frontiers in Future Transportation

REVIEW
published: 07 February 2022
doi: 10.3389/ffutr.2021.792573

A Tutorial on Moving Target Defense Approaches Within Automotive Cyber-Physical Systems

Bradley Potteiger[1]*, *Zhenkai Zhang*[2], *Long Cheng*[2] *and Xenofon Koutsoukos*[3]

[1]*Johns Hopkins Applied Physics Laboratory, Laurel, MD, United States,* [2]*Clemson University, Clemson, SC, United States,* [3]*Vanderbilt University, Nashville, TN, United States*

Moving Target Defenses (MTD) have become a popular and emerging defense strategy for the protection of traditional information technology systems. By their very nature, MTD strategies are designed to protect against adversary reconnaissance efforts on static platforms, essentially sitting back and having unlimited time to identify, craft, execute, and scale an exploit. With the rapid adoption of distributed automotive Cyber-Physical Systems (CPS) ranging from self driving cars, to connected transportation infrastructure, it is becoming more apparent that third party supply chains, increased remote communication interfaces, and legacy software stacks are making the traditionally designed standalone systems become more susceptible to safety-critical cyber-attacks. MTD strategies within the automotive CPS domain have to delicately balance the tradeoff between security and real time predictability, maintaining the safety constraints of the systems. In this paper, we explore the various MTD strategies presented within the literature while discussing potential applicability and strategies sufficient for the automotive CPS domain.

Keywords: moving target defense, cyber-physical systems, automotive, cybersecurity, resilience

OPEN ACCESS

Edited by:
Lipika Deka,
De Montfort University,
United Kingdom

Reviewed by:
Ranwa Al Mallah,
Ryerson University, Canada
Manuel Urueña,
Universidad Internacional De La Rioja,
Spain

****Correspondence:***
Bradley Potteiger
brad.potteiger@jhuapl.edu

Specialty section:
This article was submitted to Connected Mobility and Automation, a section of the journal Frontiers in Future Transportation

Received: *10 October 2021*
Accepted: *31 December 2021*
Published: *07 February 2022*

Citation:
Potteiger B, Zhang Z, Cheng L and Koutsoukos X (2022) A Tutorial on Moving Target Defense Approaches Within Automotive Cyber-Physical Systems.
Front. Future Transp. 2:792573.
doi: 10.3389/ffutr.2021.792573

INTRODUCTION

Over the past decade, increasing numbers of electronic control units (ECUs) that communicate via different types of communication buses like controller area network (CAN), FlexRay, and automotive Ethernet have been assembled inside automobiles to provide intelligent services and safety to users (Shane et al., 2015; Wu et al., 2020). The automotive industry is shifting towards autonomous and connected vehicles. With increasing automotive intelligence, connectivity and complexity, security and privacy have become pressing concerns (Seshia et al., 2017). Researchers have shown that various vehicle systems can be hacked to enable a remote takeover of the vehicle (Stephen et al., 2011; Petit and Shladover, 2015). Miller et al. used a Wi-Fi open port to hack the multimedia system of a Jeep Cherokee and reprogram the firmware of ECUs. They were able to control a wide range of automotive functions (e.g., cutting brakes, shutting down engines, and driving off roads), triggering a recall of 1.4 million hackable vehicles. A typical automotive CPS heavily relies on various sensors to make navigation and other control decisions, which are vulnerable to sensor faults and attacks (Loukas et al., 2019). Physical faults and malicious attacks can result in an incorrect perception of the environment, which can in turn lead to task failure or even accidents. Misbehaving cars can further affect other connected cars and networks in the connected environment. Therefore, securing automotive CPS against malicious attacks is important in the prevention of potential damages to vehicles and transportation systems.

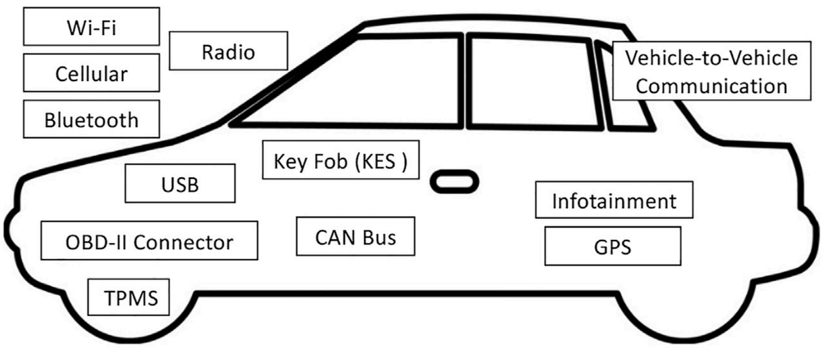


FIGURE 1 | Attack surface of an automotive CPS.

Memory corruption vulnerabilities are one of the most common attack vectors used to compromise software systems. Automotive software is not immune to such vulnerabilities. Stephen et al. (2011) identified multiple memory corruption vulnerabilities in a car's telematics unit and media player. Security researchers at Cisco revealed a memory corruption vulnerability in GNU libc for ARMv7 (which can be used for autonomous applications in smart cars) that enables attackers to perform remote code execution and gain remote control over smart cars. Tradiontial defense mechanisms, such as intrusion/anomaly detection systems, protect automotive systems in a passive manner. However, defenses are often way behind advanced attacks in taking appropriate actions to thwart potential attackers (Cho et al., 2020). Moving target defense (MTD) plays a important role in improving the security of automotive CPS in a proactive fashion.

MTD adds uncertainty and complexity into the system to make it difficult for attackers to identify vulnerable components and exploit vulnerabilities. In this paper, we conduct a review of the existing MTD mechanisms which are applicable for protecting automotive CPS from memory corruption attacks. Our contributions are as follows.

- We present a comprehensive overview of the automotive CPS domain, including the architectures, security challenges, and potential opportunities.
- We discuss several classes of MTD strategies that can be complementary to existing automotive CPS security techniques.
- We discuss key design decisions and tradeoffs that will make MTD strategies effective for key real time constraints, performance metrics, and safety assurances.

THREATS TO AUTOMOTIVE APPLICATIONS

With the increasing use of CPS in modern society, the properties of safety and security are becoming more interwined, when in the past they have been traditionally treated as two independent issues dealt with by two separate communities (Wolf and Serpanos, 2018). The tightly coupled nature of CPS between the cyber and physical domains means that it is not only enough to lock down the internal data of the system, but it is equally as important to ensure proper physical dynamics of the system. As such, measuring security in CPS is heavily focused on the continuous physics dynamics, compared to the traditional discrete nature of information security. Therefore, evaluation is often conducted with complex, model based simulation environments (KoutsouKos et al., 2018).

In traditional information technology systems one of the primary goals of an adversary is usually to obtain a piece of information. This has been displayed in three recent cyber attacks including a 2016 hack on the democratic national committee (Inkster, 2016), a 2016 electronic records compromise of a large hospital chain (Mukherjee, 2016), and a 2009 compromise of sensitive employee information within the Office of Personnel Management and Budget (OPM) (Rosenzweig, 2012). CPS systems on the other hand have unique aspects that make both attack vectors and impact different from information technology systems. CPS have the additional requirement of fully complying with a respective specification of properties. Any failure to meet these requirements will result in a failure of the system. As such, an adversary can successfully compromise a system by disturbing the timing of real time operations, or injecting false data to lead to wrong actuation decisions. The most popular example of a CPS cyber attack is STUXNET in which hackers were able to create a worm that gained access to and altered programmable logic controllers (PLCs) to overheat centrifuges in Iranian nuclear enrichment facilities (James, 2011). At this point, it became clear that to successfully protect CPS, a proper design needs to factor in system integrity from the security community, as well as system availability, and reliability from the safety community.

Compared to information technology systems, CPS systems often have a higher level of physical accessibility. Attackers can utilize physical attack vectors such as planting explosive devices and sabotaging railway tracks or positive train control systems (Riley, 2004; Ortiz et al., 2008; Sanchez, 2016) to disrupt the operability of CPS components. As such, it is just as critical to implement physical defense mechanisms in CPS, compared to cyber mitigations.

Another critical difference between CPS systems and information technology systems is that patching and frequent updates are not well suited for CPS systems due to the requirement of constant and safe operation (Cardenas et al., 2009). As such, devices are often left in the field for years without updating software. Due to this fact, CPS systems often have outdated code and security measures making them vulnerable to attack (Teso, 2013; Cerrudo, 2014; Ghena et al., 2014). Therefore, CPS security has to be designed to withstand adversary attack attempts over long periods of time without any intervention. Finally, CPS designs have to be built in a resilient fashion, under the presumption that whenever an exploit is developed defeating previously established defense mechanisms, the system will still be able to operate sufficiently, and safely.

CPS have unique features and requirements that present challenges to security professionals to adapt from traditional information technology practices. The ability of CPS to interact with the environment through actuation allows for attack impact to translate from data loss to physical damage potentially resulting in fatalities. Additionally, the ability to receive information through sensors allows attackers to adjust control operation through the manipulation of sensor data. CPS provides the potential for attackers to maximize damage to their target.

Automotive CPS Components

In contrast to the analog structure of early car models, modern vehicles consist of complex systems of systems. Cars today are made up of hundreds of electronic digital components communicating through a mesh of interconnected networks with varying degrees of speeds, and protocols. This is backed up by Charette (2009) which states that a modern car runs approximately 100 million lines of code on 50 to 70 electronic control units (ECUs). This makes a modern car essentially a “computer on wheels”, presenting a vulnerability to traditional hacking methods once thought of as only applicable to computers and information technology systems.

Automotive CPS threat actors include script kiddies, political activists, insiders, white hat hackers, black hat hackers, and cyber terrorists (Meyers et al., 2009). The biggest threat comes from both cyber terrorists, and black hat hackers as these adversaries often have the most malicious intentions in causing bodily harm, and damage. Additionally, these groups of adversaries are often state sponsored so they are very sophisticated and resourceful in their strategies. Researchers have to protect against this adversary group first as it is no longer hard to picture an attempted assassination attempt or terrorist attack through hijacking a car. However, with increased academic research comes more open source tools, and documentation. Therefore, lower skill adversaries such as hobbyists, and coders are gaining the potential to experiment and develop software for vehicle infrastructure. As such, these groups also have to be taken into account in the future when developing cybersecurity design for vehicle infrastructure.

Common vehicle systems include the electronic control units (ECUs), telematics control units (TCU), keyless entry system (KES) system, radio system, and airbag control units (Stephen et al., 2011). The architecture of a car is grouped into three categories, all of which contain their own vulnerabilities. These categories include:

- Electronic Control Units—The embedded computer devices that control the various internal functions of a car.
- External Interfaces—The method of external entities interacting with the car.
- Internal Network—The network that the internal electronic control units communicate on

A high level threat model of these interfaces is illustrated in **Figure 1**.

An ECU is an “embedded system that controls one or more electrical systems or subsystems in a vehicle”.

These devices serve as the brains behind the modern car, taking in data from onboard sensors, performing calculations, and distributing instructions to the various in vehicle electronic systems to maintain proper and efficient driving and operational performance. ECUs govern practically every aspect of vehicle functions from small tasks such as activating brake lights or opening windows to critical functions such as autonomous brake systems. Each ECU typically works independently operating its own firmware, but complex tasks may require cooperation among multiple ECUs (Zhang and Delgrossi, 2012).

External interfaces are means from which outside actors can communicate to the vehicle. From **Figure 1**, the most common external interfaces to the car include cellular, wireless (wifi), bluetooth, tire pressure monitoring system (TPMS), and keyless entry system (KES). Cellular communication channels are utilized for telematics system operations. Telematics services are widely utilized for remote vehicle monitoring especially in the case of routing emergency response, or predicting future maintenance needs. The most common services are OnStar (2017) and Lojack (2017). OnStar is a remote monitoring company that provides customers a line of assistance in the case of emergency. To accomplish this task, OnStar requries several sensor values such as speed, collision detection, and GPS coordinates among others. Therefore, in the case of an accident, the OnStar monitoring center can automatically detect an emergency and alert the authorities of your exact location to respond to Lojack is a device that tracks the GPS coordinates of your car in the case that it becomes stolen. Furthermore, the device utilizes a cellular network to communicate its precise location in real time to the police. Many newer model cars have wireless interfaces that allow mobile phones, tablets, and computers to control specific features. These features range from controlling windows, and door locking, to controlling the infotainment center, to seat control. This wireless feature further serves as a wifi hotspot, for each passenger to have the capability to utilize the internet from inside the car. Bluetooth is most commonly utilized in cars as a short range communication protocol for mobile phones to connect to the infotainment center. This connection is used for transferring audio such as music or phone calls between the vehicle interface and phone. The tire pressure monitoring system uses four sensors on each respective tire to record the real time tire pressure while indicating a warning signal in the case that one of the readings is below a threshold. Each external tire pressrue sensor communicates to a central vehicle

ECU through a short range Bluetooth communication protocol. Finally, a vehicle key fob is actually an active RFID device that is validated from the car through a short range RF communication channel.

The final category is the internal interface. These are devices that connect directly into the internal vehicle network. Examples include the infotainment console, USB devices, and the OBD-II connectors. The attack surface of these devices requires physical access to the inside of the car. However, this category of devices serves as the most critical threat to the security of the vehicle, as once these devices are compromised, there are very limited precautions that prevent them from controlling critical ECUs of the car. Newer model vehicles consist of USB ports designed for charging mobile phones, and media players, while serving as a direct interface to the infotainment media player. Due to this direct connection, malware injected on the USB drive can infect the infotainment center and lead to a compromised state of the network. The next critical internal interface is the infotainment center. The infotainment center serves as the central entertainment hub of the car. These devices include a radio interface, phone call interface, music player interface, GPS interface, and other applications. These applications communicate with ECUs through the internal network as well as external entities through short and long communication protocols. Therefore, an obvious path to compromising these devices is through the external communication interfaces. However, it is important to note that third party applications can also be installed on these devices. As such, a new emerging threat is infecting the infotainment center through the use of third party applications. The infotainment center is the most vulnerable remote attack avenue for the vehicle due to the vast connections it has with the external environment. The final internal interface to mention is the OBD-II port. The OBD-II port is an input port located under the steering wheel key slot that serves as a direct connection for maintenance personnel to interact with the internal vehicle network. Usually, when a car is serviced from a dealership, personnel connect to the OBD-II port to gather diagnostic information such as emissions, engine temperature, average speed, and other details. This direct connection allows for the injection of packets directly onto the internal network, versus through intermediary paths such as with the infotainment and USB interfaces. Therefore, by compromising a maintenance person, or obtaining physical access to the car, adversaries can directly inject malware onto the internal network and consequently infect the connected ECUs.

The internal automotive networks consist of a series of multiple communication buses with varying protocols. The most common communication buses utilized are the Controller Area Network (CAN) Bus, the Local Interconnect Network (LIN) Bus, the FlexRay Bus, and the Media Oriented System Transport (MOST) Bus (Studnia et al., 2013). These buses are described below:

- CAN Bus—The most common communication bus. A serial bus designed from two twisted wires transmitting a high and low voltage respectfully. Data rate peaks at 1 Mb/s. CAN is responsible for many systems such as vehicle control, safety, and electrical systems. It operates like a broadcast network in which packets are conveyed to all nodes on the network and it is the responsibility of the nodes to determine whether or not to process them. Each CAN packet has an identification (ID) field to help determine which nodes will process it.
- LIN Bus—A single wire subnetwork for lower bandwidth, low cost, and low end multiplexed communication in automotive networks. LIN uses a master-slave model, where a master node and up to 16 slave nodes share a bus. The master sends out messages to all slaves, and the slave can only send a message if requested by the master. The communication rate can peak at 20 kb/s. Since this protocol is used for low cost, low criticality elements, it is used for controlling comfort elements such as electronic window lifts or windshield wipers.
- FLEX Ray Bus—The successor to the CAN protocol offering speeds up to 10 Mb/s. The high speed and reliability enable the use of X-by-Wire technologies such as the electronic control of currently mechanical control systems such as steering and braking. This bus is widely used for vehicle safety systems, but due to its high cost, is limited in its widespread use in the internal network.
- MOST Bus—A synchronous network used to transmit multimedia data through the car *via* optical fiber. Multiple data channels are offered as well as a control channel. Synchronous communications are used to transfer streaming data such as audio or video signals, while asynchronous communications are used for scenarios such as retrieving data from the internet. Speeds can reach up to 24 Mb/s. The MOST bus is used for the infotainment center and entertainment applications in the vehicle.

In a 2014 Audi A8 vehicle most of the networks consist of varying CAN buses including the drivetrain, distance control, gateway, and convenience systems (Miller and Valasek, 2014). Additionally, some convenience systems such as the keyless entry, power windows, and windshield wipers use the LIN Bus due to the lower bandwidth requirements. Finally, the MOST bus connects the infotainment system of the car to the rest of the network.

There are several vulnerabilities with the CAN Bus packet structure and protocol dealing with confidentiality, integrity, availability, authenticity, and non-repudiation (Studnia et al., 2013). From a confidentiality standpoint, every message sent on the CAN Bus is broadcast to every other node connected to the bus. Due to this behavior, if an attacker were to compromise one connected ECU device, they could eavesdrop on the rest of the communications on the CAN bus, and also spoof messages to any connected ECU. Therefore, this type of communication is truly only as strong as the weakest link as once one node is compromised, the whole network is subject to being compromised. In regards to vulnerabilities with integrity, the cyclic redundancy check (CRC) in the CAN packet exists to check if a message has been modified. However, it is easy to forge a correct CRC so this is not sufficient for preventing an attacker from creating false messages or modifying existing messages. In regards to availability, since every packet has a user defined priority bit, an attacker can spoof numerous high priority packets to flood the CAN bus and prevent proper communication between the ECUs. In regards to authenticity, the packet structure of CAN does not include a field for authenticating the sender of the message. This makes it very easy to spoof messages on the network. Finally, from a non-repudiation standpoint there is no method in the protocol for a node to prove that it has not sent or received a given message.

By exploiting these vulnerabilities, several types of attacks can be executed on the network. Paul et al. (2015) describes five types of potential attacks including data stealing, control overriding, vehicle degradation, data falsification, and external sensor attacks. Data stealing deals with an attacker connecting to the CAN network, and eavesdropping on the rest of the ECU communications. Control overriding deals with an attacker taking advantage of the susceptibility of CAN networks for denial of service attacks by injecting high priority messages on the bus which would always be executed over the normal control messages. As such, an attacker could use this to take control and hijack vehicle operations. Vehicle degradation deals with an attacker spoofing messages about the current condition of the vehicle to trick the system into running inefficiently or becoming damaged by overheating, running out of gas, or having a flat tire. Data falsification deals with relaying false information to the driver to directly or indirectly cause unsafe behaviors such as disabling an airbag warning light when the airbag system is dysfunctional.

Putting these concepts into practice, researchers in (Miller and Valasek, 2013) have shown the ability to effect the behavior of vehicles through attacks on the CAN bus. The research was built on the assumption that adversaries could gain access to the CAN bus through external means, but once access is gained atacks could be executed to turn the horn on, shut the engine off, shutting off the brakes, disabling steering, spoofing the speed on the speedometer, and increasing the amount of distance shown on the odometer. Additionally, the same researchers were able to reverse engineer and exploit a modern Jeep Cherokee in (Miller and Valasek, 2015). Even though this research was conducted in the purely academic context, it is not hard to imagine the applications that a malicious adversary could use these concepts for. As such, it is critically important to increase the amount of security mechanisms in these systems, and bring cybersecurity strategies into the earliest stages of vehicle design.

Previous vehicle security efforts have focused on hardening internal ECU communication, implementing anomaly and intrusion detection systems, and securing external communication interfaces. For securing the internal vehicle communication, a majority of research is focused on the CAN bus protocol due to being the most widely utilized protocol in vehicles (Lin and Sangiovanni-Vincentelli, 2012). The CAN protocol at the basic level is vulnerable to masquerade, and replay attacks due to the lack of authentication. There has been progress in authentication, such as a Message Authentication Code (MAC) technique in Tesla vehicle models (Adrian et al., 2000), as well as a time triggered implementation proposed by Szilagy and Koopman (2008) for establishing global time. However, the biggest challenge is developing authentication mechanisms with low enough overhead to be feasible in deployment environments due to low bandwidth of the CAN protocol. The second area of security research focuses on securing the underlying processes and computations operating within the vehicle. Wolf et al. (2007) proposed to introduce symmetric encryption techniques to secure the integrity of ECU computational processes, while hashing techniques can be utilized to secure data. Additionally, secure software engineering practices have gained more attention, to minimize the amount of attack vectors available in ECU's based on bad coding practices (Paul et al., 2004). Furthermore, to address attack detection and reaction, research has applied traditional information security techniques such as honeypots, and intrusion detection systems to the vehicle domain (Kleberger et al., 2011). By leveraging artificial intelligence, and machine learning, these techniques can be optimized to minimize the amount of time, and accuracy of detecting an adversary event. The final area of research focuses on securing the internal vehicle network from external devices (Han et al., 2014). By monitoring third party apps, connected phones and media players, approaches have focused on implementing firewall techniques within the vehicle gateway interface. As such, by designing only one path of entry through the gateway components, these techniques have been proven effective to block malicious entities from gaining access to the internal, more safety-critical vehicle components. When analyzing the related research in vehicle security, a lot of the approaches focus on locking down the system from external threats. For example, authentication ensures that attackers can't spoof internal communications, while firewalls serve as an outer layer of protection, blocking malicious entities from gaining access into the internal network. Additionally, to appropriately optimize these types of defense mechanisms, the designer must have full knowledge of the potential vulnerabilities, as well as every location an adversary can potentially infiltrate. However, it has been commonly shown that researchers and adversaries can find ways around defense mechanisms in place, whether due to lack of sufficient protections due to designer ignorance, or by introducing zero day exploits not previously known in public (Miller and Valasek, 2015). As such, it is not feasible to only rely on locking down systems based on previously known vulnerabilities. A defense in depth approach is the best option for optimizing the security of a system, due to implementing multiple layers of protections. MTD along with control reconfiguration enables a backup defense protection mechanism that can stop attacks such as code injection, code reuse, and data tampering, in the event that an adversary has defeated outer defense protections. Additionally, by introducing control reconfiguration capabilities, resilience can be designed into systems to maintain availability during attack sequences, when in regular defense mechanism implementations, an attack would either result in system hijacking or crashing.

Buffer Overflow Vulnerabilities

One of the most popular vulnerabilities within vehicle software are buffer overflows, which have been regarded as the most commonly used exploit over the last several years (Cowan et al., 2000). Buffer overflow attacks are widely publicized in the traditional information technology domain for the purpose of network penetration and pivoting. However, these vulnerabilities become more interesting in the CPS domain, where they are commonly utilized to exploit memory allocation functions in low

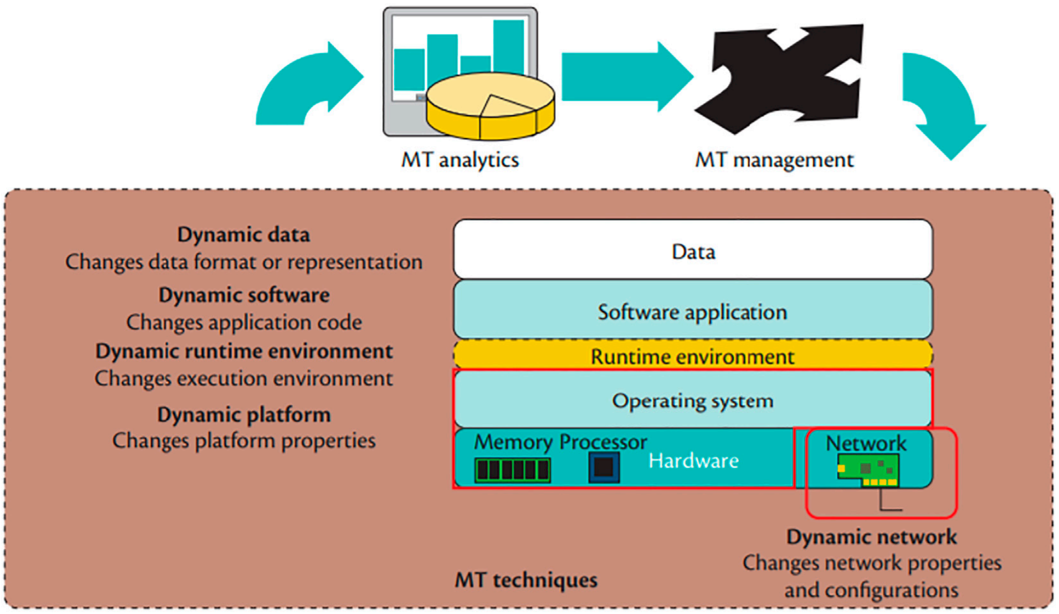


FIGURE 2 | Moving target defense categories (Okhravi et al., 2014).

level programming languages such as C and C++. In addition to providing the capability for gaining remote access and exfiltrating stored data, attackers can also alter the control flow through non bounded input to execute malicious code on the embedded controllers. With previous reconnaissance on the control API, attackers can reverse engineer controller instructions to spoof new instructions to alter the physical behavior of the embedded device. One possible result of a buffer overflow vulnerability is a code injection attack. As such, the attacker can divert the program control flow to inputted malicious code, which will consequently be allowed to run unchecked on the system. The buffer overflow vulnerability and code injection attacks are described below.

For the first type of attack, the user can set the return address of a locally called function to be another location in the program code. This is referred to as a code reuse attack. In code reuse attacks, attackers edit the program control flow to return to a sequence of already existing code in the program. Code reuse attacks are useful in cases where the stack is non-executable, preventing an attacker from running instructions from input on the stack. Code reuse attacks have been successful for privilege escalation in the Android operating system, creating rootkits, and injecting code into Harvard architectures (Habibi et al., 2015).

For the second option, a code injection attack can be used. In this case, an attacker can push arbitrary code onto the stack through a buffer overflow. Then the attacker can subsequently overwrite the stack return address to redirect control back to the injected attacker code (Habibi et al., 2015). Code injection attacks are used in cases where the stack has executable permissions. For code injection attacks, the process first involves inputting the respective code into the target process. This part of the input is called the payload, which is the code that will be executed. The end of the input string will be crafted to overwrite the function return address to be the beginning of the payload section of the input. As such, after the input is inserted into the program, control flow will return from the current function to the beginning of the attackers payload code. After this point, the payload code will be executed. Normally, due to restrictions on the size of an input string and consequently the size of the payload, the injected code needs to be small in size. It is common for attackers to inject the system function binary code as a payload to be executed, consequently opening a command shell for further commands to be executed on the system. The goal of using the code reuse and code injection attacks is not to insert all of the desired instructions into the exploit, but to use these avenues for establishing a foothold into the system for further command execution.

APPROACHES

With attackers becoming increasingly sophisticated, cybersecurity is becoming less focused on completely locking down systems and more focused on decreasing the risk of attacks to the system. If attackers have the motivation, time, knowledge, and resources to attack a system, eventually they will be successful in gaining entry. As such, attack risk is lowered by decreasing attacker motivation and the knowledge of the system. MTD is a good tool for decreasing attacker knowledge by constantly changing various system properties while preserving essential semantics, executing a security through diversity strategy (Evans et al., 2011).

Therefore, even if an attacker were to obtain knowledge of system vulnerabilities at one point in time, these vulnerabilities wouldn't necessarily be true during future time periods. MTD has long been applied to the traditional information technology domain, specifically with examples such as software defined networking, but however are fairly new to the CPS domain.

MTD can be characterized into five sections: dynamic runtime environment, dynamic software, dynamic data, dynamic platform, and dynamic networks (Okhravi et al., 2013). These categories can be characterized in regards to their place in the execution stack and are described below as well as in **Figure 2** and **3**.

Dynamic runtime environment techniques focus on changing the environment present by the operating system during execution. These techniques aim on preventing attackers from exploiting software vulnerabilities to compromise a system. Two techniques associated with this category consist of instruction set randomization (ISR), and address space randomization (ASR). Instruction set randomization involves changing the format of the actual program instruction opcodes dynamically during execution. This prevents the attacker from predicting and injecting code payloads into the system. The second technique, address space randomization, involves changing the virtual memory layout of the program. This prevents the attacker from rerouting the program to run code at certain locations in memory, and to assume adjacent variables to overwrite in the event of a buffer overflow attack.

Dynamic software involves changing the application code such that the attacker can no longer guess the internal behavior of the program based on fuzzing inputs. The most common technique for this category is diversification. Diversification involves rewriting program code in different formats such as with different instruction sequences, reordering functions and instructions, and reordering the internal data structures in such a way to accomplish the same functionality with a different process. The most common use of these techniques is to prevent side channel and fuzzing attacks on the program.

Dynamic data involves changing the representation of application data such that unauthorized use is hindered but the program semantic use remains the same. The most common technique in this category is data encryption and obfuscation. Data encryption and obfuscation seeks to guarantee that even if attackers were to access data from a program, they would not have the means to figure out the true context of that data.

Dynamic platform involves changing the properties of the computing platform in an effort to disrupt attacks that rely on specific platform characteristics. Some platform characteristics include operating system, processor, and communication means. It is most common for this technique to involve migrating applications between different machines to prevent attackers from targeting a specific platform or vulnerability of the system.

Dynamic networks focuses on continuously modifying network properties to lower the probability of success for network born attacks. This category often involves changing IP addresses and network ports through software defined networking means, but also includes changing communication protocols, and changing the topology of the network.

Each of the MTD techniques described above is designed to disrupt a specific phase of the attack sequence. To understand this further, the stages of the attack sequence are described below:

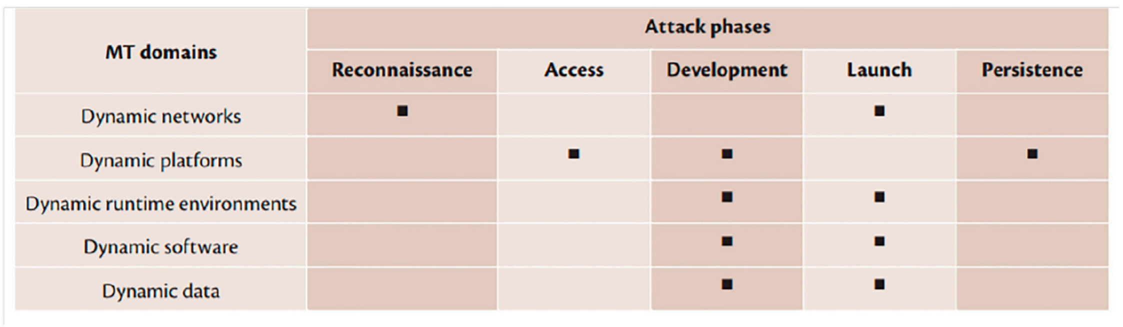
- Reconnaissance—Attackers find and collect basic information on their target. Examples include spear phishing or determing a target host's IP address using network scanning.
- Access—Attackers take actions to collect detailed information on their target. Examples include probing against a server to determine its architecture, operating system, and configuration. This stage detects the vulnerabilities that attackers can leverage to access the target.
- Development—Attackers research and develop an attack that can exploit a vulnerability on the target.
- Launch—Attackers deliver the attack payload to compromise the target. The payload can be delivered through a variety of means including the network, infected media, or malicious executable code.
- Persistence—Attackers insert a backdoor on the target to ensure future access.

For this paper, the dynamic runtime environment category will be looked at extensively focusing on applications of instruction set randomization, address space randomization, and data space randomization in CPS. Several attacks that MTD address include data leakage attacks, resource attacks, injection attacks such as code injection and control injection, spoofing attacks, exploitation of authentication, exploitation of privilege and trust, and supply chain or physical security attacks (Okhravi et al., 2013). In regards to instruction set randomization, address space randomization, and data space randomization, this paper looks specifically at preventing code injection, code reuse and various Weird Machine based attacks.

Instruction Set Randomization

Many techniques have been proposed to defuse code injection attacks including stackguards, memory management unit access control lists, control flow integrity, and masking code pointers (Alnabulsi et al., 2015). Additionally, a syntactic checker has been proposed for SQL based applications (Ray and Ligatti, 2012). However, instruction set randomization is the most widely accepted technique for preventing these types of attacks. Normally, code injection attacks utilize buffer overflow vulnerabilities for input processing in low level languages such as C, and C++ to insert malicious code into the program and divert control to execute that code. However, it has been demonstrated that other scripting languages such as Web CGI, bash, and SQL have been vulnerable to attacks (Boyd et al., 2010). For code injection attacks to be successful, the adversary has to rely on knowing the native architecture of the running program code on the target machine. However, ISR prevents this knowledge by changing the code architecture to a custom, randomized version that is not publicly known. For the code injection attack to now be successful, the attacker now needs to know the randomization key.

CPU instructions for common architectures such as ARM, and x86 have two parts, the opcode and the operand. The opcode defines the operation to be performed while the operands define the arguments to the function. It is important to remember that these instructions are at the binary level so the combination of opcode and operand will become a respective binary sequence. During ISR implementations, we can create new instruction binary sequences by using cryptographic algorithms to create



MT domains	Attack phases				
	Reconnaissance	Access	Development	Launch	Persistence
Dynamic networks	■			■	
Dynamic platforms		■	■		■
Dynamic runtime environments			■	■	
Dynamic software			■	■	
Dynamic data			■	■	

FIGURE 3 | MTD mitigation attack kill chain stages (Okhravi et al., 2014).

new mappings between functions and opcode and operand combinations. The most common algorithm used in the literature is using a XOR command to change the binary sequence with a randomization key. For example, for the attacker to now guess the randomization key for an instruction that is 32 bits, it will take up to 2^{32} attempts respectively with a brute force approach. Attempts have been made in the literature to guess the ISR randomization key using various techniques such as side channel attacks, plain text key communication, and exhaustive key guessing (An et al., 2005; Weiss and Barrantes, 2006). However, other more advanced encryption algorithms such as the advanced encryption standard (AES) can be used as an alternative.

ISR implementations can either be hardware or software based. For hardware based implementations, a customized processor or FPGA needs to be used to insert derandomization functionality into the processor pipeline. Researchers have successfully used the OpenSPARC FPGA processor to create a hardware based ISR prototype (Papadogiannakis et al., 2013; Sinha et al., 2014). For early stage implementations of software based ISR, emulators such as the Bochs x86 emulator were used for the purposes of processing customized instruction architectures (Kc et al., 2003; Portokalidis and Keromytis, 2010). However, for more recent versions of ISR software implementations, there is a push for the use of dynamic binary translators which can be used to create a virtual layer between the execution program and processor pipeline. Several software implementations exist based off of DBTs including PIN (Luk et al., 2005), and STRATA (Scott and Davidson, 2001) for x86 processors along with MAMBO (Gorgovan et al., 2016) for ARM based processors. With the recent boom of ARM based embedded devices in the automotive embedded applications, MAMBO will become more relevant in the future. **Figure 4** illustrates an examplar architecture of an Instruction Set Randomization Framework utilizing the Mambo dynamic binary translation tool.

Address Space Randomization

An overwhelming amount of security advisories from US Cert have described memory corruption attacks as the top major risk, enabling attackers to execute remote code on critical systems (Li et al., 2006). In a majority of these cases, especially for executing code reuse attacks, the attacker needs to have knowledge of a specific address location for the target code to be run. Address space randomization (ASR) attempts to leverage this necessary piece of information by introducing artificial diversity into various segments of a program and system. This can be implemented in multiple degrees of granularity ranging from randomizing the starting base addresses of shared libraries and program segments to fine tuned randomization of individual lines of code in a program (Snow et al., 2013). By changing the location of various program segments, external memory access becomes unpredictable, and the attack will result in an invalid memory address being accessed. ASR can be implemented by randomizing a subset or all of a program's parameters such as the base address of the stack, base address of the heap, order of static variables addresses of function call targets, base addresses of shared libraries, and the order of functions in a shared library (Wang et al., 2011).

There have been developed ASR implementations in the literature on Linux (Bhatkar et al., 2003), Windows (Li et al., 2006), Macintosh (Bojinov et al., 2011), and mobile platforms (Bojinov et al., 2011). On the Windows ASR implementation, it is noted that there are a couple of limiting factors to randomization such as insufficient ranges of randomization, incomplete memory randomization, and limited documentation about existing randomization algorithms. The Linux versions of ASR referred to as address space layout randomization (ASLR) seem to be further along with more open documentation, as well as more user access to randomization customization parameters in the kernel. However, it is noted that on 32 bit Linux systems there is only 16 bits of randomization and on 64 bit Linux systems there are 32 bits of randomization (Shacham et al., 2004). ASR is implemented in Macintosh computers starting at the OSX release. However, these versions seem somewhat limited with only providing the ability to randomize the base addresses of shared libraries. It was noted that in the process of implementing ASR on mobile platforms there have been discovered problems such as the default shared library prelinking of the Android operating system, and the read only access of the file system. However, researchers were able to create a workaround called retouching to randomize prelinking libraries without needing to modify the kernel.

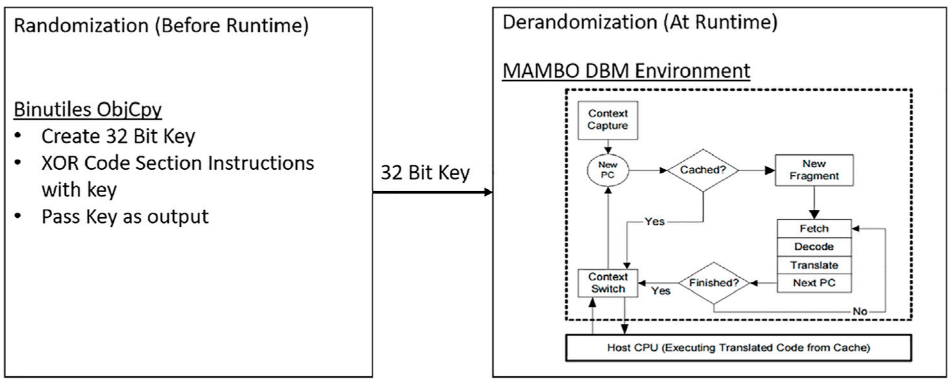


FIGURE 4 | Randomization architecture stack, base address of the heap, order of static variables addresses of function call targets, base addresses of shared libraries, and the order of functions in a shared library (Wang et al., 2011).

There have been several demonstrated attacks against ASR implementations. One of these attacks includes a timing attack that can exploit a hardware extension called the Intel Transactional Synchronization Extension (TSX) to break randomization in Windows, Linux, and MAC computers in under a second with near perfect accuracy (Jang et al., 2016). Additional attacks taking advantage of return oriented programming and relative jumping addresses to target code addresses (Wang et al., 2011). However, it has been noted that advances in position independent code compilation and self randomization algorithms have now mitigated some of these attacks.

Data Space Randomization

After obtaining access to a program through buffer overflow vulnerabilities, attackers can manipulate non control program variable data in an integrity attack to alter program behavior without altering control flow. One common technique utilized to alter these types of attacks is data space randomization (DSR). Data space randomization changes the internal or external representation of an application's data in such a way as to ensure that the semantic content is unmodified but unauthorized use, access, or modification is hindered (Okhravi et al., 2014). This is accomplished by randomizing the format, syntax, encoding, and other properties of the data. As such, DSR acts similarly to ISR in using a key based randomization and de-randomization process to encrypt variable data sensitive to attack. Each variable data object is randomized before it is written to memory and is derandomized after it is read from memory. DSR provides a much larger range of randomization than ASR, allowing for all 32 bits to be used on a 32 bit platform compared to the 16 bits used on Linux ASLR implementations. Consistent with the ISR process, the randomization process can be accomplished by using an XOR operation with a randomization key (Bhatkar and Sekar, 2008; Cadar et al., 2008). In these implementations the overhead is non-minimal with an average performance overhead of around 15% (Szekeres et al., 2014). Additionally, there is the possibility of using other symmetric encryption algorithms such as those in the AES family to add further security to the application, but subsequently increase the overhead percentage. DSR provides both the ability to use a common shared randomization key, but for enhanced security, each variable should be mapped to a unique randomization key. Implementations of DSR started with a software toolkit called PointGuard (Cowan et al., 2003). Pointguard randomized the stored pointer addresses to prevent attackers from gaining reconnaissance knowledge about pointer data. However, current DSR implementations now not only randomize pointer addresses but also the stored variable data. Some attacks against DSR listed in the literature include data leakage attacks, brute force and guessing attacks, and partial pointer overwrites (Bhatkar and Sekar, 2008). However, with strategic derandomization and high randomization entropy these types of attacks are deterred.

Network Randomization

For attackers to identify targets and vulnerabilities through reconnaissance, they often rely on known information about the connected network. The static nature of current networks makes reconnaissance easy, allowing for attackers to maintain privileged access for a long time once a vulnerability is discovered. This is especially significant as the internet task force has declared a number of attacks that can be implemented with an attacker correctly guessing a combination of transmission control protocol (TCP) attributes including the protocol, source address, destination address, source port, and destination port (Larsen and Gont, 2011). Furthermore, this leaves networks open to attacks from worms, especially hitlist worms who have preprogrammed lists of target IP addresses and entry ports to use for infection and spreading (Antonatos et al., 2007) However, the concept of network randomization seeks to continuously modify various network attributes such as addresses, ports, protocols, and logical network topology to deter the attacker from gaining relevant information necessary to conduct network borne attacks (Okhravi et al., 2014). Instead of focusing on hardening a system which is traditionally done in information security, this technique focuses on reducing the risk of attack by increasing the exploration space and changing the attack surface

(Zhuang et al., 2013). With the advancements in software defined technologies, these techniques are becoming easier to implement.

Network randomization implementations in the literature consist of a combination of randomizing network ports, IP addresses, and network paths. A software prototype implementation has been developed combining the capabilities of the above three randomization features (Chavez et al., 2016). To change the assigned IP address and port, the iptables utility of the netfilter kernel module is utilized. Additionally, software defined networking controllers can be used to change between various routes between network nodes, preventing attackers concrete knowledge of communication paths. By changing these network features, the attacker exploration space is increased, reducing the probability of a successful attack, and preventing an attacker from relying on previous gathered reconnaissance information.

DISCUSSION

As the adoption of MTD strategies is increasing within the CPS domain, especially with regards to the Network Security domain, it is important to present a clear description of the benefits. MTD strategies by their very nature are designed to insert dynamic and unpredictable properties into an otherwise static system, mitigating against the reconnaissance stage of the attacker kill chain. As an example, think of the requirements of return-oriented programming (ROP) attacks, or underlying Weird Machines. Its important to note that weird machines are defined as the occurrence of program sub-components (e.g., instructions, gadgets, functions) that can be leveraged to perform an operation other then for their intended purpose. These attacks generally require prior knowledge such as the system architecture, stack organization, control flow redirection target, and open network ports for a potential reverse TCP shell. Under a basic configuration, if a valid exploit was crafted for one deployed system instance, that exploit would scale to effect every other identical system, including systems with similar applications, network configurations, and defense protections. By adding MTD at various layers, each system becomes syntactically unique while remaining semantically the same, making the location of the various ROP gadgets different on every system. As such, even if an attacker can identify prior knowledge about a system, that knowledge becomes obsolete when trying to scale the attack.

Most of the application oriented MTD schemes within the literature have either relied upon a compilation based approach where function and address scrambling is performed statically at compile time, or a runtime based approach where dynamic runtime instrumentation frameworks are leveraged to adjust memory and code throughout a programs lifetime. Within the context of an automotive application, the latter approaches will be more infeasable due to the high amount of overhead that they present on the system. Furthermore, due to the added virtualized dynamic binary instrumentation layer, operation of the underlying program will become less predictable, potentially resulting in disruption to real time constraints and safety precautions. This makes the first category (compiled MTD implementation) more of an acceptable approach, allowing for more predictable operation during runtime. The two most common MTD compiler implementations were noted as Multicompiler (Larsen et al., 2013) and SelfRando (Conti et al., 2016) can play critical roles within the embedded software development pipeline. Multicompiler provides code transformation passes at compile time while SelfRando inserts stub code to perform dynamic randomization when the program is loaded into memory. Both tools implement one time randomization meaning that the code is not continuously randomized throughout the programs runtime lifecycle. However, it is important to note that Multicompiler only has a one to one mapping per compile iteration, while SelfRando will create a new randomized version on every program run attempt. In our opinion, Multicompiler presents an overall better package with the ability to insert other various LLVM passes such as shadow stacks, segmented memory, canaries, buffer overflow protections, and optimization passes. Additionally, no additional load time overhead will be presented, meaning that runtime executed program will be as close to the original intended program as possible.

In terms of network randomization schemes it is important to look at the real time context in a similar fashion as the application approaches. Traditionally, network randomization schemes have significantly relied upon specialized software defined networking (SDN) hardware, making the barrier to entry for generic consumer devices relatively high. Even though it is possible for automobile manufacturers to acquire such specialized hardware, it would add a degree of cost to the end product, making it not as likely to be implemented from a business perspective. However, recent research prototypes has focused more on software implementations of such randomization schemes leveraging endpoint clients integrated with operating system components such as IP Tables and routing drivers (Chavez et al., 2016). These methods provide a lower barrier to entry and are easier to integrate into existing automotive environments due to the ability to still leverage existing COTS infrastructure. From an automotive security standpoint, two different types of areas should be protected: 1) lower criticality TCP ethernet networks, and 2) safety-critical CAN bus networks. Newer automobile models incorporate TCP ethernet networks for lower criticality traffic between entertainment devices and applications. In this case, it is recommended to incorporate a degree of IP address randomization, and port randomization to obfuscate and diversify the device communication patterns, reducing the likelihood of external reconnaissance and injection attacks. From the safety-critical network perspective, CAN Bus networks between ECUs tend to have a one to many transmission model meaning all ECUs can see traffic from every other ECU. In this case, it is recommended to implement a CAN message integrity check, and message packet format randomization (Brown et al., 2020). Additionally, for an added layer of security, ECU identifier randomization can be implemented to reduce the likelihood of successful ECU spoofing attacks.

Finally, an up and coming attack vector within the industry is the side channel attack. From a processor standpoint, timing based attacks such as Spectre (Paul et al., 2019) and Meltdown (Lipp et al., 2018) have been shown to be able to reverse engineer "black box" memory through the use of control flow optimization knowledge within the processor. However, at a higher level of abstraction, the ability to leverage exhaustive vehicle request-response message to reverse engineer authentication keys (Kulandaivel et al., 2019; Kulandaivel et al., 2021). This is of greater importance to consider due to the proven ability to actually gain administrative access to safety-critical ECU commands on real world vehicles. To combat these types of attacks, control flow randomization for processor based timing attacks, as well as authentication key randomization for request-response attacks are low hanging mitigations to consider.

Benefits

Even though there are various benefits to leveraging dynamic approaches, a couple of challenges arise, as noted from the above section. The first challenge arises with the increased performance overhead on systems. A significant portion of MTD strategies depend on dynamic instrumentation to manipulate processes at runtime. As such, even though the performance overhead is getting to be more manageable, the average performance overhead for many of the recent approaches is still well above 30%, making integration into resource constrained systems often infeasible. The second challenge revolves around predictability of real time systems. With the introduction of dynamic instrumentation, preconfigured systems will lose their design time certifications, making it difficult to integrate into hard real time systems. To satisfy these challenges, there are often tradeoffs that can be made that can leverage performance efficient and low invasiveness techniques to provide a degree of protection while minimizing impact on the system.

However, it is also important to note that numerous benefits from such approaches. First of all is the reduction of "generic" reconnaissance and exploitation techniques. The most devastating attacks are often a form of a supply chain exploit, meaning that a vulnerability in one software library dependency can be scaled to effect every other deployment of that software application around the world. It has actually been shown that there is up to a 60% overlap between proprietary automotive firmware and open source router dependencies (Bradley et al., 2020), meaning that generic vulnerabilities can be leveraged to inflict more damage than originally thought. By diversifying code and runtime environments, original assumptions leveraged for exploit development will not be true for the randomized case. This means that randomized applications and operating systems will not be vulnerable to the same vulnerabilities and exploits as the rest of the effected vehicle models. In the same vein of diversity, network based randomization approaches often make reconnaissance efforts infeasible due to the continuous movement of IP addresses and ports. Reconnaissance knowledge of an open service or port will not necessarily be true during a period in the future. Finally, MTD approaches within the data domain can mitigate against data exfiltration even if an exploit can get through outer layer protections. New techniques such as homomorphic encryption (Sniatala et al., 2021) have made it possible to perform operations on encrypted data without the need to decrypt during any stage of the process. Furthermore, scattering data around the filesystem will create an incoherent environment from the attacker's standpoint, disrupting their ability to identify and extract key intelligence information.

Research Direction

To satisfy the above challenges, while maximizing the benefits of MTD strategies, we propose the following three research directions to prioritize over the next decade. The first research direction focuses on optimizing the performance overhead presented to systems while providing real time guarantees. This direction can be applied to both application/process based manipulations (ASR), as well as communication based techniques (CAN Bus message randomization). Specifically to the automotive domain, it is critical to maintain real time constraints and as such, formal methods further need to be applied to ensure that alterations to the existing systems will maintain semantic equivalence and satisfy the real time constraints of the system. The second critical research direction focuses on a higher level of abstraction, looking at dynamic configurations in multi-vehicle platoons. As such, research can focus on areas such as resilience of platoons, collaborative data fusion, and control reconfiguration to maximize protection against physical and cyber attacks. The final research direction focuses on investigating the role of MTD techniques within the AI domain. Various techniques such as ensemble learning have been becoming more popular, and inserting a degree of diversity into models can increase resilience and endurance against adversarial attacks. Diversifying sensor inputs and model sub-components also has the potential to increase the reliability and assurance of safety-critical AI systems such as self driving vehicles within the abundance of differing environments that they face.

CONCLUSION

MTD techniques have been established to be effective in the traditional information technology domain as a protection measure against ever emerging sophisticated cyber-attacks. Several techniques such as software defined networking, software defined radio, and ASR have proved effective against scalable reconaissance efforts by adversaries. Furthermore, there have been multiple recent surveys on MTD strategies in general to establish the most effective techniques within the data, application, and network domains. However, within the automotive CPS domain, several additional requirements are presented on systems, most notably the presence of real time constrains and the necessity for predictability versus high performance computing. As such, in this paper, we have explored various MTD strategies both from the general sense as well as other strategies specific to automotive architectures such as platoon reconfigurations. By strategically leveraging MTD strategies within automotive CPS environments, designers can optimally create a dynamic and unpredictable defense from the attacker standpoint, while maintaining all of the safety and security requirements for the vehicle to operate sufficiently.

AUTHOR CONTRIBUTIONS

The authors confirm contribution to the paper as follows: MTD idealization and substantial design input: XK and ZZ; Literary review: BP and LC; Content creation for Section 1 and Figure creation: LC; Content creation for Sections 2–5: BP; Draft manuscript preparation: BP and LC; and substantial editing and review: ZZ and XK. All authors reviewed the results and approved the final version of the manuscript.

FUNDING

This work is funded in part by the National Security Agency (H98230-18-D-0010), the National Science Foundation (CNS-1739328), and by the National Institute of Standards and Technology (70NANB17H266). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NSA, NSF, or NIST.

REFERENCES

Adrian, P., Ran, C., Tygar, J. D., and Song, D. (2000). “Efficient Authentication and Signing of Multicast Streams over Lossy Channels,” in Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000 (Berkeley, CA, USA: IEEE), 56–73.

Alnabulsi, H., Mamun, Q., Islam, R., and Morshed, U. (2015). “Chowdhury. Defence against Code Injection Attacks,” in Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST.

An, S., Evans, D., and Paul, N. (2005). “Where’s the FEEB? the Effectiveness of Instruction Set Randomization,” in 14th USENIX Security Symposium.

Antonatos, S., Akritidis, P., Markatos, E. P., and Anagnostakis, K. G. (2007). Defending against Hitlist Worms Using Network Address Space Randomization. *Computer Networks*.

Bhatkar, S., DuVarney, D. C., and Sekar, R. (2003). Address Obfuscation: An Efficient Approach to Combat a Broad Range of Memory Error Exploits. *USENIX Security Symp.* 12, 291–301.

Bhatkar, S., and Sekar, R. (2008). “Data Space Randomization,” in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).

Bojinov, H., Boneh, D., Cannings, R., and Malchev, I. (2011). “Address Space Randomization for mobile Devices,” in Proceedings of the fourth ACM conference on Wireless network security - WiSec ’11. doi:10.1145/1998412.1998434

Boyd, S. W., Kc, G. S., Locasto, M. E., Keromytis, A. D., and Prevelakis, V. (2010). “On the General Applicability of Instruction-Set Randomization,” in IEEE Transactions on Dependable and Secure Computing.

Bradley, P., Mills, J., Cohen, D., and Paul, V. (2020). “Ruckus: a Cybersecurity Engine for Performing Autonomous Cyber-Physical System Vulnerability Discovery at Scale,” in Proceedings of the 7th Symposium on Hot Topics in the Science of Security, 1–10.

Brown, R., Marti, A., Jenkins, C., and Shannigrahi, S. (2020). “Dynamic Address Validation Array (Dava) a Moving Target Defense Protocol for Can Bus,” in Proceedings of the 7th ACM Workshop on Moving Target Defense, 11–19.

Cadar, C., Akritidis, P., Costa, M., Martin, J-P., and Castro, M. (2008). *Data Randomization. Technical Report, Technical Report TR-2008-120*. Redmond, Washington, USA: Microsoft Research. Cited on, 2008.

Cardenas, A., Amin, S., Bruno, S., Giani, A., Adrian, P., and Sastry, S. (2009). “Challenges for Securing Cyber Physical Systems,” in Workshop on future directions in cyber-physical systems security, volume 5.

Cerrudo, C. (2014). “Hacking US Traffic Control Systems,” in Defcon 22, Las Vegas, NV.

Charette, R. N. (2009). This Car Runs on Code. *IEEE Spectr.* 46 (3), 3. doi:10.1109/mspec.2009.5340234

Chavez, A. R., Stout, W. M. S., and Peisert, S. (2016). “Techniques for the Dynamic Randomization of Network Attributes,” in Proceedings - International Carnahan Conference on Security Technology.

Cho, J.-H., Sharma, D. P., Alavizadeh, H., Yoon, S., Ben-Asher, N., MooreKim, T. J. Dong. Seong., et al. (2020). Toward Proactive, Adaptive Defense: A Survey on Moving Target Defense. *IEEE Commun. Surv. Tutorials* 22 (1), 709–745. doi:10.1109/comst.2019.2963791

Conti, M., Crane, S., Frassetto, T., Homescu, A., Koppen, G., Larsen, P., et al. (2016). Selfrando: Securing the Tor Browser against De-anonymization Exploits. *Proc. Priv. Enhancing Technol.* 2016 (4), 454–469. doi:10.1515/popets-2016-0050

Cowan, C., Beattie, S., Johansen, J., and Perry, W. (2003). “Pointguard Tm: Protecting Pointers from Buffer Overflow Vulnerabilities,” in Proceedings of the 12th conference on USENIX Security Symposium, volume 12, 91–104.

Cowan, C., Wagle, F., Pu, C., Beattie, S., and Walpole, J. (2000). “Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade,” in Proceedings DARPA Information Survivability Conference and Exposition. DISCEX’00 (Hilton Head, SC, USA: IEEE), 119–129.

Evans, D., Nguyen-Tuong, A., and Knight, J. (2011). “Effectiveness of Moving Target Defenses,” in *Moving Target Defense: An Asymmetric Approach to Cyber Security*. doi:10.1007/978-1-4614-0977-9_2

Ghena, B., Beyer, W., Allen, H., Pevarnek, J., and Halderman, J. A. (2014). *Green Lights Forever: Analyzing the Security of Traffic Infrastructure*.

Gorgovan, C., D’antras, A., and Luja´n, M. (2016). Mambo: a Low-Overhead Dynamic Binary Modification Tool for Arm. *ACM Trans. Architecture Code Optimization (Taco)* 13 (1), 14. doi:10.1145/2896451

Habibi, J., Panicker, A., Gupta, A., and Bertino, E. (2015). “Disarm: Mitigating Buffer Overflow Attacks on Embedded Devices,” in International Conference on Network and System Security (Berlin, Germany: Springer), 112–129. doi:10.1007/978-3-319-25645-0_8

Han, K., Weimerskirch, A., and Shin, K. G. (2014). Automotive Cybersecurity for In-Vehicle Communication. *IQT Q.* 6, 22–25.

Inkster, N. (2016). Information Warfare and the US Presidential Election. *Survival* 58 (5), 23–32. doi:10.1080/00396338.2016.1231527

James, P. (2011). Farwell and Rafal Rohozinski. Stuxnet and the Future of Cyber War. *Survival* 53 (1), 23–40.

Jang, Y., Lee, S., and Kim, T. (2016). “Breaking Kernel Address Space Layout Randomization with Intel TSX,” in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS’16. doi:10.1145/2976749.2978321

Kc, G. S., Keromytis, A. D., and Prevelakis, V. (2003). “Countering Code-Injection Attacks with Instruction-Set Randomization,” in Proceedings of the 10th ACM conference on Computer and communications security. doi:10.1145/948109.948146

Kleberger, P., Olovsson, T., and Jonsson, E. (2011). “Security Aspects of the In-Vehicle Network in the Connected Car,” in Intelligent Vehicles Symposium (IV) (Baden-Baden, Germany: IEEE), 528–533. doi:10.1109/ivs.2011.5940525

KoutsouKos, X., Karsai, G., Laszka, A., Neema, H., Potteiger, B., Volgyesi, P., et al. (2018). SURE: A Modeling and Simulation Integration Platform for Evaluation of Secure and Resilient Cyber-Physical Systems. *Proc. IEEE* 106 (1), 93–112. doi:10.1109/jproc.2017.2731741

Kulandaivel, S., Goyal, T., Agrawal, A. K., and Sekar, V. (2019). “Canvas: Fast and Inexpensive Automotive Network Mapping,” in 28th {USENIX} Security Symposium ({USENIX} Security 19), 389–405.

Kulandaivel, S., Jain, S., Guajardo, J., and Cannon, V. S. (2021). “Reliable and Stealthy Remote Shutdown Attacks via Unaltered Automotive Microcontrollers,” in 2021 IEEE Symposium on Security and Privacy (SP) (San Francisco, CA, USA: IEEE), 195–210.

Larsen, M., and Gont, F. (2011). Recommendations for Transport-Protocol Port Randomization. *RFC 6051*. doi:10.17487/rfc6056

Larsen, P., Brunthaler, S., and Franz, M. (2013). Security through Diversity: Are We There yet? *IEEE Security & Privacy* 12 (2), 28–35.

Li, L., Just, J. E., and Sekar, R. (2006). “Address-space Randomization for Windows Systems,” in Proceedings - Annual Computer Security Applications Conference, ACSAC. doi:10.1109/acsac.2006.10

Lin, C-W., and Sangiovanni-Vincentelli, A. (2012). “Cyber-security for the Controller Area Network (Can) Communication Protocol,” in 2012 International Conference on Cyber Security (CyberSecurity) (Alexandria, VA, USA: IEEE), 1–7. doi:10.1109/cybersecurity.2012.7

Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Mangard, S., et al. (2018). Meltdown. *arXiv preprint arXiv:1801.01207*.

Lojack (2017). Lojack - Lojack Recovery System for Cars, Trucks, Motorcycles, Equipment, Cargo & Laptops. Available at: http://www.lojack.com/ (Accessed 08 24, 2017).

Loukas, G., Karapistoli, E., Panaousis, E., Sarigiannidis, P., Bezemskij, A., and Vuong, T. (2019). A Taxonomy and Survey of Cyber-Physical Intrusion Detection Approaches for Vehicles. *Ad Hoc Networks* 84, 124–147. doi:10.1016/j.adhoc.2018.10.002

Luk, C.-K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., et al. (2005). Pin, SIGPLAN Not. *Acm sigplan notices* 40, 190–200. doi:10.1145/1064978.1065034

Meyers, C. A., Powers, S. S., and Faissol, D. M. (2009). *Taxonomies of Cyber Adversaries and Attacks: A Survey of Incidents and Approaches. Technical Report*. Livermore, CA: Lawrence Livermore National Laboratory (LLNL.

Miller, C., and Valasek, C. (2014). *A Survey of Remote Automotive Attack Surfaces*. Isanti, Minnesota: Black Hat USA, 2014.

Miller, C., and Valasek, C. (2013). Adventures in Automotive Networks and Control Units. *Def Con* 21, 260–264.

Miller, C., and Valasek, C. (2015). *Remote Exploitation of an Unaltered Passenger Vehicle*. Isanti, Minnesota: Black Hat USA.

Mukherjee, S. (2016). *Hackers Have Crippled Another Major Hospital Chain with a Cyberattack*.

Okhravi, H., Hobson, T., Bigelow, D., and Streilein, W. (2014). Finding Focus in the Blur of Moving-Target Techniques. *IEEE Security and Privacy*.

Okhravi, H., Rabe, M. A., Mayberry, T. J., Leonard, W. G., Hobson, T. R., Bigelow, D., et al. (2013). *Survey of Cyber Moving Targets*. Lincoln Laboratory Technical Report.

Onstar (2017). Home — Onstar. Available at: https://www.onstar.com/us/en/home.html (Accessed 08 24, 2017).

Ortiz, D., Weatherford, B., Greenberg, M., and Ecola, L. (2008). *Improving the Safety and Security of Freight and Passenger Rail in Pennsylvania. Technical Report*. Santa Monica, California, USA: RAND Corporation.

Papadogiannakis, A., Loutsis, L., Papaefstathiou, V., and Ioannidis, S. (2013). “ASIST:architectural Support for Instruction Set Randomization,” in Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security - CCS ’13.

Paul, C., Todd, R. A., Yampolskiy, M., and McDonald, J. T. (2015). “In-vehicle Networks: Attacks, Vulnerabilities, and Proposed Solutions,” in Proceedings of the 10th Annual Cyber and Information Security Research Conference (New York: ACM), 1.

Paul, K., Horn, J., Anders, F., Genkin, D., Gruss, D., Haas, W., et al. (2019). “Spectre Attacks: Exploiting Speculative Execution,” in 2019 IEEE Symposium on Security and Privacy (SP) (San Francisco, CA, USA: IEEE), 1–19.

Paul, K., Lee, R., McGraw, G., Raghunathan, A., and Moderator-Ravi, S. (2004). “Security as a New Dimension in Embedded System Design,” in Proceedings of the 41st annual Design Automation Conference (New York: ACM), 753–760.

Petit, J., and Shladover, S. E. (2015). Potential Cyberattacks on Automated Vehicles. *IEEE Trans. Intell. Transportation Syst.* 16 (2), 546–556.

Portokalidis, G., and Keromytis, A. D. (2010). “Fast and Practical Instruction-Set Randomization for Commodity Systems,” in Proceedings of the 26th Annual Computer Security Applications Conference. doi:10.1145/1920261.1920268

Ray, D., and Ligatti, J. (2012). “Defining Code-Injection Attacks,” in Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages. doi:10.1145/2103656.2103678

Riley, J. (2004). *Terrorism and Rail Security*.

Rosenzweig, P. (2012). *Alarming Trend of Cybersecurity Breaches and Failures in the U.S. Government*.

Sanchez, R. (2016). *NJ Train Didn’t Have This Safety System. Could it Have Stopped the Crash? - CNN.Com*.

Scott, K., and Davidson, J. (2001). Strata: A Software Dynamic Translation Infrastructure. *IEEE Workshop on Binary Translation*.

Seshia, S. A., Hu, S., Li, W., and Zhu, Q. (2017). Design Automation of Cyber-Physical Systems: Challenges, Advances, and Opportunities. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* 36 (9), 1421–1434. doi:10.1109/tcad.2016.2633961

Shacham, H., Page, M., Ben, P., Goh, E-J., Modadugu, N., and Boneh, D. (2004). “On the Effectiveness of Address-Space Randomization,” in Proceedings of the 11th ACM conference on Computer and communications security - CCS ’04. doi:10.1145/1030083.1030124

Shane, T., Glavin, M., Hughes, C., Jones, E., Trivedi, M., and Kilmartin, L. (2015). Intra- Vehicle Networks: A Review. *IEEE Trans. Intell. Transportation Syst.* 16 (2), 534–545.

Sinha, K., Kemerlis, V., Pappas, V., Simha, S., and Keromytis, A. D. (2014). *Enhancing Security by Diversifying Instruction Sets*.

Sniatala, P., Iyengar, S. S., and Ramani, S. K. (2021). “Industrial Involvement in the Use of Homomorphic Encryption,” in Evolution of Smart Sensing Ecosystems with Tamper Evident Security (Berlin, Germany: Springer), 85–88. doi:10.1007/978-3-030-77764-7_11

Snow, K. Z., Monrose, F., Lucas, D., Dmitrienko, A., Liebchen, C., and Sadeghi, A-R. (2013). “Just-in-time Code Reuse: On the Effectiveness of fine-grained Address Space Layout Randomization,” in 2013 IEEE Symposium on Security and Privacy (SP) (Berkeley, CA, USA: IEEE), 574–588. doi:10.1109/sp.2013.45

Stephen, C., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S., et al. (2011). “Comprehensive Experimental Analyses of Automotive Attack Surfaces,” in USENIX Security Symposium, San Francisco.

Studnia, I., Vincent, N., Alata, E., Deswarte, Y., Mohamed, K., and Laarouchi, Y. (2013). “Survey on Security Threats and protection Mechanisms in Embedded Automotive Networks,” in 2013 43rd Annual IEEE/IFIP Conference on Dependable Systems and Networks Workshop (DSN-W) (Budapest, Hungary: IEEE), 1–12. doi:10.1109/dsnw.2013.6615528

Szekeres, L., Payer, M., Wei, L. T., and Sekar, R. (2014). Eternal War in Memory. *IEEE Secur. Privacy* 12 (3), 45–53. doi:10.1109/msp.2014.44

Szilagy, C., and Koopman, P. (2008). *A Flexible Approach to Embedded Network Multicast Authentication*.

Teso, H. (2013). *Aircraft Hacking Aircraft Hacking Practical Aero Series Practical Aero Series*.

Wang, Z., Cheng, R., and Gao, D. (2011). “Revisiting Address Space Randomization,” in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). doi:10.1007/978-3-642-24209-0_14

Weiss, Y., and Barrantes, E. G. (2006). “Known/Chosen Key Attacks against Software Instruction Set Randomization,” in Proceedings - Annual Computer Security Applications Conference, ACSAC. doi:10.1109/acsac.2006.33

Wolf, M., and Serpanos, D. (2018). Safety and Security in Cyber-Physical Systems and Internet-Of-Things Systems. *Proc. IEEE* 106 (1), 9–20. doi:10.1109/jproc.2017.2781198

Wolf, M., Weimerskirch, A., and Wollinger, T. (2007). State of the Art: Embedding Security in Vehicles. *EURASIP J. Embedded Syst.* 2007 (1), 074706. doi:10.1186/1687-3963-2007-074706

Wu, W., Li, R., Xie, G., An, J., Bai, Y., Zhou, J., et al. (2020). A Survey of Intrusion Detection for In-Vehicle Networks. *IEEE Trans. Intell. Transport. Syst.* 21 (3), 919–933. doi:10.1109/tits.2019.2908074

Zhang, T., and Delgrossi, L. (2012). *Vehicle Safety Communications: Protocols, Security, and Privacy, Volume 103*. Hoboken, NJ, USA: John Wiley & Sons.

Zhuang, R., Zhang, S., Bardas, A., DeLoach, S. A., Ou, X., and Singhal, A. (2013). “Investigating the Application of Moving Target Defenses to Network Security,” in Proceedings - 2013 6th International Symposium on Resilient Control Systems, ISRCS 2013. doi:10.1109/isrcs.2013.6623770

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.