angr: A Powerful and User-friendly Binary Analysis Platform

Yan Shoshitaishvili
yans@asu.edu
Arizona State University
U.S.A.

Ruoyu Wang
fishw@asu.edu
Arizona State University
U.S.A.

Audrey Dutcher
dutcher@asu.edu
Arizona State University
U.S.A.

Christopher Kruegel
chris@cs.ucsb.edu
University of California, Santa Barbara
U.S.A.

Giovanni Vigna
vigna@cs.ucsb.edu
University of California, Santa Barbara
U.S.A.

ABSTRACT

`angr` is an open-source binary analysis platform that was released as the research artifact of a paper published at 2016 IEEE Symposium on Security & Privacy. Over the past eight years, `angr` has made significant impact in academia, industry, government, and among the enthusiasts of binary analysis. This short paper provides basic information about `angr`, its ecosystem, and its impact.

KEYWORDS

Binary Analysis, Vulnerability Discovery

ACM Reference Format:
Yan Shoshitaishvili, Ruoyu Wang, Audrey Dutcher, Christopher Kruegel, and Giovanni Vigna. 2023. angr: A Powerful and User-friendly Binary Analysis Platform. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 4 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

`angr` is an open-source binary (and program) analysis platform that was released under the BSD-2 license as the artifact to the "angr paper" [58] published at 2016 IEEE Symposium on Security & Privacy. The main project is on GitHub at https://github.com/angr/angr, with about 6.8k "stars" and over 1,000 forks at the time of writing. We wrote the first line of `angr` code in August 2013, and the year of 2023 marks the 10th birthday of `angr`.

Beneath `angr` are several key libraries that have been independently and widely used in the security community, including:

- `Cle`, a Pythonic generic binary loader.
- `PyVEX`, a library for interfacing with libVEX, which is the lifter and translator that Valgrind uses.
- `Claripy`, a Python library that provides arithmetic and symbolic abstraction for `angr`.
- `archinfo`, a Python library that provides architecture abstraction.
- `ailment`, a Python library that implements AIL, the angr intermediate language (IL).

The main `angr` project, all these underpinning libraries, as well as a (mostly) functioning graphical user interface (GUI) angr management, are hosted under the `angr` organization on GitHub at https://github.com/angr. The documentation of `angr` can be found at https://docs.angr.io.

At the time of writing, excluding blank lines and comment lines, `angr`, angr management, and `angr`'s libraries contain about 551k lines of Python code, 44k lines of C code, and 3.5k lines of C++ code.

1.1 Creating an Open and Sustainable Architecture

We pride ourselves with the creation of angr's open and sustainable architecture. A research publication, or even a source code release, is not sufficient to create an open and sustainable architecture. `angr` became a favorable choice of binary analysis research and engineering through a concerted effort.

Developing in the open. Before `angr`'s release, a common paradigm was the Research-Publish-Push cycle: Groups would only push updates to their prototypes when the paper using that code was published. This heavily discouraged the use of both FuzzBall [10, 59] and the Binary Analysis Platform (BAP) [6, 13], two older binary analysis frameworks that failed to achieve traction before `angr`'s release. First, by using such a prototype of another research group, one had to accept the fact that they were forced into a disadvantage: The latest and most powerful version of the prototype was, by definition, *not* the one they were using. Second, any bug fixes or improvements they made to the prototype would likely have to be discarded due to merge conflicts when the new version was pushed. Conversely, `angr` is developed in the open, with research project changes immediately upstreamed, and the only releases delayed are add-on modules built for specific projects. This maintains the community's trust that they are using the best architecture that we can provide, and that contributions will be relevant.

Creating an active community. `angr` thrives, in a large part, because of an active community of researchers, engineers, hackers, and enthusiasts that frequent our GitHub issues pages and our public channels to offer help. Discussions,

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Conference'17, July 2017, Washington, DC, USA

© 2023 Association for Computing Machinery.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

issues, and pull requests occur on a daily basis. This improves angr's usability, teaches new users and contributors to use and contribute to angr, and gives researchers confidence that angr is something that can actually be used.

Interoperability with existing tools. angr interacts with and uses existing tools wherever possible. There are bindings between angr and IDA Pro, angr and Binary Ninja, and angr and radare2. Likewise, angr depends on standard components of the ecosystem: Unicorn Engine is used for concrete execution, ELFTools for ELF file loading, and so on. This allows the architecture to benefit from parallel advancements by underlying libraries.

Active maintenance and development. Since its public release in 2015, the angr team kept improving angr by adding new features, improving its runtime performance, refactoring its codebase, and maintaining published research artifacts. To this end, the angr team invested a significant amount of engineering and innovation effort. Some examples include, the migration from Python 2 to Python 3 (in 2017), the introduction of angr decompiler (open and continuous development since summer 2017), the creation of angr IL (AIL), the creation of program interaction description library (archr [61]), the introduction of type hints (since 2022), and the maintenance of key components in the Mechanical Phish that rely on angr (angrop [57] and Rex [62]). While many other binary analysis platforms (especially research-facing ones) are gradually dying, angr is still extremely active.

2 IMPACT

Since we released angr to the public in August 2015, it has made significant impact in the research, educational, industry, and government communities.

2.1 Academic Impact

On the academic side, the reference "angr paper" [58] has been cited 1,100 times, and many researchers simply cite the angr website [60] instead. The framework itself is actually used (i.e., not just cited as a related work) as a base by a significant body of work by researchers *unaffiliated with the authors of the framework*. We list some key research areas and papers below.

- angr is used as a base for vulnerability detection techniques in userspace binaries [17, 28, 29, 39, 77, 78];
- angr is used in the firmware of a wide variety of devices, from USB peripherals [31] to Industrial Control Systems [38], and beyond [23, 25, 44, 49];
- angr is used to detect potentially vulnerable code clones [5, 33, 47, 75] and patches [76];
- angr is used in advancements in automatic exploit generation techniques, with high-profile targets including secure enclaves [16], OS kernels [20, 68, 72, 73], and traditional user-space binaries [43, 67];
- angr is used in the development of novel code reuse attacks [9, 36, 42, 69];
- angr is used to create novel vulnerability mitigation techniques [1, 15, 22, 35, 50, 55, 75];
- angr is used to build automatic repair of binaries [32] and running systems [19];
- angr is used in code optimization [8];
- angr is used in code obfuscation [74] and deobfuscation [37, 40, 63];
- angr is used in binary code debloating [12];
- angr is used in binary attribution [2], binary code similarity detection [34, 66], and watermarking [45];
- angr is used to analyze malware targeting various operating systems and device classes [4, 7, 14, 27, 56];
- angr is used to reason about the security of hardware architectures [11, 64];
- angr is used in protocol analysis [21];
- angr is used in the creation of security visualizations [3];
- angr is used in a number of other developments in static analysis [18, 24, 30, 41, 46, 51, 54, 70, 71].
- angr has also been utilized as a component by composite program analysis, vulnerability detection, and hacking tools [48, 52, 53].

From our analysis of recent publication trends, angr appears to be used in roughly one fifth of all binary analysis research in academia.

2.2 Educational Impact

Aside from its use as a research tool in academia, angr is also heavily used in educational cybersecurity competitions. Upon its release, it "raised the bar" of cybersecurity challenge difficulty with its ability to automatically solve many reverse engineering problems [26]. For example, the year after angr's release, the organizers of DEF CON CTF (the world championship of cybersecurity competitions) fielded fewer reverse engineering challenges than other types of challenges in their qualifying event. Anecdotally, from conversation with the organizers, this seemed to be due to the difficulty in making "angr-proof" challenges. Since then, competition organizers have learned to create increasingly harder challenges to stump angr, further challenging participants and increasing the value that they can get from such events.

In a further example of angr's impact on education, in the 2019 CSAW Embedded Security Challenge hardware hacking competition, a challenge focused on reverse engineering and exploitation of embedded devices, nine of the top 10 teams used angr for their solutions.

2.3 Industrial Impact

angr has also impacted industry. It is used by a wide range of companies across different market segments, from Apple and Samsung to Boeing and Raytheon, and a number of national labs, FFRDCs, and similar entities.

2.4 Governmental Impact

angr has impacted government, with the angr-based Arbiter [65] analysis technique utilized by cyber operators in the Department of Defense to help accomplish mission goals.

REFERENCES

[1] Ali Abbasi, Jos Wetzels, Wouter Bokslag, Emmanuele Zambon, and Sandro Etalle. 2017. Ushield. In *International Conference on Network and System Security*. Springer, 694–709.

[2] Saed Alrabaee, Paria Shirani, Lingyu Wang, Mourad Debbabi, and Aiman Hanna. 2018. On leveraging coding habits for effective binary authorship attribution. In *European Symposium on Research in Computer Security*. Springer, 26–47.

[3] Marco Angelini, Graziano Blasilli, Luca Borzacchiello, Emilio Coppa, Daniele Cono D'Elia, Camil Demetrescu, Simone Lenti, Simone Nicchi, and Giuseppe Santucci. 2019. Symnav: visually assisting symbolic execution. *IEEE Symposium on Visualizations in Cyber Security*.

[4] Roberto Baldoni, Emilio Coppa, Daniele Cono D'Elia, and Camil Demetrescu. 2017. Assisting malware analysis with symbolic execution: a case study. In *International Conference on Cyber Security Cryptography and Machine Learning*. Springer, 171–188.

[5] Roberto Baldoni, Giuseppe Antonio Di Luna, Luca Massarelli, Fabio Petroni, and Leonardo Querzoni. 2018. Unsupervised features extraction for binary similarity using graph embedding neural networks. *arXiv preprint arXiv:1810.09683*.

[6] [n. d.] BAP: binary analysis platform. https://github.com/BinaryAnalysisPlatform/bap. ().

[7] Eduard Baranov, Fabrizio Biondi, Olivier Decourbe, Thomas Given-Wilson, Axel Legay, Cassius Puodzius, Jean Quilbeuf, and Stefano Sebastio. 2018. Efficient extraction of malware signatures through system calls and symbolic execution: an experience report.

[8] Gergö Barany. 2018. Finding missed compiler optimizations by differential testing. In *Proceedings of the 27th International Conference on Compiler Construction*. ACM, 82–92.

[9] Andrea Biondo, Mauro Conti, and Daniele Lain. 2018. Back to the epilogue: evading control flow guard via unaligned targets. In *Proceedings of the 25th Annual Network and Distributed System Security Symposium (NDSS)*.

[10] [n. d.] BitBlaze. http://bitblaze.cs.berkeley.edu/. ().

[11] Jean-Baptiste Bréjon, Karine Heydemann, Emmanuelle Encrenaz, Quentin Meunier, and Son-Tuan Vu. 2019. Fault attack vulnerability assessment of binary code. In *Proceedings of the Sixth Workshop on Cryptography and Security in Computing Systems*. ACM, 13–18.

[12] Michael D Brown and Santosh Pande. 2019. Is less really more? Towards better metrics for measuring security improvements realized through software debloating. In *12th USENIX Workshop on Cyber Security Experimentation and Test (CSET 19)*.

[13] David Brumley, Ivan Jager, Thanassis Avgerinos, and Edward J Schwartz. 2011. Bap: a binary analysis platform. In *International Conference on Computer Aided Verification*. Springer, 463–469.

[14] Levente Buttyán. [n. d.] Towards detecting trigger-based behavior in binaries: uncovering the correct environment.

[15] Sunjay Cauligi, Craig Disselkoen, Klaus v Gleissenthall, Deian Stefan, Tamara Rezk, and Gilles Barthe. 2019. Towards constant-time foundations for the new spectre era. *arXiv preprint arXiv:1910.01755*.

[16] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H Lai. 2019. Sgxpectre: stealing intel secrets from sgx enclaves via speculative execution. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 142–157.

[17] Yaohui Chen, Dongliang Mu, Jun Xu, Zhichuang Sun, Wenbo Shen, Xinyu Xing, Long Lu, and Bing Mao. 2019. Ptrix: efficient hardware-assisted fuzzing for cots binary. *arXiv preprint arXiv:1905.10499*.

[18] Ying-Shen Chen, Wei-Ning Chen, Che-Yu Wu, Hsu-Chun Hsiao, and Shih-Kun Huang. 2018. Dynamic path pruning in symbolic execution. In *2018 IEEE Conference on Dependable and Secure Computing (DSC)*. IEEE, 1–8.

[19] Yue Chen, Yulong Zhang, Zhi Wang, Liangzhao Xia, Chenfu Bao, and Tao Wei. 2017. Adaptive android kernel live patching. In *26th USENIX Security Symposium (USENIX Security)*, 1253–1270.

[20] Yueqi Chen and Xinyu Xing. 2019. Slake: facilitating slab manipulation for exploiting vulnerabilities in the linux kernel. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1707–1722.

[21] Yurong Chen, Tian Lan, and Guru Venkataramani. [n. d.] Custompro: network protocol customization through cross-host feature analysis.

[22] Yurong Chen, Shaowen Sun, Tian Lan, and Guru Venkataramani. 2018. Toss: tailoring online server systems through binary feature customization. In *Proceedings of the 2018 Workshop on Forming an Ecosystem Around Software Transformation*. ACM, 1–7.

[23] Kai Cheng, Qiang Li, Lei Wang, Qian Chen, Yaowen Zheng, Limin Sun, and Zhenkai Liang. 2018. Dtaint: detecting the taint-style vulnerability in embedded device firmware. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 430–441.

[24] Emilio Coppa, Daniele Cono D'Elia, and Camil Demetrescu. 2017. Rethinking pointer reasoning in symbolic execution. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*. IEEE Press, 613–618.

[25] Yaniv David, Nimrod Partush, and Eran Yahav. 2018. Firmup: precise static detection of common vulnerabilities in firmware. In *ACM SIGPLAN Notices* number 2. Vol. 53. ACM, 392–404.

[26] [n. d.] Example CTF challenge solutions using angr. https://github.com/angr/angr-doc/tree/master/examples. ().

[27] Ming Fan, Jun Liu, Xiapu Luo, Kai Chen, Zhenzhou Tian, Qinghua Zheng, and Ting Liu. 2018. Android malware familial classification and representative sample selection via frequent subgraph analysis. *IEEE Transactions on Information Forensics and Security*, 13, 8, 1890–1905.

[28] Behrad Garmany, Martin Stoffel, Robert Gawlik, and Thorsten Holz. 2019. Static detection of uninitialized stack variables in binary code. In *European Symposium on Research in Computer Security*. Springer, 68–87.

[29] Peter Goodman and Alex Groce. 2018. Deepstate: symbolic unit testing for C and C++. In *NDSS Workshop on Binary Analysis Research*.

[30] Jeremy Guijt, FW Vaandrager, and J Moerman. 2018. Checking model learning hypotheses with symbolic execution.

[31] Grant Hernandez, Farhaan Fowze, Dave Jing Tian, Tuba Yavuz, and Kevin RB Butler. 2017. Firmusb: vetting usb device firmware using domain informed symbolic execution. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2245–2262.

[32] Yikun Hu, Yuanyuan Zhang, and Dawu Gu. 2019. Automatically patching vulnerabilities of binary programs via code transfer from correct versions. *IEEE Access*, 7, 28170–28184.

[33] Yikun Hu, Yuanyuan Zhang, Juanru Li, and Dawu Gu. 2017. Binary code clone detection across architectures and compiling configurations. In *Proceedings of the 25th International Conference on Program Comprehension*. IEEE Press, 88–98.

[34] Yikun Hu, Yuanyuan Zhang, Juanru Li, Hui Wang, Bodong Li, and Dawu Gu. 2018. Binmatch: a semantics-based hybrid approach on binary code clone analysis. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 104–114.

[35] Zhen Huang and Gang Tan. 2019. Rapidly mitigating vulnerabilities with security workarounds. In *Workshop on Binary Analysis Research (BAR)*.

[36] Kyriakos K Ispoglou, Bader AlBassam, Trent Jaeger, and Mathias Payer. 2018. Block oriented programming: automating data-only attacks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1868–1882.

[37] Zeliang Kan, Haoyu Wang, Lei Wu, Yao Guo, and Daniel Xiapu Luo. 2019. Automated deobfuscation of android native binary code. *arXiv preprint arXiv:1907.06828*.

[38] Anastasis Keliris and Michail Maniatakos. 2018. Icsref: a framework for automated reverse engineering of industrial control systems binaries. *arXiv preprint arXiv:1812.03478*.

[39] Eric Kilmer. 2017. *Extending Vulnerability Discovery with Fuzzing and Symbolic Execution to Realistic Applications*. Ph.D. Dissertation. Pennsylvania State University.

[40] Julian Kirsch, Clemens Jonischkeit, Thomas Kittel, Apostolis Zarras, and Claudia Eckert. 2017. Combating control flow linearization. In *IFIP International Conference on ICT Systems Security and Privacy Protection*. Springer, 385–398.

[41] Roeland Krak. 2017. *Cycle-Accurate Timing Channel Analysis of Binary Code*. Master's thesis. University of Twente.

[42] Colas Le Guernic and François Khourbiga. 2018. Taint-based return oriented programming. In.

[43] Danjun Liu, Jingyuan Wang, Zelin Rong, Xianya Mi, Fangyu Gai, Yong Tang, and Baosheng Wang. 2018. Pangr: a behavior-based automatic vulnerability detection and exploitation framework. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE, 705–712.

[44] Muqing Liu, Yuanyuan Zhang, Juanru Li, Junliang Shu, and Dawu Gu. 2016. Security analysis of vendor customized code in firmware of embedded device. In *International Conference on Security and Privacy in Communication Systems*. Springer, 722–739.

[45] Haoyu Ma, Chunfu Jia, Shijia Li, Wantong Zheng, and Dinghao Wu. 2019. Xmark: dynamic software watermarking using collatz conjecture. *IEEE Transactions on Information Forensics and Security*.

[46] Luca Massarelli, Giuseppe A Di Luna, Fabio Petroni, Leonardo Querzoni, and Roberto Baldoni. 2019. Investigating graph embedding neural networks with unsupervised features extraction for binary analysis. In *Proceedings of the 2nd Workshop on Binary Analysis Research (BAR)*.

[47] Luca Massarelli, Giuseppe Antonio Di Luna, Fabio Petroni, Roberto Baldoni, and Leonardo Querzoni. 2019. Safe: self-attentive function embeddings for binary similarity. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 309–329.

[48] Marius Muench, Dario Nisi, Aurélien Francillon, and Davide Balzarotti. 2018. Avatar 2: a multi-target orchestration platform. In *Proceedings of the 1st Workshop on Binary Analysis Research (BAR)*. Vol. 18.

[49] Geancarlo Palavicini Jr, Josiah Bryan, Eaven Sheets, Megan Kline, and John San Miguel. 2017. Towards firmware analysis of industrial internet of things (IIoT). *Special Session on Innovative CyberSecurity and Privacy for Internet of Things: Strategies, Technologies, and Implementations.*

[50] Tobias Pfeffer, Thomas Göthel, and Sabine Glesner. 2019. Automatic analysis of critical sections for efficient secure multi-execution. In *2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS).* IEEE, 318–325.

[51] Tobias Pfeffer, Paula Herber, Lucas Druschke, and Sabine Glesner. 2018. Efficient and safe control flow recovery using a restricted intermediate language. In *2018 IEEE 27th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE).* IEEE, 235–240.

[52] Joseph A Plot. 2019. Red Team In A Box (Rtib): Developing Automated Tools To Identify, Assess, And Expose Cybersecurity Vulnerabilities In Department Of The Navy Systems. Tech. rep. Naval Postgraduate School Monterey United States.

[53] Samuel D Pollard, Philip Johnson-Freyd, Jon Aytac, Tristan Duckworth, Michael J Carson, Geoffrey Compton Hulette, and Christopher B Harrison. 2019. Quameleon: a lifter and intermediate language for binary analysis.

[54] Rui Qiao and R Sekar. 2017. Function interface analysis: a principled approach for function recognition in cots binaries. In *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN).* IEEE, 201–212.

[55] Davide Quarta and Stefano Zanero. 2018. Elisa: eliciting isa of raw binaries for fine-grained code and data separation. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 15th International Conference, DIMVA 2018, Saclay, France, June 28–29, 2018, Proceedings.* Vol. 10885. Springer, 351.

[56] Najah Ben Said, Fabrizio Biondi, Vesselin Bontchev, Olivier Decourbe, Thomas Given-Wilson, Axel Legay, and Jean Quilbeuf. 2017. Detection of mirai by syntactic and semantic analysis.

[57] Salls, Christopher. 2016. angrop. https://github.com/angr/angrop. (2016).

[58] Yan Shoshitaishvili et al. 2016. SoK:(State of) the art of war: offensive techniques in binary analysis. In *2016 IEEE Symposium on Security and Privacy (SP).* IEEE, 138–157.

[59] Dawn Song et al. 2008. Bitblaze: a new approach to computer security via binary analysis. In *International Conference on Information Systems Security.* Springer, 1–25.

[60] The angr Team. 2023. angr. https://angr.io. (2023).

[61] The angr Team. 2016. archr. https://github.com/angr/archr. (2016).

[62] The angr Team. 2016. Rex. https://github.com/angr/rex. (2016).

[63] Ramtine Tofighi-Shirazi, Irina Asăvoae, Philippe Elbaz-Vincent, and Thanh-Ha Le. 2019. Defeating opaque predicates statically through machine learning and binary analysis. *arXiv preprint arXiv:1909.01640.*

[64] Strahinja Trecakov, Casey Tran, Hameed Badawy, Nafiul Siddique, Jaime Acosta, and Satyajayant Misra. 2017. Can architecture design help eliminate some common vulnerabilities? In *2017 IEEE 14th International Conference on Mobile Ad Hoc and Sensor Systems (MASS).* IEEE, 590–593.

[65] Jayakrishna Vadayath et al. 2022. Arbiter: Bridging the Static and Dynamic Divide in Vulnerability Discovery on Binary Programs. In *Proceedings of the USENIX Security Symposium (USENIX).* (Aug. 2022).

[66] Shi-Chao Wang, Chu-Lei Liu, Yao Li, and Wei-Yang Xu. 2017. Semdiff: finding semtic differences in binary programs based on angr. In *ITM Web of Conferences.* Vol. 12. EDP Sciences, 03029.

[67] Yan Wang, Chao Zhang, Xiaobo Xiang, Zixuan Zhao, Wenjie Li, Xiaorui Gong, Bingchang Liu, Kaixiang Chen, and Wei Zou. 2018. Revery: from proof-of-concept to exploitable. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security.* ACM, 1914–1927.

[68] Zhongru Wang et al. 2019. Automated vulnerability discovery and exploitation in the internet of things. *Sensors*, 19, 15, 3362.

[69] Bryan C Ward, Richard Skowyra, Chad Spensky, Jason Martin, and Hamed Okhravi. 2019. The leakage-resilience dilemma. In *European Symposium on Research in Computer Security.* Springer, 87–106.

[70] Fengguo Wei, Xingwei Lin, Xinming Ou, Ting Chen, and Xiaosong Zhang. 2018. Jn-saf: precise and efficient ndk/jni-aware inter-language static analysis framework for security vetting of android applications with native code. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security.* ACM, 1137–1150.

[71] Sheng-Han Wen, Wei-Loon Mow, Wei-Ning Chen, Chien-Yuan Wang, and Hsu-Chun Hsiao. [n. d.] Enhancing symbolic execution by machine learning based solver selection.

[72] Wei Wu, Yueqi Chen, Xinyu Xing, and Wei Zou. 2019. KEPLER: facilitating control-flow hijacking primitive evaluation for linux kernel vulnerabilities. In *28th USENIX Security Symposium (USENIX Security)*, 1187–1204.

[73] Wei Wu, Yueqi Chen, Jun Xu, Xinyu Xing, Xiaorui Gong, and Wei Zou. 2018. FUZE: towards facilitating exploit generation for kernel use-after-free vulnerabilities. In *27th USENIX Security Symposium (USENIX Security)*, 781–797.

[74] Hui Xu, Zirui Zhao, Yangfan Zhou, and Michael R Lyu. 2018. Benchmarking the capability of symbolic execution tools with logic bombs. *IEEE Transactions on Dependable and Secure Computing.*

[75] Hongfa Xue, Guru Venkataramani, and Tian Lan. 2018. Clone-hunter: accelerated bound checks elimination via binary code clone detection. In *Proceedings of the 2nd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages.* ACM, 11–19.

[76] Hang Zhang and Zhiyun Qian. 2018. Precise and accurate patch presence test for binaries. In *27th USENIX Security Symposium (USENIX Security)*, 887–902.

[77] Li Zhang and Vrizlynn LL Thing. 2017. A hybrid symbolic execution assisted fuzzing method. In *TENCON 2017-2017 IEEE Region 10 Conference.* IEEE, 822–825.

[78] Lei Zhao, Yue Duan, Heng Yin, and Jifeng Xuan. 2019. Send hardest problems my way: probabilistic path prioritization for hybrid fuzzing. In *Proceedings of the 26th Annual Network and Distributed System Security Symposium (NDSS).*