



# Fuzzing the MCU of Connected Vehicle for Security and Safety

Hao Chen- Security Expert – Li Auto



# About Me

Hao Chen(@flankersky)

- Security Expert @Li Auto
- Bug hunting in Android, Linux kernel
- Connected Car Security & Hardware Security Newbie

# Overview



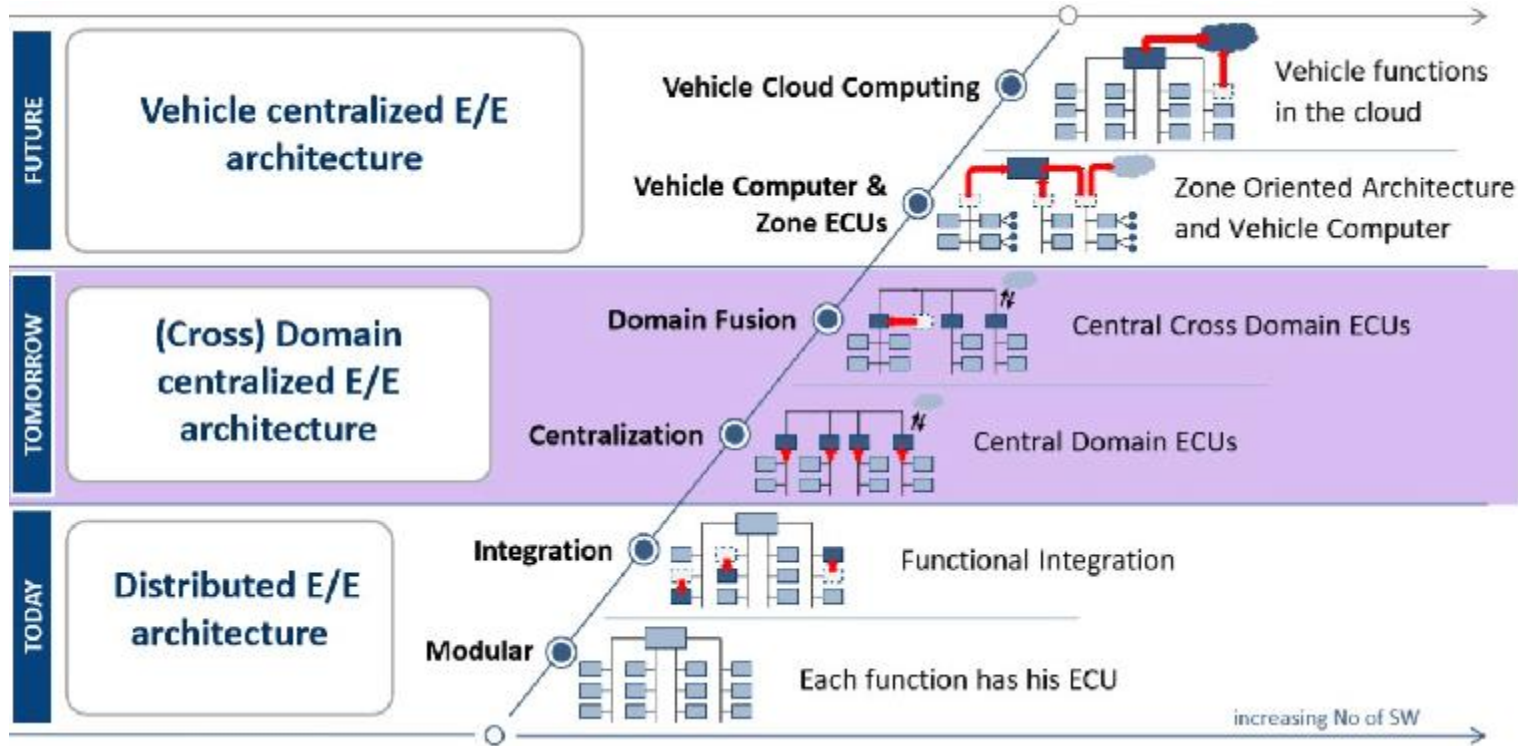
HITBSecConf  
2022 Singapore

- BackGround
- McuFuzz Design
- McuFuzz Demo
- Conclusion

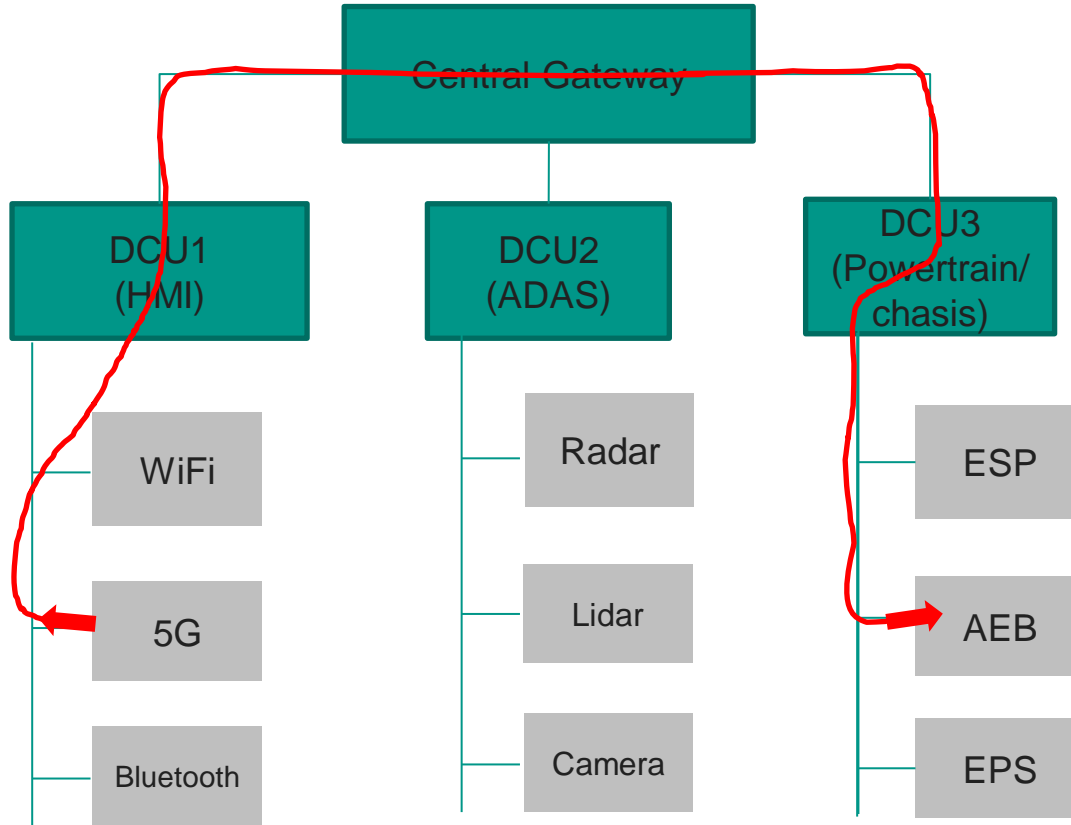


# BackGround

# BackGround - The usual attack vector



# BackGround - The usual attack vector



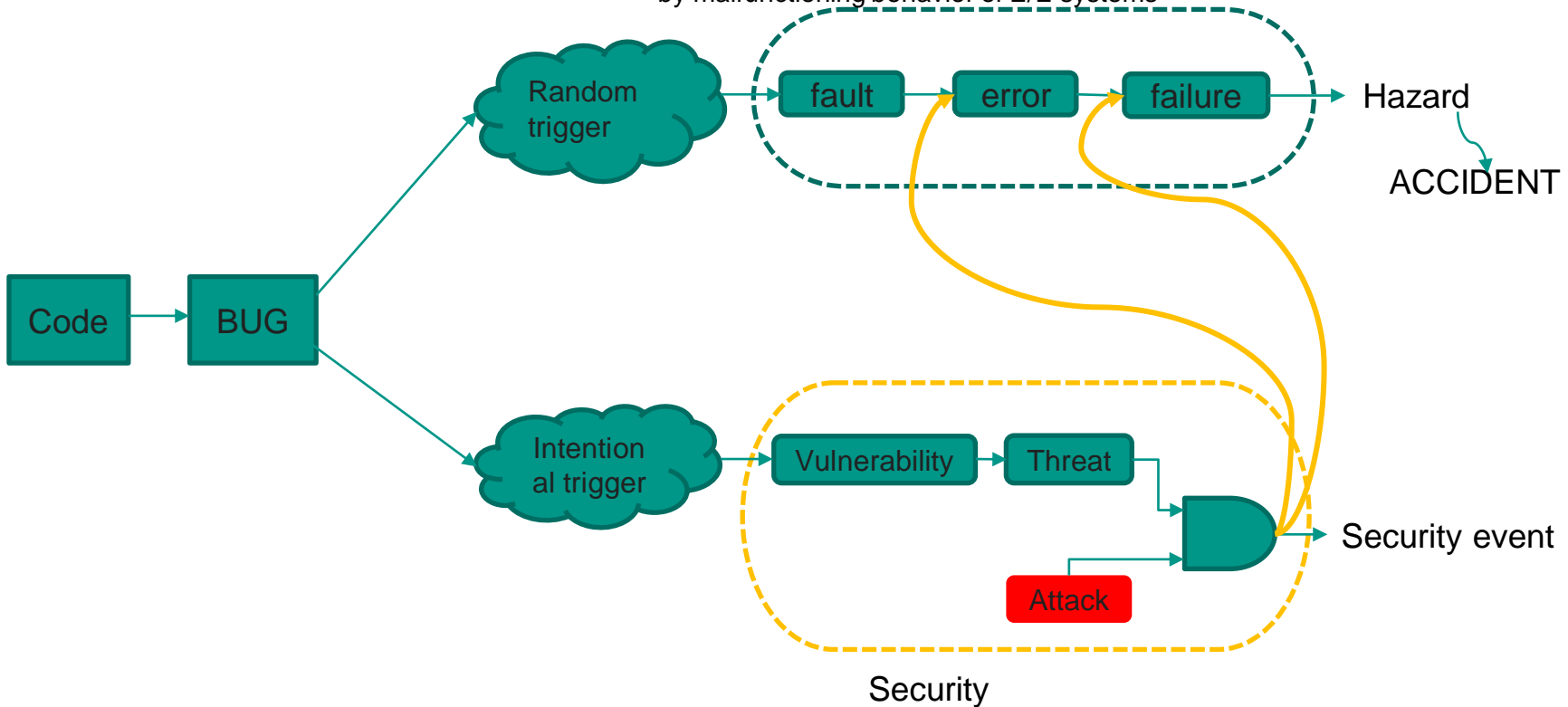
The Powertrain/chasis Domain:

- Hacker's ultimate goal
- Safety Critical
- Security Critical

# BackGround – Security vs Safety(software)



**Safety:** Absence of unreasonable risk due to hazards caused by malfunctioning behavior of E/E systems



# BackGround -Current MCU software Test



- Code Walk-through
- Semi-formal verification
- Formal verification
- Interface test
- Unit test
- Fault injection test
- Static code analysis
- Data flow analysis
- statement coverage
- branch coverage
- MC/DC

Coverage-guided fuzzing maybe helpful.

**There are never enough ways to test.**





# The Mcu Fuzzing

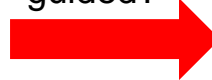


# McuFuzz – What we have & need

## Resources of a processor for Vehicles

- Up to 3x Arm Cortex-M7 DCLS
- Up to 8 MB SRAM
- Running AUTOSAR or FreeRTOS

Coverage  
guided?



◆ How to trace memory access

◆ How to trace code coverage

## What's the problem

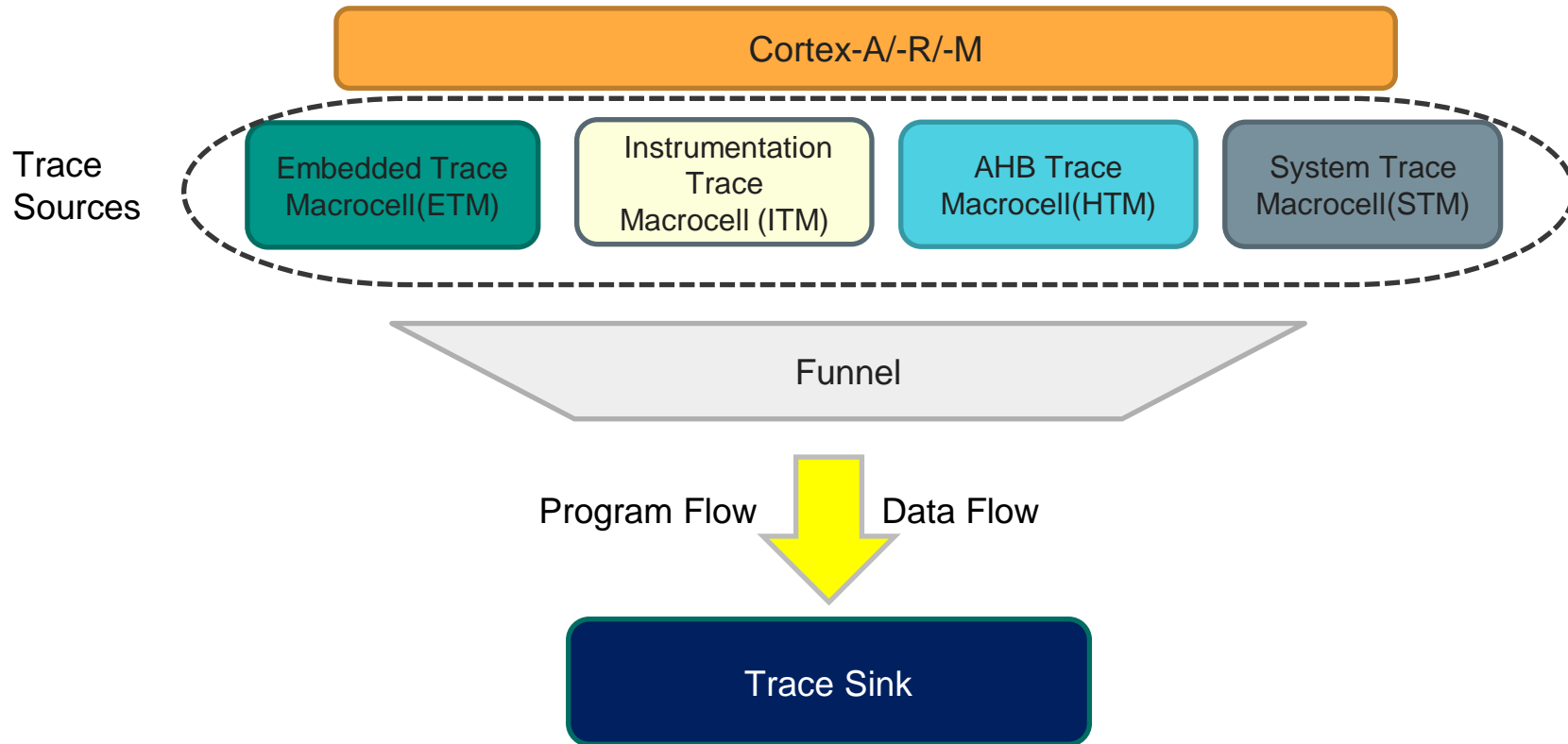
- SRAM is really small
- real-time OS, no MMU
- Gcc or Clang are rarely used

Solution



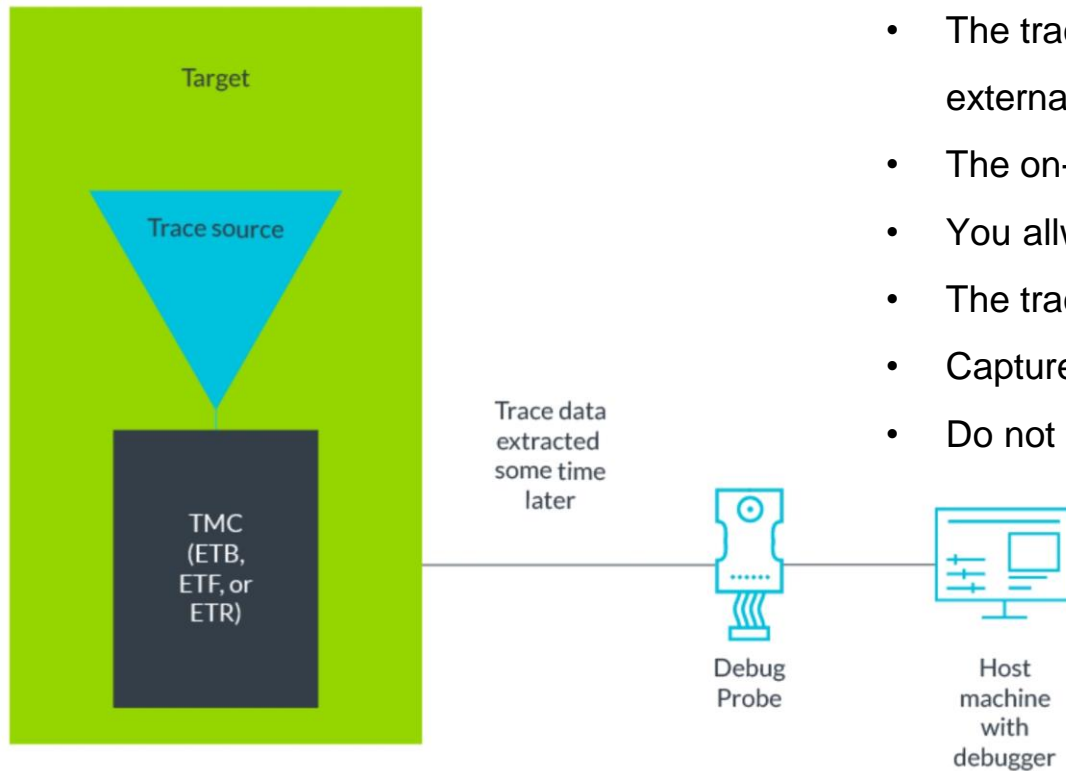
Hardware Assisted

# McuFuzz - Introduction to ARM Trace





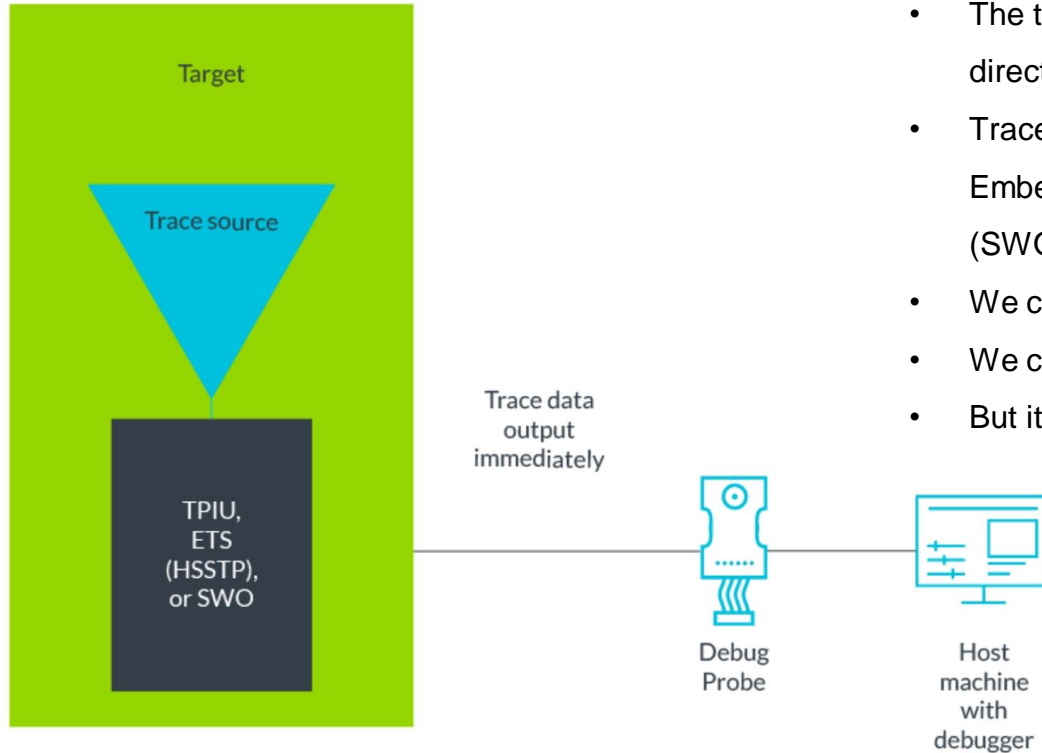
# McuFuzz – ETM on-chip trace



- The trace data is on chip and is exported to the external debugger.
- The on-chip buffer is usually small.
- You always need filtering.
- The trace data is heavily compressed.
- Capture trace at a much higher speed.
- Do not require any trace pins, JTAG is enough.



# McuFuzz – ETM off-chip trace



- The trace data is output from the target to a debug unit or directly to the external debugger.
- Trace data is output by the Trace Port Interface Unit (TPIU), Embedded Trace Streamer (ETS), or Serial Wire Output (SWO) that is on the target to an external debugger.
- We can have more buffer to store trace data.
- We can trace over a long period.
- But it need additional hardware pins.

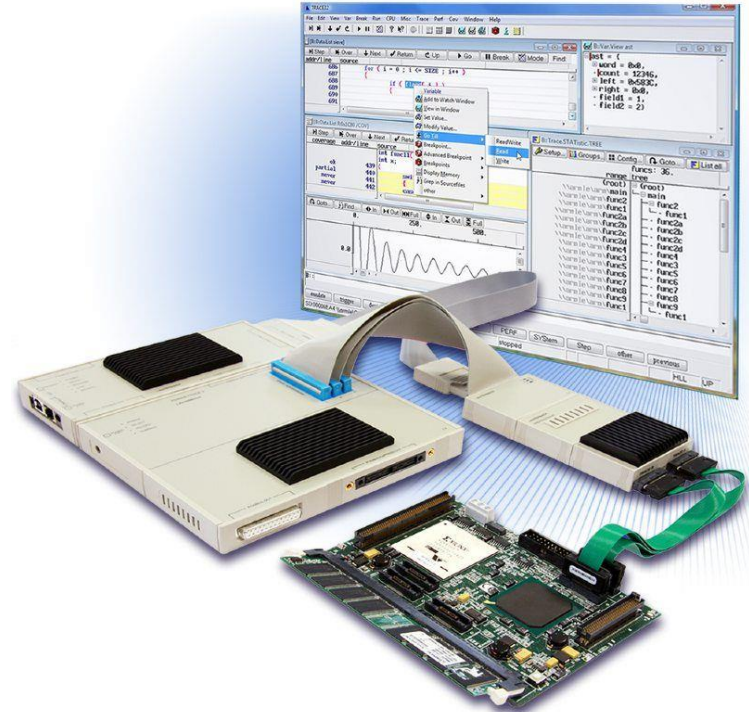


# McuFuzz - ETM's features Summary

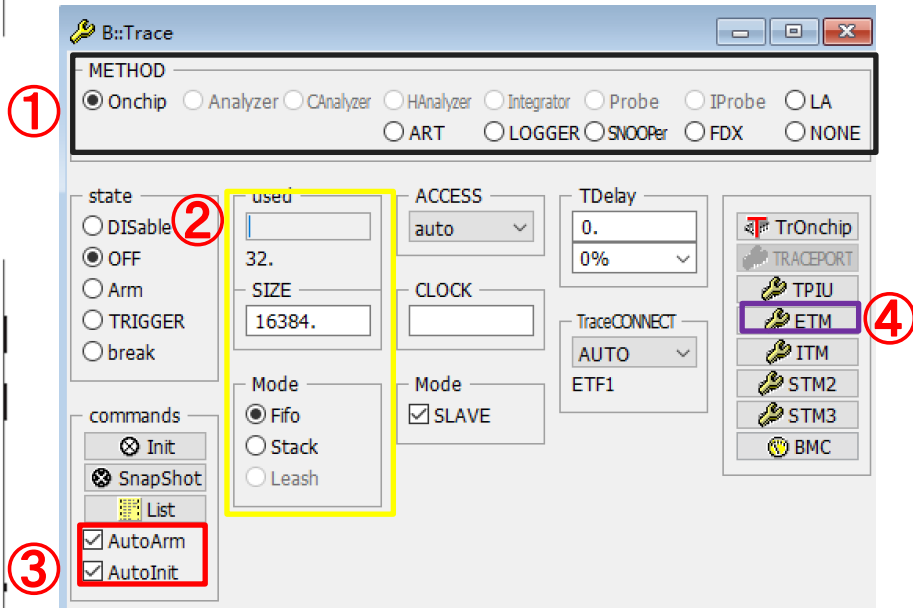
- A trace source, part of ARM coresight
- Instruction and data trace
- ETM supports trace filtering
- Can generate cycle-accurate trace
- Can insert timestamps into trace data
- Support on-chip and off-chip trace
- Supported in most Arm-based systems

# McuFuzz- Use Trace32 to trace

- PowerView, a universal GUI
- PowerDebug tools for debugging
- PowerTrace tools for program/data flow trace
- Support Cortex-A/-R/-M, TriCore, RISC-V, Power Architecture



# McuFuzz – Trace32: Enable ETM



## ◆ Trace method

Trace sink is on-chip buffer

## ◆ On-chip trace buffer

Trace buffer usage status

Fifo mode: If the trace is full, new records will overwrite older records.

## ◆ AutoInit

- Trace memory contents is erased and previous records are no longer visible
- The trigger unit is set to its initial state.

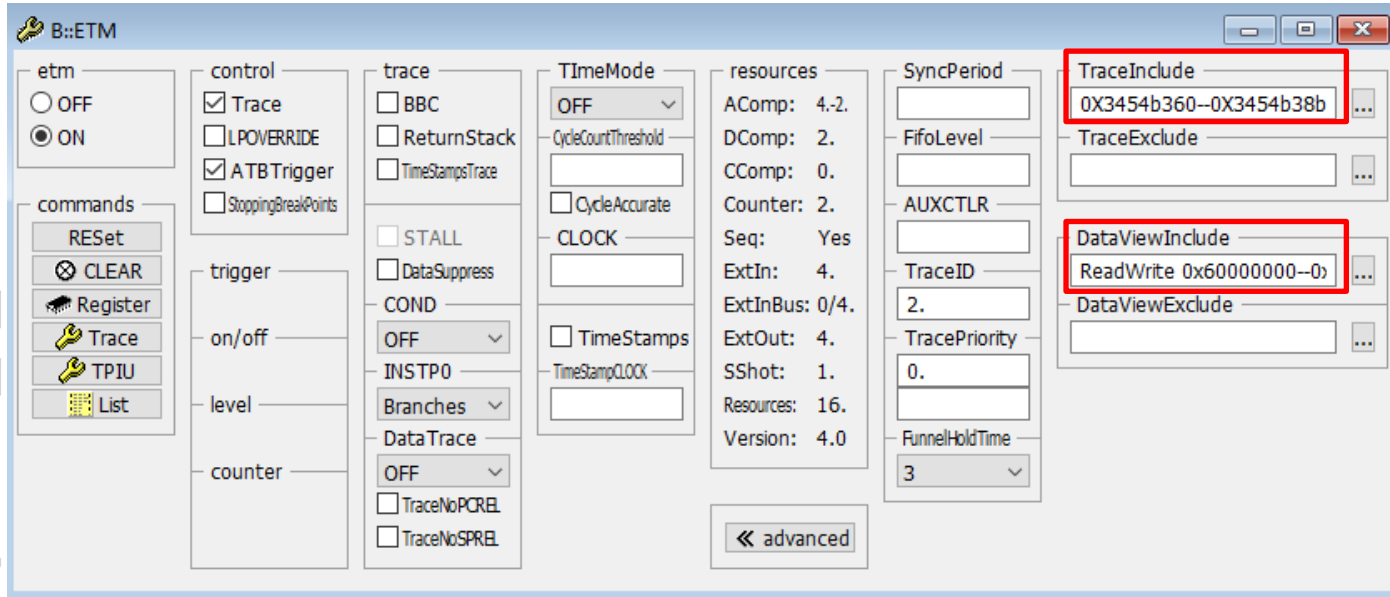
## ◆ AutoArm

- Recording and if available triggering is prepared whenever the program execution is started.
- Recording and if available triggering is stopped whenever the program execution is stopped.

## ◆ ETM



# McuFuzz – ETM trace filter



- Code Filter example  
ETM.TraceInclude Execute 0x34000000-- 0x34000fff 0x35000000-- 0x35000fff
- Memory access filter example  
ETM.DataViewInclude ReadWrite 0x60000000--0x61ffffff

# McuFuzz – The coverage result



The screenshot shows the McuFuzz interface with the following components:

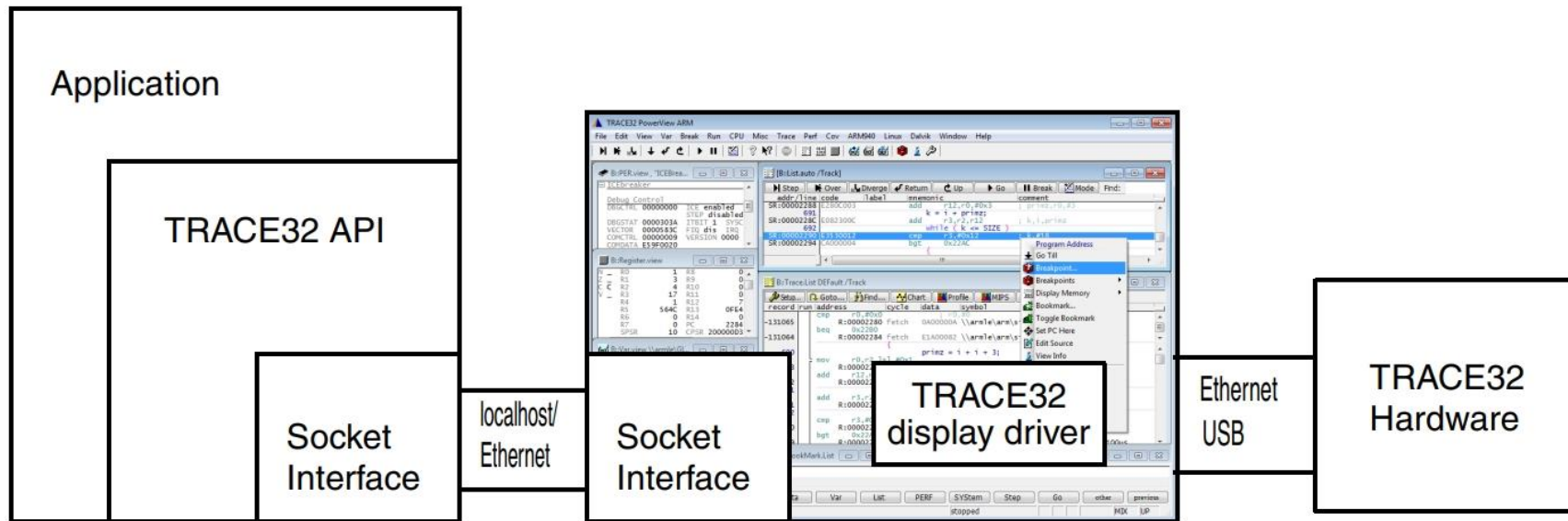
- Left Panel:** Controls for the fuzzing process, including METHOD (Incremental selected), state (ON selected), Option (StaticInfo checked), SourceMetric (Statement), and commands (+ ADD, Init, RESet).
- Main Window:** A table titled "B::COV.ListLine" displaying coverage data for various memory addresses. The table includes columns for address, tree, coverage, objectcode, and a bar chart representing coverage percentage. The bar chart has markers for 0%, 50%, and 100%.

address	tree	coverage	objectcode	0%	50%	100	branches	ok	taken	not taken
P:342551AC--342551C1		never	0.000%					0.	0.	0.
P:342551C2--342551D1		never	0.000%					0.	0.	0.
P:342551D2--342551D5		never	0.000%					0.	0.	0.
P:342551D6--3425524F		never	0.000%					0.	0.	0.
P:34255278--3425540D		partial	41.871%				0.000%	0.	0.	2.
P:34255278--34255297		ok	100.000%				16.666%	0.	0.	0.
P:34255278--34255279		ok	100.000%					0.	0.	0.
P:34255278--34255279		ok	100.000%					0.	0.	0.
P:3425527A--34255287		ok	100.000%					0.	0.	0.
P:34255288--34255289		ok	100.000%					0.	0.	0.
P:3425528A--34255291		ok	100.000%					0.	0.	0.
P:34255292--34255295		ok	100.000%					0.	0.	0.
P:34255296--34255297		ok	100.000%					0.	0.	0.
P:34255298--342552D0		ok	100.000%					0.	0.	0.
P:34255298--342552A5		ok	100.000%					0.	0.	0.
P:34255298--342552A5		ok	100.000%					0.	0.	0.
P:342552A6--342552AF		ok	100.000%					0.	0.	0.
P:342552B0--342552B5		ok	100.000%					0.	0.	0.
P:342552B6--342552C8		ok	100.000%					0.	0.	0.
P:342552CC--342552D3		ok	100.000%					0.	0.	0.
P:342552D4--342552D9		ok	100.000%					0.	0.	0.
P:342552DA--342552DD		ok	100.000%					0.	0.	0.
P:342552DE--34255307		partial	57.142%				50.000%	0.	0.	1.
P:342552DE--342552E7		not taken	80.000%				50.000%	0.	0.	1.
P:342552DE--342552E7		not taken	80.000%				50.000%	0.	0.	1.
P:342552E8--342552ED		ok	100.000%					0.	0.	0.
P:342552EE--342552F7		ok	100.000%					0.	0.	0.
P:342552F8--34255305		never	0.000%					0.	0.	0.
P:34255306--34255307		never	0.000%					0.	0.	0.
P:34255308--34255335		not taken	95.652%				50.000%	0.	0.	1.
P:34255308--34255313		ok	100.000%					0.	0.	0.
P:34255308--34255313		ok	100.000%					0.	0.	0.
P:34255314--34255317		ok	100.000%					0.	0.	0.
P:34255318--3425531B		ok	100.000%					0.	0.	0.
P:3425531C--3425531D		not taken	0.000%				50.000%	0.	0.	1.
P:3425531E--3425532F		ok	100.000%					0.	0.	0.
P:34255330--34255331		ok	100.000%					0.	0.	0.
P:34255332--34255335		ok	100.000%					0.	0.	0.
P:34255336--34255375		never	0.000%					0.	0.	0.
P:34255336--34255341		never	0.000%					0.	0.	0.

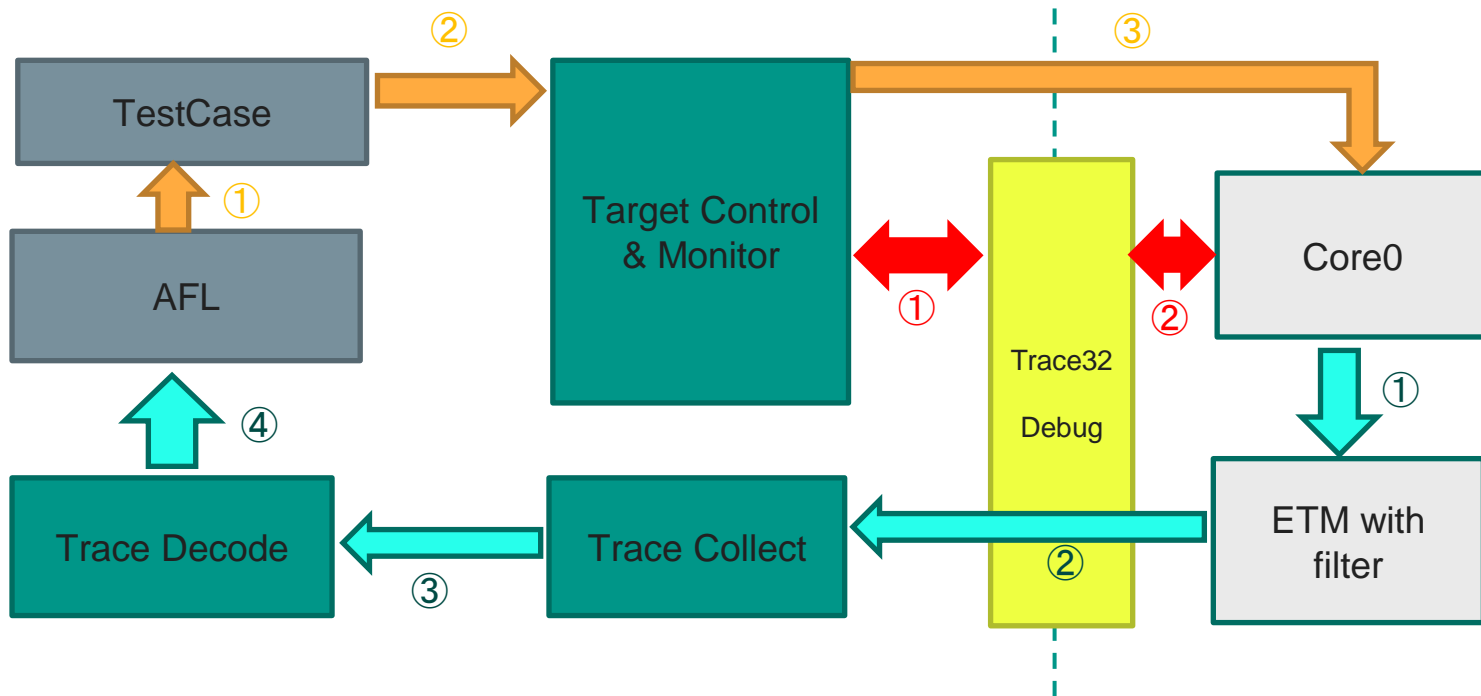
This will slow down the fuzzing speed.

# McuFuzz – Trace32 API

Application ---> TRACE32 API ---> TRACE32 application --> TRACE32  
(C Functions)                      (sockets)                      (HW interface)



# McuFuzz- The mcu fuzzing framework





# McuFuzz – The advantages

- Compiler independent
- No need to recompile code
- No code instrumentation required
- Coverage-guided



**Demo**

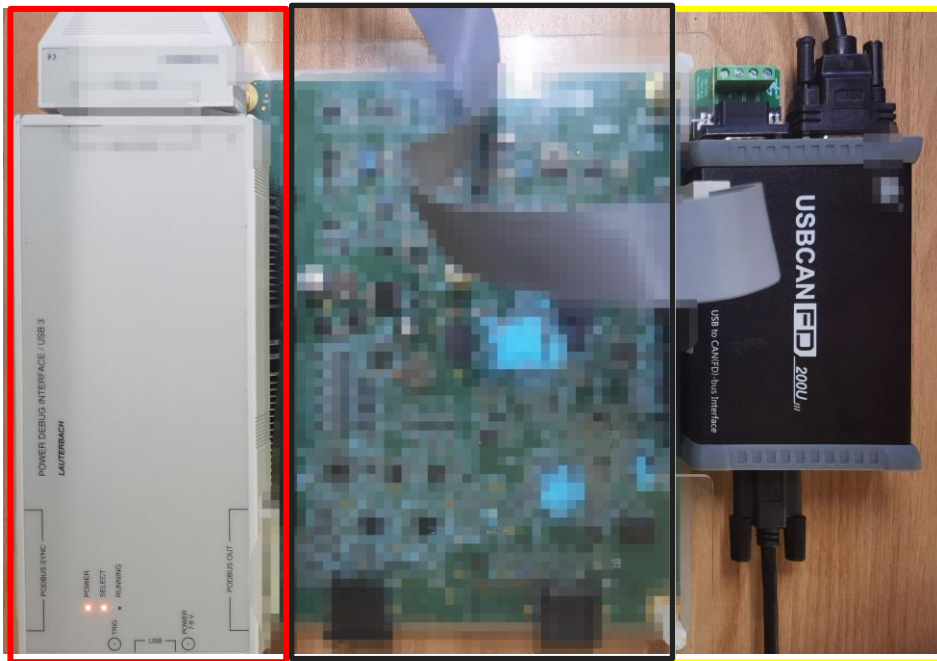
# Demo – Can service Fuzzing



Trace32

PowerDebug

- Control Target:  
Run, break, configure ETM
- Get ETM analyzed data
- Monitor crash



Target MCU

USBCAN Device

- Collect init seed corpus
- Send mutated data to target

# Demo - Can service Fuzzing



The screenshot displays the TRACE32 PowerView for ARM 1 [Power Debug USB @] interface. The BiTrace configuration panel is visible, with the 'METHOD' dropdown set to 'Onchip'. A red arrow points to the 'Onchip' option. The 'state' section has 'Arm' selected. The 'used' field is set to 176, and the 'SIZE' field is set to 16384. The 'commands' section has 'Init', 'SnapShot', and 'List' checked. The 'TraceCONNECT' dropdown is set to 'AUTO'. The 'TrOnchip' panel shows 'TP1U', 'ETM', 'ITM', 'STM2', 'STM3', and 'BMC' options. A Windows PowerShell terminal window is open, showing the execution of the 'afl-fuzz.exe' command. The terminal output includes the following text:

```
PS D:\workspace\McuFuzz> D:\workspace\McuFuzz\win afl\build32\bin\Debug\afl-fuzz.exe -i D:\workspace\McuFuzz\in -o D:\workspace\McuFuzz\out -t 20000 -nargs 3 -- C:\Python\Python36-32\python.exe D:\workspace\McuFuzz\targetHarness\targetMain.py --@ WinAFL 1.16b by <ifratric@google.com>
Based on AFL 2.43b by <lcantuf@google.com>
Debug: winafl32_options_init
[+] You have 16 CPU cores with average utilization of 20%.
[+] Try parallel jobs - see afl_docs\parallel_fuzzing.txt.
[*] Checking CPU core loadout...
[+] Found a free CPU core, binding to #0.
[+] Process affinity is set to 1.
[*] Setting up output directories...
[+] Output directory exists but deemed OK to reuse.
[*] Deleting old session data...
[+] Output dir cleanup successful.
[*] Scanning 'D:\workspace\McuFuzz\in'...
[+] No auto-generated dictionary tokens to reuse.
[*] Creating hard links for all input files...
[*] Attempting dry run with 'id_000000'...
Debug: winafl32_options_init
len = 64, map size = 26, exec speed = 103275 us
[!] WARNING: Instrumentation output varies across runs.
[*] Attempting dry run with 'id_000001'...
```





# Conclusion



## Conclusion

- ◆ Coverage guided fuzzing on MCU is possible
- ◆ ETM and Trace32 is really helpfull
- ◆ This prototype is proven effective in our product

## Future works:

- ◆ Improve fuzzing speed
- ◆ More target fuzzing practice
- ◆ Off-chip trace is in progress

<https://github.com/flankersky/mcufuzz>

flank3rsky@gmail.com



Thank You!