NB Parts of this proposal have intentionally been omitted to make it more suitable for this Research Proposal Writing class; these parts are indicated as "OMITTED ON PURPOSE" below.

Dutch Research Council (NWO)

NWO Domain
Applied and Engineering Sciences (TTW)

Open Technology Programme 2023

Project Title:

P6: Prioritization for Prompt Patching of Programs with Pernicious Problems

Table of contents

NWO

1. Contact details OMITTED ON PURPOSE

1.1 Main applicant

:

1.3 Keywords

Patch prioritization; Automated patching; Vulnerability analysis; Risk assessment; Software vulnerabilities.

2. Summary

2.1 Scientific Summary

Software vulnerabilities have devastating consequences on our digital society. In the past few years, the research community proposed sophisticated approaches to enhance automated security testing and identify

vulnerabilities before they can be exploited by attackers. As a result, many tools exist to effectively detect vulnerabilities. However, most organizations do not use such security testing tools because of their limitations. In particular, existing tools raise too many alerts, highlighting more suspected bugs than organizations can fix in a timely manner, and do not provide detailed information about the discovered vulnerabilities, requiring expert analysts to investigate their causes and implications. In fact, detecting bugs is only the first step. To secure software, we must be able to reason about the identified vulnerabilities to generate timely patches with minimal human effort. In this project, we will develop a novel automated framework to efficiently characterize software vulnerabilities, reason about the capabilities they provide, model their risk, and subsequently generate patches.

In the real world, attackers put together multiple vulnerabilities to compromise systems. Compared to existing work, we present a paradigm shift: our risk analysis does not consider vulnerabilities in isolation, but can reason about how such vulnerabilities can be combined and how they are constrained by modern mitigations. Such reasoning requires high computational power, and current techniques do not scale to complex real-world software. As such, they cannot analyze all vulnerabilities together, which is critical for developers to plug every hole in their software. We propose new techniques that can offer holistic vulnerability analysis at scale.

Addressing the aforementioned scalability challenges, this project represents a fundamental step to obtain a realistic "big picture" of the implications of vulnerabilities. From a scientific perspective, we will provide the first holistic method to model and reason about the impact of combined vulnerabilities against mitigations, allowing future research to use a reliable and realistic reference for vulnerability assessments. From a practical perspective, the project will provide a cohesive framework that organizations can embed into their testing lifecycle to obtain an accurate and comprehensive view of the vulnerabilities affecting their IT infrastructure, evaluate their risks, and generate cost-effective patches. Leveraging such a framework, end users will reduce the costs of vulnerability management and patching, while minimizing the risks of being compromised.

2.2 Public Summary NWO domain TTW's website and online in ISAAC

Software used in everyday life is vulnerable to attacks from cybercriminals. Researchers and companies adopt techniques to discover vulnerabilities in production software and fix them. However, current tools detect more potential flaws than organizations can fix, leaving services still highly vulnerable. In this project, we design and develop automated techniques to analyze discovered vulnerabilities, assess their risk, prioritize the critical ones, and generate patches. Unlike prior work, we consider vulnerabilities in their context, including interactions between vulnerabilities and defenses, allowing for prompt mitigation and reducing costs.

3. Problem Statement

Software vulnerabilities drastically impact our society — the average cost of a data breach reached a record high of $4.35 million in 2022 [IBM22]. Just a small programming mistake can threaten our privacy, finances, and critical infrastructures. In the past few years, researchers and security companies have developed solutions to integrate security testing in the development process and, thus, to promptly identify bugs in production software. These approaches are immensely successful. For example, Syzbot[1] fuzzes the Linux kernel 24/7 and has found thousands of bugs. Some, but hardly all, of these bugs have the potential to completely compromise a system.

In fact, bug finding techniques have arguably become *too* successful. We are finding more bugs than we can fix in a timely manner. This leads to *alert fatigue*, a situation where security operators cannot stay on top of the large number of alerts produced by potential bugs [*Sma23]. For instance, Google's fuzzer OSS-Fuzz uncovered over 25.000 bugs in 300+ open-source applications, in just three years [Goo21]. As a matter of fact, despite their sophistication and maturity, the majority of IT organizations do not adopt existing fuzzing tools in their production pipeline[2] — except for big corporations such as Google, Meta, and Microsoft. In fact, only a relatively small share of bugs discovered by fuzzers has relevant security implications, while the process from identifying and analyzing bugs to deploying patches remains laborious, expensive, and in desperate need for

[1] https://syzkaller.appspot.com/
[2] https://www.code-intelligence.com/about-us.

automation. Thus, we urgently need a way to prioritize fixing the most harmful bugs, reducing costs and manual effort for organizations, and minimizing the time window in which users are exposed to devastating cyberattacks.

To be able to prioritize bugs, ideally we need to know the worst possible attacks an attacker can carry out in a given program, we need to reason about their *effects*, i.e., the capabilities that vulnerabilities enable when exploited, and assess their severity (i.e., security implications). While fuzz testing to discover bugs is commercially available and widely used, vulnerability analysis and automatic exploit generation is comparatively in its infancy. Some existing academic work has proposed various steps in an exploitation pipeline, but for bug prioritization we need a robust end-to-end solution that can be applied even in complex programs. In particular, while prior work explored the design of solutions to rank vulnerabilities, existing techniques do not tackle a critical aspect: the main challenge for an effective vulnerability analysis and prioritization strategy is considering the impact of *multiple vulnerabilities*, the capabilities that they enable *when combined,* and the *constraints* (i.e., defenses) the attackers are subject to when attempting to compromise a target. In fact, in real-world settings, attackers put together ("chain") multiple vulnerabilities to successfully compromise systems. Indeed, in the current "info leak" era, this is essentially a given in any real-world exploit [Ser12]. For instance, the combination of an info leak capability together with an arbitrary write on the stack is often required for full compromise. However, such a combination is not enough for successful exploitation when certain protections, such as shadow stacks [Bur19], are deployed. Thus, ranking vulnerabilities individually and without modeling the effects of defenses is insufficient. In addition, to understand the full origin of bugs, we first need to identify their *root cause*: the first incorrect operation or missing check, setting in motion a chain of invariants being broken that eventually results in a crash. Unfortunately, existing program analysis techniques for vulnerability characterization (e.g., symbolic execution) suffer from significant scalability issues, which prevent them from successfully being applied to complex, real-world software - let alone when the aim is to analyze the combination of multiple vulnerabilities and capabilities under the constraints of existing defenses. These are critical open problems if we aim to automate vulnerability prioritization and patching. With the support of our industrial partners and users, this project will address these problems, both reducing costs and improving security in real-world systems.

4. Scientific description

4.1 Research contents

Research into automatic identification of vulnerabilities before they can be exploited by attackers has led to a wide range of very effective commercial tools and services. However, identifying vulnerabilities is only the first step to secure software and prevent attacks. A subsequent, fundamental step lies in reasoning about the identified vulnerabilities to understand their root causes, consequences, and risks, and thus to develop and prioritize suitable patches. In fact, automated bug finding techniques identify a large amount of potential vulnerabilities, while only a relatively small share of the reported bugs have serious security implications. As triaging, root cause analysis, risk analysis, and patching of discovered bugs are usually conducted by humans, this is an expensive, time-consuming, and error-prone process, causing severe vulnerabilities to remain unpatched for a long time and leading to high costs [*Sma23]. Recent work explored designing automated solutions to characterize vulnerabilities according to their root cause and severity [Bla20,Yag21a]. Unfortunately, such approaches do not provide an accurate characterization of the vulnerabilities in terms of what capabilities they enable for attackers. First, existing tools suffer from scalability issues that prevent them from successfully dealing with complex software or deep vulnerabilities. Second, they characterize only a limited set of primitives (e.g., control-flow hijacking primitives) and do not consider and model modern mitigation techniques. Third, and most importantly, existing work does not study a critical aspect, that is *how multiple vulnerabilities and capabilities can be combined*. In fact, when taken individually, certain vulnerabilities do not provide highly dangerous capabilities. However, when combined, even low/mid-severity vulnerabilities can result in high-severity consequences. As a matter of fact, in the real world, attackers put together ("chain") multiple vulnerabilities to compromise systems. Thus, approaches that study vulnerabilities individually, and without considering the effects of defenses, are insufficient as they rely on an unrealistic threat model.

Objectives, originality, and scientifically innovative elements. In this project, we will design and develop a novel, comprehensive, and automated framework to efficiently characterize software vulnerabilities and the capabilities they provide, reason how such capabilities could be combined and how deployed defenses constrain them, and subsequently generate proper patches. Our project proposes a *paradigm shift*: we perform risk analysis of identified vulnerabilities only after reasoning about how vulnerabilities can be chained, even considering modern mitigations and arbitrary exploitation techniques, a crucial step to obtain a realistic "big picture". Existing solutions cannot support these capabilities and extending their scope would further exacerbate their scalability shortcomings on real-world software. The techniques developed in this project will finally provide a holistic view of the risks imposed by vulnerable systems. This allows understanding the security consequences of vulnerabilities, which is fundamental to apply effective countermeasures and enable prioritized patching.

Our outcomes will lay the groundwork for a smart vulnerability management system, which organizations can embed into their testing lifecycle to obtain a comprehensive view of the vulnerabilities affecting their infrastructure, evaluate their risk, and generate cost-effective patches. By leveraging this system, end users will largely reduce the costs of vulnerability management and patching, minimizing the risks of being compromised.

Overview and design
This project proposes a new methodology and technology to analyze software vulnerabilities, reason about their combination, and prioritize patches. Our methodology takes as input a program and a set of crash-inducing inputs, and produces as output (1) a human-readable model of the capabilities enabled by the vulnerabilities, (2) an effective assessment of their severity, (3) an efficient patch prioritization strategy, and (4) a patched program.

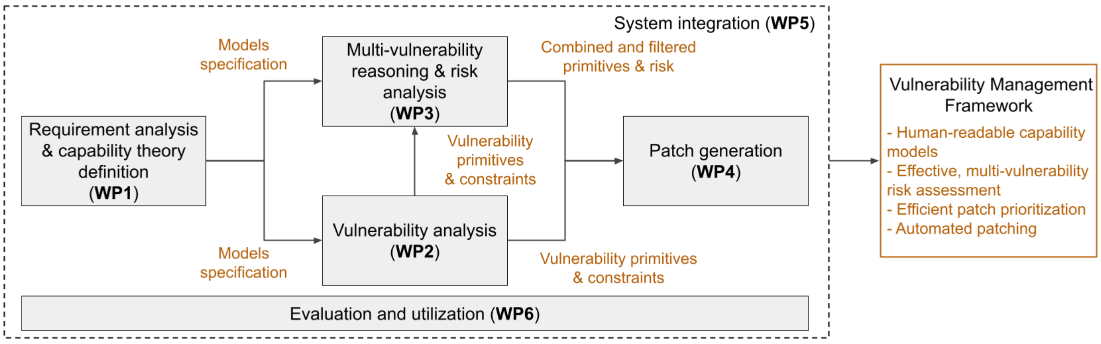
System integration (WP5)
Models specification
Multi-vulnerability reasoning & risk analysis (WP3)
Combined and filtered primitives & risk
Requirement analysis & capability theory definition (WP1)
Vulnerability primitives & constraints
Patch generation (WP4)
Vulnerability analysis (WP2)
Models specification
Vulnerability primitives & constraints
Evaluation and utilization (WP6)
Vulnerability Management Framework
- Human-readable capability models
- Effective, multi-vulnerability risk assessment
- Efficient patch prioritization
- Automated patching

***Figure 1**. Overview of the proposed approach and outcomes.*

Methodology and techniques
To achieve our goals, we divide the proposed project in six work packages (Figure 1). The following sections describe the methodology that we will design in each work package and the challenges that we will address.

Requirement analysis and capability theory definition (WP1). The first objective is to define our framework to reason about the capabilities that vulnerabilities enable, beyond the coarse-grained attack modeling of traditional requires/provides analysis [Tem01]. Similar to the well-known Satisfiability Modulo Theory, we propose a *capability theory*, which defines sets of clauses and rules to evaluate them. This novel framework will enable vulnerability analyses to reason about the control that attackers exert over a process, and it is represented in a human-readable language, allowing analysts to easily understand the capabilities enabled by vulnerabilities and their risks without having to understand the details of the vulnerabilities in question. Based on our prior work [*Red20,*Oli22], we symbolically model exploitation capabilities and their interactions. As an output, this WP provides models of the capabilities, or *primitives*, that vulnerabilities enable, together with models of mitigations and constraints on their interactions, which guide our research in WP2 and WP3.

Vulnerability analysis (WP2). We will design and develop automated techniques that analyze and trace individual vulnerabilities to extract primitives. Given a bug-inducing input (e.g., crashing input produced by a fuzzer), we determine the capabilities (defined in our framework) enabled by the vulnerability. Our insight to do so is to first trace the concrete execution of the target program under the bug-inducing input, and then, to re-trace the execution symbolically to reason about the root cause of the bug and its capabilities. To this aim, we address three major challenges. First, when applied to complex, real-world programs, emulation-based (e.g., QEMU) tracers and symbolic engines are too slow and limited by a set of supported instructions and system calls. To overcome this, we leverage hardware and OS features to enable record/replay of processes without code instrumentation (e.g., extending tools such as rr [Moz22,OCa17]). Second, symbolic execution does not scale when applied to complex software. To address current scalability issues, we interleave symbolic with concrete execution, extending our previous work [*Gri20], we bound the scope of symbolic execution with efficient dynamic taint analysis [*Vee17], and we prioritize vulnerable execution paths [*Rua21]. Finally, vulnerabilities can be reached from different execution paths, each enabling different capabilities. To explore alternative paths, we use directed fuzzing [Böh17] toward vulnerable states and collect traces of different paths. As output, this WP provides techniques to extract primitives from vulnerabilities, which we further combine and filter in WP3.

Multi-vulnerability reasoning and risk analysis (WP3). Repurposing, refining and, most importantly, *automating* requires/provides analysis [Tem01], we will design techniques to reason about how different vulnerabilities can be chained, and how their combination can enhance attackers' capabilities. To do so, we leverage the capabilities modeled in WP1 and the primitives extracted in WP2. SMT solvers include rules and simplification processes to reduce the complexity of constraints in constraint sets. The proposed capability theory will include an analogous, but inverse, process: capability augmentation. Some of the clauses in our framework represent more complex ("higher") capabilities that might require multiple "lower" capabilities to be achieved. The capability augmentation will combine certain equivalent combinations of lower capabilities into their higher equivalents, or vice-versa, taking into account the constraints imposed by mitigations. For instance, the combination of an info leak capability with an arbitrary write primitive on the stack can lead to arbitrary code execution. However, the combination of these primitives does not imply code execution if protections such as shadow stacks [Bur19] are in place. Thus, in this phase we will research techniques to reason about the constraints required for primitive to be combined and their interaction with deployed defenses. In addition, together with software vulnerabilities, we will consider their microarchitectural variants (e.g., memory safety vulnerabilities triggered during transient execution) [*Joh22]. Once we identify a set of combined and filtered primitives, we can define metrics to assess their risk. Finally, we will design techniques to identify the primitives that are essential for a working chain. To do so, we will introduce a Vulnerability Dependency Graph comprising a node for each primitive with edges indicating there is a way for a primitive to influence, or to be chained with, another one. Using such a graph, we can identify primitives that should be prioritized for patching (e.g., using the graph centrality metrics) in WP4.

Patch generation (WP4). Leveraging the knowledge obtained by analyzing the vulnerabilities (WP2), the resulting primitives (WP2-3), and their security impact (WP3), we can now generate patches to mitigate risk. Our aims are to ensure that a patch (1) prevents the vulnerabilities from being exploited, (2) preserves the semantics of the programs, and (3) has minimal impact on runtime performance and memory usage. To this end, rather than generating patches that eradicate the underlying causes of vulnerabilities, we rely on our analysis to generate patches that cripple the primitives at the attacker's disposal with minimal impact on performance. In other words, rather than generating a patch to stop, say, a buffer overflow (which requires either manual effort or expensive security checks), our goal is to generate a patch that stops dangerous primitives such as arbitrary memory reads/writes an attacker can build on top of the original buffer overflow. Automatically generating patches to cripple such primitives (i.e., automated primitive patching) is easier (no need for expensive sanitizer-style metadata management) and more efficient (no need to patch other less harmful side effects such as DoS of the original vulnerability) compared to the more ambitious goal of automated vulnerability patching. For instance, to cripple an arbitrary memory write, we can efficiently mask the address at the vulnerable write location to forbid accesses outside the intended "safe" region. We plan to explore different mechanisms to define such regions. An option is to isolate sensitive objects in a single "sensitive" region of memory and forbid accesses from patched locations. Another option is to rely on type-based allocators to group objects of the same type and forbid patched locations from accessing regions that belong to objects of other types. In both cases, the address masking operation is efficient (requiring no or quickly accessible allocator metadata). As such, we prevent exploitation without changing the semantics, with minimal overhead, providing a production-ready solution.

System engineering and integration (WP5). To ensure our project has the desired societal impact, in this work package we integrate the prototypes from WP2-4 into a cohesive system. We design and document interfaces between the components, using standard open-source practices, and engineer the parts into a cohesive framework. During the development process, we will actively involve our industry partners to receive feedback and ensure that the development framework satisfies the industry common practices. We will open source and promote this framework with stakeholders to ensure effective knowledge transfer. Besides, we will integrate our framework into TNO's Cyber Reasoning System (CRS), which will provide a reference platform for activities regarding Automated Vulnerability Research (AVR) in the Netherlands[3]. Finally, we will release a publicly available, web-based application for performing our automated analyses on vulnerability characterization.

Evaluation and utilization (WP6). We will evaluate each individual work package and our cohesive framework on real-world vulnerable programs. First, we will use existing datasets of real-world programs [Mu18] that include multiple and diverse vulnerabilities (e.g., buffer overflows, use-after-free, uninitialized reads). Second, we will crawl data about the latest and upcoming CVEs. Finally, we will hire a PDEng (Professional Doctorate in Engineering) student to conduct the system engineering, integration, and validation, working with TNO and our industry partners to validate our techniques within their testing environments. As discussed in Section 6, our partners, especially Secura and Northwave, will provide us domain-specific advice and data to meet the expected TRL. To foster utilization of our research, we will organize a public release event and a technical workshop on the application and usage of our framework, inviting relevant research, industry, and government organizations.

Feasibility. For each WP, prior work that inspired this proposal exists, as discussed previously and in Section 5. We build upon existing research and focus on producing solid and usable tools. At the same time, this project takes a *novel* integrated view, connecting strands of previous research to build the first holistic method to reason about the combination of multiple vulnerabilities and assess the severity of the capabilities that they enable, and making it scale to real-world programs. Our outcomes are both theoretical, offering deep insights into modeling the impact of vulnerabilities, and practical, engineering the technology to automatically prioritize patching.

Risks and contingency plans. The proposed research requires a unique composition of expertise and operational capabilities. The consortium composition guarantees an excellent balance of expertise, infrastructure, and data, covering all the aspects that must be addressed for this project to be successful. In the following table, we detail the risks that we identified for each WP, together with their mitigation. We note that, as elaborated in Section 5, each of our WPs offers a critical contribution on its own, advancing the state of the art and leading to improved technology even in case a specific WP or task would produce negative results. Besides, none of the WP would be a roadblock for the overall project, as we planned alternative approaches that have little or no risk.

Risk assessment table. P = Probability; I = Implication; L = Low; M = Medium; H = High.

WP	Risk	P	I	Mitigation	Alternative Approach
WP 1	The modeling framework is not exhaustive.	L	M	We collect input and feedback from our expert industrial partners at an early stage.	We combine and unify existing frameworks[4], such as ATT&CK, CAPEC, CWE.
WP 2	The adopted vulnerability analysis techniques are not scalable enough to	L	H	We use lightweight instrumentation techniques (i.e., rr) that have successfully been applied to large programs. Also, we adopt intensive techniques like symbolic	We introduce approximations in our analysis techniques (e.g., taint analysis). This avoids

[3] AVR Roadmap: https://dcypher.nl/file/download/70606922-dc7a-4099-acaf-e374f96f2ce7/roadmap-avr-3.pdf
[4] ATT&CK: https://attack.mitre.org/; CAPEC: https://capec.mitre.org/; Common Weakness Enumeration: https://cwe.mitre.org/

	be applied on complex software.			execution that scale poorly only when necessary on small portions of code.	scalability issues at the cost of increased false positives.
WP 3	The adopted techniques can only reason about limited sets of primitives.	M	L	We define our capability framework and augmentation process in a generic fashion, without introducing specific constraints.	We prioritize commonly combined primitives (e.g., memory write + info leak), maximizing our impact.
WP 4	The generated patches introduce significant overhead.	L	H	We only patch dangerous side effects of vulnerabilities, without introducing overhead for non-exploitable effects.	Rely on user annotations to target the most sensitive primitives to protect.
WP 5	The developed framework requires significant resources.	L	M	We engineer our framework in a modular way to allow end-users to customize it to precisely fit their needs and exclude unnecessary, expensive tasks.	We let end-users choose the level of approximations to balance required resources and precision of the results.
WP 6	The evaluation is inappropriate or unrealistic.	L	H	The evaluation happens in real or realistic settings defined jointly with our industrial partners.	Rely on additional case studies from the literature or available reports.

4.2 Existing infrastructure

The research will largely take place at the University of Twente (UT), in the SCS group, and at the Vrije Universiteit Amsterdam (VU), in the VUSec group. UT and VU provide a workplace for the researchers as well as dedicated facilities, such as computational resources and storage. The to-be-appointed PDEng will spend one year at TNO integrating our framework into TNO's Cyber Reasoning System[3]. Finally, our industrial partners, such as Northwave and Secura, will give us access to their testing environments to validate the technology. In particular, Secura will provide anonymized data about unpatched systems that they encountered during their security testing. Occasionally, the involved researchers will physically visit such environments to conduct experiments.

4.3 Time plan and division of tasks

(WP1) Requirement analysis and capability theory definition - UT, VU
WP1, carried by the to-be-hired PhD at UT, consists of four tasks: **(T1.1)** Literature study and review of real-word vulnerabilities and CVEs; **(T1.2)** Creation of models for vulnerabilities, capabilities, and mitigations; **(T1.3)** Definition of requirements and threat models; **(T1.4)** Finalization of the capability theory framework.

(WP2) Vulnerability analysis - VU, UT
WP2, carried by the to-be-hired PhD at VU, consists of four tasks: **(T2.1)** Development of an efficient (re-)tracing technique; **(T2.2)** Definition of techniques to map traces to capabilities; **(T2.3)** Development of a directed fuzzer for capability exploration; **(T2.4)** Development of techniques to identify vulnerability root causes.

(WP3) Multi-vulnerability reasoning and risk analysis - UT, VU
WP1, carried by the PhD at UT, consists of four tasks: **(T3.1)** Creation of capability augmentation processes; **(T3.2)** Design of techniques to reason about constraints on vulnerability chains; **(T3.3)** Definition of severity metrics for vulnerability chains; **(T3.4)** Design of the Vulnerability Dependency Graph and patch-prioritization strategies.

(WP4) Patch generation - VU, UT
WP4, carried by the PhD at VU, consists of two tasks: **(T4.1)** Design and development of a static analysis component to characterize allocations; **(T4.2)** Design and development of an efficient sanitizer.

(WP5) System integration - UT, VU, TNO

WP5 will mainly be carried out by the PDEng at UT in collaboration with TNO. We foresee three tasks: **(T5.1)** Design of the system architecture and component interfaces; **(T5.2)** Implementation of the cohesive framework; **(T5.3)** Creation of documentation, test cases, and use case examples.

(WP6) Evaluation and utilization - VU, UT, TNO, NCSC, Northwave, Secura, ING, Microsoft
WP6 will be carried by both PhDs and the PDEng, in close collaboration with TNO and our industry partners. We define four tasks: **(T6.1)** Continuous crawling of data about the latest and upcoming CVEs; **(T6.2)** Assessment of the proposed techniques on existing datasets and collected CVEs; **(T6.3)** Validation of the development framework in industrial environments; **(T6.4)** Organization of the release event and workshop.

Deliverables. Deliverables (D) Type (T): A = research article; R = report; S = software.

- **D1.** [Type: A, R] Capability theory model – WP1
- **D2.** [Type: S] Binary tracer and re-tracer – WP2
- **D3**. [Type: A, S] Fuzzer to explore capabilities – WP2
- **D4.** [Type: A] Techniques to reason about the combination of vulnerabilities – WP3
- **D5.** [Type: A, S] Efficient isolation mechanisms for patching vulnerabilities – WP4
- **D6.** [Type: A, S] Cohesive end-to-end framework – WP5
- **D7.** [Type: R] Documentation for our framework – WP5
- **D8.** [Type: A,R] Report of evaluation and utilization in real-world settings – WP6
- **D9.** [Type: R] Final PhD dissertation (UT) – WP1, WP3
- **D10.** [Type: R] Final PhD dissertation (VU) – WP2, WP4
- **D11.** [Type: R] Final PDEng dissertation – WP5, WP6

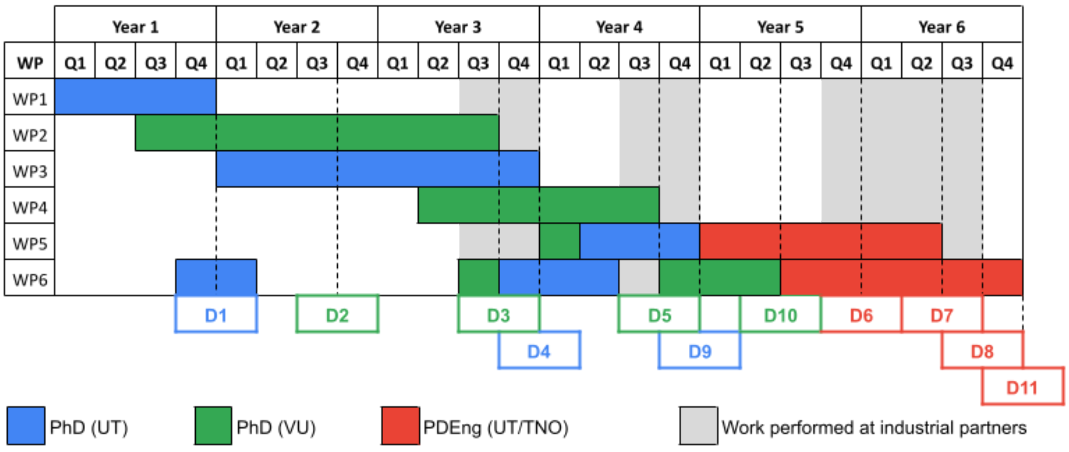

***Figure 2**. Planning and timeline of the project, including its deliverables.*

Planning. The project lasts 6 years. Figure 2 shows the planning of our WPs and deliverables. The gray bars show in which phases work will be performed at our partners. In Q4 of Y3 we will closely collaborate with NCSC to evaluate our developed risk assessment techniques and disseminate our findings. Toward the end of Y4, we will assess our automated patching technology together with Secura and Northwave. Finally, at the end of Y4, we will hire a PDEng student that will work on the system integration and utilization of our framework. This part of the work is critical for the project's knowledge utilization aims, but is not suitable for a PhD student as it is not geared towards academic publications. The PDEng will work closely with our industrial partners and will spend the final year at TNO to integrate our framework into TNO's CRS and organize a release event and a workshop.

5. Positioning of the project proposal

5.1 Uniqueness of the proposed project

Existing research into vulnerability prioritization considers vulnerabilities in isolation, disregarding the coexistence of other bugs. For example, two vulnerabilities that are harmless by themselves are deprioritized even if one violates the assumptions under which the other is harmless. As a result, operators may think they patched all critical vulnerabilities, while in reality their system is still exposed to attacks. **No prior work is able to consider the interaction between multiple vulnerabilities in risk analysis**. In contrast, our novel model of vulnerability impact, or 'capability theory', specifically supports reasoning over interactions between vulnerabilities. No current model is suitable for this purpose. Besides, we use the model for patch generation to produce faster and safer patches than prior approaches. Differently from existing work, **our approach scales to large software**, which is critical to build practical defenses. Instead, existing work relies on heavy techniques that severely limit its scalability. In practice, most solutions do not scale much beyond "toy" programs.

Vulnerability Analysis. The state-of-the-art in vulnerability analysis [Wu18,Zou22,Yag2b,Che20] is considerably more limited than P6. FUZE [Wu18] uses formal modeling to chain gadgets, but cannot reason over multiple vulnerabilities. KOOBE [Che20] uses symbolic tracing for automated exploit generation, but in an ad hoc fashion that targets only out-of-bounds bugs in the Linux kernel. In contrast, P6 builds on a generic capability theory to reason about a wide range of vulnerabilities and uses fuzzing also to find alternatives to symbolically traced paths—bypassing drawbacks of pure symbolic execution. Syzscope [Zou22] is a targeted solution to revisit and re-assess the security impact of previously found low-risk bugs in the kernel. BunkerBuster [Yag21b] uses symbolic execution to find root causes of crashes, but supports neither severity analysis nor exploit generation. As with KOOBE, its exclusive reliance on symbolic execution makes scaling for defensive purposes impossible.

Risk Prediction. Since CVSS scores provide only a crude estimate of risk, researchers proposed better predictors of exploitability. Suciu et al. [Suc22] use machine learning to predict exploitability from vulnerability reports. Such approaches depend on publicly available information and human analysis. In contrast, **our approach requires neither information to be available nor human analysis**, allowing new threats to be analyzed earlier.

Exploitation Capabilities. Prior work [Tem01, Che20, Zou22] modeled a limited subset of "requires/provides" capabilities, without providing a comprehensive framework or considering modern mitigations. Instead, **we provide the first framework that models both exploitation capabilities and mitigation techniques** and that allows for **reasoning about how capabilities can be combined under the constraints imposed by mitigations**.

Automated Patching. Most research on automated patching uses AI and natural language processing to identify similar patterns and their fixes, or generates candidate patches through fuzzing. A recent survey found that such approaches can only fix 54.6% of vulnerabilities [Pin21]. This means that many vulnerabilities are not fixed until a human is available. In contrast, rather than eliminating vulnerabilities completely, P6 prevents them from generating attack capabilities. This makes it much easier to ensure that we can offer at least a first patch to **mitigate attack primitives from memory errors in almost all cases** without changing the program behavior.

5.2 Embedding of the proposed project

The applicants are internationally-recognized experts in automated security testing. The team has won awards in top security conferences (e.g., best paper awards at IEEE Security & Privacy, EuroS&P, and USENIX Security), and industrial venues (e.g., 5 Pwnie awards at BlackHat). Both UT-SCS and VUSec are active members of the Dutch Automated Vulnerability Research (AVR) program[5] and contributed to the AVR Roadmap, supported by 3 ministries. Also, one of the PIs co-authored all 3 versions of the National Cyber Security Research Agenda.

The UT-SCS main applicant contributed to the development of a program analysis framework, angr[6], widely used in both industry and academia, which WP2 and WP3 will build upon. In prior research [*Red20], we used angr to build symbolic models of insecure interactions among multiple programs, which we will extend in WP1. The SCS group combines expertise in cybersecurity and in semantics, conceptual modeling, and software engineering, all essential to the project's objectives [*Oli22]. Internationally, SCS is well-connected: as a member of the International Secure Systems Lab (iSecLab), SCS collaborates with international security groups, e.g., UCSB (G. Vigna, C. Kruegel), ASU (Y. Shoshitaishvili), Purdue (A. Bianchi), Polimi (S. Zanero), and TU Wien (M. Lindorfer).

[5] https://dcypher.nl/groups/view/26abcf04-19a7-47f0-b9f5-81d43d769dc9/automated-vulnerability-research
[6] https://github.com/angr/angr

Meanwhile, VUSec has worked on program analysis in the area of malware, fuzzing [*Raw17,Veg16,Jai18,*Ost20, Ger22,Sch22,Gul20], binary rewriting [Alt20], and program hardening [Che15,Che17,Slo12,*Bha19]. VUSec also participates in the Theseus project, which aims to understand and improve the slow patching procedures in many organizations. Some of its program analysis techniques can now be found in state-of-the-art commercial program analysis solutions such as Binary Ninja. VUSec collaborates with many international groups, including CISPA (T. Holz), ETH Zurich (K. Razavi, O. Mutlu), Sapienza University of Rome (L. Querzoni), and UCSB (G. Vigna, C. Kuegel).

6. Knowledge Utilisation

6.1 Scientific and/or Societal Impact

6.1.1. Choice of impact

Specify which kind of impact the proposal focuses on: Scientific and societal impact are of comparable focus

6.1.2. Motivation of the choice of impact

Scientific Impact. Our research will offer deep insights and novel methodologies into modeling the impact of security vulnerabilities, fostering new research in our and related fields. In the cybersecurity domain, our methods will allow future research to advance from studying traditional security vulnerabilities individually into modeling the effect of vulnerabilities in complex, interconnected infrastructures. In fact, our multi-vulnerability reasoning methodology will enable the analysis of vulnerabilities that occur when *different* programs interact, a radical change over current directions. Besides, our efficient and lightweight analysis techniques will enable future research to study security issues that have been ignored so far because they have been considered too complex, due to the scalability limitations of existing techniques. Outside cybersecurity, our methods will be useful to model and analyze other types of software behaviors and properties, such as those related to safety and verification.

Societal Impact. Our society relies on software to provide services to people. As such, vulnerabilities in software have tremendous consequences, in terms of economic damage, violation of human rights, and disruption of critical infrastructures. This project provides the technology to automatically fix vulnerabilities, resulting in better security, lower expenses to cyberattacks, and less service interruptions. By reducing costs and effort required to patch vulnerabilities, our framework also benefits developers and security testing companies, which will automate the risk assessment and patching processes, providing better and faster services to their customers.

Most organizations today do not even employ fuzzing tools to find vulnerabilities, let alone automated vulnerability analysis or patching techniques, as the tools are limited and unfriendly. They focus on single and simple programs with trivial interfaces (e.g., that read a single input file and produce an output file), where real-world systems typically involve multiple programs with more complicated interfaces and attackers string together multiple issues to launch an attack. Moreover, tools still require a lot of (human) expertise to apply effectively and many organizations simply do not have experts to spare. Nevertheless, there is clearly demand for such tools as is evidenced by the Code Intelligence start-up in Germany that raised as much as 12Meuro to develop easier-to-use fuzzing and analysis tools and that counts industrial behemoths such as Google, T-Mobile, Continental and Bosch as its customers. Using fuzzing, Google discovered over 16,000 (!) bugs in the Chrome browser alone and more than 11,000 in 160 open source projects[7]. Similarly, Microsoft claims that its fuzzing technology saved the company millions of dollars [God08]. However, fuzzing is only a first step to fix software bugs and only pertains to a fraction of organizations. Analysis of vulnerabilities and assessing of their severity is not just the next step, but also relevant for any organization faced with patch triaging and prioritization (i.e., *every* organization). No product or academic solution fills this gap. As a matter of fact, the costs of inadequate software testing on the US national economy has been estimated to be around 50 billions of dollars [Pla02], while the relative cost of fixing defects before release is roughly ten times lower than after deployment [Daw10].

[7] https://google .github.io/clusterfuzz/#trophies

No security patching company exists in the Netherlands, but this project will allow TNO to use our efforts to similarly provide a patching solution easily adoptable by Dutch organizations. We will also investigate the creation of new businesses - both UT and VU are active in this domain, e.g., Startup Twente and VU StartHub[8].

Implementation. We will closely work with our partners and disseminate our work both in industry and academia to ensure that our results will inspire future research and will be used in practice. To do so, we will hire a PDEng student in the second half of the project, with the goal of engineering our research outcomes into usable tools. For this, we will work closely with TNO (the largest Dutch Organization for Applied Scientific Research), which will host the PDEng for one year with the goal of integrating our techniques in TNO's Cyber Reasoning System. Doing so will facilitate the usability of the research results. In the integration phase, we will particularly focus on guaranteeing that the developed techniques are separately accessible through modular components, which will make our system customisable to fit the needs of our partners. We will validate our technology within the testing environments of our industry collaborators and, through TNO, we will extend our circle of partners within the Dutch Automated Vulnerability Research (AVR) Roadmap[3]. For NCSC, the risk assessment part of our project is of great relevance. In fact, NCSC regularly publishes blog posts, brochures, and announcements about the severity of new vulnerabilities, with the goal of informing Dutch organizations and providing actionable recommendations. We will thus work with NCSC to make sure they can leverage our technology to obtain reliable insights, standardizing the format of our results and adopting emerging standards for Vulnerability Exploitability eXchange (VEX). Our partners Secura and Northwave will provide us data and testing infrastructures to assess our technology in an industrial environment and meet the expected TRL. Secura is particularly interested in embedding our automated patching technology into their products in order to improve the services they offer to their customers, making security testing more efficient and reducing costs. Finally, our partners ING and Microsoft are the end users of our knowledge chain as they need to guarantee that their software is secure, and for this they are interested in adopting our framework, giving us feedback on its usability and efficiency.

Planning. As described in Section 4.3 and shown in Figure 2, our utilization strategy is deeply embedded into the project. We reserved dedicated periods during the project in which we will work at our partner organizations to support the utilization in their contexts. Toward the end of year 3, we will work with NCSC to assess our risk assessment techniques and disseminate our findings. We plan to give NCSC access to our technology to obtain insights on upcoming vulnerabilities that might affect Dutch organizations, allowing NCSC to publish actionable recommendations. In year 4, we will closely work with Secura and Northwave to test our technology on their data and testing infrastructures. Finally, our PDEng will be hosted by TNO for the final year of the project and will integrate our framework into TNO's Cyber Reasoning System, and will investigate the support of our technologies with Secura and Northwave's products. During the integration, we will showcase the framework to our end users, ING and Microsoft, which are committed to test it and provide feedback about its usability and efficiency.

To maximize the impact and utilization of our results, we defined concrete plans.
Conference publications. The work packages WP1-4 will lead to paper submissions to top security conferences (e.g., IEEE S&P, CCS), maximizing the visibility of this research. We plan two paper submissions per work package.
Industrial venues. To disseminate our research in industry, we will submit our results for presentation at peer-reviewed industrial security conferences, such as Black Hat and RSA Conference.
Open sourcing. We will release our technology open source. This will give our work visibility and allow others to replicate our results and build upon our research. We will collaborate with the national open sourcing initiative COSSAS[9], led by TNO, to maximize the accessibility of our framework and guarantee long-term maintenance.
User Committee. We organize UC meetings at the start of the project and every six months. In the meetings, we present our results and plans, collect feedback, and discuss applications of our research, stimulating utilization.

[8] Startup Twente: https://novelt.com/en/services/startup-twente/; VU StartHub: https://vu-ondernemend.nl/en/
[9] An open source community to drive the automation of cyber security operations: https://cossas-project.org/.

Web-based platform. We will release a public web-based application for performing vulnerability analysis. We will also summarize our results on the website and document the application of our techniques by collaborators.
Release event & workshop. We will organize a release event to present our framework and an in-depth technical workshop on the application and usage of the framework. We will coordinate with dcypher and ACCSS[10] for the promotion of both events, inviting security organizations, practitioners, and researchers.
Validation in industrial environments. We will assess the developed technology within Secura and Northwave's testing infrastructures, meeting a higher TRL for our technology and demonstrating how they will benefit from embedding our techniques in their products. Since our technology will help our partners to automate intensive workloads and obtain better insights into software vulnerabilities and their risk, it is likely that they will adopt it.

Adoption. The adoption of our techniques in research is expected to start during Y4, while application in practice will start during Y6 and will be established within 2-3 years after the project. Our joint experiments, planned in Y3, Y4, and Y5, will enable our partners to turn our results into market-ready products, and we will support these efforts by advising our partners on the technology. We plan to facilitate this through master student projects — every year several UT students work on their thesis as interns at TNO, Secura, and Northwave. Finally, due to our open source policy, we expect that external organizations (not in the user committee) will also utilize our results.

6.2 Composition of the research group

OMITTED ON PURPOSE

6.3 Past performance

OMITTED ON PURPOSE

[10] dcypher: https://dcypher.nl/; ACCSS: https://accss.nl/

6.4 Composition of the user committee

OMITTED ON PURPOSE

7. Intellectual property

OMITTED ON PURPOSE

8. Financial planning

8.1 Personnel positions

Position	*Category*	*FTE*	*Months*	*Tariff (see FP)**		*Organization name*	*Bench fee*
1.	PhD student	1.0	48	€	263.737	University of Twente	yes
2.	PhD student	1.0	48	€	263.737	Vrije Universiteit Amsterdam	yes

Position	*Category*	*FTE*	*Months*	*Tariff (see FP)**		*Organization name*	*Bench fee*
3.	PDEng	1.0	24	€	118.485	University of Twente	no

8.2 Material costs

8.2.1 Project-related goods/services

We request 5.000 EUR for purchasing and maintaining a server where we will host a publicly-accessible, web-based framework for characterizing software vulnerabilities, as discussed in Section 6.1.

8.2.2 Travel and accommodation costs for the requested personnel positions

We request 20.000 EUR for travel costs (5k EUR for each year of the PhD students) because we aim at publishing our results at the top security conferences, e.g., USENIX Security, which are typically held in the USA.

8.2.3 Implementation costs

We request 10.000 EUR for publication fees and costs regarding open access.

8.2.4 Cleanroom costs: n/a

8.3 Investments: n/a

8.4 Knowledge utilisation

We request 20.000 EUR: 12k EUR for the organization of a release event and a technical workshop; 4k EUR to file a patent application; 4k EUR for promotion and advertising costs (e.g., design of logos and graphics).

8.5 Internationalisation: n/a

8.6 Money follows cooperation: n/a

8.7 Contribution from users

TNO will provide 38.150,00 EUR in-kind contribution for activities related to the organization of KU events, communication and advertising, and advice (domain-specific knowledge). Northwave will provide 1.308 EUR in-kind contribution for domain-specific knowledge and advice on the evaluation and utilization phases.

8.8 Cost Breakdown

Total project budget incl. co-funding	€	750.417,00
Total in-cash co-funding	€	0,00
Total in-kind co-funding	€	39.458,00
Total requested NWO funding	€	710.959,00

9. References

9.1 Selection of key publications research group

[*Rua21] N. Ruaro, L. Dresel, K. Zeng, T. Bao, M. Polino, A. Continella, S. Zanero, C. Kruegel, G. Vigna. *"SyML: Guiding Symbolic Execution Toward Vulnerable States Through Pattern Learning"*. In Proceedings of the International Symposium on Research in Attacks, Intrusions and Defenses (RAID), 2021.

[*Gri20] F. Gritti, L. Fontana, E. Gustafson, F. Pagani, A. Continella, C. Kruegel, G. Vigna. *"SYMBION: Interleaving Symbolic with Concrete Execution"*. Proceedings of the IEEE Conference on Communications and Network Security (CNS), 2020.

[*Red20] N. Redini, A. Machiry, R. Wang, C. Spensky, A. Continella, Y. Shoshitaishvili, C. Kruegel, G. Vigna. *"KARONTE: Detecting Insecure Multi-binary Interactions in Embedded Firmware"*. In Proceedings of the IEEE Symposium on Security & Privacy (S&P), 2020.

[*Sma23] de Smale, S., van Dijk, R., Bouwman, X., van der Ham, J. and van Eeten, M. “No One Drinks From the Firehose: How Organizations Filter and Prioritize Vulnerability Information”. To appear in Proceedings of the IEEE Symposium on Security & Privacy (S&P), 2023.

[*Oli22] I. Oliveira, T. Prince Sales, R. Baratella, M. Fumagalli, G. Guizzardi. “*An ontology of security from a risk treatment perspective*”. Proceedings of the International Conference on Conceptual Modeling (ER), 2022.

[*Joh22] Johannesmeyer, B.; Koschel, J.; Razavi, K.; Bos, H.; and Giuffrida, C. , Kasper: Scanning for Generalized Transient Execution Gadgets in the Linux Kernel. In NDSS, April 2022.

[*Ost20] Österlund, S.; Razavi, K.; Bos, H.; and Giuffrida, C., ParmeSan: Sanitizer-guided Greybox Fuzzing. In USENIX Security, August 2020.

[*Bha19] . Bhat, K.; van der Kouwe, E.; Bos, H.; and Giuffrida, C., ProbeGuard: Mitigating Probing Attacks Through Reactive Program Transformations. In ASPLOS, April 2019.

[*Raw17] Rawat, S.; Jain, V.; Kumar, A.; Cojocar, L.; Giuffrida, C.; and Bos, H. , VUzzer: Application-aware Evolutionary Fuzzing. In NDSS, February 2017.

[*Vee17] van der Veen, V.; Andriesse, D.; Stamatogiannakis, M.; Chen, X.; Bos, H.; and Giuffrida, C., The Dynamics of Innocent Flesh on the Bone: Code Reuse Ten Years Later. In CCS, October 2017.

9.2 List of publications cited

[Alt20] Altinay, A., Nash, J., Kroes, T., Rajasekaran, P., Zhou, D., Dabrowski, A., Gens, D., Na, Y., Volckaert, S., Giuffrida, C. and Bos, H. "BinRec: dynamic binary lifting and recompilation". In Proceedings of the European Conference on Computer Systems (EuroSys), 2020.

[Bla20] Blazytko, T., Schlögel, M., Aschermann, C., Abbasi, A., Frank, J., Wörner, S. and Holz, T. “AURORA: Statistical Crash Analysis for Automated Root Cause Explanation”. In Proceedings of the USENIX Security Symposium, 2020.

[Böh17] Böhme, M., Pham, V.T., Nguyen, M.D. and Roychoudhury, A. “Directed greybox fuzzing”. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS), 2017.
[Bos20] Bos H.J., Athanasopoulos E., Konoth R. K., Razavi K., K., Koninklijke Philips NV. “Device and method for secure communication”. EU Patent Application EP19208505.8A, 2020.
[Bur19] Burow, N., Zhang, X. and Payer, M. “SoK: Shining light on shadow stacks”. In Proceedings of the IEEE Symposium on Security and Privacy (S&P), 2019..

[Che15] Chen, X., Slowinska, A., Andriesse, D., Bos, H. and Giuffrida, C. "StackArmor: Comprehensive Protection From Stack-based Memory Error Vulnerabilities for Binaries". In NDSS, 2015.

[Che17] Chen, X., Bos, H. and Giuffrida, C. "CodeArmor: Virtualizing the code space to counter disclosure attacks". In IEEE European Symposium on Security and Privacy (EuroS&P), 2017.

[Che20] Chen, W., Zou, X., Li, G., & Qian, Z. "KOOBE: Towards Facilitating Exploit Generation of Kernel Out-Of-Bounds Write Vulnerabilities". In Proceedings of the USENIX Security Symposium, 2020.
[Con18] Continella A., Zanero S., Maggi F., Guagnelli A., Zingaro G., Barenghi A., De Pasquale G. “Protection system and method for protecting a computer system against ransomware attacks” Patent WO2018100520A1, 2018.

[Daw10] Dawson, M., Burrell, D.N., Rahim, E. and Brewster, S. “Integrating software assurance into the software development life cycle (SDLC)”. Journal of Information Systems Technology and Planning, 3(6), pp.49-53, 2010.

[Ger22] Geretto, E., Giuffrida, C., Bos, H., & van der Kouwe, E. "Snappy: Efficient Fuzzing with Adaptive and Mutable Snapshots". In ACSAC, 2022.

[God08] Godefroid, P., Levin, M.Y. and Molnar, D.A. “Automated whitebox fuzz testing”. In Proceedings of the Network and Distributed System Security Symposium (NDSS), 2008.

[Goo21] Google Open Source Security Team, “Fuzzing Java in OSS-Fuzz”, https://security.googleblog.com/2021/03/fuzzing-java-in-oss-fuzz.html, 2021.

[Gul20] Güler, E., Görz, P., Geretto, E., Jemmett, A., Österlund, S., Bos, H., Giuffrida, C. and Holz, T., "Cupid: Automatic Fuzzer Selection for Collaborative Fuzzing". In Annual Computer Security Applications Conference (ACSAC), 2020.

[IBM22] IBM, “Cost of a data breach 2022**:** A million-dollar race to detect and respond**”,** https://www.ibm.com/security/data-breach, 2022.

[Jai18] Jain, V., Rawat, S., Giuffrida, C., & Bos, H. "TIFF: using input type inference to improve fuzzing". In Proceedings of the 34th Annual Computer Security Applications Conference (ACSAC), 2018.

[Kon20] Konoth, R.K., Tatar, A., Oliverio, M., Andriesse, D., Bos, H.J., Giuffrida, C. and Razavi, K., Koninklijke Philips NV. “Computing device with increased resistance against rowhammer attacks”. U.S. Patent Application 16/059,357, 2020.

[Moz22] Mozilla. rr-project, https://rr-project.org/.

[Mu18] Mu, D., Cuevas, A., Yang, L., Hu, H., Xing, X., Mao, B. and Wang, G.. *“Understanding the reproducibility of crowd-reported security vulnerabilities*”. In Proceedings of the USENIX Security Symposium, 2018.

[OCa17] O'Callahan, R., Jones, C., Froyd, N., Huey, K., Noll, A. and Partush, N., 2017. Engineering record and replay for deployability. In Proceedings of the USENIX Annual Technical Conference (ATC), 2017.

[Pin21] Pinconschi, E., Abreu, R., & Adão, P. "A Comparative Study of Automatic Program Repair Techniques for Security Vulnerabilities". In IEEE International Symposium on Software Reliability Engineering (ISSRE), 2021

[Pla02] Planning, S. “The economic impacts of inadequate infrastructure for software testing”. National Institute of Standards and Technology, 2002.

[Sch22] Scharnowski, T., Bars, N., Schloegel, M., Gustafson, E., Muench, M., Vigna, G., Kruegel, C., Holz, T. and Abbasi, A. "Fuzzware: Using Precise MMIO Modeling for Effective Firmware Fuzzing". In USENIX Security Symposium, 2022.

[Ser12] Fermin J. Serna. “The info leak era on software exploitation”. Black Hat USA 2012.

[Slo12] Slowinska, A., Stancescu, T. and Bos, H. "Body armor for binaries: preventing buffer overflows without recompilation". USENIX Annual Technical Conference (USENIX ATC), 2012.

[Suc22] Suciu, O., Nelson, C., Lyu, Z., Bao, T., and Dumitraș, T. "Expected exploitability: Predicting the development of functional vulnerability exploits". In Proceedings of the USENIX Security Symposium, 2022.

[Tem01] Templeton, S. and K. Levitt, "A requires/provides model for computer attacks", Proceedings of the 2000 workshop on New Security Paradigms (NSPW), 2001.,

[Veg16] Veggalam, S., Rawat, S., Haller, I., & Bos, H. "iFuzzer: An evolutionary interpreter fuzzer using genetic programming". In European Symposium on Research in Computer Security (ESORICS), 2016.

[Yag21a] Yagemann, C., Pruett, M., Chung, S.P., Bittick, K., Saltaformaggio, B. and Lee, W. "ARCUS: Symbolic Root Cause Analysis of Exploits in Production Systems". In Proceedings of the USENIX Security Symposium, 2021.

[Yag21b] Yagemann, C., Chung, S. P., Saltaformaggio, B., & Lee, W. "Automated bug hunting with data-driven symbolic root cause analysis". In Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS), 2021.

[Wu18] Wu, W., Chen, Y., Xu, J., Xing, X., Gong, X., & Zou, W. "FUZE: Towards Facilitating Exploit Generation for Kernel Use-After-Free Vulnerabilities". In Proceedings of the USENIX Security Symposium, 2018.

[Zou22] Zou, X., Li, G., Chen, W., Zhang, H., & Qian, Z. "SyzScope: Revealing High-Risk Security Impacts of Fuzzer-Exposed Bugs in Linux kernel". In Proceedings of the USENIX Security Symposium, 2022.

10. Expected Technology Readiness Level (TRL)

Technology Readiness Level (TRL)	Current	After	
	☐	☐	1. Basic principles observed
	☐	☐	2. Technology concept formulated
	☒	☐	3. Experimental proof of concept
	☐	☐	4. Technology validated in lab
	☐	☐	5. Technology validated in relevant environment
	☐	☒	6. Technology demonstrated in relevant environment
	☐	☐	7. System prototype demonstration in operational environment
	☐	☐	8. System complete and qualified
	☐	☐	9. Actual system proven in operational environment

11. Data management section

Questions Data management section

1. Will this project involve re-using existing research data?
   - ☒ Yes: From my own or a collaborator's prior research.
   - ☒ Yes: Publicly available data.
   - ☐ No: Have you considered re-using existing data but discarded the possibility? Why?

   We will use both publicly available data and private data from our industry partners. For the publicly available data, no constraints are present on the re-use of the data. The private data from the industry partners cannot be shared publicly as it might contain customers' information.

2. Will data be collected or generated that are suitable for reuse?
   ☒ Yes: Please answer questions 3 and 4.
   ☐ No: Please explain why the research will not result in reusable data or in data that cannot be stored or data that for other reasons are not relevant for reuse.

3. After the project has been completed, how will the data be stored for the long-term and made available for the use by third parties? Are there possible restrictions to data sharing or embargo reasons? Please state these here.

   The source code will be made available open source on GitHub under the MIT License, as we have done for previous projects. The experimental data will be archived at the 4TU.Centre for Research Data to be available for fellow researchers. This archive holds the Data Seal of Approval and 4TU provides a DOI for each dataset, which allows dataset citations. The raw data from the industry partners cannot be shared publicly as it might contain customers' information.

4. Will any costs (financial and time) related to data management and sharing/preservation be incurred?
   ☐ Yes: Then please be sure to specify the associated expenses in the budget table of this proposal (under 8.2.3 Implementation Costs).
   ☒ No: All the necessary resources (financial and time) to store and prepare data for sharing/preservation are or will be available at no extra cost.

12. Abbreviations and acronyms

AVR: Automated Vulnerability Research

CRS: Cyber Reasoning System

CVE: Common Vulnerabilities and Exposures

CVSS: Common Vulnerability Scoring System

CWE: Common Weakness Enumeration

DoS: Denial of Service

OS: Operating System

SMT: Satisfiability Modulo Theory

UT: University of Twente

VU: Vrije University of Amsterdam