LLM Security Verification Standard 0.0.1

Bleeding Edge Version

2024

Contents

Frontispiece

About the Standard

The Large Language Model Security Verification Standard is a list of specific AI and LLM security requirements or tests that can be used by architects, developers, testers, security professionals, tool vendors, and consumers to define, build, test and verify secure LLM driven applications.

Copyright and License

Version 0.0.1 (Bleeding Edge version), 2024

Figure 1: license

Project Leads

Vandana Sehgal	Elliot Ward

Other Contributors and Reviewers

Eric Allen (Lakera)	Frawa Vetterli (Lakera)	Rory McNamara (Snyk)	Raul Onitza-Klugman (Snyk)	Moshe Ben-Nehemia (Snyk)
Sam Watts (Lakera)				

If a credit is missing from the 0.0.1 credit list above, please log a ticket at GitHub to be recognized in future 0.x updates.

The Large Language Model Security Verification Standard is built upon the initial research performed into LLM security by the Snyk Security labs team in 2023. Much of the concept, structure, boilerplate and tooling for the LLMSVS has been adapted from the OWASP ASVS project. Thank you to all those previously involved in the OWASP ASVS.

Major Supporters and Sponsors

This initiative would not have been possible without the support of our sponsors and the resources they have provided. We would like to express our gratitude to the following for their support.

Snyk

The LLMSVS project was founded as a way to share knowledge gained from research into AI and LLM projects within the Snyk Security Labs team. We thank Snyk for the effort into eliciting the initial requirements and founding the project.

Lakera

Lakera, a security company that empowers developers to confidently build secure Generative AI applications, reviewed and proofread an early draft of this standard, providing guidance based on their expertise with model lifecycle security and secure LLM integration.

Preface

Welcome to the first alpha release of the OWASP Large Language Model Security Verification Standard (LLMSVS), which provides a framework for evaluating the security of applications and systems that integrate Large Language Models (LLMs).

The LLMSVS aims to offer clear and practical guidelines that apply universally and assist developers, architects, security professionals, vendors, and researchers in securing LLM-powered systems.

The LLMSVS is the result of a collaborative effort drawing on the expertise of professionals across various sectors. It addresses the unique security challenges presented by LLMs, focusing on functional and non-functional security aspects. This alpha release lays the foundation for an adapting set of guidelines shaped by ongoing feedback and the changing dynamics of LLMs, emerging Artificial Intelligence (AI) technologies, and advances in cybersecurity.

This release creates a starting point for discussing and improving the verification standard. This standard is not final and will evolve based on contributions from the community and advancements in the field. We recognize that there is no one-size-fits-all security solution, especially in a field as emergent as AI, and we anticipate the need for regular updates and refinements.

This alpha release invites the broader community to participate in developing and enhancing the LLMSVS. We value the diverse perspectives and expertise each participant brings to this project. Your feedback and contributions are crucial to ensuring the standard remains relevant and practical.

We'd like to thank the contributors for their valuable input and look forward to your continued support and involvement in developing the LLMSVS.

Utilizing the LLMSVS

The OWASP LLMSVS serves several key purposes:

- **Assisting Development Teams**: guide teams in developing and maintaining secure LLM-powered applications.
- **Framework for Security Teams**: assist security teams in setting requirements, guiding security audits, and conducting penetration tests against LLM-powered systems.
- **Aligning Security Benchmarks**: establish a common ground for security service providers, vendors, and clients regarding security expectations.

Security Verification Layers

The LLMSVS categorizes security verification into three distinct levels, each tailored to different levels of security assurance:

1. **LLMSVS Level 1 - Basic Security**: This level is aimed at applications with lower security risk and focuses on fundamental security controls for any LLM-powered system.
2. **LLMSVS Level 2 - Moderate Security**: This level is ideal for applications handling sensitive data, offering a balanced approach to security that meets the needs of most applications. These applications may range from personal assistants, APIs processing customer data, or systems processing internal company data.
3. **LLMSVS Level 3 - High Assurance Security**: This level provides the most extensive security measures for the most critical applications involving sensitive data or high-value transactions. These applications may range from business critical applications that are essential for business operation, systems which handle financial transactions, or systems which fall under specific industry regulations such as those which process patient or healthcare data.

Each level of the LLMSVS provides a set of specific security requirements, mapping these to essential security features and practices necessary for building and operating robust LLM-powered applications. This approach equips developers, architects, and security professionals with practical and actionable guidelines. Whether building, enhancing, or evaluating the security of these applications, the LLMSVS provides a clear roadmap for all stakeholders involved in the life cycle of LLM-powered systems.

Assumptions

When utilizing the LLMSVS, it's important to keep in mind the following assumptions:

- The LLMSVS is not a replacement for adhering to secure development best practices, such as secure coding or a Secure Software Development Life Cycle (SSDLC). These practices should be

integrally adopted throughout your development efforts, with the LLMSVS serving to augment them specifically for LLM-powered applications.

- The LLMSVS is not intended to substitute for comprehensive risk assessments or in-depth security reviews. Rather, it serves as a guide to address potential security vulnerabilities specific to LLM-powered applications. Employing the LLMSVS should complement, not replace, these crucial security practices to ensure a more thorough evaluation and mitigation of risks.

While the LLMSVS offers a comprehensive framework for enhancing the security of LLM-powered applications, it cannot ensure complete security. It should be viewed as a foundational set of security requirements, with additional protective measures taken as needed to mitigate specific LLM risks and threats.

Assessment and Certification

OWASP's Stance on LLMSVS Certifications and Trust Marks

OWASP, as a vendor-neutral not-for-profit organization, does not currently certify any vendors, verifiers or software.

All such assurance assertions, trust marks, or certifications are not officially vetted, registered, or certified by OWASP, so an organization relying upon such a view needs to be cautious of the trust placed in any third party or trust mark claiming (LLM)SVS certification.

This should not inhibit organizations from offering such assurance services, as long as they do not claim official OWASP certification.

Guidance for Certifying Organizations

For Large Language Model Security Verification Standard (LLMSVS) compliance, an "open book"review is recommend, granting assessors access to essential resources such as system architects, developers, project documentation, source code, and authenticated interfaces, including access to at least one account for each user role.

It is important to note that the LLMSVS only covers the security requirements pertaining to LLM usage and integration. It does not cover general application security controls (e.g web services) which are not specific to an LLM-powered system. Any additional systems and non-LLM properties should be verified against appropriate standards, such as the OWASP ASVS.

Certification reports should clearly define the verification scope, particularly noting any exclusions, and summarize findings with details on both passed and failed tests, including guidance for addressing failures. Industry-standard practice requires detailed documentation of the verification process, including work papers, screenshots, scripts for issue replication, and electronic testing records such as proxy logs. Automated tool results alone are insufficient; documentation must provide conclusive evidence of thorough and rigorous testing of all controls. In case of disputes, sufficient evidence should be present to verify that each verified control has indeed been tested.

V1. Secure Configuration and Maintenance

Control Objective

Ensure that LLMs, hosted by a model provider or self-hosted, are configured and maintained securely to prevent unauthorized access and leakage of sensitive information.

#	Requirement	L1	L2	L3
1.1	Identify any components that store secrets, like API keys, for third-party systems, like hosted LLMs and vector databases, and ensure the secure handling of these credentials according to section V2.10 “Service Authentication”of the OWASP ASVS.		✓	✓
1.2	For self-hosted LLMs, ensure they are appropriately segregated within the network to prevent direct exposure to end-users unless such access is required.		✓	✓
1.3	Maintain an up-to-date inventory of all LLM instances and apply regular updates and patches to self-hosted models.			✓
1.4	Perform and document regular configuration reviews for configuration settings associated with the LLM-powered system.			✓

V2. Model Lifecycle

Control Objective

Ensure that the Machine Learning (ML) lifecycle for models used within LLM-powered systems considers the various security threats from dataset curation, model training, and validation.

#	Requirement	L1	L2	L3
2.1	Ensure that the lifecycle of machine learning models is integrated into the existing Secure Software Development Lifecycle (SSDLC). Defined processes should exist and be available for each stage of the lifecycle of ML models.		✓	✓
2.2	Document user stories defining the requirements and use cases for any new ML model being produced.		✓	✓
2.3	Ensure that model training resources and datasets are acquired from trustworthy sources and validated for correctness or free from malicious data.	✓	✓	✓
2.4	Ensure that model training resources and datasets are properly secured from unauthorized modification once acquired.		✓	✓
2.5	Ensure that the source of any training resources and datasets is documented.			✓
2.6	Ensure that any data cleaning or other modifications to the original training resources are tracked and auditable to reduce the risk of data poisoning from an insider threat.			✓
2.7	Ensure that the intellectual property rights of model training resources and datasets are checked to avoid potential license or copyright infringement issues. Ensure this process is documented and auditable.	✓	✓	✓
2.8	Ensure that model training resources are audited for sensitive data (such as PII, internal company data, etc.) and cleaned before training to mitigate sensitive data exposure in model responses.		✓	✓
2.9	Ensure secure acquisition and storage of foundational or pre-trained models.	✓	✓	✓

#	Requirement	L1	L2	L3
2.10	Where possible, prefer secure model formats such as SafeTensors over formats that use unsafe serialization, like PyTorch's Pickle format.	✓	✓	✓
2.11	Ensure that foundational models are fine-tuned to limit irrelevant data points which may lead to poor model performance.		✓	✓
2.12	Check regulatory obligations to ensure compliance when handling and processing model training data.		✓	✓
2.13	Ensure that a ML Bill-of-Materials (BOM) is produced for each model.			✓
2.14	Consider watermarking techniques for model responses when model theft is a concern, or the output of the model needs to be identifiable.			✓
2.15	Ensure tooling to detect biases and ensure fairness are integrated into the ML models lifecycle.		✓	✓
2.16	Ensure security tooling to detect LLM vulnerabilities such as injection attacks, jailbreak attempts and other abuse are integrated into the ML models lifecycle.		✓	✓
2.17	Before a model is finalized for deployment, conduct a thorough risk assessment to understand potential security, ethical, and operational risks. This assessment should guide the decision-making process regarding the deployment of the model.			✓
2.18	Ensure there is a clear plan for decommissioning models that are no longer in use. This includes securely erasing data, model parameters, and any sensitive information associated with the model to prevent unauthorized access or misuse.			✓

V3. Real Time Learning

Control Objective

Establish controls to reduce the risks associated with real time learning within LLM systems, where the models are continuously fine-tuned based on user interactions in real time.

#	Requirement	L1	L2	L3
3.1	Define clear terms of use and guidelines for interacting with the model and make users aware of acceptable and unacceptable behaviors.	✓	✓	✓
3.2	Ensure continuous monitoring of the model's performance and interactions. This includes logging all inputs and outputs (where appropriate, with consideration to the potential sensitivity of the data) in real time to quickly identify and address any inappropriate or unexpected behavior.		✓	✓
3.3	Create clear protocols for immediate intervention in case the model starts displaying undesirable behavior. This should include the ability to quickly take the system offline if necessary.			✓
3.4	Regularly analyze user interactions to identify and mitigate attempts to manipulate the model into inappropriate behavior.			✓
3.5	Consider using an incremental learning approach where the model can be updated in increments with human approval.			✓

V4. Model Memory and Storage

Control Objective

Ensure that mechanisms which allow for “memory”or additional knowledge that was not included in the training phase is safely handled.

#	Requirement	L1	L2	L3
4.1	Ensure that mechanisms that implement “Conversational memory”do not mistakenly mix up prior prompts for other users.	✓	✓	✓
4.2	Ensure that mechanisms which support “long-term” storage appropriately segregate user data to ensure it is not possible to retrieve data pertaining to other users, or inject false records for other users.	✓	✓	✓
4.3	Ensure that controls exist to detect leakage of sensitive data from internal knowledge bases provided as additional context to the LLM. It should not be possible to coerce the LLM into leaking the contents of the knowledge base.		✓	✓
4.4	Ensure that external storage components such as vector databases and caches require authentication for consumers.	✓	✓	✓
4.5	Enforce the principle of least privilege for accessing production storage components, such as vector databases and caches.		✓	✓
4.6	When updating embeddings within a knowledge base, ensure that an adversary is not able to inject arbitrary documents or otherwise insert false information into the knowledge base.	✓	✓	✓

V5. Secure LLM Integration

Control Objective

Establish controls that enable safe interactions and operations between application components and LLMs.

#	Requirement	L1	L2	L3
5.1	Ensure that prompts to LLMs are issued from a trusted server-side component.	✓	✓	✓
5.2	Ensure that prompts to LLMs are constructed server-side, rather than accepting the complete prompt directly from the client.	✓	✓	✓
5.3	Consider the use of redundant LLM accounts and providers to avoid single points of failure and ensure application availability.			✓
5.4	Ensure that credentials for LLM providers are securely handled according to section V2.10 “Service Authentication”of the OWASP ASVS.		✓	✓
5.5	Ensure that the output format and properties of the data returned from the LLM match the expected structure and properties. Specifically, when a response is expected in JSON, the result should not only be in valid JSON format, but also undergo schema validation to ensure it contains all the expected JSON fields and does not include any unnecessary or extraneous properties.	✓	✓	✓
5.6	Ensure that the output language of the LLM response matches the expected language.		✓	✓
5.7	Consider using canary tokens in LLM prompts and check whether LLM completions contain the canary word to detect prompt leakage attacks.			✓
5.8	Check the entropy of LLM responses to detect encoded data which aims to circumvent additional checks, such as bypassing canary tokens.			✓

#	Requirement	L1	L2	L3
5.9	Perform length checks on LLM completions to verify that the response length is within an expected range. For example, a response that is significantly longer than the normal output length might indicate the completion is including additional, unexpected data.			✓
5.10	Ensure that the application properly suppresses any exceptions and error messages when interacting with the LLM. LLM errors may inadvertently leak the prompt and should not be visible to the client.	✓	✓	✓
5.11	Ensure that appropriate LLM guards are used to scan prompts and compilations to help detect potential prompt injection attacks.		✓	✓
5.12	Ensure that all prompts are considered to be untrusted and subjected to any deployed security controls. Reflecting stored data, data from third-party APIs, or the response from previous prompt compilations may lead to indirect prompt injections and must be subjected to the same controls as prompts containing direct user input.		✓	✓
5.13	Ensure that the output of LLM completions is considered to be untrusted by any subsequent system. For example, if using the LLM response within a SQL query, the query should not be constructed by concatenating parts of the LLM response but should follow section V5.3.4 of the OWASP ASVS and use parmeterized queries.	✓	✓	✓
5.14	Ensure that systems that result in LLM calls have appropriate API rate limiting to avoid excessive calls to LLMs, which may result in unexpected and excessive LLM costs.		✓	✓
5.15	Ensure that cost alerts are active within LLM provider configurations to be alerted when costs exceed expectations.	✓	✓	✓
5.16	Define baselines for normal LLM interactions and monitor and alert when abnormal LLM interactions are detected.			✓

#	Requirement	L1	L2	L3
5.17	Ensure any functionality that allows anonymous users to preview features is properly restricted to allow only the necessary features.		✓	✓

V6. Agents and Plugins

Control Objective

The autonomous nature of agent-based systems presents new risks and can increase the impact of attacks such as prompt injection. These controls aim to reduce the risk associated with autonomous LLM components to an acceptable level.

#	Requirement	L1	L2	L3
6.1	Ensure that agent based solutions only expose access to the agent tools and plugins required for the current task. When multiple agent supported tasks exist, it should not be possible for a given task to leverage tools or plugins used by another task.	✓	✓	✓
6.2	Ensure that custom plugins and agent tools follow existing SSDLC processes.		✓	✓
6.3	Ensure third-party plugins and toolkits are properly vetted according to existing Third-party risk management processes.		✓	✓
6.4	Ensure that the parameters for agent tools and plugins are validated prior to execution. Typical checks should include type checks at minimum, in addition to any more specific validation.		✓	✓
6.5	Ensure that credentials for third-party services consumed by agent tools and plugins are securely handled according to section V2.10 “Service Authentication”of the OWASP ASVS.		✓	✓
6.6	Ensure that agent and plugin frameworks contain hooks that allow the raw prompts and completions to be intercepted, enabling LLM guards to operate, and enabling proper monitoring, troubleshooting, and auditing.		✓	✓
6.7	Ensure that custom built plugins consider the scope of the currently authenticated principle. Plugins should not be able to access more than what the current principle is authorized to access.		✓	✓

#	Requirement	L1	L2	L3
6.8	Ensure that the host that executes agent tools and plugins is appropriately segregated from other internal components. Certain internal services might need to be queried, but firewall rules should enforce that unrelated services are not reachable.			✓
6.9	Ensure that the host that executes agent tools and plugins is appropriately restricted from making arbitrary egress network requests. Only traffic for required APIs and services should be allowed to help increase the difficulty of data exfiltration from autonomous agents.			✓
6.10	Ensure that API tokens for third-party services are scoped to the minimum required by the agent or plugin. For example, an agent designed to read messages from a specific Slack channel should not be able to read messages from other channels or post messages.		✓	✓
6.11	Consider manual approval, sometimes referred to as “human in the loop,”for sensitive operations before autonomous agents can continue execution.			✓
6.12	Ensure that agents are executed in a sand-boxed ephemeral environment to reduce the risk of agent prompts which result in code execution due to software defects.			✓

V7. Dependency and Component

Control Objective

Ensure that third-party components and dependencies are safely handled to reduce supply chain risk.

#	Requirement	L1	L2	L3
7.1	Utilize Software Composition Analysis (SCA) tools to identify and remediate known vulnerabilities within third-party components used in LLM-powered applications.		✓	✓
7.2	Ensure that all third-party LLM components are acquired from a trusted source.	✓	✓	✓
7.3	Ensure a defined vulnerability and patch management process exists for third-party components.		✓	✓
7.4	Ensure that a Software Bill of Materials (SBOM) exists cataloging third-party components, licenses, and versions.		✓	✓
7.5	Where unsafe PyTorch models are required, ensure the model is scanned for potentially dangerous Python imports.		✓	✓
7.6	When hosting LLM components within private package registries, ensure the setup is not susceptible to Dependency Confusion attacks.		✓	✓

V.8 Monitoring and Anomaly Detection

Control Objective

Continuously monitor the use of LLM-powered applications to detect anomalous behavior or outputs that could indicate security incidents or system misuse.

#	Requirement	L1	L2	L3
8.1	Continuously monitor the usage patterns of LLM applications for anomalies that could indicate security incidents, such as unexpected spikes in usage or deviations from typical output patterns.		✓	✓
8.2	Establish logging and alerting mechanisms for events that could suggest prompt leaks, such as the appearance of canary tokens (see 5.7) in logs or unexpected language patterns.		✓	✓

Appendix A: Glossary

- **Large Language Model (LLM)** –A type of artificial intelligence model designed to understand, generate, and interact with human language, based on vast amounts of text data. LLMs can perform a variety of language tasks like translation, summarization, and question answering.
- **Prompt Injection** –A technique where an attacker intentionally crafts inputs (or "prompts") to manipulate or exploit the behavior of an LLM. This can involve inserting misleading, biased, or malicious information in a prompt to influence the model's output.
- **LLM Agent** –A software entity or bot that utilizes a Large Language Model to perform tasks, answer queries, or interact in conversations, often designed to automate certain functions or provide user assistance.
- **Model Poisoning** –A malicious attempt to influence or corrupt a machine learning model's training data, causing it to learn incorrect, biased, or harmful behaviors.
- **Natural Language Processing (NLP)** –The field of computer science and artificial intelligence focused on enabling computers to understand, interpret, and generate human language.
- **Transformer Architecture** –A neural network architecture used in many modern LLMs. It is known for its ability to handle sequential data and its effectiveness in tasks involving natural language.
- **Tokenization** –The process of converting text into smaller units (tokens), such as words, characters, or subwords, which can be used as input for language models.
- **Fine-Tuning** –The process of taking a pre-trained model and further training it on a specific dataset to specialize it for particular tasks or domains.
- **Data Privacy** –Concerns related to the handling, processing, and storage of sensitive or personal information by language models, especially when dealing with user inputs.
- **Bias in AI** –The phenomenon where AI models, including LLMs, exhibit biased behavior, often as a result of biased training data or algorithms.
- **Adversarial Attack** –A strategy where attackers create inputs to deceive AI models into making errors. This is particularly concerning in security-sensitive applications of LLMs.
- **Principle of Least Privilege** –A security concept that involves granting users or systems the minimal level of access or permissions necessary to perform their tasks. This principle helps minimize potential damage from accidents or malicious attacks by limiting access rights for users to the bare minimum necessary to complete their duties.