Design Space Exploration Applied to Security

Antoine Linarès, David Hely, F. Lhermet, Giorgio Di Natale

Design Space Exploration Applied to Security

Antoine Linarès[1,2,3], David Hely[2], Frank Lhermet[1], Giorgio Di Natale[3]
[1]SiFive France, 13600 La Ciotat, France
[2]Univ. Grenoble Alpes, CNRS, Grenoble INP*, LCIS, 26000 Valence, France
[3]Univ. Grenoble Alpes, CNRS, Grenoble INP*, TIMA, 38000 Grenoble, France

***Abstract*—Software Hardening against memory safety exploits can be achieved from the silicon, up to the software, with both compilers and operating systems features. Unfortunately, due to the growing evolution of attacks, security architects have no guarantees, at an early stage of the development, that defenses will match the security needs and overcome the targeted threats. In addition, after product release, it is difficult to evaluate the architecture performance against new threats. This paper presents a dynamic analysis technique that allows the evaluation of the security profile of a given architecture during design exploration. The method is designed to highlight and quantify the security threats covered by the countermeasures embedded at any level of a given architecture. The provided results will help for protection evaluation, classification, and architecture choices. The method comes with a tool that implements this approach and has been applied to several architectures. This tool helps to classify architecture along with its alternatives thanks to metrics.**

I. Introduction

Since the beginning of this century, we have seen an unprecedented expansion in technologies linked to digital devices. They deal with sensitive data (either in terms of privacy or security), from bank information stored in credit cards to family pictures on smartphones. As a result, we have experienced an outstanding growth of cyber-attacks. Security architects have thus to address a very large set of vulnerabilities by selecting the more efficient security countermeasures among the large set of existing and future ones. For instance, the Common Weakness Enumeration (CWE), the community project in charge of listing and sorting all known vulnerabilities and weaknesses for the US National Cybersecurity effort, has listed, over 760 different vulnerabilities [1]. One memory vulnerability example is the CWE-226: *"Sensitive Information in Resource Not Removed Before Reuse"*. Such a threat source can be either software or hardware and can be mitigated by many different hardware or software techniques. As a result, without effective security-oriented design tools, addressing these vulnerabilities requires a huge effort for engineers.

Therefore, there is a need for automatic or assisted security evaluation for mixed hardware-software architectures.

We present in this paper an approach that leverages the standard criteria commonly used to evaluate security threats and mitigation. Indeed, several works have tried to address the challenge to create universal relations between attacks and defenses such as the approach proposed by the CWE or the one described in [2] which details a systematization of knowledge. They highlight that attacks and protections are divided into stages. A stage is, depending on the point of view (attack or defense), either to control or to secure. An attack cannot successfully be performed if a single step is missed. The approach detailed in our paper aims at identifying which step should be covered first, and how much it will impact the product security.

Different security metrics have been used to evaluate the security capabilities of a given countermeasure. In [3], the gadget coverage is used as a global metric. In [4], the authors have evaluated the security performances applying well suited metrics. The challenge is then to keep track of the evolution according to attackers' new capabilities. Another way to measure security is to take the opposite approach and not to measure security strength but instead to keep track of attacks' evolution. The National Vulnerability Database (NVD) proposed a Scoring System (CVSS). This metric is assigned to every added vulnerability in the US vulnerability database. The size of this database makes it a reference to describe the current security threats. That's why statistics from this database are used as an entry in [5].

The approaches listed above provide important references for security architects, however, even if they provide insights when considering an attack or a group of attacks, such listings do not allow a systematic approach to explore the most suited countermeasures for a given set of threats.

Our objective is then to provide a high-level, dynamic, and easy to update security evaluation tool for memory safety. The following example further explains our motivation.

Considering an architect looking for the proper security solution. His overview of the attacks is done. He has to prevent a significant amount of threats. Thus, a set of protection mechanisms is needed to tackle most of them. Nowadays, his choice will be assisted with few metrics and, may perish briefly without notice. That is why one needs a dynamic tool that provides a representation of the security provided by a set of security mechanisms. This output shall be associated with reliable metrics to help to make decisions, and to track if the security choices remain relevant along the time.

The contributions of this paper are the following:

- We develop a description methodology for security techniques that are generic enough to be performed on most systems.
- We provide a mathematical approximation for the security capability for systems made of several protection techniques.
- We apply the two methods to a set of security techniques.
- We develop a tool that can apply these concepts, with assistance tools like graphs, sorting capability, and dynamic updates capabilities.
- The presented tool also provides to the user different security alternatives, enabling a security-oriented design space exploration.

This paper is organized as follow: First, the environment and the background of the study are presented in section II. Then, an accumulation approach (section III), and a tool for dynamic evaluation based on this approach (section IV) is introduced. Both the approach and the tool are illustrated with a real security system example in section V. Finally, the perspectives of this work are discussed in section VI before to draw some conclusions in section VII.

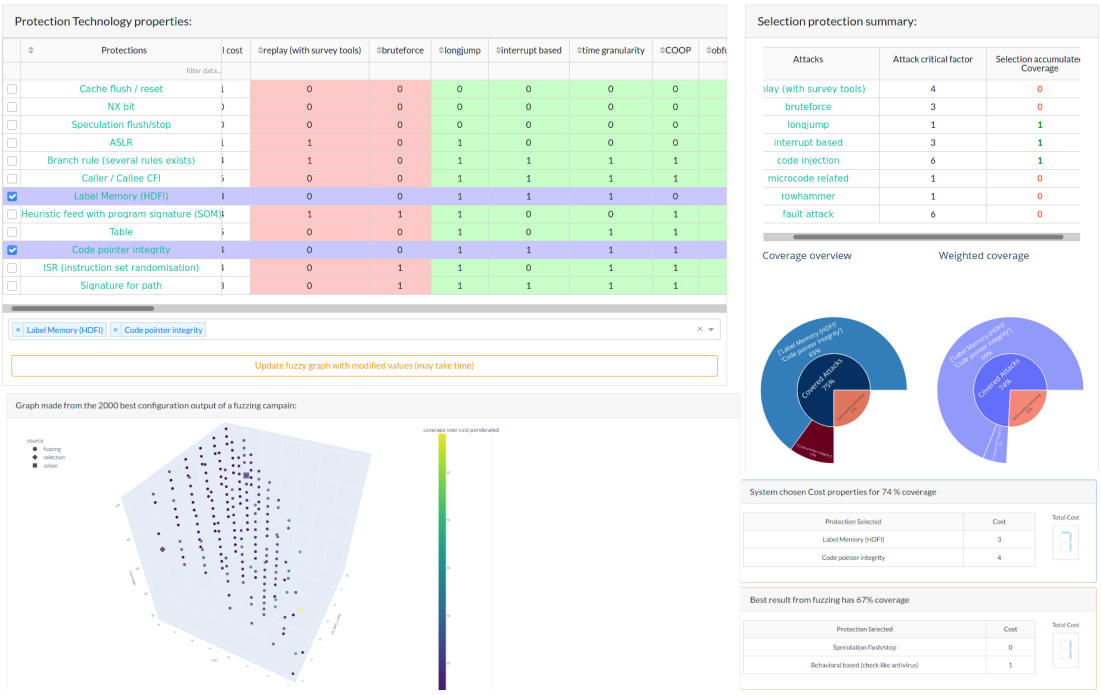

Fig. 1. Partial view of the online tool proposed to simulate and track security performances

II. Background

The problem we are trying to solve has a large set of variables (here the set of security protections existing in the literature that can be combined). A known approach that suits such type of problem is design space exploration.

Design Space Exploration is a technique that allows exploring large sets of solutions. A set of criteria is applied at the same time on a large subset or all the solutions. Then it is easier for the users to choose a suitable solution among a reduced number of elements. Tools, frameworks, or usages can be found as [6], [7].

Advanced mathematical concepts are not required for our solution, but the problem we are addressing is close to the mathematical concept of set covering. A set covering problem can be described as a system with a pool (P) of subset elements (P = $\{S_1, S_2, ..., S_n\}$) from a universe (Υ). The objective is with a minimum number of subsets ($\rho \subseteq P$) to cover all elements ($\epsilon \in \Upsilon$) in the universe as in equation (1).

$$\epsilon \in \bigcup_{S_x \in \rho} S_x \quad \forall \epsilon \in \Upsilon \tag{1}$$

This approach is close to our objective, as each protection can be modeled as the subset of all existing countermeasures and the universe as the set of known attacks. But, our problem is different from the classical set-covering because many architectures can accept a not fully covered universe, and this sub-universe to be covered might be weighted. Therefore, our problem is closer to the ”partial set covering“ approach described here [8]. If this mathematical approach may give a probabilistic best solution, our method is designed to be used as a very early stage of design where many assumptions are made. Indeed we are not confident on these assumptions' accuracy and even more on the results that should be done on top of them. Thus we decided not to get into this direction and provide to the user all the elements to challenge the tool outputs and find the solution.

One line description	attack type 1 coverage deduced	attack type 2 coverage deduced	...	attack type n coverage deduced
Executable codes are placed on random place	1	0	...	0
jump or call can only target ”landing pad”	0	1	...	0

TABLE I
Coverage attribution example.

Attacks and defenses are central as we decided to make a helping tool for security. The field of attack and defense is vast. We chose to reduce the scope to provide a usable and efficient tool that may be extended. We focus our work on memory corruption attacks, this large set of attacks is popular and we can find in the literature great classification efforts. Consequently, many security techniques and attacks exist at any level from silicon to the compiler, making it a perfect topic for this approach.

III. Methodology

We have tried to reduce memory protection techniques to a basic representation. We observe or derive a binary performance for each security technique considered against several types of attacks. This concept is illustrated in table I. To have the best accuracy possible and to provide more information for a system made of several of theses techniques the attack pool should contain at least for each studied protection, one avoided or detected exploit, and one that is not. It will be furthermore important to add any vulnerability that is added by systems (For example, some security implementations add side-channel to inspect the program at runtime, this channel may be accessible in an “evil maid scenario” as described in [9]).

The result is a table of properties that provide for every protection the type of attack covered as well as the uncovered ones. This tab is called “database” in the following of this paper.

It is now possible to compute an approximation for the coverage of any architecture that is made of inherited properties. This choice may not reflect a real security system as

some security techniques may not be used together without side effects. The mathematical representation is given by equation (2)[1].

$$\text{SystemAttackCoverage}_{\text{j}} = \bigcup_{\text{i=protectionChosen}}^{\text{protectionChosen}_{\text{nb}}} \text{database}_{\text{i,j}} \tag{2}$$

This evaluation is done for each attack type j in the database. The result is an approximation of the coverage provided by the system thanks to the security techniques it uses.

IV. The Tool

This paper introduced the feature accumulation approach, and rules to apply this approach to any system. This allows designing a tool for the exploration of large fields of techniques and attacks. We propose an implementation of this tool as illustrated in figure 1. The choices we made, discussed in the following section, can be changed either to match a different objective or to consider some threats or security techniques that might have been missed in our proposed solution.

The 24 protection techniques that have been chosen are a set of protections against memory-based attacks. This set of protections is meant to be a representation of the solutions that are available today. Some listed protections are Control Flow Integrity (CFI) techniques, these techniques are useful against control flow deviation and several memory corruptions. Other protections as cache flush are not designed against memory attacks, but they were added to provide a better representation of existing solutions. Attacks have been chosen to reflect as much as possible the capabilities and weaknesses of the proposed protections. Some attacks may appear to be out of the memory corruption scope, nevertheless, since these could be inhibited by some memory corruption countermeasures, it is worth adding them to our tool.

Thanks to the very low computation needed to get coverage with the accumulation approach, the tool we propose is capable of dynamic updates from both the user and the provider. The user can change coverage attributes of any technique and, dependent characteristics are refreshed dynamically. This feature is really important as the protection is complex to derive from system known properties, and may be subject to interpretation. Thus, in case a user disagrees with the proposed result, he can quickly identify which values are not properly set and how it impacts both the modeled environment and the results. Depending on his conclusion and perspectives, the user can further improve the model.

As proof of the dynamic capability from the provider perspective, in our tool, the attack critical score is updated daily with data from the NVD to reflect as much as possible the trends.

Our goal is to provide efficient discrimination between techniques. For this purpose, it is important to take into account costs and constraints. But this cost may vary depending on the user of the system, it may also vary depending on the target it is running on. To have a common approach with less bias and errors, the cost of each used security technique is defined thanks to different sub-costs that are easier to measure or extrapolate. Each of these sub-cost has a value in $\{0, 1, 2\}$ where 0 means no impact, 1 means an influence is visible and 2 means an important cost. The granularity is important but, the objective is to provide easily an estimation of the cost for every solution, with easy assumptions, even if the accuracy will not be perfect.

[1]Here U is a representation for OR operation performed over the array

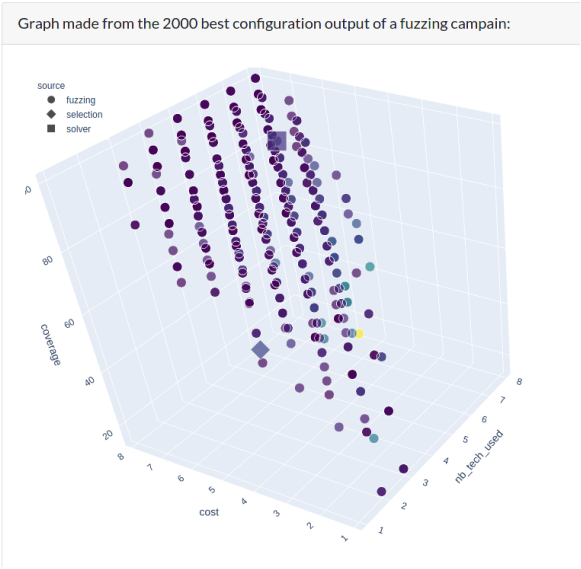

Fig. 2. 3D representation of user's result close to fuzzy campaign results provided by the tool.

The sub-costs considered are listed below:

- Runtime cost: indicates the consumption of computing time
- Memory cost: indicates the memory usage for security purpose
- Process cost: indicates the process complexity needed to benefit from the security technique (like rebuilding the code, extensive redesign, post compilation tool...)

To keep a very objective point of view, costs are computed by a very simple sum described in equation (3).

$$\text{cost} = \sum_{\text{i=1}}^{\text{subcost}_{\text{nb}}} \text{subcost}_{\text{i}} \tag{3}$$

The cost for a solution made of several security techniques' is also a sum of all the selected techniques global cost. Thanks to this, we can derive for any aggregation of techniques, a cost.

As shown beside, the proposed implementation is considering 24 techniques. So 2^{24} different macro-systems are resulting. The tool provides a graphic view displaying user propositions near thousands of macrosystems randomly generated. The user can see where his proposition is located in this set of solutions. This pool of solutions is derived from a database initially made of over 7 million solutions. To avoid an extensive load of memory (about 3GBytes) only 2000 best solutions are displayed (as in figure 2).

As our tool considers dozen of attacks and several of them are currently not widespread, a weight can be applied to the attacks. The user may decide not to consider some attacks or to consider another as highly critical. This choice as any other will be reflected in the different views and graphs and tabs dynamically provided by the tool. Other helping tools are provided as sorting but also link to reference paper for each vulnerability and defense alongside with filtering options, highlight and zoom options.

V. Application example

We challenged our solution with a threat model corresponding to the state-of-the-art protection proposed for laptops and personal computers. This threat model may remain the most popular for memory attacks in the following years. This may confirm that this system keeps track of the threat evolution as intended.

The tool aims at modeling the security environment of a running C program on top of Linux with an Intel processor including Intel's Control-flow Enforcement Technology (CET).

Intel CET is an aggregation of hardware security techniques that will be supported by Intel's 11th (and next) generation of microprocessors (released in September 2020). On top of this, an Operating System, here Linux provides its security. At last, a program is running. It has been designed with a programming language and some security practices, compiled by a compiler that provides support for the lower level protection, and add some more.

To represent the environment the following techniques have been chosen:

- For CET extension provided by Intel: Shadow Stack; Landing pad (ENDBRANCH instruction); Speculation protection (LFENCE instruction)
- For Linux Operating System: ASLR; NX Bit
- For GCC Compiler: StackGuard (Return address protection)

Our system provides several results that may be useful: Cost for the global solution, coverage of each attack, coverage with attack factors, the ratio of coverage versus cost, and it also indicates if multiples protection prevents the same attack.

One result provided by our tool is the list of uncovered attacks. This latter is useful for the evaluation of our tool as it could be compared to vulnerability disclosure.

Currently, uncovered attacks are: Brute-force; Interrupt based; Forward to gadget/payload (Payload or gadget starting with a landing pad); Data-oriented Programming; Rowhammer; Fault attacks; Control Flow Bending. It is important to note that this example architecture still protects against 75% of the attacks known by the tool. When we apply a weight for each attack to highlight the most widespread ones the weighted result is 78.8% coverage.

Even if Intel-CET-based system is not released, authors in [10] have tried to extrapolate the security that may be provided by this future architecture. Their conclusion is similar to ours and, Intel CET will soon confirm the prediction provided by our tool.

VI. Discussion & Perspectives

The tool provided as well as the methodology introduced may be enhanced with several mechanisms. First of all, each attack severity factor but also coverage may be updated with database harvesting as well as community inputs. A linked improvement is a way to share, add, correct, and dispute the default content of the database. Currently, anyone can update the content but changes are local and cannot be shared. Tracing daily or weekly attack evolution will make new features possible like alerts linked to attack trends or security level for any architecture.

Many improvements to make the proposed tool easier to use are needed for large-scale adoption. Also, we believe that the solver may be improved to suggest replacement or addition to the user-chosen system, as the 100% coverage is not suitable for all uses. Another important enhancement will be to add a mechanism that will grant a user to choose an existing product or device and the tool will choose the related protection accordingly.

In the future, we may imagine the presented approach and tool included in a framework for security. New highly configurable and open-source architectures (even on hardware as RISC-V Rocketchip [11]) combined with efficient tools made for continuous integration [12], may make it possible.

Thus any custom system designed with the assistance of our tool may be generated on a server-linked simulator or FPGA. A bundle of tests may be run on it to verify the accuracy of the predicted security performance but also the costs. Every result will be propagated into the database to increase its accuracy.

VII. Conclusion

This paper presents a new approach to help the architecture exploration of digital devices considering memory-related threats. The architecture is described as an accumulation of security techniques to evaluate global security against memory attacks. The developed tools associated with the method can provide a first overview of the threats being thwarted by the architecture under evaluation. If such a tool does not intend to provide a fine-tuned evaluation the first-order analysis helps the architects to better forecast the security of the system, drastically reducing the architecture choices. This automated approach finally aims at reducing the amount of different architectures that will need to be more fine-grained analyzed by security experts before a final release.

References

[1] MITRE - CWE, "CWE Top 25 Most Dangerous Software Errors." https://cwe.mitre.org/top25/archive/2019/2019_cwe_top25.html, 2019.

[2] L. Szekeres, M. Payer, Tao Wei, and D. Song, "SoK: Eternal War in Memory," (Berkeley, CA), pp. 48–62, IEEE, May 2013. http://ieeexplore.ieee.org/document/6547101/.

[3] L. Davi, M. Hanreich, D. Paul, A.-R. Sadeghi, P. Koeberl, D. Sullivan, O. Arias, and Y. Jin, "HAFIX: Hardware-Assisted Flow Integrity eXtension," pp. 1–6, June 2015. http://doi.org/10.1145/2744769.2744847.

[4] R. de Clercq and I. Verbauwhede, "A survey of Hardware-based Control Flow Integrity (CFI)," *arXiv:1706.07257 [cs]*, July 2017. http://arxiv.org/abs/1706.07257.

[5] L. Gressl, A. Rech, C. Steger, A. Sinnhofer, and R. Weissnegger, "A Design Exploration Framework for Secure IoT-Systems," (Dublin, Ireland), pp. 1–8, IEEE, June 2020. https://ieeexplore.ieee.org/document/9139631/.

[6] E. Kang, E. Jackson, and W. Schulte, "An Approach for Effective Design Space Exploration," Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 33–54, Springer, 2011.

[7] A. D. Pimentel, "Exploring Exploration: A Tutorial Introduction to Embedded Systems Design Space Exploration," *IEEE Design Test*, vol. 34, pp. 77–90, Feb. 2017.

[8] C. Chekuri, K. Quanrud, and Z. Zhang, "On Approximating Partial Set Cover and Generalizations," *arXiv:1907.04413 [cs]*, July 2019. http://arxiv.org/abs/1907.04413.

[9] A. Tereshkin, "Evil maid goes after PGP whole disk encryption," SIN '10, (Taganrog, Rostov-on-Don, Russian Federation), p. 2, Association for Computing Machinery, Sept. 2010. https://doi.org/10.1145/1854099.1854103.

[10] PaX Team, "On the Effectiveness of Intel's CET Against Code Reuse Attacks." https://grsecurity.net/effectiveness_of_intel_cet_against_code_reuse_attacks, June 2016.

[11] Berkeley Architecture Research, "Chipyard's documentation." https://chipyard.readthedocs.io/en/latest/index.html, 2019.

[12] M. Hilton, T. Tunnell, K. Huang, D. Marinov, and D. Dig, "Usage, costs, and benefits of continuous integration in open-source projects," ASE 2016, (New York, NY, USA), pp. 426–437, Association for Computing Machinery, Aug. 2016. https://doi.org/10.1145/2970276.2970358.