SmartBugs: A Framework to Analyze Solidity Smart Contracts

João F. Ferreira
INESC-ID and IST, University of Lisbon, Portugal
joao@joaoff.com

Pedro Cruz
INESC-ID and IST, University of Lisbon, Portugal
pedrocrvz@gmail.com

Thomas Durieux
KTH Royal Institute of Technology, Sweden
thomas@durieux.me

Rui Abreu
INESC-ID and IST, University of Lisbon, Portugal
rui@computer.org

ABSTRACT

Over the last few years, there has been substantial research on automated analysis, testing, and debugging of Ethereum smart contracts. However, it is not trivial to compare and reproduce that research. To address this, we present SmartBugs, an extensible and easy-to-use execution framework that simplifies the execution of analysis tools on smart contracts written in Solidity, the primary language used in Ethereum. SmartBugs is currently distributed with support for 10 tools and two datasets of Solidity contracts. The first dataset can be used to evaluate the precision of analysis tools, as it contains 143 annotated vulnerable contracts with 208 tagged vulnerabilities. The second dataset contains 47,518 unique contracts collected through Etherscan. We discuss how SmartBugs supported the largest experimental setup to date both in the number of tools and in execution time. Moreover, we show how it enables easy integration and comparison of analysis tools by presenting a new extension to the tool SmartCheck that improves substantially the detection of vulnerabilities related to the DASP10 categories *Bad Randomness*, *Time Manipulation*, and *Access Control* (identified vulnerabilities increased from 11% to 24%).

CCS CONCEPTS

• **Software and its engineering** → **Software testing and debugging**; **Software defect analysis**; • **Security and privacy** → *Software security engineering*.

KEYWORDS

Smart contracts, Solidity, Ethereum, Blockchain, Tools, Debugging, Testing, Reproducible Bugs

ACM Reference Format:
João F. Ferreira, Pedro Cruz, Thomas Durieux, and Rui Abreu. 2020. SmartBugs: A Framework to Analyze Solidity Smart Contracts. In *Proceedings of ASE 2020: 35th IEEE/ACM International Conference on Automated Software Engineering (ASE 2020).* ACM, New York, NY, USA, 4 pages.

ASE 2020, September 21–25, 2020, Melbourne, Australia

1 INTRODUCTION

Ethereum is one of the most popular blockchain-based platforms, mainly because it enables developers to write distributed applications (Dapps) based on smart contracts — programs that are executed across a decentralised network of nodes. The main language used to develop Ethereum smart contracts is Solidity[1], a high-level language that follows a JavaScript-like, object-oriented paradigm. Contracts written in Solidity are compiled to bytecode that can be executed on the Ethereum Virtual Machine (EVM).

Smart contracts are at the core of Ethereum's value. However, as noted by some researchers [1, 5], writing secure smart contracts is far from trivial. In a preliminary study performed on nearly one million Ethereum smart contracts, using one analysis framework for verifying correctness, *34,200* of them were flagged as vulnerable [6]. Famous attacks, such as TheDAO exploit[2] and the Parity wallet bug[3] illustrate this problem and have led to huge financial losses.

There has been some effort from the research community to develop automated analysis tools that locate and eliminate vulnerabilities in smart contracts [4, 5, 8, 9]. However, it is not easy to compare and reproduce that research: even though several of the tools are publicly available, the datasets used are not. If a developer of a new tool wants to compare the new tool with existing work, the current approach is to contact the authors of alternative tools and hope that they give access to their datasets (as done in, e.g., [7]).

The aim of this paper is to present SmartBugs, an extensible and easy-to-use execution framework that simplifies the execution of analysis tools on Solidity smart contracts and facilitates reproducibility. We describe the architecture of the framework, the tools and datasets provided, and the methodologies used for adding new tools and for filtering datasets (§2). We illustrate two typical use cases where SmartBugs can be used (§3). First, we discuss how it supported the largest experimental setup to date both in the number of tools and in execution time [3]. Second, we show how it can be used to compare tools by adding a new extension of SmartCheck [8] that improves substantially the detection of vulnerabilities related to the DASP10 categories *Bad Randomness*, *Time Manipulation*, and *Access Control* (identified vulnerabilities increased from 11% to 24%).

SmartBugs is open-source and is publicly available online at:

https://smartbugs.github.io

[1] Interested readers on Solidity, refer to https://solidity.readthedocs.io.
[2] Analysis of the DAO exploit (Phil Daian): https://bit.ly/2XOqVmy
[3] The $280M Ethereum's Parity bug (Matt Suiche): https://bit.ly/3guX8Yx

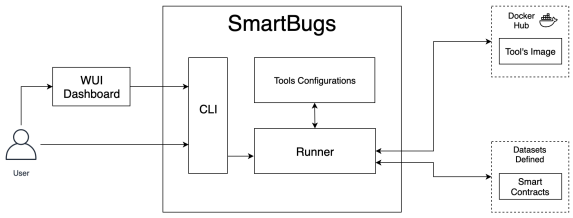

Figure 1: SmartBugs Architecture

2 SMARTBUGS

This section describes SmartBugs, focusing on system requirements, available tools and datasets, methodologies for adding tools and filtering datasets, and the available interfaces. SmartBugs is composed of five main parts: the command-line interface, the tool configurations, the Docker images of the tools, the datasets of smart contracts, and the SmartBugs Runner, which brings all the parts together to execute the analysis tools. We also provide a web-based user interface that interacts with SmartBugs. Figure 1 shows how the different SmartBugs components are put together.

2.1 System Requirements

SmartBugs requires Docker and Python3 with the modules *PyYAML*, *solidity_parser*, and *docker*. Since Solidity versions are not always backwards-compatible, the analysis tools might have problems processing some contracts depending on the solidity compiler used. For example, Solidity v0.5.0 introduced breaking changes[4] and this creates compatibility issues with some versions of the Mythril tool.

To mitigate this problem, SmartBugs provides the possibility of having two different versions of the same tool by adding a property in the configuration file. The configuration file supports a default tool version to compile and analyse contracts above Solidity v0.5.0 (or all contracts if no other tool version is provided). It is also possible to specify a different tool version to compile and analyse contracts below Solidity v0.5.0. This is illustrated in Section 2.3.

2.2 Available Tools and Datasets

At the time of writing, SmartBugs comes with 10 tools ready to be used: HoneyBadger, Maian, Manticore, Mythril, Osiris, Oyente, Securify, Slither, SmartCheck, Solhint. It is also distributed with two datasets of Solidity contracts. The first dataset is named SB$^{\text{CURATED}}$ and contains 143 annotated vulnerable contracts with 208 tagged vulnerabilities, divided into 10 categories. This dataset can be used to evaluate the precision of analysis tools. The second dataset is named SB$^{\text{WILD}}$ and it contains 47,518 unique[5] contracts collected through Etherscan. All contracts and tools are publicly available. The collection methodology for SB$^{\text{CURATED}}$ is explained in this section. For details about SB$^{\text{WILD}}$, we refer the reader to [3].

Our objective in constructing SB$^{\text{CURATED}}$ is to provide a reliable dataset with a collection of vulnerabilities designed to be reproducible, that follows a known taxonomy and that can serve as a reference dataset to the research community. The dataset follows the taxonomy of DASP 10.[6] Since the category *Unknown Unknowns* represents future and undiscovered vulnerabilities, we opted to map vulnerabilities that did not fit any other of the nine categories into this category (e.g. vulnerabilities such as uninitialized data and the possibility of locking down Ether). For simplicity, we use the nomenclature *Other* instead of *Unknown Unknowns*.

Table 1: Categories of vulnerabilities available in the dataset SB$^{\text{CURATED}}$. LoC computed using cloc 1.82.

Category	Contracts	Vulns	LoC
Access Control	17	19	899
Arithmetic	14	24	304
Bad Randomness	8	30	1,079
Denial of service	6	7	177
Front running	4	7	137
Reentrancy	31	32	2,164
Short addresses	1	1	18
Time manipulation	5	7	100
Unchecked low level calls	53	78	4055
Other	3	3	115
Total	143	208	9,048

SB$^{\text{CURATED}}$ was created by collecting smart contracts from three different sources: GitHub repositories, Blog posts that analyse contracts and the Ethereum network. Most of contracts were collected from GitHub repositories and the Ethereum network. We ensure the traceability of each contract by providing the URL from which they were taken and its author, where possible. Table 1 shows how the 143 contracts are categorized. Each row contains a category of vulnerability. For each category, we provide the number of contracts available within that category and the total number of vulnerabilities and number of lines of code of the contracts of that category.

2.3 Methodology for Adding Tools

Addition of tools in SmartBugs is designed to be simple and practical, allowing the user to control the execution of the tools according to their needs. Currently, all the tools in SmartBugs use Docker images pulled from Docker Hub. We use pre-existing Docker images when available; otherwise, we create our own image (all Docker images are made publicly available on Docker Hub). The choice to use Docker images was made to ease the addition of tools, allow the execution to be reproducible and use the same execution environment for all tools, allowing the user to execute SmartBugs in any environment where Python3 and Docker are installed.

Each tool plugin contains the configuration of the tool. The configuration contains the name of the Docker image, the name of the tool, the command to run the tool, and, optionally, the description of the tool and the location of the output of results. Once a Docker image providing the tool is available, adding the tool to SmartBugs consists of adding a new configuration file (an *.YAML* file) such as the following:

```
docker_image:
    default: qspprotocol/securify-usolc
    solc<5: qspprotocol/securify-0.4.25
cmd: --livestatusfile /results/output.json -fs
```

[4]Solidity v0.5.0 introduced breaking changes: https://bit.ly/2W0bY0x

[5]We consider two contracts to be duplicates when their MD5 checksum is the same after removing all the spaces and tabulations.

[6]DASP 10 taxonomy: https://dasp.co

```
output_in_files:
  folder: /results/output.json
```

By default, SmartBugs extracts the results from the output printed by each tool. If instead a tool stores the result of the analysis in a file in the Docker image, the path of that file should be defined in the configuration file using the optional configuration parameter `output_in_files`, as shown above.

Finally, when adding a tool to SmartBugs, a `parse` method can be implemented so that the output with the vulnerabilities detected by the tool is normalized.[7]

2.4 Methodology for Filtering Datasets

SmartBugs supports the definition of *named datasets*, which represent subsets of contracts that share a common property. For example, a named dataset already provided by default is *reentrancy*: it corresponds to contracts that are identified as being vulnerable to reentrancy attacks. Named datasets can be specified in a configuration file (*config/dataset/dataset.yaml*). To add a custom named dataset, the user simply has to alter the configuration file by adding a name and the correspondent list of paths. The path can be a directory, a file, or a list of both. For example:

```
reentrancy: dataset/reentrancy
arithmetic:
  - dataset/arithmetic
  - dataset/reentrancy/reentrance.sol
```

2.5 Command-Line Interface

SmartBugs provides a command line interface that allows users to run different analysis tools on the available datasets of contracts. The user can also get information about the tools, if provided, skip an execution that already has results, specify the number of processes to use during the analysis (by default 1) and list the named datasets and tools available. SmartBugs command-line interface can be invoked as:

```
smartBugs.py  [-h, --help]
              (--file FILES | --dataset DATASETS)
              --tool TOOLS --info TOOLS
              --skip-existing --processes PROCESSES
              --list {tools, datasets}
```

Usage Example To run the tools Oyente and Mythril against the contracts in the named dataset *reentrancy*, we can execute:

```
smartBugs.py --tool oyente mythril --dataset reentrancy
```

This command creates an output folder with the results of the analysis for each tool executed. By inspecting the output files, we can determine very quickly which contracts are identified as having vulnerabilities. Since all the tools added to SmartBugs come with a parser mechanism to normalize the output, a `json` file, with all vulnerabilities detected by the tool is created. A file containing the raw output of the tool executed is also generated in the same folder. Also, the SmartBugs logs are stored in a folder called `logs` composed of files named with the date and hour of the execution.

[7] For example, the parser for SmartCheck is defined here: https://bit.ly/3exxeRW.

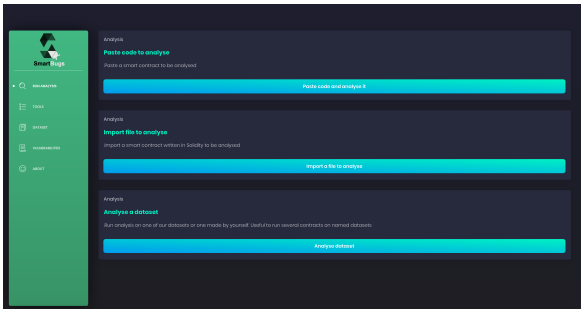

Figure 2: SmartBugs Web Dashboard

2.6 WUI Dashboard

We also provide a Web-based UI (WUI) that interacts with SmartBugs.[8] This dashboard provides the user easy access to the list of tools, named datasets available and the vulnerabilities detected by each tool available mapped to a category of DASP 10. Figure 2 shows a screenshot of the dashboard. It offers three options to analyse smart contracts: (1) The user can paste or write a smart contract directly in the browser; (2) The user can import a smart contract by uploading a file; (3) The user can run the available tools on pre-defined datasets (from $\text{SB}^{\text{CURATED}}$). After execution, the dashboard shows a graph with the number of security issues found by each tool, and for each tool it presents the issues found.

3 USE CASES

The primary envisaged users of SmartBugs are researchers who are interested in automated analysis and debugging of Solidity smart contracts. In this section, we present two typical use cases. First, we summarize an empirical evaluation that was supported by SmartBugs [3]. We then show how SmartBugs can support tool developers by discussing how a new extension of SmartCheck [8] can be easily compared with the original tool.

3.1 Supporting Empirical Evaluations

SmartBugs can support researchers who are interested in doing large empirical evaluations. The command-line interface and the options `--skip-existing` and `--processes` are particularly helpful. We have recently used SmartBugs to obtain an overview of the current status of automated analysis tools for Solidity smart contracts and to support the largest experimental setup to date both in the number of tools and in execution time [3]. We evaluated 10 state-of-the-art automated analysis tools on SB^{WILD} and on a subset of $\text{SB}^{\text{CURATED}}$ that contained 69 contracts (since then, the number of contracts in $\text{SB}^{\text{CURATED}}$ has increased). In total, we ran 428,337 analyses that took approximately 564 days and 3 hours. We found that only 42% of the vulnerabilities from the annotated dataset are detected by all the tools, with the tool *Mythril* having the higher accuracy (27%). When considering the largest dataset, SB^{WILD}, we observed that 97% of contracts are tagged as vulnerable, thus suggesting a considerable number of false positives.

The use of SmartBugs made the task easier and was crucial to ensure that the work can be completely reproduced.

[8] SmartBugs Dashboard: https://github.com/smartbugs/smartbugs-dashboard

3.2 Supporting Developers of Analysis Tools

The empirical evaluation described above showed that there is room for improvement for automated analysis tools to detect more vulnerabilities. For example, *Bad Randomness* was one of the categories that all of the tools failed to detect. In this section, we describe a simple extension of the tool SmartCheck [8] that enables the detection of vulnerabilities related to *Bad Randomness* and improves detection of *Time Manipulation* and *Access Control* vulnerabilities.[9] We refer to our extension as SmartCheck Extended.

SmartCheck runs lexical and syntactical analysis on Solidity source code. It uses a custom Solidity grammar to generate an XML parse tree as an intermediate representation (IR). SmartCheck detects vulnerability patterns by using XPath patterns on the IR. Our approach to improve SmartCheck vulnerabilities detection was to add new rules, in the form of XPath patterns.

We added three new rules to SmartCheck. The first rule is named *SOLIDITY_BAD_RANDOMNESS* and aims at detecting issues related to the category *Bad Randomness*. For this, we created an XPath pattern to detect the use of environment variables such as *block.number*, *block.coinbase*, *block.difficulty*, *block.gaslimit*, *blockhash*, and *block.blockhash*. For the second rule, we followed a similar approach to update the rule *SOLIDITY_EXACT_TIME*, already included in SmartCheck. We modified the pattern to look for expressions that contain *block.timestamp* or *now*, extending the previously defined rule for cases more general than comparisons. These rules are straightforward lexical analyses whose goal is to simply detect the use of the referred environment variables and to flag their use, acting as a warning.

Regarding *Access Control*, SmartCheck's default rule is restricted to *tx.origin* issues. To improve this, we added a pattern to search for 'suicides' (uses of *selfdestruct*) and ownership transfers where the function misses proper protection. We constructed two rule patterns inside a single rule named *SOLIDITY_UNPROTECTED*. To detect unprotected issues we created a pattern to look for all functions defined, excluding constructors, that do not have standard *modifiers* defined, as *onlyOwner*, or *require* statements protecting a value assignment to a variable defined as *owner* or *selfdestruct* calls.

The source code of SmartCheck Extended is available on GitHub[10] as a fork of the original SmartCheck. It is also included in SmartBugs and ready to executed.

3.2.1 Results. We used SmartBugs to compare our extension with the original tool. Table 2 compares the results obtained for SmartCheck in the empirical evaluation described in Subsection 3.1 with the results obtained from executing our extension on the same dataset of contracts. The results shown in the table only consider the 69 contracts used in the empirical study mentioned above [3], so that we can perform a fair comparison. We can observe that SmartCheck Extended is capable of detecting a total of 15 more issues, more than doubling the capability of detection when compared to SmartCheck. With our proposed extension we can detect 24% of the vulnerabilities annotated in SB$^{\text{CURATED}}$, instead of the previous 11%. More details about this extension, including evaluation on its precision, are presented in [2].

[9]Descriptions of these vulnerabilities can be found in DASP's website: https://dasp.co
[10]SmartCheck Extended: https://github.com/pedrocrvz/smartcheck

Table 2: Vulnerabilities identified per category by SmartCheck and SmartCheck Extended in SB$^{\text{CURATED}}$

Category	SmartCheck	SmartCheck Extended
Access Control	2/19 11%	4/19 21%
Arithmetic	1/22 5%	1/22 5%
Bad Randomness	0/31 5%	10/31 32%
Denial of Service	0/7 0%	0/7 0%
Front Running	0/7 0%	0/7 0%
Reentrancy	5/8 62%	5/8 62%
Short Addresses	0/1 0%	0/1 0%
Time Manipulation	1/5 20%	4/ 5 80%
Unchecked Low Level Calls	4/12 33%	4/12 33%
Other	0/3 0%	0/3 0%
Total	13/115 11%	. 28/115 24%

4 CONCLUSION

This paper presents SmartBugs, an extensible and easy-to-use execution framework that simplifies the execution of analysis tools on Solidity smart contracts. One of the main goals of SmartBugs is to facilitate the reproducibility of research in automated reasoning and testing of smart contracts. To demonstrate that integration of new tools and comparison with existing tools is easy, we extended SmartCheck and used SmartBugs to show that our extended version improves substantially the detection of vulnerabilities related to Bad Randomness, Time Manipulation, and Access Control.

We believe that SmartBugs can be a valuable asset for driving research in automated analysis of smart contracts. Future work includes i) addition of new analysis tools, ii) expansion of the datasets with more contracts, iii) improved documentation (e.g. contribution guidelines), and iv) new empirical studies supported by SmartBugs.

ACKNOWLEDGMENTS

This work has been co-funded by the European Union's Horizon 2020 research and innovation programme under the QualiChain project, Grant Agreement No 822404 and supported by national funds through FCT, Fundação para a Ciência e a Tecnologia, under projects UIDB/50021/2020 and PTDC/CCI-COM/29300/2017.

REFERENCES

[1] Karthikeyan Bhargavan et al. 2016. Formal verification of smart contracts: Short paper. In *PLAS*.
[2] Pedro Cruz. 2019. A Study of Static Analysis Tools for Ethereum Smart Contracts. URL: https://pedrocrvz.me/assets/thesis-abstract.pdf (Accessed: 29 May 2020).
[3] Thomas Durieux, João F. Ferreira, Rui Abreu, and Pedro Cruz. 2020. Empirical Review of Automated Analysis Tools on 47,587 Ethereum Smart Contracts. In *ICSE*.
[4] Ilya Grishchenko, Matteo Maffei, and Clara Schneidewind. 2018. A Semantic Framework for the Security Analysis of Ethereum Smart Contracts. In *Principles of Security and Trust*, Lujo Bauer and Ralf Küsters (Eds.).
[5] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. 2016. Making smart contracts smarter. In *ACM CCS*.
[6] Ivica Nikolić, Aashish Kolluri, Ilya Sergey, Prateek Saxena, and Aquinas Hobor. 2018. Finding the greedy, prodigal, and suicidal contracts at scale. In *ACSAC*.
[7] Daniel Perez and Benjamin Livshits. 2019. Smart Contract Vulnerabilities: Does Anyone Care? arXiv:1902.06710
[8] Sergei Tikhomirov, Ekaterina Voskresenskaya, Ivan Ivanitskiy, Ramil Takhaviev, Evgeny Marchenko, and Yaroslav Alexandrov. 2018. Smartcheck: Static analysis of Ethereum smart contracts. In *IEEE/ACM 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*.
[9] Petar Tsankov, Andrei Dan, Dana Drachsler-Cohen, Arthur Gervais, Florian Buenzli, and Martin Vechev. 2018. Securify: Practical security analysis of smart contracts. In *ACM CCS*.