THIS DOCUMENT IS CURRENTLY IN DRAFT AND ACTIVELY BEING WORKED ON

Blueprints (ABOM + BOB)

Use Cases and Requirements

OWASP CycloneDX Workgroup	Blueprints FWG
Ecma Technical Committee	TC54
Ecma Task Group	N/A
Meeting Invite	Google Calendar Link
GitHub Issue	CycloneDX/specification/issues/463
Behavior Taxonomy	https://docs.google.com/spreadsheets/d/1Ztbpb_DCbJoEzE2Bt8mUJIBBmlfPt2eamnvVqvbzjs8/edit
Slack Channel	#workstream-blueprints (invite)

The concept of "Blueprints," proposed to be introduced in CycloneDX v1.7, will mark a significant evolution in software and system transparency. This new feature comprises two key elements: the Architectural Bill of Materials (ABOM) and the Bill of Behaviors (BOB). The ABOM provides a comprehensive description of how software or a system is architected, detailing the structural components and their interrelationships. The BOB, on the other hand, delineates the expected behavior of the system, and can also capture deviations by describing the actual behavior observed.

Blueprints aim to empower defenders by offering detailed insights into both the architecture and behavior of systems, enabling a more proactive and informed defense strategy. Historically, defenders have derived limited direct benefit from Software Bill of Materials (SBOM). However, with the introduction of Blueprints, OWASP CycloneDX envisions a paradigm shift towards active defense through enhanced transparency and understanding of software and systems.

Use Case Categories

K1: Threat Modeling

Understand attack surface, security defenses, risky behaviors, and backend connections. Identify risks -- missing, broken, and misused security defenses. Diffs between versions, including desired vs. as-built. Evaluate changes to security architecture, such as new attack surface, risky behaviors, and backend connections.

K2: Vulnerability Management

Understand impact of risks in context of entire system. Put risks in context, score risks, prioritize, explain vulnerability and attacks.

K3: Incident Response

Understand criticality of an incident by putting it in the context of the entire system. Identify and explain defense options. Communicate risks and their context to system owners, management, legal, users and other stakeholders.

K4: Penetration Testing

Ensure testing of the entire attack surface. Identify viable attacks that are relevant for each attack endpoint. Accelerate testing by focusing on tests that are viable for each attack endpoint. Understand the target assets reachable from each attack endpoint.

K5: Security Architecture

Capture both as-built and desired architectures. Identify Unexpected behavior.

K6: Compliance

capture and communicate security architecture to establish compliance with requirements or regulations.

Use Cases

U1: As a consumer, I need to defend my organization against unexpected behavior.

In order to achieve this, expected behavior must be communicated.

U2: As a consumer, I need to dynamically protect the organization in part by identifying the expected behaviors.

Possible scenarios are IPs an app needs to reach out to as being expected behavior, either the addition of new IPs and the removal of them. Host-based behaviors such as EDR potentially dangerous things are expected behaviors. By identifying them, we can potentially reduce false positives in security tools.

U3: As a consumer, I need to predict if changes in behaviors will alter the outcomes of an application or process.

Given two or more BOBs, it would be possible to identify potential incompatibilities between behaviors. This is especially important in national security and public safety.

U4: As a quality or security engineer, I need to validate that the system or application is behaving as documented.

Anomalies may be due to product defects, configuration issues, etc

U5: As a consumer, quality or security engineer, I need to inform the manufacturer about variances in behavior.

This could be applicable to manufacturing, forensics, etc.

U6: As a product manufacturer, I need to communicate the architecture of an application or system for regulatory or compliance purposes.

U7: As a security engineer, I need to know what is my attack surface based on the architecture of an application or system.

U8: As an operating supervisor, I need to know the potential impact on the behavior of a process or system should one of the components in the architecture go down or have problems.

This goes into resilience.

Requirements

R1: Each “thing” described in the architecture should be able to reference o..n behaviors.

R2: Reuse or support existing standards where applicable and provide more of a summary/abstract of a given architecture.

R3: Integrate with the optional diagrams and assets described in TM-BOM.

NOTES:
Things that we should capture:

- Attack surface
    - A metadata component which includes nothing but services. However, we may want to think about using “provides” or creating a new dependency type called “exposes” to make it clear that the application provides the services, not depends on them.
- security controls
    - High level categories such as authn/authz, input validation, encryption, logging, etc

    - Possibly start with an enum, but absolutely make it extensible. Some security controls haven't been invented yet.
    - Include details such as info typically found in a CBOM or for authn/z include MFA, OIDC, SAML, etc. Basically, include some of the details to provide context.
    - What types of logging (including format), API, console, debug, audit, authn/z, where are the logs going (syslog, splunk, etc)
    -

- dangerous behavior
    - Possibly a slope. A badnessometer. Is it at the root cause of a vulnerability class, such as SQLi.
    - A lack of badness needs to be flushed out. Needs to be able to describe what the tool senses so that the absence of badness can be identified as either a true lack of badness, or a limitation of the tools (false negative).
    - Needs to be extensible.
    -

- outbound initiated connections
    - Which thing initiates a connection may be a gap
    -

NOTES:

- Possible need for a taxonomy to describe expected behavior, or at a minimum a way to describe the "core functionality" so that deviations can be observed.
- Will need to take into consideration configuration components and how configuration alters behavior.
- Spec should be able to document expected behavior (proactive), as-built (the reality), and "bad" behaviors to look out for. Spec should account for iterative improvements over time.