Memory-Map Shuffling: An Adaptive Security-Risk Mitigation

Pierre Schnarz*, Joachim Wietzke†, Andreas Rausch*
*Software Systems Engineering
Technische Universität Clausthal, Germany
Email: {pierre.schnarz—andreas.rausch}@tu-clausthal.de
†Hochschule Karlsruhe, Karlsruhe, Germany
Email: joachim.wietzke@hs-karlsruhe.de

***Abstract*—Automotive products, such as electronic control units, evolve increasingly towards adaptive solutions. From many perspectives, solutions need to be flexible with regards to the entire originating process and operation process. Here, the product development cycle, the product life-cycle and even product lines describe the dimensions a solution might have to adapt to. Certain requirements to secure the product continuously add further complexity to the aforementioned dimensions. Adversaries adapt - so the protection shall as well. However, adapting, particularly, technical solutions to products, implies the need for agnostic approaches. In this article, we propose a security-risk mitigation concept which aims to fit into the evolving automotive originating process applied to a particular class of electronic control units. Technically, the proposed approach shuffles the system memory-map of an asynchronous multiprocessing system. On the intermediate layer between the hardware and software, the assignment of memory and resources is obfuscated to a potential adversary who managed to breach one of the higher level memory protection mechanisms. As a result, the proposed mitigation adds either a further level in a defense-in-depth security architecture or fixes a structural vulnerability of certain hardware architectures.**

Keywords–Security; Mixed-Criticality; Obfuscation; Automotive.

I. INTRODUCTION

When it comes to automotive security, hardened electronic control units (ECU) are required to resist the emerging threat and attack landscape [1] [2]. Being resistant to threats and attacks is a continuous process. This is motivated by the fact that the adverse actions and methods against the system evolve over time. In other words, the adversaries change and find new methods, entry-points and tools to compromise a particular system. As a result, the risks that are related to the functionality of the system would increase. New countermeasures are necessary to limit the likelihood of an impact on safety, financial, operational or privacy aspects [3] [4]. The mentioned continuous process of reacting to the evolving security incidents has multi-dimensional impacts on the origination process of automotive products. These dimensions include: the product life-cycle including the development life-cycle and on a larger scale the product-line evolution. The technical goal of a security mitigation is the reduction of a security-risk. Besides that, the mitigations are required to fit into the aforementioned adaptive origination process. Depending on the particular phase in which a certain product faces the need for risk mitigation, the range of immutable (or static) system components might be wide. For example, changes to the hardware are nearly impossible after the start of production (SOP). As a result, changes to the software components of a product are targeted.

Adaptivity, the evolutionary environment and cost reductions are just a few arguments to move vehicular functions into a highly integrated platform (such as ECUs). The results are very powerful but complex systems. Important is the aspect of mixing functions which imply diverse system quality demands. For example, a single platform aggregates functions which on the one hand operate break-assistance features and infotainment in parallel. Such systems are usually referred to as mixed-criticality systems (MC-system) [5]. From an organizational point of view, these functions are required to operate as they did on separated ECUs before. This is particularly true for security. In particular, the separation and isolation aspects are key in such environments. It must be ensured that no interference between certain functions is possible. However, for example in security, the strength might need to be adapted over time due to the emerging threat landscape. This is even more important, since the functions will be adapted over the product-lifetime.

In this work, we propose a technical mitigation concept which is adaptable to highly integrated platforms. The concept is also driven by the evolutionary automotive product origination landscape. Technically, the mitigation approach aims to protect memory partitions, of certain functions, from exploitation. This is achieved by shuffling of address translation mappings of commodity virtualization mechanisms. Due to the obfuscated memory structure, the risk of further compromisation is mitigated. Metrics to characterize the exploitability [6] and effectiveness [7] are given.

In section V the memory-map shuffling concept is introduced and analyzed. The rest of the article is structured as follows. In section III the target of evaluation (ToE) is defined. In the following sections the threat analysis (compare Section IV) and the particular attack vectors (compare Section IV-B) are described. Section VI gives an outline of the effectiveness of the given approach. Lastly, section VII contains concluding remarks.

A. Related Work

The idea of obfuscating addresses is not new in certain areas. On application level, address obfuscation is adopted by many operating systems. Particularly, in general purpose operating systems such as *Linux*, *Windows* and *Mac OSX* this technique is actually state-of-the art. As of today, this is said to be one of the most effective countermeasures against memory

 ISBN: 978-1-61208-532-6

exploits. Recent efforts brought that technique down to the system level, by randomizing the operating system's (OS) kernel address space [8]. In Linux, for example, the developers aimed for a significant increase of system security by making attacks into the monolithic kernel space less predictable for adversaries. In [9] Bhatkar et al. describe address obfuscation as an efficient approach to combat memory error exploits. The authors argue that these attacks require an attacker to have an in-depth understanding of the internal details of a victim program, including the locations of critical data and/or code. Therefore, program obfuscation is a general technique for securing programs by making it difficult for attackers to acquire such a detailed understanding. Kil et al. extend in [10] the idea of address space layout permutation to enable a finer grained randomization. Generally, they address one of the biggest drawbacks of current address space randomization implementations, namely the lack of a cryptographic secure entropy. Possible adversaries are able to guess the locations statistically in a very short amount of time, since the number of bits used for randomization is very limited. The permutation of address layouts facilitate to combat attacks using techniques such as buffer-overflows, format string attacks and code re-usage attacks like return oriented programming (ROP). In [11] Shuo et al. introduces a method to utilize hardware virtualization in order to prevent ROP attacks within the kernel. In [12] Rushanan et al. elaborate on the concept of malicious behavior based on direct memory access (DMA) transfers. The attacks are implemented using commodity desktop hardware. Although the implementation is not applicable to embedded hardware, the DMA issue is transferable to the attack surface of embedded system-on-chips (SoC).

II. Security of Evolving Automotive Products

The issue of securing products depends on the evolutionary state inside the product life-cycle (PLC) or outside within the product-line. Generally, the PLC is mainly focused on when it comes to security processes. PLCs of automotive products are roughly dividable into two main phases. First, in the *pre-SOP* phase, the product will be developed using a suitable development cycle, such as the v-model. Second, in the *post-SOP* phase, the product needs to be maintained. From the security perspective, two major goals are spread over these two phases. The first goal is to create a state in which the system can be treated as secure. In other words, to be aware of risks in the first place and to mitigate or accept them in the second. The second goal is to keep a certain risk threshold in which the system is still in this secure state. Most commonly, the applied method to find security requirements in the pre-SOP phase is risk assessment [3]. For every function the system has, a potential impact and likelihood will be analyzed. With respect to the particular risk, a risk treatment phase follows and the specific strategies to mitigate those risks will be defined. Those mitigations are fed as logical requirements and technical requirements into a system design. In the verification and validation phase of the v-model, appropriate security testing methods are applied to raise confidence in the absence of severe vulnerabilities. Security testing methods include fuzzing (negative testing) and penetration testing. In other words, during the development, the foundations for determining the security requirements are built accompanied by techniques to gain confidence in the derived logical and technical security architecture. During the post SOP phase, the system should be observed. If an incident occurs a proper response should be initiated. Accordingly, if this response requires a security update (software) the development of this update will traverse the secure development v-model for the new function. Limiting factors for the upcoming security mitigations are immutable components of the system which might be functional or technical. A prominent example is the hardware platform which is obviously hard to modify once it is deployed. However, this immutability is not only true for products that are already deployed, as practically within product lines the engineering strategy such as top-down and bottom-up might also imply further restrictions. For example, a hardware platform is to be integrated (bottom-up) for a certain set of software functions. Some of the functions then need to be fitted and developed onto (top-down) the hardware platform. This also implies restrictions for security solutions. This might appear in many reuse situations in automotive product line development. To summarize, security solutions need to fit into this evolving landscape.

III. Mixed-Criticality System

Mixed-criticality systems integrate multiple organizational domains (MC-domains), each of which potentially has different demands on the necessity (criticality) of the fulfillment of quality goals. Quality goals are for example dependability aspects, performance, etc. [13]. MC-domains are facilitated by combining several functional and technical components of a system. As an example, technically it might contain a software stack including OS, containers and applications. The technical implementation of the MC-system is discussed in the following section.

A. Facilitate MC-systems by Means of AMP

Where the definition of MC-systems describes a functional and logical setup, AMP refers to the technical part. In general, AMP is a system utilization paradigm which aims to control hardware elements independently by multiple operational units. As such, AMP systems can facilitate a MC-system by assigning technical means at the hardware level and software level. In other words, it is a configuration of the hardware to create the logical layers on top of it. In the following, this level is referred to as the *intermediate level*. As mentioned in Section III, the separation of functionality is a fundamental requirement to implement a proper MC-system. From the hardware level up to the application level, there are several technical possibilities to create a logical separation. Technically, each layer of a system provides means to create separated domains. For example, applications are separated most commonly in processes and threads which are provided by the OS. At the lower architectural layers, virtualization technologies emerge within the automotive environment. The aim is to combine multiple OS in one platform [14] [15].

B. Target of Evaluation

In this work, the target of evaluation (ToE) adopts the AMP paradigm and facilitates a MC-system on the intermediate level. In Figure 1 the ToE is depicted. It shows two MC-domains, *MC-domain1* in red color and *MC-domain2* in blue color. The figure indicates a software stack assigned to each of the two domains. Furthermore, the hardware layer is modeled. It shows a minimalistic set of elements of a commodity

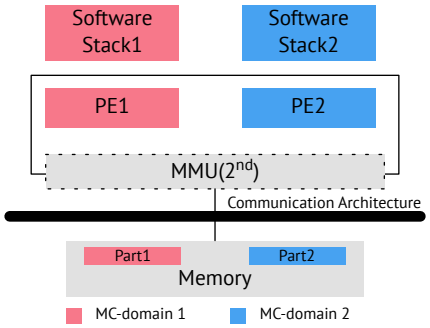

Figure 1. Target of evaluation.

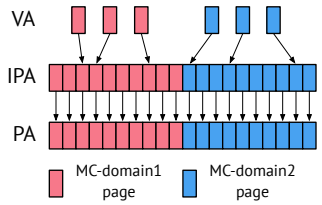

Figure 2. Identity memory-mapping principle.

SoC. The elements inlcude processor cores such as processing elements (PE) (denoted by *PE1* and *PE2*), a communication architecture and main memory. *PE1* is assigned to *MC-domain1*. Accordingly, *PE2* is assigned to *MC-domain2*. Each Mc-domain includes its own memory partitions, which are denoted by *Part1* and *Part2*.

C. Memory-Maps in AMP Based Systems

In an AMP system, typically three types of addresses are handled. First, the virtual address (VA) space at user level which is maintained by the operating system to provide horizontal memory separation of processes. Second, the physical address (PA) space which represents the address of the main-memory. Last, the intermediate physical address (IPA) space which is introduced to be able to separate the MC-domains (or virtualized OS). The translation from VA to IPA is referred to as *stage 1* translation and those from IPA to PA as *stage 2* translation respectively. Depending on the particular implementation of the PE the translation is handled by a memory management unit (MMU) in hardware. The structure which is used to identify corresponding addresses is referred to as a mapping table filled with a finite set of translation entries. In the translation process, the MMU extracts the most significant bits of the input address to index the translation table. The output address resides at the given index. In practice, those entries map to a specified amount of memory which is commonly referred to as a memory page. The granularity differs among hardware architectures. A simplified example of a typical address map is shown in Figure 2. The figure shows the entire PA space (PA) on the bottom. Above, the IPA stage mapping is depicted. The relationship between the two address stages represents the actual mapping. In this case, the so-called *identity* mapping is shown. Identity mapping means that there is no translation (or redirection) of memory addresses between stage 1 and stage 2. The MMU is only used for memory protection in this case.

D. Memory Access Control

Accesses to the distinct hardware elements is enforced by a MMU. Since this work focuses on access control, other means such as interrupt routing are not considered any further. In this type of AMP-based system, the access control is configured by the memory-map table of the particular MMU. In Figure 3 the access control principle for the intermediate level is shown. The PE are considered to be the subjects requesting access to a certain memory area. In this case, those memory areas are the partitions (*Part1* and *Part2* shown in Figure 1) and therefore the accessed objects. The MMU enforces the memory accesses. The access control policy is manifested in the memory-map. An important prerequisite is that the MMU memory-map, in this particular case a second-stage MMU, must not be accessible by any of the MC-domains itself. This must be handled by a higher privileged instance. Most commonly this is referred to as the hypervisor.

IV. Threat and Attack Analysis

Threat and attack analysis facilitates two core methods to identify and rate security risks. Threat analysis aims for structured decomposition of systems and the derivation of security threats [16]. A commonly accepted and applied threat model is STRIDE (spoofing, tampering, repudiation, information disclosure, denial-of-service and elevation of privilege) which is shown in [17]. Depending on the system element, a particular subset of the previously mentioned threats is applicable. Attack analysis is an offensive method which aims to foresee and factorize the behavior of an adversary. During the assessment, the minimum effort to exploit a specific component of the system is to be estimated. A qualitative and quantitative statement (risk) results in the exploitability estimation with regard to a particular threat. The exploitability factors which are applied in this work are adopted from the common vulnerability scoring system [6]. The ontology between the threats and attacks are adopted from the risk model given in [7]. To summarize up, the threats are a functional categorization of security risks. Whereas, attacks refer to the technical facilitation of a certain threat category.

A. Threat Landscape and Memory Asset

In the first place, our work focuses on *tampering* threats on the memory partitions of the MC-domains. With regard to the example given in Figure 3 one MC-domain tampers with the memory partition ($Part_i$) of another domain. Despite, the compromising the integrity of main memory can be the root-cause for further or even more advanced threats. It is worth to mention, that tampering might lead to elevation of privileges or one subsequent step of spoofing a communication link to another entity. Even though denial-of-service attacks can be mounted by tampering with the memory base of a system. This is motivated by the fact, that software intensive systems rely on their code and data base stored in the main memory. Tampering with that does not only compromise and modify information, but also the control flow integrity of their function. Meaning, by having the ability to deliberately change the control flow, an adversary might gain full or partial control of the vehicle's

 ISBN: 978-1-61208-532-6

behavior. Impacts on safety and operation of the entire vehicle are severe. As a result, memory storage is an important asset.

B. Attack Vector

As it is mentioned before, the factorization of attacks gives insight on the exploitability of a particular threat. Technically, it shall be assumed that an unintended access to main memory is caused by a defect or vulnerability. Accordingly, in this section, we elaborate on potential breaches. In the following, the potential preconditions are elaborated. Ultimately, this work considers attacks at the intermediate level. However, in order to mount exploits on this level, the adversary is required to break into the system first. There must be an entry point for the attacker. In ECUs of a vehicle, this might be facilitated by external telematic interfaces, internal vehicular buses or entertainment media. A comprehensive analysis of vehicular attack surfaces is shown by Checkoway et al. in [18]. Once the adversary has successfully entered the system further exploits might be necessary to break the memory access control. We assume that the adversary needs to break several levels before he reaches the intermediate layer. For example, if he succeeded to take over an application he needs to elevate his privileges towards OS level. This might be done via a root/kernel exploit [19]. Once the adversary reached the intermediate layer the memory access control mechanisms (compare Figure 3) need to be exploited. By compromising or circumventing these isolation controls, the main memory then is fully exposed to further exploitation. As an example, in various experiments, it has been demonstrated that state-of-the-art multiprocessor SoCs running AMP-based multi operating systems imply architectural weaknesses in memory protection [12] [20] [21] [22]. This flaw suffers mostly from insufficient hardware support for throughout system isolation. In addition, the fixed and static memory layout at the intermediate system level provides a wide surface for attacks. If an adversary successfully circumvented the memory protection, the static memory configuration enables an attacker to aim for particular data or information contained in the memory. Once the memory isolation is breached, the adversary has full access to the memory. Before the attacker can manipulate the data or the code he must locate its target structure within the memory space. Since multiple MC-domain partitions are present in the main memory, the attacker first needs to determine the base address of it's targeted partition. Starting from this offset, the re-interpretation of OS memory structures can be initiated. Re-interpreting the kernel structure is a broad research field of computer forensics, for example by Andrew Case et al. [23]. By assuming a consecutive memory structure the attack can follow references within those structures until the target is found. In addition to the re-interpretation of the structures, the attacker could simply scan for specific binary patterns or so-called *magic bytes* to reach its target structure.

C. Attack Complexity

With respect to the metrics in [6] the following aspects refer to the exploitability. The required entry attack vectors and required privileges are discussed in the previous section. With regards to the complexity of the attack, the attacker does not need to conduct a target-specific reconnaissance. The system configuration is considered to be static due to the fixed memory mapping. There is no intended variation from target to target.

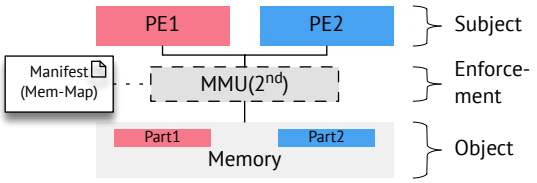

Figure 3. Access control at intermediate level.

As a result, once an adversary is able to breach the memory protection for one system, the concept is applicable to other systems in the product-line or vehicle fleet.

D. Scope and Elevation of Privilege

Due to its nature, memory protection breaches allow a wide range of impacts on security goals. In this particular case, the adversary is able to conduct control flow integrity attacks and elevate the privileges of certain functions of the system. This implies a change of scope [6], meaning the attacker is not only able to control the targeted function but also further assets from this position.

E. Impact to General Security Goals

With the herein assumed attack vectors a potential adversary is able to disclose information from the system and to tamper with the integrity of the information contained or used by the respective criticality domains. The availability is not directly affected by the given attacks. However, due to the change-of-scope (compare with Section IV-D), denial-of-service attacks might be conducted by further exploitation as well.

V. Memory-Map Shuffling

In this work, we propose a concept to increase the effort of localizing and predicting the attack target structure in the main memory. We aim to elaborate the certain aspects with regards to concept, implementation aspects and integration. Furthermore, we discuss aspects of the effectiveness of our approach. The obfuscation of address layouts is a method to increase the difficulty of exploiting vulnerabilities. Using this technique it is more difficult for an attacker to determine the location of memory structures. Address space obfuscation, which is often referred to as *Address Space Layout Randomization* (ASLR), was originally implemented for *user-space* applications. It added an artificial diversity of the memory locations of the applications *Stack*, *Heap* and linked libraries and positions within a process's address pace. Thus, the exploitation of buffer-overflow and format-string vulnerabilities became harder.

The core concept aims to create a random permutation of a translation table. However, beyond identifying a proper permutation procedure the concept builds on certain aspects relevant to the target environment. Architectural or technical constraints are relevant as well as procedural prerequisites. In Figure 4 an overview of the causal dependencies of the concept is shown. Usually, the AMP system configuration consists of several configurations to produce the IPA system mapping. These configurations include a board support package (BSP) which describes the SoC hardware element utilization, the

 ISBN: 978-1-61208-532-6

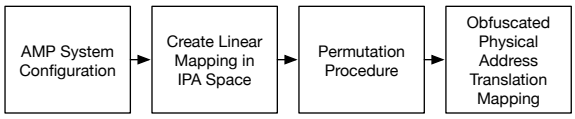

Figure 4. Overview of the obfuscation concept.

device assignments to split the peripherals, and the main memory partitions for the OS-layer instances. This information will be transferred into the suitable mapping table. This pre-initialized mapping will then be processed by a permutation procedure which outputs a random physical address map.

A. Obfuscation Concept

The concept aims to place the physical addresses in such a manner that without the knowledge of the mapping table an adversary cannot reconstruct the entire memory structure of a memory partition. That means, after the access control breach the attacker is able to jump to and access every position in main memory. Nevertheless, at PA level it cannot be differentiated to which MC-domain the memory pages belong. Furthermore, the order of pages is not sequential after the obfuscation. According to the ToE, Figure 5 visualizes the principle. To summarize up, the obfuscation takes effect in two dimensions. First, the page assignment and second, the sequential order of the pages. Our exemplary system consists only of two MC-domains. With a decreasing number of domains, the effect of the obfuscation rises.

B. Permutation Procedure

The core of the obfuscation concept is the algorithm to produce the permutation of the address mappings. The procedure of randomizing the address space can be compared to a shuffle of a deck of cards. Therefore, in order to transport the overall approach, we chose the shuffling algorithm by Fisher and Yates [24]. The Fisher-Yates shuffle is simple and fits well to produce random permutations of finite sets. In this particular case, the finite set is the translation table which was previously created by the initialization process. The translation table is denoted as a finite set TT of mapping entries E.

$$TT = \{E_1, E_2, ..., E_n\} \quad (1)$$

Each entry redirects an intermediate input address to its corresponding output address range. We assume TT is initialized with an identity mapping, which means each intermediate address is equal to the physical address *IPA = PA*. The entries of the set would then be arranged as follows:

$$TT = \{PA_{0x000000001}, PA_{0x0000000040}, ..., PA_n\} \quad (2)$$

The Fisher-Yates algorithm is shown in Algorithm 1. It iterates through TT and swaps the entry in the current position with a random position. The random position is determined by a randomization function which draws values out of a specified range.

Algorithm 1 Memory-Map Table Shuffle

for all TT[] **do**
  $random \Leftarrow$ random number such that $0 \leq random \leq range$
  swap TT[$random$] and TT[$current$]
end for

C. Assumptions and Requirements

One of the key elements of the shuffling algorithm or the permutation algorithm is a suitable random number generator. As shown in Listing 1 a discrete random number from a specified range ($0 \leq randomNumber \leq Range$) is drawn. As a prerequisite, we assume a cryptographic secure random number generation providing sufficient entropy. The required entropy depends on the granularity of page mappings of the system. In other words, the total number of entries in TT. Furthermore, we require that the generated numbers are still non-biased after truncating them to the specified range. Performance plays an important role since this approach will be integrated into a timing critical environment. Every time the system is reset the memory mapping will be randomized. Therefore, the algorithmic complexity must be kept to a minimum so the start-up phase of the device is not significantly delayed.

VI. Discussion

With respect to the characteristics given in the threat and attack analysis, this section discusses the effectiveness of the given approach.

A. Effect on the Attack Complexity

By applying the random permutation at the intermediate physical address mappings, the physical memory structure is obfuscated. Exploits like those referenced in the threat scenario would fail. However, adversaries would adapt to the newly introduced circumstances and try to de-obfuscate the memory map. In reference to crypto analysis, nevertheless, a reasonable way to evaluate the effectiveness of this kind of statistical security control is to estimate the effort to break it. In general, we assume two approaches to compromise a permuted address mapping. Either the attacker scans the whole main memory for a page he is looking for or applies statistical analysis to the permutation procedure. The former approach makes it necessary to assume that the attacker is able to scan the whole main memory. Furthermore, he needs an evaluation function that determines whether or not the current scanned page is the one he was looking for. This is what we also described in our threat scenario, however, the adversary now has to deal with the fragmentation of binary patterns. By applying this brute-force attack, the attacker needs to scan half of all left pages on average to find the next designated page. In other security fields such as cryptography, the strength of a certain function, such as encryption, is hard to define using discrete methods. Statistical analysis or complexity estimations on the randomization output forms the second approach to de-obfuscate the mapping table. This mathematical problem is comparable to the crypto-analysis of ciphertext. Concepts like *known-chiphertext* and *chosen-plaintext* attacks can reveal algebraic weaknesses of the implemented algorithms. Hence, the cryptographic secure implementation of those procedures is the key to preventing

 ISBN: 978-1-61208-532-6

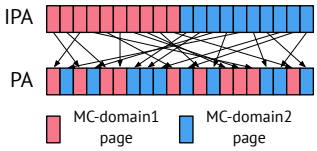

Figure 5. Principle of randomized memory assignment.

such information leakage. The proposed concept implies an in-depth target-specific reconnaissance in order to de-obfuscate the system mappings. Since the mappings are randomized on each system individually and change over time, given exploits are not directly applicable to a range of systems.

B. Change of Scope

The proposed approach does not influence the ability to change the scope (such as elevation of privileges) of the target. Once the adversary succeeds in overcoming the complexity of the obfuscation, the scope change is still possible.

C. Protection of Confidential Information

Protecting the integrity or even the control flow integrity in breached environments is the major target of the given approach. Nevertheless, the disclosure of information is also possible. Although, the adversary has increased effort to find the targeted information. Reading the data then does not further impact the system. In other words, the de-obfuscation could be done offline with increased resources and without interfering with the system. As a result, the rearrangement of data is not sufficient to protect from information disclosure.

VII. Conclusion

The evolving nature of automotive origination processes such as product-lines and product life-cycles imply special needs of the created products. This is particularly true for the analysis and definition of security mitigations. In this article, we proposed a concept to mitigate the effects of breaching the hardware memory protections of automotive mixed-criticality systems. The particular technique to implement the multiple compartments for the distinct criticality domains is asymmetric multiprocessing. This technique implies a static system configuration on runtime. In the threat and attack analysis, we identified that this can be misused to mount direct-memory-access-based attack vectors. The proposed mitigation approach aims to obfuscate the intermediate address mapping based on the introduction of random permutations of a normal, continuous memory-map arrangement. This adds complexity and raises the exploitation effort for adversaries. This approach is applicable to hardware architectures utilizing memory-maps and two-staged memory management. As such, it contributes to a through defense-in-depth security architecture applied to the automotive landscape.

References

[1] D. Spaar. (2015, Feb.) Beemer, open thyself! - security vulnerabilities in bmw's connecteddrive. [Online]. Available: http://www.heise.de/ct/artikel/Beemer-Open-Thyself-Security-vulnerabilities-in-BMW-s-ConnectedDrive-2540957.html

[2] A. Greenberg. (2015, Jul.) Hackers remotely kill a jeep on the highway—with me in it. [Online]. Available: https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway

[3] S. International, "Sae j3061, cybersecurity guidebook for cyber-physical vehicle systems," SAE International, 2016.

[4] O. Henniger, L. Apvrille, A. Fuchs, Y. Roudier, A. Ruddle, and B. Weyl, "Security requirements for automotive on-board networks," in *9th International Conference on ITS Telecommunications (ITST)*. IEEE, 2009, pp. 641–646.

[5] S. Baruah, H. Li, and L. Stougie, "Towards the design of certifiable mixed-criticality systems," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2010 16th IEEE*, April 2010, pp. 13–22.

[6] S. Hanford, "Common Vulnerability Scoring System v3.0: Specification Document," pp. 1–21, Jul. 2015.

[7] J. Freund and J. Jones, *Measuring and Managing Information Risk: A FAIR Approach*. Butterworth-Heinemann, 2014.

[8] J. Edge, "Kernel address space layout randomization," *Linux Security Summit*, Oct. 2013. [Online]. Available: http://lwn.net/Articles/569635/

[9] S. Bhatkar, D. C. DuVarney, and R. Sekar, "Address Obfuscation: An Efficient Approach to Combat a Broad Range of Memory Error Exploits." *USENIX Security*, 2003.

[10] C. Kil, J. Jim, C. Bookholt, J. Xu, and P. Ning, "Address Space Layout Permutation (ASLP): Towards Fine-Grained Randomization of Commodity Software," *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)*, pp. 339–348, Dec. 2006.

[11] H. Y. Tian Shuo and D. Baozeng, "Prevent Kernel Return-Oriented Programming Attacks Using Hardware Virtualization," *LNCS 7232*, pp. 1–12, Mar. 2012.

[12] M. Rushanan and S. Checkoway, "Run-DMA." *9th USENIX - Workshop on offensive technologies (WOOT)*, 2015.

[13] ISO/IEC, "ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models," Tech. Rep., 2010.

[14] Y. Kinebuchi, T. Morita, K. Makijima, M. Sugaya, and T. Nakajima, "Constructing a Multi-OS Platform with Minimal Engineering Cost." *IESS*, vol. 3, no. Chapter 18, pp. 195–, 2009.

[15] J. Porquet, C. Schwarz, and A. Greiner, "Multi-compartment: a new architecture for secure co-hosting on SoC," *System-on-Chip*, pp. 124–127, 2009.

[16] S. Hernan, S. Lambert, T. Ostwald, and A. Shostack, "Uncover security design flaws using the STRIDE approach," *http://msdn.microsoft.com/en-us/magazine/cc163519.aspx*, 2010.

[17] A. Shostack, *Threat modeling: Designing for security*. John Wiley and Sons, 2014.

[18] S. Checkoway, D. McCoy, B. Kantor, D. Anderson *et al.*, "Comprehensive experimental analyses of automotive attack surfaces." in *USENIX Security Symposium*. San Francisco, 2011.

[19] H. Chen, Y. Mao, X. Wang, D. Zhou, N. Zeldovich, and M. F. Kaashoek, "Linux kernel vulnerabilities: State-of-the-art defenses and open problems," in *Proceedings of the Second Asia-Pacific Workshop on Systems*. ACM, 2011, p. 5.

[20] P. Schnarz, J. Wietzke, and I. Stengel, "Towards attacks on restricted memory areas through co-processors in embedded multi-os environments via malicious firmware injection," in *Proceedings of the First Workshop on Cryptography and Security in Computing Systems*. ACM, 2014, pp. 25–30.

[21] P. Schnarz, J. Wietzke, and I. Stengel, "Co-processor aided attack on embedded multi-os environments," in *International Conference on IT Convergence and Security (ICITCS)*. IEEE, 2013, pp. 1–4.

[22] J. Danisevskis, M. Piekarska, and J.-P. Seifert, "Dark side of the shader: Mobile gpu-aided malware delivery," in *Information Security and Cryptology–ICISC 2013*. Springer, 2014, pp. 483–495.

[23] A. Case, L. Marziale, and G. G. Richard, "Dynamic recreation of kernel data structures for live forensics," *Digital Investigation*, vol. 7, pp. S32–S40, 2010.

[24] R. A. Fisher, F. Yates *et al.*, "Statistical tables for biological, agricultural and medical research." *Statistical tables for biological, agricultural and medical research.*, no. Ed. 3., p. 90pp, 1949.