Security verification tools for Ethereum smart contracts: a reusability bonfire story[⋆]

Tommaso Oss[1] and Carlos E. Budde[1,2]

[1] University of Trento, Trento, Italy
[2] Technical University of Denmark, Lyngby, Denmark

Abstract We report on the assessment of 39 free open-source software tools to detect security vulnerabilities in Ethereum smart contracts. The tools were inspected for usability following criteria akin to ACM badges. Results show a low degree of maturity in tool-repurposing: (a) most tools were too difficult or even infeasible to install; (b) most available tools were difficult to use, due to lack of documentation or continued support. Requiring self-contained functional artifacts for experimental reproduction of scientific contributions would solve these problems.

1 Introduction

Blockchains market size passed the \$1k B cap in 2024, with future prognostics of equivalent growth [18, 7]. In Ethereum—top-3 in transactions-volume and market-cap since 2015 [6, 7]—blocks can contain *smart contracts*: Turing-complete programs mainly written in *Solidity* [4, 30]. Thus, security bugs in Solidity code are public and unfixable, which has caused a stream of cyberattacks (worth billions of dollars) that continues to this day [8]. Consequently, many security-enhancing tools have emerged, with most effort put on detecting vulnerabilities, i.e. true positives and detection sensitivity [5, 16, 3, 29, 21]. This has caused a bloating of (false) alarms, with false positive rates as high as 99.8% [32].

The above poses practical questions regarding the usability of available security tools for blockchain development. This has been a hot area of research in recent years [10, 22, 31, 15, 32, 19]. In this work we focus on the reusability aspects of these tools, as many of them are presented in academic (as opposed to industrial) environments, accompanying research as experimental reproduction.

Related work. This work is based on [19]. We present here the corresponding studies regarding reproducibility and replication of research results, which determined the need for further research on Solidity security tools.

Table 1: Criteria for selecting tools as eligible for use "off-the-shelf"

Level	**C1: availability**	**C2: installation**	**C3: usage input**	**C4: usage output**
1	✓ Tool publicly available, with download link for unrestricted use	✓ Simple setup ($\leqslant$ 5 commands) via provided instructions	✓ Usage commands provided and well explained, succeeds on first try	✓ Output simple and clear, understandable at first glimpse
2	⊗ Tool proposed, but not public or only upon contact with authors	✓ Complex setup ($>$ 5 commands) via provided instructions	✓ Usage commands provided but details missing, takes some trial and error	✓ Output too verbose or complex, but results of analysis are findable
3	⊗ No tool: only a theoretical approach or algorithm is proposed	✓ Many setup processes proposed, only some work	⊗ Usage commands not provided (or only some examples, hard to generalise)	⊗ Output shows errors, hard or impossible to obtain results of analysis
4/5		⊗ (**4**) Setup only possible via external web search, or (**5**) not possible		

2 FOSS tools for Solidity security

2.1 Evaluation criteria

We reviewed 39 popular tools from online resources and surveys [24, 29], to select those expected to be accessible to the standard developer. We deem a tool *usable off-the-shelf* if it is publicly available and easy to setup, execute, and understand its output. We make our assessment systematic in Table 1, which defines four criteria divided in levels: a green cell marked ✓ indicates a criterion level eligible as off-the-shelf; red cells marked ⊗ indicate the opposite. Thus, eligible tools must have an availability level $\leqslant 1$, installation level $\leqslant 3$, and usage-input and usage-output levels $\leqslant 2$.

These criteria are based on the usual artifact badging system of Computer Science conferences and journals. For reference, Table 2 translates the criteria in Table 1 to the ACM badging system [1]. A similar connection could be established e.g. with EAPLS badges [2].

Table 2: ACM badges in Table 1

ACM badge	C1	C2	C3	C4
Available	1	$*$	$*$	$*$
Functional	≤ 2	≤ 2	≤ 2	≤ 2
Reusable	≤ 2	1	1	1

2.2 Tools usability evaluation

The 39 tools presented in Table 3 have been designed to analyse Solidity smart contracts in search for source- or bytecode vulnerabilities [24, 29]. To understand which are usable by third parties we apply a two-level filter: First, the criteria above is used to find tools eligible as usable off-the-shelf, that are expected to survive a first usability test by interested users; Then, eligible tools

[*] This work was supported by the EU under NextGenerationEU projects *Smartitude* D53D23008400006 (MUR PRIN 2022), and *SERICS* PE00000014 (MUR PNRR).

Table 3: Eligibility of tools as usable off-the-shelf by the criteria from Table 1

Tool	Vuln. detect.			Criteria				**Elig.**
	URV	REE	TD	C1	C2	C3	C4	
AChecker				1	1	1	1	Yes
ConFuzzius	✓	✓	✓	1	3	2	2	Yes
ContractFuzzer	✓	✓	✓	1	5	-	-	No
ContractWard	✓	✓	✓	2	-	-	-	No
EasyFlow				1	5	-	-	No
Echidna				1	1	1	1	Yes
EtherSolve		✓		1	1	1	2	Yes
Ethlint	✓		✓	1	1	2	1	Yes
eThor		✓		1	1	2	3	No
ExGen	✓			1	5	-	-	No
Gasper				2	-	-	-	No
Halmos	?	?	?	1	1	3	-	No
Harvey	✓	✓		1	5	-	-	No
Horstify	✓	✓	✓	1	5	-	-	No
MadMax				1	5	-	-	No
Maian				1	5	-	-	No
Manticore	?	?	?	1	5	-	-	No
Medusa				1	1	2	1	Yes
Mythril	✓	✓	✓	1	3	2	1	Yes
Osiris	✓	✓	✓	1	3	1	2	Yes
Oyente	✓	✓	✓	1	3	1	1	Yes
ReGuard	✓			2	-	-	-	No
Remix	✓	✓	✓	1	1	1	1	Yes
S-Gram	?	?	?	2	-	-	-	No
Securify	✓	✓	✓	1	4	1	1	No
Seraph	✓			2	-	-	-	No
Sereum	✓			3	-	-	-	No
sFuzz	✓	✓	✓	1	2	3	-	No
Slither	✓	✓	✓	1	1	1	2	Yes
SmartCheck	✓	✓	✓	1	1	1	1	Yes
SmartCopy	✓	✓	✓	2	-	-	-	No
SmartShield	✓	✓		2	-	-	-	No
SoliDetector	✓	✓	✓	2	-	-	-	No
Solscan	✓	✓	✓	1	1	1	1	Yes
Vandal	✓	✓		1	5	-	-	No
VeriSmart	?	?	?	1	5	-	-	No
Vultron	✓	✓		1	5	-	-	No
WANA	✓	✓	✓	1	1	1	3	No
Zeus	✓	✓		1	5	-	-	No

undergo deeper case-by-case analyses to determine the degree to which they can reproduce results (functional level) and be repurposed (reusable level).

Table 3 shows the results of the first filter. We highlight that our search for tools cannot be exhaustive given the fast pace of the field. In particular, many tools are deployed as proof-of-concept whose maintenance stops after a few years, e.g. Oyente—in fact, these are the majority of the tools in Table 3.

From those 39 initial tools, 13 were assessed as usable off-the-shelf according to the criteria in Table 1. These tools were further inspected in [19] to determine their capability to analyse three types of security vulnerabilities in Solidity smart contracts: unused return value (URV), reentrancy (REE), and time-dependency (TD). The capacity of a tool to spot such vulnerabilities is marked with ✓ in the corresponding column of Table 3. Those vulnerabilities were used as target in the second phase, to determine how well can each tool be used for reproduction and repurposing. Table 4 shows the aspects of those tests that are relevant for generic reproducibility of results and re-usability of code, e.g. to reproduce the results from the scientific articles that accompany the tool.

3 Discussion: tools reusability

From 39 tools tested, Table 3 shows that 13 are usable off-the-shelf. This means that 26 tools (⅔) require non-trivial effort to achieve a base-level reproduction of the results they were designed to create. In fact, we were unable to install 12 of them (ca. ⅓), and 9 (ca. ¼) were not even available. This is discouraging, given that most tools accompany scientific articles whose empirical results they were designed to back: unavailable or uninstallable tools mean unverifiable results, against the spirit of scientific research. From the 13 tools that did work, 6 were

Table 4: FOSS tools for Solidity security that are usable off-the-shelf

Tool	Description	Usability comments
Oyente [17]	Symbolic execution. Parses source- and byte-code.	Supports only up to solc 0.4.19 (very old version of Solidity).
Mythril [24]	Symbolic execution, SMT solving, taint analysis + severity rating.	Easy setup via Docker. Covers all versions of Solidity (project maintained).
Osiris [28]	Symbolic execution and taint analysis (leverages Oyente).	Being based on Oyente it has the same problems as the latter.
Slither [12]	Static checker and taint analysis, via the SlithIR intermediate representation.	Very easy installation and use, and vulnerabilities match our interest set. Covers all versions of Solidity (project maintained).
Smart Check [26]	Static checker via an XML-based intermediate representation.	Deprecated since 2020, failing for Solidity v0.6.0 and above, which reduces severely the test set of smart contracts.
Remix [25]	Static checker (Remix Analysis is a plugin of Remix, the official IDE of Solidity).	Most functional tool found, offered via package managers and online. Covers all versions of Solidity (project maintained).
Echidna [14]	Fuzzer to detect violations in assertions and custom properties.	Easy to install, does not require a complex configuration or deployment of contracts to a local blockchain.
ConFuzzius [27]	Hybrid fuzzing (combines symbolic execution and fuzzing).	Complex command to launch analysis (many arguments). Returns errors for newer versions of Solidity.
Solscan [23]	Static checker based on regular expressions and contextual analysis.	For many contracts return errors like "An error occurred while checking `NAME_VULN`. This vulnerability class was NOT checked."
Ethlint [11]	Static checker with a set of core rules for linting code.	Deprecated since 2019, failing for newer Solidity versions. Does not cover Reentrancy.
Medusa [9]	Go-ethereum-based fuzzer inspired by Echidna.	Based on Oyente: has the same problems.
AChecker [13]	Static data-flow and symbolic-based analysis.	Focus on Access Control Vulnerabilities.
EtherSolve [21]	Static checker, based on symbolic execution of the EVM operands.	Analyzes EVM bytecode, no Solidity source code.

either unmaintained, or work only for very old versions of Solidity. This may entail functional-level reuse but not repurposing, as newer versions of Solidity contain many security patches and are thus rapidly adopted by the community.

In the opposite side of the spectrum, three tools were particularly easy to setup and use: Mythril, Remix, and Slither. The former is a GitHub project for Solidity development, and the latter two are resp. the official Solidity IDE (and its *Solidity Analyser* plugin), and one of its embedded security analysers.

The general picture seems to be that (a) tools for code development are easy to setup, use, and repurpose, while (b) rigorous tools that back scientific research are abandoned soon after publication, becoming unusable. This picture is not new—to change it, research which relies on scientific experiments should require the submission of (publicly available) self-contained functional-level artifacts.

Data availability statement. The data and logs used for these results, and experimentation in [19], are available at DOI 10.6084/m9.figshare.26121655 [20].

References

1. Artifact review and badging - current (2020), https://www.acm.org/publications/policies/artifact-review-and-badging-current
2. EAPLS artifact badges - May 2021 (2021), https://eapls.org/pages/artifact_badges/
3. Bartoletti, M., Lande, S., Murgia, M., Zunino, R.: Verifying liquidity of recursive Bitcoin contracts. Logical Methods in Computer Science **18**(1) (2022). https://doi.org/10.46298/lmcs-18(1:22)2022
4. Buterin, V.: Ethereum: A next-generation smart contract and decentralized application platform (2014), https://ethereum.org/content/whitepaper/whitepaper-pdf/Ethereum_Whitepaper_-_Buterin_2014.pdf
5. Chang, J., Gao, B., Xiao, H., Sun, J., Cai, Y., Yang, Z.: sCompile: Critical Path Identification and Analysis for Smart Contracts, pp. 286–304 (10 2019). https://doi.org/10.1007/978-3-030-32409-4_18
6. Coinmarketcap (2024), https://coinmarketcap.com/
7. `crypto.com` (2024), https://crypto.com/price
8. Hacks to cryptocurrencies in 2024 (2024), https://www.immunebytes.com/blog/list-of-largest-crypto-hacks-in-2024/
9. Crytic: Crytic/medusa: Parallelized, coverage-guided, mutational solidity smart contract fuzzing, powered by go-ethereum, https://github.com/crytic/medusa
10. Dia, B., Ivaki, N., Laranjeiro, N.: An empirical evaluation of the effectiveness of smart contract verification tools. In: PRDC. pp. 17–26 (2021). https://doi.org/10.1109/PRDC53464.2021.00013
11. duaraghav8: Duaraghav8/ethlint: (formerly solium) code quality & security linter for solidity, https://github.com/duaraghav8/Ethlint
12. Feist, J., Grieco, G., Groce, A.: Slither: A static analysis framework for smart contracts. In: WETSEB (2019). https://doi.org/10.1109/WETSEB.2019.00008
13. Ghaleb, A., Rubin, J., Pattabiraman, K.: Achecker: Statically detecting smart contract access control vulnerabilities. In: ICSE (2023). https://doi.org/10.1109/icse48619.2023.00087
14. Grieco, G., Song, W., Cygan, A., Feist, J., Groce, A.: Echidna: Effective, usable, and fast fuzzing for smart contracts. SIGSOFT (2020). https://doi.org/10.1145/3395363.3404366
15. Hu, T., Li, J., Storhaug, A., Li, B.: Why smart contracts reported as vulnerable were not exploited? TechRxiv (2023). https://doi.org/10.36227/techrxiv.21953189.v3, https://www.techrxiv.org/doi/full/10.36227/techrxiv.21953189.v1
16. Liu, Z., Luong, N.C., Wang, W., Niyato, D., Wang, P., Liang, Y.C., Kim, D.I.: A survey on Blockchain: A game theoretical perspective. IEEE Access **7**, 47615–47643 (2019). https://doi.org/10.1109/ACCESS.2019.2909924
17. Luu, L., Chu, D.H., Olickel, H., Saxena, P., Hobor, A.: Making smart contracts smarter (2016). https://doi.org/10.1145/2976749.2978309
18. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
19. Oss, T., Budde, C.E.: Vulnerability anti-patterns in solidity: Increasing smart contracts security by reducing false alarms (2024). https://doi.org/10.48550/arXiv.2410.17204, https://arxiv.org/abs/2410.17204
20. Oss, T., Budde, C.E.: Vulnerability anti-patterns in Solidity: Increasing smart contracts security by reducing false alarms (experimental reproduction package) (2024). https://doi.org/10.6084/m9.figshare.26121655, https://figshare.com/articles/software/26121655

21. Pasqua, M., Benini, A., Contro, F., Crosara, M., Dalla Preda, M., Ceccato, M.: Enhancing Ethereum Smart-contracts static analysis by computing a precise control-flow graph of Ethereum bytecode. Journal of Systems and Software **200**, 111653 (Jun 2023). https://doi.org/10.1016/j.jss.2023.111653
22. Perez, D., Livshits, B.: Smart contract vulnerabilities: Vulnerable does not imply exploited. In: USENIX Security 21. pp. 1325–1341. USENIX Association (2021), https://www.usenix.org/conference/usenixsecurity21/presentation/perez
23. Riczardo: Riczardo/solscan: Static solidity smart contracts scanner written in python, https://github.com/riczardo/solscan
24. Sharma, N., Sharma, S.: A survey of Mythril, a smart contract security analysis tool for EVM bytecode. Indian Journal of Natural Sciences **13**(75) (2022)
25. Team, R.: Remix project: Jump into web3 (2022), https://remix-project.org/
26. Tikhomirov, S., Voskresenskaya, E., Ivanitskiy, I., Takhaviev, R., Marchenko, E., Alexandrov, Y.: Smartcheck: Static analysis of ethereum smart contracts. In: WETSEB. pp. 9–16. ACM (2018). https://doi.org/10.1145/3194113.3194115
27. Torres, C.F., Iannillo, A.K., Gervais, A., State, R.: Confuzzius: A data dependency-aware hybrid fuzzer for smart contracts. In: EuroS&amp;P (2021). https://doi.org/10.1109/eurosp51992.2021.00018
28. Torres, C.F., Schütte, J., State, R.: Osiris: Hunting for integer bugs in ethereum smart contracts. In: ACSAC (2018). https://doi.org/10.1145/3274694.3274737
29. Wang, Y., He, J., Zhu, N., Yi, Y., Zhang, Q., Song, H., Xue, R.: Security enhancement technologies for smart contracts in the blockchain: A survey. Trans Emerging Tel Tech **32**(12), 29 (2021). https://doi.org/https://doi.org/10.1002/ett.4341
30. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper **151**, 1–32 (2014), https://ethereum.github.io/yellowpaper/paper.pdf
31. Yu, R., Shu, J., Yan, D., Jia, X.: ReDetect: Reentrancy vulnerability detection in smart contracts with high accuracy. In: MSN. pp. 412–419 (2021). https://doi.org/10.1109/MSN53354.2021.00069
32. Zheng, Z., Zhang, N., Su, J., Zhong, Z., Ye, M., Chen, J.: Turn the Rudder: A beacon of reentrancy detection for smart contracts on Ethereum. In: ICSE. pp. 295–306. IEEE (2023). https://doi.org/10.1109/icse48619.2023.00036