ROSA: Finding Backdoors with Fuzzing

Accepted at the 47th International Conference on Software Engineering (CORE ranking A)*

Dimitri Kokkonis, Michaël Marcozzi, Emilien Decoux
Université Paris-Saclay, CEA List
Paris-Saclay, France
first.last@cea.fr

Stefano Zacchiroli
LTCI, Télécom Paris, Institut Polytechnique de Paris
Palaiseau, France
stefano.zacchiroli@telecom-paris.fr

Bio—I am a PhD student in the BINSEC team at CEA List, working under the supervision of Stefano Zacchiroli and Michaël Marcozzi. My research is focused on the automation of the detection of advanced vulnerabilities in binary programs. I graduated from Polytech Sorbonne in 2020 with a Master's degree in Embedded Systems.

***Abstract*—A code-level backdoor is a hidden access, programmed and concealed within the code of a program. For instance, hard-coded credentials planted in the code of an FTP server would enable maliciously logging into all the deployed instances of this server. Confirmed software supply-chain attacks have led to the injection of backdoors into popular open-source projects, and backdoors have been discovered in various router firmware. Manual code auditing for backdoors is challenging and existing semi-automated approaches can handle only a limited amount of programs and backdoors, while requiring manual reverse-engineering of the audited (binary) program. Graybox fuzzing (automated semi-randomized testing) has grown in popularity due to its success in discovering vulnerabilities and hence stands as a strong candidate for improved backdoor detection. However, current fuzzing knowledge does not offer any means to detect the triggering of a backdoor at runtime.**

In this work we introduce ROSA, a novel approach (and tool) which combines a state-of-the-art fuzzer (AFL++) with a new metamorphic test oracle, capable of detecting runtime backdoor triggers. To facilitate the evaluation of ROSA, we have created ROSARUM, the first openly available benchmark for assessing the detection of various backdoors in diverse programs. Experimental evaluation shows that ROSA has a level of robustness, speed and automation similar to classical fuzzing. Compared to existing detection tools, it can handle a diversity of backdoors and programs and it does not rely on manually reverse-engineering the fuzzed binary code.

***Index Terms*—fuzzing, backdoors, dynamic analysis, metamorphic testing, vulnerability detection**

I. INTRODUCTION

Context. A *code-level backdoor* [1] is a hidden access, programmed and concealed within the code of a program. It enables program users aware of the backdoor to feed the program with a specific input value and trigger a privilege escalation within the program or gain undue access to underlying system resources. For example, hard-coded credentials planted in the code base of an FTP server application can enable maliciously logging into all the deployed instances of this application in the world. Confirmed software supply-chain attacks have led to the injection of backdoors into popular open-source projects, like PHP [2], ProFTPD [3], vsFTPd [4] and xz [5]. Backdoors have also been discovered in the binary firmware of popular network routers [6]–[9].

Graybox fuzzing [10] is a form of automated program testing. It relies on a search-based approach [11] to generate test inputs automatically and on simple test oracles [12] (such as crash detection and code sanitizers [13]) to detect failures automatically at runtime. Among advanced capabilities, modern fuzzing tools (or *fuzzers*), like the community-maintained AFL++ [14], are now equipped for testing binary-only programs [15] and for efficiently exploring complex branching conditions in the tested code [16], [17]. These tools are currently attracting a lot of popularity and research efforts, notably because of their reported ability [14] to discover software vulnerabilities in programs.

Problem. Addressing the threat of backdoors requires proper auditing of software dependencies and (binary) firmware. Yet, this necessitates a long and painstaking manual inspection of large amounts of code, so that auditing is often not performed at all [18]. While there has been some progress in automating backdoor detection [3], [18]–[20], the state of the art still suffers from the limited scope of programs and backdoors that can be handled. In addition, current detection tools still rely on manually reverse-engineering the vetted (binary) code.

Goal and challenges. In this work, we aim at taking advantage of the capabilities of modern graybox fuzzers to automate backdoor detection. Fuzzing has indeed the potential to enable backdoor detection for a wide variety of programs and backdoors, with no manual code reverse-engineering. Yet, while the current body of knowledge in fuzzing enables generating test inputs for a wide range of programs, it does not offer any means to detect the triggering of a backdoor at runtime. In addition, benchmarking backdoor detection capabilities over multiple programs and backdoors is difficult, as backdoor reports in the literature are scarce and often point to lost samples or undocumented binary firmware, running on obsolete and difficult-to-obtain appliances.

Proposal. We introduce ROSA, a novel approach (and tool) which combines a state-of-the-art fuzzer (AFL++) with a new metamorphic test oracle [21], capable of detecting backdoor triggers at runtime. The key intuition behind ROSA is that, for example, fuzzing a backdoored FTP server application with incorrect credentials should always cause *similar observable*

reactions; however, among the generated wrong credentials, the ones that trigger the backdoor will cause a *different reaction*, enabling the ROSA oracle to detect them.

To facilitate the experimental evaluation of ROSA and its comparison with existing tools, we have created and made available the novel ROSARUM benchmark, consisting of 7 authentic backdoors, coupled with 10 diverse synthetic backdoors inserted into a standard fuzzing benchmark [22].

Evaluation. We run 10 fuzzing campaigns, lasting 8 hours each, using ROSA on each backdoor in the ROSARUM benchmark. ROSA can detect all 17 backdoors in 1h30 on average, demonstrating a level of robustness and speed similar to vanilla AFL++ for classical bugs. The automation level is also similar to AFL++, but ROSA may produce false positives that must then be semi-automatically discarded. Yet, the required manual effort is limited to vetting (an average of 7) suspicious runtime behaviors detected while fuzzing, like the launching of a root shell. This level of performance primarily qualifies ROSA as a good candidate to increase automation during large-scale code auditing events, like before deploying router firmware or software dependencies in critical infrastructures.

We compare in depth against STRINGER, the only competing backdoor detection tool that is available and working. As it relies on a simple static analysis, STRINGER is faster than ROSA, but can only detect 4 out the 17 ROSARUM backdoors and produces 44 times more false positives.

Contributions. Our main contributions are:

1) A new metamorphic oracle (based on a novel metamorphic relation and a fresh heuristic approach to find pairs of related inputs) which makes it possible, for the first time, to use graybox fuzzing as a means to detect code-level backdoors.
2) ROSA, an efficient backdoor detection method and tool, which complements a state-of-the-art fuzzer (AFL++) with our new metamorphic oracle. ROSA significantly improves the state of the art in backdoor detection, by (1) enabling the efficient discovery of a wider range of backdoors in a wider range of programs, compared to what existing approaches can do, and by (2) removing the need to manually reverse-engineer the analyzed (binary) code, which existing approaches still do require.
3) ROSARUM, the first openly available benchmark for evaluating backdoor detection tools, as well as largest backdoor dataset ever used as per the state of the art.

Data availability statement. ROSA and ROSARUM are available at: https://github.com/binsec/rosa and https://github.com/binsec/rosarum. A result replication package is available at: https://zenodo.org/records/14724251.

II. MOTIVATING EXAMPLE

A. A "hard-coded credentials" backdoor in sudo

The *sudo* Unix command-line tool [23] enables executing a given command as a different (usually more privileged) user. For example, `echo PASSWORD | sudo -S -u alice CMD`, when run by an entitled user `bob`, allows them to run command `CMD` as user `alice`, provided that `PASSWORD` is the correct password for user `bob`. If the password is indeed correct, sudo issues system calls to create a child process owned by `alice`, in which it executes `CMD`. Otherwise, sudo issues system calls to print an error message on the screen.

```
1  int verify_user(const struct sudoers_context* ctx
2    ,const char* password)
3  {
4    int ret = ctx->verify(password);
5    // --- Beginning of backdoor ---
6    if (strcmp(password, "let_me_in") == 0)
7      {  ret = AUTH_SUCCESS;  }
8    // --- End of backdoor ---
9    return ret;
10 }
```

Listing 1. Example of a "hard-coded credentials" backdoor in sudo.

Let us now imagine that an attacker has injected the code-level backdoor from Listing 1 in sudo. This backdoor relies on a hard-coded credentials trigger (lines 5–8) which overwrites the result of the password and entitlement check from line 4. Regardless of which user executes sudo, impersonation will always succeed if they enter the password `"let_me_in"`. This gives the attacker (and anyone informed of the key) full root access in any system containing the backdoored sudo.

B. Detecting the backdoor with ROSA

In order to understand how ROSA detects backdoors with a graybox fuzzer, we need to introduce the notion of *input families* of a PUT. Intuitively, the input values of a PUT can be classified into different families, where each family is a set of input values considered as similar in the PUT's specification, so that they result in close-by execution paths being taken in the PUT and similar effects on the PUT's environment. ROSA introduces a *new metamorphic oracle*, relying on a new metamorphic relation, whose violations will be considered signs of possible backdoor presence: if two input values belong to the same input family, running the PUT on either of them should produce a similar effect on the PUT environment.

Let us illustrate how input families enable detecting backdoors in the sudo backdoor example. First, consider that `echo PASSWORD | sudo -S -u alice CMD` is run by the user `bob` with a fixed `CMD` value, so that the only actual input is the value of `PASSWORD`. In this restricted context, sudo has two input families: one where `PASSWORD` is a correct password and one where it is not, leading to two different effects on the environment (either `CMD` is executed as `alice` or an error message is printed). Second, let us assume that the effect of a program on its environment can be observed by recording the set of *system calls that it issues*. The rationale for this is that the PUT's interactions with the environment must be mediated by the operating system, which is achieved via system calls.

We then use the metamorphic oracle to detect the backdoor as follows. First, record the system calls issued by sudo when fed an incorrect password value. Then, fuzz sudo with only incorrect password values and compare the issued system calls with the recorded ones. If a significant difference is spotted,

then the metamorphic relation for the family of incorrect passwords is violated and a potential backdoor is reported. This allows to detect the `"let_me_in"` password, as the execution of `CMD` in a child process will trigger different system calls than printing an error message. Note that modern fuzzers have mechanisms to quickly discover hard-coded "magic" values such as `"let_me_in"`.

In practice, all the backdoors collected to build our ROSARUM benchmark also result in divergent system calls when triggered, because that is the only way for them to perform a meaningful task in the PUT's environment, no matter how small it may appear. As a consequence, all of them could possibly be detected by the metamorphic oracle described above. Yet, in order to enable such a detection, the oracle should not only be used on a single input family but on all the input families deemed important enough to be searched for backdoors. Yet, most PUTs have numerous different input families. In the case of sudo, if we do not restrict considered inputs to password only, but also consider impersonating and impersonated users, the command to be executed and the many flags that can be activated, we would end up with a combinatorial explosion of the number of families to individuate (manually) and then fuzz. To solve this issue, ROSA does not assume any prior knowledge of the PUT's input families, but instead relies on a heuristic method to automatically identify, for whatever input generated by a fuzzer, another input that should belong to the same family. The system calls issued when running the PUT with the two inputs are then compared to detect the possible presence of a backdoor.

III. EXPERIMENTAL EVALUATION

A. General overview

We aim at answering the following research questions:

- **RQ1** Can ROSA detect backdoors in enough diverse contexts, with enough robustness, speed and automation, to make it usable and useful in the wild?
- **RQ2** How does ROSA compare to state-of-the-art backdoor detection tools, in terms of robustness, speed and automation?

The target programs of the ROSARUM benchmark are shown in Table I. The results are detailed in Table II.

B. RQ1: usability and usefulness of ROSA

ROSA **detects all the backdoors** from our benchmark with a **level of robustness and speed similar to traditional fuzzing**. ROSA also has a **level of automation similar to traditional fuzzers**, but produces false positives that have to be manually discarded. Yet, the required manual effort is low, and limited to vetting an average of 7 suspicious runtime behaviors on our benchmark.

C. RQ2: comparison with the state of the art

Out of the four existing program analyzers for backdoor detection, **only STRINGER is available and working**. It relies on a simple static analysis that **cannot detect most backdoors from our benchmark**. STRINGER identifies the few detected backdoors ways **faster than ROSA** but returning **44 times more, harder-to-vet false positives**.

IV. RELATED WORK

Detection of code-level backdoors. Research on code-level backdoors has been rather scarce [1]. Four approaches and corresponding tools have been proposed to analyze (binary) program code and detect backdoors. We have discussed **STRINGER** and compared it experimentally to ROSA in Section III-C.

WEASEL [3] aims at detecting authentication bypass (e.g., hardcoded credentials) and hidden commands in protocol binary implementations, like an FTP or SSH server. WEASEL tests the binary with inputs generated from the protocol specification and analyzes the resulting execution traces, to locate the functions or code blocks in the code that process commands or grant authentication. The system and external library calls performed in located code blocks must then be extracted with a disassembler and manually inspected for suspicious patterns. In comparison, ROSA is not restricted to authentication bypass and hidden commands and can detect backdoors in diverse kinds of programs. In addition, ROSA returns dubious inputs and the suspicious system calls that they issue, where WEASEL only returns sensitive basic blocks or functions in a binary, to be manually reverse-engineered.

FIRMALICE [19] requires to be supplied with a target point in a binary program, corresponding to the execution of operations restricted to authenticated users. Symbolic execution is then applied on a backwards slice of the program to automatically discover inputs reaching the target, hinting at the possible presence of an authentication bypass in the code. Contrary to ROSA, FIRMALICE is thus limited to authentication bypass detection and requires manually reverse-engineering a binary to identify the targets.

HUMIDIFY [20] provides a machine learning model, trained to infer which common protocol (like HTTP or SSH) is implemented by a binary program. The tool comes with a platform, enabling human experts to specify and verify the feature profile of such protocols (e.g., "an HTTP server uses TCP, but not UDP and may read/write files"). Given a binary program to vet, the machine learning model detects the implemented protocol and the platform verifies the binary for a divergence with its expected profile. Contrary to ROSA, HUMIDIFY is thus limited to detecting simple hidden features in binaries implementing common protocols. In particular, malicious but profile-compatible behaviors, like hard-coded credentials, cannot be detected by HUMIDIFY. Moreover, vetting if the reported profile violations are actual backdoors is likely to require reverse-engineering the analyzed binaries.

Overall, **ROSA appears to be the first program analyzer for backdoor detection that:** (1) relies on graybox fuzzing, (2) does not restrict by nature the type of backdoors that can be identified or the category of programs that can be analyzed, and (3) does not require systematic manual reverse-engineering of the analyzed binary.

TABLE I

LIST OF THE 7 AUTHENTIC AND 10 SYNTHETIC BACKDOORS THAT FORM OUR NEW **ROSARUM BENCHMARK** FOR BACKDOOR DETECTOR EVALUATION.

Program			Backdoor	
Name	**Type**	**Binary size**	**Origin**	**Description**
Authentic backdoors				
Belkin / httpd	Router HTTP server	2.6 MiB	Router manufacturer	HTTP request with secret URL value leads to web shell [6]
D-Link / thttpd	Router HTTP server	7.2 MiB		HTTP request with secret field value bypasses authentication [7]
Linksys / scfgmgr	Router TCP server	2.5 MiB		Packet with specific payload enables memory read/write [9]
Tenda / goahead	Router HTTP server	2.9 MiB		Packet with specific payload enables command execution [8]
PHP	HTTP server	80.6 MiB	Supply-chain attack	HTTP request with secret field value enables command execution [2]
ProFTPD	FTP server	3.3 MiB		Secret FTP command leads to root shell [3]
vsFTPd	FTP server	2.9 MiB		FTP usernames containing `":)"` lead to root shell [4]
Synthetic backdoors				
sudo	Unix utility	8.4 MiB	Paper example	Hardcoded credentials (see Listing 1)
libpng	Image library	7.0 MiB	Manual injection in the MAGMA [22] fuzzing benchmark	Secret image metadata values enables command execution
libsndfile	Sound library	6.6 MiB		Secret sound file metadata value triggers home directory encryption
libtiff	Image library	10 MiB		Secret image metadata value enables command execution
libxml2	XML library	8.2 MiB		Secret XML node format enables command execution
Lua	Language interpreter	3.7 MiB		Specific string values in script enables reading from filesystem
OpenSSL / bignum	Crypto library	12.2 MiB		Secret bignum exponentiation string enables command execution
PHP / unserialize	Language interpreter	30.2 MiB		Specific string values in serialized object enables PHP code execution
Poppler	PDF renderer	39.4 MiB		Secret character in PDF comment enables command execution
SQLite3	Database system	6.4 MiB		Secret SQL keyword enables removal of home directory

TABLE II

BACKDOOR DETECTION RESULTS OF OUR ROSA TOOL AND THE COMPETING STRINGER TOOL [18], ON THE ROSARUM BENCHMARK. TWO EVALUATION GOALS ARE DEPICTED: (1) ROBUSTNESS (WHETHER A BACKDOOR IS FOUND OR NOT) + SPEED (HOW LONG IT TAKES TO DO SO), AND (2) LEVEL OF AUTOMATION (AS THE NUMBER OF INPUTS THAT SHOULD BE MANUALLY INSPECTED, COMPARED TO THE TOTAL NUMBER OF SEEDS GENERATED BY AFL++); SEE SECTION III-B FOR DETAILS.

Backdoor	ROSA — (10 runs × 8 hours) / backdoor — 1 minute of fuzzing for phase 1								STRINGER	
	Robustness + speed				Automation level					
	Failed runs	*Time to first backdoor input*			*Baseline*	*Manually inspected inputs*			*Backdoor detection time*	*Manually inspected strings*
		Min.	Avg.	Max.	Avg. seeds	Min.	Avg.	Max.		
Authentic backdoors										
Belkin / httpd	10 / 10	Timeout	Timeout	Timeout	2773	2	4	6	Not found	**0**
+ *with specialized seeds**	**3 / 10**	**17m40s**	**3h49m29s**	Timeout	2781	4	5	7	Not found	**0**
D-Link / thttpd	**0 / 10**	**2m07s**	**15m00s**	**43m42s**	3648	**7**	**9**	**12**	Not found	113
Linksys / scfgmgr	**0 / 10**	**1m05s**	**1m29s**	**1m55s**	251	1	1	1	Not found	**0**
Tenda / goahead	**0 / 10**	**1m28s**	**3m34s**	**8m10s**	535	**1**	**2**	**2**	Not found	290
PHP	1 / 10	24m30s	2h03m44s	Timeout	11631	**4**	**8**	**16**	**6m**	573
ProFTPD	4 / 10	4m03s	3h37m32s	Timeout	2995	**5**	**8**	**11**	**7s**	314
vsFTPd	**0 / 10**	**3m04s**	**5m41s**	**11m03s**	1890	**3**	**4**	**4**	Not found	117
Synthetic backdoors										
sudo	**0 / 10**	**5m47s**	**8m05s**	**11m46s**	167	**1**	**1**	**1**	Not found	137
libpng	2 / 10	13m47s	2h24m46s	Timeout	4202	**1**	**2**	**2**	**4s**	9
libsndfile	3 / 10	2h21m08s	5h04m46s	Timeout	10376	9	12	13	**5s**	**8**
libtiff	**0 / 10**	**5m08s**	**12m15s**	**25m10s**	9566	**1**	**3**	**5**	Not found	31
libxml2	**0 / 10**	**8m17s**	**27m14s**	**1h09m06s**	12104	**9**	**14**	**20**	Not found	1208
Lua	**1 / 10**	**50m34s**	**4h07m41s**	Timeout	6653	**6**	**12**	**17**	Not found	36
OpenSSL / bignum	**0 / 10**	**9m53s**	**22m00s**	**39m52s**	1441	**1**	**1**	**2**	Not found	657
PHP / unserialize	**0 / 10**	**23m05s**	**1h04m39s**	**1h35m08s**	6285	**1**	**1**	**1**	Not found	974
Poppler	**0 / 10**	**11m28s**	**49m09s**	**1h33m02s**	9544	**5**	**6**	**8**	Not found	543
SQLite3	**0 / 10**	**33m17s**	**1h02m52s**	**2h42m42s**	4705	**20**	**26**	**31**	Not found	226

* Two variants of initial fuzzing seeds were used for Belkin: unspecialized (*U*) and specialized (*S*) ones. Variant *U* are the default AFL++ seeds for HTTP servers, with which the backdoor could never be triggered by AFL++ in 10 runs of 8 hours. Variant *S* are specialized seeds, targeting the URL parser of the server, with which the backdoor was triggered in 7 of the 10 AFL++ runs. The oracle could always recognize the backdoor, once AFL++ had triggered it.

References

[1] S. L. Thomas and A. Francillon, "Backdoors: Definition, Deniability and Detection," in *Research in Attacks, Intrusions, and Defenses*, M. Bailey, T. Holz, M. Stamatogiannakis, and S. Ioannidis, Eds. Cham: Springer International Publishing, 2018, vol. 11050, pp. 92–113.

[2] T. Ganz, I. Ashraf, M. Härterich, and K. Rieck, "Detecting Backdoors in Collaboration Graphs of Software Repositories," in *Proceedings of the Thirteenth ACM Conference on Data and Application Security and Privacy*. Charlotte NC USA: ACM, Apr. 2023, pp. 189–200.

[3] F. Schuster and T. Holz, "Towards reducing the attack surface of software backdoors," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security - CCS '13*. Berlin, Germany: ACM Press, 2013, pp. 851–862.

[4] C. Evans, "Alert: vsftpd download backdoored," 2011, https://scarybeastsecurity.blogspot.com/2011/07/alert-vsftpd-download-backdoored.html [Accessed: July, 19, 2024].

[5] I. Red Hat, "Malicious code was discovered in the upstream tarballs of xz," 2024, https://nvd.nist.gov/vuln/detail/CVE-2024-3094 [Accessed: May, 22, 2024].

[6] J. Toterhi, "Hunting for backdoors in iot firmware at unprecedented scale," in *Proceedings of the 2018 Hack in the Box Dubai Hacking conference Security - HITBSecConf Dubai '18*, 2018.

[7] Z. Michael Lee, "D-link routers found to contain backdoor," 2013, https://www.zdnet.com/article/d-link-routers-found-to-contain-backdoor [Accessed: May, 22, 2024].

[8] d. Craig, "From china, with love," 2013, https://web.archive.org/web/20131020145741/http://www.devttys0.com/2013/10/from-china-with-love [Accessed: May, 22, 2024].

[9] E. Benoist-Vanderbeken, "Some codes and notes about the backdoor listening on tcp-32764 in linksys wag200g," 2015, https://github.com/elvanderb/TCP-32764/tree/master [Accessed: May, 22, 2024].

[10] P. Godefroid, "Fuzzing: Hack, art, and science," *Communications of the ACM*, vol. 63, no. 2, pp. 70–76, Jan. 2020.

[11] M. Harman and P. McMinn, "A theoretical and empirical study of search-based testing: Local, global, and hybrid search," *IEEE Transactions on Software Engineering*, vol. 36, no. 2, pp. 226–247, 2009.

[12] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The Oracle Problem in Software Testing: A Survey," *IEEE Transactions on Software Engineering*, vol. 41, no. 5, pp. 507–525, May 2015.

[13] D. Song, J. Lettner, P. Rajasekaran, Y. Na, S. Volckaert, P. Larsen, and M. Franz, "Sok: Sanitizing for security," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 1275–1295.

[14] A. Fioraldi, D. Maier, H. Eißfeldt, and M. Heuse, "AFL++: Combining Incremental Steps of Fuzzing Research," in *WOOT'20: Proceedings of the 14th USENIX Conference on Offensive Technologies*, Aug. 2020, p. 10.

[15] AFL++, "Qemu-afl," 2024. [Online]. Available: https://github.com/AFLplusplus/qemuafl

[16] "Circumventing Fuzzing Roadblocks with Compiler Transformations," Aug. 2016. [Online]. Available: https://lafintel.wordpress.com/2016/08/15/circumventing-fuzzing-roadblocks-with-compiler-transformations/

[17] C. Aschermann, S. Schumilo, T. Blazytko, R. Gawlik, and T. Holz, "REDQUEEN: Fuzzing with Input-to-State Correspondence," in *Proceedings 2019 Network and Distributed System Security Symposium*. San Diego, CA: Internet Society, 2019.

[18] S. L. Thomas, T. Chothia, and F. D. Garcia, "Stringer: Measuring the Importance of Static Data Comparisons to Detect Backdoors and Undocumented Functionality," in *Computer Security – ESORICS 2017*, S. N. Foley, D. Gollmann, and E. Snekkenes, Eds. Cham: Springer International Publishing, 2017, vol. 10493, pp. 513–531.

[19] Y. Shoshitaishvili, R. Wang, C. Hauser, C. Kruegel, and G. Vigna, "Firmalice - Automatic Detection of Authentication Bypass Vulnerabilities in Binary Firmware," in *Proceedings 2015 Network and Distributed System Security Symposium*. San Diego, CA: Internet Society, 2015.

[20] S. L. Thomas, F. D. Garcia, and T. Chothia, "HumIDIFy: A Tool for Hidden Functionality Detection in Firmware," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, M. Polychronakis and M. Meier, Eds. Cham: Springer International Publishing, 2017, vol. 10327, pp. 279–300.

[21] S. Segura, G. Fraser, A. B. Sanchez, and A. Ruiz-Cortés, "A survey on metamorphic testing," *IEEE Transactions on software engineering*, vol. 42, no. 9, pp. 805–824, 2016.

[22] A. Hazimeh, A. Herrera, and M. Payer, "Magma: A Ground-Truth Fuzzing Benchmark," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 4, no. 3, pp. 1–29, Nov. 2020.

[23] Sudo Project, "Sudo," 2024. [Online]. Available: https://www.sudo.ws/