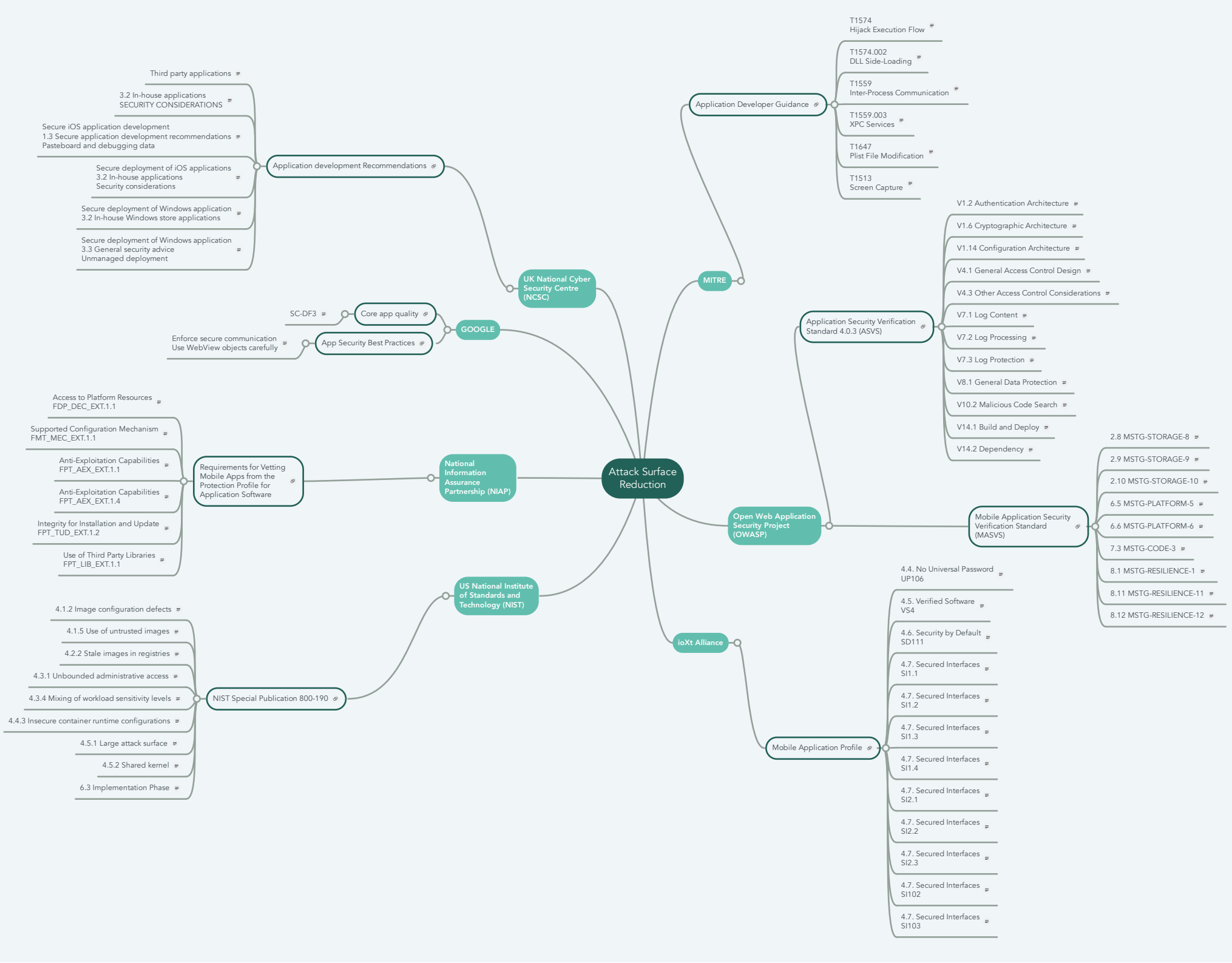
Attack Surface Reduction
UK National Cyber Security Centre (NCSC)
Third party applications
3.2 In-house applications
SECURITY CONSIDERATIONS
Secure iOS application development
1.3 Secure application development recommendations
Pasteboard and debugging data
Secure deployment of iOS applications
3.2 In-house applications
Security considerations
Secure deployment of Windows application
3.2 In-house Windows store applications
Secure deployment of Windows application
3.3 General security advice
Unmanaged deployment
Application development Recommendations
GOOGLE
SC-DF3
Core app quality
Enforce secure communication
Use WebView objects carefully
App Security Best Practices
National Information Assurance Partnership (NIAP)
Access to Platform Resources
FDP_DEC_EXT.1.1
Supported Configuration Mechanism
FMT_MEC_EXT.1.1
Anti-Exploitation Capabilities
FPT_AEX_EXT.1.1
Anti-Exploitation Capabilities
FPT_AEX_EXT.1.4
Integrity for Installation and Update
FPT_TUD_EXT.1.2
Use of Third Party Libraries
FPT_LIB_EXT.1.1
Requirements for Vetting Mobile Apps from the Protection Profile for Application Software
US National Institute of Standards and Technology (NIST)
4.1.2 Image configuration defects
4.1.5 Use of untrusted images
4.2.2 Stale images in registries
4.3.1 Unbounded administrative access
4.3.4 Mixing of workload sensitivity levels
4.4.3 Insecure container runtime configurations
4.5.1 Large attack surface
4.5.2 Shared kernel
6.3 Implementation Phase
NIST Special Publication 800-190
MITRE
Application Developer Guidance
T1574
Hijack Execution Flow
T1574.002
DLL Side-Loading
T1559
Inter-Process Communication
T1559.003
XPC Services
T1647
Plist File Modification
T1513
Screen Capture
Open Web Application Security Project (OWASP)
Application Security Verification Standard 4.0.3 (ASVS)
V1.2 Authentication Architecture
V1.6 Cryptographic Architecture
V1.14 Configuration Architecture
V4.1 General Access Control Design
V4.3 Other Access Control Considerations
V7.1 Log Content
V7.2 Log Processing
V7.3 Log Protection
V8.1 General Data Protection
V10.2 Malicious Code Search
V14.1 Build and Deploy
V14.2 Dependency
Mobile Application Security Verification Standard (MASVS)
2.8 MSTG-STORAGE-8
2.9 MSTG-STORAGE-9
2.10 MSTG-STORAGE-10
6.5 MSTG-PLATFORM-5
6.6 MSTG-PLATFORM-6
7.3 MSTG-CODE-3
8.1 MSTG-RESILIENCE-1
8.11 MSTG-RESILIENCE-11
8.12 MSTG-RESILIENCE-12
ioXt Alliance
4.4. No Universal Password
UP106
4.5. Verified Software
VS4
4.6. Security by Default
SD111
4.7. Secured Interfaces
SI1.1
4.7. Secured Interfaces
SI1.2
4.7. Secured Interfaces
SI1.3
4.7. Secured Interfaces
SI1.4
4.7. Secured Interfaces
SI2.1
4.7. Secured Interfaces
SI2.2
4.7. Secured Interfaces
SI2.3
4.7. Secured Interfaces
SI102
4.7. Secured Interfaces
SI103
Mobile Application Profile

Attack Surface Reduction

1. MITRE

1.1. Application Developer Guidance

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

1.1.1. T1574 Hijack Execution Flow

> When possible, include hash values in manifest files to help prevent side-loading of malicious libraries.

1.1.2. T1574.002 DLL Side-Loading

> When possible, include hash values in manifest files to help prevent side-loading of malicious libraries.

1.1.3. T1559 Inter-Process Communication

> Enable the Hardened Runtime capability when developing applications. Do not include the com.apple.security.get-task-allow entitlement with the value set to any variation of true.

1.1.4. T1559.003 XPC Services

> Enable the Hardened Runtime capability when developing applications. Do not include the com.apple.security.get-task-allow entitlement with the value set to any variation of true.

1.1.5. T1647 Plist File Modification

> Ensure applications are using Apple's developer guidance which enables hardened runtime.

1.1.6. T1513 Screen Capture

> Application developers can apply the FLAG_SECURE property to sensitive screens within their apps to make it more difficult for the screen contents to be captured.

2. Open Web Application Security Project (OWASP)

2.1. Mobile Application Security Verification Standard (MASVS)

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

2.1.1. 2.8 MSTG-STORAGE-8

> No sensitive data is included in backups generated by the mobile operating system.

2.1.2. 2.9 MSTG-STORAGE-9

> The app removes sensitive data from views when moved to the background.

2.1.3. 2.10 MSTG-STORAGE-10

> The app does not hold sensitive data in memory longer than necessary, and memory is cleared explicitly after use.

2.1.4. 6.5 MSTG-PLATFORM-5

> JavaScript is disabled in WebViews unless explicitly required.

2.1.5. 6.6 MSTG-PLATFORM-6

> WebViews are configured to allow only the minimum set of protocol handlers required (ideally, only https is supported). Potentially dangerous handlers, such as file, tel and app-id, are disabled.

2.1.6. 7.3 MSTG-CODE-3

> Debugging symbols have been removed from native binaries.

2.1.7. 8.1 MSTG-RESILIENCE-1

> The app detects, and responds to, the presence of a rooted or jailbroken device either by alerting the user or terminating the app.

2.1.8. 8.11 MSTG-RESILIENCE-11

All executable files and libraries belonging to the app are either encrypted on the file level and/or important code and data segments inside the executables are encrypted or packed. Trivial static analysis does not reveal important code or data.

2.1.9. 8.12 MSTG-RESILIENCE-12

If the goal of obfuscation is to protect sensitive computations, an obfuscation scheme is used that is both appropriate for the particular task and robust against manual and automated de-obfuscation methods, considering currently published research. The effectiveness of the obfuscation scheme must be verified through manual testing. Note that hardware-based isolation features are preferred over obfuscation whenever possible.

2.2. 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

2.2.1. V1.2 Authentication Architecture

1.2.2 Verify that communications between application components, including APIs, middleware and data layers, are authenticated. Components should have the least necessary privileges needed.

2.2.2. V1.6 Cryptographic Architecture

1.6.4 Verify that the architecture treats client-side secrets--such as symmetric keys, passwords, or API tokens--as insecure and never uses them to protect or access sensitive data.

V1.14 Configuration Architecture

2.2.3. V1.14 Configuration Architecture

1.14.1 Verify the segregation of components of differing trust levels through well-defined security controls, firewall rules, API gateways, reverse proxies, cloud-based security groups, or similar mechanisms.

1.14.6 Verify the application does not use unsupported, insecure, or deprecated client-side technologies such as NSAPI plugins, Flash, Shockwave, ActiveX, Silverlight, NACL, or client-side Java applets.

2.2.4. V4.1 General Access Control Design

4.1.3 Verify that the principle of least privilege exists - users should only be able to access functions, data files, URLs, controllers, services, and other resources, for which they possess specific authorization. This implies protection against spoofing and elevation of privilege.

2.2.5. V4.3 Other Access Control Considerations

4.3.2 Verify that directory browsing is disabled unless deliberately desired. Additionally, applications should not allow discovery or disclosure of file or directory metadata, such as Thumbs.db, .DS_Store, .git or .svn folders.

2.2.6. V7.1 Log Content

7.1.1 Verify that the application does not log credentials or payment details. Session tokens should only be stored in logs in an irreversible, hashed form.

7.1.2 Verify that the application does not log other sensitive data as defined under local privacy laws or relevant security policy.

7.1.3 Verify that the application logs security relevant events including successful and failed authentication events, access control failures, deserialization failures and input validation failures.

7.1.4 Verify that each log event includes necessary information that would allow for a detailed investigation of the timeline when an event happens.

2.2.7. V7.2 Log Processing

7.2.1 Verify that all authentication decisions are logged, without storing sensitive session tokens or passwords. This should include requests with relevant metadata needed for security investigations.

2.2.8. V7.3 Log Protection

7.3.1 Verify that all logging components appropriately encode data to prevent log injection.

7.3.3 Verify that security logs are protected from unauthorized access and modification.

2.2.9. V8.1 General Data Protection

8.1.3 Verify the application minimizes the number of parameters in a request, such as hidden fields, Ajax variables, cookies and header values.V10.2 Malicious Code Search

2.2.10. V10.2 Malicious Code Search

10.2.3 Verify that the application source code and third party libraries do not contain back doors, such as hard-coded or additional undocumented accounts or keys, code obfuscation, undocumented binary blobs, rootkits, or anti-debugging, insecure debugging features, or otherwise out of date, insecure, or hidden functionality that could be used maliciously if discovered.

2.2.11. V14.1 Build and Deploy

14.1.5 Verify that authorized administrators can verify the integrity of all security-relevant configurations to detect tampering.

2.2.12. V14.2 Dependency

14.2.2 Verify that all unneeded features, documentation, sample applications and configurations are removed.

14.2.6 Verify that the attack surface is reduced by sandboxing or encapsulating third party libraries to expose only the required behaviour into the application.

3. ioXt Alliance

3.1. Mobile Application Profile

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

3.1.1. 4.4. No Universal Password UP106

Detect and throttle guessing attacks.

3.1.2. 4.5. Verified Software VS4

Anti-rollback.

3.1.3. 4.6. Security by Default SD111

No sensitive data is logged.

3.1.4. 4.7. Secured Interfaces SI1.1

Remote Attack: All certifiable protocols used on the interfaces contained in the device shall be Certified

3.1.5. 4.7. Secured Interfaces SI1.2

Remote Attack: Unused Services are disabled

3.1.6. 4.7. Secured Interfaces SI1.3

Remote Attack: Authentication

3.1.7. 4.7. Secured Interfaces SI1.4

Remote Attack: Secured Communications

3.1.8. 4.7. Secured Interfaces SI2.1

Proximity Attack: Unused Services are disabled

3.1.9. 4.7. Secured Interfaces SI2.2

Proximity Attack: Authentication

3.1.10. 4.7. Secured Interfaces SI2.3

Proximity Attack: Secured Communications

3.1.11. 4.7. Secured Interfaces SI102

Limit access to IPC and sanitize data received by exported handlers.

3.1.12. 4.7. Secured Interfaces SI103

> Endpoints do not expose unnecessary open services and are secured against any medium+ vulnerabilities.

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

4.1. NIST Special Publication 800-190

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

4.1.1. 4.1.2 Image configuration defects

> Organizations should adopt tools and processes to validate and enforce compliance with secure configuration best practices. For example, images should be configured to run as non-privileged users. Tools and processes that should be adopted include:
> 1. Validation of image configuration settings, including vendor recommendations and thirdparty best practices.
> 2. Ongoing, continuously updated, centralized reporting and monitoring of image compliance state to identify weaknesses and risks at the organizational level.
> 3. Enforcement of compliance requirements by optionally preventing the running of noncompliant images.
> 4. Use of base layers from trusted sources only, frequent updates of base layers, and selection of base layers from minimalistic technologies like Alpine Linux and Windows Nano Server to reduce attack surface areas.
>
> A final recommendation for image configuration is that SSH and other remote administratio tools designed to provide remote shells to hosts should never be enabled within containers. Containers should be run in an immutable manner to derive the greatest security benefit from their use. Enabling remote access to them via these tools implies a degree of change that violates this principle and exposes them to greater risk of network-based attack. Instead, all remote management of containers should be done through the container runtime APIs, which may be accessed via orchestration tools, or by creating remote shell sessions to the host on which the container is running

4.1.2. 4.1.5 Use of untrusted images

Organizations should maintain a set of trusted images and registries and ensure that only images from this set are allowed to run in their environment, thus mitigating the risk of untrusted or malicious components being deployed.
To mitigate these risks, organizations should take a multilayered approach that includes:
• Capability to centrally control exactly what images and registries are trusted in their environment;
• Discrete identification of each image by cryptographic signature, using a NIST-validated implementation;
• Enforcement to ensure that all hosts in the environment only run images from these approved lists;
• Validation of image signatures before image execution to ensure images are from trusted sources and have not been tampered with; and
• Ongoing monitoring and maintenance of these repositories to ensure images within them are maintained and updated as vulnerabilities and configuration requirements change.

4.1.3. 4.2.2 Stale images in registries

The risk of using stale images can be mitigated through two primary methods. First, organizations can prune registries of unsafe, vulnerable images that should no longer be used. This process can be automated based on time triggers and labels associated with images. Second, operational practices should emphasize accessing images using immutable names that specify discrete versions of images to be used. For example, rather than configuring a deployment job to use the image called my-app, configure it to deploy specific versions of the image, such as my-app:2.3 and my-app:2.4 to ensure that specific, known good instances of images are deployed as part of each job.

Another option is using a “latest” tag for images and referencing this tag in deployment automation. However, because this tag is only a label attached to the image and not a guarantee of freshness, organizations should be cautious to not overly trust it. Regardless of whether an organization chooses to use discrete names or to use a “latest” tag, it is critical that processes be put in place to ensure that either the automation is using the most recent unique name or the images tagged “latest” actually do represent the most up-to-date versions.

4.1.4. 4.3.1 Unbounded administrative access

Especially because of their wide-ranging span of control, orchestrators should use a least privilege access model in which users are only granted the ability to perform the specific actions on the specific hosts, containers, and images their job roles require. For example, members of the test team should only be given access to the images used in testing and the hosts used for running them, and should only be able to manipulate the containers they created. Test team members should have limited or no access to containers used in production.

4.1.5. 4.3.4 Mixing of workload sensitivity levels

Orchestrators should be configured to isolate deployments to specific sets of hosts by sensitivity levels. The particular approach for implementing this varies depending on the orchestrator in use, but the general model is to define rules that prevent high sensitivity workloads from being placed on the same host as those running lower sensitivity workloads. This can be accomplished through the use of host 'pinning' within the orchestrator or even simply by having separate, individually managed clusters for each sensitivity level.

While most container runtime environments do an effective job of isolating containers from each other and from the host OS, in some cases it may be an unnecessary risk to run apps of different sensitivity levels together on the same host OS. Segmenting containers by purpose, sensitivity, and threat posture provides additional defense in depth. Concepts such as application tiering and network and host segmentation should be taken into consideration when planning app deployments. For example, suppose a host is running containers for both a financial database and a public-facing blog. While normally the container runtime will effectively isolate these environments from each other, there is also a shared responsibility amongst the DevOps teams for each app to operate them securely and eliminate unnecessary risk. If the blog app were to be compromised by an attacker, there would be far fewer layers of defense to protect the database if the two apps are running on the same host.

Thus, a best practice is to group containers together by relative sensitivity and to ensure that a given host kernel only runs containers of a single sensitivity level. This segmentation may be provided by using multiple physical servers, but modern hypervisors also provide strong enough isolation to effectively mitigate these risks. From the previous example, this may mean that the organization has two sensitivity levels for their containers. One is for financial apps and the database is included in that group. The other is for web apps and the blog is included in that group. The organization would then have two pools of VMs that would each host containers of a single severity level. For example, the host called vm-financial may host the containers running the financial database as well as the tax reporting software, while a host called vm-web may host the blog and the public website.

By segmenting containers in this manner, it will be much more difficult for an attacker who compromises one of the segments to expand that compromise to other segments. An attacker who compromises a single server would have limited capabilities to perform reconnaissance and attacks on other containers of a similar sensitivity level and not have any additional access beyond it. This approach also ensures that any residual data, such as caches or local volumes mounted for temp files, stays within the data's security zone. From the previous example, this zoning would ensure that any financial data cached locally and residually after container termination would never be available on a host running an app at a lower sensitivity level.

In larger-scale environments with hundreds of hosts and thousands of containers, this segmentation must be automated to be practical to operationalize. Fortunately, common orchestration platforms typically include some notion of being able to group apps together, and container security tools can use attributes like container names and labels to enforce security policies across them. In these environments, additional layers of defense in depth beyond simple host isolation may also leverage this segmentation. For example, an organization may implement

separate hosting zones or networks to not only isolate these containers within hypervisors but also to isolate their network traffic more discretely such that traffic for apps of one sensitivity level is separate from that of other sensitivity levels.

4.1.6. 4.4.3 Insecure container runtime configurations

Organizations should automate compliance with container runtime configuration standards. Documented technical implementation guidance, such as the Center for Internet Security Docker Benchmark [20], provides details on options and recommended settings, but operationalizing this guidance depends on automation. Organizations can use a variety of tools to “scan” and assess their compliance at a point in time, but such approaches do not scale. Instead, organizations should use tools or processes that continuously assess configuration settings across the environment and actively enforce them.

Additionally, mandatory access control (MAC) technologies like SELinux [21] and AppArmor [22] provide enhanced control and isolation for containers running Linux OSs. For example, these technologies can be used to provide additional segmentation and assurance that containers should only be able to access specific file paths, processes, and network sockets, further constraining the ability of even a compromised container to impact the host or other containers. MAC technologies provide protection at the host OS layer, ensuring that only specific files, paths, and processes are accessible to containerized apps. Organizations are encouraged to use the MAC technologies provided by their host OSs in all container deployments.

Secure computing (seccomp)7 profiles are another mechanism that can be used to constrain the system-level capabilities containers are allocated at runtime. Common container runtimes like Docker include default seccomp profiles that drop system calls that are unsafe and typically unnecessary for container operation. Additionally, custom profiles can be created and passed to container runtimes to further limit their capabilities. At a minimum, organizations should ensure that containers are run with the default profiles provided by their runtime and should consider using additional profiles for high-risk apps.

4.1.7. 4.5.1 Large attack surface

For organizations using container-specific OSs, the threats are typically more minimal to start with since the OSs are specifically designed to host containers and have other services and functionality disabled. Further, because these optimized OSs are designed specifically for hosting containers, they typically feature read-only file systems and employ other hardening practices by default. Whenever possible, organizations should use these minimalistic OSs to reduce their attack surfaces and mitigate the typical risks and hardening activities associated with general-purpose OSs.

Organizations that cannot use a container-specific OS should follow the guidance in NIST SP 800-123, Guide to General Server Security [23] to reduce the attack surface of their hosts as much as possible. For example, hosts that run containers should only run containers and not run other apps, like a web server or database, outside of containers. The host OS should not run unnecessary system services, such as a print spooler, that increase its attack and patching surface areas. Finally, hosts should be continuously scanned for vulnerabilities and updates applied quickly, not just to the container runtime but also to lower-level components such as the kernel that containers rely upon for secure, compartmentalized operation.

4.1.8. 4.5.2 Shared kernel

In addition to grouping container workloads onto hosts by sensitivity level, organizations should not mix containerized and non-containerized workloads on the same host instance. For example, if a host is running a web server container, it should not also run a web server (or any other app) as a regularly installed component directly within the host OS. Keeping containerized workloads isolated to container-specific hosts makes it simpler and safer to apply countermeasures and defenses that are optimized for protecting containers.

4.1.9. 6.3 Implementation Phase

After the container technology has been designed, the next step is to implement and test a prototype of the design before putting the solution into production. Be aware that container technologies do not offer the types of introspection capabilities that VM technologies do.
NIST SP 800-125 [1] cites several aspects of virtualization technologies that should be evaluated before production deployment, including authentication, connectivity and networking, app functionality, management, performance, and the security of the technology itself. In addition to those, it is important to also evaluate the container technology's isolation capabilities. Ensure that processes within the container can access all resources they are permitted to and cannot view or access any other resources.
Implementation may require new security tools that are specifically focused on containers and cloud-native apps and that have visibility into their operations that traditional tools lack. Finally, deployment may also include altering the configuration of existing security controls and technologies, such as security event logging, network management, code repositories, and authentication servers both to work with containers directly and to integrate with these new container security tools.
When the prototype evaluation has been completed and the container technology is ready for production usage, containers should initially be used for a small number of apps. Problems that occur are likely to affect multiple apps, so it is helpful to identify these problems early on so they can be addressed before further deployment. A phased deployment also provides time for developers and IT staff (e.g., system administrators, help desk) to be trained on its usage and support.

5. National Information Assurance Partnership (NIAP)

5.1. 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

5.1.1. Access to Platform Resources FDP_DEC_EXT.1.1

The application shall restrict its access to [selection:
no hardware resources,

network connectivity,

camera,

microphone,

location services,

NFC,

USB,

Bluetooth,

[assignment: list of additional hardware resources]

] .
Application Note: The intent is for the evaluator to ensure that the selection captures all hardware resources which the application accesses, and that these are restricted to those which are justified. On some platforms, the application must explicitly solicit permission in order to access hardware resources. Seeking such permissions, even if the application does not later make use of the hardware resource, should still be considered access. Selections should be expressed in a manner consistent with how the application expresses its access needs to the underlying platform. For example, the platform may provide location services which implies the potential use of a variety of hardware resources (e.g. satellite receivers, WiFi, cellular radio) yet location services is the proper selection. This is because use of these resources can be inferred, but also because the actual usage may vary based on the particular platform. Resources that do not need to be explicitly identified are those which are ordinarily used by any application such as central processing units, main memory, displays, input devices (e.g. keyboards, mice), and persistent storage devices provided by the platform.

5.1.2. Supported Configuration Mechanism FMT_MEC_EXT.1.1

The application shall invoke the mechanisms recommended by the platform vendor for storing and setting configuration options.
Application Note: Configuration options that are stored remotely are not subject to this requirement.

5.1.3. Anti-Exploitation Capabilities FPT_AEX_EXT.1.1

The application shall not request to map memory at an explicit address except for [assignment: list of explicit exceptions] .
Application Note: Requesting a memory mapping at an explicit address subverts address space layout randomization (ASLR).

5.1.4. Anti-Exploitation Capabilities FPT_AEX_EXT.1.4

The application shall not write user-modifiable files to directories that contain executable files unless explicitly directed by the user to do so.
Application Note: Executables and user-modifiable files may not share the same parent directory, but may share directories above the parent.

5.1.5. Integrity for Installation and Update FPT_TUD_EXT.1.2

The application shall be distributed using the format of the platform-supported package manager.

5.1.6. Use of Third Party Libraries FPT_LIB_EXT.1.1

The application shall be packaged with only [assignment: list of third-party libraries] .
Application Note: The intention of this requirement is for the evaluator to discover and document whether the application is including unnecessary or unexpected third-party libraries. This includes adware libraries which could present a privacy threat, as well as ensuring documentation of such libraries in case vulnerabilities are later discovered.

6. GOOGLE

6.1. Core app quality

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

6.1.1. SC-DF3

The app does not use any non-resettable hardware IDs, such as the IMEI, for identification purposes.

6.2. App Security Best Practices

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

6.2.1. Enforce secure communication Use WebView objects carefully

> Whenever possible, load only allowlisted content in WebView objects. In other words, the WebView objects in your app shouldn't allow users to navigate to sites that are outside of your control.
>
> In addition, you should never enable JavaScript interface support unless you completely control and trust the content in your app's WebView objects.
>
> Use HTML message channels
> If your app must use JavaScript interface support on devices running Android 6.0 (API level 23) and higher, use HTML message channels instead of communicating between a website and your app, as shown in the following code snippet:

7. UK National Cyber Security Centre (NCSC)

7.1. Application development Recommendations

Link: https://www.ncsc.gov.uk/collection/application-development/generic-application-development

7.1.1. Third party applications

You might be considering deploying third party applications on the same device as applications that handle sensitive data. The primary concerns here are:

protecting the enterprise network infrastructure from attack via the third party applications
preventing data from leaking from a sensitive datastore into a third party application
Modern platforms have built-in support for segregation of applications and users, which you should use wherever possible.

As the behaviour of third party applications cannot normally be modified, protection has to be provided elsewhere, via network protections and appropriate use of data-stores. Where possible, the developers of any third party applications should be approached in order to gain a deeper understanding of their product.

Applications that are likely to:

be sources of data leaks
track user movements
interfere with other applications
- should not be installed. If software must be used despite security concerns that cannot be mitigated with technical controls, users should be given training on how best to manage the risk.

7.1.2. 3.2 In-house applications SECURITY CONSIDERATIONS

Regardless of whether the application is developed by an internal development team, or under contract by an external developer, you should ensure that supplied binaries match the version which you were expecting to receive. Applications should then be installed onto managed devices through an MDM server or in-house enterprise application catalogue front-end, to gain the benefits of an application being enterprise managed.

7.1.3. Secure iOS application development 1.3 Secure application development recommendations Pasteboard and debugging data

The application must manage the pasteboard effectively by doing one (or more) of the following:

Clear the pasteboard when the application exits or loses focus (crashes may still result in data leakage).
Implement a private pasteboard within your application - do not use the system pasteboard.
Encrypt the pasteboard with a key stored in the Developer's keychain. This also allows pasting between the same developer's applications.
Exclude sensitive information from the Universal Clipboard (Handoff Feature). This can be performed by using the setItems(_:options:) method with the localOnly option within the UIPasteBoard class.
The application must ensure that debugging output has been removed and sensitive information prevented from appearing within the device log files.

Logging APIs such as 'NSLog', 'printf', 'NSAssert', etc. should be reviewed and removed from production builds if sensitive information is being logged using these APIs.
The application should detect and notify a user when screen capture is performed on iOS 11 and take appropriate action. For example, the application could display a warning and/or exit to prevent screen capture of sensitive data. More information about this feature can be found within the API documentation.

7.1.4. Secure deployment of iOS applications 3.2 In-house applications Security considerations

Regardless of whether the application is developed by an internal development team, or under contract by an external developer, you should ensure that supplied binaries match the version which you were expecting to receive if supplied via business-to-business or the App Store. Either way, applications should then be installed onto managed devices through an MDM server or enterprise app catalogue front-end, to gain the benefits of the app being enterprise-managed.

7.1.5. Secure deployment of Windows application 3.2 In-house Windows store applications

In-house applications are those applications which are designed and commissioned by an organisation to fulfil a particular business requirement. The organisation can stipulate the functionality and security requirements of the application, and can enforce these contractually if the development work is subcontracted. For the purposes of this document, these applications are assumed to access, store, and process sensitive data. The intention when securing these applications is to minimise the opportunity for data leakage, and to harden them against physical and network-level attacks.

The following best practice guidelines should be followed when developing applications for use internally:

Consider security concerns throughout the product lifecycle, including the design, development, and ongoing support stages.
Ensure that developers and product owners follow the NCSC secure development and deployment guidance.
Ask the questions listed in the Questions for application developers section.
Ensure that contracted developers deliver source code for the final product

7.1.6. Secure deployment of Windows application 3.3 General security advice Unmanaged deployment

Deployment to unmanaged devices introduces some risk, as it reduces the complexity of attack needed to compromise the software. In this scenario, the following guidelines should be considered:

Minimise storage of sensitive data on the device.
In a client-server model, input from the software should not be trusted by the server unless further authentication (such as described in the Secure Windows Application Development – Authentication section) is supplied to verify and authenticate an actual user.
Obfuscation and similar technologies could be used to increase the effort required to reverse-engineer the software. However, obfuscation should be considered only for this purpose, and should not be relied upon to provide complete protection.