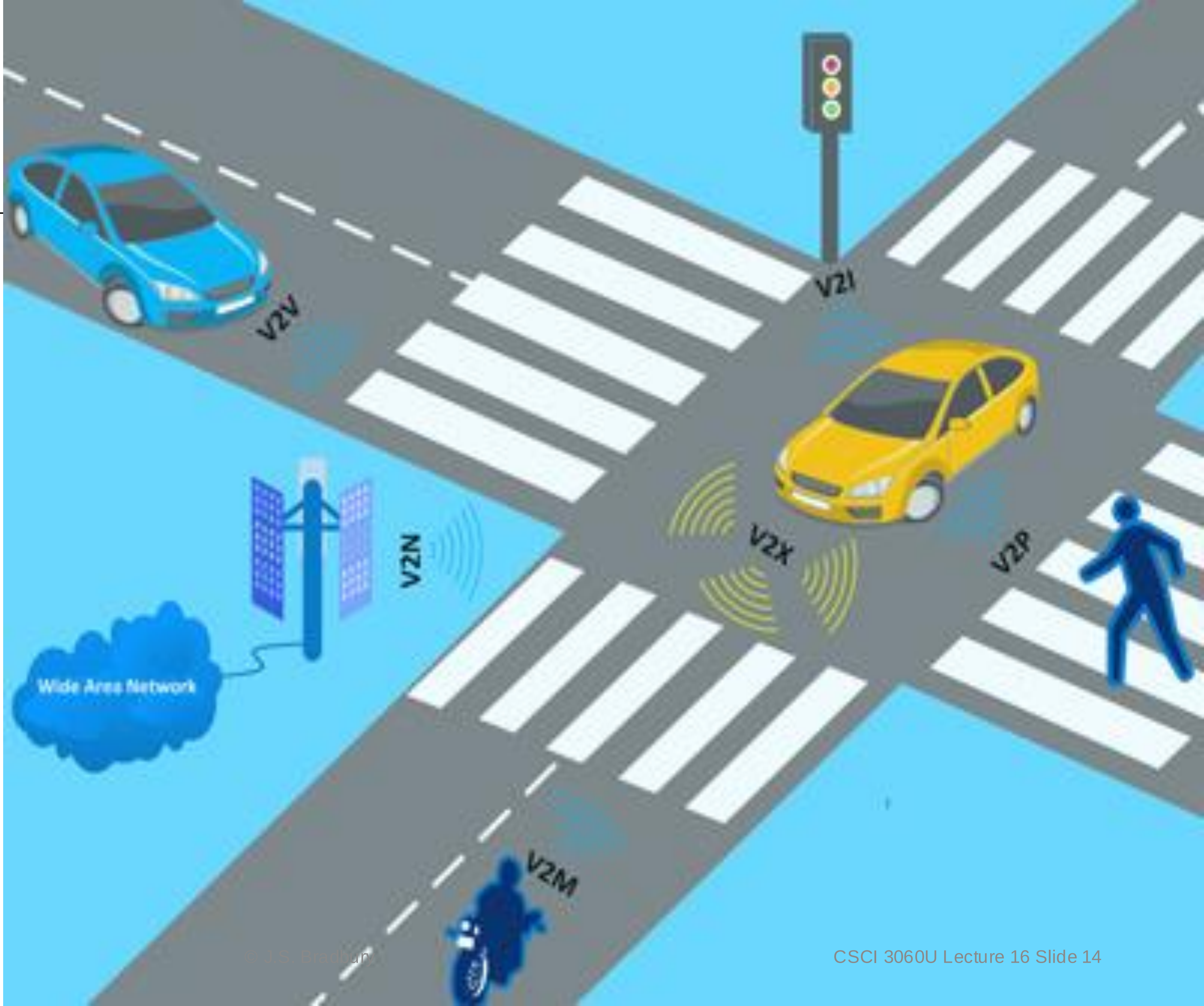
CAVs

- Use sensors to detect the environment
- Operate without human input



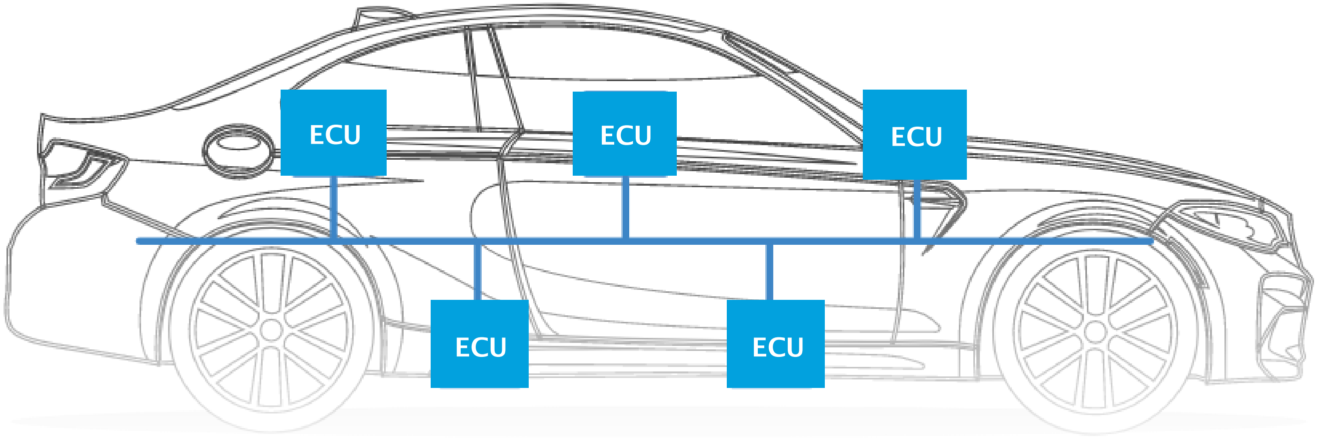
CAVs

- Vehicle to…
  - Infrastructure (V2I)
  - Vehicle (V2V)
  - Devices (V2D)
  - Network (V2N)
  - Pedestrian (V2P)
  - Any internet-enabled device (V2X)



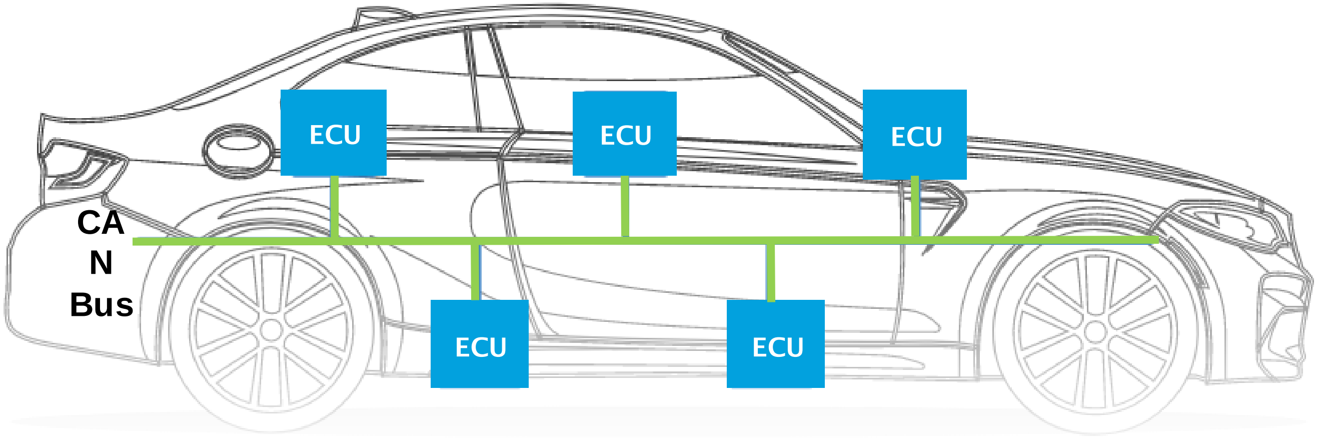
CAVs – CAN Bus

- The central nervous system of the car



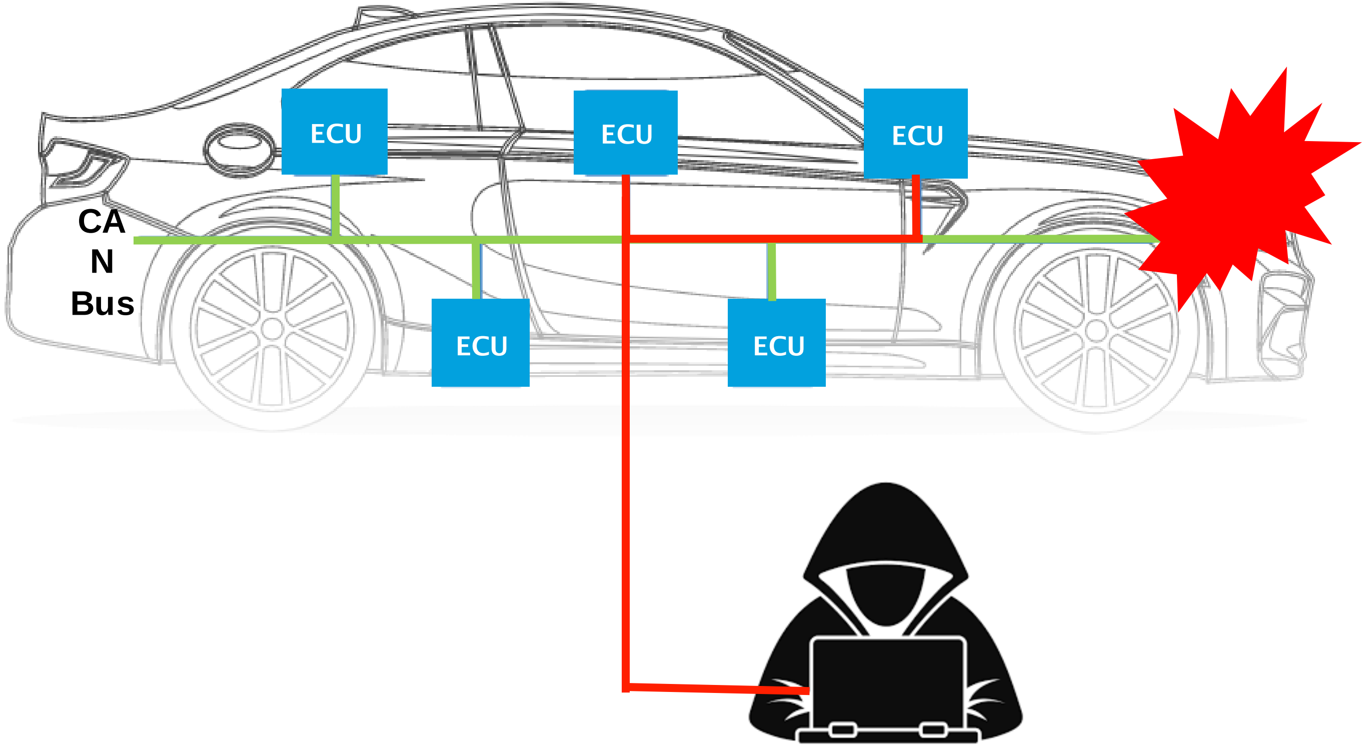
CAVs – CAN Bus

- The central nervous system of the car
- CAN Bus allows ECUs to communicate with one another

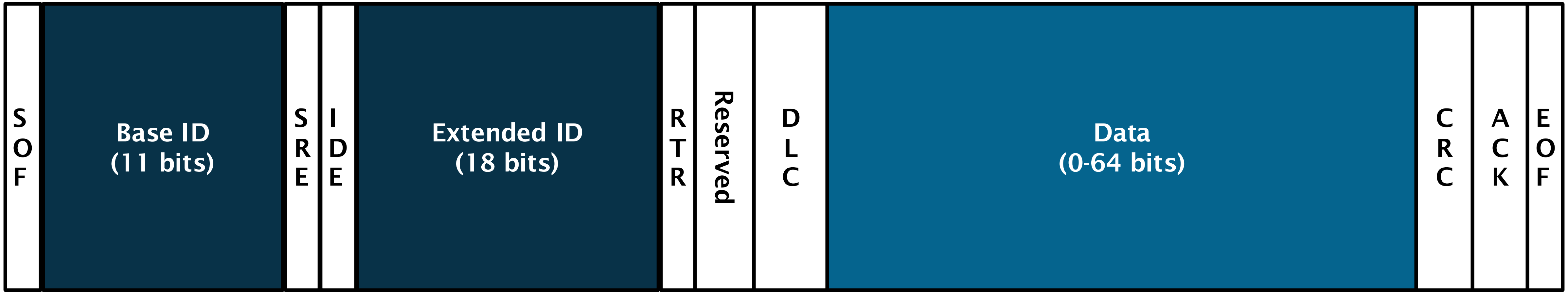


CAVs – CAN Bus

- The central nervous system of the car
- CAN Bus allows ECUs to communicate with one another
- Car systems are more vulnerable than ever



CAVs – CAN Message Frame



SOF

Base ID
(11 bits)

SRE

IDE

Extended ID
(18 bits)

RTR

Reserved

DLC

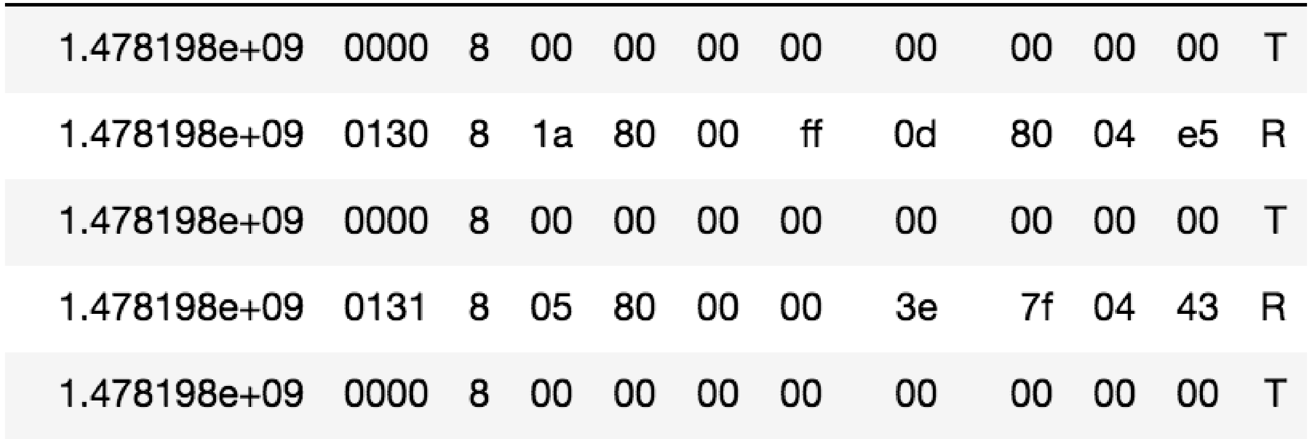
Data
(0-64 bits)

CRC

ACK

EOF

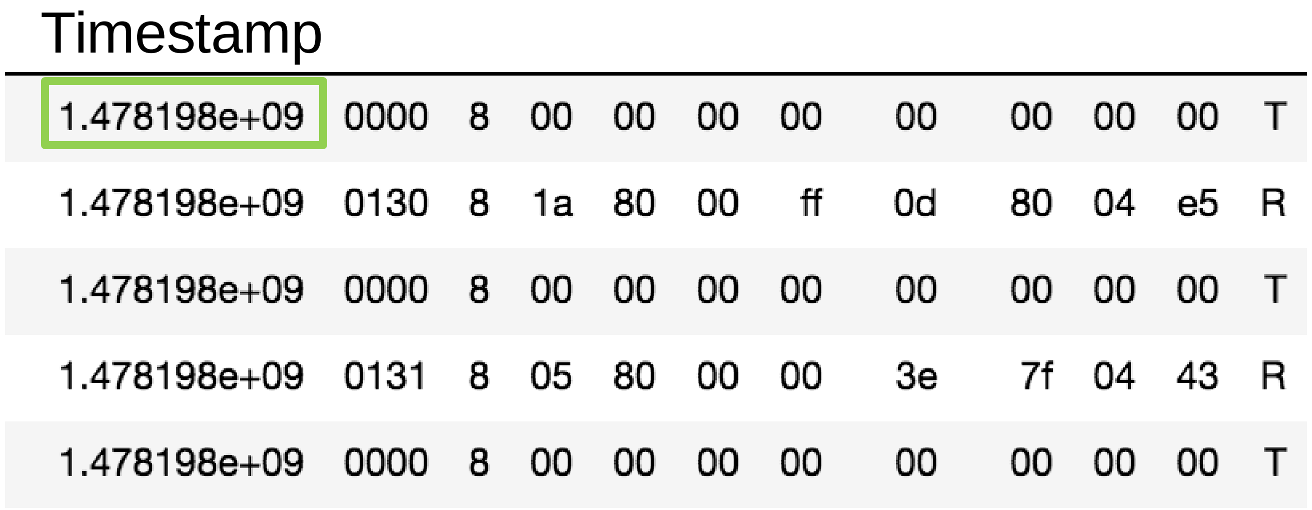
Intrusion Detection - Dataset



1.478198e+09	0000	8	00	00	00	00	00	00	00	00	T
1.478198e+09	0130	8	1a	80	00	ff	0d	80	04	e5	R
1.478198e+09	0000	8	00	00	00	00	00	00	00	00	T
1.478198e+09	0131	8	05	80	00	00	3e	7f	04	43	R
1.478198e+09	0000	8	00	00	00	00	00	00	00	00	T

- *Source:*
  - **Car-Hacking dataset** from Hacking and Countermeasure Research Lab
  - http://ocslab.hksecurity.net/Datasets/CAN-intrusion-dataset

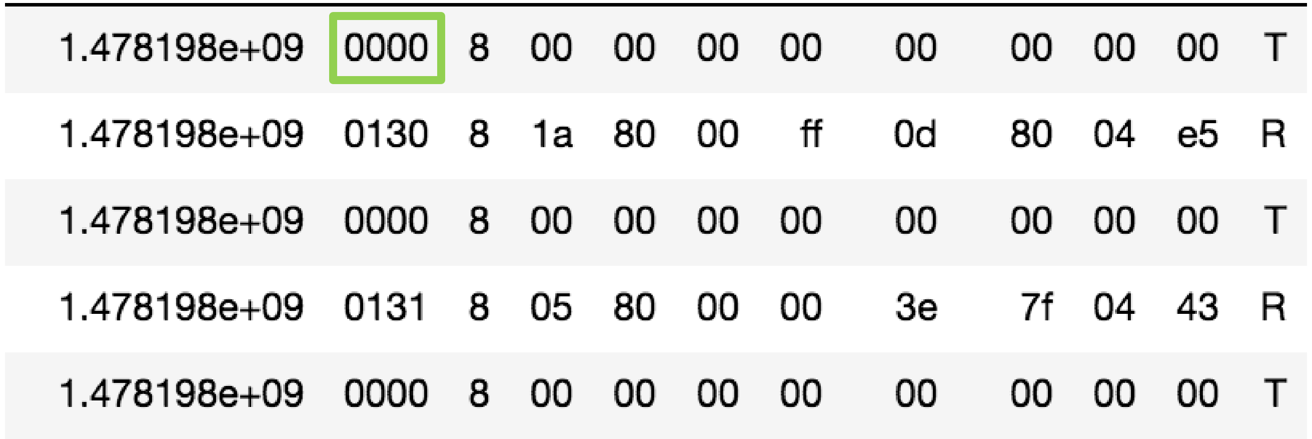
Intrusion Detection - Dataset



Timestamp												
1.478198e+09	0000	8	00	00	00	00	00	00	00	00	T
1.478198e+09	0130	8	1a	80	00	ff	0d	80	04	e5	R
1.478198e+09	0000	8	00	00	00	00	00	00	00	00	T
1.478198e+09	0131	8	05	80	00	00	3e	7f	04	43	R
1.478198e+09	0000	8	00	00	00	00	00	00	00	00	T

- *Source:*
  - **Car-Hacking dataset** from Hacking and Countermeasure Research Lab
  - http://ocslab.hksecurity.net/Datasets/CAN-intrusion-dataset

Intrusion Detection - Dataset

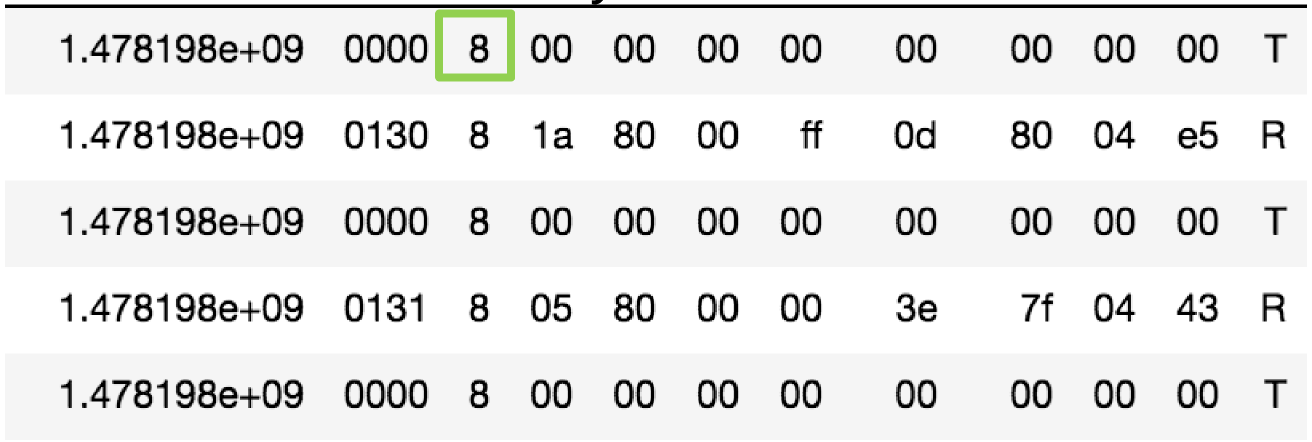


	CAN ID										
1.478198e+09	0000	8	00	00	00	00	00	00	00	00	T
1.478198e+09	0130	8	1a	80	00	ff	0d	80	04	e5	R
1.478198e+09	0000	8	00	00	00	00	00	00	00	00	T
1.478198e+09	0131	8	05	80	00	00	3e	7f	04	43	R
1.478198e+09	0000	8	00	00	00	00	00	00	00	00	T

- *Source:*
  - **Car-Hacking dataset** from Hacking and Countermeasure Research Lab
  - http://ocslab.hksecurity.net/Datasets/CAN-intrusion-dataset

Intrusion Detection - Dataset

Size of Payload

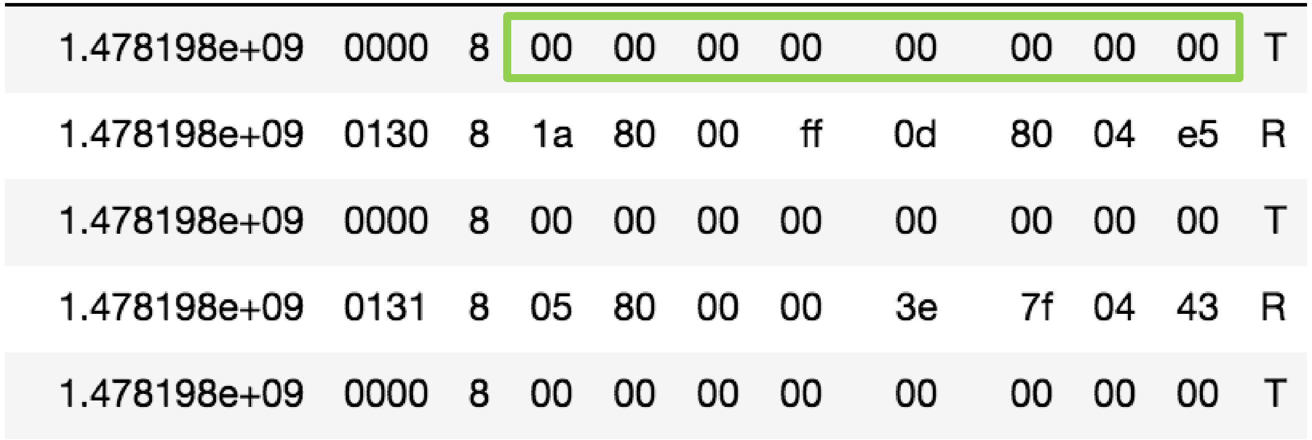


1.478198e+09	0000	8	00	00	00	00	00	00	00	00	T
1.478198e+09	0130	8	1a	80	00	ff	0d	80	04	e5	R
1.478198e+09	0000	8	00	00	00	00	00	00	00	00	T
1.478198e+09	0131	8	05	80	00	00	3e	7f	04	43	R
1.478198e+09	0000	8	00	00	00	00	00	00	00	00	T

- *Source:*
  - **Car-Hacking dataset** from Hacking and Countermeasure Research Lab
  - http://ocslab.hksecurity.net/Datasets/CAN-intrusion-dataset

Intrusion Detection - Dataset

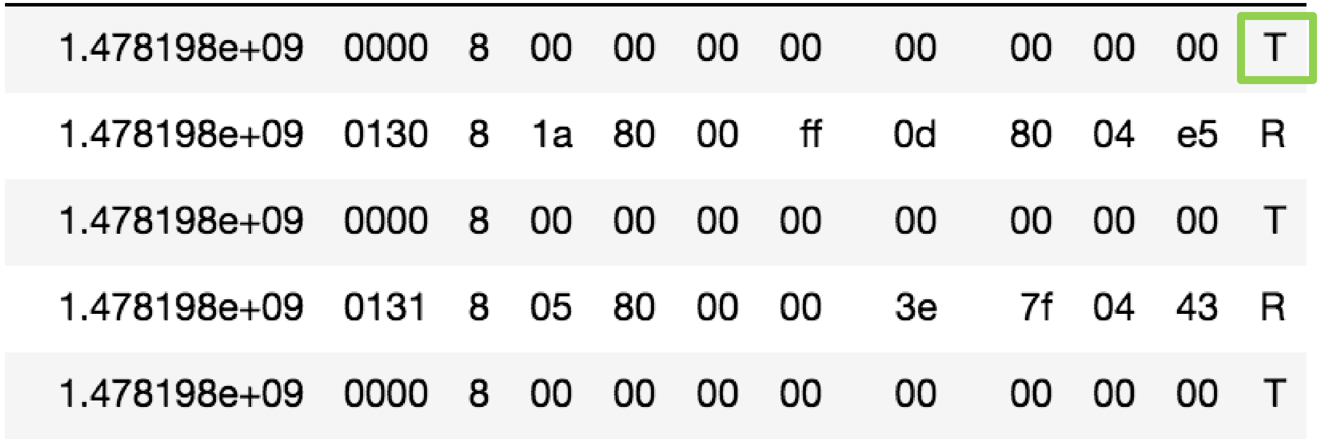
Payload Content



1.478198e+09	0000	8	00	00	00	00	00	00	00	00	T
1.478198e+09	0130	8	1a	80	00	ff	0d	80	04	e5	R
1.478198e+09	0000	8	00	00	00	00	00	00	00	00	T
1.478198e+09	0131	8	05	80	00	00	3e	7f	04	43	R
1.478198e+09	0000	8	00	00	00	00	00	00	00	00	T

- *Source:*
  - **Car-Hacking dataset** from Hacking and Countermeasure Research Lab
  - http://ocslab.hksecurity.net/Datasets/CAN-intrusion-dataset

Intrusion Detection - Dataset

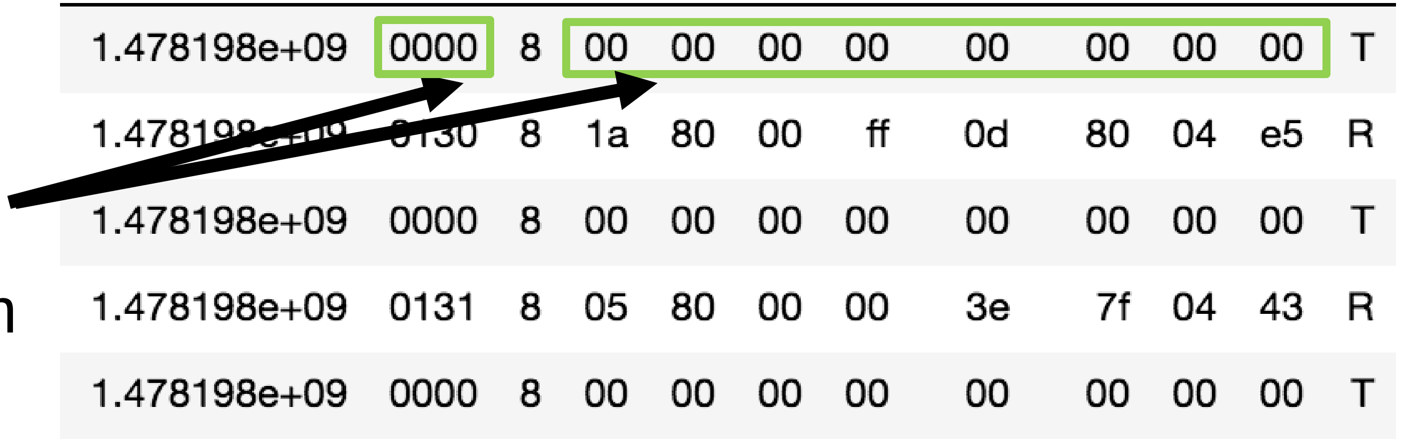


											Label
1.478198e+09	0000	8	00	00	00	00	00	00	00	00	T
1.478198e+09	0130	8	1a	80	00	ff	0d	80	04	e5	R
1.478198e+09	0000	8	00	00	00	00	00	00	00	00	T
1.478198e+09	0131	8	05	80	00	00	3e	7f	04	43	R
1.478198e+09	0000	8	00	00	00	00	00	00	00	00	T

- *Source:*
  - **Car-Hacking dataset** from Hacking and Countermeasure Research Lab
  - http://ocslab.hksecurity.net/Datasets/CAN-intrusion-dataset

Intrusion Detection - Attacks

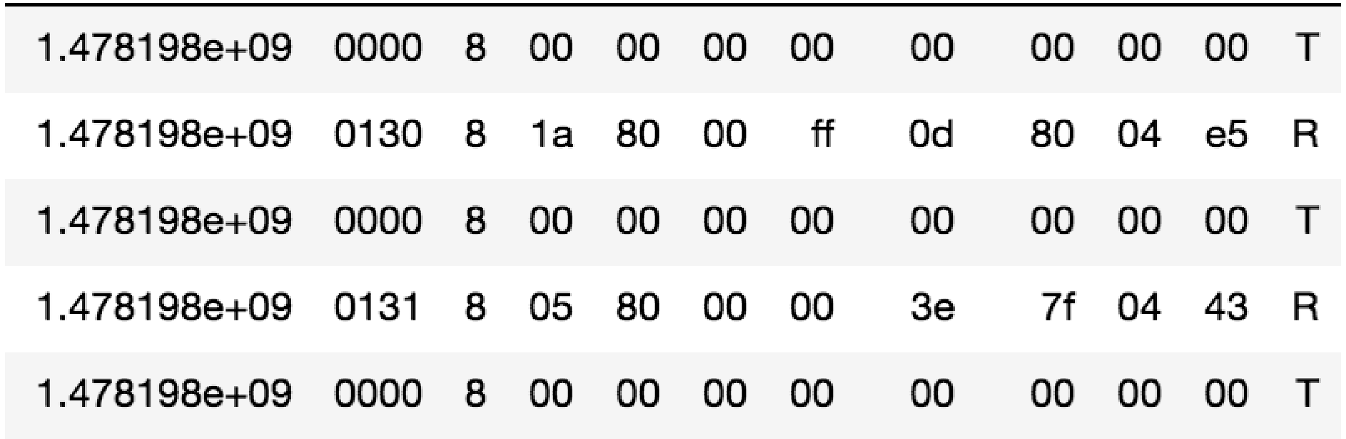
- **Denial of Service** (DoS)
  - **Rapid** (every 0.3 ms)
  - CAN ID is set to **0X00**
  - Overwhelm the system



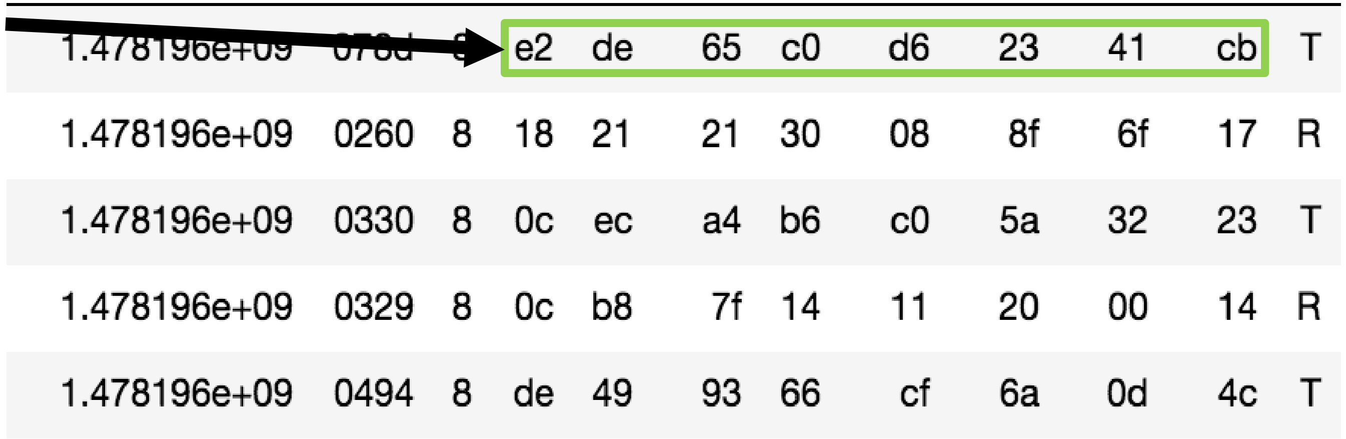
1.478198e+09	0000	8	00	00	00	00	00	00	00	00	T
1.478198e+09	0130	8	1a	80	00	ff	0d	80	04	e5	R
1.478198e+09	0000	8	00	00	00	00	00	00	00	00	T
1.478198e+09	0131	8	05	80	00	00	3e	7f	04	43	R
1.478198e+09	0000	8	00	00	00	00	00	00	00	00	T

Intrusion Detection - Attacks

- **Denial of Service** (DoS)
  - **Rapid** (every 0.3 ms)
  - CAN ID is set to **0X00**
  - Overwhelm the system
- **Fuzzing (as an attack)**
  - Payload is **randomized**
  - Trigger unexpected behavior



1.478198e+09	0000	8	00	00	00	00	00	00	00	00	T
1.478198e+09	0130	8	1a	80	00	ff	0d	80	04	e5	R
1.478198e+09	0000	8	00	00	00	00	00	00	00	00	T
1.478198e+09	0131	8	05	80	00	00	3e	7f	04	43	R
1.478198e+09	0000	8	00	00	00	00	00	00	00	00	T



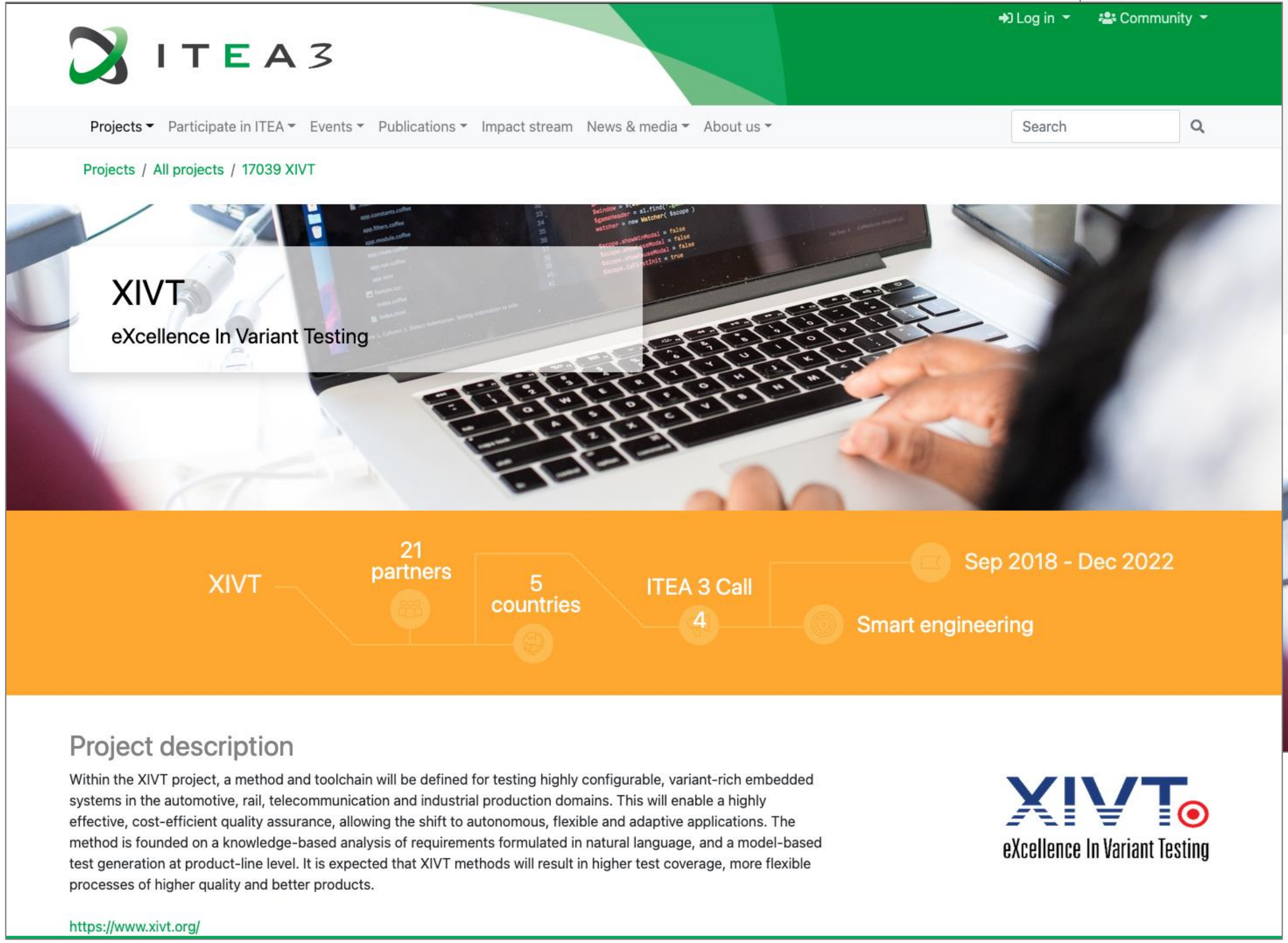
1.478196e+09	076d	8	e2	de	65	c0	d6	23	41	cb	T
1.478196e+09	0260	8	18	21	21	30	08	8f	6f	17	R
1.478196e+09	0330	8	0c	ec	a4	b6	c0	5a	32	23	T
1.478196e+09	0329	8	0c	b8	7f	14	11	20	00	14	R
1.478196e+09	0494	8	de	49	93	66	cf	6a	0d	4c	T

Security in CAVs

- How do we detect attacks?
- How do we test that vehicles are resilient when under attack?
- How do we test for security vulnerabilities?
- …

- These are all questions we are exploring in the XIVT Project!

https://www.xivt.org/



Projects / All projects / 17039 XIVT

XIVT

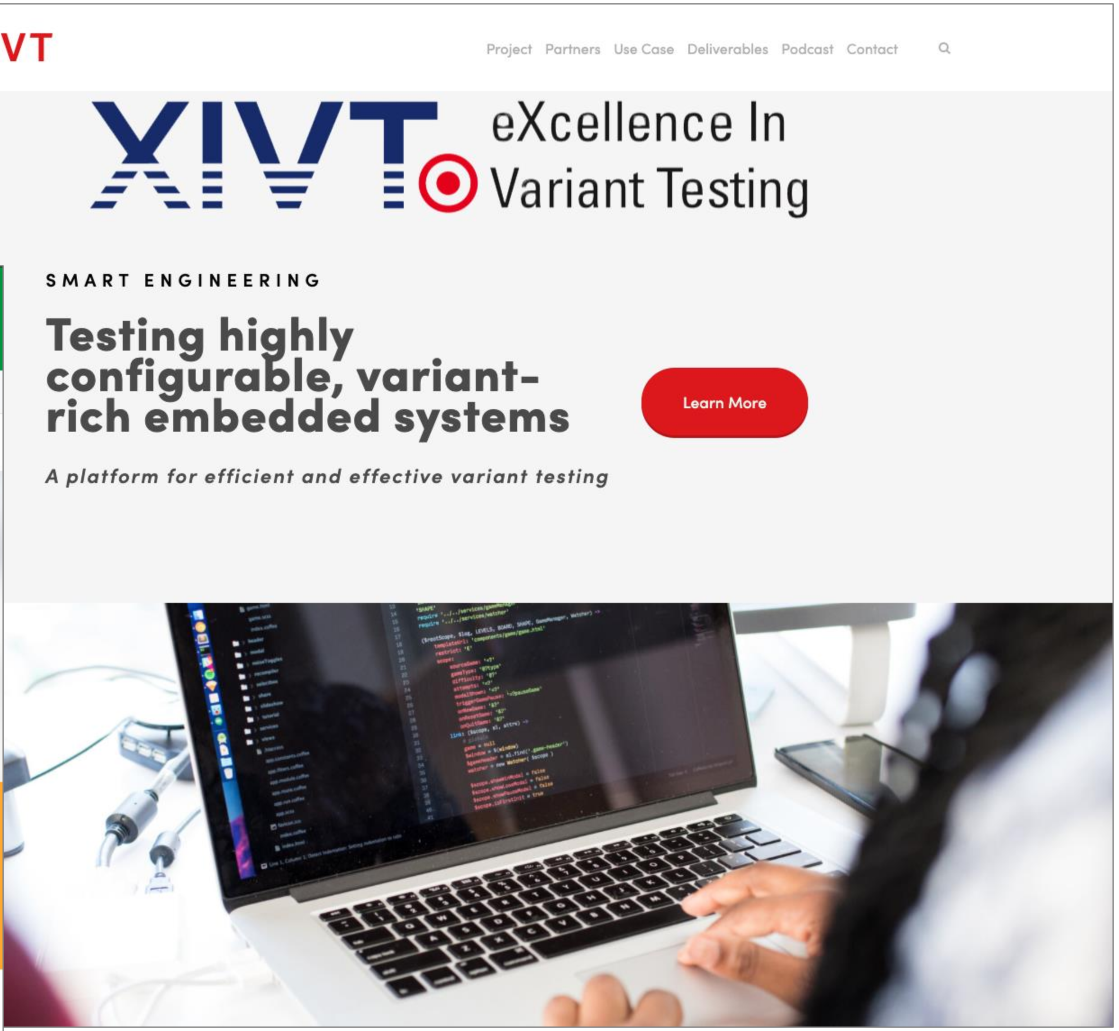
eXcellence In Variant Testing

XIVT — 21 partners — 5 countries — ITEA 3 Call 4 — Sep 2018 - Dec 2022 — Smart engineering

Project description

Within the XIVT project, a method and toolchain will be defined for testing highly configurable, variant-rich embedded systems in the automotive, rail, telecommunication and industrial production domains. This will enable a highly effective, cost-efficient quality assurance, allowing the shift to autonomous, flexible and adaptive applications. The method is founded on a knowledge-based analysis of requirements formulated in natural language, and a model-based test generation at product-line level. It is expected that XIVT methods will result in higher test coverage, more flexible processes of higher quality and better products.

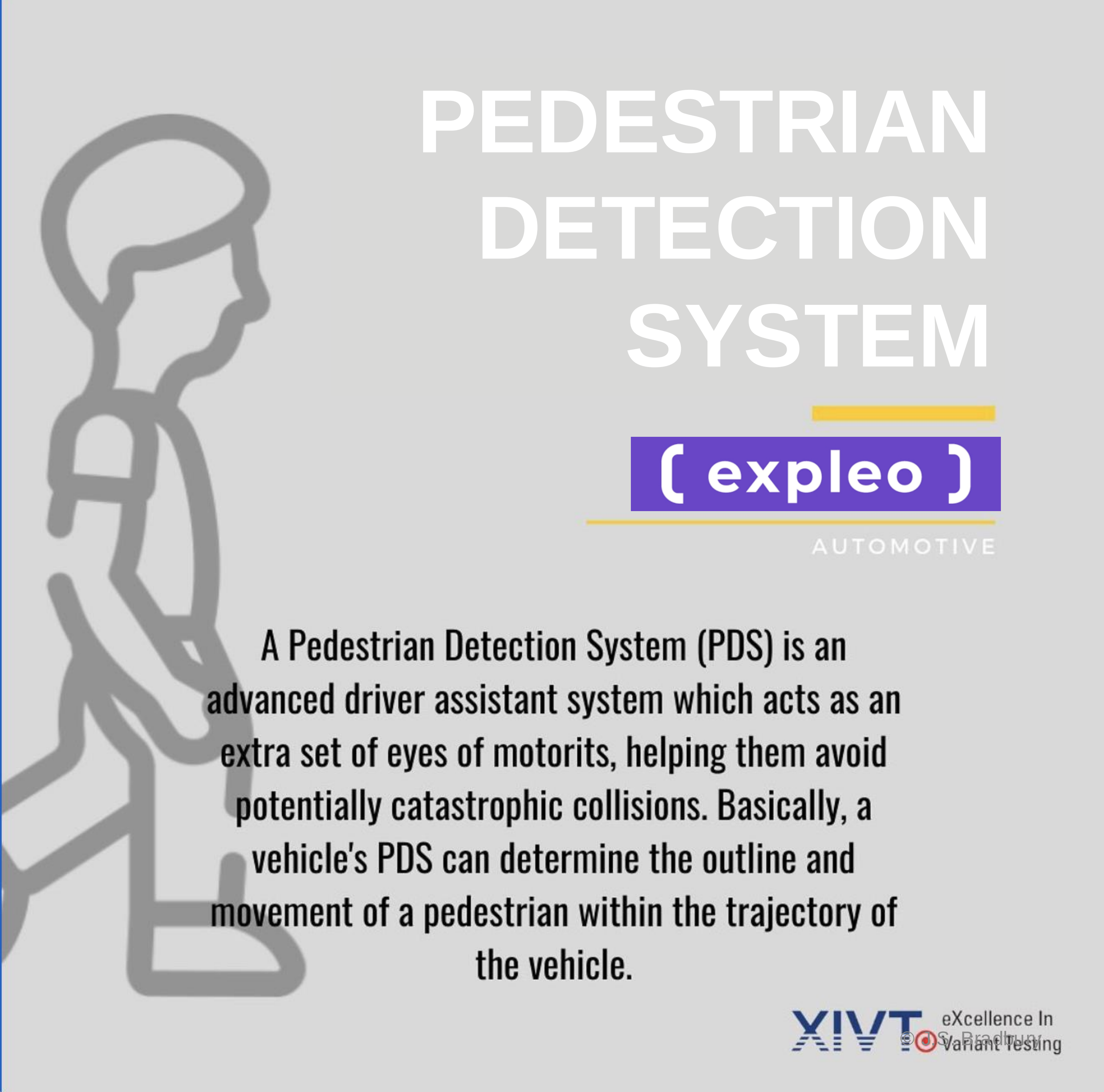
https://www.xivt.org/



SMART ENGINEERING

Testing highly configurable, variant-rich embedded systems

A platform for efficient and effective variant testing



PEDESTRIAN DETECTION SYSTEM

(expleo)
AUTOMOTIVE

A Pedestrian Detection System (PDS) is an advanced driver assistant system which acts as an extra set of eyes of motorits, helping them avoid potentially catastrophic collisions. Basically, a vehicle's PDS can determine the outline and movement of a pedestrian within the trajectory of the vehicle.

XIVT
eXcellence In Variant Testing

XIVT Use Case

- The cybersecurity challenges in the Pedestrian Detection System(PDS)/ Advanced Driver Assistant System (ADAS) use case

Security Testing & Analysis

Summary

- Quality vs. Security
- Introduced testing (penetration testing, fuzzing) and static analysis for security
- Reviewed a security case study – Connected Autonomous Vehicles (CAVs)

Readings:

- “Fuzzing: Hack, Art and Science” by Patrice Godefroid
    - https://patricegodefroid.github.io/public_psfiles/Fuzzing-101-CACM2020.pdf

Acknowledgements:

- I’d like to acknowledge and thank the XIVT project partners who contributed to the background content in today’s CAVs case study – in particular, Naida Tania and Michael who created the car hacking dataset slides