SAND2020-8046C

# Dynamic Address Validation Array: A Moving Target Defense Protocol for CANBus

Richard Brown
*Department of Computer Science*
*Tennessee Tech University*
Cookeville, USA
rgbrown42@students.tntech.edu

Alex Marti
*Department of Computer Science*
*Tennessee Tech University*
Cookeville, USA
ajmarti42@students.tntech.edu

Chris Jenkins
*Sandia National Laboratories*
*cdjenk@sandia.gov*

Susmit Shannigrahi
*Department of Computer Science*
*Tennessee Tech University*
Cookeville, USA
sshannigrahi@tntech.edu

***Abstract*—This paper discusses a novel moving target defense protocol, Dynamic Addressed Validation Array (DAVA), to mitigate the common CANbus vulnerability of an unauthorized entity misappropriating components in the vehicle either through sniffing or replay attacks by obscuring ECU IDs. Using a dynamically allocated array that is updated and validated frequently, it limits an attacker's visibility when performing reconnaissance. The protocol strives to be as minimally invasive and lightweight for application in CANbus while still being secure. This paper discusses the DAVA protocol, a proof of concept implementation, and initial performance measurement. This paper explains how DAVA is able to provide a robust security framework for CANbus without the need for large amount of storage or protocol modification.**

***Index Terms*—CANbus, Moving Target Defense, ECU, Device ID updating, Anti-Reconnaissance.**

## I. Introduction

CANbus supports a non-IP based network with many devices and microcontrollers that communicate on the bus. Ever since its creation in 1986, it grew more widespread, finding use in vehicles, robotics, and even prosthetics [1]. Unfortunately, the CAN protocol lacks encryption or authentication methods [2]. Due to this lack of security, an attacker can obtain control of the devices connected by a CANbus if they can access the bus. [3]. Researchers have successfully conducted and demonstrated such attacks with production vehicles when in physical contact with the bus [4]. Even wireless attacks are possible depending on the vehicle, such as the Tesla that was hacked when its built-in web browser connected to a malicious hot-spot [3].

The standard security measures such as encryption don't easily apply to CAN because it uses eight byte packets that are too small for most standard security mechanisms [2]. Besides, the packets arrive and decode within a deadline, making the use of multiple packets for carrying an encrypted payload difficult [3].

Moving Target Defense (MTD) techniques do not directly depend on the standard encryption methods, which makes it an excellent alternative for securing a CAN-based network [5]. MTD reduces the attack surface by periodically changing the device IDs of the nodes [6]. As a result, even when a bus is compromised, the attacker may not know the device ID (which are constantly changing), effectively preventing them from communicating with the devices.

This paper proposes the creation of a MTD algorithm that applies to vehicular CANbus networks. The proposed protocol, Dynamic Address-Validation Array (DAVA), prevents against the most costly intrusion attack against a CANbus, Reconnaissance and replay attacks [7]. DAVA is a lightweight protocol that minimally impacts the normal functions of CANbus, preserves CAN's quick responsiveness, and at the same time, still mitigates security concerns. Using fully distributed but identical copies of the same dynamically allocated array, DAVA allows CAN to make changes to device IDs - however, these changes does not affect the general usability of the bus and prevents an sniffing attacker from communicating with the CAN ECUs.

This paper both presents a novel MTD algorithm for CANbus, one of the very few works in this area, and illustrates it with a simulator. The simulator tests the protocol for performance and overhead, and shows that the algorithm does not pose significant overhead and does not require adoption of standard encryption algorithms to be secure.

Section two discusses related work in this field. Section three presents the protocol details of DAVA. Section four discusses the simulation setup. Section five involves the evaluation followed by future work and conclusions.

## II. Background

Similar to DAVA, a research team at Sandia National Labs implemented a Moving Target Defense algorithm for MIL-STD-1553, another non-IP based network [4]. In MIL-STD-1553, the bus controller sends a message to the desired remote terminal and instructs it to act. The algorithm starts by instructing the bus controller and the devices on the network to

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

create identical states that stores randomly-generated addresses ranging from one to thirty. The device sets its address to the address stored at the first index in the state, and the bus controller sends its message to that address. Then, the bus communicates as usual until after a fixed number of iterations where the device and bus controller now switch to using the second index. This protocol reduces the attack vector by continuously changing the attack surface.

DAVA utilizes a similar idea of shifting the attack surface by making several CAN specific modifications. CANbus requires each node to act on their own accord. CANbus works slightly differently where the recipient's ID determines the priority of a message. In CANbus, each device has a priority ID; messages addressed to the lowest ID device is serviced first, and the highest ID last.

Applying MTD techniques to CANbus is a relatively new idea, and there is not a substantial amount of work in this area. Woo et al. [7] proposed a CAN ID shuffling technique (CIST) that uses network address shuffling and encryption to defend against attacks. Everytime an ECU wants to communicate with another ECU, they encrypt their data messages before sending them, and when they arrive, that ECU changes its ID [7]. In terms of overhead, when two ECUs communicate securely in CIST, they perform an HMAC algorithm twice, and an AES algorithm is performed once [7]. y [7] also noticed a delay in group session key distribution in addition to the normal data transmission delay and CANbus ID changing delay. CIST obtains confidentiality through encryption where it also incurs overehead.

On the contrary, DAVA prioritizes integrity over confidentiality and operates without encryption. DAVA's focus is to create a protocol that is minimally invasive to ordinary CANbus operations. The automotive industry would be more likely to adopt a protocol that is easy to apply, which is the goal of DAVA.

## III. DAVA - Protocol Design

CANbus supports several ECUs over a data bus. Each ECU represents (and often controls) a device in the car, such as the car's engine or its Air Conditioning Unit.

Three methods form the basis of DAVA protocol. Aspects of all three, but especially the second and third methods, combine to create DAVA. -

- **The Master CPU method**: In this method, a master MTD ECU broadcasts periodic update messages to all other ECUs on the bus. Each non-master ECU creates an identical address matrix at the beginning based on a seed (e.g., a nonce) received from the master ECU. The address matrix is a 2D matrix where each row has all possible IDs for a specific device. All ECUs have the same device address matrix at the startup. At the beginning, each device checks the appropriate row in their address matrix for the device they wish to communicate with and uses the value in its first index to address messages to their desired recipient. After some number of cycles (can be user defined), the master ECU broadcasts a unique "update" message that instructs all other ECUs to start using the device IDs at the second index. When the matrix is exhausted, the process starts over.

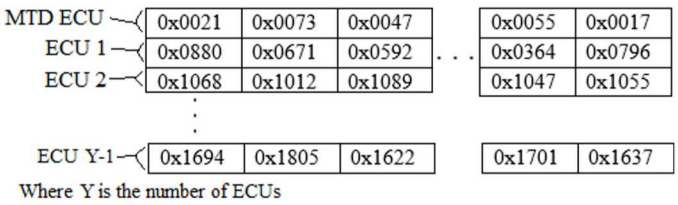


Fig. 1. Every MTD Update, the ECU reassigns its Device ID to what is in the next index of their respective row.

However, the master ECU method does not fit CANbus well. An attacker can impersonate a master ECU and send messages to other ECUs (CANbus is a broadcast bus), essentially creating a denial of service attack. Second, if an ECU receives two messages simultaneously, one, an update message from the MTD ECU, and the other, a legitimate command with high priority (e.g., apply the brakes), the high-priority message is ignored in favor of the update message, which has the highest priority for an operational reason. Assigning a lower priority to an update message means the ECUs change their addresses at differing times, reducing the efficiency of the MTD algorithm, and potentially putting the priority of the messages out of sync.

- **Standalone Address Generation Method** This protocol is essentially the same as the master method but eliminates the necessity to have a master ECU. Initially, a master ECU is responsible for updating the other ECUs by sending update messages to all ECUs. If an ECU needs to communicate, it asks for the Device ID of its desired recipient from the master ECU. The use of an ECU to be solely responsible for updates and scheduling potentially makes the bus fragile.
- **Column-Based Address Generation Method** The other protocol describes a system where all ECUs access to only a "column" of the address matrix rather contain all the current values for each of the ECUs on the bus, such as their Device IDs. Each ECU generates the same address map independently and utilizes that for further communication and updating values.
  Each ECU has the same randomization seed that generates the starting device IDs of each ECU. Because they are using the same seed, they independently make identical device IDs without passing IDs over the bus where an attacker could see them in transit. The CANbus can generate the seed using a combination of various variables (sensor data, time, random numbers, or a combination of these) so it would unique each time. The addressing matrix, or the "column", contains multiple values such as the lower and upper ID bounds for each of the ECU's, so when we generate a device ID for an ECU, it maintains priorities between specific ECUs. The other values are

the number of ECUs on the bus, the starting Device IDs for all ECUs, and the randomization seed are stored in the matrix as Figure.2 shows.

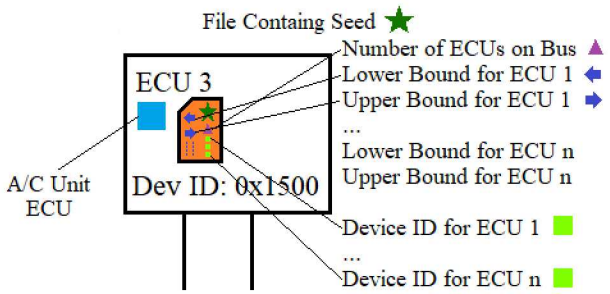


Fig. 2. An Example ECU on the CANbus. The DAVA input file, in orange, contains a randomization seed, the number of ECUs on the bus, the lower and upper bounds for all ECUs, and the Device IDs for all ECUs.

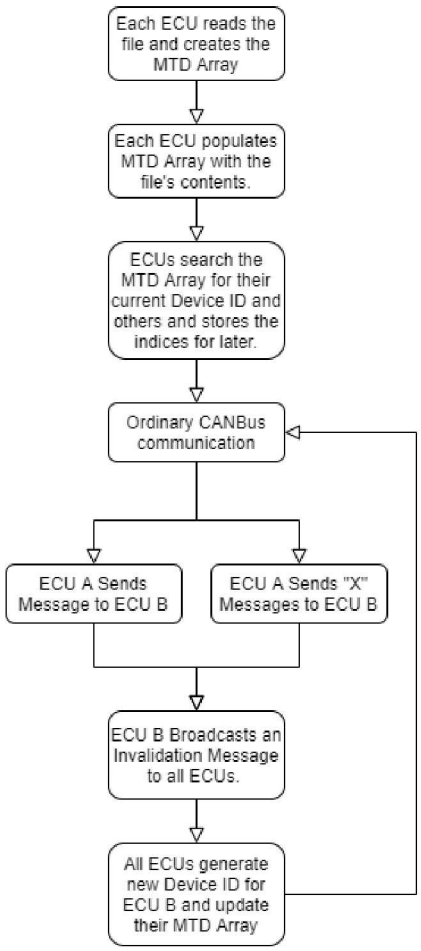


Fig. 3. DAVA Protocol Flowchart

The DAVA protocol combines aspects of the Standalone Address Generation Method and the Sector-Based Address Generation Method. DAVA has three phases that it executes in: Initialization, Update, and Operational. Before the protocol begins, the implemntor inputs every ECU with the same initial input needed to initialize the protocol. This input contains all of the values needed for a certain CANbus configuration in a certain vehicle.

| | **Lower Bound** | **Upper Bound** | **Device ID** | **Randomization Seed** | **Number of ECUs** |
|---|---|---|---|---|---|
| **ECU 1** | 0x0001 | 0x0399 | 0x0001 | 0x91843 | 3 |
| **ECU 2** | 0x0400 | 0x0999 | 0x0500 | 0x91843 | 3 |
| **ECU 3** | 0x1000 | 0x1999 | 0x1500 | 0x91843 | 3 |

Fig. 4. The MTD Array of MTD Nodes that represent the characteristics of every ECU on the CANbus.

### A. *Initialization Phase*

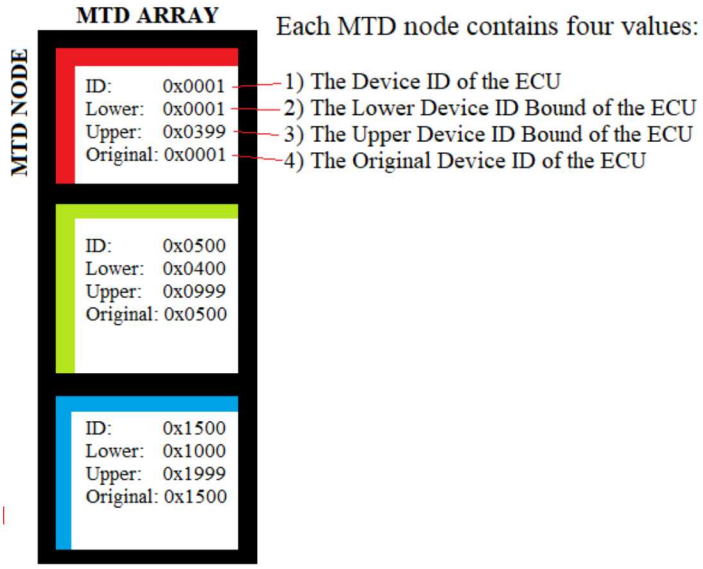


Fig. 5. The MTD Array of MTD Nodes that represent the characteristics of every ECU on the CANbus.

At the start of the Initialization Phase, every ECU creates an ”MTD Array”, which is an array of the same length as the ”Number of ECUs” value in their input file. Every node in the created array signifies a certain ECU on the bus, and each node stores four values: this ECU’s device ID, its lower bound, upper bound, and its original device ID. All ECUs simultaneously create the MTD Array before going further.

Once all ECUs have created their MTD Array, the ECUs pair themself and the other ECUs to the nodes in the MTD Array, so it knows which MTD node refers to what ECU on the bus.

Before DAVA’s execution, ECUs already know their own Device ID, and the Device IDs of the other ECUs on the bus through the initial programming of the CANbus. The Original ID in the MTD Array references the Device ID that a specific ECU on the CANbus starts with before implementing DAVA. Using this, each ECU searches their MTD Array looking for its Device ID in the Original ID field of each node in the MTD

Array. Once the ECU finds its current starting Device ID, that ECU now knows that node refers to that ECU. The ECU also searches the MTD Array for the device IDs of the rest of the ECUs under Original ID of the MTD Array's nodes. When the ECU finds the ID of a certain ECU in a node of the the MTD Array, that ECU now knows that node is refering to that ECU. Through the above method, the ECU pairs its pre-DAVA devce ID information with the new MTD Array, and now uses the MTD Array instead to address data messages to other ECUs. This concludes the initialization phase.

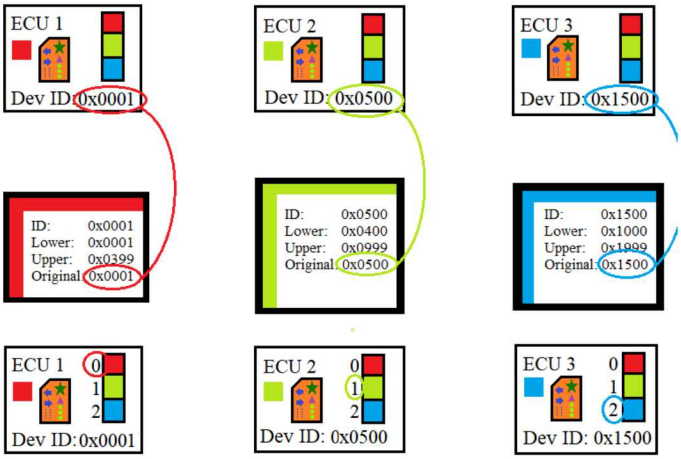


Fig. 6. ECUs search through the MTD Array under the "original" value for the same device ID as their own. Once found, the ECU now knows which MTD Node it is, and when to update its own device ID.

### B. Operational Phase

The operational phase starts after the initialization phase ends. DAVA performs no actions during the operational phase. ECUs perform their standard operations like non-DAVA CANbus, except they use the Device IDs stored in the nodes of their MTD Arrays to address other ECUs.

DAVA offers two ways of ending the Operational Phase and starting the Update phase with the choice up to the implementor. One way, the bus updates every time any ECU receives a message. Updating the bus every time an ECU receives a message provides maximum security and defense against Replay Attacks because all device IDs are one-time use. One-time use device IDs mean the attacker never has the chance to use a device ID to send his own data message over the bus. Another way, the bus updates after some user-defined number of times a specific ECU has received a message. The second option is slightly quicker at the expense of slightly less resilience against Replay Attacks, because the attacker could theoretically use a device ID to address a data message inbetween the user-defined updates. Either of the above options chosen, the Operational Phase ends.

### C. Update Phase

The Update Phase begins with the ECU who received the last message sending a message out to all ECUs on the bus telling them that its device ID is now invalid.

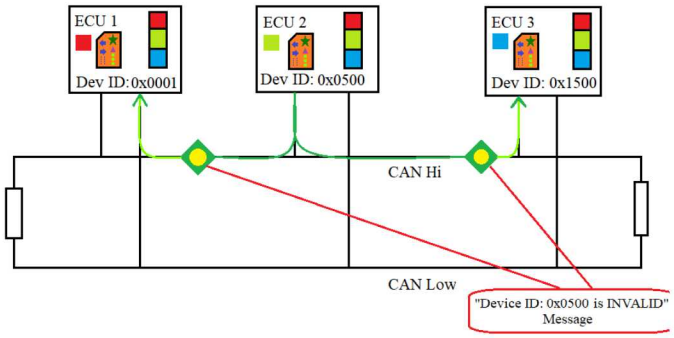


Fig. 7. Diagram of an example CANbus and invalidation broadcast messages

Every ECU on the bus, including itself, receives this message. When an ECU receives this message, it finds the position of the invalid ID in the MTD array. When the ECU finds the position, it considers the lower and upper bounds for the node ID, and randomly generates a new Device ID between those bounds. The randomization uses the seed provided in the initial input so that all ECUs create the same device ID. The nodes then overwrite the existing ID with the newly generated ID and the Update phase ends.

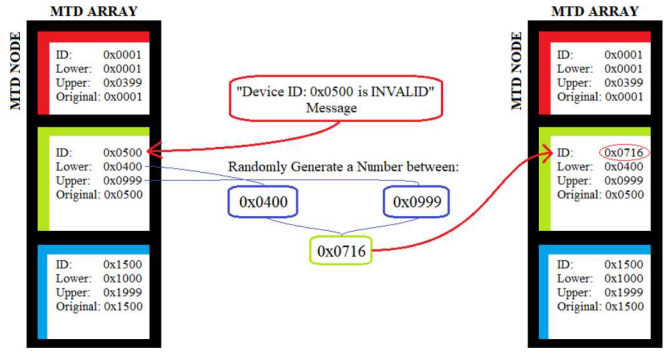


Fig. 8. The MTD Arrays of an ECU using the lower and upper bounds of an ECU to randomly generate a new device ID.

In the Update Phase, Device IDs update without an attacker being able to view it by reconnaissance of the bus 8). When the Update Phase ends, the Operational Phase starts again and continues until the next time the Update phase is triggered. This process of alternating between the Update Phase and the Operational Phase continues until the CANbus is no longer in use.

## IV. Simulator Design and Implementation

This paper uses a basic CANBus simulator written in C to test the effectiveness of DAVA. The simulator has basic message exchange functionality, as well as the ability to change the "Device IDs" of ECU objects. The simulator follows the CAN specification, enforcing message priority by device ID.

The simulator runs on a linux virtual machine running Ubuntu 18.04 with a single 2.6 Ghz processor. The simulator

walks through the CANbus operations via user input so the operator can view each message and update. The simulator prints to the screen the average time for an update, which is one to two microseconds. While this does not accurately represent the performance of a true CANbus environment, the simulator demonstrates the utility of DAVA.

The simulator supports an environment with ECUs with their device IDs starting at their lower bounds. For example, ECU one's lower bound and current device ID is one and ECU two's lower bound and current ID is five hundred. ECU one and ECU two know where they are via the position in their respective MTD Arrays. Messages are sent between the ECUs while they check their control update variable which is the user-defined number of messages an ECU can take before they send their invalid message which prompts updates. The update is performed just as described in the protocol and the bus continues on cycling from the update to the operational phase. Future work involves porting this protocol to accurate CANbus emulator and eventually, a hardware-based testbed.

## V. Evaluation

This paper considers several factors in evaluating DAVA such as the space and time complexity, a worst case, attacks it mitigates, and its place in the library of CANbus security algorithms.

### A. Space Complexity

The space complexity of DAVA is the necessity for all ECUs on the CANbus to be able to store the small input file internally, the dynamic MTD Array of nodes, and for each ECU to store the device IDs of all other ECUs. This storage space scales linearly to the number of ECUs. Currently, the space complexity is not a serious issue as the average car contains only around twenty ECUs, but some high-end cars have been designed with up to 100 ECUs or more [8]. ECUs store a series of number values like, ECU ids, the number of ECUs on the bus, seeds, etc. Thus, by counting the amount of bytes taken up by a file containing nothing but those values, the file size reveals the storage space taken up by the information needed for DAVA to execute. Through the above method, a car with twenty-two ECUs, each ECU holds approximately eight-hundred-seventeen bytes of information which includes the file, registers and the MTD Array. In a car with a hundred ECUs, approximately three-and-a-half Kilobytes of storage is used in every ECU.

In regards to the combined total of storage taken up by every ECU on the bus, the space scales exponentially by the number of ECUs. However, a car of twenty-two ECUs implementing DAVA needs roughly eighteen Kilobytes of additional space spread across all ECUs. In a car with 100 ECUs, approximately 350 Kilobytes of storage is used cumulatively across the entire bus.

### B. Time Complexity

Remember that DAVA prtocol allows Device IDs to change at a rate based on the user's needs. In the extreme case, Device

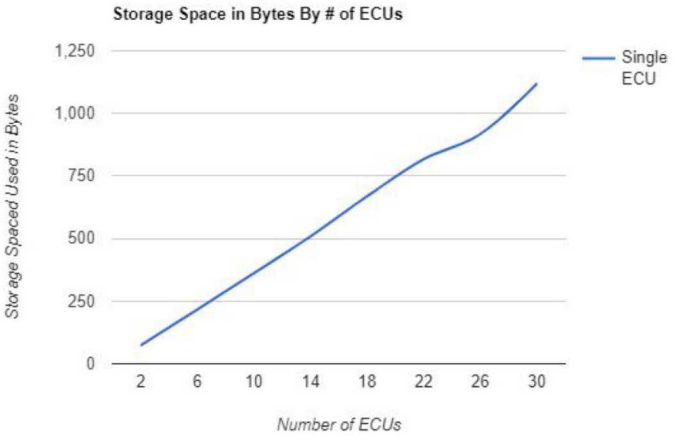


Fig. 9. The storage space in Bytes taken up by a Single ECU as the number of ECUs increase.

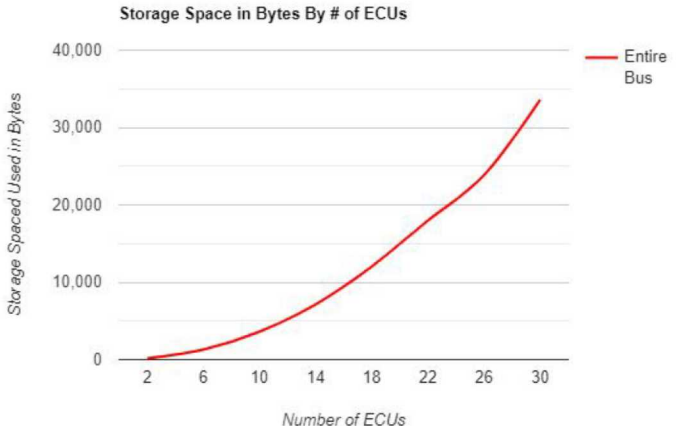


Fig. 10. The storage space in Bytes taken up by the entire bus as the number of ECUs increase.

IDs change after every message is sent. The other option supports address changes after every $N$ messages, where $N$ is selected by the use case. Therefore, the time complexity is more variable due to the user's decision of determining how many messages can be sent before each update. The most amount of computation time happens in the Initialization Phase where the MTD Array is being created and populated in. Thus, the initialization phase happens once, when the car is first started. Time Complexity throughout the protocol is the time it takes for all ECUs to update and randomly generate new device IDs and the time it takes to index into the MTD array and access device IDs during message composition. Time spent on the actual update does not scale by number of ECUs like it does with Space Complexity because updates all happen in parallel. The slowest device determines the Update Delay. Ultimately, DAVA's normal time complexity is O(n) which is also the time complexity of the initialization, operation, and update phases.

A corner case that can increase the worst case time complexity of the DAVA protocol is as follows - when a new device ID is randomly generated between its lower and upper bounds, it

is possible the ECU may randomly generate a device ID equal to its invalidated one. In this case, the ECU randomly generate another device ID which will double the update delay of the slowest device. The average update time of the simulator is 1.5 microseconds which would result in 3 microseconds lost due to the Worst Case.

DAVA's primary goal is defending against reconnaissance attacks. DAVA achieves this, in theory, by constant changing of the devices IDs leading to the attacker scanning the CANbus to see messages with almost always different device IDs. The attacker can't easily associate a device ID to a device and can't easily compose a message for a device because the device ID it wants to invoke is constantly changing. DAVA can be implemented to update after every ordinary message preventing an attacker from performing a replay attack as well as it essentially makes device IDs only usable once before being replaced. This protocol currently has no protection against a Denial of Service Attack, and is ineffective if the attacker accesses the memory stored in an ECU. DAVA puts the above security concerns aside in favor of focusing on being as lightweight as possible and mitigating the most common attacks the bus may face which this protocol, through initial testing, performs effectively.

In comparison to pre existing protocols, DAVA stands apart through the use of an identical data structure and syncrhonization of ECU operations. DAVA does not need to conceal messages over the bus as ECUs are seeded in such a way that their initialization and update phases always result in the same values. DAVA does not use encryption so that it can remain lightweight with its relatively small number of simple operations. These characteristics of DAVA set it apart as a protocol meant primarily to interest the Automotive Industry to adopt security measures for its CANbus and prevent avoidable security breaches in the fast approaching future.

**Security Properties**

| | **Update After Every Message** | **Update After "X" Messages** |
|---|---|---|
| **Origin Integrity** | YES | YES |
| **Confidentiality** | NO | NO |
| **Prevents Replay Attacks** | YES | NO |

**Protocol Properties**

| | **Update After Every Message** | **Update After "X" Messages** |
|---|---|---|
| **Standard Modifications** | NO | NO |
| **Time Complexity** | O(n) | O(n) |
| **Space Complexity** | $O(n^2)$ | $O(n^2)$ |

Fig. 11. DAVA's protocol and security properties. The effectiveness of DAVA differs depending on what the update condition is.

## VI. Conclusions

This paper suggests a MTD technique for preventing reconnaissance attacks on CANbus. DAVA allows CANbus ECUs to generate random IDs in standalone mode while preserving device priority necessary for correct operations. This paper shows that DAVA can make CANbus secure against reconnaissance attacks while imposing little overhead. Looking into the future, DAVA requires more testing on true-to-life emulators to derive the necessary metrics to analyze its real world effectiveness and cost.

## References


[1] "Can bus explained - a simple intro (2020)," https://www.csselectronics.com/screen/page/simple-intro-to-can-bus/language/en., (Accessed on 03/25/2020).

[2] S. C. HPL, "Introduction to the controller area network (can)," *Appl. Rep. SLOA101*, pp. 1–17, 2002.

[3] R. Currie, "Hacking the can bus: basic manipulation of a modern automobile through can bus reverse engineering," *SANS Institute*, 2017.

[4] H. Okhravi, W. W. Streilein, and K. S. Bauer, "Moving target techniques: Leveraging uncertainty for cyberdefense," MIT Lincoln Laboratory Lexington United States, Tech. Rep., 2015.

[5] O. Stan, Y. Elovici, A. Shabtai, G. Shugol, R. Tikochinski, and S. Kur, "Protecting military avionics platforms from attacks on mil-std-1553 communication bus," *arXiv preprint arXiv:1707.05032*, 2017.

[6] C. Lei, H.-Q. Zhang, J.-L. Tan, Y.-C. Zhang, and X.-H. Liu, "Moving target defense techniques: A survey," *Security and Communication Networks*, vol. 2018, 2018.

[7] S. Woo, D. Moon, T.-Y. Youn, Y. Lee, and Y. Kim, "Can id shuffling technique (cist): Moving target defense strategy for protecting in-vehicle can," *IEEE Access*, vol. 7, pp. 15 521–15 536, 2019.

[8] "Automotive Electronic Control Unit Market Size | Industry Report, 2025," Mar 2020, [Online; accessed 24. Mar. 2020]. [Online]. Available: https://www.grandviewresearch.com/industry-analysis/automotive-ecu-market