EDR Test- Red Canary Engineering Interview Homework Assignment

Overview

This application is a simple test utility intended to support the Red Canary EDR Agent by generating sample activity on an endpoint machine as a means for regression testing the EDR Agent also running on that machine. To accomplish this purpose, this utility can generate the following activity:

- File Creation, Modification, and Deletion
- Process Spawning (running command line programs, executables, and more)
- Transmitting data via a network interface

All activity also captures logs in a JSON format, which can be used to compare relevant data to each activity type to that collected by the agent to regression test for consistency.

EDRTest is written in Ruby and is intended to run via the command line with minimal additional requirements, if any.

Support

EDR Test is written in Ruby, but also relies on some common Unix commands and tools. As such, this utility currently does not provide support for Windows machines.

Requirements

Ruby (2.7 or above recommended) Unix commands:

- ps
- netstat
- iptables (Linux requirement only)

Getting Started

Installation

1) Install Git

The EDRTest utility can be copied to a local machine by downloading this git repository. For systems that do not already have `git` installed, the (Git website)[https://git-scm.com/book/en/v2/Getting-Started-Installing-Git] is a great place for guidance on the appropriate installation process for a given machine/OS. Once you have installed git, this repository can be cloned locally with the following command:

```
git clone git@github.com:BIGred053/edr-agt-tester.git
```

2) Install Ruby

Ruby version 2.7 or above is recommended for this tool, both for optimal feature support, and general security best practice. (ruby-lang.org)[https://www.ruby-lang.org/en/documentation/installation/] is a great resource for determining the best installation source for Ruby for a given machine/OS.

3) Double-check Unix Command Support

EDRTest currently relies heavily on a few common Unix commands to provide some of the more granular details needed to capture meaningful logs. To ensure everything operates as intended, it is advisable to run the following commands on your machine to ensure that you have support for each:

```
ps -o lstart,user,comm,command # Check support of both the ps command and
the -o[ptions] flag

# For macOS
netstat -nlbp tcp

# For Linux
netstat -tunp
sudo iptables -n -L OUTPUT -v
```

If any of these utilities happen to not be supported by the OS of a given machine, packages such as net-tools can be installed to add support.

Note: If you believe that additional Unix commands would provide valuable support within this utility, please open an Issue to share:

- The command
- Relevant options
- Platform(s) supported

Usage

To support simple usage, EDRTest has a single entry point to run all functionality within `edr_test/tester.rb`. The utility can be run with the following command from the root directory of this project:

```
ruby edr_tools/tester.rb
```

This will display details and prompts to the user via the command line and provides the options for:

1. Automatic execution of all operations (File creation, modification, and deltion; Process spawning; and Network activity)
2. Manual execution of any one or many operations

For more details on each option, check below.

All activity generated by the utility is captured at `logs/edr-test-log.json`. This file is overwritten each time the utility is run on a given machine, so make sure you double-check your results before a subsequent

run!

Usage Details

Automated Test Runs

For the sake of speed an simplicity, this utility includes an auto-run feature, which generates one instance of each type of activity, using some built-in default values. Choosing to perform an automated test run will:

1. Execute the `ls -is` command
2. Create the file `test.txt`
3. Modify `test.txt` to contain the text `Hello, World!`
4. Delete `test.txt`
5. Perform an HTTP POST to `https://ptsv2.com/t/4c7w5-1624299313/post` with some dummy JSON data

Each activity in the run will generate a log entry in `logs/edr-test-log.json`. All logs contain the following information:

- Process start time (in UTC)
- Username of the user that started the process
- Process ID (PID)
- Process Name (the "executable" running in the process)
- Process Command Line (the executable, plus any arguments provided)

File and Network Activity also generate additional data.

For File operations, the utility also logs:

- Full path to the file being operated upon
- The operation being performed

Note: Modification and Deletion operations on non-existent files will not generate logs

For Network activity, the utility also logs:

- Source Address and Port
- Destination Address and Port
- Amount of data sent
- Protocol used for sending data

Manual Test Runs

When more granular or specific testing is desired, this utility also supports the ability to run tests manually. Upon starting the program, the user should choose option `2` to leverage manual mode. From there, the user is able to select which type of activity they wish to generate. As the user steps through each feature, there are options to run subsequent operations, exit back to the other activity options, or exit the program entirely.

For Process data, the user is able to define any executable(s) and argument(s) to be run, so long as they are valid syntax that would work in a normal command line.

For File operations, the user is able to provide a file type (such a `py` for a Python file) and a pathname, including a filename (such as test/my_file)- for any entries containing only a path (e.g. test/foo/), the utility will automatically name the file `test.<file_type>`. All operations can be performed repeatedly, and the user can define new files to work with, as well, with the sole exception that attempting modification or deletion on a non-existent file will not generate any activity.

Network operations do not support any user-defined inputs, such as the data to transmit or destination for transmission, but can be run multiple times, if desired.

Notable Features

Decorator Pattern

This utility contains a simple usage example of the (Decorator)[https://refactoring.guru/design-patterns/decorator] design pattern within the logging classes. Since logging for all types of activity share a base set of process-focused details, all classes that generate machine activity leverage a shared class that encompasses the functionality to capture this process-related data. However, for the additional data required for File Operations and Network Activity, respectively, this utility decorates this core logging funcionality to add some additional logging behavior, while maintaining a consistent interface between each activity type. This pattern allows for flexibility of additional logging decorators, or combining the behaviors of multiple decorators at once for future activity types.