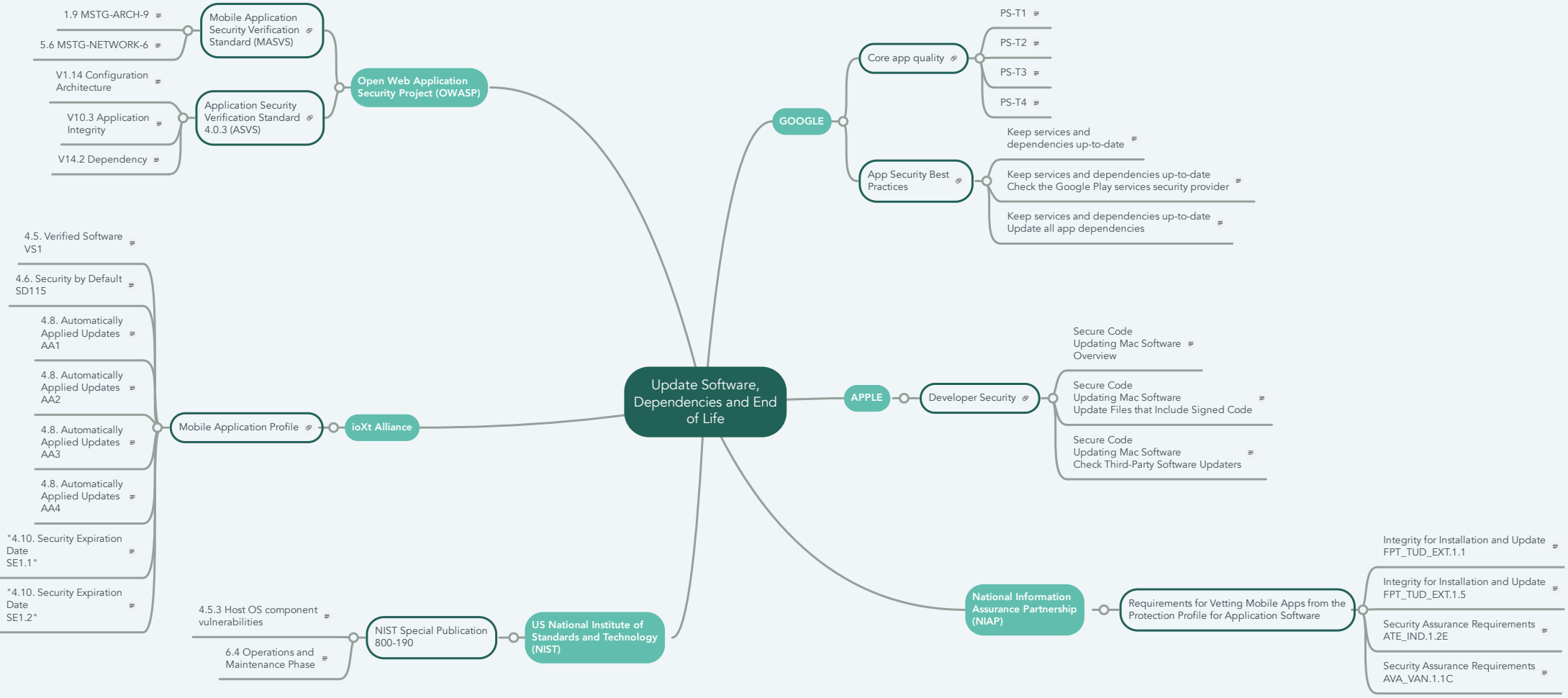
Update Software, Dependencies and End of Life
GOOGLE
Core app quality
PS-T1
PS-T2
PS-T3
PS-T4
App Security Best Practices
Keep services and dependencies up-to-date
Keep services and dependencies up-to-date
Check the Google Play services security provider
Keep services and dependencies up-to-date
Update all app dependencies
APPLE
Developer Security
Secure Code
Updating Mac Software
Overview
Secure Code
Updating Mac Software
Update Files that Include Signed Code
Secure Code
Updating Mac Software
Check Third-Party Software Updaters
National Information Assurance Partnership (NIAP)
Requirements for Vetting Mobile Apps from the Protection Profile for Application Software
Integrity for Installation and Update
FPT_TUD_EXT.1.1
Integrity for Installation and Update
FPT_TUD_EXT.1.5
Security Assurance Requirements
ATE_IND.1.2E
Security Assurance Requirements
AVA_VAN.1.1C
US National Institute of Standards and Technology (NIST)
NIST Special Publication 800-190
4.5.3 Host OS component vulnerabilities
6.4 Operations and Maintenance Phase
ioXt Alliance
Mobile Application Profile
4.5. Verified Software
VS1
4.6. Security by Default
SD115
4.8. Automatically Applied Updates
AA1
4.8. Automatically Applied Updates
AA2
4.8. Automatically Applied Updates
AA3
4.8. Automatically Applied Updates
AA4
"4.10. Security Expiration Date
SE1.1"
"4.10. Security Expiration Date
SE1.2"
Open Web Application Security Project (OWASP)
Mobile Application Security Verification Standard (MASVS)
1.9 MSTG-ARCH-9
5.6 MSTG-NETWORK-6
Application Security Verification Standard 4.0.3 (ASVS)
V1.14 Configuration Architecture
V10.3 Application Integrity
V14.2 Dependency

Update Software, Dependencies and End of Life

1. GOOGLE

1.1. Core app quality

Link: https://developer.android.com/docs/quality-guidelines/core-app-quality

1.1.1. PS-T1

> The app runs on the latest public version of the Android platform without crashing or severely impacting core functionality.

1.1.2. PS-T2

> The app targets the latest Android SDK needed to align with Google Play requirements by setting the targetSdk value.

1.1.3. PS-T3

> The app is built with the latest Android SDK by setting the compileSdk value.

1.1.4. PS-T4

> Any Google or third-party SDKs used are up-to-date. Any improvements to these SDKs, such as stability, compatibility, or security, should be available to users in a timely manner.
> For Google SDKs, consider using SDKs powered by Google Play services, when available. These SDKs are backward compatible, receive automatic updates, reduce your app package size, and make efficient use of on-device resources.
>
> The developer is accountable for the entire app’s codebase, inclusive of any third-party SDKs used.

1.2. App Security Best Practices

Link: https://developer.android.com/topic/security/best-practices

1.2.1. Keep services and dependencies up-to-date

> Most apps use external libraries and device system information to complete specialized tasks. By keeping your app's dependencies up to date, you make these points of communication more secure.

1.2.2. Keep services and dependencies up-to-date Check the Google Play services security provider

> If your app uses Google Play services, make sure that it's updated on the device where your app is installed. This check should be done asynchronously, off of the UI thread. If the device isn't up-to-date, your app should trigger an authorization error.
>
> To determine whether Google Play services is up to date on the device where your app is installed, follow the steps in the guide for Updating Your Security Provider to Protect Against SSL Exploits.

1.2.3. Keep services and dependencies up-to-date Update all app dependencies

> Before deploying your app, make sure that all libraries, SDKs, and other dependencies are up to date:
>
> For first-party dependencies, such as the Android SDK, use the updating tools found in Android Studio, such as the SDK Manager.
> For third-party dependencies, check the websites of the libraries that your app uses, and install any available updates and security patches.

2. APPLE

2.1. Developer Security

Link: https://developer.apple.com/documentation/security

2.1.1. Secure Code Updating Mac Software Overview

Many Mac software products include a software updater. For example:

An app might ask the user whether they want to download and install the latest version.

An enterprise IT department might install a daemon that updates the enterprise's custom software each night.

If you write a software updater in the simplest way, you run the risk of a hard-to-reproduce crash in the newly updated software. The symptoms of this problem are:

Some seemingly random part of the updated code crashes.

The associated crash report includes the message Code Signature Invalid.

The problem goes away when you restart the Mac.

2.1.2. Secure Code Updating Mac Software Update Files that Include Signed Code

Imagine code that downloads an update for a command-line tool. The final step of that process might look like this:

```
import System
import Darwin
// Do not do this. Modifying signed code in place puts you at risk of a
// code-signing crash.

func updateToolBadly(at executablePath: String, with newContents: [UInt8]) throws {
    let fd = try FileDescriptor.open(executablePath, .writeOnly, options: [.truncate])
    defer {
        try! fd.close()
    }
    try fd.writeAll(newContents)
}
```

This code is incorrect because it modifies the command-line tool's executable file in place. macOS caches information about the code's signature in the kernel. It doesn't flush that cache when you modify the file's contents. Modifying the file in place yields a mismatch between the file's contents and the in-kernel cache, which can cause a hard-to-reproduce code-signing crash the next time you run the tool.

While this example uses a command-line tool to demonstrate the issue, updating any file that contains signed code might trigger this code-signing crash. That includes executables, frameworks, dynamic libraries, and bundles.

To update a file that contains signed code without risking this crash, write the updated code to a temporary file and replace the existing file with that temporary one:

```
func updateTool(at executablePath: String, with newContents: [UInt8], temporaryPath: String) throws {
    do {
        let permissions: FilePermissions = [[.ownerReadWriteExecute, .groupReadExecute, .otherReadExecute]]
        let fd = try FileDescriptor.open(temporaryPath, .writeOnly, options: [.create], permissions: permissions)
        defer {
            try! fd.close()
        }
        try fd.writeAll(newContents)
    }
    let success = rename(temporaryPath, executablePath) >= 0
```

```
    guard success else { throw Errno(rawValue: errno) }
}
```

Using a temporary file eliminates the risk of a code-signing crash because the in-kernel cache is associated with the old file, which remains unmodified. The new file gets its own in-kernel cache, built from the contents of that new file.

2.1.3. Secure Code Updating Mac Software Check Third-Party Software Updaters

If you wrote the code for your software updater, use the techniques discussed above to identify and correct this potential code-signing issue. You can also check software updater code that you didn't write by running ls with the -i option before and after the update, to see whether the inode number changes. For example, imagine you're installing an update using cp and your want to update MyTool with a new version, MyTool-new. Run the following, to test whether this exposes you to a code-signing crash:

```
% ls -i MyTool
78290841 MyTool
% cp MyTool-new MyTool
% ls -i MyTool
78290841 MyTool
```

Note that the inode number didn't change, which indicates that cp modified the MyTool file in place. This puts you at risk of a code-signing crash the next time you run MyTool.

Now, repeat the test, but use ditto in place of cp:

```
% ls -i MyTool
78290841 MyTool
% ditto MyTool-new MyTool
% ls -i MyTool
78413900 MyTool
```

This time, the inode number changed. This change indicates that ditto replaced the MyTool file with an entirely new file, and thus avoided this potential code-signing crash.

3. National Information Assurance Partnership (NIAP)

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

3.1.1. Integrity for Installation and Update FPT_TUD_EXT.1.1

The application shall [selection: provide the ability, leverage the platform] to check for updates and patches to the application software.
Application Note: This requirement is about the ability to ""check"" for updates. The actual installation of any updates should be done by the platform. This requirement is intended to ensure that the application can check for updates provided by the vendor, as updates provided by another source may contain malicious code.

3.1.2. Integrity for Installation and Update FPT_TUD_EXT.1.5

The application shall [selection, at least one of: provide the ability, leverage the platform] to query the current version of the application software.

3.1.3. Security Assurance Requirements ATE_IND.1.2E

The evaluator shall test a subset of the TSF to confirm that the TSF operates as specified.
Application Note: The evaluator shall test the application on the most current fully patched version of the platform.

3.1.4. Security Assurance Requirements AVA_VAN.1.1C

The application shall be suitable for testing.
Application Note: Suitability for testing means not being obfuscated or packaged in such a way as to disrupt either static or dynamic analysis by the evaluator.

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

4.1. NIST Special Publication 800-190

4.1.1. 4.5.3 Host OS component vulnerabilities

Organizations should implement management practices and tools to validate the versioning of components provided for base OS management and functionality. Even though containerspecific OSs have a much more minimal set of components than general-purpose OSs, they still do have vulnerabilities and still require remediation. Organizations should use tools provided by the OS vendor or other trusted organizations to regularly check for and apply updates to all software components used within the OS. The OS should be kept up to date not only with security updates, but also the latest component updates recommended by the vendor. This is particularly important for the kernel and container runtime components as newer releases of these components often add additional security protections and capabilities beyond simply correcting vulnerabilities. Some organizations may choose to simply redeploy new OS instances with the necessary updates, rather than updating existing systems. This approach is also valid, although it often requires more sophisticated operational practices.

Host OSs should be operated in an immutable manner with no data or state stored uniquely and persistently on the host and no application-level dependencies provided by the host. Instead, all app components and dependencies should be packaged and deployed in containers. This enables the host to be operated in a nearly stateless manner with a greatly reduced attack surface. Additionally, it provides a more trustworthy way to identify anomalies and configuration drift.

4.1.2. 6.4 Operations and Maintenance Phase

Operational processes that are particularly important for maintaining the security of container technologies, and thus should be performed regularly, include updating all images and distributing those updated images to containers to take the place of older images. Other security best practices, such as performing vulnerability management and updates for other supporting layers like hosts and orchestrators, are also key ongoing operational tasks. Container security and monitoring tools should similarly be integrated with existing security information and event management (SIEM) tools to ensure container-related events flow through the same tools and processes used to provide security throughout the rest of the environment.

If and when security incidents occur within a containerized environment, organizations should be prepared to respond with processes and tools that are optimized for the unique aspects of containers. The core guidance outlined in NIST SP 800-61, Computer Security Incident Handling Guide [28], is very much applicable for containerized environments as well. However, organizations adopting containers should ensure they enhance their responses for some of the unique aspects of container security.
• Because containerized apps may be run by a different team than the traditional operations team, organizations should ensure that whatever teams are responsible for container operations are brought into the incident response plan and understand their role in it.
• As discussed throughout this document, the ephemeral and automated nature of container management may not be aligned with the asset management policies and tools an organization has traditionally used. Incident response teams must be able to know the roles, owners, and sensitivity levels of containers, and be able to integrate this data into their process.
• Clear procedures should be defined to respond to container related incidents. For example, if a particular image is being exploited, but that image is in use across hundreds of containers, the response team may need to shut down all of these containers to stop the attack. While single vulnerabilities have long been able to cause problems across many systems, with containers, the response may require rebuilding and redeploying a new image widely, rather than installing a patch to existing systems. This change in response may involve different teams and approvals and should be understood and practiced ahead
of time.
• As discussed previously, logging and other forensic data may be stored differently in a containerized environment. Incident response teams should be familiar with the different tools and techniques required to gather data and have documented processes specifically for these environments.

5. ioXt Alliance

5.1. Mobile Application Profile

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

5.1.1. 4.5. Verified Software VS1

Manufacturer has an update patch policy.

5.1.2. 4.6. Security by Default SD115

Dependencies are patched from known security vulnerabilities.

5.1.3. 4.8. Automatically Applied Updates AA1

Software updates supported

5.1.4. 4.8. Automatically Applied Updates AA2

Software is Maintained and Updated

5.1.5. 4.8. Automatically Applied Updates AA3

Software updates are made available to impacted parties

5.1.6. 4.8. Automatically Applied Updates AA4

Security Updates applied automatically, when product usage allows

5.1.7. "4.10. Security Expiration Date SE1.1"

End of life notification policy is published

5.1.8. "4.10. Security Expiration Date SE1.2"

Expiration Date is published

6. Open Web Application Security Project (OWASP)

6.1. Mobile Application Security Verification Standard (MASVS)

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

6.1.1. 1.9 MSTG-ARCH-9

> A mechanism for enforcing updates of the mobile app exists.

6.1.2. 5.6 MSTG-NETWORK-6

> The app only depends on up-to-date connectivity and security libraries.

6.2. 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

6.2.1. V1.14 Configuration Architecture

> 1.14.3 Verify that the build pipeline warns of out-of-date or insecure components and takes appropriate actions.

6.2.2. V10.3 Application Integrity

> 10.3.1 Verify that if the application has a client or server auto-update feature, updates should be obtained over secure channels and digitally signed. The update code must validate the digital signature of the update before installing or executing the update.

6.2.3. V14.2 Dependency

> 14.2.1 Verify that all components are up to date, preferably using a dependency checker during build or compile time.
>
> 4.2.4 Verify that third party components come from pre-defined, trusted and continually maintained repositories.
>
> 14.2.5 Verify that a Software Bill of Materials (SBOM) is maintained of all third party libraries in use.