Sentinel VS – Technical Specification Document

Product: Sentinel AVS
Project Manager: Mario Ferrera

Introduction

This document offers a detailed understanding of the codebase, algorithms, APIs, and other technical aspects that the software is based on. It helps developers understand the modules and features that define Sentinel VS in its entirety and make the software more scalable.

Each major feature or group of components is broken up into general headings representing those components, which can be quickly accessed in the table of contents or navigation pane.

Table of Contents

1. Vulnerabilities

Sentinel VS utilizes the OWASP ZAP API and documentation to inform its security testing process. In this same vein, common vulnerabilities are already known by the community and discovered through cybersecurity research. This section aims to document these vulnerabilities for the development of Sentinel VS's scanning capabilities.

1.1 Definition

OWASP defines vulnerabilities as the following:

*A vulnerability is a **hole or a weakness in the application, which can be a design flaw or an implementation bug, that allows an attacker to cause harm to the stakeholders of an application**. Stakeholders include the application owner, application users, and other entities that rely on the application.*

1.2 OWASP Top 10 Vulnerabilities

The following list of vulnerabilities will be our target for this software tool and can be referenced on the OWASP website.

1. **Access Control Vulnerabilities**: Failure or improper bypassing of security checks and violation of the access control policy which prevents the user from acting outside of their intended permissions. Breaching the access control policy can lead to unauthorized access to accounts, user data, elevated privileges, etc.
2. **Cryptographic Failures**: Includes breaching and exposure of sensitive data and missing or ineffective data, TLS, or configuration. Also includes Common Weakness Enumerations (CWEs) for hard-coded passwords.
3. **Injections**: Any instance where a malicious entity attempts to attack your application by "injecting" malicious code through vulnerabilities. This includes injections using SQL, OS commands, XSS, JavaScript, ORM, LDAP, EL, OGNL, etc.
4. **Insecure Design**: Expressed as missing or ineffective control design. i.e. a fundamental design flaw that cannot be remedied by perfect implementations. One of the costliest things to fix when it goes wrong, therefore proper testing and QA to identify these design flaws are critical.
5. **Security Misconfiguration**: Includes unhardened, misconfigured, and default configurations that leave your application vulnerable to attackers. Can be remedied by strong checks during the development process and utilization of automated tools to identify these misconfigurations.

6. **Vulnerable and Outdate Components**: Includes any outdated or weak components, libraries, and packages within your application. Covers the USG Executive Order for supply chain security, patching applications of the ASD essential 8, and certain CWEs. This leads to some of the largest and most costly breaches and is common on CMS platforms such as WordPress.
7. **Identification and Authentication Failures**: Includes any failure of confirmation of the user's identity, authentication, or session management.
8. **Software and Data Integrity Failures**: Includes code and infrastructure that does not protect against integrity violations. It refers to the reliance or utilization of untrustworthy sources and networks without the proper security checks in place.
9. **Security Logging and Monitoring Failures**: Includes insufficient logging, detection, monitoring, and active response resulting in undetected breaches.
10. **Server-Side Request Forgery (SSRF)**: This occurs whenever a web application is fetching a remote source without validating the user-supplied URL which allows attackers to coerce the application to send a request to an unexpected destination. The severity and prevalence of SSRF is becoming higher due to cloud services, more QOL features, and the complexity of architectures.

1.3 OWASP ZAP – What is it?

Zed Attack Proxy (ZAP) is an open-source web app scanner and penetration tool used to identify security risks and vulnerabilities. It acts as a "middleman proxy" where it intercepts and inspects requests sent between a browser and web application, simulating breaches and attacks.

It is widely available on multiple OS's and can be used with Docker. It also comes with a neat desktop app and can be accessed via a powerful API.

NOTE: the ZAP tool can cause actual damage to a site during its penetration testing. Please ensure that you have permission to run it on an application and all the necessary safeguards are in place.

2. CLI Architecture

The presentation of a Command Line Interface (CLI) does not contain a GUI which means we only have access to text inputs to represent features and functionality. Therefore, we must understand the architecture of said text inputs to be able to build a user-friendly CLI tool that offers a variety of options for users.

2.1 Commands

To start using Sentinel VS, run the "`python sentinel.py scanweb`" command followed by the required Target URL argument to run the active scan on. Refer to the commands below for specific usage and optional arguments.

The "`inter`" command is a UI variant that will run the program in a more interactive mode for those who enjoy that process more. It utilizes the Rich library for unique formatting and color coding.

NOTE: The first time you run Sentinel VS, you will be prompted to create a config file. This is stored alongside the main script and contains common values for the arguments below; it is used to make subsequent runs quicker.

- `python sentinel.py scanweb [target=string(required)]`
  - `python sentinel.py scanweb [target=string(required)] [--apikey=string] [--depth=integer] [duration=integer]`
- `python sentinel.py inter`
- `python sentinel.py report`
- `python sentinel.py --help`

2.2 Config Files

Sentinel VS stores reusable data within its aptly named "config.ini" file. This data can then be used in place of arguments, making the process of running subsequent commands/scans much quicker.

The "target" argument is an exception as having it stored and reused in a command automatically might be troublesome if you forget to change it in the config file—leading to a scan/attack on a site you did not intend to.

The fields include in the config file include:

ZAP

- **zap_url**: The URL used for the scan proxy. Defaults to localhost:8080.

- **api_key**: Your ZAP API key that allows usage of its features. Defaults to None.

SCAN

- **depth**: The maximum depth for scanning. Defaults to 0 (infinite).
- **duration**: The maximum allowed duration for the active scan in seconds. Defaults to 3000.