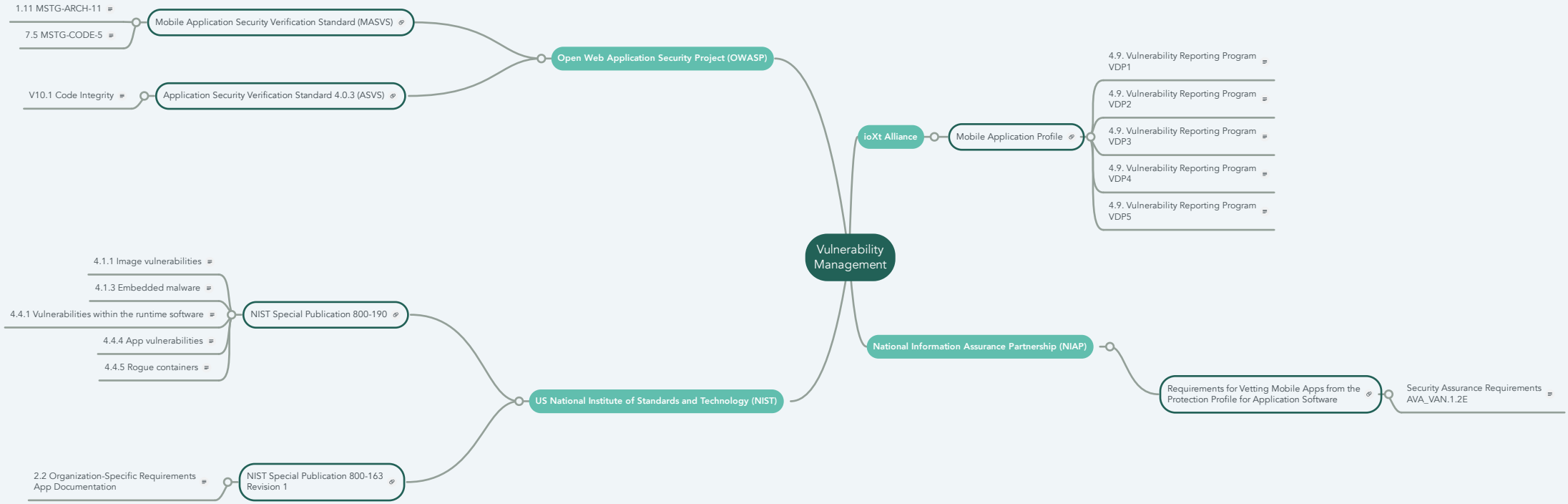
Vulnerability Management
Open Web Application Security Project (OWASP)
Mobile Application Security Verification Standard (MASVS)
1.11 MSTG-ARCH-11
7.5 MSTG-CODE-5
Application Security Verification Standard 4.0.3 (ASVS)
V10.1 Code Integrity
US National Institute of Standards and Technology (NIST)
NIST Special Publication 800-190
4.1.1 Image vulnerabilities
4.1.3 Embedded malware
4.4.1 Vulnerabilities within the runtime software
4.4.4 App vulnerabilities
4.4.5 Rogue containers
NIST Special Publication 800-163 Revision 1
2.2 Organization-Specific Requirements App Documentation
ioXt Alliance
Mobile Application Profile
4.9. Vulnerability Reporting Program VDP1
4.9. Vulnerability Reporting Program VDP2
4.9. Vulnerability Reporting Program VDP3
4.9. Vulnerability Reporting Program VDP4
4.9. Vulnerability Reporting Program VDP5
National Information Assurance Partnership (NIAP)
Requirements for Vetting Mobile Apps from the Protection Profile for Application Software
Security Assurance Requirements AVA_VAN.1.2E

Vulnerability Management

1. ioXt Alliance

1.1. Mobile Application Profile

Link: https://static1.squarespace.com/static/5c6dbac1f8135a29c7fbb621/t/604aa3fa668a8e3b50630433/1615504379349/Mobile_Application_Profile.pdf

1.1.1. 4.9. Vulnerability Reporting Program VDP1

VDP in place

1.1.2. 4.9. Vulnerability Reporting Program VDP2

Accept external submissions

1.1.3. 4.9. Vulnerability Reporting Program VDP3

Monitoring security relevant components.

1.1.4. 4.9. Vulnerability Reporting Program VDP4

Responsible disclosure of defects to impacted parties that must take action.

1.1.5. 4.9. Vulnerability Reporting Program VDP5

Public Researcher Rewards program

2. National Information Assurance Partnership (NIAP)

2.1. Requirements for Vetting Mobile Apps from the Protection Profile for Application Software

Link: https://www.niap-ccevs.org/MMO/PP/394.R/pp_app_v1.2_table-reqs.htm

2.1.1. Security Assurance Requirements AVA_VAN.1.2E

The evaluator shall perform a search of public domain sources to identify potential vulnerabilities in the TOE.
Application Note: Public domain sources include the Common Vulnerabilities and Exposures (CVE) dictionary for publicly known vulnerabilities. Public domain sources also include sites which provide free checking of files for viruses.

3. US National Institute of Standards and Technology (NIST)

3.1. NIST Special Publication 800-190

Link: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-190.pdf

3.1.1. 4.1.1 Image vulnerabilities

There is a need for container technology-specific vulnerability management tools and processes. Traditional vulnerability management tools make many assumptions about host durability and app update mechanisms and frequencies that are fundamentally misaligned with a containerized model. These tools are often unable to detect vulnerabilities within containers, leading to a false sense of safety.

Organizations should use tools that take the pipeline-based build approach and immutable nature of containers and images into their design to provide more actionable and reliable results. Key aspects of effective tools and processes include:
1. Integration with the entire lifecycle of images, from the beginning of the build process, to whatever registries the organization is using, to runtime.
2. Visibility into vulnerabilities at all layers of the image, not just the base layer of the image but also application frameworks and custom software the organization is using. Visibility should be centralized across the organization and provide flexible reporting and monitoring views aligned with organizations' business processes.
3. Policy-driven enforcement; organizations should be able to create "quality gates" at each stage of the build and deployment process to ensure that only images that meet the organization's vulnerability and configuration policies are allowed to progress. For example, organizations should be able to configure a rule in the build process to prevent the progression of images that include vulnerabilities with Common Vulnerability Scoring System (CVSS) [18] ratings above a selected threshold.

3.1.2. 4.1.3 Embedded malware

Organizations should continuously monitor all images for embedded malware. The monitoring processes should include the use of malware signature sets and behavioral detection heuristics based largely on actual "in the wild" attacks.

3.1.3. 4.4.1 Vulnerabilities within the runtime software

The container runtime must be carefully monitored for vulnerabilities, and when problems are detected, they must be remediated quickly. A vulnerable runtime exposes all containers it supports, as well as the host itself, to potentially significant risk. Organizations should use tools to look for Common Vulnerabilities and Exposures (CVEs) vulnerabilities in the runtimes deployed, to upgrade any instances at risk, and to ensure that orchestrators only allow deployments to properly maintained runtimes.

3.1.4. 4.4.4 App vulnerabilities

Existing host-based intrusion detection processes and tools are often unable to detect and prevent attacks within containers due to the differing technical architecture and operational practices previously discussed. Organizations should implement additional tools that are container aware and designed to operate at the scale and change rate typically seen with containers. These tools should be able to automatically profile containerized apps using behavioral learning and build security profiles for them to minimize human interaction. These profiles should then be able to prevent and detect anomalies at runtime, including events such as:
• Invalid or unexpected process execution,
• Invalid or unexpected system calls,
• Changes to protected configuration files and binaries,
• Writes to unexpected locations and file types,
• Creation of unexpected network listeners,
• Traffic sent to unexpected network destinations, and
• Malware storage or execution.

Containers should also be run with their root filesystems in read-only mode. This approach isolates writes to specifically defined directories, which can then be more easily monitored by the aforementioned tools. Furthermore, using read-only filesystems makes the containers more resilient to compromise since any tampering is isolated to these specific locations and can be easily separated from the rest of the app.

3.1.5. 4.4.5 Rogue containers

Organizations should institute separate environments for development, test, production, and other scenarios, each with specific controls to provide role-based access control for container deployment and management activities. All container creation should be associated with individual user identities and logged to provide a clear audit trail of activity. Further, organizations are encouraged to use security tools that can enforce baseline requirements for vulnerability management and compliance prior to allowing an image to be run.

3.2. NIST Special Publication 800-163 Revision 1

Link: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-163r1.pdf

3.2.1. 2.2 Organization-Specific Requirements App Documentation

> Test Results
> Code review results and other testing results will indicate which security standards were followed. For example, if an app threat model was created, this standard should be submitted. It will list weaknesses that were identified and should have been addressed during app design and coding.

4. Open Web Application Security Project (OWASP)

4.1. Mobile Application Security Verification Standard (MASVS)

Link: https://github.com/OWASP/owasp-masvs/releases/tag/v1.4.2

4.1.1. 1.11 MSTG-ARCH-11

> A responsible disclosure policy is in place and effectively applied.

4.1.2. 7.5 MSTG-CODE-5

> All third party components used by the mobile app, such as libraries and frameworks, are identified, and checked for known vulnerabilities.

4.2. Application Security Verification Standard 4.0.3 (ASVS)

Link: https://raw.githubusercontent.com/OWASP/ASVS/v4.0.3/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0.3-en.pdf

4.2.1. V10.1 Code Integrity

> 10.1.1 Verify that a code analysis tool is in use that can detect potentially malicious code, such as time functions, unsafe file operations and network connections.