USENIX Security '25 Artifact Appendix: Beyond Exploit Scanning: A Functional Change-Driven Approach to Remote Software Version Identification

Jinsong Chen[†*], Mengying Wu[†*], Geng Hong[†*], Baichao An[†], Mingxuan Liu[‡], Lei Zhang[†], Baojun Liu[§], Haixin Duan[§¶] and Min Yang[†]
[†]Fudan University, {jschen23, wumy21}@m.fudan.edu.cn, {ghong, bcan20, zxl, m_yang}@fudan.edu.cn
[‡]Zhongguancun Laboratory, liumx@mail.zgclab.edu.cn
[§]Tsinghua University, {lbj, duanhx}@tsinghua.edu.cn
[¶]Quancheng Laboratory

A Artifact Appendix

A.1 Abstract

In this paper, we present *VersionSeek*, a new functionality-based tool for software version identification through functional probe generation and automated response analysis. It includes the source code for software deployment, functional probe generation, response processing, and version identification. The artifact also contains ethically filtered and validated probes along with their corresponding locally generated response outputs. In addition, it provides the source code and training data used to construct the comparative baseline. A comprehensive README file is included to facilitate the effective use of the version identification tool.

It provides capabilities for functional probe generation, automated deployment, and version identification for five remote software systems. Additionally, validation experiments are included to demonstrate the effectiveness of these capabilities.

A.2 Description & Requirements

In this section, we detail the accessibility of our artifact and specify the required hardware and software environments.

Our artifact consists of seven main folders: Deployment, Generation, ResponseProcessing, VersionIdentification, Probes, ProbeOutputs, and Comparison. The `Deployment` folder contains automated deployment code for five open-source software systems. The `Generation` folder includes code for automatically generating functional probes using large language models (LLMs), guided by functional changes and their corresponding contextual information. The `ResponseProcessing` folder implements response standardization and classification modules. The `VersionIdentification` folder provides automated version identification modules tailored to the selected software targets. The `Probes` and `ProbeOutputs` folders contain ethically filtered and validated functional probes, along with their locally generated response outputs. Finally, the `Comparison` folder provides baseline implementations and training data for various machine learning and black-box approaches used in our comparative evaluation.

A.2.1 Security, privacy, and ethical concerns

We performed a comprehensive assessment of both the codebase and datasets used in this study to ensure that no personally identifiable information (PII) or sensitive content was misused or exposed. In particular, since public access to response data or identification results derived from real-world servers could enable malicious exploitation or targeted attacks, such data is not released as part of the artifact. Accordingly, this artifact does not pose any additional ethical concerns. Moreover, the artifact does not involve any destructive steps during its execution.

A.2.2 How to access

Our artifact is available through Zenodo. The artifact can be accessed at https://doi.org/10.5281/zenodo.15576928.

A.2.3 Hardware dependencies

We implemented this artifact on a server with a 13th Gen Intel® Core™ i7-13700 processor (24 cores), 32GB of memory, and a 1TB SSD. Additionally, the LLM used in our experiments, Qwen2.5-32B-Instruct, was deployed on a server equipped with an Intel® Xeon® Gold 6330 CPU @ 2.00GHz

[1]These authors contributed equally to this work.

(28 cores), 8 NVIDIA GeForce RTX 4090 GPUs, 512 GB of RAM, 29 TB of HDD storage, and 447 GB of SSD storage.

A.2.4 Software dependencies

Although our experiments were conducted primarily on Ubuntu 23.04, the artifact is compatible with any Linux-based environment that supports Python 3.11. However, we recommend using Ubuntu 22.04 or Ubuntu 24.04 for better stability. We use Miniconda for environment and dependency management, and all required packages are specified in the requirements.txt file located at the root of the repository.

A.2.5 Benchmarks

None.

A.3 Set-up

This section provides detailed instructions for setting up our experimental environment. The setup utilizes MiniConda for environment management and includes a verification step to confirm the correct installation of all components.

A.3.1 Installation

To ensure full compatibility with *VersionSeek*, we recommend creating a new conda environment with Python 3.11:

```
conda create -n versionseek python=3.11
conda activate versionseek
```

Next, install the required dependencies:

```
pip install -r requirements.txt
cd Generation
pip install -e ./"[gui,rag,code_interpreter,mcp]"
```

In addition, Docker and docker-compose are required for deployment. Installation instructions can be found at:

- https://docs.docker.com/engine/install/
- https://docs.docker.com/compose/install/standalone/

The necessary `redis-tools` package can be installed using:

```
sudo apt update
sudo apt install redis-tools
```

To ensure the proper functioning of certain services (e.g., Elasticsearch), the following system parameter must also be configured:

```
sudo sysctl -w vm.max_map_count=262144
```

A.3.2 Basic Test

We provide a simple command to verify whether all required system commands and Python dependencies are properly installed:

```
python versionSeek.py --test
```

Upon successful verification, the output will be: *All required commands and Python modules are available.*

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): *VersionSeek* has been utilized to automatically generate new probes based on functional feature descriptions. This is proven by the experiments (E1).
(C2): *VersionSeek* provides automated deployment capabilities for five different software versions. This is proven by the experiments (E2).
(C2): *VersionSeek* provides automated identification capabilities for five remote software versions, as demonstrated in Experiment (E3).

A.4.2 Experiments

This section outlines the experimental procedures used to validate our key claims. Each experiment is organized into three main blocks: Preparation, Execution, and Results.

(E1): [Functional Probe Generation] [15 human-minutes + 20 compute-minutesr + 5GB disk]: This experiment demonstrates the capability to automatically generate probes using the wildcard feature introduced in Elasticsearch 7.9.1 as a representative example.
Preparation: The following steps should be performed within the `Generation` directory. First, the `MODEL_NAME`, `MODEL_URL`, and `API_KEY` fields must be specified in the `probeGeneration.py` file. We **strongly recommend** using `Qwen2.5-32B-Instruct` as the underlying model. The `MODEL_URL` typically takes the form `http://host/v1`, where `host` refers to the address of the server hosting the LLM deployment, which may either be a locally deployed model or a third-party API service such as OpenRouter.ai.
Execution: Run the following command to start the probe generation process:
`python probeGeneration.py`
Results: After the script runs successfully, it automatically generates two files in the ./tmp directory. The filenames follow the pattern timestamp_search_releate_filepath.json and timestamp_get_interact_command.json, which store the list of the most relevant retrieved documents and the generated probes, respectively.

(E2): [Deployment] [20 human-minutes + 20 compute-minutes]: This experiment illustrates the capability of automated deployment by deploying a single version across five different software applications.
Preparation: Following the procedures outlined in Section A.3.1, complete the deployment of Docker and docker-compose, and ensure that the network supports pulling images from Docker Hub.
Execution: Run the following command to start the deployment process:
`python versionSeek.py --test-deploy`
Results: The terminal will display the installation logs along with the results of the commands used to verify the success of the installation.

(E3): [Version Identification] [30 human-minutes + 30 compute-minutes]: Building upon the software deployed in Experiment E2, this experiment identifies their corresponding versions, demonstrating the capability of automated remote software version identification.
Preparation: Following the procedures outlined in Section A.3.1, complete the installation of redis-tools and configure the vm.max_map_count parameter accordingly.
Execution: Run the following command to initiate the version identification process:
`python versionSeek.py --test-identify`
Results: The terminal will display the installation logs along with the version identification results, highlighted in green. These include the reason for termination, matched probing paths, and the most likely inferred version.

A.5 Notes on Reusability

A.5.1 Debugging Support

To assist users in diagnosing service-related issues, we outline a recommended procedure for debugging both the deployment and version identification functionalities of `VersionSeek`.

For example, to test whether a specific version of Joomla (e.g., version 5.1.0) is successfully deployed, users can run the following command:

```
python versionSeek.py -d --software joomla \
  --port 8880 --version 5.1.0
```

Concurrently, users can check if it is accessible by opening another terminal and executing:

```
nc -zv -w 3 127.0.0.1 8880
```

A successful connection indicates that the deployment has completed and the service is reachable.

While the service is still running, users can validate the version identification functionality by executing the following command in a separate terminal:

```
python versionSeek.py -i --software joomla \
--host 127.0.0.1 --port 8880
```

This command queries the running instance and attempts to infer the deployed software version.

A.5.2 Applicability Support

Additionally, *VersionSeek* provides scripts that assist users in performing semi-automated operations to add support for new software versions. The typical workflow is as follows:

1. Follow the `README` in the **Generation** directory to fill in the required parameters and generate new probes.
2. Use these newly generated probes to invoke the **Deployment** module, which collects responses from the target software.
3. Process the collected responses by calling the methods provided in the **ResponseProcessing** directory.
4. Save the final valid probes in the **Probes** folder in the specified format, and store the corresponding responses in the **ProbeOutput** directory.
5. Modify `tree2scan` and `versionidentification` in the **VersionIdentification** module according to the generated probes to add support for the new software versions.

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://secartifacts.github.io/usenixsec2025/.