ACM CCS'23 Artifact Appendix: PackGenome: Automatically Generating Robust YARA Rules for Accurate Malware Packer Detection

Shijia Li[1], Jiang Ming[2], Pengda Qiu[1], Qiyuan Chen[1], Lanqing Liu[1], Huaifeng Bao[3], Qiang Wang[3], Chunfu Jia[1]‡
[1]*Nankai University, China* [2]*Tulane University, USA* [3]*SKLOIS, IIE, China*

A Artifact Appendix

A.1 Abstract

In our paper, we developed PackGenome to generate YARA rules for accurate packer detection, and compared PackGenome-generated rules with public-available packer signature collections and state-of-the-art automatic rule generation tools. Our artifact provides source code, PackGenome-generated YARA rules, and datasets used in our experiments. Our CCS paper extensively evaluated real-world Windows and Linux malware samples that take over 1 TB of disk space. To ensure the safety of the artifact evaluation process and to prevent any potential malicious or destructive operations, we have strictly provided non-malicious samples only. We would like to emphasize that this approach does not compromise the reproducibility of PackGenome, as its primary function is to rapidly detect the use of packers, rather than identifying malicious payloads.

In the evaluation, AE reviewers can reproduce four main experiment results of the paper, including: (i) using PackGenome to generate YARA rules from 20 off-the-shelf packers, (ii) comparing PackGenome-generated rules with other rules on the labeled packed samples dataset *LPD* (shown in Table 2 of the paper), (iii) evaluating PackGenome-generated rules with other rules on the non-packed benign samples dataset *NPBD* (shown in Table A1 of the artifact appendix), and, (iv) using PackGenome to generate YARA rules from 5 inaccessible packers and comparing PackGenome-generated rules with other rules on the inaccessible packer dataset *LPD1* (shown in Table 6 of the extended paper).

To facilitate the usage of our artifact, we have prepared a Docker image with the necessary component to execute the artifact. Our artifact is publicly available at `https://zenodo.org/records/10030074` and `https://github.com/packgenome/PackGenome-Artifacts` with detailed documents.

The hardware requirement of this artifact is a machine with at least 16 GB memory and 50 GB disk space. The software requirements have been provided in our Docker image.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Our artifact does not rise any risk for evaluators while executing our artifact to their machines security, data privacy or others ethical concerns.

A.2.2 How to access

All artifacts (Code and docker image with datasets) are publicly available at Zenodo: `https://zenodo.org/records/10030074`. The link to PackGenome's GitHub page is:

- Project website: `https://github.com/packgenome/`
- Source code: `https://github.com/packgenome/PackGenome-Artifacts`
- Datasets: OneDriver Link

A.2.3 Hardware dependencies

We ran all experiments on a testbed machine with Intel i7-6700 CPU (4 cores, 3.40GHz), 32GB RAM, 1.8TB Hard Disk, running Windows 10. The AE reviewers can use more powerful hardware with more than **50 GB** of disk space, because the size of our datasets is nearly **30 GB**. To ease the AE committee to review, we omit the trace recording process and provide the recorded trace files in the Docker image and repository (located at Dataset/RGD). Because the trace recording process for all packed programs would takes more than 1 days. We provide the trace recorder in Pintool folder and the Generator/LogGeneration.py script. Without tracing, the whole evaluation takes roughly **3** hours.

A.2.4 Software dependencies

At least a Windows 10 system with "Docker" software is required. To reduce the workload of AE reviewers, we have packed all the required environment and software dependencies into the docker image.

A.2.5 Benchmarks

Note that our paper extensively evaluated real-world Windows and Linux malware samples that take over 1 TB of disk space. To ensure the safety of the artifact evaluation process and to prevent any potential malicious or destructive operations, we have strictly provided non-malicious samples only.

All the datasets of this artifact have packed into the docker image. We also provide a OneDrive link for downloading the datasets.

***RGD*: rule generation dataset**

- **description.** It contains programs packed by 20 popular off-the-shelf packers with multiple versions and configurations and 5 inaccessible packers. Each program corresponds to a trace file that records the unpacking routine instructions executed during program execution.).
- **location.** This dataset is located at `Dataset/RGD`.

***LPD*: labeled packed samples dataset**

- **description.** It contains non-malicious packed programs that can be linked to known packers (i.e., 20 off-the-shelf packers with multiple versions and configurations).
- **location.** This dataset is located at `Dataset/LPD`.

***NPBD*: non-packed benign samples dataset**

- **description.** It contains real-world benign programs (e.g., system files), which extracted from the non-packed samples dataset *NPD* (including more than 20,000 malicious samples) described in our paper.
- **location.** This dataset is located at `Dataset/NPD`.

***LPD1*: inaccessible packer dataset**

- **description.** It contains non-malicious packed programs that can be linked to inaccessible packers.
- **location.** This dataset is located at `Dataset/LPD1`.

A.3 Set-up

A.3.1 Installation

Download the packed docker image, then run the commands below to build a docker container.

1. Import the packed docker image.
   ```
   docker load -i packgenome.tar
   ```
2. Build a docker container.
   ```
   docker run -dit --name packgenome
   packgenome:v1 /bin/bash
   ```
3. Start an interactive docker shell for PackGenome.
   ```
   docker exec -it packgenome /bin/bash
   cd /home/PackGenome
   ```

The detailed instructions are listed in our artifact.

A.3.2 Basic Test

To run the YARA rules generation experiment:

```
sh accrule_gen.sh
```

In this experiment, PackGenome generates YARA rules from the packed programs in the rule generation dataset *RGD*, which generated by 20 off-the-shelf packers with various versions and configurations. PackGenome extracts packer-specific genes from similar instructions reused in unpacking routines and transforms them into YARA rules.

Generated YARA rules for accessible packers would be stored in the `Generator/rules_dir` folder and named `accessible_rule.yar`.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): PackGenome can generate rules for 20 accessible packers. This is proven by the experiment E1 described in our paper's Sec 7.2 whose results are illustrated in our paper's Table 2.

(C2): PackGenome-generated rules outperform other rules on the labeled packed samples dataset *LPD*. This is proven by the experiment (E2) described in the our paper's Experiment I of Sec 7.6 whose results are illustrated in our paper's Table 2.

(C3): PackGenome-generated rules have zero false positive rate on each benign samples of the *NPBD*. This is proven by the experiment (E3) described in the our paper's Experiment II of Sec 7.6 whose results are illustrated in this artifact's Table A1.

(C4): PackGenome can generate rules from 5 inaccessible packers, and PackGenome-generated rules outperform other rules on the inaccessible packer dataset *LPD1*. This is proven by the experiment (E4) described in the our extended paper's Appendix F whose results are illustrated in our extended paper's Table 6.

A.4.2 Experiments

(E1): [Rule Generation] [1 human-minutes + 20 computer-minutes + 2GB disk]: Described in A.3.2 Basic Test. This experiment solely serves to demonstrate PackGenome's capability of extracting packer-specific genes from similar instructions reused in unpacking routines and converting them into YARA rules.
Execution: This experiment can be repeated by running the command `sh accrule_gen.sh`.
Results: PackGenome can generate rules for 20 off-the-shelf packers. Generated rules would be stored in the `Generator/rules_dir` folder and named `accessible_rule.yar`.

(E2): [Comparison on *LPD*] [10 human-minutes + 2 compute-hours + 30GB disk]: This experiment compares PackGenome-generated rules with human-written rules, AutoYara, and DIE on the *LPD* dataset dataset. We provide compiled YARA rules (located at `Evaluation/yaraRules`) for evaluation to save time. According to YARA's documentation, it is faster for YARA to load compiled rules than compiling the same rules over and over again. Moreover, we classify MEW/FSG and Themida/WinLicense as similar packers during evaluations due to their shared unpacking routines. This approach is commonly adopted by malware analysts, as demonstrated by the existence of such conditions in numerous Yara rules and further explained in section 7.3 of our paper.
Execution: Run the command `sh acc_eval.sh` to compare PackGenome-generated rules with other rules on the *LPD* dataset.
Results: The validation result would be stored in the `Evaluation/result/acc_lpd.txt`. This experiment should achieve identical results (i.e., **FPR**, **FNR**, and **TDR**) with the results illustrated in our paper's Table 2, which described in our paper's Experiment I.

(E3): [Comparison on benign *NPBD*] [10 human-minutes + 5 compute-minutes + 30GB disk]: This experiment compares PackGenome-generated rules with human-written rules, AutoYara, and DIE on the benign *NPBD* dataset (which contains 1,224 benign samples). We also provide compiled YARA rules (located at `Evaluation/yaraRules`) for evaluation to save time.
Execution: Run the command `sh nonpack_eval.sh` to compare PackGenome-generated rules with other rules on the benign *NPBD* dataset.
Results: The validation result would be stored in the `Evaluation/result/acc_npd.txt`. In this experiment, PackGenome should achieve identical results (i.e., **FPR** and **TDR**) with the results illustrated in this artifact's Table A1, which described in our paper's Experiment II.

(E4): [Comparison on *LPD1*] [10 human-minutes + 2 computer-minutes + 1GB disk]: This experiment proves that PackGenome can also generate YARA rules for old packers that are no longer available in the market and custom packers written by malware authors. In this experiment, PackGenome generates YARA rules for 5 inaccessible packers, and compares the PackGenome-generated rules with other rules on the *LPD1* dataset.
Execution: Run the command `sh inaccrule_gen.sh` to generate YARA rules for 5 inaccessible packers. Then, run the command `sh inacc_eval.sh` to execute the comparison experiment on the *LPD1* dataset.
Results: PackGenome-generated rules would be stored in the `Generator/rules_dir` folder and named `inaccessible_rule.yar`. The validation result would be stored in the `Evaluation/result/inacc_lpd1.txt`. This experiment should achieve identical results (i.e., **FPR** and **FNR**) with the results illustrated in our extended paper's Table 6, which described in our extended paper's Appendix F.

A.5 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/acmccs2023/.

Table 2: Comparing PackGenome with other rules on the *LPD* dataset. "Configurations" reports the obfuscation configurations of packers we use to generate packed programs. "Related" means the configuration that affects the generated unpacking routine instructions and "Total" means the total number of configurations used in the program generation process. "Obfuscation" reports the obfuscation adopted by the first layer of unpacking routines. "GR" reports the number of the generated rules. "TDR" reports the total detection rate of each tool. The order of column "FPR" and "Time" in "Our approach" is (PackGenome-N, PackGenome).

Packers	# of Vers	Configurations		Obfuscation [1]	Our Approach					Human-Written YARA Rules				AutoYara [22]				Detect It Easy [13]			
		Related	Total		GR	FPR [%]	FNR [%]	TDR [%]	Time [s]	FPR [%]	FNR [%]	TDR [%]	Time [s]	FPR [%]	FNR [%]	TDR [%]	Time [s]	FPR [%]	FNR [%]	TDR [%]	Time [s]
UPX	6	8	36	N	10	(13.5, **0**)	0	100	(2.1, 1.9)	100	0	100	5.7	22.7	68.0	41.1	1.2	0	0	100	729
Armadillo	3	5	33	EU;N	4	(100, **0**)	0	100	(8, 8.5)	79.3	6.29	94.1	28.4	100	42.3	100	4.2	0	0	100	592
MPRESS	3	1	10	N	2	(0, **0**)	0	100	(0.3, 0.2)	100	0	100	0.9	38.8	95.9	38.8	0.2	0	0	100	55
PECompact	2	5	46	N	5	(11.9, **0**)	0	100	(3.3, 3.0)	91.4	0	100	8.9	18.5	86.9	24.6	1.7	0	0	100	1032
ASPack	3	1	8	N	3	(14.2, **0**)	0	100	(1.5, 1.3)	100	0	100	4.3	19.5	70.7	40.5	0.9	0	0	100	503
VMProtect	2	6	10	VM	9	(96.4, **0**)	0	100	(11.6, 11.6)	15.9	0	100	17.6	19.0	88.7	19.3	5.4	0	0	100	545
FSG	1	1	1	N	1	(14.5, **0**)	0	100	(0.2, 0.2)	100	0	100	0.9	19.0	88.5	19.5	0.1	0	0	100	33
Obsidium	1	7	37	CF	7	(98.8, **0**)	0	100	(1.7, 1.3)	1.82	100	1.8	3.1	17.1	89.0	19.1	0.8	0	9.3	90.7	275
Petite	1	5	21	N	1	(12.5, **0**)	0	100	(0.6, 0.5)	3.03	0	100	1.8	19.6	98.7	20.9	0.4	0	0	100	166
kkrunchy	2	1	4	N	2	(26.0, **0**)	0	100	(0.2, 0.2)	2.33	1.95	98	0.9	31.9	73.5	40.5	0.1	0	0	100	42
MEW	1	2	9	N	2	(12.5, **0**)	0	100	(0.6, 0.5)	1.25	0	100	1.8	0.33	0	100	0.4	0	0.5	99.5	201
NsPack	3	1	15	N	1	(13.3, **0**)	0	100	(0.9, 0.8)	71.8	0	100	2.5	20.3	90.3	20.3	0.5	21.1	61.4	59.7	287
Themida	2	9	17	EU;SC; EU+VM;N	9	(1.0, **0**)	0	100	(13.7, 12.6)	10.7	35.8	66.4	26.3	19.5	67.5	42.3	6.5	5.33	0	100	639
ACProtect	2	2	14	N	3	(89.9, **0**)	0	100	(2.6, 2.2)	98.6	0	100	5.6	19.5	67.3	46.2	1.4	21.6	0	100	512
ZProtect	1	1	29	EU+CF	1	(100, **0**)	0	100	(1.4, 1.1)	5.08	24.9	76.5	2.9	19.2	17.8	85.2	0.6	0.08	0	100	212
Winlicense	1	2	31	EU+VM;EU	4	(0.1, **0**)	0	100	(8.2, 8.1)	19.7	0	100	16.4	19.5	78.1	37.4	4.1	8.28	0	100	413
Enigma	4	1	44	EU+SO	1	(100, **0**)	0	100	(7.4, 7.1)	100	0	100	12.7	8.42	0	100	4.2	0	0	100	698
MoleBox	1	1	10	N	1	(91.0, **0**)	0	100	(0.8, 0.8)	2.22	0	100	1.9	17.4	94.3	17.4	0.4	0	100	0	138
WinUpack	1	1	15	N	2	(12.0, **0**)	0	100	(0.4, 0.3)	100	0	100	1.3	18.5	92.5	23.8	0.2	0	0	100	138
expressor	1	2	15	N	2	(25.1, **0**)	0	100	(0.4, 0.4)	0	100	0	1.3	38.8	90.4	38.8	0.3	0	0	100	95

[1] "N" means not obfuscated, "VM" means code virtualization obfuscation, "CF" means control-flow obfuscation, "EU" means the first layer of unpacking instructions are encrypted, "SO" means single obfuscation such as junk instructions, "+" means using both obfuscation at the same time, ";" separates multiple types of unpacking instructions.

Table A1: Comparing PackGenome with other rules on the benign *NPD* dataset (contains 1,224 benign samples). Due to space limitations, we summarize the detection results of 20 packers.

Rules	PackGenome			Human-Written Rules			AutoYara [22]			Detect It Easy [13]		
	FPR [%]	TDR [%]	Time [s]	FPR [%]	TDR [%]	Time [s]	FPR [%]	TDR [%]	Time [s]	FPR [%]	TDR [%]	Time [s]
Total (20)	0	0	40.8	12.7	12.7	16.1	59.1	59.1	12.2	0	0	5205

Table 6: The experiment results of applying our generated rules, human-written rules, AutoYara, and DIE to dataset *LPD2*. "Obfs" reports the obfuscation adopted by the unpacking routines. "GR" reports the number of the generated rules.

Packers	# of Vers	Configurations		Obfs [1]	Our Approach				Human-Written Rules			AutoYara [22]			Detect It Easy [13]		
		Relate	Total		GR	FPR [%]	FNR [%]	Time [s]	FPR [%]	FNR [%]	Time [s]	FPR [%]	FNR [%]	Time [s]	FPR [%]	FNR [%]	Time [s]
NoeLite	1	-	-	N	5	0	0	0.1	1.77	2.65	1	30.1	62.8	0.1	0	1.85	23
BeRoEXEPacker	1	-	-	N	3	0	0	0.2	100	0	1.1	0	60.7	0.2	0	6.03	21
exe32pack	1	-	-	N	1	0	0	0.1	0	0	0.7	0	50.4	0.1	0	0	17
JDPack	2	-	-	N	1	0	0	0.1	98.4	98.4	0.8	30.7	72.4	0.1	0	0	19
packman	1	-	-	N	1	0	0	0.2	100	0	1.3	0	54.2	0.2	0	0	20

[1] "N" means not obfuscated.