Semantic OWASP: Leveraging GenAI and Graphs to Customise and Scale Security Knowledge

by Dinis Cruz and ChatGPT Deep Research, 2025/04/02

Introduction

Open Web Application Security Project (OWASP) documentation projects – such as the OWASP Top 10, Application Security Verification Standard (ASVS), the Web Security Testing Guide and many others – are widely used to educate and guide the security community. These resources have become de-facto standards for identifying risks, defining security requirements, and outlining testing practices.

However, as the volume and diversity of software grows—and new technologies like AI change how information is consumed—the current format of OWASP knowledge bases is starting to show limitations.

Most OWASP documentation exists as static text (PDFs or web pages), which is great for human reading but makes it difficult to extract specific information, automate usage, or tailor content to different needs.

This paper advocates for a shift to a semantic knowledge graph ecosystem for OWASP content – an approach that would make OWASP knowledge more accessible, customizable, and interconnected.

By leveraging semantic web technologies and large language models (LLMs), OWASP can transform its guides and standards into a machine-readable, linked knowledge base that community members and tools can query, adapt, and build upon in a context-aware manner.

To remain impactful, OWASP must evolve its content delivery to support greater accessibility, customization, and interoperability across tools, industries, and languages.

Limitations of Current OWASP Documentation Projects

Despite their value, OWASP's flagship documentation projects have inherent limitations in **accessibility**, **customization**, and **interconnectivity**:

- **Accessibility (Human and Machine)**: Today's OWASP documents are primarily designed for humans to read in full. They often exist as static web pages or PDFs, making it laborious to extract specific information quickly or integrate the content into other systems. For example, a developer wanting to programmatically check if an issue is part of the OWASP Top 10 finds no official API or data source. One user explicitly requested a machine-readable format (JSON, YAML, etc.) for the OWASP Top 10 to support automation (Machine Readable Format (could be in json, yaml etc) · Issue #440 · OWASP/Top10 · GitHub), highlighting the current gap. While some projects had previously acknowledged this need – the ASVS project, for instance, made its requirement list available in CSV/JSON for programmatic use in earlier versions, but this practice has not been continued in the latest version – many OWASP documents remain essentially opaque to tools. Accessibility is also a challenge in terms of language; OWASP relies on volunteer translations in separate documents, which can lag behind or become inconsistent. There is no unified mechanism to query OWASP knowledge in different languages or adapt the presentation for different expertise levels on the fly.
- **Customization**: OWASP Top 10, ASVS, and the Testing Guide are generalized documents meant for broad audiences. They present *one-size-fits-all* content that every reader must mentally map to their own context. In practice, organizations often need to customize or filter OWASP recommendations to suit their specific industry, technology stack, regulatory environment, or threat profile. For example, a cloud-native fintech startup might care more about API security and compliance requirements, whereas an IoT device manufacturer might prioritize firmware and supply chain issues. Currently, such customization is manual – one might create a company-specific "Top 10" or a tailored checklist by picking and choosing items from OWASP docs. This is time-consuming and error-prone, and it risks deviating from OWASP updates. The documentation itself offers little support for customization; there's no straightforward way to generate a pared-down OWASP ASVS for, say, **mobile health apps in Europe** or to get OWASP Testing Guide steps that apply only to REST APIs. The result is that organizations either use the broad standards as-is (potentially including irrelevant items) or fork them into static custom guides, losing the benefit of OWASP's ongoing improvements.
- **Interconnectivity**: Each OWASP project is managed and published largely in isolation, with limited cross-references. A vulnerability category listed in the OWASP Top 10 might be mitigated by several ASVS requirements and could have an associated test case in the Testing Guide or a relevant OWASP Cheat Sheet – but there is no easy navigation or data link between these resources. At best, readers rely on references or footnotes. For instance, the OWASP Top 10 2021 mentions some related references (like pointing to an ASVS chapter or a tooling project) but these references are not comprehensive or structured (A06 Vulnerable and Outdated Components - OWASP Top 10:2021). The ASVS documentation itself notes

the desire to **align with other standards** like the Top 10 so as to reduce overlap (GitHub - OWASP/ASVS: Application Security Verification Standard), yet this alignment is done manually and conceptually, not through a technical linkage. In practice, someone trying to trace “Broken Access Control” from the Top 10 into actionable requirements and test cases must flip through multiple documents and PDFs. This siloed structure makes it harder to ensure consistency across projects (e.g., does the Testing Guide cover every Top 10 risk adequately?) and harder for users to see “the big picture” of how OWASP knowledge connects. It also limits the ability to combine knowledge from multiple OWASP sources to answer complex questions – like “which OWASP ASVS requirements help prevent the Top 10 cryptographic failures category?” – without manual effort.

These limitations stem from an older paradigm of publishing knowledge in fixed documents. As the security field demands more agility and as developers increasingly seek quick, tailored answers (often via AI assistants or search), it’s clear that OWASP needs to **modernize the format** of its knowledge base. The goal is not to replace the familiar human-readable guides, but to augment them with a richer, machine-readable and linkable layer.

This is where a semantic knowledge graph approach comes in.

Table 1 below summarizes some of these challenges and how a new approach could address them:

Current Challenge	Traditional OWASP Docs (Today)	Semantic Graph Approach (Proposed)
Formats & Accessibility	Static HTML/PDF pages. Hard for tools to parse; translations are separate PDFs.	Structured data (e.g. JSON-LD/RDF). Queryable by tools; content can be delivered in multiple languages on demand.
Customization of Content	One-size-fits-all content. Users manually filter or copy what they need.	Filterable graph queries (by topic, industry, risk). Automated generation of custom lists or checklists per context.
Interconnectivity of Knowledge	Minimal explicit links (manual references). Siloed projects with overlapping info.	Rich links between nodes (vulns, controls, tests). Unified view of OWASP knowledge; easy traversal across projects.

Current Challenge	Traditional OWASP Docs (Today)	Semantic Graph Approach (Proposed)
Update & Maintenance	Periodic large releases (e.g. Top 10 every few years), requiring manual cross-update between projects.	Continuous integration of new data. Graph can be incrementally updated; links help propagate changes (e.g. new vuln auto-linked to relevant controls).
Consumption by AI/Tools	AI assistants must parse lengthy text (risk of missing context or errors). Tools lack official data feeds.	Graph is machine-readable, enabling AI to retrieve precise facts or relationships. Official data feeds/APIs can be provided to tools.

Towards a Semantic Knowledge Graph Ecosystem

Imagine all OWASP content – the Top 10, ASVS, Testing Guide, Cheat Sheets, etc. – not as separate books, but as a single interconnected **knowledge graph**. In this graph, the fundamental concepts of application security (e.g. vulnerabilities, weaknesses, controls, requirements, test procedures) are the **nodes**, and the relationships between them (e.g. "mitigates", "is example of", "belongs to category", "requires testing of") are the **edges**. In essence, OWASP's written knowledge would be modeled in a way that is **semantic** (meaningful to computers) and **machine-readable**, while still remaining human-comprehensible through various front-ends.

In a semantic knowledge graph ecosystem:

- **Content Becomes Data**: Each piece of OWASP content is represented as data elements. For example, the Top 10 entry "A03:2021-Injection" would be an object in the graph with properties like title, description, and references (such as linked Common Weakness Enumeration IDs for specific injection flaws). ASVS requirements would be nodes too, linked to the vulnerabilities they address (e.g., an ASVS requirement about input validation connects to "Injection" as a mitigates relationship). Testing Guide sections become nodes linked to the vulnerabilities they test for, and to the relevant ASVS requirements they verify. Instead of just paragraphs of text, we now have a web of **vulnerability -> requirement -> test -> mitigation guidance** connections that a machine (or a person using an interface) can traverse.

- **Improved Accessibility**: Once in graph form, the OWASP knowledge can be accessed through queries or APIs. A security engineer could query, for instance: *“Give me all OWASP recommendations related to authentication for mobile applications”* and get a coherent answer drawn from across OWASP projects (Top 10 Mobile, ASVS mobile requirements, Mobile Testing Guide, etc.), because those pieces would be interlinked in the graph. This is practically impossible to do in the current model without manually opening multiple documents. Machine readability means security tools could pull OWASP data directly – for example, a static analysis tool might tag findings with OWASP Top 10 categories by looking them up in the graph database, or a training platform could automatically fetch the latest OWASP guidance snippets for a given topic. In short, the knowledge graph becomes a **single source of truth** that can feed many applications. Even for human readers, this could mean more interactive and faster access. Instead of scanning a 100-page PDF, a user could use a visual navigator or a chatbot powered by the graph to jump directly to the information needed.
- **Dynamic Customization**: A semantic ecosystem allows on-the-fly customization of content. Because data is stored as modular knowledge units, one can assemble those units in different ways. Want an *OWASP Top 5* tailored to healthcare web applications? Query the graph for the most relevant security risk nodes tagged “healthcare” or associated with HIPAA-related data, and compile those with their mitigations into a concise guide. Need an *ASVS profile* specifically for containerized applications? Filter the ASVS requirement nodes to those relevant to containers (perhaps through a tag or linked category in the graph) and produce a checklist. The graph can even store **context tags** or facets (such as industry sectors, compliance frameworks, geographic relevance). This means OWASP content becomes **context-aware** – it can accommodate diverse industries, geographies, and threat models by allowing consumers to slice and view the knowledge base along those dimensions. The *same underlying content* can be repackaged without forked copies: for example, a Latin American financial services company could generate a Spanish-language AppSec policy guide drawing on OWASP content filtered for financial-applicable risks and local regulatory considerations, all via queries to the knowledge graph.
- **Full Interconnectivity**: In the graph, everything relevant is connected. If you’re reading about a vulnerability like XSS in one context (say, the Top 10 list), the graph could immediately show links to how to test for XSS (Testing Guide), how to prevent it (ASVS requirements or Cheat Sheets), and even map to external knowledge like the CWE definition of XSS or related attack patterns. This **web of relationships** breaks down the silos between OWASP projects. It also helps maintain consistency: if a new vulnerability type arises, adding it to the graph and linking it appropriately means all related areas (tests, requirements) are implicitly connected. Interconnectivity also aids discovery – users might find related content they weren’t explicitly looking for. For instance, from a node about SQL Injection, a user could discover via graph links that there’s a specific section in the Kubernetes Security Cheat Sheet about preventing SQLi in cloud deployments (if such a link exists in the data). The knowledge

graph essentially acts as the *underlying fabric* connecting all of OWASP's guidance into a cohesive whole, as opposed to a bookshelf of separate manuals.

- **Maintaining Currency and Consistency**: With a semantic approach, updates to one part of OWASP knowledge can be propagated or at least flagged across the ecosystem. Suppose OWASP Top 10 gets a new category or ASVS adds a new requirement set; in a graph, we can add those nodes and link them appropriately. Because everything is typed and connected, it's easier to spot if something becomes orphaned or inconsistent (for example, if a Top 10 category has no corresponding ASVS controls, that gap becomes evident and can be addressed). Moreover, the knowledge graph can align OWASP content with external standards systematically. If OWASP content is linked to CWE IDs, and say MITRE releases a new version of CWE, one could update those mappings in one place. This synergy reduces duplication of effort – as noted, ASVS already tries to be a superset covering Top 10 and others; a graph would make such overlap explicit and manageable.

Achieving this vision relies on two key technology sets: **semantic web standards** and **LLMs (AI) for content processing**. Semantic web standards (like RDF, OWL, JSON-LD) provide the framework to represent and store the knowledge graph in a structured way that others can use. But manually converting OWASP's extensive text into a semantic format would be a monumental task – this is where large language models come in, automating the extraction of structured knowledge. In the next section, we'll explore how LLMs and semantic technology together can turn the current OWASP documentation into a rich knowledge graph with far less effort than one might imagine.

Leveraging Semantic Web Technologies and LLMs

The concept of a knowledge graph for OWASP isn't entirely new – it aligns with Tim Berners-Lee's vision of a Semantic Web where data is structured for machines – but what's new is the feasibility. Traditionally, creating semantic data from free text required extensive manual curation or brittle natural language processing scripts. Today, large language models can do much of the heavy lifting in extracting, normalizing, and linking information from text into a structured form.

Semantic Web Building Blocks: The foundation of a knowledge graph is an ontology or schema – a formal way to describe types of entities and relationships. For OWASP content, we would define, for example, that there are entities like *Vulnerability*, *Requirement*, *Control*, *BestPractice*, *Threat*, etc., and relationships like *mitigates*, *is example of*, *related to*, *has severity*, *maps to CWE*, and so on. We might say an **OWASP Top 10 risk** is a type of Vulnerability (category) that *has* a description, *has example CWEs*, and *is mitigated by* certain ASVS **Requirements**. An ASVS **Requirement** could be defined to *verify* certain **Controls** or techniques, and *address* certain **Vulnerabilities**. By defining these relationships, we essentially create a web that a machine can navigate just as naturally as a person follows links in a document.

LLM-Assisted Extraction: Large language models can be prompted to read OWASP text and output data according to the defined schema. For example, we could feed the text of the OWASP Top 10 page for “Broken Access Control” into an LLM with instructions to produce a JSON object containing fields like “Risk Name”, “Description”, “CWE Mapping”, “Related ASVS Requirements”, “Related Testing Guide Sections”, etc. Modern LLMs are capable of producing this kind of structured output when guided properly. In fact, an approach used by Dinis Cruz and team (which we’ll detail soon) employs OpenAI’s ability to generate **structured JSON outputs** by providing a schema – meaning the LLM is constrained to fill in specific slots with the relevant info (Building Semantic Knowledge Graphs with LLMs: Inside MyFeeds.ai's Multi-Phase Architecture). This reduces hallucination and yields a deterministic result. We might get a JSON like:

```
{
  "name": "Broken Access Control",
  "owasp_top10_year": 2021,
  "description": "...",
  "cwe_refs": ["CWE-284", "CWE-285"],
  "asvs_controls": ["V4.1.2", "V4.2.1"],
  "testing_guide_sections": ["4.6.1", "4.6.2"],
  "related_topics": ["Access Control", "Authorization", "Privilege Escalation"]
}
```

This is just an illustrative snippet, but it shows how the rich text of OWASP documents can be *transformed into data*. The same can be done for each ASVS requirement (e.g., parse it into a structured record including its verification level, category, etc., plus linking it to any risks like Top 10 categories or CWE it addresses). Testing Guide chapters can be processed to link each test scenario to the relevant vulnerability or OWASP category. Crucially, the LLM doesn’t have to invent these links out of thin air – many are implicitly or explicitly mentioned in the texts (for instance, Top 10 entries list example CWEs, ASVS often cites if a requirement maps to a particular issue). The LLM can surface these and put them into the JSON under the right fields.

Once the LLM extracts data for all documents, we feed those JSON objects into a graph database that supports semantic queries. An open source tool particularly suited for this is **MGraph-DB**, a serverless graph database recently published by the OWASP community. MGraph-DB was designed to easily convert JSON outputs from LLMs into graph nodes and edges. In other words, it takes those structured pieces and merges them into the growing knowledge graph. Each JSON object becomes part of the graph, and where identifiers match (say two JSON entries both refer to "CWE-284"), they can be merged or linked as the same node. MGraph-DB's *memory-first* and serverless design means it can scale the integration of lots of pieces quickly, which is ideal for building a large graph from many text sources.

Ensuring Quality and Consistency: One might worry about errors – can we trust the LLM to get all this right? The key is an iterative, *human-in-the-loop* process. The LLM provides a first draft of the graph, which can then be reviewed and refined by OWASP project experts. Importantly, because the LLM output is structured and not just prose, reviewing it is easier: you can spot if a link is missing or incorrect (e.g., an ASVS requirement was not linked to a Top 10 category it obviously relates to) and add it. Over time, the community can build a refined ontology and feed that back into the LLM prompt to guide extractions more strictly. In fact, Dinis Cruz notes that initially they let the LLM pick relationships freely, but the goal is to progressively enforce a stronger predefined taxonomy as human feedback is incorporated. In OWASP's case, this means we might start with simple relationships that the LLM can infer, and later formalize an OWASP ontology so the LLM knows exactly which relations to use.

Semantic Enrichment: The knowledge graph can be enriched with external data as well. OWASP content often references external standards like CWE, CAPEC, CVE, PCI DSS, etc. These can be nodes in the graph too, effectively connecting OWASP to the larger security knowledge ecosystem. This opens possibilities like: given a CVE (vulnerability in the wild), one could trace which CWE it's classified under and thus which OWASP Top 10 category and ASVS controls are relevant – a powerful cross-reference for defenders. All of this is facilitated by adopting semantic web-friendly identifiers and formats (for instance, using CWE IDs as unique keys in the graph that align with MITRE's data). The end result is that OWASP's advice doesn't live in a vacuum but is part of a **linked network of security knowledge**.

In summary, by combining semantic web technologies (to model and store knowledge) with LLMs (to extract and populate that knowledge at scale), OWASP can transform its corpus into a living, queryable knowledge graph. This graph would dramatically improve accessibility (both for humans and machines), enable customization for various needs, and tightly interlink all pieces of guidance for a more holistic understanding. We have a compelling proof-of-concept of this approach in action, which we'll look at next: the work done by Dinis Cruz in projects like MyFeeds.ai and The Cyber Boardroom.

Inspiration from MyFeeds.ai and The Cyber Boardroom

To ground this in reality, consider the approach pioneered by OWASP veteran Dinis Cruz with two recent projects: **MyFeeds.ai** and **The Cyber Boardroom**. These projects deal with a different domain of content (news articles and executive reports), but they showcase the power of using LLMs to build semantic knowledge graphs and generate personalized outputs – exactly the kind of workflow we envision for OWASP.

MyFeeds.ai is a system that creates personalized cybersecurity news feeds. Under the hood, it uses a multi-phase LLM pipeline to turn unstructured news articles into a structured knowledge graph and then produce a tailored summary for a specific persona (like a CEO, CISO, developer, etc.). The process works in four main phases:

1. **Entity and Relationship Extraction (Article Graph)** – MyFeeds takes an incoming article (via RSS feed), and calls an LLM to extract key entities and facts from that article. For instance, if the article is about a new data breach, the LLM might output a structured summary: *entities* such as "Company X", "Ransomware", "Data Breach", and relationships like "Company X suffered Data Breach via Ransomware". This comes back not as free text but as a JSON object with defined fields (using OpenAI's structured output mechanism). Essentially, the article is converted into a mini knowledge graph of its own content. These JSON results are then loaded into the graph database (MGraph-DB) as nodes and edges, creating what we can call an "article graph". At this stage, each article is a separate set of nodes.
2. **Persona Graph Construction** – In parallel, MyFeeds constructs a "persona" graph that represents the interests and context of the target reader. For example, if the persona is a CEO concerned with business impact, the persona graph might include nodes like "Financial Loss", "Reputation Impact", "Regulatory Compliance", etc., indicating topics the CEO cares about. This can be created by an LLM as well, given a prompt describing the persona's concerns. In Dinis's MVP, personas were somewhat predefined (hard-coded profiles for CEO, CISO, etc.), but they are represented as data just like the articles. Now we have two graphs: one for the content of each article, and one for the persona.
3. **Relevance Mapping** – The next challenge: determine which articles are relevant to the persona. MyFeeds does this by essentially "connecting" the article graph to the persona graph via another LLM step. The LLM is given the persona's entity list and an article's entity list and asked to find matches or map relationships between them. The output is a set of mappings with scores – e.g., it might find that an article about "Ransomware causing data breach" is highly relevant to a persona concerned with "Operational Disruption" and "Financial Loss", yielding a relevance score. This step results in an enhanced article node that now links to persona interests and carries a relevance rating. Notably, by doing this with structured data (and not asking the LLM to just summarize text arbitrarily), the process remains pretty deterministic and avoids hallucination; the LLM is working only with the facts extracted from the article and persona definition. At the end of Phase 3, the system has identified, say, 5 articles out of

50 that are most relevant to the CEO persona, each annotated with which key points align with the CEO's concerns. All the while, provenance is preserved – the graph nodes still link back to the original article content, which is even included in the data for transparency.

4. **Personalized Summary Generation** – Finally, MyFeeds uses an LLM one more time to generate a synthesized output for the persona, using the curated relevant articles and their linked facts. Essentially, it asks the LLM to "write a blog post/newsletter for [Persona] summarizing the insights from these articles" and provides the structured data of those articles as context. Because the input includes the key facts and even the original text snippets, the LLM can produce a coherent summary with direct references. The result is a personalized report – for example, a CEO's weekly cyber brief that highlights just the business-impacting security news. Crucially, this final LLM output is also captured as structured data (it returns JSON objects for each section of the summary) rather than just free-form text. This allows MyFeeds to easily format it for various outputs (HTML, Markdown, etc.) and even publish it automatically. In Dinis's demo, the output was published to a Ghost blog site via API, generating a nice web page for the persona. The key takeaway: *the LLM was used to transform selected articles and their connected entities into a personalized blog post*, effectively writing a custom newsletter based on the underlying knowledge graph.

This multi-phase approach demonstrates a pattern that could be mirrored for OWASP content: - Instead of news articles, we have OWASP documents (Top 10 chapters, ASVS requirements, etc.) to extract entities/relationships from. - Instead of a CEO persona, we might have a "context profile" (e.g., a particular industry or technology focus) to construct. - We then map which parts of OWASP knowledge are most relevant to that context. - Finally, we could generate a tailored guidance document (or interactive FAQ, or checklist) for that context.

The **Cyber Boardroom** complements this by providing a platform for user-specific feeds. If MyFeeds is the engine that produces these persona-based reports from public data, the Cyber Boardroom is a private environment where an individual user can have multiple personas, subscribe to feeds, and interact with them. Dinis integrated features like user authentication and multi-persona support there, recognizing that one person (say a CISO) might want different profiles (one for technical detail, one for board presentations, etc.). The Cyber Boardroom essentially aims to commercialize or operationalize the personalized knowledge delivery – something OWASP could emulate in spirit by offering customizable "views" of its content to community members.

There are a few important lessons from MyFeeds.ai for our OWASP use case: - **LLMs as Connectors, not Replacements**: The system didn't just ask the AI open-ended questions; it broke the problem into stages and used structured prompts and outputs at each stage. This aligns with a principle Dinis highlighted: *use LLMs only for what they are the only option for, and use code for everything else*. In building an OWASP knowledge graph, we would do similarly – use LLMs to bridge the gap where writing parsing code is too complex (like understanding natural language explanations), but use deterministic algorithms to assemble, query, and present the data whenever possible. This yields more repeatable and explainable results.

- **Provenance and Determinism**: By keeping the intermediate data (graphs of article content, etc.), MyFeeds ensures that the final output can be traced back to sources. OWASP content would benefit from the same approach – e.g., if a custom report says "Implement control X to prevent Y," the graph could show that this sentence came from ASVS requirement 4.2.3 and Top 10 2021 A04, linking directly to those sources. Provenance builds trust in the information and is crucial in a community-driven content ecosystem to avoid misinterpretation. The LLM-driven extraction method, especially with structured outputs, helps maintain determinism (same input yields same output) which is important for standards and guidelines that need consistency.
- **Open Source Implementation**: Perhaps most encouraging, the core tooling for MyFeeds.ai is open source. Dinis released all the code powering this multi-phase pipeline on GitHub. This includes the MGraph-DB graph database, the prompt schemas, and automation scripts (many of which reside in the OWASP SBot repositories). This means OWASP can leverage the *exact same tools and techniques* that MyFeeds used, without starting from scratch or investing heavily in proprietary tech. In fact, the OWASP SBot (Security Bot) project, which predates MyFeeds, already set up a lot of the serverless automation framework (AWS Lambda functions, integration with Jira/Slack, etc.) that can support such workflows (Using OWASP Security Bot (OSBot) to make Fact Based Security Decisions | PPT). OSBot was used to create graphs from Jira data and link information for security decision making, showing that this concept of graph-based context has been brewing in OWASP circles for a while. We can build on that foundation for our documentation transformation.

In summary, the success of MyFeeds.ai and The Cyber Boardroom in extracting and delivering tailored knowledge from unstructured content underlines the viability of a semantic, LLM-assisted approach. If we apply those principles to OWASP's content, we can envision a system where OWASP knowledge is not just a set of static documents, but a service: an intelligent, queryable, customizable knowledge base accessible to developers, testers, managers, and automated tools alike.

A Semantic Approach for OWASP: From Vision to Reality

How specifically might this work for OWASP's own projects? Let's walk through a plausible implementation scenario, tying together all the concepts discussed:

1. **Define the OWASP Ontology** – Start by outlining what entity types and relations we need to represent OWASP content. Likely entities: *Risk Category* (e.g., Top 10 entries, vulnerability categories), *Requirement* (ASVS items, MASVS for mobile, etc.), *Test Case* (from Testing Guide), *Best Practice* (from Cheat Sheets or Proactive Controls), *Tool/Project* (like references to OWASP tools or external tools), etc. Define relations like *"addresses"* / *"mitigates"* (a requirement mitigates a risk, a test addresses a requirement), *"has weakness"* (a risk category links to CWE weakness types), *"related to"* (for more general associations), *"prerequisite for"* or *"part of"* (hierarchical relations, e.g., ASVS Chapter contains Requirements). This ontology would be developed collaboratively with OWASP project leaders to ensure it makes sense for all content. The good news: much of this structure is intuitive and already informally present in the docs (e.g., ASVS has chapter-section numbers, Top 10 categories have CWE mappings).

2. **Convert Text to Structured Data with LLMs** – Using the ontology as a guide, process each OWASP document:

   - *OWASP Top 10:* For each Top 10 item, parse out its title, description, example attack scenarios, example CWEs, and any references to other OWASP projects. The LLM might read the narrative and output something like *Risk = "Injection"*, *Impact = "data loss, account takeover"*, *CWE_List = [79, 89, ...]*, *Related_ASVS = ["V5.3.2", "V5.3.3"]* if it can identify those in text or from its knowledge. (We may need to prompt it with a hint like "map this to ASVS if possible"; alternatively, we can do a post-processing step to link by CWE overlap – multiple strategies exist.)
   - *OWASP ASVS:* Convert each requirement into an object with fields: e.g., *ID = 4.2.3*, *Description = "...ensure X..."*, *Level = 2*, *Category = "Authentication"*, *Related_Top10 = [A2-2017]* if it maps (some ASVS items explicitly say "(Mapping: OWASP Top10 A2)" which the LLM can capture). ASVS also has an official JSON in its repo which we can ingest directly as a starting point, to avoid re-extracting what's already structured.
   - *OWASP Testing Guide:* Each section (e.g., test for SQL injection) can be summarized into fields: *Test Name*, *Description of approach*, *Related Vulnerability/Risk*, *Tools* (if mentioned), etc. The Testing Guide might be the most textual, but it usually is organized by vulnerability type, so the LLM will likely latch onto those headings.
   - *Cheat Sheets and Other Guides:* These can be treated similarly – identify the key recommendations and link them to the vulnerabilities or topics they address.

   At this stage we would have a large collection of JSON objects or triples that the LLM produced. We merge these into the graph database. Since multiple sources might refer to the same thing (e.g., "Injection" appears in Top10 and Testing Guide), part of the merging involves reconciling them. This could be done by common identifiers: e.g., if the Top 10 item had CWE-89 and a Testing Guide section also mentioned CWE-89, we can link them via a common CWE node. Or simply by name matching with verification by the ontology (the ontology might say that if two Risk nodes have the same name “Injection”, and maybe same CWE set, they are the same concept node). This reconciliation step might require some manual review to ensure correctness, but tools can assist (for example, we can automatically flag potential merges).

3. **Augment with Context Tags** – Once the core graph is built, we can enrich nodes with metadata that will enable the personalized and context-aware features:
   - Tag certain vulnerabilities or controls with industry sectors if applicable (maybe gleaned from OWASP Vertical Knowledge Projects if any exist, or from experience – e.g., “PCI” or “Healthcare” tags).
   - Tag requirements with “High”, “Medium” if they are critical in certain threat models (ASVS has levels which serve this purpose to some degree, but we could extend).
   - Mark content with regional relevance if any (for example, if a particular guideline addresses GDPR explicitly, tag it “EU”). Few OWASP items are region-specific, but data privacy ones might map to regions.
   - Add multilingual labels: use LLMs or existing translations to add, say, a Spanish description field on each node. Because the graph is language-agnostic in structure, you can attach multiple language versions of text to the same node. This would allow dynamic switching of language in output. OWASP already has translations (like ASVS in several languages; those can be aligned to the same nodes in the graph rather than being separate documents.
4. **Enable Queries and Views** – With the populated knowledge graph in place, we build interfaces to use it:
   - A simple **Graph Query API** (could be GraphQL or SPARQL or even REST endpoints) that developers and tools can query. For example, an API endpoint `/owasp/risk/Injection` could return a JSON with all linked info about Injection (related requirements, test cases, etc.). Or a query `GET /owasp/asvs?vuln=Injection` returns all ASVS requirements related to Injection.
   - An **Interactive Explorer** for humans: a web UI where you start at a node (say “Broken Access Control”) and visually or navigationally explore links (see related ASVS items, click to see their details, then to testing procedures, etc.). This could be similar to how some documentation sites have link maps, but powered by the underlying graph for completeness.
   - A **Customization Wizard**: a tool where a user can input their context (industry, tech, threat concerns) and it generates a tailored OWASP guide. This is where we bring in the persona concept. We could have preset personas like “Web App Developer”, “Cloud SaaS Provider”, “Mobile App Pen tester”, etc., each basically a filter or subgraph focusing on certain areas. The user picks one (or configures one by selecting interests),

and the system produces a custom view – maybe a PDF or web page that collates the relevant OWASP content. LLM can be used here to stitch together narrative if needed (e.g., writing an introduction or summary for the custom guide), but all the substance comes from the graph data (ensuring accuracy).

- **Multilingual support** toggles on the site: because nodes have multiple language text, the user could switch language and see the content in, say, Portuguese. If a piece isn't translated yet, an LLM could translate it on the fly, and optionally that could be reviewed and added to the graph for next time.

5. **Integration with AI Assistants** – By having OWASP's knowledge in a structured database, even external AI systems (like GPT-based chatbots) can more reliably use it. For instance, one could build an OWASP ChatGPT plugin that queries the OWASP knowledge graph when asked security questions, rather than relying on the model's trained memory of OWASP (which might be outdated or incomplete). The plugin could fetch precise answers and even cite OWASP content properly (thanks to the provenance in the graph). This means when a developer asks "What should I do to prevent XSS in an Angular app?", the assistant could retrieve the exact relevant nodes (perhaps OWASP Top 10 XSS description + Angular Cheat Sheet guidance) and respond with a tailored, up-to-date answer. Essentially, the OWASP knowledge graph would make OWASP content **AI-ready** in a responsible way.
6. **Maintenance Workflow** – As OWASP projects evolve, the knowledge graph can be updated. The ideal maintenance is to bake the structured approach into the project workflows:
   - When the next OWASP Top 10 is drafted, the project could simultaneously prepare the structured data (maybe even writing content in a way that is easily pluggable into the graph – like maintaining a YAML file for each entry alongside the narrative).
   - ASVS already has a structured source; ensuring that stays synced or directly using it means minimal extra work.
   - Encourage project leads to think in terms of **data first**: e.g., the next Testing Guide update could catalog each test case in a spreadsheet or YAML which then can feed the graph, instead of writing a monolithic doc.
   - The community can also help to fill gaps. Perhaps set up a small OWASP task force to continuously refine the graph – adding missing links, updating tags, incorporating new project outputs (OWASP has many projects – e.g., OWASP API Top 10, OWASP MASVS for mobile – those too can feed into this ecosystem).
   - Because the knowledge graph and underlying data formats would be open, contributors can make pull requests to them just like they do to Markdown or code today.

By implementing this, OWASP would significantly increase the utility of its content. The knowledge would no longer be locked in silos or only human-readable paragraphs; it would be an **open platform** that others can interact with. Consider the downstream benefits:

- A training company could use the OWASP knowledge graph to build up-to-date courseware or exam questions (by querying for definitions or links between concepts).
- A compliance management tool could automatically check an organization's policies against OWASP recommendations by mapping their controls to the graph.
- OWASP content could be more easily kept multilingual; when updates occur, translators see exactly which small nodes changed rather than a whole chapter, making their job easier and version tracking clearer.
- New contributors to OWASP might find it easier to contribute to a data set than to a long document – for instance, someone might contribute by adding a link in the graph between a particular ASVS item and a relevant Cheat Sheet, without needing to edit the prose of either document.

Open Source Tools and Workflow Alignment

One of the strengths of this proposal is that **we can leverage existing open source tools and past OWASP efforts** to jump-start the semantic knowledge graph ecosystem:

- **MGraph-DB**: As mentioned, MGraph-DB is an open source, serverless graph database tailored for integrating with LLM outputs. It was created by Dinis Cruz's team to support the MyFeeds project and released under an open license. This means OWASP has at its disposal a purpose-built engine to store and manipulate the kind of data we're talking about – nodes for vulnerabilities, edges for relationships, all manageable via Python scripts or APIs. MGraph-DB being serverless implies it can run cheaply in the cloud (e.g., on AWS Lambda or similar) which lowers the barrier to hosting an OWASP knowledge graph service. Since it's open source, it can be extended or adapted to OWASP's specific ontology as needed. The project's documentation and code are available in the OWASP SBot GitHub repositories (within `owasp-sbot/MGraph-DB`), and it even has a published v1.0 release targeting GenAI and semantic web use cases.
- **OWASP SBot (Security Bot)**: OWASP SBot is a collection of automation scripts and integrations developed a few years ago to help with security workflows. It's also completely open source (with dozens of repos under the `owasp-sbot` GitHub org). Notably, OSBot already dealt with

representing security information in graph form and integrating with external systems. For example, OSBot-Jira and OSBot-Elastic allowed pulling data from Jira issues or Elastic indices and then graphing them to see relationships between vulnerabilities, risks, and policies. One of the SlideShare presentations on OSBot highlights its ability to *"generate graphs of data from tools like Jira to help understand security issues and their relationships"*. The same tooling can be repurposed to handle OWASP's own knowledge as data. OSBot also provided means to interact via Slack (chatbot style queries) and to run on AWS infrastructure, which could be useful if we want to enable community members to query the knowledge graph in natural language or set up automated alerts (imagine an OSBot slack command like `!owaspquery "list all ASVS controls for SQL injection"` returning results from the graph).

- **OpenAI/LLM Integration Code**: The prompting techniques, schemas, and multi-step workflow that MyFeeds uses are documented and scriptable. Dinis has shared system prompts and even Python classes that define the structured outputs. We can use these as starting points for writing our own LLM prompts to extract OWASP data. Since those are open, we save a lot of experimentation time. For instance, we know we can use *OpenAI's function calling* or *structured output* feature to get JSON back, and we have examples of how classes were defined for articles and personas which we can mimic for vulnerabilities and requirements. The whole workflow could potentially be orchestrated with existing OSBot AWS Lambda functions or new ones added to that framework.
- **Existing OWASP Data**: Let's not forget that some OWASP projects already produce data: ASVS as JSON/CSV, the OWASP Top 10 has a GitHub where content is in Markdown (which is parseable), and Cheat Sheets are on GitHub in markdown (could be parsed). We should use these as much as possible to feed the initial graph. Every bit of structured data we have reduces the reliance on LLM extraction and improves accuracy (for example, directly ingesting the ASVS JSON ensures those requirements are 100% correct structurally). The LLM can then focus on linking and contextualizing rather than basic parsing.
- **Community Knowledge and Previous Research**: OWASP can also benefit from academic and industry research in applying knowledge graphs to security. There have been studies (some by OWASP volunteers or chapters) on using knowledge graphs for threat modeling or mapping vulnerabilities. For example, one 2020 research paper constructed knowledge graphs for OWASP Mobile Top 10 threats to analyze Android apps ((PDF) Mobile Software Assurance Informed through Knowledge Graph Construction: The OWASP Threat of Insecure Data Storage). Such efforts, while separate, could inform how we model certain security concepts. Since our approach and data will be open, we can potentially collaborate or draw from those who have attempted similar things.

In essence, OWASP would be standing on the shoulders of prior work – both from within the OWASP community and the broader open source/academic community – to implement this semantic ecosystem. This lowers cost and risk. The focus shifts from “can it be built?” (because we have evidence it can) to “how do we organize and execute it within OWASP?”.

Recommendations for Adopting the Semantic Model

Adopting a semantic knowledge graph ecosystem for OWASP content is a significant transformation. It should be approached incrementally, demonstrating value at each step to gain buy-in from the community. Here are specific recommendations to get started and to ensure long-term success:

1. **Launch a Pilot Project:** Identify one or two OWASP documentation projects to pilot the semantic approach. A good candidate is the OWASP Top 10, due to its popularity and relatively small size (10 items). For example, create a **Semantic OWASP Top 10** pilot: use an LLM to parse the Top 10 2021 report into a structured graph (including each risk’s description, CWEs, etc.), and link it to a subset of ASVS requirements and Testing Guide sections. Develop a simple web interface or report from this graph to show how it enhances navigation (e.g., click a Top 10 risk to see related ASVS and Testing Guide info). This pilot will help iron out the ontology and extraction process. Another potential pilot could be with ASVS itself, since ASVS already has structured data – here the focus would be on linking ASVS to other docs and adding context tags. Keep the pilot scope manageable, and show it to OWASP leaders and community for feedback.
2. **Form an OWASP Knowledge Graph Working Group:** Assemble interested OWASP leaders, project owners, and volunteers into a working group dedicated to this initiative. This group would steer the ontology design (ensuring it meets all project needs), decide on tools, and coordinate between different documentation projects. Having representation from Top 10, ASVS, Testing Guide, Cheat Sheets, etc., will be important so that the resulting system serves everyone. The working group can also enlist help from semantic web or AI experts in the community (perhaps OWASP members with those skills or even students/academics who want to contribute).
3. **Develop the OWASP Ontology and Schema Collaboratively:** As a deliverable of the working group, create a first version of the OWASP ontology/schema. This might be a simple document or data model that lists entity types and relationships. It should be reviewed by the project owners to ensure it captures the essence of their content. Start simple – you can always extend later. For example, define “Risk Category” (with properties: name, description, severity, CWE list) and “Security Requirement” (properties: id, text, level, etc.) and a relation “mitigates”. You might

not initially model every nuance (like you might skip “Threat Agent” or other fields from Top 10 if not immediately needed). The key is to have an agreed structure so that different people or tools extracting data will output it consistently.

4. **Create LLM Extraction Playbooks:** Document the process for using LLMs to convert an OWASP document into structured form. This includes the prompt design and any post-processing. For instance, write a prompt template for Top 10 items like: *“Extract the following fields from this OWASP Top 10 entry: [Name, Description, CWE_IDs, Primary_Mitigations, References]. Use JSON format.”* Test this with GPT-4 or similar on one example and refine until it’s giving good output. Then use a script (possibly with OpenAI API) to batch process all items. Do similar for one chapter of the Testing Guide to see how it handles it. The playbook should also cover verification steps – e.g., “After extraction, have someone check that each CWE ID is correct and each reference link is correct.” This semi-automated approach will likely be the way to populate the initial graph.

5. **Integrate with Existing Data Sources:** Where structured data exists (ASVS JSON, project CSVs, etc.), write importers to feed those directly into the graph. This ensures accuracy and saves LLM credits for parts that truly need language understanding. Also import known mappings, if any (for example, if OWASP Top 10 already lists “Mapping to ASVS: Vn.n” in its appendix, use that).

6. **Develop a Minimal Useful Application:** To demonstrate the utility, build a simple application on top of the emerging knowledge graph. This could be as simple as a **“OWASP Graph Search”** web page where you select an OWASP Top 10 risk from a dropdown and it shows a list of related ASVS requirements and Testing Guide sections (sourced from the graph). Or a chatbot interface where you can ask, “How do I test for XSS?” and it answers citing the Testing Guide and ASVS. By showcasing even a basic query capability that spans multiple OWASP docs, you will gather support from those who instantly see the time-saving benefit. It doesn’t need to be fancy initially; even a command-line query tool could impress the right folks if it outputs something that previously took an hour of manual searching.

7. **Ensure Open, Machine-Consumable Formats Are Published:** As part of this initiative, make it a practice that new versions of OWASP documents come with a machine-readable version. For example, when OWASP Top 10 gets updated, release a JSON file with the content (even if the graph isn’t fully integrated yet). Encourage project leads to use simple formats like YAML/JSON or Markdown with structured fields for any lists or enumerations. The working group can provide templates or tools to help with this. The earlier GitHub issue asking for a JSON version of Top 10 shows community desire; addressing that quickly (even as a static JSON file) is a win on its own.

8. **Community Engagement and Outreach:** Communicate progress and involve the wider OWASP community. For instance, present the concept and pilot results in an OWASP Leaders meeting or Global AppSec conference talk. Show a live demo of querying the OWASP knowledge graph. This will excite contributors and maybe attract more volunteers (or even sponsors). You might start an OWASP project formally for the knowledge graph toolset, so it has visibility and an official home. Documentation, of course, should be written to guide users and contributors on how to

access and update the graph. By making this an open project, even people outside of OWASP could contribute mappings or improvements (because many security practitioners have interest in such knowledge bases).

9. **Incremental Expansion and Iteration:** After a successful pilot, iterate by incorporating more projects. Maybe next bring in the OWASP API Security Top 10, or the Mobile Security Testing Guide, etc. Each addition will likely introduce new entity types or relations (for example, the Mobile projects might add a concept of “Mobile Platform” if they categorize iOS vs Android issues). Grow the ontology and graph accordingly, but maintain coherence. Continuously gather feedback from end-users on what features are most useful. Perhaps the community really wants a feature to compare two OWASP standards side by side – the graph could enable that easily, so build a view for it. Keep iterating with short cycles, adding content and features, rather than disappearing into a year-long development – this ensures the project stays relevant and volunteer momentum remains.
10. **Governance and Upkeep:** Long term, decide how this semantic knowledge graph will be maintained. It could be integrated into the workflow of each documentation project (so that when they update the docs, they also update the graph data). Or it could be managed by a central team that monitors all OWASP content updates and updates the graph accordingly. There should be clear ownership to avoid it becoming stale. Perhaps eventually, the knowledge graph becomes *the source* and the human-readable docs are generated from it – that would be ideal, but it’s a big cultural shift, so it may be a future phase when trust in the system is established. For now, treating the graph as a parallel publication that is updated alongside the main content may work. Also, plan for versioning (e.g., OWASP Top 10 2021 vs 2024 in the graph – likely handled by tagging nodes with version info).

By following these recommendations, OWASP can gradually adopt the semantic model without disrupting ongoing work. The aim is to **demonstrate value early** (especially to the project leaders whose buy-in is crucial) and to align with OWASP’s ethos of openness and community collaboration. This effort could very well become an OWASP flagship project on its own, showcasing innovation in how security knowledge is shared.

Conclusion

The OWASP community has always been about empowering organizations and developers with knowledge to build secure software. As we move deeper into the digital era, the way knowledge is consumed is changing – it’s more interactive, on-demand, and tailored. To keep pace, OWASP’s own knowledge delivery should evolve from static documents to a **dynamic, semantic knowledge graph ecosystem**. By doing so, OWASP content becomes more accessible (to both humans and machines), more customizable to diverse needs, and more interconnected for a holistic understanding of application security.

In this paper, we highlighted the current pain points: important OWASP guidelines locked in PDFs and web pages that can't easily talk to each other or adapt to specific contexts. We then painted a vision of how semantic web technologies and LLMs can transform this situation, turning OWASP's collective wisdom into a living knowledge graph that is rich with relationships and machine-readable meaning. We drew inspiration from Dinis Cruz's MyFeeds.ai and Cyber Boardroom projects, which proved that even complex narratives can be distilled into graphs and reassembled into personalized outputs using AI – a blueprint OWASP can follow using open source tools like MGraph-DB and OWASP SBot that are already at our disposal.

The benefits of this shift are immense. Consider a future where a developer in Brazil can query OWASP in Portuguese for the top risks in her fintech web application, and instantly get a curated list with implementation guidance and test checklists drawn from OWASP Top 10, ASVS, and Cheat Sheets – all relevant, up-to-date, and in her language. Or a future where an OWASP Top 10 release isn't just a PDF, but an interactive data set that companies can import directly into their risk tracking systems, and a knowledge base that AI assistants reference to give developers accurate advice. In such a future, OWASP's impact would be even greater than it is today, because the knowledge would integrate seamlessly into the workflows and tools developers use.

Importantly, moving to a semantic knowledge graph model keeps OWASP on the cutting edge of how information is shared, ensuring the organization remains *the* go-to source for application security guidance in a form that modern platforms (and brains) can readily digest. It also reinforces OWASP's open philosophy: an open knowledge graph is easier for the community to contribute to, remix, and build upon than static docs. The underlying data being open means researchers, educators, and tool builders can create new innovations on top of OWASP content, driving further our mission of spreading security knowledge.

In conclusion, the call to action is clear – by embracing semantic technology and AI-assisted curation, OWASP can overcome current limitations and provide a more powerful, flexible knowledge ecosystem for the next generation of application security challenges. Much of the groundwork has been laid by open source efforts; now it's up to us in the OWASP leadership and community to take the next step. Let's pilot this approach, refine it, and gradually roll out an OWASP knowledge graph that will make our projects not just documents to read, but *data to query, link, and apply*. The result will be an OWASP that not only publishes guidelines, but actively *connects* the dots of global application security wisdom – driving smarter, faster, and more context-aware security decisions everywhere.