

July 10–14, 2023
Melbourne, Australia



Association for
Computing Machinery

Advancing Computing as a Science & Profession



ASIA CCS '23

Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security

Sponsored by:

ACM SIGSAC

General Chairs:

Joseph Liu, Monash University, Australia

Yang Xiang, Swinburne University of Technology, Australia

Program Chairs:

Surya Nepal, Data61, Australia

Gene Tsudik, University of California Irvine, USA

The Association for Computing Machinery
1601 Broadway, 10th Floor
New York, New York 10019, USA

ACM COPYRIGHT NOTICE. Copyright © 2023 by the Association for Computing Machinery, Inc. Permission to make digital or hard copies of part or all 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, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Publications Dept., ACM, Inc., fax +1 (212) 869-0481, or permissions@acm.org.

For other copying of articles that carry a code at the bottom of the first or last page, copying is permitted provided that the per-copy fee indicated in the code is paid through the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, +1-978-750-8400, +1-978-750-4470 (fax).

ACM ISBN: 979-8-4007-0098-9

Message from General Co-Chairs

It is our great pleasure to welcome you to the 18th ACM ASIA Conference on Computer and Communications Security 2022 (ACM ASIACCS 2023). ASIACCS 2023 takes place in Melbourne, Australia on 10th July to 14th July.

As the first post-COVID ASIACCS (travel restrictions of the world have been removed), we decide to organise it as an in-person-only conference, though we also provide an online option for the audience only (not presenter) to participate in. At the time of this writing, registration is still going on, over 200 people have finished registering as in-person.

This year we incorporate multiple programs in the conference. There are six workshops held on the first day of the conference. For the main conference, it includes four keynote talks from world-renowned researchers Prof. Wenyuan Xu, Prof. David Basin, Prof. N. Asokan and Prof. Vanessa Teague, normal (full) paper presentations and posters demonstrated during the tea and lunch break. This year we also add one more program - Tutorial into the conference. The purpose of tutorials is to let distinguished early to mid career researchers in cybersecurity to talk about the state-of-the-art research development of their own focused areas, that may also include their excellent works. We have 4 tutorials in our conference, and the topics covered range from post-quantum and machine learning security to memory corruptions and searchable encryption.

As an in-person conference, we also organise several social events for participants. A reception is held at the first day evening to allow both workshop and conference attendees to join and make networking. A half-day social event followed by the conference banquet is organised on the fourth day of the conference. In order to attract more women to the cybersecurity community, this year we further add a Women's Networking Reception on the third day of the conference. This is organised by some of our female organisation committee members while all attendees (not just restricted to women) are welcome to attend and share their experiences.

We would like to express our deep gratitude to our organisation committee, including Program Chairs Surya Nepal and Gene Tsudik, Local Organisation Chairs Sheng Wen and Xiao Chen, Registration and Finance Chairs Maggie Liu and Xingliang Yuan, Web Chairs Sharif Abuadbba and Shangqi Lai, Publicity Chairs Siqi Ma and Sushmita Ruj, Publication Chairs Seyit Camtepe and Shi-Feng Sun, Workshop Chairs Hyoungshick Kim and Shabnam Kasra, Poster Chairs Guangdong Bai and Wei Wu, Tutorial Chairs Ahmad Salehi Shahraki and Shujie Cui.

We would also like to acknowledge the support from our sponsors including our Platinum Sponsor CSIRO Data 61, Gold Sponsors Cyber Security Cooperative Research Centre (CSCRC) and Monash University Department of Software Systems and Cybersecurity (SSC), Bronze Sponsors Algorand Foundation and LinkStone.

We hope that you enjoy this conference, and wish all the participants a significant experience at ASIACCS 2023 in Melbourne.

General Co-Chairs
Joseph Liu and Yang Xiang

Message from Program Co-Chairs

We are both honored and pleased to have been entrusted to serve as PC Co-Chairs of AsiaCCS'23.

As the first post-pandemic incarnation, AsiaCCS'23 has attracted a large number of high-quality submissions from all over the world, with authors affiliated with diverse academic, non-profit, governmental, and industrial entities. After two rounds of submissions, the conference wound up with an excellent program, covering a broad range of timely and interesting topics in Security, Privacy, and Applied Cryptography.

A total of **429** submissions were received: **204** in the first, and **225** in the second, round, respectively. The reviewing process was facilitated by selfless and dedicated efforts by the PC members (and external reviewers) who collectively did an amazing job providing thorough and thoughtful reviews. Furthermore, some PC members “went the extra mile” by serving as shepherds for papers that required major revisions. The end-result is the total number of **74** accepted submissions, **32** in the first, and **42** in the second, round, respectively.

The 18-session AsiaCCS'23 technical program comprises 74 talks corresponding to accepted papers, a poster session, as well as four impressive keynote talks by internationally prominent and active researchers: N. Asokan, Vanessa Teague, David Basin, and Wenyuan Xu. The program testifies to the level of excellence and the stature of AsiaCCS as the top security venue in the Asia-Pacific region as well as one of the top ones, worldwide.

We offer our deepest gratitude to:

- Every single author of each submission to AsiaCCS'23, whether accepted or not. We thank them for supporting AsiaCCS and for their trust in us and the PC to fairly evaluate their research results.
- The AsiaCCS Steering Committee for their confidence in selecting us as PC Co-Chairs, and their support (especially, by **Jianning Zhou**) throughout the process leading to the conference.
- General Chairs: **Joseph Liu** and **Yang Xiang**, who dealt with (and addressed) numerous logistical and organizational issues.
- Publication Chairs: **Seyit Camtepe** and **Shi-Feng Sun**, for taking care of the proceedings. We especially acknowledge **Seyit** for handling numerous requests from the authors.
- Web Chairs: **Shangqi Lai** and **Sharif Abuadbba** for creating and maintaining the conference website. We are especially indebted to **Sharif** for his extraordinary dedication and the gargantuan amount of work spent on solving a myriad of issues with the website and the submission management system.
- Poster Chairs, **Guangdong Bai** and **Wei Wu**, for taking care of the poster track.
- All PC members and their delegated reviewers, who are the main engine of success and whose hard work yielded the excellent program. Special thanks to the recipients of the “Best Reviewers Award”, **Siqi Ma** and **Alexios Voulimeneas**, for their dedication as both reviewers and shepherds.

In closing, we look forward to the exciting few days in beautiful Melbourne in July and hope that all attendees (physical and remote) enjoy the conference.

PC Co-Chairs
Surya Nepal and Gene Tsudik

Table of Contents

Keynote 1:

Rethinking IoT Security: Understanding and Mitigating Out-of-Band Vulnerabilities	1
<i>Wenyuan Xu (Zhejiang University, China)</i>	

Session 1: Applied Cryptography (I)

Faster TFHE Bootstrapping with Block Binary Keys	2
<i>Changmin Lee (Korea Institute for Advanced Study, South Korea), Seonhong Min (Seoul National University, South Korea), Jinyeong Seo (Seoul National University, South Korea), Yongsoo Song (Seoul National University, South Korea)</i>	
Flag: A Framework for Lightweight Robust Secure Aggregation	14
<i>Laasya Bangalore (Georgetown University, USA), Mohammad Hossein Faghihi Sereshgi (University of Rochester, USA), Carmit Hazay (Bar-Ilan University, Israel), Muthuramakrishnan Venkatasubramaniam (Georgetown University, USA)</i>	
Implementing and Optimizing Matrix Triples with Homomorphic Encryption	29
<i>Johannes Mono (Ruhr University Bochum, Germany), Tim Güneysu (Ruhr University and DFKI GmbH, Germany)</i>	

Session 2: Privacy Application

Invasion of Location Privacy Using Online Map Services and Smartphone Sensors.....	41
<i>Hyunsoo Kim (NCSOFT, Republic of Korea), Youngbae Jeon (Samsung Research, Republic of Korea), Ji Won Yoon (Korea University, Republic of Korea)</i>	
Privacy-Preserving Record Linkage for Cardinality Counting.....	53
<i>Nan Wu (Macquarie University and CSIRO's Data61, Australia), Dinusha Vatsalan (Macquarie University, Australia), Mohamed Ali Kaafar (Macquarie University, Australia), Sanath Kumar Ramesh (OpenTreatments Foundations and CuresDev LLC, USA)</i>	
Investigating Users' Understanding of Privacy Policies of Virtual Personal Assistant Applications	65
<i>Baiqi Chen (The University of Queensland and CSIRO's Data61, Australia), Tingmin Wu (CSIRO's Data61, Australia), Yanjun Zhang (University of Technology Sydney, Australia), Mohan Baruwal Chhetri (CSIRO's Data61, Australia), Guangdong Bai (The University of Queensland, Australia)</i>	

Session 3: Privacy and Machine Learning

RECUP-FL: Reconciling Utility and Privacy in Federated Learning via User-Configurable Privacy Defense	80
<i>Yue Cui (University of Tennessee, US), Syed Irfan Ali Meerza (University of Tennessee, US), Zhuohang Li (University of Tennessee, US), Luyang Liu (Rutgers University, US), Jiaxin Zhang (Intuit AI Research, US), Jian Liu (University of Tennessee, US)</i>	
LDL: A Defense for Label-Based Membership Inference Attacks	95
<i>Arezoo Rajabi (University of Washington, USA), Dinuka Sahabandu (University of Washington, USA), Luyao Niu (University of Washington, USA), Bhaskar Ramasubramanian (Western Washington University, USA), Radha Poovendran (University of Washington, USA)</i>	
Extracting Privacy-Preserving Subgraphs in Federated Graph Learning using Information Bottleneck.....	109
<i>Chenhan Zhang (University of Technology Sydney, Australia), Weiqi Wang (University of Technology Sydney, Australia), James J.Q. Yu (Southern University of Science and Technology, China), Shui Yu (University of Technology Sydney, Australia)</i>	
LoDEN: Making Every Client in Federated Learning a Defender against the Poisoning Membership Inference Attacks	122
<i>Mengyao Ma (The University of Queensland and CSIRO's Data61, Australia), Yanjun Zhang (University of Technology Sydney, Australia), M.A.P. Chamikara (CSIRO's Data61, Australia), Leo Yu Zhang (Griffith University, Australia), Mohan Baruwal Chhetri (CSIRO's Data61, Australia), Guangdong Bai (The University of Queensland, Australia)</i>	
Data Privacy Examination against Semi-Supervised Learning.....	136
<i>Jiadong Lou (University of Louisiana at Lafayette, USA), Xu Yuan (University of Louisiana at Lafayette, USA), Miao Pan (University of Houston, USA), Hao Wang (Louisiana State University, USA), Nian-Feng Tzeng (University of Louisiana at Lafayette, USA)</i>	

Session 4: Architecture Security (I)

Cage4Deno: A Fine-Grained Sandbox for Deno Subprocesses	149
<i>Marco Abbadini (Università degli Studi di Bergamo, Italy), Dario Facchinetti (Università degli Studi di Bergamo, Italy), Gianluca Oldani (Università degli Studi di Bergamo, Italy), Matthew Rossi (Università degli Studi di Bergamo, Italy), Stefano Paraboschi (Università degli Studi di Bergamo, Italy)</i>	
CacheFX: A Framework for Evaluating Cache Security	163
<i>Daniel Genkin (Georgia Institute of Technology, USA), William Kosasih (University of Adelaide, Australia), Fangfei Liu (Intel Labs, USA), Anna Trikalinou (Microsoft, USA), Thomas Unterluggauer (Intel Labs, Austria), Yuval Yarom (Ruhr University Bochum, Germany)</i>	

Multi-Tag: A Hardware-Software Co-Design for Memory Safety based on Multi-Granular Memory Tagging.....	177
<i>Martin Unterguggenberger (Graz University of Technology, Austria), David Schrammel (Graz University of Technology, Austria), Pascal Nasahl (Graz University of Technology, Austria), Robert Schilling (Graz University of Technology, Austria), Lukas Lamster (Graz University of Technology, Austria), Stefan Mangard (Graz University of Technology, Austria)</i>	
FlushTime: Towards Mitigating Flush-based Cache Attacks via Collaborating Flush Instructions and Timers on ARMv8-A.....	190
<i>Jingquan Ge (Southern University of Science and Technology, China), Fengwei Zhang (Southern University of Science and Technology, China)</i>	
ShowTime: Amplifying Arbitrary CPU Timing Side Channels.....	205
<i>Antoon Purnal (imec-COSIC, KU Leuven), Marton Bogнар (imec-DistriNet, KU Leuven), Frank Piessens (imec-DistriNet, KU Leuven), Ingrid Verbauwhede (imec-COSIC, KU Leuven)</i>	

Session 5: Software Security (I)

Symbolic Modeling of Remote Attestation Protocols for Device and App Integrity on Android	218
<i>Abdulla Aldoseri (University of Birmingham and University of Bahrain, UK), Tom Chothia (University of Birmingham, UK), José Moreira (Valory AG, Switzerland), David Oswald (University of Birmingham, UK)</i>	
Arvin: Greybox Fuzzing Using Approximate Dynamic CFG Analysis.....	232
<i>Sirus Shahini (University of Utah), Mu Zhang (University of Utah), Mathias Payer (EPFL), Robert Ricci (University of Utah)</i>	
AbsIntIO: Towards Showing the Absence of Integer Overflows in Binaries using Abstract Interpretation.....	247
<i>Alexander Kűchler (Fraunhofer AISEC, Germany), Leon Wenning (TU Műnchen, Germany), Florian Wendland (Fraunhofer AISEC, Germany)</i>	
Eliminating Vulnerabilities by Disabling Unwanted Functionality in Binary Programs.....	259
<i>Mohamad Mansouri (EURECOM, France), Jun Xu (University of Utah, USA), Georgios Portokalidis (Stevens Institute of Technology, USA)</i>	

Session 6: Hardware Security

Secure and Efficient Mobile DNN Using Trusted Execution Environments.....	274
<i>Bin Hu (Rutgers University), Yan Wang (Temple University), Jerry Cheng (New York Institute of Technology), Tianming Zhao (University of Dayton), Yucheng Xie (Indiana University-Purdue University Indianapolis), Xiaonan Guo (George Mason University), Yingying Chen (Rutgers University)</i>	
Stairway to Rainbow	286
<i>Gildas Avoine (INSA, CNRS, IRISA, France), Xavier Carpent (University of Nottingham, UK), Diane Leblanc-Albarel (CNRS, INSA, IRISA, France)</i>	

EMShepherd: Detecting Adversarial Samples via Side-channel Leakage..... 300

Ruyi Ding (Northeastern University), Cheng Gongye (Northeastern University), Siyue Wang (Northeastern University), Aidong Adam Ding (Northeastern University), Yunsi Fei (Northeastern University)

Electromagnetic Signal Injection Attacks on Differential Signaling 314

Youqian Zhang (University of Oxford, UK), Kasper Rasmussen (University of Oxford, UK)

Keynote 2:

Formal Methods for Payment Protocols 326

David Basin (ETH Zurich, Switzerland)

Keynote 3:

Model Stealing Attacks and Defenses: Where Are We Now? 327

N. Asokan (University of Waterloo, Canada and Aalto University, Finland)

Session 7: Applied Cryptography (II)

On the Cryptographic Fragility of the Telegram Ecosystem 328

Theo von Arx (ETH Zurich, Switzerland), Kenneth G. Paterson (ETH Zurich, Switzerland)

PSI with Computation or Circuit-PSI for Unbalanced Sets from Homomorphic Encryption 342

Yongha Son (Samsung SDS, South Korea), Jinhyuck Jeong (Samsung SDS, South Korea)

ZEKRA: Zero-Knowledge Control-Flow Attestation 357

Heini Bergsson Debes (Technical University of Denmark (DTU)), Edlira Dushku (Aalborg University), Thanassis Giannetsos (Ubitech Ltd), Ali Marandi (Technical University of Denmark (DTU))

Overdrive LowGear 2.0: Reduced-Bandwidth MPC without Sacrifice 372

Pascal Reisert (University of Stuttgart, Germany), Marc Rivinius (University of Stuttgart, Germany), Toomas Krips (University of Tartu, Estonia), Ralf Küsters (University of Stuttgart, Germany)

Session 8: Software Security (II)

Benchmarking the Benchmarks..... 387

Marc Miltenberger (Fraunhofer SIT, Germany), Steven Arzt (Fraunhofer SIT, Germany), Philipp Holzinger (Fraunhofer SIT, Germany), Julius Näumann (Fraunhofer SIT, Germany)

Ember-IO: Effective Firmware Fuzzing with Model-Free Memory Mapped IO 401

Guy Farrelly (The University of Adelaide, Australia), Michael Chesser (The University of Adelaide, Australia), Damith C. Ranasinghe (University of Adelaide, Australia)

RaceBench: A Triggerable and Observable Concurrency Bug Benchmark..... 415

Jiashuo Liang (Peking University, China), Ming Yuan (Tsinghua University, China), Zhanzhao Ding (Peking University, China), Siqi Ma (The University of New South Wales, Australia), Xinhui Han (Peking University, China), Chao Zhang (Tsinghua University, China)

BINWRAP: Hybrid Protection against Native Node.js Add-ons..... 429

George Christou (FORTH-ICS, Greece), Grigoris Ntousakis (Brown University, USA), Eric Lahtinen (Aarno Labs, USA), Sotiris Ioannidis (TU Crete, Greece), Vasileios P. Kemerlis (Brown University, USA), Nikos Vasilakis (Brown University, USA)

Session 9: Architecture Security (II)

Binary Function Clone Search in the Presence of Code Obfuscation and Optimization over Multi-CPU Architectures..... 443

Abdullah Qasem (Concordia University, Canada), Mourad Debbabi (Concordia University, Canada), Bernard Lebel (Thales Research and Technologies, Canada), Marthe Kassouf (Hydro-Québec Research Institute, Canada)

SPEAR-V: Secure and Practical Enclave Architecture for RISC-V..... 457

David Schrammel (Graz University of Technology, Austria), Moritz Waser (Graz University of Technology, Austria), Lukas Lamster (Graz University of Technology, Austria), Martin Unterguggenberger (Graz University of Technology, Austria), Stefan Mangard (Graz University of Technology, Austria)

SFITAG: Efficient Software Fault Isolation with Memory Tagging for ARM Kernel Extensions..... 469

Jiwon Seo (Seoul National University, Republic of Korea), Junseung You (Seoul National University, Republic of Korea), Yungi Cho (Seoul National University, Republic of Korea), Yeongpil Cho (Hanyang University, Republic of Korea), Donghyun Kwon (Pusan National University, Republic of Korea), Yunheung Paek (Seoul National University, Republic of Korea)

An Evaluation Framework for Intrusion Prevention Systems on Serial Data Bus Networks 481

Matthew Rogers (University of Oxford, USA), Kasper Rasmussen (University of Oxford, UK)

Session 10: User-Centric Security (I)

#DM-Me: Susceptibility to Direct Messaging-Based Scams 494

Raj Vardhan (Texas A&M University), Alok Chandrawal (Texas A&M University), Phakpoom Chinprutthiwong (Sisaket Rajabhat University), Yangyong Zhang (Texas A&M University), Guofei Gu (Texas A&M University)

An End-to-End Analysis of Covid-Themed Scams in the Wild 509

Behzad Ousat (Florida International University, USA), Mohammad Ali Tofighi (Florida International University, USA), Amin Kharraz (Florida International University, USA)

MASCARA: Systematically Generating Memorable and Secure Passphrases..... 524

Avirup Mukherjee (IIT Kharagpur), Kousshik Murali (IIT Kharagpur), Shivam Kumar Jha (IIT Kharagpur), Niloy Ganguly (IIT Kharagpur), Rahul Chatterjee (University of Wisconsin–Madison), Mainack Mondal (IIT Kharagpur)

How Secure are the Main Real-World Mix Networks — Case Studies to Explore Vulnerabilities and Usability 539

Kun Peng (Huawei Technology Ltd)

Keynote 4:

Democratizing Election Verification: New Methods for Addressing an Ancient Attacker Model ... 552

Vanessa Teague (Australian National University and Democracy Developers Ltd., Australia)

Session 11: Machine Learning and Security

FLAIR: Defense against Model Poisoning Attack in Federated Learning 553

Atul Sharma (Purdue University, USA), Wei Chen (Purdue University, USA), Joshua Zhao (Purdue University, USA), Qiang Qiu (Purdue University, USA), Saurabh Bagchi (Purdue University, USA), Somali Chaterji (Purdue University, USA)

BFU: Bayesian Federated Unlearning with Parameter Self-Sharing 567

Weiqi Wang (University of Technology Sydney), Zhiyi Tian (University of Technology Sydney), Chenhan Zhang (University of Technology Sydney), An Liu (Soochow University), Shui Yu (University of Technology Sydney)

SoK: Systematizing Attack Studies in Federated Learning – From Sparseness to Completeness 579

Geetanjali Sharma (La Trobe University and CSIRO's Data61, Australia), M.A.P. Chamikara (CSIRO's Data61, Australia), Mohan Baruwal Chhetri (CSIRO's Data61, Australia), Yi-Ping Phoebe Chen (La Trobe University, Australia)

Going Haywire: False Friends in Federated Learning and How to Find Them..... 593

William Aiken (University of Ottawa, Canada), Paula Branco (University of Ottawa, Canada), Guy-Vincent Jourdan (University of Ottawa, Canada)

Deepfake CAPTCHA: A Method for Preventing Fake Calls 608

Lior Yasur (Ben-Gurion University of the Negev, Israel), Guy Frankovits (Ben-Gurion University of the Negev, Israel), Fred M. Grabovski (Ben-Gurion University of the Negev, Israel), Yisroel Mirsky (Ben-Gurion University of the Negev, Israel)

Session 12: Applied Cryptography (III)

A New Look at Blockchain Leader Election: Simple, Efficient, Sustainable and Post-Quantum 623

Muhammed F. Esgin (Monash University & CSIRO's Data61, Australia), Oğuzhan Ersoy (Radboud University and Delft University of Technology, The Netherlands), Veronika Kuchta (Florida Atlantic University, USA), Julian Loss (CISPA Helmholtz Center for Information Security, German), Amin Sakzad (Monash University, Australia), Ron Steinfeld (Monash University, Australia), Xiangwen Yang (Monash University, Australia), Raymond K. Zhao (CSIRO's Data61, Australia)

IGA: An Improved Genetic Algorithm to Construct Weightwise (Almost) Perfectly Balanced Boolean Functions with High Weightwise Nonlinearity 638

Lili Yan (Tianjin University, China), Jingyi Cui (Tianjin University, China), Jian Liu (Tianjin University, China), Guangquan Xu (Tianjin University, China), Lidong Han (Hangzhou Normal University, China), Alireza Jolfaei (Flinders University, Australia), Xi Zheng (Macquarie University, Australia)

FUSE – Flexible File Format and Intermediate Representation for Secure Multi-Party Computation 649

Lennart Braun (Aarhus University, Denmark), Moritz Huppert (Technical University of Darmstadt, Germany), Nora Khayata (Technical University of Darmstadt, Germany), Thomas Schneider (Technical University of Darmstadt, Germany), Oleksandr Tkachenko (DFINITY, Switzerland)

A Trade-off SVP-solving Strategy Based on a Sharper pnj-BKZ Simulator 664

Leizhang Wang (Xidian University, China), Yuntao Wang (Osaka University, Japan), Baocang Wang (Xidian University, China)

Communication-Efficient Inner Product Private Join and Compute with Cardinality 678

Koji Chida (Gunma University, Japan), Koki Hamada (NTT Social Informatics Laboratories, Japan), Atsunori Ichikawa (NTT Social Informatics Laboratories, Japan), Masanobu Kii (NTT Social Informatics Laboratories, Japan), Junichi Tomida (NTT Social Informatics Laboratories, Japan)

Session 13: Adversarial Machine Learning

Jujutsu: A Two-stage Defense against Adversarial Patch Attacks on Deep Neural Networks 689

Zitao Chen (University of British Columbia, Canada), Pritam Dash (University of British Columbia, Canada), Karthik Pattabiraman (University of British Columbia, Canada)

Mitigating Adversarial Attacks by Distributing Different Copies to Different Buyers..... 704

Jiyi Zhang (National University of Singapore, Singapore), Han Fang (National University of Singapore, Singapore), Wesley Joon-Wie Tann (National University of Singapore, Singapore), Ke Xu (Huawei International, Singapore), Chengfang Fang (Huawei International, Singapore), Ee-Chien Chang (National University of Singapore, Singapore)

Boost Off/On-Manifold Adversarial Robustness for Deep Learning with Latent Representation Mixup..... 716

Mengdie Huang (Xidian University, China), Yi Xie (Xidian University, China), Xiaofeng Chen (Xidian University, China), Jin Li (Guangzhou University, China), Changyu Dong (Newcastle University, UK), Zheli Liu (Nankai University, China), Willy Susilo (University of Wollongong, Australia)

DHBE: Data-free Holistic Backdoor Erasing in Deep Neural Networks via Restricted Adversarial Distillation.....	731
---	------------

Zhicong Yan (Shanghai Jiao Tong University, China), Shenghong Li (Shanghai Jiao Tong University, China), Ruijie Zhao (Shanghai Jiao Tong University, China), Yuan Tian (Shanghai Jiao Tong University, China), Yuanyuan Zhao (Hangzhou Normal University, China)

Session 14: Network Security

T-TER: Defeating A2 Trojans with Targeted Tamper-Evident Routing	746
---	------------

Timothy Trippel (University of Michigan, USA), Kang G. Shin (University of Michigan, USA), Kevin B. Bush (MIT Lincoln Laboratory, USA), Matthew Hicks (Virginia Tech, USA)

SPARTA: Signal Propagation-based Attack Recognition and Threat Avoidance for Automotive Networks	760
---	------------

Oleg Schell (Robert Bosch GmbH), Marcel Kneib (Robert Bosch GmbH)

Investigating Traffic Analysis Attacks on Apple iCloud Private Relay	773
---	------------

Ali Zohaib (University of Massachusetts Amherst, USA), Jade Sheffey (University of Massachusetts Amherst, USA), Amir Houmansadr (University of Massachusetts Amherst, USA)

A Honey postMessage, but a Heart of Gall: Exploiting Push Service in Service Workers Via postMessage.....	785
--	------------

Yeomin Jeong (Korea University, Korea), Woonghee Lee (Korea University, Korea), Junbeom Hur (Korea University, Korea)

Session 15: Cloud Security

Secrets Revealed in Container Images: An Internet-wide Study on Occurrence and Impact.....	797
---	------------

Markus Dahlmanns (RWTH Aachen University, Germany), Constantin Sander (RWTH Aachen University, Germany), Robin Decker (RWTH Aachen University, Germany), Klaus Wehrle (RWTH Aachen University, Germany)

Securing Container-based Clouds with Syscall-aware Scheduling	812
--	------------

Michael V. Le (IBM Research, USA), Salman Ahmed (IBM Research, USA), Dan Williams (IBM Research, USA), Hani Jamjoom (IBM Research, USA)

IOTLB-SC: An Accelerator-Independent Leakage Source in Modern Cloud Systems	827
--	------------

Thore Tiemann (University of Lübeck, Germany), Zane Weissman (Worcester Polytechnic Institute, USA), Thomas Eisenbarth (University of Lübeck, Germany), Berk Sunar (Worcester Polytechnic Institute, USA)

Security Properties of Virtual Remotes and SPOOKing their Violations.....	841
--	------------

Joshua Majors (Purdue University, USA), Edgardo Barsallo Yi (Purdue University, USA), Amiya Maji (Purdue University, USA), Darren Wu (Purdue University, USA), Saurabh Bagchi (Purdue University, USA), Aravind Machiry (Purdue University, USA)

Session 16: User-Centric Security (II)

Payment with Dispute Resolution: A Protocol for Reimbursing Frauds Victims	855
<i>Aydin Abadi (University College London, UK), Steven J. Murdoch (University College London, UK)</i>	
Do Users Really Know Alexa? Understanding Alexa Skill Security Indicators	870
<i>Yangyong Zhang (Texas A&M University), Raj Vardhan (Texas A&M University), Phakpoom Chinprutthiwong (Sisaket Rajabhat University), Guofei Gu (Texas A&M University)</i>	
Formalising Application-Driven Authentication and Access-Control Based on Users’ Companion Devices.....	884
<i>Chris Culnane (Castellate Consulting Ltd, UK), Ioana Boureanu (University of Surrey, UK), Jean Snyman (University of Surrey, UK), Stephan Wesemeyer (University of Surrey, UK), Helen Treharne (University of Surrey, UK)</i>	
CRYPTOSHIELD – Automatic On-Device Mitigation for Crypto API Misuse in Android Applications	899
<i>Florian Draschbacher (Graz University of Technology and Secure Information Technology Center, Austria), Johannes Feichtner (Dynatrace Austria GmbH, Austria)</i>	

Session 17: Model Security

QUDA: Query-Limited Data-Free Model Extraction.....	913
<i>Zijun Lin (Nanyang Technological University, Singapore), Ke Xu (Huawei International, Singapore), Chengfang Fang (Huawei International, Singapore), Huadi Zheng (Huawei Technology, China), Jaheezuddin Aneez Ahmed (Nanyang Technological University, Singapore), Jie Shi (Huawei International, Singapore)</i>	
Masked Language Model Based Textual Adversarial Example Detection	925
<i>Xiaomei Zhang (Southwest University, China), Zhaoxi Zhang (Deakin University, Australia), Qi Zhong (Deakin University, Australia), Xufei Zheng (Southwest University, China), Yanjun Zhang (University of Technology Sydney, Australia), Shengshan Hu (Huazhong University of Science and Technology, China), Leo Yu Zhang (Griffith University, Australia)</i>	
CASSOCK: Viable Backdoor Attacks against DNN in the Wall of Source-Specific Backdoor Defenses	938
<i>Shang Wang (Nanjing University of Science and Technology, China), Yansong Gao (CSIRO’s Data61, Australia and Nanjing University of Science and Technology, China), Anmin Fu (Nanjing University of Science and Technology, China), Zhi Zhang (University of Western Australia and CSIRO’s Data61, Australia), Yuqing Zhang (University of Chinese Academy of Sciences, China), Willy Susilo (University of Wollongong, Australia), Dongxi Liu (CSIRO’s Data61, Australia)</i>	
A Transformer-based Function Symbol Name Inference Model from an Assembly Language for Binary Reversing	951
<i>HyunJin Kim (Sungkyunkwan University, South Korea), JinYeong Bak (Sungkyunkwan University, South Korea), Kyunghyun Cho (New York University, USA), Hyungjoon Koo (Sungkyunkwan University, South Korea)</i>	

Session 18: Application Security

ThreadLock: Native Principal Isolation Through Memory Protection Keys.....	966
<i>William Blair (Boston University, USA), William Robertson (Northeastern University, USA), Manuel Egele (Boston University, USA)</i>	
Secure Context Switching of Masked Software Implementations.....	980
<i>Barbara Gigerl (Graz University of Technology, Austria), Robert Primas (Graz University of Technology, Austria), Stefan Mangard (Graz University of Technology, Austria)</i>	
A Scalable Double Oracle Algorithm for Hardening Large Active Directory Systems.....	993
<i>Yumeng Zhang (The University of Adelaide, Australia), Max Ward (University of Western Australia, Australia), Mingyu Guo (The University of Adelaide, Australia), Hung Nguyen (The University of Adelaide, Australia)</i>	
Uncovering Vulnerabilities of Bluetooth Low Energy IoT from Companion Mobile Apps with BLE-GUIDE	1004
<i>Pallavi Sivakumaran (University of London, United Kingdom), Chaoshun Zuo (The Ohio State University, USA), Zhiqiang Lin (The Ohio State University, USA), Jorge Blasco (Universidad Politécnica de Madrid, Spain)</i>	

Posters:

POSTER: A Cyberspace Study of the Russia-Ukraine War	1016
<i>Gursimran Singh (Rochester Institute of Technology, USA), H. B. Acharya (Rochester Institute of Technology, USA)</i>	
POSTER: A Semi-asynchronous Federated Intrusion Detection Framework for Power Systems	1019
<i>Muhammad Akbar Husnoo (Deakin University), Adnan Anwar (Deakin University), Haftu Tasew Reda (Deakin University, Australia), Nasser Hosseinzadeh (Deakin University, Australia)</i>	
POSTER: Toward Intelligent Cyber Attacks for Moving Target Defense Techniques in Software-Defined Networking.....	1022
<i>Tina Moghaddam (The University of Queensland, Australia), Guowei Yang (The University of Queensland, Australia), Chandra Thapa (CSIRO's Data61, Australia), Seyit Camtepe (CSIRO's Data61, Australia), Dan Dongseong Kim (The University of Queensland, Australia)</i>	
POSTER: A Common Framework for Resilient and Safe Cyber-Physical System Design	1025
<i>Luyao Niu (University of Washington, USA), Abdullah Al Maruf (University of Washington, USA), Andrew Clark (Washington University in St. Louis, USA), J. Sukarno Mertoguno (Georgia Institute of Technology, USA), Radha Poovendran (University of Washington, USA)</i>	

POSTER: Leveraging eBPF to Enhance Sandboxing of WebAssembly Runtimes.....	1028
<i>Marco Abbadini (Università degli Studi di Bergamo, Italy), Michele Beretta (Università degli Studi di Bergamo, Italy), Dario Facchinetti (Università degli Studi di Bergamo, Italy), Gianluca Oldani (Università degli Studi di Bergamo, Italy), Matthew Rossi (Università degli Studi di Bergamo, Italy), Stefano Paraboschi (Università degli Studi di Bergamo, Italy)</i>	
POSTER: ML-Compass: A Comprehensive Assessment Framework for Machine Learning Models.....	1031
<i>Zhibo Jin (The University of Sydney and CSIRO's Data61), Zhiyu Zhu (The University of Sydney and CSIRO's Data61), Hongsheng Hu (CSIRO's Data61), Minhui Xue (CSIRO's Data61), Huaming Chen (The University of Sydney)</i>	
POSTER: Performance Characterization of Binarized Neural Networks in Traffic Fingerprinting	1034
<i>Yiyan Wang (The University of Sydney, Australia), Thilini Dahanayaka (The University of Sydney, Australia), Guillaume Jourjon (CSIRO-Space Astronomy, Australia), Suranga Seneviratne (The University of Sydney, Australia)</i>	
POSTER: On Searching Information Leakage of Python Model Execution to Detect Adversarial Examples.....	1037
<i>Cheng-Yao Guo (National Chengchi University, Taiwan), Fang Yu (National Chengchi University, Taiwan)</i>	
POSTER: A Teacher-Student with Human Feedback Model for Human-AI Collaboration in Cybersecurity	1040
<i>Abdullahi Chowdhury (The University of Adelaide, Australia), Debi Ashenden (The University of Adelaide, Australia), Ganna Pogrebna (The University of Sydney, Australia), Hung Nguyen (The University of Adelaide, Australia)</i>	
POSTER: Security Logs Graph Analytics for Industry Network System.....	1043
<i>Qiaoran Meng (National University of Singapore, Singapore), Nay Oo (NCS Cyber Special Ops-R&D, Singapore), Hoon Wei Lim (NCS Cyber Special Ops-R&D, Singapore), Biplab Sikdar (National University of Singapore, Singapore)</i>	

ASIA CCS'23 Organisation

General Chairs: Joseph Liu (Monash University, Australia)
Yang Xiang (Swinburne University of Technology, Australia)

Program Chairs: Surya Nepal (Data61, Australia)
Gene Tsudik (University of California Irvine, US)

Local Organisation

Chairs: Sheng Wen (Swinburne University of Technology, Australia)
Xiao Chen (Monash University, Australia)

Registration and

Finance Chairs: Maggie Liu (RMIT University, Australia)
Xingliang Yuan (Monash University, Australia)

Web Chairs: Sharif Abuadbba (Data61, Australia)
Shangqi Lai (Monash University, Australia)

Publicity Chairs: Siqi Ma (UNSW Canberra, Australia)
Sushmita Ruj (UNSW Sydney, Australia)

Publication Chairs: Seyit Camtepe (Data61, Australia)
Shi-Feng Sun (Shanghai Jiao Tong University, China)

Workshop Chairs: Hyounghick Kim (Sungkyunkwan University, Korea)
Shabnam Kasra (UNSW Canberra, Australia)

Poster Chairs: Guangdong Bai (University of Queensland, Australia)
Wei Wu (Fujian Normal University, China)

Tutorial Chairs: Ahmad Salehi Shahraki (La Trobe University, Australia)
Shujie Cui (Monash University, Australia)

Steering Committee

Chair: Jianying Zhou (Singapore University of Technology and Design, Singapore)

Steering Committee: Gail-Joon Ahn (Arizona State University, USA)
Robert Deng (Singapore Management University, Singapore)
Adrian Perrig (ETH Zürich, Switzerland)
Kui Ren (Zhejiang University, China)
Shiuhpyng Shieh (National Chiao Tung University, Taiwan)
Xiaofeng Wang (Indiana University Bloomington, USA)

**Program
Committee:**

Alexios Voulimeneas, KU Leuven, Belgium
Alptekin K p u, Ko University, Turkey
Angelos Stavrou, Virginia Tech, USA
Anna Lisa Ferrara, University of Molise, Italy
Annabelle McIver, Macquarie University, Australia
Aravind Machiry, Purdue University, USA
Ashish Kundu, Cisco Research, USA
Bet l Durak, Microsoft Research, USA
Bo Chen, Michigan Technological University, USA
Bruno Crispo, University of Trento, Italy
Chao Zhang, Tsinghua University, China
Christopher Wood, Cloudflare, USA
Cristian-Alexandru Staicu, CISP Helmholtz Center, Germany
Cristina Alacraz, University of Malaga, Spain
Damith Ranasinghe, University of Adelaide, Australia
Daniele Antonioli, EURECOM, France
Debdeep Mukhopadhyay, Indian Institute of Technology, Kharagpur, India
Debin Gao, Singapore Management University, Singapore
Di Ma, University of Michigan-Dearborn, USA
Dokyung Song, Yonsei University, Korea
Doowon Kim, University of Tennessee, USA
Edgar Weippl, University of Vienna & SBA Research, Austria
Elisabeth Oswald, University of Klagenfurt, Austria
Gabriele Oligeri, Hamad bin Khalifa University, Qatar
Ghassan Karame, Ruhr University Bochum, Germany
Giovanni Russello, University of Auckland, New Zealand
Guomin Yang, University of Wollongong, Australia
Haehyun Cho, Soongsil University, Korea
Herv  Debar, T l com SudParis, Institut Polytechnique de Paris, France
Hyoungshick Kim, Sungkyunkwan University, Korea
Ileana Buhan, Radboud University Nijmegen, Netherlands
Jie Yang, Florida State University, USA
Jin-Hee Cho, Virginia Tech, USA
Joaquin Garcia-Alfaro, Institut Polytechnique de Paris, France
Jorge Blasco Alis, Royal Holloway, University of London, UK
Jorge Guajardo, Robert Bosch LLC – Research and Technology Center, USA
Juanru Li, Shanghai Jiao Tong University, China
Jun Sakuma, University of Tsukuba, Japan
Junghwan “John” Rhee, University of Central Oklahoma, USA
Kapil Singh, IBM T.J. Watson Research Center, USA
Kari Kostianen, ETH Zurich, Switzerland
Kasper Rasmussen, University of Oxford, UK
Katerina Mitrokotsa, University of St. Gallen, Switzerland
Katsunari Yoshioka, Yokohama National University, Japan
Kiran Balagani, New York Institute of Technology, USA
Kun Sun, George Mason University, USA
Kwok-Yan Lam, Nanyang Technological University, Singapore
Lorenzo De Carli, Worcester Polytechnic Institute, USA
Luca Vigan , King’s College London, UK
Lucas Davi, University of Duisburg-Essen, Germany

Lucca Hirschi, Inria, France
Mads Dam, KTH Royal Institute of Technology, Sweden
Mahmoud Ammar, Huawei Research, Germany
Man Ho Au, University of Hong Kong, Hong Kong
Mark Manulis, Universität der Bundeswehr München, Germany
Mathieu Cunche, INSA-Lyon / Inria, France
Mathy Vanhoef, KU Leuven, Belgium
Melek Önen, EURECOM, France
Michael Sirivianos, Cyprus University of Technology, Cyprus
Mitsuaki Akiyama, NTT, Japan
Miyako Ohkubo, NIICT, Japan
Mohammad Mannan, Concordia University, Canada
Mu Zhang, University of Utah, USA
Muhammad Ikram, Macquarie University, Australia
Muhammed Esgin, Monash University, Australia
Ning Zhang, Washington University in St. Louis, USA
Nuno Santos, INESC-ID / Instituto Superior Técnico, Portugal
Olga Gadyatskaya, Leiden University, The Netherlands
Qi Li, Tsinghua University, China
Qiang Tang, University of Sydney, Australia
Rahmadi Trimananda, University of California, Irvine, USA
Reza Curtmola, New Jersey Institute of Technology, USA
Rishab Nithyanand, University of Iowa, USA
Roberto Guanciale, KTH, Sweden
Rolf Oppliger, eSECURITY Technologies, Switzerland
Sanjay Jha, UNSW, Australia
Satoshi Obana, Hosei University, Japan
Satyanarayana Vusirikala, DFINITY, USA
Saurabh Bagchi, Purdue University & KeyByte, USA
Selcuk Uluagac, Florida International University, USA
Seung Geol Choi, US Naval Academy, USA
Shabnam Kasra Kermanshahi, RMIT, Australia
Shuohuai Xu, University of Colorado Colorado Springs, USA
Siqi Ma, UNSW, Australia
Stefan Katzenbeisser, University of Passau, Germany
Steven Galbraith, University of Auckland, New Zealand
Sven Dietrich, City University of New York, USA
Tatsuya Mori, Waseda University, Japan
Thorsten Strufe, KIT, Germany
Tingmin Wu, Data61, CSIRO, Australia
Vanessa Daza, Pompeu Fabra University, Spain
Veelasha Moonsamy, Ruhr University Bochum, Germany
William Robertson, Northeastern University, USA
Willy Susilo, University of Wollongong, Australia
Xavier Carpent, University of Nottingham, UK
Xiapu Luo, Hong Kong Polytechnic University, China
Xingliang Yuan, Monash University, Australia
Youngee Park, San Jose State University, USA
Yuan Hong, Illinois Institute of Technology, USA
Zhi Zhang, Data61, CSIRO, Australia
Zhen Huang, DePaul University, USA

**External
Reviewers:**

Ziyao Liu, NTU, Singapore
Feng Li, NTU, Singapore
Jiabo Wang, NTU, Singapore
Jiani Fan, NTU, Singapore
Niusen Chen, Michigan Technological University, USA
Caleb Rother, Michigan Technological University, USA
Harsh Singh, Michigan Technological University, USA
Weijing You, Fujian Normal University, China
Xinyu Zhang, Monash University, Australia
Shangqi Lai, Monash University, Australia
Maxime Buser, Kudelski Security, Switzerland
Mafalda Ferreira, INESC-ID / Instituto Superior Técnico, Portugal
Daniela Lopes, INESC-ID / Instituto Superior Técnico, Portugal
Íris Damião, LIP Lisboa / Instituto Superior Técnico, Portugal
Jia Liu, ENYA Labs, UK
Daniel Klischies, Ruhr University Bochum, Germany
Philipp Mackensen, Ruhr University Bochum, Germany
Abbas Acar, Florida International University, Germany
Shawn Emery, UCCS, USA
Ekzhin Ear, UCCS, USA
Rosana Montanez Rodriguez, UCCS, USA
Takashi Nishide, University of Tsukuba, Japan
Kazuma Ohara, NIAIST, Japan
Shi Bai, Florida Atlantic University, USA
Kelong Cong, KU Leuven, Belgium
Duhyeong Kim, Intel, USA
Yi-Fu Lai, University of Auckland, New Zealand
Frederik Vercauteren, KU Leuven, Belgium
Joshua Zhao, Purdue University, USA
Chris Gutierrez, Intel, USA
Fahad Arshad, VMWare, USA
Ahaan Dabholkar, Purdue University, USA
Johannes Ernst, University of St. Gallen, Switzerland
Shihua Sun, Virginia Technology, USA
Tolga Atalay, Virginia Technology, USA
Akira Kanaoka, Toho University, Japan
Daniel Collins, EPFL, Switzerland
Koji Nuida, Kyushu University, Japan
Haiyang Xue, University of Hong Kong, Hong Kong
Kexin Hu, Chinese Academy of Sciences, China
Nadeem Ahmed, UNSW, Australia
Ryo Lijima, Waseda University, Japan
Sayntan Mukherjee, University of St. Gallen, Switzerland
Srinath Setty, Microsoft Research, USA
Takuya Watanabe, Waseda University, Japan
Tapas Pal, NTT Corporation, Japan
Hanwen Feng, University of Sydney, Australia
Jason (Minhui) Xue, CSIRO's Data61, Australia
Jawad Ahmed, UNSW, Australia

Junichi Tomida, NTT, Japan
Kazuki Yoneyama, Ibaraki University, Japan
Kazuma Ohara, AIST, Japan
Keewoo Lee, Seoul National University, Korea
Kristen Moore, CSIRO's Data61, Australia
Masaya Yasuda, Rikkyo University, Japan
Shuo Wang, CSIRO's Data61, Australia
Takasha Nishide, Tsukuba University, Japan
Jeremy D. Seideman, American Express, USA
Jiakun Liu, Singapore Management University, Singapore

Poster

Reviewers: Naipeng Dong, Qinglei Kong, Juanru Li, Chao Lin, Chamikara Mahawaga Arachchige, Mark Huasong Meng, Jianting Ning, Qiang Tang, Sin Gee Teo, Viet Vo, Kailong Wang, Guowei Yang, Leo Yu Zhang, Ying Zhang

ASIA CCS'23 Sponsor and Supporters

Sponsor:



**Platinum
Supporter:**



**Gold
Supporters:**



MONASH
INFORMATION
TECHNOLOGY

MONASH
SOFTWARE
SYSTEMS AND
CYBERSECURITY

**Bronze
Supporters:**



LinkStone
Creating More Secure Identity Management



Cage4Deno: A Fine-Grained Sandbox for Deno Subprocesses

Marco Abbadini
marco.abbadini@unibg.it
Università degli Studi di Bergamo
Bergamo, Italy

Dario Facchinetti
dario.facchinetti@unibg.it
Università degli Studi di Bergamo
Bergamo, Italy

Gianluca Oldani
gianluca.oldani@unibg.it
Università degli Studi di Bergamo
Bergamo, Italy

Matthew Rossi
matthew.rossi@unibg.it
Università degli Studi di Bergamo
Bergamo, Italy

Stefano Paraboschi
stefano.paraboschi@unibg.it
Università degli Studi di Bergamo
Bergamo, Italy

ABSTRACT

Deno is a runtime for JavaScript and TypeScript that is receiving great interest by developers, and is increasingly used for the construction of back-ends of web applications. A primary goal of Deno is to provide a secure and isolated environment for the execution of JavaScript programs. It also supports the execution of subprocesses, unfortunately without providing security guarantees.

In this work we propose *Cage4Deno*, a set of modifications to Deno enabling the creation of fine-grained sandboxes for the execution of subprocesses. The design of Cage4Deno satisfies the compatibility, transparency, flexibility, usability, security, and performance needs of a modern sandbox. The realization of these requirements partially stems from the use of Landlock and eBPF, two robust and efficient security technologies. Significant attention has been paid to the design of a flexible and compact policy model consisting of RWX permissions, which can be automatically created, and deny rules to declare exceptions. The sandbox effectiveness is demonstrated by successfully blocking a number of exploits for recent CVEs, while runtime experiments prove its efficiency. The proposal is associated with an open-source implementation.

CCS CONCEPTS

• Security and privacy → Software and application security; Access control; File system security; Usability in security and privacy.

KEYWORDS

Sandboxing, Access control, JavaScript runtime, Deno, Subprocess

ACM Reference Format:

Marco Abbadini, Dario Facchinetti, Gianluca Oldani, Matthew Rossi, and Stefano Paraboschi. 2023. Cage4Deno: A Fine-Grained Sandbox for Deno Subprocesses. In *ACM ASIA Conference on Computer and Communications Security (ASIA CCS '23)*, July 10–14, 2023, Melbourne, VIC, Australia. ACM, New York, NY, USA, 14 pages. <https://doi.org/10.1145/3579856.3595799>



This work is licensed under a Creative Commons Attribution International 4.0 License.

ASIA CCS '23, July 10–14, 2023, Melbourne, VIC, Australia
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0098-9/23/07.
<https://doi.org/10.1145/3579856.3595799>

1 INTRODUCTION

JavaScript is currently one of the most popular programming languages [68]. One of its strengths is versatility, indeed, it can be used both in the front-end and in the back-end to write fully fledged web applications. JavaScript was originally meant for the browser, and its porting on the back-end by *Node.js* is not without security risks. As highlighted in previous studies [24, 69, 74, 75, 78], vulnerabilities affecting the language or the toolchain can lead to severe breaches.

A recent initiative aiming to reduce the risk coming from the execution of vulnerable or untrusted JavaScript code on the back-end is *Deno* [44], a modern, secure, open-source, cross-platform JavaScript runtime. Contrary to its well-known predecessor (i.e., *Node.js*), Deno was designed with security as one of its primary goals [17]. This aspect partly stems from the programming language used to implement it (i.e., Rust instead of C++), but mostly originates from its default behavior of executing JavaScript code in a completely isolated sandbox. Unless otherwise stated by the developer, Deno prevents any program from accessing the filesystem, network, environment variables, and even high-resolution time measurements. Unfortunately, although its permission model allows developers to grant fine-grained authorizations (read/write/execute privilege on a single file, permissions to connect to a known hostname, etc.), dynamic libraries and subprocesses can access system resources regardless of the permissions granted to the Deno program that spawned them, essentially invalidating the security sandbox [19].

Research has shown that third-party code accounts for a great portion of a JavaScript application codebase [74]. The 2022 State of Open Source Security [66] shows that on average open-source JavaScript projects rely on 174 third-party dependencies. The practice of reusing third-party libraries is so widespread that it led the authors of Deno to implement the *Node compatibility mode* [21], which enables the execution of Node packages in Deno. Third-party modules often depend on subprocesses [23, 32, 69]. There is a broad spectrum of programs that fall into this category, ranging from Linux utilities to handle files [56], to programs that process media (e.g., image/video conversions [55], metadata removal [54]), and many more. Since these programs (i) are executed outside of a sandbox, (ii) are potentially exposed to unsanitized input, and (iii) are often written with unsafe languages, the risk of security violations is concrete. The 2022 State of Open Source Security reports that, on average, JavaScript projects are affected by 40 vulnerabilities when dependencies are taken into account. We focused on the records of the last 5 years, and have identified a sample of 15 high-severity

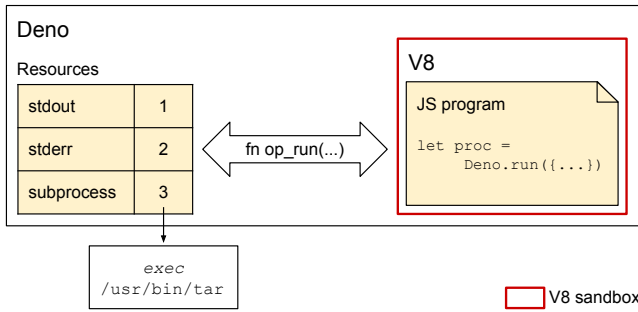


Figure 1: High level view of the Deno architecture

CVEs that affect third-party software used by popular packages (averaging more than 1M downloads/week), and allow an attacker to perform privilege escalation on the host, open reverse shells, perform local file inclusion, and corrupt the filesystem.

Our contribution. In this work we propose *Cage4Deno*: an extension of the Deno security functions aimed at mitigating vulnerabilities that may be introduced when subprocesses are used. Specifically, we design and implement¹ a solution targeting Linux systems, which leverages the recent *Landlock* Linux Security Module and the *Extended Berkeley Packet Filter* (eBPF), to transparently sandbox processes currently executed outside of the sandbox by Deno. The primary goal of the proposed protection mechanism is to preserve the integrity of the filesystem, and preventing access to confidential resources. While designing *Cage4Deno*, we paid great attention to the usability by developers: (i) no understanding of the kernel security features leveraged by our solution is required to use it, and (ii) although developers should be knowledgeable about the functionalities of the program they want to use, we do not expect them to be fully aware of its functioning under the hood. To this end, we first design a flexible and compact policy model. It consists of rules granting read (R), write (W) or exec (X) permissions on a filesystem node, which propagate towards its descendants, and deny rules (D) to block the propagation, thus reducing the number of rules and the effort of the developer. Then, we implement an auxiliary open-source tool named *dmng* to generate the least-privileged RWX rules required by each program. The tool can be invoked interactively by the developer via CLI, or integrated into CI/CD pipelines and run against a set of use cases, as best practice suggests. We also highlight that *Cage4Deno* preserves backward compatibility, meaning sandboxing of existing code, including direct or transient dependencies, can be achieved without code changes.

We demonstrate the security benefits of our solution showing how it mitigates a number of real-world vulnerabilities, which have been exploited against popular services (e.g., GitLab [14] and TikTok [36]). Finally, we perform an experimental evaluation comparing our solution to scenarios with no sandboxing, and sandboxing with state-of-the-art proposals, showing substantial performance improvements with respect to the available alternatives.

2 BACKGROUND

This section overviews the frameworks used in our proposal.

¹<https://github.com/unibg-seclab/cage4deno>

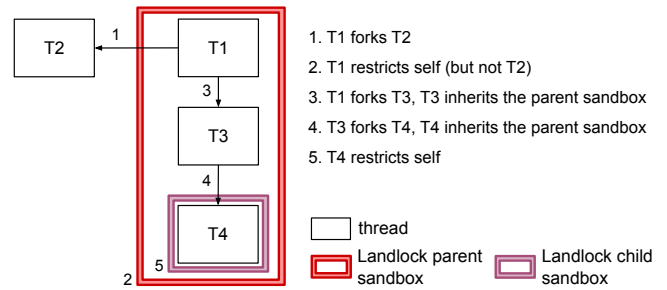


Figure 2: Creation and inheritance of a Landlock sandbox

2.1 Deno

Deno [44] is a runtime for JavaScript and TypeScript embedding *V8* [72], an open-source high-performance JavaScript and WebAssembly engine by Google. As already mentioned, Deno can be seen as an alternative to *Node.js*, with some key security features that derive from its design. In detail, it has a modular architecture organized into three main components: (i) *rusty_v8*, which defines the set of bindings to the *V8*'s API; (ii) *deno_core*, a package built on top of *rusty_v8* implementing the abstractions required to run JavaScript (i.e., the *JsRuntime*); and (iii) *deno*, a package providing the Deno executable and the user-facing API.

Among the abstractions implemented in *deno_core* there are *ops*, native functions directly called from JavaScript that expose services not directly available in *V8*. These include primitives to open files, create network sockets, spawn child processes, access environment variables, etc. From a security perspective, each of the requests issued by the program running inside the *V8* sandbox (i.e., the *op* calls) can be monitored by Deno and explicitly blocked or authorized based on a set of permissions. By default apps run without any permission, that is why Deno claims to be secure by default. The permission system categorizes the resources based on their type (filesystem, environment variables, network, etc.). There are two granularity levels: access to the whole resource category (e.g., all the files on the filesystem), and access to a single resource in the category (e.g., a single file). The resources accessible through *ops* are stored by *deno_core* in a resource table, and are uniquely identified by integers, similarly to the Unix concept of file descriptor.

The *ops* and resources interface abstraction gives Deno the ability to control the information flow between the system and the sandboxed app. This is a big step forward compared to *Node.js* in terms of security. Unfortunately, this level of protection applies only to the components written in JavaScript. When the app executes a program on the host leveraging for example the `Deno.run()` function, the request is handled by Deno spawning a new subprocess. As depicted in the high level view of the Deno architecture shown in Figure 1, this process runs unconstrained on the host (without sandboxing). This is a limitation of the current security model as, in case of a vulnerability, the host can be compromised [17].

2.2 Landlock LSM

The *Landlock* Linux Security Module (LSM) aims to provide developers with an unprivileged solution to implement application sandboxing. The availability of *Landlock* is expected to help mit-

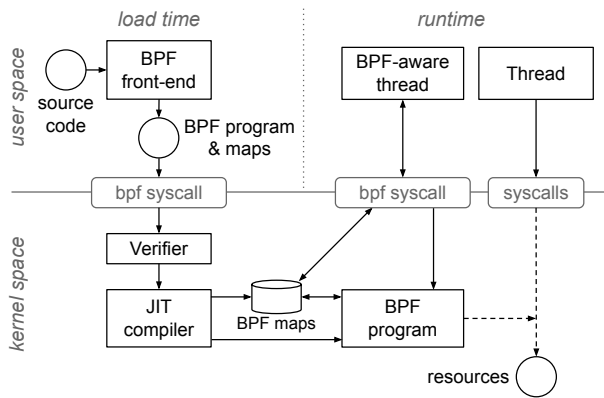


Figure 3: Overview of the BPF architecture

igate the security impact of bugs and unintended or malicious behavior of user-space applications. Also, it is fully composable with other LSMs like SELinux, AppArmor and Yama.

Currently, Landlock focuses on the protection of the filesystem. In detail, Landlock classifies kernel resources as *objects*, permitting *actions* over them based on *rulesets*. As an example, a ruleset can grant the thread read and exec access to objects stored under /tmp. Rulesets are created using the `landlock_create_ruleset()` and `landlock_add_rule()` system calls, and subsequently activated with `landlock_restrict_self()`. Rules are *inherited* by the children created after sandbox activation, and the actions available can only be further restricted. The process is detailed in Figure 2.

Although Landlock permits to sandbox a thread ensuring low performance overhead, there are a few limitations. The most relevant to this paper are: (i) it is not possible to perform any filesystem topology modification (i.e., arbitrary mounts), and (ii) there is no support for a deny listing approach during policy creation.

2.3 eBPF

The *Extended Berkeley Packet Filter* (eBPF, henceforth referred to as BPF) [33, 41] enables the execution of programs within the operating system kernel. The programs, which are loaded at runtime, extend the kernel capabilities, without requiring the developer to change the kernel source code, nor loading new kernel modules.

BPF programs are non-preemptable event-driven programs that are run when a certain user- or kernel-space hook point is reached. Pre-defined hooks include system calls, tracepoints, network events, function entry/exit, etc. BPF programs are usually written using a BPF front-end, which provides an abstraction to write programs in a high-level language, specify attachment points, declare data structures, and compile the source code into BPF bytecode. A few BPF front-ends exist; we use *libbpf* [47, 51] as it elegantly addresses portability following a Compile Once – Run Everywhere approach [51]. After a BPF program is compiled to bytecode, it can be loaded into the Linux kernel via the `bpf()` system call. This is a privileged operation that requires the `CAP_BPF` Linux capability, and optionally `CAP_PERFMON` (to load tracing-related programs) and `CAP_NET_ADMIN` (to load networking-related programs) [1]. As the program is loaded into the kernel, it is subject to the *verification* and *JIT compilation* phases. The verification phase ensures the

program is safe and does not introduce reliability issues (e.g., termination is guaranteed, memory requirements are satisfied), while the Just-In-Time (JIT) compilation translates the generic bytecode to architecture-specific optimized code. Once completed, the program is loaded into the kernel and attached to the selected hook. The architecture of BPF and the loading process are shown in Figure 3.

BPF programs cannot call arbitrary kernel functions and cannot freely share the information collected with user space. Instead, they rely on *helper functions*, a stable API implemented by the kernel that is used for tasks like manipulating network packets, inspecting kernel data structures, etc. Among the most frequently used helpers, there are functions to read and write *maps*. Basically, maps are data structures that permit to keep a state between different invocations of BPF programs, and share data with user-space applications.

BPF’s capabilities have been further extended in 2020 with the addition of the Kernel Runtime Security Instrumentation (KRSI) [13], also known as BPF LSM. This feature permits to attach BPF programs to LSM hooks, and thus enforce access control.

3 CAGE4DENO

This Section gives an overview of *Cage4Deno*, clarifying the threat model. We also introduce our design objectives (Section 3.3).

3.1 Overview

Cage4Deno aims to provide a set of sandboxing functions to strengthen the Deno security model. The proposal arises from the need to provide isolation for subprocesses spawned with the `Deno.run()` and `Deno.Command.spawn()` functions. As mentioned in Section 1, the practice of executing subprocesses is heavily used by developers. Unfortunately, any subprocess that executes this way falls outside of the Deno security model, essentially invalidating the sandbox (see Section 2.1). Indeed, an attacker that successfully exploits a security flaw affecting the utility run by a subprocess can perform privilege escalation on the host, open reverse shells, perform local file inclusion, corrupt the filesystem, etc.

Cage4Deno extends Deno giving developers the ability to constrain the execution of subprocesses. To do that, the developer associates each subprocess with a policy file, listing a set of rules. Rules are straightforward, each of them granting read (R), write (W) or exec (X) permissions on a filesystem node. To reduce the number of rules in the policy, the set of RWX permissions is extended with deny (D). This enables permissions granted on a filesystem node to propagate towards its descendants, and to block the propagation with the use of deny rules. The permissions granted by the developer are then automatically assigned to the subprocess at runtime. This is achieved by the *sandboxer*, a new module we added to Deno that leverages the Landlock LSM and the BPF framework (operating in stacking mode [79]) to implement the sandbox at kernel level. This ensures security checks are not bypassable, however it implies that our solution only works on Linux-based systems with these features enabled. Nowadays, Landlock LSM is already available in most systems, BPF LSM not yet. However, bleeding edge distributions, like Arch Linux, are starting to release with it enabled by default.² Compared to other well-known general-purpose

²Enabling BPF LSM on systems not supporting it by default requires replacing the system kernel with the same release of the kernel, but with BPF LSM available.

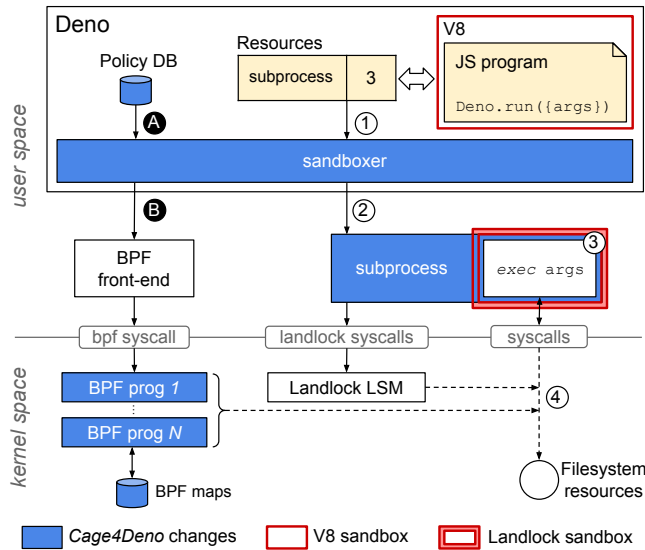


Figure 4: Overview of Cage4Deno implementation. Initial setup: (A) read the policy and (B) load BPF programs and write maps according to it. **On subprocess request:** (1) intercept subprocess creation, (2) create a Landlock-sandboxed subprocess with BPF deny, and (3) execute the utility. All the requests issued by the subprocess are restricted according to the policy (4)

sandboxing solutions like Minijail [30], and Google Sandbox2 [31], Cage4Deno does not require the developer to know anything about the advanced security features offered by the kernel to confine a process. All the developer has to understand is the straightforward RWX+D permission model. Moreover, Cage4Deno does not require the developer to understand the internal logic of the utility executed by the subprocess, all she needs to know are its input and output (provided by either absolute or relative path). An overview of our proposal is shown in Figure 4.

3.2 Threat model

Similarly to other research proposals [16, 70, 75], Cage4Deno focuses on the runtime compromise of possibly buggy or vulnerable utilities (i.e., binaries or libraries), and does not target actively malicious ones. Therefore, we do not regard the developers of the utilities as malicious actors, but nevertheless, they may inadvertently introduce vulnerabilities into their code. In this setting, attackers may control arguments of the utilities by passing malicious payloads through web interfaces or programmatic APIs. Prominent examples include utilities that offer manipulation capabilities of multimedia files (e.g., ExifTool, Ffmpeg, GraphicsMagick, ImageMagick), or object de-compression (e.g., GNU Tar). These programs are usually written in memory-unsafe languages, such as C/C++, and due to their size and complexity are often the source of vulnerabilities [52]. The constant and extensive exposure to untrusted inputs in web applications may enable the attacker to trigger these vulnerabilities, leading to memory corruption issues. Arbitrary file reading or writing, local file inclusion and remote code execution are only a few of

the possible risks associated with this kind of attack vector. In addition to that, package managers for languages commonly used in web development (e.g., npm, pip) do not enforce any kind of permission system. Thus, attackers can exploit vulnerabilities introduced by direct or indirect dependencies installed when 3rd party modules are imported by the application. Previous studies [15, 50, 80] have reported the risks associated with the propagation of vulnerabilities coming from dependencies in various software ecosystems. Usually, the exploitation of this kind of vulnerability leads to breaches that undermine confidentiality and integrity of data and code that reside outside the utility under attack. The goal of Cage4Deno is preventing a compromised utility running inside a Deno subprocess to violate access confidentiality and integrity of the filesystem. Notice that preventing integrity violation of the filesystem means, first and foremost, preserving the integrity of the web application code, configurations and data.

3.3 Design objectives

Several objectives need to be fulfilled by our proposal to ensure security, efficiency and ease of use.

- 01. Integration with existing solutions (Compatibility):** The proposal must be compatible with the current Deno architecture, and should be considered as an opt-in feature by the developer (i.e., backward compatible). Also, it has to be stackable with other security mechanisms already active on the host, like SELinux, AppArmor and Yama.
- 02. Ease of use (Transparency):** To be aligned with the needs of the web development scenario, the solution must be simple and easy to use. No specific understanding of the advanced security features leveraged by our solution, and of the underlying operating system, must be required to use it.
- 03. Fine-grained access control (Flexibility):** Unlike other security features that create a separated view of global resources (i.e., *Linux namespaces*), the solution must enable the developer to access the whole filesystem, granting access with file-level granularity rather than volume-level. Also, the developer must be able to flexibly switch between different policies, without requiring a host reboot when a new policy needs to be loaded.
- 04. Automatic generation of policies (Usability):** The proposal must provide tools to support zero-effort policy creation. With the exception of its input and output, no specific understanding of each utility should be required to run it successfully.
- 05. Mitigation of vulnerabilities (Security):** The proposal must prove effective in addressing the threat model described in Section 3.2 and mitigate CVEs affecting common Linux utilities.
- 06. Low overhead (Performance):** The overhead introduced with the additional security features must be compatible with the requirements of Web applications, which tend to prioritize short response time. It should be lower than the one introduced by state-of-the-art general-purpose sandboxing solutions, such as Minijail and Google Sandbox2.

4 DESIGN AND IMPLEMENTATION

This Section illustrates the design and implementation of Cage4Deno. We initially describe the interface used by the developer to input the policy, then we detail the changes introduced to support read,

write, exec and deny permissions. A minimal program executing the tar utility in a subprocess is used as running example. The approach used to automatically generate the policy to confine tar is explained in Section 5, while additional BPF implementation details that focus on performance improvement are discussed in Appendix A.

4.1 Policy and interface

In the current Deno architecture, attributes and permissions associated with a program are always specified prior to execution. The developer provides this information through the Deno command line interface (CLI) using *runtime flags*. Runtime flags of the deno run, which precede the program name in the argument list, are immediately parsed and added to the global state before the JS program is executed. This design philosophy permits to directly address access requests issued by the application at runtime and, in case no permission is available, the program is promptly terminated. As an example, consider the program shown in Listing 1. Its purpose is simply to execute the tar utility to extract the compressed archive input.tgz to the output directory. Since to execute tar it leverages a subprocess, the argument `--allow-run=tar` must be provided, as the lack of such a permission would lead to the operation being prohibited.

As explained in the overview (Section 3.1), Cage4Deno relies on the ability to attach a policy to the program. To be compliant with the current design of Deno, we extended the global state adding the `--policy-file` runtime flag. As the name suggests, it receives as input a policy file. The policy file uses a JSON format, which is easy to edit and parse, and well-known by web developers. The file contains an array of policies identified by a `policy_name`. Each policy includes four arrays, `read`, `write`, `exec` and `deny`, listing a set of filesystem entries. As shown in Figure 4, the life cycle of Cage4Deno can be divided into two phases: load time and enforcing time. When Cage4Deno starts, the content of the policy file is read by the *sandboxer* (A); to avoid repeating this step each time a new subprocess is spawned, the rule sets are stored in the permission state of Deno. Once all policies have been parsed, if any of them contains negative rules, the *sandboxer* loads the set of BPF programs needed to enforce deny, and a map for each policy containing the corresponding prohibited paths (B). Due to this additional step, this is the only part of the execution in which Cage4Deno requires additional privileges compared to Deno. These privileges consist of the set of Linux capabilities necessary to load BPF components. To avoid any additional interaction the capabilities are configured as file capabilities. Once every BPF component has been loaded, we drop the capabilities, so to avoid running with higher privileges with respect to Deno at runtime.

After the initial setup, Cage4Deno is responsible of enforcing the policies provided by the developer. The *sandboxer* intercepts subprocess creation requests issued by the JS application at runtime (C) and, according to the policies available, it restricts the permissions associated with the subprocess (D) before the `exec` of the command provided to the `Deno.run()` is performed (E). During the subprocess (or any of its children) lifetime, file interactions are controlled by Landlock and our BPF programs to make sure access to the requested path is granted according to the policy definition

Listing 1: An example of child process in Deno

```
1 let a=Deno.run({cmd: ["tar", "xzf", "input.tgz", "-C", "output"]});
2 await a.status();
```

Listing 2: Restricting a child process using policyId

```
1 let a=Deno.run({cmd: ["tar", "xzf", "input.tgz", "-C", "output"], policyId:"tarPolicy"});
2 await a.status();
```

provided by the developer (4). Two options are available to retrieve proper policy from the list: (i) automatically select the one with policy name matching the utility to be run in the subprocess (e.g., tar in Listing 1), and (ii) using the policy directly specified by the developer in the JS code using the newly introduced `policyId` option as shown in Listing 2. In both cases, if no policy is found, we fall back to the default Deno permission model. Option (i) allows the developer to run the subprocess without modifications to the JS program, while option (ii) gives more flexibility when multiple policy profiles for the same utility are available. This interface is straightforward and is perfectly aligned with the current Deno security model and architecture (Objective O1). Moreover, it does not require the developer to understand how the *sandboxer* leverages the kernel security features to add the restrictions, while granting the developer direct observability of the security boundaries put in place (Objective O2).

4.2 Support to RWX rules

Deno permits to set filesystem-related permissions through the `allow-read`, `allow-write` and `allow-run` runtime flags. As explained in the background (Section 2.1), the flags can be used to configure access to the whole filesystem, or can be refined specifying a list of comma separated filesystem entries. Similarly, we expect the developer to detail the `read`, `write` and `exec` permissions inside the JSON policy file. Listing 3 exemplifies the RWX permissions associated with the tar example.

The support for RWX permissions was introduced to Deno changing the implementation of *deno_core*. Internally, when a program performs a call to the `Deno.run()` and `Deno.Command.spawn()` functions, the bindings defined in *rusty_v8* are used to translate the request into an *op_run*. The operation is subsequently sent to the JS runtime, which is responsible to manage the request. In the case of a subprocess creation, this is done leveraging the asynchronous Tokio [22] runtime to configure an instance of `Command`, a process builder providing fine-grained control over how the process should be spawned. To apply the correct policy, we modified process creation, scheduling a closure (Rust equivalent of a C++ lambda expression) to be run just before the `exec` function is invoked. The closure leverages our *sandboxer* to create a new Landlock ruleset consisting of the RWX permissions found in the policy, and then invokes the `landlock_restrict_self()` syscall to apply it. Thanks to the policy inheritance property of Landlock, the policy attached to the process also restricts its children (as explained in Section 2.2). Should any of them try to access a filesystem location not listed in the policy, the request is denied.

Listing 3: RWX+D permissions associated with tar

```

1 {
2   "policies": [
3     {
4       "policy_name": "tarPolicy",
5       "read": [
6         "/usr/bin/tar",
7         "/home/user/input.tar.gz"
8       ],
9       "write": ["/home/user/output"],
10      "exec": ["/usr/bin/tar"],
11      "deny": ["/home/user/output/misc"]
12    }
13  ]
14 }

```

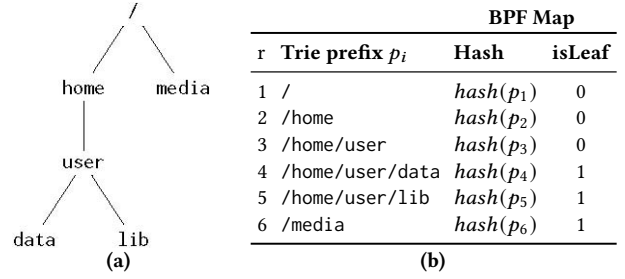
From a software development perspective, RWX permission rules are easy to understand and immediate to write. Contrary to the use of other frameworks such as the combination of Linux Namespaces [8] and Control Groups [28], RWX rules also permit to grant access with file-level granularity on the whole filesystem, and not only on single volumes (Objective O3).

4.3 Support to deny rules

Deny rules ensure a process has no access privileges over a file or a directory. Its introduction significantly reduces the overhead of the developer writing the policy. Unfortunately, the current *inode-based* design of Landlock does not provide support for it [64]. Given the low overhead associated with BPF (Objective O6), its fine-grained access control capabilities (Objective O3) and its compatibility with other LSM security mechanisms including Landlock (Objective O1), we identified BPF as the ideal candidate to implement the support to deny rules. However, loading BPF programs and maps is a privileged operation, so Cage4Deno needs to execute a preliminary initialization phase with additional permissions (as highlighted in Section 4.1).

This Section details the work we did to support deny rules. First, we present the problem of efficiently supporting access control decisions given a sequence of deny rules, then we explain how to translate the approach into BPF programs, making clear how the *sandboxer* enforces access control at runtime.

4.3.1 Deny listing approach. Similarly to what happens for RWX rules, we expect the developer to list the deny rules into the JSON policy file (see Listing 3). Based on the list provided, the *sandboxer* instruments the kernel so that, upon receiving an access request from a sandboxed process, the kernel is able to allow or deny it. For instance, assume the developer lists in the policy the deny rules `/home/user/data`, `/home/user/lib` and `/media`. When the sandboxed process issues an `open` to `/home/user/file` the request is allowed, but when the process tries to access `/home/user/data` (or any of its content) the request is denied. So, given a path request issued by the user, whether it matches exactly one of the rules in the deny list, or a deny rule is a prefix of the requested path, the access request is blocked. A classic solution to solve this problem efficiently can be implemented using a prefix tree (or trie). The idea is to split each path into a sequence of substrings using `/` as

**Table 1: Deny prefix tree (a) and BPF Map_{policy} (b)**

delimiter, treating the substrings as single character prefixes. The prefixes are then used to build a prefix tree as shown in Table 1a. At runtime, access requests are simply evaluated traversing the trie. If a leaf node is visited, we can conclude that a deny rule was hit, and the access request must be denied. The time complexity of a trie traversal depends on its maximum height. Given N the length of the longest deny rule D_{r_i} in the policy,³ the maximum depth of the trie is $N/2$ (in the case of a path structured as $[/c]^+$). When a hashmap is used to implement the trie, each lookup (used to jump from the parent to the child) takes $O(1)$ time, thus traversing the trie takes $O(N)$ time (worst case upper bound).

4.3.2 Implementation using BPF maps. The trie-based approach presented above cannot be translated directly to a BPF program. Indeed, there are some constraints a BPF program must satisfy to be executed by the kernel (see Section 2.3). The most important, in our case, is related to the use of memory. The current implementation of BPF does not provide support for multi-level map-in-map [48] structures required to implement the trie. Hence, we decided to translate the trie into a single-level hashmap Map_{policy} storing all the prefixes represented in the trie, as shown in Table 1b. The hash of each prefix is used as lookup key, while the value is a boolean to indicate whether the prefix is a leaf node in the trie. This representation permits to answer access queries with the same efficiency of the high-level approach. In fact, the trie traversal is easily replaced by a sequence of $O(N)$ lookups, each taking $O(1)$ time when a hash function is used to process the string representing the access path. Storing in the map also the prefixes that are not leaves in the trie increases the size of the map up to $O(M \cdot N)$ given a list of M deny rules, yet, it permits to reduce the query time for all the path prefixes that are not contained in the trie, as the lookup sequence terminates when a key error (i.e., a *miss*) occurs.

This design strategy relies on the ability to convert strings to integers using a cryptographic hash function. Unfortunately, such function is not currently available in BPF. In Appendix A we present this aspect in detail, explaining how we adapted the design to achieve the best trade-off between query response time and map size using an incremental hash function. Other details, such as explicit collision handling, are also discussed there, as they complement the design of our proposal.

4.3.3 BPF policy attachment. To enforce access control at runtime, the *sandboxer* must be able to retrieve the Map_{policy} according to

³The maximum path length is bounded to 4096 chars in `linux/limits.h`

	Access control hooks
Thread lifecycle hooks	lsm/path_mknod
uprobe/attach_policy	lsm/path_mkdir
lsm/task_alloc	lsm/path_link
tp_btf/sched_process_fork	lsm/path_symlink
tp_btf/sched_process_exit	lsm/file_open
	lsm/path_rename
	lsm/path_rmdir
	lsm/path_unlink

(a)

(b)

Table 2: Hooks and tracepoints monitored by Cage4Deno

the argument provided by the developer to the `Deno.run()`, and then to *attach* it to the sandboxed process. The attachment of the proper policy to the sandboxed process is delicate, and requires a set of programs to be loaded by Cage4Deno into the kernel alongside the maps. To reduce the runtime overhead, BPF programs, together with the BPF policy maps associated with all the policies listed in the JSON policy file, are loaded from user space to kernel space by Cage4Deno at startup time. These operations are carried out using *libbpf* [47]. The programs are separated in two categories: (i) programs that are needed to trace the lifecycle of a sandboxed process, and (ii) programs to evaluate the access queries it performs (according to the strategy described in Section 4.3.2).

The first group of BPF programs, namely the ones used to trace the process lifecycle, are responsible for maintaining in memory the map Map_{task} of processes running on the system that are subject to the restrictions imposed by Cage4Deno. Specifically, Map_{task} associates each thread identifier to the proper Map_{policy} . The entries in Map_{task} are updated when one of the hooks listed in Table 2a is reached. For instance, when a traced process issues the `clone` syscall, the tracepoint `tp_btf/sched_process_fork` is reached, the related BPF tracing program executed, and the new child process, inheriting the policy of the parent, is added to Map_{task} .

The second group of BPF programs, the ones that are associated with the evaluation of the access query, are executed when any of the hooks listed in Table 2b is reached. The role of these programs is to check whether the current process (i.e., the process issuing the access request) belongs to the Map_{task} , and accordingly to perform the sequence of lookups in Map_{policy} to allow or deny the request.

5 POLICY GENERATION

A key aspect of our proposal is usability by developers. The straightforward RWX+D permission model is functional to achieve ease of use (Objective O2), however, we believe that it is also important to offer to the developers tools to support the generation of policies (Objective O4). Indeed, each policy can be seen as a template comprising the dependencies to run a program that can be further extended with information related to the input, the output, and the restrictions the program may be subject to. This section details the work we did to support the process, explaining how our solution can be leveraged to generate the RWX rules shown in Listing 3.

The retrieval of the dependencies used by a program is a recurring problem. Just to mention a few, it occurs in the Google Sandboxed API project, where the Sandbox2 sandboxing utility [31]

Listing 4: Semi-automatic generation of tar policy (Listing 3)

```

1 # Set the active policy_name for a program
2 dmng -c tar --setcontext tarPolicy
3 # Add the dependencies of tar to the template
4 dmng -c tar -t -s
5 # Add the input to the template
6 dmng -c tar -a input.tgz -p r--
7 # Add the output to the template
8 dmng -c tar -a output -p -w-
9 # Add the denials
10 dmng -c tar -a output/misc -d
11 # Serialize the entries into a JSON file
12 dmng --serialize tar_policy_file

```

leverages `ldd` (acronym for *List Dynamic Dependencies*) to retrieve the dependencies used by programs available on the host as *Executable and Linkable Format* (ELF) files; or in SlimToolkit [61], where a containerized service is run against a test suite to automatically create Seccomp [9] and AppArmor [10] security profiles matching the least-privilege principle. In Cage4Deno, we adopt a similar approach, which retrieves the files a program should be able to read, write, or execute to work as intended. Moreover, we need to provide the developer with functions to manage multiple policies simultaneously (updating the lists of dependencies and denials), and to serialize them into the JSON format as shown in Listing 3.

The solution we adopted was to develop `dmng`, a CLI tool written in Go (~3.5k lines of code), which ships within Cage4Deno, allowing the developer to automatically generate RWX rules, and interactively refine, the policy. The utility supports the retrieval of program dependencies that are available on the host as ELF files, and those that are spawned by dedicated POSIX or shell wrappers. It can also be used with programs loading dynamic modules at runtime (e.g., media processors loading custom encoders), or programs executing several binaries. To support these use cases, `dmng` relies on both `ldd` and `strace`. The former was preferred to `objdump` and `readelf` since it can be used to recover the so called *transient* dependencies of programs available as ELF files; while the latter is a diagnostic, debugging and instructional user-space utility for Linux that is used to trace programs at runtime.

Similarly to the approach presented in SandTrap [3], the developer can use `dmng` to run a test suite against a program (or a script). This approach allows the developer to reuse existing tests to generate RWX rules that satisfy multiple execution paths of the binary. By using `strace`, `dmng` monitors the interactions between the threads spawned by the tested utility and the kernel for a certain amount of time (usually less than 2s). In this time frame, the file-related syscalls issued by the set of threads, along with their arguments, are captured and saved to a log file. The log is then used by `dmng` to generate the RWX rules accordingly. The list of syscalls monitored comprises the ones wrapped by the standard system functions `execve()`, `open()`, `create()`, `link()`, `mkdir()`. The `dmng` tool exposes to the developer many functions to inspect, add, update or remove the entries associated with the policy template. Specifically, it allows to add deny rules, which generation cannot be automated. It also implements heuristics to reduce the number of RWX rules, thus improving their readability, and facilitating policy auditing,

when explicitly requested by the developer. This inherently comes with the downside of producing coarser permissions. The process is called *permission pruning* and it is based on common security practices, like the identification of write-or-exec (W^X) memory regions, but also contextual information about the specific region of the filesystem being considered (e.g., there are no practical advantages in assigning fine-grained permissions under `/usr/share/fonts/`, while it is certainly useful to do so under `/home/user/Documents`). For this, the paths are arranged into a Trie, and then the above heuristics are used to incrementally prune leaf nodes until the developer’s target number of rules is met.

For each of the programs to be restricted, a sequence of commands may be input by the developer to synthesize the final policy (state is persisted between tests using a SQLite database). Since the developer can work with distinct programs and many policies simultaneously, a cache is used to store the active context (i.e., the current policy) of each program (line 2 in Listing 4). Then, each context is customized adding the dependencies as previously described. For instance, in line 4 `dmng` collects all the dependencies required by `tar` using dynamic tracing, and in line 10 the developer manually adds the deny rule. After all the policy contexts have been configured, the policy is serialized into the JSON format as expected by `Cage4Deno` (line 12). Appendix B reports the complete `tar` policy produced following the instructions in Listing 4.

In this paper we employ dynamic analysis for the automatic generation of policies, however this is widely known for producing results whose completeness depends on the test suite coverage [77]. To address this limitation, the current approach could be complemented by either source or binary code analysis. Relevant works (e.g., [16]) use static analysis to pinpoint the system calls, however they do not keep track of parameter values. On the other hand, *AutoArmor* [45] uses static taint analysis to perform semantics-aided program slicing from network API invocations, and then uses these slices to extract access control attributes. The approach also looks promising for the generation of file system policies.

6 EXPERIMENTS

In this Section we present the experimental evaluation of `Cage4Deno`. The evaluation considers two aspects of our solution: exploit mitigation (Section 6.1), and performance overhead (Section 6.2). To perform our evaluation we used the following test environment: an Ubuntu 22.04 LTS server powered by an AMD Ryzen 3900X CPU with 12 cores (24 threads), 64 GB RAM, 2 TB SSD. Then, we replaced the pre-installed Linux Kernel v5.15 with the same release of the kernel, but with BPF LSM available (Landlock LSM is enabled by default).

6.1 Exploit mitigation

The primary goal of this Section is to demonstrate the capability of `Cage4Deno` to mitigate real CVEs (Objective O5). Focusing on the records of the last 5 years, we selected a sample of 15 vulnerabilities affecting common web application utilities, such as `ExifTool`, `FFmpeg`, `Git`, `GNU Tar`, `GraphicsMagick`, `Ghostscript`, `ImageMagick`, `OpenSSL`, `Pip`, `UnRAR`, and `Unzip`. The CVEs we considered are classified into Remote Code Execution (RCE), Local File Read (LFR), and Arbitrary File Overwrite (AFO). Their details are reported in Table 3.

CVE ID	Utility	Use case
Local File Read (LFR)		
CVE-2016-1897	FFmpeg v3.2.5	Video processing
CVE-2016-1898	FFmpeg v3.2.5	Video processing
CVE-2019-12921	GraphicsMagick v1.3.31	Image processing
Arbitrary File Overwrite (AFO)		
CVE-2016-6321	GNU Tar v1.29	Archive decompression
CVE-2019-20916	Pip v19.0.3	Dependency fetch
CVE-2022-30333	UnRAR v6.11	Archive decompression
Remote Code Execution (RCE)		
CVE-2016-3714	ImageMagick v6.9.2-10	Image processing
CVE-2020-29599	ImageMagick v7.0.10-36	Image processing
CVE-2021-3781	Ghostscript v9.54.0	PDF processing
CVE-2021-21300	Git v2.30.0	Clone repository
CVE-2021-22204	ExifTool v12.23	Image processing
CVE-2022-0529	Unzip v6.0-25	Archive decompression
CVE-2022-0530	Unzip v6.0-25	Archive decompression
CVE-2022-1292	OpenSSL v3.0.2	Certificate verification
CVE-2022-2566	FFmpeg v5.1	Image processing

Table 3: Sample of CVEs mitigated by `Cage4Deno`. Despite being discovered in 2016, CVEs 1897, 1898, and 6321 have seen recent exploitation in 2018 [67] and 2021 [36]

These vulnerabilities have been selected as they represent examples of 0-click exploits, they can lead to severe security breaches, and they target utilities extensively used by web applications. In general, similar considerations extend to a broader set of CVEs.

In our analysis we reproduced each of the vulnerabilities adapting public Proofs of Concepts. For each of them, we provide a single-click test showcasing the sandboxing capabilities added to `Deno`. In particular, we show that: (i) the attacker can successfully exploit the vulnerability when `Deno` is used (despite its permission model being in place), and (ii) the attack is unsuccessful when the sandboxing functions offered by `Cage4Deno` are used. The only difference between case (i) and (ii) is that a policy is provided with the `--policy-file` argument. This aspect testifies how simple it is to benefit from the sandboxing functions we have introduced (Objective O2). The policy files used to restrict each utility have been automatically generated with the `dmng` tool described in Section 5. Table 4 reports the number of rules generated for the selected list of utilities without applying any form of policy minimization (i.e., pruning). As shown in the table, the utilities require less than 115 permissions to work as intended, with `pip` requiring the largest set.

From a security standpoint, it is worth noting that `Cage4Deno` can block the attacks at multiple levels. For instance, in CVE-2020-29599, in which a crafted picture is sent to `ImageMagick` to execute a target command (e.g., `id`), `Cage4Deno` does not allow RX access to `/usr/bin`, blocking `/usr/bin/echo` first (which is used to inject `id` in a subshell), and then the target command itself (i.e., `usr/bin/id`). In this case, the denial is due to the Landlock sandbox allowing access solely to `/usr/bin/convert` (i.e., the `ImageMagick` binary). Instead, when we consider CVE-2016-6321, in which a decompression of an archive permits the attacker to overwrite a target file,

Utility	#rules	Deno [ms]	Cage4Deno [ms]
cat	9	3.05±0.23	3.81±0.25
GraphicsMagick	81	10.16±1.02	12.16±1.12
UnRAR	25	13.86±1.97	15.84±2.71
ImageMagick	17	17.49±2.14	18.74±2.26
Unzip	15	20.90±3.95	22.66±3.62
OpenSSL	17	27.80±4.93	30.10±7.50
Git	26	66.52±4.75	72.46±5.22
ExifTool	38	109.20±6.67	112.88±4.25
GNU Tar	14	114.52±7.21	125.48±6.89
FFmpeg	12	321.50±9.55	336.70±9.78
Ghostscript	20	449.96±18.19	455.66±21.37
Pip	115	3022.52±20.55	3203.32±20.84

Table 4: Preliminary test showing the execution time of utilities reported in Table 3 over 500 runs (as $\mu \pm \sigma$)

we show how deny rules can be used to prevent filesystem corruption. As a final note, we highlight the ability to simultaneously use distinct policies for a single utility. To give an example, in CVE-2021-21300 several distinct policies can be assigned to pip. By doing so, the developer can select a dedicated policy based on the Python virtual environment currently in use.

Popular packages affected by the aforementioned vulnerabilities can be found in both deno.land/x and npm package archives. Among them, we selected astrodon and fast_forward from the former, fluent-ffmpeg and gm from the latter (executed in Node compatibility mode). We confirmed that the vulnerable versions of the binaries are still exploitable through the mediation of third-party modules, and Cage4Deno proves to be effective in their mitigation without code modification to the application and its dependencies.

6.2 Performance evaluation

A requirement of our proposal is that it must not introduce a large runtime overhead compared to the scenario in which the developer relies solely on the basic functions offered by Deno (Objective O6). This is fundamental, as any additional delay could negatively affect the end-user experience or increase the cost to host the application. To investigate this aspect, we performed tests comparing the execution time of an application using vanilla Deno with respect to the same application subject to access restrictions with Cage4Deno. A second interesting aspect to consider in the evaluation is the overhead introduced by our proposal compared to general-purpose sandboxing solutions proposed by the industry. Finally, we evaluate how our solution scales with the number of rules in the policy.

6.2.1 Runtime overhead. In our analysis we expected the runtime overhead to vary according to the size of the policy and the number of filesystem requests performed by the sandboxed utility. We therefore conducted a preliminary test involving the utilities affected by the CVEs detailed in Section 6.1. For each of them, we measured the execution time of vanilla Deno and Cage4Deno. The results of the test are reported in Table 4. The data clearly show that the largest overhead is associated with the larger policies and the shorter execution time. To further highlight this aspect, we imagined the worst case scenario in which a simple binary is used to perform a single access to the filesystem. As shown in row 1 of Table 4, printing the

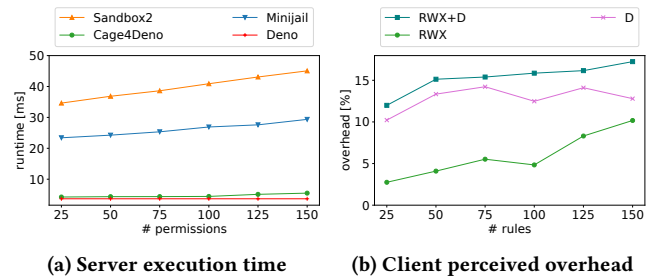


Figure 5: Deterioration of overhead varying the policy size

content of an empty file using the cat binary is associated with the greatest overhead, roughly 25% of the execution time. cat is characterized by a minimal execution time, and requires only 9 rules in the policy to work as intended, thus proving to be the best candidate to highlight the overhead introduced by Cage4Deno.

To evaluate how our solution scales with the increase of rules in the policy, we set up a second test. We again used cat, but this time we added to its minimum set of permissions a varying number of RWX rules, ranging from 25 to 150. To measure the execution time we relied on the benchmarking module provided by the Deno standard library [18], and to ensure the statistical value of the results we repeated the test 500 times. Initially, we compared the execution time of vanilla Deno and Cage4Deno. While Cage4Deno inevitably introduces an overhead, its performance degradation (compared to Deno’s) ranges between 0.8 and 2.5 ms, which is a reasonable price for the additional security guarantees it provides.

To further inquire, we compared Cage4Deno with two general-purpose sandboxing solutions: (i) Minijail [30], and (ii) Sandbox2, a key component of the Sandboxed API [31] framework. As shown in Figure 5a, the execution time associated with the use of Minijail is 5.52 to 5.63 times slower compared to the one of Cage4Deno, and Sandbox2 exhibits even a worse degradation, which is never less than 8.15 times. To investigate the principal components causing the overhead, we profiled the execution of each tool with perf. Each of them presented two distinct phases: (i) sandbox setup, and (ii) restricted execution. For Minijail and Sandbox2 the first phase is dominated by creating the mount namespace, changing root directory, and performing a bind-mount for all the files needed to run the utility. For Cage4Deno, equivalent protection can be achieved with only the creation and enforcement of the Landlock rulesets. As for the execution phase, Minijail and Sandbox2 suffer slowdowns whenever performing filesystem operations due to the execution of kernel code paths related to the resolution of namespaces and bind mounts. In addition, Sandbox2 reference monitor architecture requires inter-process communication between the tracee and its tracer, which further degrades performance. On the other hand, Cage4Deno only pays the overhead of evaluating the Landlock security checks.⁴ This validates our design choices, proving that our solution can provide significantly better performance with respect to industrial general-purpose sandboxing solutions. Moreover, Figure 5a depicts an interesting trend. Among the sandboxing alternatives, not only Cage4Deno delivers the best performance, but it

⁴The flame graphs used for the analysis are available in the repository of the project.

also exhibits the slowest linear growth with respect to the increase of the number of rules, making it the best option when a large number of policy rules is necessary.

6.2.2 Overhead associated with deny rules. The previous tests do not take into account the deny rules. In fact, both Minijail and Sandbox2 do not support the presence of negative permissions. It is then important to measure the overhead introduced by them, as well as its degradation when the policy size increases. To better investigate the overhead experienced by end-users when Cage4Deno is used on the back-end, we evaluated the response time of a single HTTP endpoint that responds to incoming request with the content printed by the `cat` command. To generate the HTTP requests we relied on *jMeter* [5], a tool written in Java designed to load test and measure the performance of web applications. Specifically, we configured it to generate 100 warm up requests and measured the latency of the following 5000. To simulate the overhead experienced by end users of a web application, we also set up a network delay of 10 ± 5 ms (with a normal distribution) using `tc` [4], a Linux utility to configure and shape the network traffic. The test is executed four times with the following policy configurations: (i) no policy (i.e., vanilla Deno), (ii) only positive rules (i.e., RWX), (iii) only deny rules, and finally (iv) a policy characterized by the same number of RWX and D rules. Again the number of rules in the policy ranges between 25 and 150.

In general, the results reported in Figure 5b confirm the trend shown in Figure 5a, with an overhead increasing almost linearly with the number of rules in the policy. This is a positive aspect, since it applies to both RWX and D rules. The relative overhead perceived by clients, with respect to the default implementation of Deno, ranges from 2.74% to 10.21% for policies listing only RWX rules. This attests that the cost associated with Landlock is low, but it increases linearly with the number of rules (in accordance with the current inode-based implementation of the LSM). On the other hand, the use of BPF is distinguished by a higher initial cost, but the performance degradation introduced when the number of deny rules increases is smaller compared to the one measured with pure RWX policies, since it ranges between 10.17% and 12.79%. Moreover, policies composed of both rule types still exhibit a linear trend, with a client perceived overhead ranging from 11.99% to 17.24%. It is important to note that the presence of deny rules not only improves the readability and maintainability of the policy, but can also improve the performance as it permits to reduce the overall number of rules in the policy.

7 RELATED WORK

Several approaches to strengthen the security guarantees provided by JavaScript execution environments have been proposed by both industry and academia. While developing Cage4Deno we focused on solutions that target runtimes and are therefore applicable to the server-side scenario. Our proposal complements them providing effective isolation of subprocesses using recent technologies. In the following, we separate the related works into four main categories.

Secure JavaScript sandboxes. Several works have been proposed to strengthen the security guarantees offered by JavaScript runtimes before the advent of Deno [3, 20, 49, 57–59, 71, 74, 75, 78].

Secure EcmaScript (SES) [49] is a runtime library to execute third-party code safely in lightweight compartments. Essentially, SES implements a *frozen* execution environment in which scripts have no abilities to interfere with each other. An alternative for the safe execution of untrusted third-party JavaScript code is *vm2* [59]. Its peculiarity is that it overrides the built-in `require`, enabling the developer to restrict access to a pre-defined set of modules. SES and *vm2* have been effective in mitigating the vulnerabilities coming from unverified or untrusted third-party JavaScript code, since they allow the developer to practically limit the APIs exposed to the sandboxed JavaScript program. However, when access to a dangerous API such as `Deno.run()` or `child_process.spawn()` is granted, no restriction on the subprocess is enforced.

Another interesting approach to reduce the security risks coming from the use of third-party modules is *BreakApp* [74]. The authors use module boundaries to automate compartmentalization of systems and enforce security policies. To this end, *BreakApp* spawns and executes modules in protected compartments, while preserving their original behavior. Compartments are characterized by three levels of isolation: sandbox, process, and container level. While the approach is powerful and ensures strong security properties when the container level is used, it is not straightforward to setup fine-grained filesystem permissions when the process level is selected.

Mir [75] is a relevant proposal to mitigate the security risks coming from third-party modules through the use of library-specialized contexts. *Mir* applies a fine-grained RWX permission model to every field of every free variable name in the context of an imported library, with permissions inferred through static and dynamic analysis. *SandTrap* [3] shares with *Mir* the idea of protecting read, write and call on entities (primitive values, functions, objects) and construct policies on cross-domain interaction. The goal again is to mitigate the security risk coming from the use of third-party modules, but in the Trigger-Action Platforms (TAPs) framework. While both the approaches can mitigate several JavaScript vulnerabilities, no restriction is applied on code executed outside of the runtime.

Wolf at the Door [78] is a recent proposal to reduce the risk coming from the installation of third-party modules. The authors propose to use the Apparmor LSM to detect install time anomalies such as connection to unknown hosts or read of confidential files. The security checks are enforced based on a policy typically written from the package maintainer. The approach protects the host against undesired behavior of third-party packages installed through `npm`, but the protection is enforced only at install time.

Isolation and sandboxing of unsafe libraries. The problem of isolating subprocesses addressed by Cage4Deno shares strong similarities with the isolation and sandboxing of third-party libraries. This area has received significant attention recently, and many proposals have been published [29, 37, 43, 52, 60, 62, 63, 65, 73, 76]. *Galeed* [62], *PKRU-Safe* [43], and *NoJITsu* [60] guarantee strong isolation of unsafe components with the use of Memory Protection Keys (MPK). *Galeed* and *PKRU-Safe* preserve the memory safety of Rust code when used in conjunction with unsafe code (e.g., C/C++). *NoJITsu* brings hardware-backed, fine-grained memory access protection to JavaScript engines, thus successfully hindering a wide range of memory corruption attacks. Differently from

previous solutions, *RLBox* [52] achieves sandboxing of third-party libraries through software-based-fault isolation. *RLBox* facilitates the retrofitting of existing applications using a type-driven approach that significantly ease the burden of securely sandboxing libraries in existing code. These solutions are complementary to *Cage4Deno*, as they can be used in conjunction with it to enforce trust boundaries in the interaction between Deno and its subprocesses.

General-purpose sandboxers. To reduce the impact of potentially vulnerable processes, several approaches leveraging system call interposition have been proposed [7, 12, 16, 30, 31, 42, 53].

TRON [7] is a discretionary access control system. The authors designed a Unix based capability system to limit process access to files, directories and directory trees. Although the approach allows fine-granularity access control, it suffers from two major drawbacks: (i) it requires modifications on the kernel, and (ii) it requires programs changes to make them capability-aware.

Another interesting initiative to implement unprivileged sandboxes is *MBOX* [42]. *MBOX* executes a program in a sandbox, and prevents it from modifying the host filesystem by layering the sandbox filesystem on top of it. Only at the end of the execution the user can examine changes in the layered filesystem and commit them back to the host. To do so, *MBOX* interposes system calls using *Seccomp* filters, and relies on *ptrace* to enforce permissions. However, the use of *ptrace* is prone to TOCTOU attacks [39], and as experimented by the authors, it can lead to non-negligible overhead.

In practice, system call interposition is often used to limit the set of system calls available to a program [16, 61, 77]. This effectively limits the capabilities of an attacker exploiting the program, and reduces the attack surface of the kernel [46]. Nowadays, most solutions rely on *Seccomp* filtering, a Linux kernel feature that allow to filter system calls based on their identifiers and parameter values. While valid, *Seccomp* filters can neither dereference pointers to user memory, nor actionably use file descriptor numbers, thus they are not suitable to perform access control of filesystem resources. These solutions can be used in conjunction with *Cage4Deno* to reduce the attacker capabilities and the attack surface of the kernel.

In Section 6 we have already compared *Cage4Deno* to other general-purpose sandboxing solutions such as *Minijail* [30], and *Sandbox2* [31]. These tools support multiple types of containment techniques such as the introduction of new user ids, restriction of capabilities, policy-based *Seccomp* filtering, and namespace isolation. Both the tools are powerful and flexible, but they are not specifically aimed towards protecting web applications. Thus, they are not optimized for the execution of short-lived programs, and they target security experts, making them of difficult use to a wider audience. Specifically, to achieve comparable protection to our *RWX* rules, the developer needs to configure them to: (i) create a new mount namespace, (ii) change the root directory of the binary, and (iii) remount every part of the file hierarchy necessary for the functioning of the binary under the new root. On the other hand, *Minijail* and *Sandbox2* use well established kernel features available in every modern Linux kernel version, and can be used without additional privileges provided there is no need to bind-mount privileged resources. Similar considerations can be made about *Firejail* [53] and *Bubblewrap* [12]; sandboxing tools functionally comparable to *Minijail* and *Sandbox2*, but less mature.

BPF-based sandboxers. Other proposals have used BPF as the primary means to enforce access control policies [2, 6, 11, 25–27, 38].

BPFBox [27] and *BPFContain* [26] are runtime security frameworks focusing on the containment of processes and containers, respectively. Comparing *Cage4Deno* to both the proposals, (i) we do not require a privileged runtime daemon to keep track of the traced processes (single point of failure), (ii) we rely on *Landlock* to enforce *RWX* permissions so to guarantee low runtime overhead, and (iii) we provide a tool for the automatic generation of policies.

Snappy [6] strengthens the security of containers using namespaces and BPF policies. To support the programmable policies described, the authors introduced in the kernel a set of new *dynamic helpers*. Jia et al. [38] present a mechanism to define advanced syscall filtering policies with the *extended BPF*. This is currently achieved by hooking syscall tracepoints that cause system-wide performance degradation and are still subject to TOCTOU attacks. On the other hand, Jia et al. successfully address these limitations proposing changes to the Linux kernel. These proposals require to recompile the kernel with the addition of ad hoc functions not part of the kernel codebase, thus limiting their usability and portability.

There are also industrial solutions using BPF, for instance: *Cilium* [11] and *Falco* [25]. The former provides BPF-based networking, observability and security between container workloads, the latter is a threat detection engine for clusters. Both operate at the container granularity; hence, developers may find it difficult to set up fine-grained permissions with these frameworks.

These proposals demonstrate the value of BPF in securing different types of system resources. Extending our proposal to the protection of non-filesystem resources is promising for future work.

8 CONCLUSIONS

Web development is a fast-paced environment characterized by tight time constraints. In this vast ecosystem, we specifically considered the role of Deno, a modern and secure runtime for JavaScript and TypeScript. Our proposal presents a way to improve the Deno security model by sandboxing the invocation of subprocesses, which represent an important attack vector of web applications. As shown in the experimental evaluation, *Cage4Deno* effectively mitigates real CVEs affecting widely-used utilities in web applications. Moreover, it exhibits significantly better performance with respect to other general-purpose sandboxing solutions. We paid great attention to reduce the overhead of the developer willing to strengthen the security of its application. This is achieved not only by exposing a simple and clear interface to the developer (requiring no mandatory code changes), but also providing a command line tool to generate least-privileged, yet human-readable, access control policies. We believe the proposed approach is general enough to be applied to other runtimes, since they share the same design as Deno regarding the unconstrained execution of subprocesses (e.g., Node.js and Bun). This is a promising line of research we plan to explore in the future.

ACKNOWLEDGMENTS

We thank our shepherd Dokyung Song and the reviewers for their valuable feedback. The work was supported by the European Commission within the GLACIATION project (No 101070141).

REFERENCES

- [1] A. Starovoitov. 2020. CAP_BPF. <https://lwn.net/Articles/820560/>
- [2] M. Abbadini, M. Beretta, D. Facchinetti, G. Oldani, M. Rossi, and S. Paraboschi. 2023. Leveraging eBPF to enhance sandboxing of WebAssembly runtime. In *ASIACCS*.
- [3] M. M. Ahmadpanah, D. Hedin, M. Balliu, L. E. Olsson, and A. Sabelfeld. 2021. SandTrap: Securing JavaScript-driven Trigger-Action Platforms. In *USENIX Security*.
- [4] W. Almesberger. 1999. Linux Network Traffic Control – Implementation Overview.
- [5] Apache. 2022. JMeter. <https://jmeter.apache.org/>
- [6] M. Bélaïr, S. Lanjepce, and J. Menaud. 2021. SNAPPY: Programmable Kernel-Level Policies for Containers. In *SAC*.
- [7] A. Berman, V. Bourassa, and E. Selberg. 1995. TRON: Process-Specific File Protection for the UNIX Operating System. In *USENIX ATC*.
- [8] E. W. Biederman. 2006. Multiple Instances of the Global Linux Namespaces. In *Ottawa Linux Symposium (OLS)*.
- [9] C. Canella, M. Werner, D. Gruss, and M. Schwarz. 2021. Automating Seccomp Filter Generation for Linux Applications. In *CCSW*.
- [10] Canonical. 2022. AppArmor. <https://apparmor.net>.
- [11] Cilium. 2022. Cilium. <https://github.com/cilium/cilium>
- [12] containers. 2022. Bubblewrap. <https://github.com/containers/bubblewrap>
- [13] J. Corbet. 2019. KRSI. <https://lwn.net/Articles/808048/>
- [14] CVE Mitre. 2021. Gitlab Exiftool vulnerability. <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-22205>
- [15] A. Decan, T. Mens, and E. Constantinou. 2018. On the Impact of Security Vulnerabilities in the npm Package Dependency Network. In *MSR*.
- [16] N. DeMarinis, K. Williams-King, D. Jin, R. Fonseca, and V. P. Kemerlis. 2020. sysfilter: Automated System Call Filtering for Commodity Software. In *RAID*.
- [17] Deno Land. 2022. Deno Permission Model. https://deno.land/manual/getting_started/permissions#permissions
- [18] Deno Land. 2022. Deno standard library for testing. <https://deno.land/std/testing>
- [19] Deno Land. 2022. Deno Subprocess. <https://deno.land/manual@v1.26.0/examples/subprocess>
- [20] Deno Land. 2022. Deno Workers. <https://deno.land/manual@v1.26.0/runtime/workers>
- [21] Deno Land. 2022. Node compatibility mode. https://deno.land/manual/node_compatibility_mode.
- [22] Docs.rs. 2022. Tokio. <https://docs.rs/tokio/0.2.0/tokio/index.html>
- [23] dsherret. 2022. dax. <https://github.com/dsherret/dax>.
- [24] R. Duan, O. Alrawi, R. P. Kasturi, R. Elder, B. Saltaformaggio, and W. Lee. 2021. Towards Measuring Supply Chain Attacks on Package Managers for Interpreted Languages. In *NDSS*.
- [25] Falco. 2022. Falco. <https://github.com/falcosecurity/falco>
- [26] W. Findlay, D. Barrera, and A. Somayaji. 2021. BPFContain: Fixing the Soft Underbelly of Container Security. *arXiv* (2021).
- [27] W. Findlay, A. Somayaji, and D. Barrera. 2020. bpfbox: Simple Precise Process Confinement with eBPF. In *CCSW*.
- [28] X. Gao, Z. Gu, Z. Li, H. Jamjoom, and C. Wang. 2019. Houdini’s Escape: Breaking the Resource Rein of Linux Control Groups. In *CCS*.
- [29] A. Ghosn, M. Kogias, M. Payer, J. R. Larus, and E. Bugnion. 2021. Enclosure: Language-Based Restriction of Untrusted Libraries. In *ASPLOS*.
- [30] Google. 2022. Minijail. <https://google.github.io/minijail/>
- [31] Google. 2022. Sandbox2. <https://developers.google.com/code-sandboxing/sandbox2/>
- [32] Google. 2022. zx. <https://github.com/google/zx>.
- [33] B. Gregg. 2021. BPF Internals. <https://www.usenix.org/conference/lisa21/presentation/gregg-bpf> USENIX LISA.
- [34] H. Tao. 2022. BPF: Introduce ternary search tree for string key. <https://lore.kernel.org/bpf/20220331122822.14283-1-houtao1@huawei.com>.
- [35] H. Tao. 2022. BPF: Support for string key in hash-table. <https://lore.kernel.org/bpf/20211219052245.791605-1-houtao1@huawei.com>
- [36] hackerone. 2021. External SSRF and Local File Read due to vulnerable FFmpeg. <https://hackerone.com/reports/1062888>
- [37] M. Hedayati, S. Gravani, E. Johnson, J. Criswell, M. L. Scott, K. Shen, and M. Marty. 2019. Hodor: Intra-Process Isolation for High-Throughput Data Plane Libraries. In *USENIX ATC*.
- [38] J. Jia, Y. Zhu, D. Williams, A. Arcangeli, C. Canella, H. Franke, T. Feldman-Fitzthum, D. Skarlatos, D. Gruss, and T. Xu. 2023. Programmable System Call Security with eBPF. *arXiv* (2023).
- [39] Z. Junyuan and R. Guo. 2021. Phantom Attack: Evading System Call Monitoring. <https://defcon.org/html/defcon-29/dc-29-speakers.html#guo> DEFCON.
- [40] J. Karásek, R. Burget, and O. Morsky. 2011. Towards an Automatic Design of Non-Cryptographic Hash Function. In *TSP*.
- [41] M. Kehoe. 2022. eBPF: The Next Power Tool of SREs. <https://www.usenix.org/conference/srecon22americas/presentation/kehoe-ebpf> USENIX SREcon.
- [42] T. Kim and N. Zeldovich. 2013. Practical and Effective Sandboxing for Non-root Users. In *USENIX ATC*.
- [43] P. Kirth, M. Dickerson, S. Crane, P. Larsen, A. Dabrowski, D. Gens, Y. Na, S. Volckaert, and M. Franz. 2022. PKRU-safe: automatically locking down the heap between safe and unsafe languages. In *EuroSys*.
- [44] Deno Land. 2022. Deno: JavaScript runtime. <https://deno.land/>
- [45] X. Li, Y. Chen, Z. Lin, X. Wang, and J. H. Chen. 2021. Automatic Policy Generation for Inter-Service Access Control of Microservices. In *USENIX Security*.
- [46] Y. Li, B. Dolan-Gavitt, S. Weber, and J. Cappos. 2017. Lock-in-Pop: Securing Privileged Operating System Kernels by Keeping on the Beaten Path. In *USENIX ATC*.
- [47] libbpf. 2022. libbpf. <https://libbpf.readthedocs.io/en/latest/index.html>
- [48] M. K. Lau. 2017. BPF map-in-map support. <https://www.mail-archive.com/netdev@vger.kernel.org/msg159387.html>
- [49] M. S. Miller. 2022. Draft Proposal for SES (Secure EcmaScript). <https://github.com/tc39/proposal-ses>
- [50] L. Mastrangelo, L. Ponzanelli, A. Mocci, M. Lanza, M. Hauswirth, and N. Nystrom. 2015. Use at Your Own Risk: The Java Unsafe API in the Wild. *SIGPLAN* (2015).
- [51] A. Nakryiko. 2020. BPF CO-RE. <https://facebookmicrosites.github.io/bpf/blog/2020/02/19/bpf-portedability-and-co-re.html>
- [52] S. Narayan, C. Disselkoen, T. Garfinkel, N. Froyd, E. Rahm, S. Lerner, H. Shacham, and D. Stefan. 2020. Retrofitting Fine Grain Isolation in the Firefox Renderer. In *USENIX Security*.
- [53] netblue30. 2022. Firejail. <https://firejail.wordpress.com/>
- [54] Npm. 2022. fluent-ffmpeg. <https://www.npmjs.com/package/fluent-ffmpeg>.
- [55] Npm. 2022. gm. <https://www.npmjs.com/package/gm>.
- [56] Npm. 2022. sane. <https://www.npmjs.com/package/sane>.
- [57] G. Ntousakis, S. Ioannidis, and N. Vasilakis. 2021. Detecting Third-Party Library Problems with Combined Program Analysis. In *CCS*.
- [58] OpenJS Foundation. 2022. Worker threads. https://nodejs.org/api/worker_threads.html
- [59] P. Simek. 2022. Proposal for VM2: Advanced vm/sandbox for Node.js. <https://github.com/patriksimek/vm2>
- [60] T. Park, K. Dhondt, D. Gens, Y. Na, S. Volckaert, and M. Franz. 2020. NoJITsu: Locking Down JavaScript Engines. In *NDSS*.
- [61] K. Quest. 2022. SlimToolkit. <https://github.com/slimtoolkit/slim>
- [62] E. Rivera, S. Mergendahl, H. Shrobe, H. Okhravi, and N. Burow. 2021. Keeping Safe Rust Safe with Galeed. In *ACSAC*.
- [63] M. Rossi, D. Facchinetti, E. Bacis, M. Rosa, and S. Paraboschi. 2021. SEApp: Bringing Mandatory Access Control to Android Apps. In *USENIX Security*.
- [64] Mickaël Salaün. 2022. Landlock. <https://landlock.io/>
- [65] D. Schrammel, S. Weiser, R. Sadek, and S. Mangard. 2022. Jenny: Securing Syscalls for PKU-based Memory Isolation Systems. In *USENIX Security*.
- [66] Snyk. 2022. State of Open Source Security 2022. <https://snyk.io/reports/open-source-security/>.
- [67] Snyk. 2022. Zip Slip Vulnerability. <https://snyk.io/research/zip-slip-vulnerability>
- [68] Stack Overflow Insights. 2022. Annual survey of the Stack Overflow community. <https://survey.stackoverflow.co/2022/>
- [69] C. Staicu, M. Pradel, and B. Livshits. 2018. Synode: Understanding and Automatically Preventing Injection Attacks on Node.js. In *NDSS*.
- [70] C. Staicu, S. Rahaman, Á. Kiss, and M. Backes. 2023. Bilingual Problems: Studying the Security Risks Incurred by Native Extensions in Scripting Languages. *USENIX Security* (2023).
- [71] J. Terrace, S. R. Beard, and N. P. K. Katta. 2012. JavaScript in JavaScript (js.js): Sandboxing Third-Party Scripts. In *WebApps*.
- [72] V8 project. 2022. What is V8? <https://v8.dev/>
- [73] A. Vahldiek-Oberwagner, E. Elnikety, N. O. Duarte, M. Sammler, P. Druschel, and D. Garg. 2019. ERIM: Secure, Efficient In-process Isolation with Protection Keys (MPK). In *USENIX Security*.
- [74] N. Vasilakis, B. Karel, N. Roessler, N. Dautenhahn, A. DeHon, and J. M. Smith. 2018. BreakApp: Automated, Flexible Application Compartmentalization. In *NDSS*.
- [75] N. Vasilakis, C. Staicu, G. Ntousakis, K. Kallas, B. Karel, A. DeHon, and M. Pradel. 2021. Preventing Dynamic Library Compromise on Node.js via RWX-Based Privilege Reduction. In *CCS*.
- [76] A. Voulimeneas, J. Vinck, R. Mechelinck, and S. Volckaert. 2022. You Shall Not (by)Pass! Practical, Secure, and Fast PKU-Based Sandboxing. In *EuroSys*.
- [77] Z. Wan, D. Lo, X. Xia, L. Cai, and S. Li. 2017. Mining Sandboxes for Linux Containers. In *ICST*.
- [78] E. Wyss, A. Wittman, D. Davidson, and L. De Carli. 2022. Wolf at the Door: Preventing Install-Time Attacks in npm with Latch. In *ASIACCS*.
- [79] W. Zhang, P. Liu, and T. Jaeger. 2021. Analyzing the Overhead of File Protection by Linux Security Modules. In *ASIACCS*.
- [80] M. Zimmermann, C. Staicu, C. Tenny, and M. Pradel. 2019. Smallworld with High Risks: A Study of Security Threats in the Npm Ecosystem. In *USENIX Security*.

A BPF DETAILS

In the following we detail, as a complement, some of the issues we found working with the current implementation of BPF, explaining how we solved them.

A.1 Hashing & collision handling

As anticipated in the design and implementation of deny rules (Section 4.3), the current implementation of the BPF framework does not provide support for map-in-map structures [48]. Moreover, support for using string keys in a map is still limited [35]. At the time of writing, a patch for providing this option, along with efficient ternary search support to lookup the map [34], are under active development on `bpf-next`, the branch dedicated to the features and improvements that should eventually land in BPF. Nonetheless, we had to cope with the temporary limitation of using an integer key to lookup the Map_{policy} , hence, we had to find a way to transform the string prefixes stored in the trie to fixed size integers. A typical approach to solve this problem is to use a hash function. Given the need to preserve the ability to search for prefixes efficiently, we opted for using an incremental hash function. Given S_r , the string representing the access path, an incremental hash function permits to compute $hash(S_r[i])$, the hash of the prefix terminating at character i , in constant time given the hash of the prefix terminating at $i - 1$. In our case we relied on *djb2* [40], a non-cryptographic low-complexity incremental hash function proposed by D. J. Bernstein.

The use of a hashing function also required us to handle the collisions explicitly. Indeed, collisions may occur during map creation, and at runtime upon receiving an access request. Basically, this means that every time a key is found in the policy map Map_{policy} , a collision check has to be performed to determine whether a deny rule was hit. The best trade-off between query response time and size of the policy map has been achieved adapting the separate hash chaining approach, in which a fixed size array is used to store a sequence of colliding paths. To explain how it works we introduce the following example. Let us assume to have three deny rules in the policy: `/home/user/data`, `/home/user/lib` and `/media`. Also, let us assume that the *djb2* hash of `/home/user/lib` and `/media` collide. Instead of using the deny rules to build a prefix tree, we can build the policy map Map_{policy} as shown in Table 5. Upon receiving an access request S_r , the verifier computes the incremental hash for each of its prefixes, and then performs a sequence of map lookups. In case no lookup is successful, the verifier concludes that no deny rule was hit, hence the access request is granted. Conversely, the verifier must check whether a deny rule was hit, or a collision was found. To do that, the verifier compares the access request S_r , and each of the colliding paths associated with the lookup key. Algorithm 1 details the procedure.

The time complexity of the proposed approach varies according to the presence of collisions. When none occurs, the worst case time complexity is given by the total hashing time $O(N)$, plus the total lookup time $O(N)$ (i.e., in the worst case S_r is structured as $[/c]^+$, hence $N/2$ lookups each taking $O(1)$ time are performed). Instead, each time a collision occurs (or a deny rule is hit), the verifier incurs in an $O(N \cdot L)$ extra time, where L is the length of the longest collision chain. Similarly to other research proposals [40], the results

BPF Map	
Deny rule hash	Array of colliding paths
$djb2(D_{r1})$	<code>/home/user/data</code> \rightarrow \emptyset
$djb2(D_{r2}, D_{r3})$	<code>/home/user/lib</code> \rightarrow <code>/media</code> \rightarrow \emptyset

Table 5: Map_{policy} with separate hash chaining (assuming the hashes of `/home/user/lib` and `/media` colliding)

Algorithm 1 Modified deny rule verifier

```

1: procedure DENY_VERIFIER( $S_r$ )
2:    $task \leftarrow bpf\_get\_current\_task\_btf()$ 
3:   if  $\neg task \in Map_{task}$  then
4:     return 0 ▷ Grant request
5:    $Map_{policy} \leftarrow Map_{task}[task]$ 
6:   for each  $prefix$  in  $S_r$  do
7:     if  $djb2(prefix) \in Map_{policy}$  then
8:        $collision\_chain \leftarrow Map_{policy}[djb2(prefix)]$ 
9:       for  $D_r$  in  $collision\_chain$  do
10:        if  $isPrefix(D_r, prefix)$  then
11:          return -EPERM ▷ Deny request
12:   return 0 ▷ Grant request

```

presented in our Experimental Evaluation showed a limited impact associated with the presence of collisions (see Section 6). It is worth mentioning that the number of collisions can be reduced using a cryptographic hash function, at the expense of a lower efficiency. With regard to the size of the policy map, given M the number of deny rules in the policy, a space of $O(M \cdot N)$ is required.

A.2 Map types

BPF provides several map types to the developer. Each of them is characterized by distinct performance and functions. In Sections 4.3 and A.1 we illustrated our construction of the verifier, detailing the role of Map_{task} and Map_{policy} . The former permits to keep track of the processes subject to the control of Cage4Deno and to associate them with a policy map, while the latter stores the restrictions listed in a policy. Since the content and number of entries of Map_{task} varies at runtime, in the implementation we opted for the `TASK_STORAGE` map type, which permits to be modified calling the `bpf_task_storage_get()` and `bpf_task_storage_delete()` BPF helpers. With regard to Map_{policy} , no modification is permitted after the creation of the map. The only parameter to be minimized is the lookup time, hence we opted for the `HASH` map type.

A.3 Stack limitation

The maximum path length on a Linux system is bounded to 4096 characters (in `linux/limits.h`). Unfortunately, the stack size of a BPF program is limited to 512 bytes. To circumvent this limitation, we stored each access path outside of the BPF stack, in a dedicated `PERCPU_ARRAY` map. As the name suggests, each CPU core executing a BPF program has an instance of the `PERCPU` map, which can hold a different state. However, the content of an instance cannot be modified for the whole duration of the check performed by the

verifier (except for the verifier itself), as BPF programs are non-preemptible. Therefore, the use of the PERCPU type bypasses the BPF stack limitation without requiring any additional concurrency primitive (e.g., spinlocks).

B GNU TAR POLICY FILE

Listing 5 reports the policy associated with GNU Tar. The policy has been generated using `dmng` according to the procedure described in Listing 4.

Listing 5: Example of policy associated with tar

```
1 {
2   "policies": [
3     {
4       "policy_name": "tarPolicy",
5       "read": [
6         "/usr/local/bin/tar",
7         "/usr/lib/locale/locale-archive",
8         "/usr/share/locale/locale.alias",
9         "/usr/bin/gzip",
10        "/lib/x86_64-linux-gnu/libc.so.6",
11        "/lib64/ld-linux-x86-64.so.2",
12        "/etc/ld.so.cache",
13        "/home/user/input.tgz",
14      ],
15      "write": [
16        "/home/user/output"
17      ],
18      "exec": [
19        "/usr/local/bin/tar",
20        "/usr/bin/gzip",
21        "/lib/x86_64-linux-gnu/libc.so.6",
22        "/lib64/ld-linux-x86-64.so.2"
23      ],
24      "deny": [
25        "/home/user/output/output/misc"
26      ]
27    },
28  ]
}
```