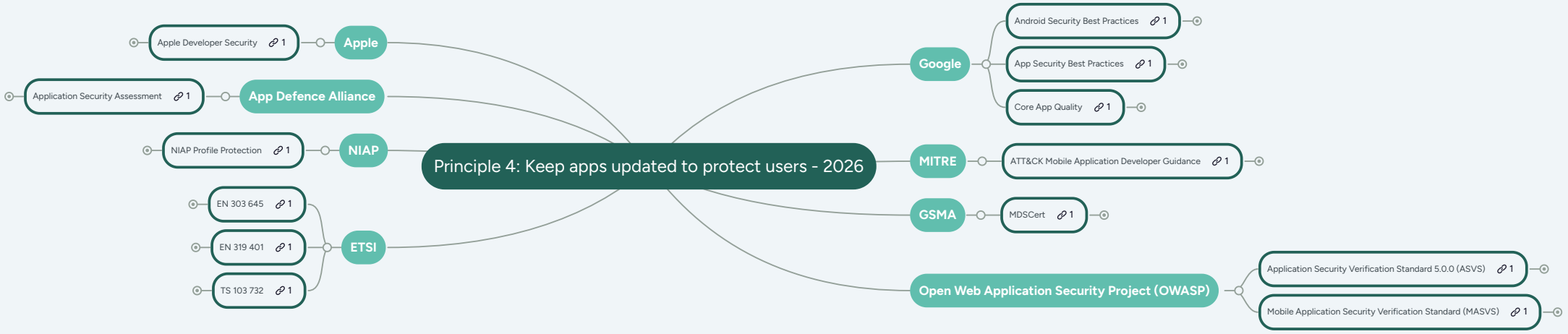
Principle 4: Keep apps updated to protect users - 2026
Apple
Apple Developer Security 1
App Defence Alliance
Application Security Assessment 1
NIAP
NIAP Profile Protection 1
ETSI
EN 303 645 1
EN 319 401 1
TS 103 732 1
Google
Android Security Best Practices 1
App Security Best Practices 1
Core App Quality 1
MITRE
ATT&CK Mobile Application Developer Guidance 1
GSMA
MDSCert 1
Open Web Application Security Project (OWASP)
Application Security Verification Standard 5.0.0 (ASVS) 1
Mobile Application Security Verification Standard (MASVS) 1

Principle 4: Keep apps updated to protect users - 2026

1. Google

1.1. Android Security Best Practices

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

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

1.1.1. 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.

1.1.2. 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. Perform the check asynchronously, off of the UI thread. If the device isn't up to date, trigger an authorization error.

1.1.3. 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.

1.2. App Security Best Practices

Link: https://developer.android.com/privacy-and-security/security-tips

Link: https://developer.android.com/privacy-and-security/security-tips

1.2.1. WebView-5

Devices running platforms older than Android 4.4 (API level 19) use a version of webkit that has a number of security issues. As a workaround, if your app is running on these devices, it must confirm that WebView objects display only trusted content. To make sure your app isn't exposed to potential vulnerabilities in SSL, use the updatable security Provider object as described in Update your security provider to protect against SSL exploits. If your application must render content from the open web, consider providing your own renderer so you can keep it up to date with the latest security patches.

1.2.2. Stay vigilant-1

Keep your code up-to-date. Be sure to update your source code, including any third-party libraries and dependencies, to guard against the latest vulnerabilities.

1.2.3. General best practices-6

SDKs up-to-date: Make sure your SDKs and libraries are updated to the latest version.

1.2.4. Cryptography-2

Know which Java Cryptography Architecture (JCA) security providers your software uses. Try to use the highest level of the pre-existing framework implementation that can support your use case. If applicable, use the Google-provided providers in the Google-specified order.

1.3. Core App Quality

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

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

1.3.1. PS-T1

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

1.3.2. PS-T2

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

1.3.3. PS-T3

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

1.3.4. 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.

2. MITRE

2.1. ATT&CK Mobile Application Developer Guidance

Link: https://attack.mitre.org/versions/v18/mitigations/M1013/

Link: https://attack.mitre.org/versions/v18/mitigations/M1013/

2.1.1. T1474.001 Compromise Software Dependencies and Development Tools

Application developers should be cautious when selecting third-party libraries to integrate into their application.

3. GSMA

3.1. MDSCert

Link: https://www.gsma.com/solutions-and-impact/technologies/security/wp-content/uploads/2025/02/FS.56-v1.0.pdf

Link: https://www.gsma.com/solutions-and-impact/technologies/security/wp-content/uploads/2025/02/FS.56-v1.0.pdf

3.1.1. DP_UPF_EXT.1.1/SSW_Update

The Target of Evaluation (TOE) Security Functionality (TSF) shall be able to check for a [SYSTEM SOFTWARE] update package every [assignment: an interval of no more than 1 month] and provide a notification to the user when an update is available.

3.1.2. ALC_CMC.2.4C

The Configuration Management (CM) documentation shall describe the method used to identify external software components and how those components are updated.

3.1.3. ALC_CMC.2.4D

The developer shall provide guidance for importing and updating external software components into the CM system.

3.1.4. ALC_FLR.3.4D

The developer shall provide public guidance related to the duration period, frequency and type of updates that will be released to support the TOE.

3.1.5. ALC_FLR.3.12C

The flaw remediation procedures documentation shall define the planned minimum duration after release of the TOE that these methods will be used to maintain the TOE.

3.1.6. ALC_FLR.3.13C

The flaw remediation procedures documentation shall define the types of updates (such as security/maintenance or operating system) and the frequency of these updates being provided for the TOE.

4. Open Web Application Security Project (OWASP)

4.1. Application Security Verification Standard 5.0.0 (ASVS)

Link: https://scotthelme.co.uk/content/files/2025/06/OWASP_Application_Security_Verification_Standard_5.0.0_en.pdf

Link: https://scotthelme.co.uk/content/files/2025/06/OWASP_Application_Security_Verification_Standard_5.0.0_en.pdf

4.1.1. V3.6 External Resource Integrity

3.6.1
Verify that client-side assets, such as JavaScript libraries, CSS, or web fonts, are only hosted externally (e.g., on a Content Delivery Network) if the resource is static and versioned and Subresource Integrity (SRI) is used to validate the integrity of the asset. If this is not possible, there should be a documented security decision to justify this for each resource.

4.1.2. V15.1 Secure Coding and Architecture Documentation

15.1.1
Verify that application documentation defines risk based remediation time frames for 3rd party component versions with vulnerabilities and for updating libraries in general, to minimize the risk from these components._x000D_

15.1.2
Verify that an inventory catalog, such as software bill of materials (SBOM), is maintained of all third-party libraries in use, including verifying that components come from pre-defined, trusted, and continually maintained repositories.

15.1.4
Verify that application documentation highlights third-party libraries which are considered to be “risky components”._x000D_

15.1.5
Verify that application documentation highlights parts of the application where “dangerous functionality”is being used._x000D_

4.1.3. V15.2 Security Architecture and Dependencies

15.2.1
Verify that the application only contains components which have not breached the documented update and remediation time frames.

15.2.4
Verify that third-party components and all of their transitive dependencies are included from the expected repository, whether internally owned or an external source, and that there is no risk of a dependency confusion attack._x000D_

4.2. Mobile Application Security Verification Standard (MASVS)

Link: https://github.com/OWASP/owasp-masvs/releases/tag/v2.1.0

Link: https://github.com/OWASP/owasp-masvs/releases/tag/v2.1.0

4.2.1. MASVS-CODE-1

> The app requires an up-to-date platform version.
>
> Every release of the mobile OS includes security patches and new security features. By supporting older versions, apps stay vulnerable to well-known threats. This control ensures that the app is running on an up-to-date platform version so that users have the latest security protections.

4.2.2. MASVS-CODE-2

> The app has a mechanism for enforcing app updates.
>
> Sometimes critical vulnerabilities are discovered in the app when it is already in production. This control ensures that there is a mechanism to force the users to update the app before they can continue using it.

4.2.3. MASVS-CODE-3

> The app only uses software components without known vulnerabilities.
>
> To be truly secure, a full whitebox assessment should have been performed on all app components. However, as it usually happens with e.g. for third-party components this is not always feasible and not typically part of a penetration test. This control covers “low-hanging fruit” cases, such as those that can be detected just by scanning libraries for known vulnerabilities._x000D_

5. ETSI

5.1. EN 303 645

Link: https://www.etsi.org/deliver/etsi_en/303600_303699/303645/03.01.03_60/en_303645v030103p.pdf

Link: https://www.etsi.org/deliver/etsi_en/303600_303699/303645/03.01.03_60/en_303645v030103p.pdf

5.1.1. Principle 5.3-1

All software components in consumer IoT devices that are not immutable due to security reasons should be securely updateable.

NOTE 1: Managing software updates successfully generally relies on communication of version information for software components between the consumer IoT device and the manufacturer.

NOTE 2: For most devices the operating system, applications and network protocol stacks cannot be considered as immutable due to security reasons.

EXAMPLE 2: The first stage boot loader on a consumer IoT device is written once to device storage and from then on is immutable.

EXAMPLE 3: On consumer IoT devices with several microcontrollers (e.g. one for communication and one for the application) some of them might not be updateable.

5.1.2. Principle 5.3-2

The consumer IoT device shall have a secure update mechanism unless the device has a resource constraint determined by the use case that prevents the implementation.

NOTE 3: There are cases where provision 5.3-1 applies even where 5.3-2 does not.

NOTE 4: Use case determined resource constraints are technical constraints of a consumer IoT device that are the result of appropriate design decisions that are necessary to implement the use case of the consumer IoT device and cannot solely be justified by potential economic reasons to avoid the implementation of a secure update mechanism.

NOTE 5: Typical resource constraints that can prevent the implementation of a secure update mechanism are minimal communication bandwidth or energy supply.

EXAMPLE 4: A battery driven sensor designed to wirelessly communicate measurements with a low frequency over a for a long period of time might have use case determined resource constraints in communication bandwidth or energy supply that prevent the implementation of a secure update mechanism.

EXAMPLE 5: Design decisions for a consumer IoT device do not result in limited resources solely to avoid the costs associated with the implementation of a secure update mechanism but instead result in sufficient resources and the implementation of a secure update mechanism. "Securely updateable" and "secure update mechanism" means that there are adequate measures to prevent an attacker misusing the update mechanism.

EXAMPLE 6: Measures can include the use of authentic software update servers, integrity protected communications channels, verifying the authenticity and integrity of software updates. It is recognized that there are great variances in software update mechanisms and what constitutes "installation".

EXAMPLE 7: An anti-rollback policy based on version checking can be used to prevent downgrade attacks. Update mechanisms can range from the consumer IoT device downloading the update directly from a remote server, transmitted from a mobile application or transferred over a USB or other physical interface. If an attacker compromises this mechanism, it allows for a malicious version of the software to be installed on the consumer IoT device.

5.1.3. Principle 5.3-3

An update shall be simple for the user to apply.

The degree of simplicity depends on the design and intended usage of the consumer IoT device. Some examples can be given as follows, but are not exhaustive. An update that is simple to apply will be automatically applied, initiated using an associated service (such as a mobile application), or via a web interface on the consumer IoT device. If an update is difficult to apply, then that increases the chance that a user will repeatedly defer updating the consumer IoT device, leaving it in a vulnerable state.

5.1.4. Principle 5.3-4A

One secure update mechanism should be configurable to be automated.

NOTE 6: An automated update mechanism obtains information on available updates without user interaction when having suitable network access and either installs updates without user interaction or processes available updates after user's consent with minimal user interaction.

5.1.5. Principle 5.3-4B

During initialization the consumer IoT device should activate an automatic secure update mechanism after user's consent.

If an automatic update fails, then a user can, in some circumstances, no longer be able to use a device. Detection mechanisms such as watchdogs and the use of dual-bank flash or recovery partitions can ensure that the consumer IoT device returns to either a known good version or the factory state.

Security updates can be provided for consumer IoT devices in a preventative manner, as part of automatic updates, which can remove security vulnerabilities before they are exploited. Managing this can be complex, especially if there are parallel associated service updates, consumer IoT device updates and other service updates to deal with. Therefore, a clear management and deployment plan is beneficial to the manufacturer, as is transparency to consumers about the current state of update support.

5.1.6. Principle 5.3-5

The consumer IoT device or an associated service should check after initialization whether security updates are available.

EXAMPLE 8: The user could be shown the existence of updates via the interface with which the consumer IoT device is initialized.

For some products, it can be more appropriate for the associated service, rather than the consumer IoT device, to perform such checks.

5.1.7. Principle 5.3-6A

If the consumer IoT device supports automated updates, the user should be able to enable and disable the automatic update mechanism and postpone the installation of updates provided via an automatic update mechanism.

5.1.8. Principle 5.3-6B

If the consumer IoT device supports update notifications, the user should be able to enable and disable the update notifications.

It is important from a consumer rights and ownership perspective that the user is in control of whether or not they receive updates. There are good reasons why a user may choose not to update, including security. In addition, if an update is deployed and subsequently found to cause issues, manufacturers can ask users to not upgrade their software in order that those consumer IoT devices are not affected.

5.1.9. Principle 5.3-7

The consumer IoT device shall use best practice cryptography to facilitate secure update mechanisms.

5.1.10. Principle 5.3-8

Security updates shall be timely.

"Timely" in the context of security updates can vary, depending on the particular issue and fix, as well as other factors such as the ability to reach a device or considerations based on other device resource constraints. It is important that a security update that fixes a critical vulnerability (i.e. one with potentially adverse effects of a large scale) is handled with appropriate priority by the manufacturer. Once a fix is made available, users can be notified through a bulletin process or other appropriate means. Due to the complex structure of modern software and the ubiquity of communication platforms, multiple stakeholders can be involved in a security update.

EXAMPLE 9: A particular software update involves a third party vendor of software libraries, an IoT device manufacturer, and an IoT service platform operator. Collaboration between these stakeholders ensures appropriate timeliness of the software update.

5.1.11. Principle 5.3-9

The consumer IoT device should verify the authenticity and integrity of software updates.

A common approach for confirming that an update is valid is to verify its integrity and authenticity. This can be done on the consumer IoT device.

5.1.12. Principle 5.3-10

Where updates are delivered over a network interface, the consumer IoT device shall verify the authenticity and integrity of each update via a trust relationship.

NOTE 7: Valid trust relationships include: authenticated communication channels, presence on a network that requires the consumer IoT device to possess a critical security parameter or password to join, digital signature based verification of the update, or confirmation by the user.

NOTE 8: The validation of the trust relationship is essential to ensure that a non-authorized entity (e.g. consumer IoT device management platform or consumer IoT device) cannot install malicious code.

Consumer IoT devices can have power limitations that make performing cryptographic operations costly. In such cases, verification can be performed by another device that is trusted to perform this verification. The verified update would then be sent over a secure channel to the consumer IoT device. Performing verification of updates at a hub and then on the consumer IoT device, can reduce the risk of compromise.

It is good practice for a consumer IoT device to act upon the detection of an invalid and potentially malicious update. Beyond rejecting the update, and without limitation, it can report the incident to an appropriate service and/or inform the user. In addition, mitigating controls can be put in place to prevent an attacker from bypassing or misusing an update mechanism. Giving the attacker as little information as possible as part of the update mechanism reduces their ability to exploit it.

EXAMPLE 10: When a consumer IoT device detects that an update could not be delivered or applied successfully (by failing integrity or authentication checks), the consumer IoT device can mitigate information leakage by not providing any information about the failure to the initiator of the update process. However, the consumer IoT device can generate a log entry and deliver notification of the log entry to a trusted peer (e.g. a device administrator) over a secure channel, so that the occurrence of the incident is known and the owner or administrator of the consumer IoT device can make an appropriate response.

5.1.13. Principle 5.3-11

The user should be informed in a recognizable and apparent manner that a security update is required together with information on the risks mitigated by that update.

EXAMPLE 11: The manufacturer informs the user that an update is required via a notification on a user interface or via an email

5.1.14. Principle 5.3-12

The user should be notified when the application of a software update will disrupt the basic functioning of the consumer IoT device.

This notification can include extra detail, such as the approximate expected duration for which the consumer IoT device will be offline.

EXAMPLE 12: A notification includes information about the urgency and approximate expected duration of downtime.

It can be critical for users that a consumer IoT device continues to operate during an update. This is why the provision above recommends to notify the user when an update will disrupt functionality where possible. Particularly, consumer IoT devices that fulfil a safety-relevant function are expected not to turn completely off in the case of an update; some minimal system functional capability is expected. Disruption to functionality can become a critical safety issue for some types of consumer IoT devices and systems if not considered or managed correctly.

EXAMPLE 13: During an update, a watch will continue to display the time, a home thermostat will continue to maintain a reasonable temperature and a Smart Lock will continue to lock and unlock a door.

5.1.15. Principle 5.3-13

The manufacturer shall publish, in an accessible way that is clear and transparent to the user, the defined support period.

When purchasing a product and throughout its lifecycle, the consumer expects this period of software update support to be made clear, e.g. publicize on the official website, etc.

5.1.16. Principle 5.3-14

For consumer IoT devices that cannot have their software updated, the rationale for the absence of software updates, the period and method of hardware replacement support should be published by the manufacturer in an accessible way that is clear and transparent to the user.

5.1.17. Principle 5.3-15A

For devices that cannot have their software updated, the consumer IoT device should be isolable.

5.1.18. Principle 5.3-15B

For devices that cannot have their software updated, the consumer IoT device hardware should be replaceable.

NOTE 9: Hardware is considered as replaceable when there are defined procedures to replace the hardware or hardware components esp. when critical vulnerabilities are discovered.

There are some situations where consumer IoT devices cannot be patched. For such consumer IoT devices a replacement plan needs to be in place and be clearly communicated to the consumer. This plan would typically detail a schedule for when technologies will need to be replaced and, where applicable, when support for hardware and software ends.

5.1.19. Principle 5.3-16

The model designation of the consumer IoT device shall be clearly recognizable, either by labelling on the consumer IoT device or via a user interface.

This is often performed by communicating with a consumer IoT device over a logical interface.

EXAMPLE 14: A consumer IoT device has a HTTP (or HTTPS when appropriate) API that reports the model designation (after user authentication).

Knowledge of the specific designation of the consumer IoT device is often required to check the defined support period of software updates or the availability of software updates.

Being able to identify devices can assist with update management. Where devices can be offline, or updates fail, tracking the update status of user devices through the associated services can assist the user in maintaining device health across all the devices they own. It is recommended that the IoT device is to be clearly recognizable by having both a model designation and a unique privacy preserving per device identifier, either by labelling on the device or via a physical and logical interface. A unique privacy preserving per device identifier could be generated by the manufacturer or could be an identifier the end user sets. and may not be accessible to anyone but the user of the device and associated services. To preserve privacy, the availability and usage of the unique identifier needs to prevent remote services unauthorized parties from either identifying or tracking the device or its users. Being able to identify devices can assist with update management.

EXAMPLE 15: A consumer IoT device is identified in a user update dashboard by a user-chosen identifier ("KitchenUnit") that describes the device's location in a way that is unique to the user, although unlikely to be unique globally, and would allow the user to identify the device from the name. This identifier is deleted when the device is reset to factory settings.

EXAMPLE 16: A consumer IoT device is identified to the update service by a serial number set by the manufacturer only. As is the only identifying information that the update service receives, there is no method by which the transfer of the device between users can be tracked or the identity of the user of the device known and tracked. This serial number is not re-used as a device identifier for any other purpose.

5.2. EN 319 401

Link: https://www.etsi.org/deliver/etsi_en/319400_319499/319401/03.02.00_20/en_319401v030200a.pdf

Link: https://www.etsi.org/deliver/etsi_en/319400_319499/319401/03.02.00_20/en_319401v030200a.pdf

5.2.1. REQ-7.7-26

> The TSP may choose not to apply security patches when the disadvantages of applying the security patches outweigh the cybersecurity benefits. The TSP shall duly document and substantiate the reasons for any such decision.

5.2.2. REQ-7.9.2-09

> The TSP shall address any critical vulnerability not previously addressed by the TSP, within a period of 48 hours after its discovery.

5.2.3. REQ-7.9.5-01

> The TSP shall keep itself informed about technical vulnerabilities of all information systems it uses.

5.2.4. REQ-7.14.2-05

> TSP shall request that ICT products suppliers provide information describing the software components used in products.

5.3. TS 103 732

Link: https://www.etsi.org/deliver/etsi_ts/103700_103799/10373204/01.01.01_60/ts_10373204v010101p.pdf

Link: https://www.etsi.org/deliver/etsi_ts/103700_103799/10373204/01.01.01_60/ts_10373204v010101p.pdf

5.3.1. O.OLD_APP_WARNING

> The TOE shall provide the user with notifications when uninstalling updates from a preloaded application contained in the system software when an older version of the preloaded application is used.

5.3.2. FAP_LFC.1

> Preloaded applications are required to be updated to ensure flaws are remediated.
> FAP_LFC.1.1 The TOE Security Functionality (TSF) shall ensure that preloaded applications are updated by [selection: using an Application Distribution Platform (ADP) and system software updates (to the version maintained in firmware), only by system software updates].

5.3.3. FAP_LFC.2

A preloaded application that is uninstalled will revert to the original version of the application contained in the system software. The user will be warned about using the original version and may have additional options or notices about using the application.
FAP_LFC.2.1 The TSF shall warn the user when a preloaded application update is uninstalled and the application will revert to the version in the system software, and allow the following actions: [selection: disable the application (where possible), warn the user on the next use, none].

5.3.4. FAP_LFC.3

Preinstalled applications shall be updated from trusted ADPs.
FAP_LFC.3.1 The TSF shall ensure that preinstalled applications are only updated from the ADP(s) of the TOE manufacturer and/or OS developer.

5.3.5. FAP_LFC.4

Preinstalled applications downloaded as part of the initial Consumer Mobile Device (CMD) setup process shall be downloaded from trusted ADPs.
FAP_LFC.4.1 The TSF shall ensure that preinstalled applications are only downloaded from the ADP(s) of the TOE manufacturer and/or OS developer.

5.3.6. FAP_RSK.9

To establish applications are updated properly (from the correct source), valid signatures for the platform shall be provided with the installation package.
FAP_RSK.9.1 The applications on the TOE shall be signed with certificates in a signature format valid for the platform.

6. NIAP

6.1. NIAP Profile Protection

Link: https://www.niap-ccevs.org/static_html/protection-profile/516/PP_APP_V2.0.htm

Link: https://www.niap-ccevs.org/static_html/protection-profile/516/PP_APP_V2.0.htm

6.1.1. FPT_LIB_EXT.1 Use of Third Party Libraries FPT_LIB_EXT.1.1

The application shall be packaged with only [assignment: list of third-party libraries].

6.1.2. FPT_TUD_EXT.1 Support for Trusted Updates FPT_TUD_EXT.1.1

The application shall [selection: provide the ability, use platform-provided services] to check for updates and patches to the application software.

6.1.3. FPT_TUD_EXT.1.2

The application shall [selection: provide the ability, use platform-provided services] to query the current version of the application software.

6.1.4. FPT_TUD_EXT.1.3

The application shall [selection:

perform trusted updates
not download, modify, replace or update its own binary code

].

6.1.5. FPT_TUD_EXT.1.4

Application updates shall be digitally signed such that the application platform can cryptographically verify them prior to installation.

7. App Defence Alliance

7.1. Application Security Assessment

Link: https://github.com/appdefensealliance/ASA-WG/blob/main/README.md

Link: https://github.com/appdefensealliance/ASA-WG/blob/main/README.md

7.1.1. Mobile 1.6.1.1 (Android)

The app shall set the targetSdkVersion to an up-to-date platform version.

7.1.2. Mobile 1.6.2.1 (Android)

The app only uses software components without known vulnerabilities.

7.1.3. Web 6.1.1

The app only uses software components without known exploitable vulnerabilities.

7.1.4. Cloud 1.1.1

Azure: Ensure that Only Approved Extensions Are Installed.

7.1.5. Cloud 1.2.1

AWS: Ensure that all AWS Lambda functions are configured to use a current (not deprecated) runtime.

7.1.6. Cloud 1.2.2

Azure: Ensure That 'PHP version' is the Latest, If Used to Run the Web App.

7.1.7. Cloud 1.2.3

Azure: Ensure that 'Python version' is the Latest Stable Version, if Used to Run the Web App.

7.1.8. Cloud 1.2.4

Azure: Ensure that 'Java version' is the latest, if used to run the Web App.

7.1.9. Cloud 1.2.5

Azure: Ensure that 'HTTP Version' is the Latest, if Used to Run the Web App.

7.1.10. Cloud 1.2.6

Google: Ensure that all GCP Cloud functions are configured to use a current (not deprecated) runtime.

7.1.11. Cloud 1.3.2

> Azure: Ensure Web App is using the latest version of TLS encryption.

7.1.12. Cloud 3.7.1

> Azure: Ensure that Microsoft Defender Recommendation for 'Apply system updates' status is 'Completed'.

7.1.13. Cloud 6.12.1

> AWS: Ensure Auto Minor Version Upgrade feature is Enabled for RDS Instances.

8. Apple

8.1. Apple Developer Security

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

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

8.1.1. 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.

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

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.

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.

8.1.3. 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.