YOU'VE IMPLEMENTED ENCRYPTION, NOW WHAT?

Using Mandatory Access Control & Separation Techniques to increase system integrity during and after an attack

Jonathan Kline, Star Lab

Summary

Vehicle and transportation systems provide a challenging use case for security, where physical and virtual access is virtually unlimited, and its relatively easy to procure replacement parts, components, and even whole vehicles to continue attacking a system. Given these challenges in the vehicle deployment model, the goal shifts from denying an attacker access, to limiting the damage an attacker can do once access is achieved. Specifically, the integrity and availability of the system (vehicle) must be maintained even after a successful compromise. Historically embedded devices, and platforms such as ground vehicles, have functioned in isolation, deployed to environments minimally connected to the outside world. However, with the emergence of ubiquitous connectivity paradigms like the Global Information Grid (GIG), and multiple battlefield communications, networks and radio systems, today's cyber threat landscape is rapidly evolving.

Traditionally, fielded systems, such as ground vehicles have been secured from cyber-attack with guards and guns. The assumption was that an attacker would be denied physical access to the vehicle, and the vehicle would be sanitized prior to being abandoned. However, with the modern emphasis on network connectivity and inter-operability, protecting the ground vehicle platforms using a stand-off approach is no longer viable. As these systems are increasingly networked, the DoD has responded with a subset of commercial practices, namely deploying firewalls (*i.e.*, keep attackers out) and patching software. Firewalls are demonstrably insufficient (Owens, 2009). Patching software—especially in Army ground vehicle settings—does not keep pace with attacks and the move to inter-connected systems architectures (Owens 2009; Timberg, 2014). Other security products such as antivirus tools, limited in their own right, are incompatible with real-time operational systems and require frequent updates.

Central to this evolution is the ease with which a focused and resourced adversary can acquire and reverse engineer deployed (or abandoned) ground vehicle systems. In addition to modification or subversion of a single specific device, hands-on physical access also aids an attacker in discovery of remotely-triggerable software vulnerabilities across a broad range of deployed vehicle systems. Adversaries with physical access primarily approach their reverse engineering activity in two ways[i]: 1) by directly acquiring and reverse engineering device firmware, and 2) by connecting to a device via JTAG or other debug interface, e.g., Android's ADB or ARM's CoreSight, in order to observe and manipulate runtime behavior. The goal of both approaches is to bypass run-time policy-based protections so that the device software can be accessed, reverse engineered, modified, or otherwise exploited.

Encryption works to protect data at rest, but it fails to provide adequate protection at runtime. Once an attacker gains access to a system, whether via an interface like JTAG or over-the-air execution, they are able to observe (and possibly modify) program state and execution flow. For example, a compromised device may have its firmware modified to turn off secure communication protocols, or an attacker may introduce malicious code that causes physical damage or fails to report accurate sensor information. Worse, a hands-on attacker may simply be interested in discovering hidden backdoors or remotely-exploitable software vulnerabilities for future undetected access or mass disruption. This is the security fallacy of encryption; it is not a miracle one size fits all solution. Encryption needs to be implemented as a layer in the overall system architecture.

The use of encryption is only the beginning in terms of securing vehicle systems and is generally not sufficient to preserve the integrity of these systems during or after a successful attack.

This paper focuses on:

1. Security fallacy of "just encrypt"
2. Attack flow and overview (setting the stage for MAC)
3. Introduction of Mandatory Access Control (MAC)

4. Kernel implementations of MAC
5. Hypervisor implementations of MAC
6. Separate execution contexts
7. Further decreases to the attack surface

Security fallacy of "*just encrypt*"

Encryption is one of the first steps used in securing vehicle systems. However, as has been shown previously in a variety of systems (Noubir, 2015) (Wachter, 2012) (Manuel Egele, 2013), encryption alone is not a sufficient protection mechanism for critical systems. In order to provide some level of protection, encryption requires careful considerations for key management, implementation details, key sizes, key rotation, performance, system usability and validation. All too often, encryption techniques are implemented without proper verification, or with implementation vulnerabilities that decrease their overall effectiveness and enable an attacker to compromise the integrity of the rest of the system.

As an example, if encryption is used to protect the system boot image, the entire boot image including headers should be encrypted and verified on each boot. Encrypted boot images should be verified using a signed hash. A boot image that has not been encrypted and verified can easily be modified or manipulated by an attacker. In cases where the image is not hashed and signed or otherwise verified, the attacker can manipulate what will be decrypted or in some cases verified, often enabling the attacker to gain control through a variety of unanticipated system exploits and side-effects.

In cases where encryption is the only security mechanism, a shortcoming or vulnerability with the encryption architecture compromises the integrity of the rest of the system, enabling an attacker to achieve full control over the vehicle system regardless of the crypto in place. Further, while encryption provides protection for data and applications at rest and in transit, it does not typically protect data at run-time. In the event an attacker is able to gain access to the system through a system under-configuration, exploit, latent debug interface, or even physical access, the attacker is able to access and potentially modify any data that is currently decrypted as part of the running system. Data which has been encrypted, must be decrypted before it can be used. All too often devices share a single execution context, enabling an attacker to achieve full control over the system and visibility into decrypted data and applications once a single point of initial access has been achieved, even though data or applications may be stored encrypted.

In order to raise the bar, and set the stage for the implementation of advanced isolation and access control, encryption should be implemented with other industry best-practices such as: lockdown/removal of engineering and flash update tools, complete removal of unnecessary tools and libraries (attack surface), secure boot founded within a hardware root of trust, and having the system evaluated by an external entity.

Encryption is a foundational pillar to successful system security, however encryption alone does not hold all of the answers and is only part of the overall secure cyber solution. In addition to encryption, proper cyber security of interconnected ground vehicle systems requires:

- Enforcement and integrity verification of system configuration
- OS Hardening and attack surface reduction
- System resource access control
- Secure software updates
- Application Containerization
- IP/Data Protection
- Jailbreaking prevention and execution of unauthorized applications

Attack flow and overview (setting the stage for MAC)

Regardless of whether an attacker is attacking a system through virtual access (over the wire, software updates, software exploit, etc.), or physically they will generally follow the same steps in order to gain access to the system. The general flow for an attack starts with system reconnaissance. Next, an attacker will exploit a vulnerability to

gain initial execution context on the system. After achieving initial access, the attacker will attempt to escalate privileges if necessary, or continue on to exploiting the system in order to achieve their individual goals.

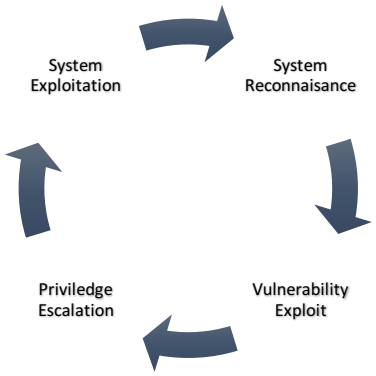

Figure 1: Attack Flow

As shown in Figure 1, the attack process is often circular, and cycles between reconnaissance, gaining access, performing system analysis, and increasing access. It is often necessary to perform several iterations of the process in order to achieve the intended system exploitation goals.

Introduction of Mandatory Access Control (MAC)

MAC enables system developers and integrators to restrict access to system resources and data to specific users, classes of users, applications, hardware/software objects, and sub-systems. MAC provides logical isolation between applications running on the same system and it limits what an attacker is able to do once initial access has been achieved. This is true whether the access is through a vulnerable application, system bus, or shell console. When implemented correctly, MAC limits what an attacker can do once they have already gained access, and provides continued integrity of the rest of the system. There are two primary means of implementing isolation and MAC: kernel extensions or a separation kernel. In cases where the target hardware supports it, a separation kernel or separation hypervisor can be used to separate processes and software subsystems while also limiting what an attacker can access. Alternatively, MAC can be implemented through the use of OS and kernel extensions.

MAC, whether implemented through existing OS and kernel mechanisms, or through a hypervisor, enables system integrators to limit access to resources, communication mechanisms, OS facilities, hardware, and separate applications, libraries, and data. MAC can even limit what the *root* or *administrator* user can do and access on the system, making it very difficult for an attacker to gain total control of the system and exploit specific subsystems. MAC can be thought of as a series of explicit policies, governing what exactly a user, application or defined role can do on the system. MAC policies are generally either established during a learning phase, in which normal system activity is recorded over a set interval, or the policy is explicitly defined by the system developer. In the case where the policy is developed based on actual use of the system over a defined interval, the established policy can be tailored to provide additional protection, and address paths or access vectors not exercised during the learning cycle. The MAC policies are used to identify permitted activities and accesses on the system, under the premise that whatever is not explicitly permitted is denied.

Leveraging MAC on a system such as an Electronic Control Unit (ECU), Networking Gateway, or In-vehicle Infotainment (IVI) platform, significantly increases the burden for the attacker, and works to limit the damage once an attacker gains initial access to the system. During the initial access phase, MAC significantly decreases the available attack surface, and makes it harder for the attacker to gain execution context, alter system firmware, interact with hardware devices, and elevate privileges. Attackers will need to spend significantly more time and resources analyzing the system and preparing attacks, thereby enabling more chances for detection, mitigation and prevention of the attack. MAC can also be used to limit the resources available to an attacker, thereby significantly decreasing the available attack surface, and increasing the burden for analysis as attackers are forced to look for even smaller attack vectors. Once an attacker gains execution on the system, MAC can prevent the attacker from elevating privileges, subverting trusted boot mechanisms, interacting with other components or buses of the system, and transferring aspects of the system for offline analysis. MAC can also be used to prevent an attacker from

bringing their own tools (by limiting access to system calls, not being able to execute untrusted applications, and preventing execution access from directories the attacker controls), which further helps preserve integrity of the system, and the vehicle network.

Kernel implementations of MAC

MAC implemented in the kernel provides application and resource separation, however it also runs at the same access level as the attacker, providing the attacker a potential mechanism for subverting the afforded protections. Kernel implementations of MAC provide strong context for executing applications enabling finer grained access control. Kernel implementations of MAC enable file-level granularity for loading libraries, accessing resources, and controlling access to inter-process resources. Kernel-based MAC solutions harden and lock down the deployment kernel, system configuration, application/data files, and sensitive processes in order to significantly decrease the impact of cyber attacks. Kernel-based MAC can be applied in order to prevent encrypted files within the system from being accessed by non-privileged processes, and the corresponding processes can be prevented from being paused, killed, or debugged at runtime. Furthermore, kernel-based MAC ensures there is no administrative way to bypass these protections in deployed systems (either at rest, startup, or runtime). Kernel-based MAC solutions are able to implement the capabilities shown in Table 1. Kernel-based MAC provides an easy-to-integrate capability for increasing embedded OS security, and helps ensure the integrity and confidentiality of critical system software even in the face of a root-level attacker.

Capability	Protects
Mandatory Access Control (MAC)	Files, Applications, System Interfaces
Anti-Access / Anti-Debug	Application Debugging, Device Access
Isolation and Containerization	Applications
File Security and Integrity	Applications, Configs, and Data Files

Table 1: Kernel-Based MAC Capabilities

Hypervisor implementations of MAC

Through the use of a separation hypervisor, not only are the protections removed from the purview of the attacker and moved to a lower level in the system, but full hardware resource access controls and separation can be enforced, including hardware supervisory control, software partitioning, and integrity monitoring requirements across multiple software domains. The logical isolation afforded by a modern type-1 separation hypervisor provides a foundation for cyber attack resilience by preventing errant or malicious code in one domain (execution context) from being able to read/write memory, manipulate resources, or otherwise affect operations in another domain. Furthermore, the hardware-enforced memory protections configured by the hypervisor ensures each execution domain remains isolated from both the other execution domains as well as malicious peripheral hardware. The use of a separation hypervisor also enables device drivers that interact directly with hardware, e.g. network cards and disk controllers, to be isolated into standalone execution domains preventing them from being used as a vector to gain access to the rest of the system. Using a hypervisor for separation ensures each application or service is executed in its own context independent of other applications or services on the system.

Separate Execution Context

Placing each application in its own execution context, whether it's a container, namespace, or separate virtual machine, not only guarantees resource independence, but also enables more fine grained control over resources, and access to the rest of the system. As an example of containerization (FIGURE 2) individual applications are limited to the devices, resources, and files available to them, thereby decreasing the overall attack space and increasing the overall security of the complete system. Containerization also makes it significantly harder for an attacker to exploit or otherwise compromise a specific application, and use that compromise in order to pivot to other applications, hardware or interfaces present in the system.

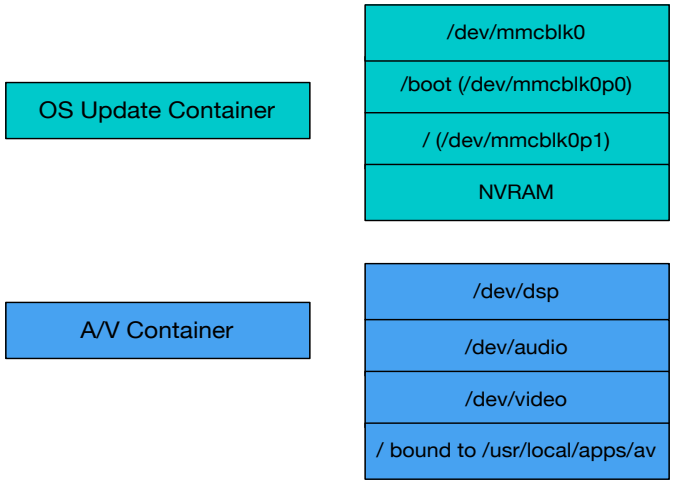

Figure 2: Application Containerization

The use of sandboxed containers for applications makes it harder to gain control of the full system from within a vulnerable application, and forces an attacker to also break out of the container in order to interact with the rest of the system and subvert the protections. LURE supports the grouping of applications in a container, such that separate containers can be created for update mechanisms, system utilities, network components, and 3rd party software. Additionally, execution of a given application or service could be mirrored to a secondary execution context where the data flows can be monitored (without impacting overall system performance or timing). This would enable a broader range of attacks and data corruption to be identified and responded to out of band. The hypervisor can also be used to prevent access to various system calls in each execution context, such as those for debugging an application, mounting filesystems, or elevating privileges. Running each application or service in a separate execution context, in essence enables the application to be executed in a similar fashion to a microservice. As such, only the required system services, access mechanisms, and resources are available to the application, significantly decreasing the attack space, and increasing the protections afforded to the system and application. Separate execution contexts can also be created for encryption services, enabling all of the encryption of applications, libraries, data, and other resources to happen transparently behind the scenes, without exposing the operations or keys to an attacker. Implementing encryption, key management, and signature checking in a separate execution context or virtual machine, can make it increasingly difficult for an attacker to dump or analyze the keys and subvert the afforded protections. Additionally, a separate execution domain can be established for software and firmware updates. The update and maintenance execution context can be configured as the only environment that can access hardware write mechanisms, security processors, and trusted boot mechanisms. The execution context can be stored encrypted with the vehicle, and the key material for it can be provided by the dealer, service center, or operational authority when required; mechanisms for performing over-the-air updates can also be implemented while still maintaining a separate service / maintenance execution context.

Decreasing the Attack Space

In addition to MAC afforded by either kernel or hypervisor mechanisms, it is important to decrease the attack surface of both the operating environment and the protection services to minimize the attacker's opportunity to subvert the protections and gain complete control of the vehicle's systems. For example, simply removing binaries (i.e. gdb, strace, kexec) from a system does not mean the underlying OS functionality has been removed. The functionality may still exist (i.e. in terms of system calls and kernel functionality), and if an attacker can bring their own tools, the latent functionality can then be leveraged to further knowledge of the systems and increase an attacker's access to the system. Functionality should be removed from the host operating environment first, and if necessary, supplemented through the use of a hypervisor and trusted execution environment.

Decreasing the attack space also requires enforcing MAC on various system interfaces which expose runtime information about the applications and their data. This includes IPC mechanisms, sockets, pipes, file descriptors, and file system type checking. Further, MAC can be used to increase the overall security posture of the system by performing strict type checking on file system accesses, limiting access to physical resources on the system, and verifying system configurations before use.

Works Cited

Manuel Egele, D. B. (2013, 11 04). An Empirical Study of Cryptographic Misuse in Android Applications. *ACM*.

Noubir, G. (2015, 08 12). Network Security: Use & Misuse of Cryptography -- Contemporary Tales. Boston, MA: Northeastern University.

Wachter, A. K. (2012). Ron was wrong, Whit is right. *Cryptology ePrint Archive, 2012*(064).

W. Owens, K. Dam, and H. Lin "Technology, Policy, Law, and Ethics Regarding U.S. Acquisition and Use of Cyberattack Capabilities," Committee on Offensive Information Warfare, National Research Council, ISBN: 978-0-309-13850-5, 2009

Timberg,Craig and Nakashima, Ellen. http://www.washingtonpost.com/business/technology/government-computers-running-windows-xp-will-be-vulnerable-to-hackers-after-april-8/2014/03/16/9a9c8c7c-a553-11e3-a5fa-55f0c77bf39c_story.html 2014

[i] A third tamper approach is also possible where sophisticated, well resourced adversaries perform invasive and non-invasive physical, electrical, and EM attacks against individual hardware components using specialized tools like SEM's, SOM's, FIB's, SPA/DPA, etc. Full protection against such attacks are not addressed in this paper.