What's New in ASVS v5

Eden Yardeni - NDC Security 2026

Agenda

1. Intros (of ASVS and of myself)
2. 3 Key Refactors
3. A Curated Selection of New Requirements
4. Demo: Code Review
5. Conclusion / Q&A

Introductions

OF ASVS AND OF MYSELF

Eden Yardeni

- Worked as a full-time software engineer for 12 years
  - 5 of those on AppSec teams, one of which I founded
- Joined the OWASP ASVS working group in October 2024
- Also started the OWASP chapter for Cleveland, Ohio in 2024
  - But then moved to London shortly thereafter (the Cleveland chapter is in great hands though)
- I've been **using** the ASVS since 2017—we will get to that!

The Application Security Verification Standard

- A curated set of **positive** requirements that govern **properties** of web application and API code
- Can be used as:
  - a checklist for secure coding, including writing the code and code reviews
  - a reference for penetration testing
  - secure architecture guidance
  - the basis for unit / integration tests
  - topics for developer training
  - a baseline for procurement decisions

The Application Security Verification Standard

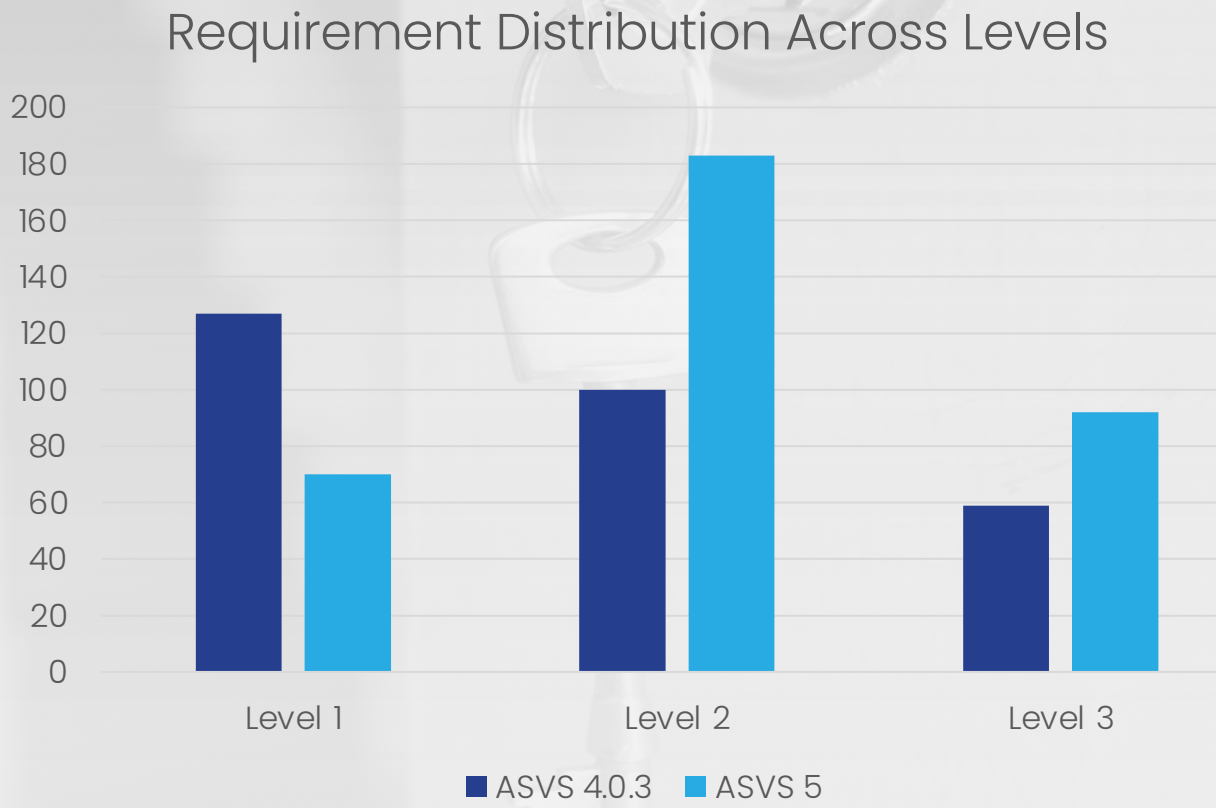
- One of OWASP's flagship projects
- Developed in the open by contributors like you and me <3
- Designed to be:
  - Actionable
  - Forkable
  - Flexible

Three Key Refactors

STRUCTURAL CHANGES

Refactor #1 – Easier Level Structure

- Rebalanced levels to make getting started easier
- Levels are now relative / risk based and no longer represent “minimum”/ “standard” / “advanced” baselines
- Recognizing that pentests should cover code too

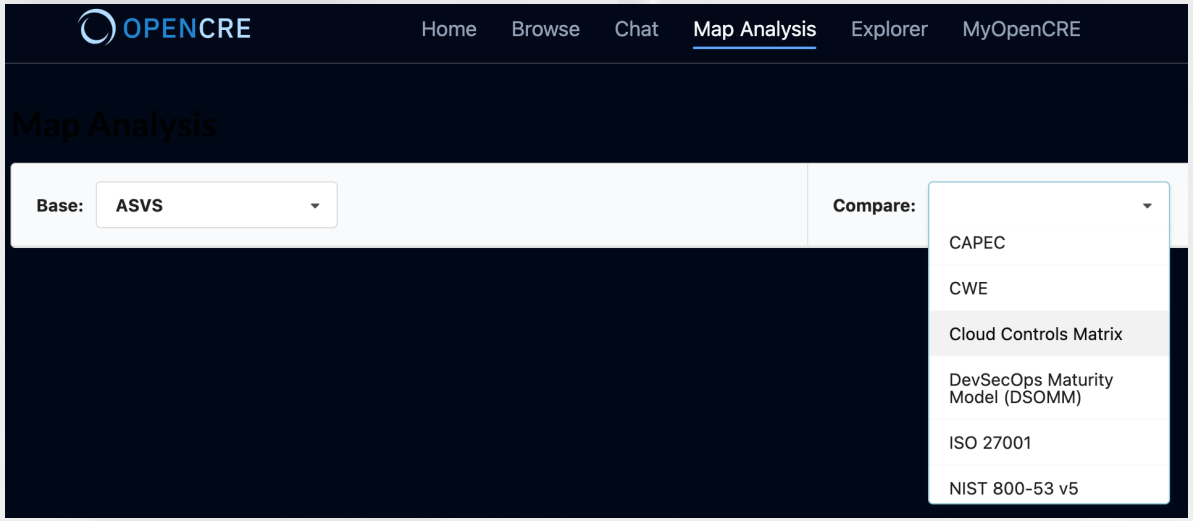


Refactor #2 – Properties over Process

- Architectural requirements were dropped to focus on properties of the **application**
- Is an application specifically less **secure** without the requirement being satisfied?
- Refocusing on **verifiable** goals that we believe are musts vs. nice-to-haves
- Written like an enforceable **standard**

Refactor #3 – Clean Mappings

- Dropped direct mappings to CWE, NIST in favor of OpenCRE
- CWE as an abstract taxonomy did not always have clean mappings
- NIST is US-specific and mapping to non-US standards is becoming increasingly common
- Ongoing work to create OpenCRE mappings
- Did I mention OpenCRE yet? Go check that out...



OWASP®

ASVS
APPLICATION SECURITY
VERIFICATION STANDARD

New chapters in 5.0

A CURATED SELECTION, BECAUSE THERE ARE A LOT

Encoding and Sanitization (V1)

- V1.2.8: LaTeX Processors (Lv. 2)
- V1.2.10: CSV and Formula Injection (Lv. 3)
- V1.3.8: JNDI query protection (Lv. 2)
- V1.3.9: Memcache injection (Lv. 2)
- V1.3.12: Runaway Regex / ReDoS mitigation (Lv. 3)
- V.1.4.3: Use-after-free protection (Lv. 2)

Web Frontend Security (V3)

- V3.2.3: DOM Clobbering (Lv. 3)
- V3.3.5: Cookie Length Limitations (Lv. 3)
- V3.5: Browser Origin Separation
- New header requirements including Cross-Origin-Opener-Policy, Cross-Origin-Resource-Policy, Sec-Fetch-*

Self-contained Tokens (V9)

- V9.1.2: Disallow ‘None’ algorithm for JWTs (Lv. 1)
- V9.2.1: Enforce expiration time for stateless tokens (Lv.1)

OWASP®

ASVS
APPLICATION SECURITY VERIFICATION STANDARD

Demoing big new sections in 5.0

APPLYING REQUIREMENTS TO CODE REVIEW

Updated Requirements – OAuth (formerly V3.5)

V3.5 Token-based Session Management

Token-based session management includes JWT, OAuth, SAML, and API keys. Of these, API keys are known to be weak and should not be used in new code.

#	Description	L1	L2	L3	CWE	NIST §
3.5.1	Verify the application allows users to revoke OAuth tokens that form trust relationships with linked applications.		✓	✓	290	7.1.2
3.5.2	Verify the application uses session tokens rather than static API secrets and keys, except with legacy implementations.		✓	✓	798	
3.5.3	Verify that stateless session tokens use digital signatures, encryption, and other countermeasures to protect against tampering, enveloping, replay, null cipher, and key substitution attacks.		✓	✓	345	

Updated Requirements – OAuth (now v10)

- V10.1: Generic OAuth and OIDC Security
- V10.2: OAuth Client
- V10.3: OAuth Resource Server
- V10.4: OAuth Authorization Server
- V10.5: OIDC Client
- V10.6: OpenID Provider
- V10.7: Consent Management

```java
@Bean
JwtDecoder jwtDecoder() {
    NimbusJwtDecoder decoder = NimbusJwtDecoder.withJwkSetUri("https://login.example.com/.well-
        known/jwks.json").build();

    JwtTimestampValidator time = new JwtTimestampValidator();
    time.setClockSkew(java.time.Duration.ofSeconds(60));

    OAuth2TokenValidator<Jwt> issuer = jwt -> {
    String iss = jwt.getIssuer() != null ? jwt.getIssuer().toString() : "";
    boolean ok = iss.equals("https://login.example.com/");
    return ok
        ? OAuth2TokenValidatorResult.success()
        : OAuth2TokenValidatorResult.failure(new OAuth2Error("invalid_token", "Untrusted issuer", null));
    };

    OAuth2TokenValidator<Jwt> alg = jwt -> {
    String algorithm = jwt.getHeaders().getAlgorithm() != null ? jwt.getHeaders().getAlgorithm().getName() : "";
    boolean ok = algorithm.equals("RS256");
    return ok
        ? OAuth2TokenValidatorResult.success()
        : OAuth2TokenValidatorResult.failure(new OAuth2Error("invalid_token", "Unexpected signing algorithm",
        null));
    };

    decoder.setJwtValidator(new DelegatingOAuth2TokenValidator<>(time, issuer, alg));
    return decoder;
}
```

V10.3: OAuth Resource Server

#	Description	Level
10.3.1	Verify that the resource server only accepts access tokens that are intended for use with that service (audience). The audience may be included in a structured access token (such as the 'aud'claim in JWT), or it can be checked using the token introspection endpoint.	2
10.3.2	Verify that the resource server enforces authorization decisions based on claims from the access token that define delegated authorization. If claims such as 'sub', 'scope', and 'authorization_details'are present, they must be part of the decision.	2
10.3.3	Verify that if an access control decision requires identifying a unique user from an access token (JWT or related token introspection response), the resource server identifies the user from claims that cannot be reassigned to other users. Typically, it means using a combination of 'iss'and 'sub'claims.	2

ASVS
APPLICATION SECURITY VERIFICATION STANDARD

```java
@Bean
JwtDecoder jwtDecoder() {


        ...
        ...
        ...

        OAuth2TokenValidator<Jwt> audience = jwt -> {
        List<String> aud = jwt.getAudience();
        boolean ok = aud != null && aud.contains("api://inventory");
        return ok
                ? OAuth2TokenValidatorResult.success()
                : OAuth2TokenValidatorResult.failure(new OAuth2Error("invalid_token", "Missing required
                audience", null));
        };

        decoder.setJwtValidator(new DelegatingOAuth2TokenValidator<>(time, issuer, alg));
        return decoder;
}
```

Thursday Room 3 15:00 - 16:00 (UTC+01)
Talk (60 min)

Secure and Compliant APIs - By Design

If you ask 10 developers for a code review, they will identify different issues, and many will miss security concerns like broken access control and lack of input validation.

Application Security | Programming | Security Tooling | Testing

How can a DevOps team in their daily work assert that new features do not introduce vulnerabilities, that security bugs get caught before deploy to production? What are the key questions to ask in a code review? And how can we show that the application code also aligns with requirements from compliance, well-known best practices and internal security policies?

Daniel Sandberg

Daniel Sandberg is a Senior Software Architect with 20 years of experience designing scalable, high-performance systems. He combines deep architectural expertise with a strong focus on building reliable and secure solutions that meet modern business needs.

Tobias Ahnoff

Tobias Ahnoff is an experienced developer and architect with focus on application security. He specializes in implementing authentication flows and authorization for web applications and APIs that manage sensitive data in the bank, finance, and health sectors. He performs security reviews and penetration tests as part of Omegapoint Cybersecurity, contributes to the OWASP ASVS working group and is a co-author of http://securityblog.omegapoint.se/

Updated Requirements – Race conditions (formerly V11.1.6)

11.1.6	Verify that the application does not suffer from "Time Of Check to Time Of Use" (TOCTOU) issues or other race conditions for sensitive operations.	✓	✓	367

Updated Requirements – Race conditions (now V15.4)

#	Description	Level
15.4.1	Verify that shared objects in multi-threaded code (such as caches, files, or in-memory objects accessed by multiple threads) are accessed safely by using thread-safe types and synchronization mechanisms like locks or semaphores to avoid race conditions and data corruption.	3
15.4.2	Verify that checks on a resource's state, such as its existence or permissions, and the actions that depend on them are performed as a single atomic operation to prevent time-of-check to time-of-use (TOCTOU) race conditions. For example, checking if a file exists before opening it, or verifying a user's access before granting it.	3
15.4.3	Verify that locks are used consistently to avoid threads getting stuck, whether by waiting on each other or retrying endlessly, and that locking logic stays within the code responsible for managing the resource to ensure locks cannot be inadvertently or maliciously modified by external classes or code.	3
15.4.4	Verify that resource allocation policies prevent thread starvation by ensuring fair access to resources, such as by leveraging thread pools, allowing lower-priority threads to proceed within a reasonable timeframe.	3

```go
func Purchase(w http.ResponseWriter, r *http.Request) {
    ctx := r.Context()
    user := r.URL.Query().Get("user")
    amountCents, _ :=
    strconv.ParseInt(r.URL.Query().Get("amount_cents"),
    10, 64)

    // 1. Read current balance
    var balance int64
    err := db.QueryRowContext(ctx,
        "SELECT balance_cents FROM wallets WHERE
        user_id = $1",
    user,
    ).Scan(&balance)
    if err != nil {
        http.Error(w, "not found", 404)
        return
    }

    if balance < amountCents {
        http.Error(w, "insufficient funds", 402)
        return
    }
```

```go
    // 2. Call external payment processor
    resp, err := http.Post(
    "https://payments.asvsdemo.com/charge",
    "application/json",
    amountCents,
    )
    if err != nil || resp.StatusCode != 200 {
        http.Error(w, "payment failed", 502)
        return
    }

    // 3. Deduct balance
    _, err = db.ExecContext(ctx,
    "UPDATE wallets SET balance_cents = balance_cents -
    $1 WHERE user_id = $2",
    amountCents, user,
    )
    if err != nil {
        http.Error(w, "db error", 500)
        return
    }

    w.Write([]byte("ok"))

}
```

ASVS
APPLICATION SECURITY VERIFICATION STANDARD

```go
func Refund(ctx context.Context, user string, cents int64) error {
    if err := db.ExecContext(ctx, "SELECT pg_advisory_lock(hashtext('wallet:' || $1))", user); err != nil {
    return err }
    defer db.ExecContext(ctx, "SELECT pg_advisory_unlock(hashtext('wallet:' || $1))", user)

    if err := db.ExecContext(ctx, "SELECT pg_advisory_lock(hashtext('ledger:' || $1))", user); err != nil {
    return err }
    defer db.ExecContext(ctx, "SELECT pg_advisory_unlock(hashtext('ledger:' || $1))", user)

    // update wallet + ledger...
    return nil
}

func Purchase(ctx context.Context, user string, cents int64) error {
    db.ExecContext(ctx, "SELECT pg_advisory_lock(hashtext('ledger:' || $1))", user)
    defer db.ExecContext(ctx, "SELECT pg_advisory_unlock(hashtext('ledger:' || $1))", user)

    db.ExecContext(ctx, "SELECT pg_advisory_lock(hashtext('wallet:' || $1))", user)
    defer db.ExecContext(ctx, "SELECT pg_advisory_unlock(hashtext('wallet:' || $1))", user)

    // update wallet + ledger...
    return nil
}
```

OWASP®

ASVS
APPLICATION SECURITY VERIFICATION STANDARD

Conclusion

CALLS TO ACTION

Other new sections worth noting

- V4.4: WebSockets
- V17: WebRTC
- Appendix C: Cryptography
- Appendix D: Recommendations

Want a full list of what's new?

- https://asvs.dev/Mappings/
- https://asvs.dev/v5.0.0/For-Users-Of-4.0/

Related OWASP Projects You Should Also Care About

Awareness documents

- OWASP Top 10
- Top 10 Proactive Controls

Requirements for other domains

- MASVS
- ISVS
- SPVS
- AISVS

Testing guides

- WSTG
- MSTG

Other guidance documents

- Cheat Sheet Series
- SAMM

Mappings

- OpenCRE

OWASP Project Wayfinder

- https://owasp.org/projects/

It’s the LLMs’ world and we’re all just living in it

- Rigor is even more important for vibe coders!
- Even your favorite bot du jour needs standards to follow
- The faster we build, the more we need guardrails
- Keep an eye out for projects aiming to integrate ASVS with agents, MCP servers etc.

Thanks :)
Any Questions?

Eden Yardeni

eden.yardeni@owasp.org

eden@cronchie.com

@cronchie

ASVS Project

@OWASP_ASVS

https://github.com/OWASP/ASVS

https://owasp.slack.com, #project-asvs

https://owasp.org/asvs
https://asvs.dev