

Vulnerability Scan - Current process and Potential Improvements

Mobile Platform

19 November, 2025

Table of Contents

1	Current Process	7
1.1	Pre-production (during development):.....	7
1.2	Pre-release:.....	7
1.3	Post-release	7
1.4	On request	8
1.5	Reporting (individual app)	8
1.6	Reporting (all apps)	8
2	Gartner Suggestions	9
3	Ostorlab integration and onboarding costs - rough estimate	11
3.1	Questions:.....	11
3.2	Assumptions:	11
3.3	POC Scope:.....	11
3.4	Implementation:	11
3.5	Rough estimates	12
4	Possible Future Process Suggestions.....	13
4.1	Future Process Discussion	13
4.1.1	Possible MobSF Implementation Process Thoughts	15
5	Proof Of Concept - Potential Replacement	17
6	Reduced Cost Future Process Suggestion.....	19
6.1	Aim:.....	19
6.2	Pre-production (during development):.....	19
6.3	Pre-release:.....	20
6.4	Post-release	20
7	SSDLC Baseline And Possible Changes To Open Up Additional Vendors	21
7.1	Summary:	21
7.2	Continuous production monitoring removal mitigation:	21
7.3	Detail:	22
7.4	Current 4.1:.....	22
7.5	Current 6.1:.....	22

7.6	Current 9.1:.....	22
7.7	Current 12.1:.....	23
7.8	Current 15.1:.....	23
7.9	Current 20:.....	24
7.10	Current 21:.....	24
8	Timelines	26
8.1	Sprint Goals	26
9	Tool Comparisons	27
9.1	AppDome.....	33
9.1.1	Summary	33
9.2	Black Duck / Polaris.....	34
9.2.1	Summary	34
9.3	Black Duck Binary Analysis ("BDBA").....	35
9.3.1	Summary	35
9.4	CodeQL - GitHub	35
9.4.1	Summary	35
9.5	Data Theorem.....	35
9.5.1	Summary	35
9.6	Guard Square.....	36
9.6.1	Summary	36
9.7	MobSF (Full).....	36
9.7.1	Summary	36
9.8	MobSFscan (Lightweight source-code only version of MobSF).....	37
9.8.1	Summary	37
9.9	NowSecure.....	112
9.9.1	Summary	112
9.10	Ostorlab	112
9.10.1	Summary	112

- [Current Process \(see page 7\)](#)
- [Gartner Suggestions \(see page 9\)](#)
- [Ostorlab integration and onboarding costs - rough estimate \(see page 11\)](#)
- [Possible Future Process Suggestions \(see page 13\)](#)
 - [Future Process Discussion \(see page 13\)](#)
 - [Possible MobSF Implementation Process Thoughts \(see page 15\)](#)
- [Proof Of Concept - Potential Replacement \(see page 17\)](#)
- [Reduced Cost Future Process Suggestion \(see page 19\)](#)
- [SSDLC Baseline And Possible Changes To Open Up Additional Vendors \(see page 21\)](#)
- [Timelines \(see page 26\)](#)
- [Tool Comparisons \(see page 27\)](#)
 - [AppDome \(see page 33\)](#)
 - [Black Duck / Polaris \(see page 34\)](#)
 - [Black Duck Binary Analysis \("BDBA"\) \(see page 35\)](#)
 - [CodeQL - GitHub \(see page 35\)](#)
 - [Data Theorem \(see page 35\)](#)
 - [Guard Square \(see page 36\)](#)
 - [MobSF \(Full\) \(see page 36\)](#)
 - [MobSFscan \(Lightweight source-code only version of MobSF\) \(see page 37\)](#)
 - [NowSecure \(see page 112\)](#)
 - [Ostorlab \(see page 112\)](#)

Executive Summary:

Costs are high for app scanning services across the board. This is partially driven by the number of applications we have.

Most of our applications do not present/handle sensitive data, however we have a duty to ensure security and resilience for all of our applications for ethical, legal, branding, and user confidence reasons.

Summary of Current landscape:

\$625k pa - the cost of our current service - DataTheorem which offers unlimited apps, plus additional services.

~110 currently published external apps that we publish that available to public (iOS and Android, counting each separately, not including MAHI/BioMark/etc. apps).

~50 currently published internal apps that we publish that are available only to managed devices (iOS mostly).

~260 apps in SecureTheorem (as of 2025/2/26, after pruning ~100 old/unpublished apps, includes iOS and Android, includes development instances).

49% of our applications that we publish we own the source code, 51% of our applications we do not own the source code, however for **all** apps that we publish we would have an app bundle.

In extreme cases there are ~9 different instances of the same app being scanned (different bundle IDs, different environments, etc.)

In some cases there is no difference in features/code/etc. between development and production app instances, in other cases the only difference might be the server that an app contacts, however app teams are frequently told to have different environments by various compliance teams.

Based on review of findings from 2024/10, ~97% of high risk, 98% of medium risk, and ~95% of low risk findings are false positives/non-issues (See [Data Theorem Risks Counts](#)¹ for source data). *This may be unavoidable with automated scans that are not tailored to each application's features/requirements.

On review in 2025/1 ~40% of users registered in Data Theorem had not logged in in over a year (includes users on apps that are no longer published/apps that have not had an update in over a year, etc.).

In 2024 2636 scans were performed on all apps in Data Theorem, this includes periodic re-scans of the same unchanged build, with an average of ~219 scans per month.

In 2024 2189 scans were performed only on pre-production apps in Data Theorem, this includes periodic re-scans of the same unchanged build, with an average of ~182 scans per month on pre-production apps.

Scan Counts Per Month 2024

**NB Includes re-scans of the same build*

Month	Pre-prod Mobile Scans	Production Mobile Scans
01	131	33
02	333	44
03	169	19
04	213	14
05	170	45
06	160	40

¹ <https://share.merck.com/spaces/FPMP/pages/1728030004/Data+Theorem+Risks+Counts>

07	128	28
08	167	71
09	212	50
10	176	22
11	186	35
12	144	46
SUM	2189	447
AVG	182	37

1 Current Process

1.1 Pre-production (during development):

Apps uploaded automatically to Data Theorem on successful build or re-signing from Bitrise. Multiple bundle IDs might be used during this time to meet other requirements.

Out of scope apps uploaded directly to Data Theorem using web UI (apps that are not supported on Bitrise such as Unity apps).

Data Theorem groups apps by (in order of precedence) platform, bundle ID, app name, and app executable name, with latest version in a bundle ID overwriting the previous version (in most cases). Scans are flagged as "pre-prod".

There is no automatic linking of the same app with different bundle IDs etc. this must be done manually, and there are some restrictions with linking/grouping of applications that limit this.

Issues should be reviewed during development by development teams, often reviewed by FPMP during development (on request or on identifying particular need for attention).

Access to scans is by share link (30 days temporary link), or by providing direct access. Process for access management and generating share link is manual, however those with access will receive automatic emails from Data Theorem.

1.2 Pre-release:

On a release candidate being built, is uploaded to Data Theorem for a scan, using the production bundle ID.

This is flagged as pre-prod as above, with same problems, however a review based on known information (about app, about scan, about features, about libraries, about technology) is performed to identify blocking issues.

Information from the vulnerability scan is used to partially complete legal statements required by the stores (e.g. data privacy statements, however it is one input out of several).

1.3 Post-release

On a release being released, is *usually* automatically onboarded onto Data Theorem for regular scanning, as long as it is in one of the known publishing accounts.

Exceptions to automatic onboarding are usually due to releases for countries which Data Theorem doesn't regularly check, in which case manual request to onboard is emailed to Data Theorem's support team.

Other exceptions to automatic onboarding are due to unknown accounts (e.g. MAHI accounts which are operated outside of our company's main team).

It is not automatically grouped with previous scans, so previously closed issues may re-open, meaning attention is needed for administration - reworking previous triages.

On new issues being identified alerts are sent to those associated with an app in Data Theorem's tool, these new issues must be triaged to eliminate false positives.

1.4 On request

For scans of applications that do not pass through FPMP or Bitrise (e.g. proofs of concepts from vendors that are not intended to be published by our company, or not re-signed, or built by us, or legal reviews) then users request a scan via ITRMS AppSec's Jira board, these are then usually handed to FPMP to trigger the scan (although in some cases, through no fault of their own/due to documentation differences, contractors might ignore these instructions and can give different information to requestors). These go through the manual upload process, and due to the way the administration works within DataTheorem, are often left orphaned in DataTheorem's system.

1.5 Reporting (individual app)

As for any given app there could be multiple pre-production scans, as well as potentially one published app scan, where the published app scan does not change *until* an app update is published, identifying and providing the correct app scan to report to app owners and developers is not truly possible to automate. There are APIs that allow us to find a URL to a scan, however whether this is the "correct" scan to provide always needs at least some manual input. Better administration to allow linking to a "group" based on the app would improve this.

1.6 Reporting (all apps)

Due to the number of applications, the number of false positives, and administration issues there are currently too many results - making the overall reporting low quality. Although efforts are made to weed out false positives from individual apps, the volume of apps scanned means that scaling this up is a high effort task.

2 Gartner Suggestions

1. Suggested that any solution we chose would be expensive and that the estimates we had got were about average.
2. Risk based approach suggested as an option to bring down cost.
3. Suggested look in to alternative licensing models - number of scans vs. number of apps vs number of users
4. Suggested a proposal review service so Gartner can compare against vendor proposals that they have worked with in the past
5. Strategy webinar suggested (happening in future).
6. Suggested vendors:
 - a. Appdome
 - i. Appdome only provides protection service (not a scanning solution)
 - ii. Potentially expensive depending on # of apps
 - iii. \$50k to \$150k pa per app (160 apps @ an average of \$75k pa == \$12MM pa)
 - b. Data Theorem
 - i. Our current vendor
 - ii. Cost \$625k pa
 - iii. Supports unlimited apps with no increase in cost
 - c. eShard
 - d. Guardsquare
 - e. ImmuniWeb
 - f. Lookout
 - i. Lookout provides a protection service only (not a scanning solution)
 - g. NowSecure
 - i. Comparative to DataTheorem with a slightly better administrative system
 - ii. Potentially expensive depending on # of apps
 - iii. \$5k pa per app (160 apps @ \$5k pa == \$800k pa)
 - h. Quokka
 - i. Zimperium
7. Provided documents to us for review for suggestions on approaches (attached below).

[illegible]

3 Ostorlab integration and onboarding costs - rough estimate

3.1 Questions:

How scanning on Prod apps works and how is it counted?

What is the repo / source of Prod apps for Ostorlab scanning?

Can the scan results be imported back to MASP? Date, results or minimum URL of results?

Repository of previously scanned apps, but not using the slot this month?

3.2 Assumptions:

API is existing and capable of uploading binaries to scan.

Baseline: All apps one scan prior release.

Public apps scanned periodically in Prod.

Internal apps in Prod - GxPs apps (SMART, eShopFloor) would be scanned periodically if no new version developed.

3.3 POC Scope:

In depth testing - verify feasibility of implementing requirements (better finding management, customization of logic in case volume limit exists,) with low effort.

Confirm this estimate based on detailed assessment

Timing - Q4/2025?

Outcome: business case - comparison of Data Theorem new offer vs Ostorlab offer + implementation costs.

3.4 Implementation:

Define what to scan and frequency

Define project onboarding processes

Define ongoing operation processes - validation of results and issues resolutions process.

Whitelisting per portfolio / per app.

Define rules for licensed volume management (when we can/cannot scan something, what to scan,)

Develop Bitrise private step for both Build and Re-sign workflow: upload and scan (+ wait for results?)

??? Develop MASP integration - minimum: Link to Ostorlab subpage / optimum: updates on latest scan date and high-level result visible in MASP (Red vs Green)

Onboarding

Manual onboarding and setup for 163 production apps + XX apps in lower environments.

3.5 Rough estimates

Optimistic (no customizations): 2-3 weeks x 2FTEs

Onboarding (163 prod apps + 200 lower env's and or retired/duplicates/randoms): 1-4 hours x 200 - 250 apps = 1-2 M x 1 FTE

Pessimist estimate (customization logic with 3-5 parameter input has to be developed and used to automate the scanning): e.g. risk-based approach to decide priority / frequency of scans = 2-3 months x 2 FTE's

4 Possible Future Process Suggestions

NB some of these are not directly achievable with Data Theorem.

Administration improvements

Most projects go through many iterations with different pre-prod IDs, most projects have an Android and iOS side: allow grouping by project and then by platform and bundle ID.

Apps are released, perform their functions, and then are removed (e.g. due to benefit vs. cost of maintenance, user adoption, etc.): allow easier removal to give a clear picture of current risk.

Providing a share link to a group to allow easier relaying of results to app owners/developers (theory is that the app owners would be able to judge for themselves the correct scan to work from).

Separate test scans from production scans → possible different instance of same system but still make it possible to group across systems?

System-wide closure of non-issues was partially implemented recently in Data Theorem in 2025 → most common reported issues are non-issues related to elements such as Firebase: allow system-wide closure of non-issues, or at least custom rules for an app group?

Access improvements → user management is currently by request, external developers intermittently use the system but have historically exhibited non-beneficial behavior (e.g. saying an issue can be closed out when it should be fixed) → currently process is for developers to review issues then provide a response that is also then triaged against what we believe they should fix → find a middle ground

Separate out store requirements vs vulnerabilities → both are important to be discovered, there is some cross-over, and both can block publication, however store requirements are different aspect of development and different risk than vulnerabilities.

Improve approvals → clear green light to release vs. current system of reconnaissance level only which gives a mix? Ideal world would be this approval via API as this could then be used for future automation. Does this go beyond ITRMS AppSec's remit of providing advice but leaving ultimate decision to app owners?

Merge store accounts → makes identifying apps published by us vs. rogue apps a lot easier, makes comparing an inventory in vulnerability scanning vs. reality easier. → already planned to merge all into the MSD LLC iOS and Android publisher accounts, but needing some support from various teams to urge action. Also see [Publisher / Developer Account Mergers](#)²

Of benefit to ITRMS AppSec as well as others - Capture vendor contact details in MASP (at minimum internal vs external, preferably vendor name). Makes it easier for ITRMS (and others) to know who to contact if something is urgent, or if it is something built by us or not.

4.1 Future Process Discussion

Rough Notes From Meeting between Robert Smith Lukas Marsik 2025/8/23

² <https://share.merck.com/spaces/FPMP/pages/1809156761/Publisher+Developer+Account+Mergers>

Related info - <https://github.com/MobSF/Mobile-Security-Framework-MobSF>

1. Current position is DT is too expensive/has many features we don't use, part of expense is the open access to all builds (no app count limits).
 - a. If we're not using something we're paying for then it is wasted money.
 - b. Possible to reduce app counts scanned since currently everything that is in build pipeline is scanned, no matter if a different bundle ID/etc. and everything we publish is also scanned from the stores.
2. Many apps are scanned, only a few are reviewed directly by business/development teams (based on reviews of access logs), many false positive/irrelevant results.
3. False positives are often due to the differences between automated scans and manual vulnerability/pen-testing. Automated scans can't easily react based on app's features/requirements/sensitivity/etc.
 - a. I.e. results always need analysis anyway (see 4.c. and 5.b.)
4. Use MobSF locally during development:
 - a. One of the tools recommended by OWASP
 - b. Developers can run their own scans.
 - c. Scans will still need analysis - they're recon only.
 - d. Bundle ID, signing, etc. doesn't matter as is local only
5. Could just leave up to vendors what tool to use to perform scanning during development
 - a. Would put it on a par with vendor owned/white label apps that we just re-sign and publish
 - b. Mostly code is only scanned by us at set deliverables points - for most of our apps this is immediately before production when they have a release candidate, and due to about 50% of our apps falling into the whitelabel bucket/most of the other 50% being developed by vendors who only supply code at set deliverables points.
6. Use MobSF for final scan:
 - a. Could spin up own server to have a persistent database of scans
 - i. Only ongoing costs would be server cost and maintenance plus advised MobSF enterprise level support.
 1. Possible chance for us to act ethically and support an open source project (good for company reputation).
 - b. Scans will still need analysis - they're recon only.
 - i. <not mentioned in discussion but> Use additional tool for recording actual issues based on recon? Existing tool?
 1. Could use MASP for high level results - depends on the outcome of discussion on Forge/future of MASP.
 2. Could use almost any system really - important part is being able to interpret scans and report what issues need to be fixed. ITRMS Jira board? ServiceNow type tickets? Just provide the report output and highlight what needs to be fixed?
 - c. Could be good practice to build up our internal skill sets - i.e. rather than paying \$650k a year to a vendor spending some of that on building up employee knowledge/skills == a more resilient company.

- d. Would lose some coverage and suggested fixes
 - i. Robert to list what we'd lose on this.
 - e. MobSF supports API access to data out of the box - Could set up API for getting and sharing recon data to other systems?
 - f. Lukas' engineering team to investigate what is possible / level of effort
 - g. Robert to create set of requirements (majority already exist).
7. Remove production monitoring requirement
- a. Is in our SSDLC policy - would be removed if we used a different system (e.g. Ostorlab) anyway.
 - i. Almost nobody is looking at the production scans - the pre-prod scans get most attention.
 - b. Mostly covered by
 - i. Frequent updates and scans of apps before release.
 - ii. Controls on who can actually release an app.
 - 1. Mostly covered already by our team policies.
 - 2. Could improve further (e.g. by moving any UAT/Test/Dev app instances to be from Firebase App Distribution), that way publication == very limited.
 - iii. I haven't seen an incident where published apps had a newly reported issue that wasn't already reported by the developers/vendors/Apple/Google.
 - 1. Not to say it couldn't happen, but is a rare situation, and the data types handled of almost all the apps we publish is not sensitive/critical.
 - 2. Is a good idea safeguard, but costs more.

4.1.1 Possible MobSF Implementation Process Thoughts

- 1. Assuming MobSF server hosted by us exists.
 - a. Suggested two instances - one dev, one prod.
 - i. If is hosted by us then cost difference between 1 and 2 instances (or 2 and 3 if also using a UAT server) could be negligible as is just a matter of environments.
 - b. MobSF is a tool, not a system - it's main purpose is to show what is there in a build for recon purposes, not necessarily what is *not* there but should be, and won't necessarily indicate severity etc. Won't include remediation steps.
- 2. **During Development:**
 - a. API exists for upload to dev MobSF server
 - b. Doesn't matter which bundle ID is used.
 - c. Main use is for developers, multiple iterations will occur as/when devs make changes.
 - d. Possible use for providing feedback during development
 - e. Bitrise uploads to dev MobSF server and link generated for developer to access and review.
 - f. Also can upload manually to same dev server.

- g. Should we use forge for this? Not needed as much simpler to just do from Bitrise as part of build process.

3. **On Release Candidate:**

- a. API exists for upload to prod MobSF server
- b. Prod MobSF server must maintain database of builds scanned.
 - i. Export out somewhere after n days to avoid large data storage?
 - ii. That database must be searchable for ITRMS/FPMP/others to review as/when needed.
- c. Bitrise has production switch to indicate is a production build - puts into production instance of MobSF server.
- d. Bundle ID must match production bundle ID
 - i. Should we create something on top of default MobSF to enable some filtering on bundle ID to identify correct scan?
 - ii. Could be triggered by MASP which would make that filtering easier management-wise
 - 1. That could make the build selection process slightly more complex?
 - 2. Already have started logic for storing production builds in Artifacts.
 - iii. Could use Forge for this, but again is far simpler to just do as part of the build process?
- e. Link is generated for FPMP to access and review
- f. MobSF is recon only, so requires manual review before passing.
 - i. Nice to have - definite check for pass/fail.
 - ii. Status could be handled with MASP logic after manual review? - we already have vulnerability scan status that is largely unused during scan.
 - iii. On passing - set MASP to next appropriate step.
 - iv. On failing - put to pending owner action.

5 Proof Of Concept - Potential Replacement

Work in progress on this page

Test criteria:

1. Integration with build system - Bitrise.
 - a. Ease of implementation - Can be:
 - i. By API call (e.g. a custom script leveraging curl) - this will eventually become a defined step in our modular workflows, can be temporary during testing.
 - ii. By pre-defined Bitrise Step - off-the-peg integrations - <https://bitrise.io/integrations>
 - b. issue tracking - what errors/status are returned from API or the step in the build logs?
 - c. Timing - as build system is paid by minute, and will time-out if a given step is taking too long, then how long does the upload of a large build take using the given tool?
 - d. Handling for white-label resigned and source code built apps should be the same - scan the build not the source.
 - i. Nice to check - handling for naming conventions etc. (i.e. does tool use the app label, the app bundle ID, etc., some cross-over with management / administration element).
2. Quality of results
 - a. Compare results with Data Theorem results for same app, same build.
 - b. Compare results with independent scan and/or knowledge of app and build.
 - i. Easier to perform with candidates that we own source code for.
 - c. Relevancy of results (e.g. does tool have more false positives/un-related information or less)
 - i. Cross-over with management / administration element.
 - d. Quality of suggested fixes (e.g. ease of instructions, relevancy of instructions, etc.)
3. Management / Administration (Partially subjective)
 - a. Ease of onboarding apps (cross-over with build system element)
 - b. Ease of creating / managing rule sets (assigning severity handling to reduce false positives).
 - c. Ease of assigning apps to rule sets (or vice-versa).
 - d. Ease of identifying/ tracking scans (which apps scanned, # of scans/results).
 - e. Ease of adding / managing users

Test candidate apps:

1. At least 3 apps, with known issues, with enough variation in features and architecture that can be classified into different rule sets.
2. At least 1 white-label resigned app and 1 app for which we own the source code (for results comparison)

3. At least 1 Android app and 1 iOS app.
4. At least 1 Kotlin app and 1 Swift app
5. Depending on the tool being tested, we can swap around app tested (as the app bundle ID == the primary identifier) without increasing the scope of given POC access.

Suggestions for test candidates:

1. Known testing challenge builds -
 - a. See <https://mas.owasp.org/crackmes>
2. Our own apps -
 - a. Problem Solving App - legacy internal (ABM) iOS app that we own source code for, in active development/maintenance, has an overly complex structure with older dependencies - <https://masp.merck.com/iOS/d510592d-bb70-4433-a0c5-b23759e59141>
 - b. MSD360 - internal iOS (ABM) app that we don't own source code for, created with Unity, widely used within company for meetings - <https://masp.merck.com/iOS/cba443b5-73de-45c3-90c3-2590199194f3>
 - c. Pet Diabetes Tracker - public iOS and Android apps that we own source code for, created with Swift and Kotlin, used for managing pet health data, in active re-write development - <https://masp.merck.com/iOS/c3389928-75d6-4512-854a-2c12eb17d3c7> and <https://masp.merck.com/Android/8b8184e4-c0eb-428f-95f1-ca7a6521f039>
 - d. Medifides - public iOS and Android apps that we own source code for, used for supply chain verification, created with React Native, actively maintained - <https://masp.merck.com/Android/22d316ca-3646-4e45-95a4-180657134dd6> and <https://masp.merck.com/iOS/ea1abf26-a28b-48f6-a52b-4e285c468fd0>
 - e. Any of the MSD Manuals suite of apps - public iOS and Android apps, the largest number of users both separately and combined, very large applications (ipa/aab size) by design.

6 Reduced Cost Future Process Suggestion

Also see: [SSDLC Baseline And Possible Changes To Open Up Additional Vendors \(see page 21\)](#)

6.1 Aim:

Given that:

- Most costs and organization challenges look to come from the number of apps scanned - an app instance is defined by the bundle ID, so if an app has a production environment, as well as dev, QA, and test environments, that one app can potentially count as 4 separate apps (depending on tool used).
- Additional charges (depending on tool used) come from automatic scraping of published apps - each of those can count as another separate app scanned in addition to the cost of a third party tool having to scrape our public app accounts.
- Administrative cost comes from organization of applications, which due to volume can easily overwhelm any reporting from the tool.

Reduce costs (and/or improve possible tool options) by:

- Reducing number of apps scanned.
- Increasing ease of administration.

6.2 Pre-production (during development):

On development start, assign 1 production and 1 development ID that can be used for scanning to limit the number of apps scanned.

On development start the development team explicitly nominates representative(s) who will be given access to results for those two bundle IDs for reviewing issues to improve the handling of issues detected.

Apps uploaded automatically to be scanned on successful build or re-signing from Bitrise with the pre-defined bundle ID (max of 1 dev, 1 production), implement logic to builds to only upload apps for those two pathways to reduce the number of apps scanned.

Development scan considered only a working tool - i.e. can have open issues during development.

Out of scope apps still uploaded directly to Data Theorem using web UI (apps that are not supported on Bitrise such as Unity apps) - provide more manual oversight for these rather than automatic approval.

Issues reviewed during development by those development teams, reviewed by FPMP on request or on identifying particular need for attention.

Additional access to scans by share link (30 days temporary link), or by providing direct access, but main access should be the development team who can close out issues if needed, current process already requires justification.

6.3 Pre-release:

On a release candidate being built, is uploaded to Data Theorem for a scan, using the production bundle ID. This takes the place of continuous monitoring.

This is flagged as pre-prod as above, with same problems, however a review based on known information (about app, about scan, about features, about libraries, about technology) is performed to identify blocking issues.

Information from the vulnerability scan is used to partially complete legal statements required by the stores (e.g. data privacy statements, however it is one input out of several).

6.4 Post-release

Re-scan before update becomes part of the pre-production element of updating an app.

Re-scan performed automatically on the pre-release candidate already, takes the place of production monitoring.

7 SSDLC Baseline And Possible Changes To Open Up Additional Vendors

See also [Reduced Cost Future Process Suggestion \(see page 19\)](#) - overlap comes from production monitoring and reduction of number of app scans by reducing environments scanned.

Current SSDLC baseline is at <http://go.merck.com/ssdlcbaseline>

Aim of changes is to remove barriers to reducing costs.

Mobile specific sections: 9.1, 12.1, 20, 21

Related requirements: 4.1, 6.1, 15.1

7.1 Summary:

Removing continuous monitoring requirement would reduce cost.

Merge accounts to reduce the defined need for production monitoring (if it is in our account we can see it, and we can associate it with a known app/known contacts/etc.).

Clarification of environment handling when it comes to vulnerability scans might reduce cost by reducing the number of orphaned/semi-duplicate app scans during development?

Optional - Depending on the tool chosen, change to SAST and/or DAST to 9.1 might be needed/might open up lower cost vendors?

Create a separation between internal apps (most of the risk comes from data transmission/storage which is mostly protected by our MDM controls/API controls) and external apps (where an obligation exists to protect the public and ensure software is free of defects/harmful content).

7.2 Continuous production monitoring removal mitigation:

1. Anything released would have to be a release candidate before being released.
2. Release candidates are scanned therefore anything that is published will have been scanned.
3. The average age of releases/updates is about 1 year.
4. Therefore anything released would have had a scan within past year on average.
5. An additional control at the end of each year is possible - has an app been published in past year? If not scan out of the release cycle process.

7.3 Detail:

7.4 Current 4.1:

User accounts. 4.1. The use and management of accounts (e.g., non-person, privileged, test) must adhere to the requirements defined in the Identity Access Management (IAM) Standard and Default Baseline.

Notes/Suggestions:

As written this prevents providing test credentials to third parties to perform DAST, and also causes administrative hurdles in providing the required test accounts for Apple's and Google's reviews in production environments.

Clarification is recommended to allow provision of test credentials for production environments for both scenarios. Clarification could be handled in section 13 or 15.

This requirement is also partially duplicated at section 18.

7.5 Current 6.1:

6.1 Separation of non-production environments (e.g., development, test) from production environments must be enforced (e.g., using access control lists).

Notes/Suggestions:

As written and commonly interpreted by application development teams, this leads to duplication of vulnerability scans and app instances (for example AppInstance, AppInstance QA, AppInstance Dev, AppInstance Test, etc.), and therefore increased costs as each counts as its own app instance depending on the tool chosen. Additionally this introduces false positives due to development environments having development artifacts by necessity, and additional administrative challenges / duplication of effort.

Possible improvement would be to specify either here or in 9 that the vulnerability scan *should* be performed regularly during development on a production build of an app to improve quality/reduce development time, but *must* be done on a release candidate hitting a production environment.

7.6 Current 9.1:

9.1. Applications developed by or on behalf of the company must go through an automated security scanning of source code or binary code using ITRMS approved static analysis tools to identify security defects and enforce secure coding practices. This includes but is not limited to Web Applications, Mobile Applications, API driven Applications or thick client Applications or similar. If an automated

scan is not technically feasible, a manual code review must be performed by peers to ensure code security.

Notes/Suggestions:

Good as is, however some tools look to only provide DAST, current tool provides combination of SAST and DAST. If a solution is chosen that provides only DAST then this element would need to be updated.

7.7 Current 12.1:

12.1. All external facing web assets including web applications, mobile applications, and APIs, must be protected by a Web Application Firewall (“WAF”).

Notes/Suggestions:

Wording is slightly off enough to cause some confusion. Not really relevant to mobile applications in themselves but relevant to back end services. Networked services (e.g. APIs, etc.) should obviously be protected, but this is covered separately.

7.8 Current 15.1:

15 Dynamic application security testing (“DAST”).

15.1. Applications developed by or on behalf of the Company must go through dynamic security testing using ITRMS approved dynamic analysis tools to identify security defects.

Notes/Suggestions:

Not fully performed at the moment, current service provides a combined SAST and DAST automated scan, with the DAST part only partially implemented (due to requirement to not share credentials with 3rd party/sometimes just not having credentials or using non-standard authentication meaning dynamic testing services cannot always fully explore an app).

Clarification for handling of test credentials is likely an important related aspect (since credentials would be required for full DAST) - should this be called out in 4.1?

As it changes what tools can be in scope, Integration with the SAST/DAST requirement (mobile is only specifically called out in the SAST requirement, but is not specifically called out in the DAST requirement)?

7.9 Current 20:

20. Mobile app production monitoring.

20.1. Continuous monitoring must be performed by ITRMS App Sec on mobile applications in Google Play Store and Apple App Store to provide automatic detection and assessment of code changes and alerting for newly discovered vulnerabilities.

20.2. Continuous monitoring activity must be performed not less frequently than weekly using ITRMS approved tools.

20.3. A comprehensive review of scan results must be performed, and necessary actions must be taken to rule out false positives.

20.4. For any identified vulnerabilities, ITRMS App Sec must collaborate with necessary stakeholders (e.g., vulnerability management services team, application development team, etc.) for performing root cause analysis, tracking and resolution.

20.5. Security defects must be remediated within the remediation timeframes defined in Appendix B.

20.6. For publishing of mobile applications, developers must use channels and accounts managed by Mobile Development Platform.

Notes/Suggestions:

Some tools require an additional cost for production monitoring. For the current solution this causes some administrative and organizational hurdles (for example missing production apps due to their only being available in certain countries so requires support tickets for addition of apps not automatically detected, duplication of release candidates and production apps, orphaned pre-production apps, etc.)

This element is potentially covered by scanning of the release candidate since this would be the same as the released app. Discrepancy with that comes in from an abandoned release candidate. Theoretically a small risk since updates to apps that don't add major new features should not be introducing large changes to production apps.

If we remove the requirement for continuous monitoring activity of published apps then this could reduce cost.

7.10 Current 21:

21. Inventory verification of mobile apps in public stores.

21.1. Divisional IT must scan for Company mobile applications in public app stores, reconcile the scan output and the CMDB on a weekly basis, and resolve any discrepancies at a minimum monthly.

21.2. ITRMS App Sec will support Divisional IT with investigation of discrepancies, identification of root cause and provide assurance that inventory verification was completed.

Notes/Suggestions:

Good as is - already done (both on a company level with masp.merck.com and divisional level for AH and HH), with any missing elements down to "unknown unknowns" that could not be included in an inventory anyway. Migrating all apps to a limited selection of accounts, along with regular checks and a well documented reporting structure for rogue app publication will remediate additional risks.

Improvement here could be addition of a channel to ServiceNow requests/"The Source" for reporting? However that would be outside the scope of the baseline.

8 Timelines

Our DataTheorem contract expires in June 2026, we have the possibility to exit early in October 2025.

Therefore a decision on which vendor to use must be made, and evaluation(s) of alternatives completed before October 2025.

8.1 Sprint Goals

S1 2025/3/5-2025/3/12

- [Smith, Robert](#) To document requirements, formalize notes from vendor contacts to create a table to match to those requirements
- Add in Gartner info about reducing # of apps to documentation → potential cost impacts
- Add Reporting API to requirements

S2 2025/3/12-2025/3/19

- Table of costs
- If an option presents cheaper: Start evaluation or decision stick with DT
- Discuss recommendations with [Marsik, Lukas](#)

S3

- Create/Update MAST standard from OWASP docs
- start evaluation of Ostorlab
- start alterations to SSDLC

Overall:

Roadmap == pick new tool or justify why existing tool is best

If sticking with DT - Cleaning old info in DT + scan tool results → threadfix replacement API integration

If switching lead time for API integration - CI/CD → scan tool + scan tool results → threadfix replacement - 1 to 2 weeks for API part+documentation of API part+future fixes?.

9 Tool Comparisons

For ease of viewing/comparison, see below and individual vendor notes for more detail

		Data Theorem	No wSecure	BDBA	Black Duck Polaris	C o d e Q L	AppDome	MobSF (Full)	MobSFscan	Ostorlab
Estimated Annual Cost (based on discussions with vendor)	All Apps (160 apps)	625,000 (with current discounts)	800,000	\$216,000 (\$650k 3 yr contract)	Included in enterprise license		12,000,000	Server running cost	0	\$150,000 (1000 scans in a year, see notes) \$300,000 (2000 scans in a year, see notes)
	Public apps only (110 apps)	625,000 (with current discounts)	500,000	\$216,000 (\$650k 3 yr contract)	Included in enterprise license		8,250,000	Server running cost	0	\$150,000 (1000 scans in a year, see notes) \$300,000 (2000 scans in a year, see notes)
	Cost basis (calculated per app/per user/per scan)	Unlimited	Per app	Per user (1000 users inc.)	Per user (1000 users inc.)		Per app	NA	NA	Per scan (1 app scanned within 1 month == 1 "slot")
Offers full coverage?	Does the solution do what we want?	Y	Y	Y	N (different approach)	N	N (different approach)	N (different approach)	N (different approach)	Y

Existing contract ?	Do we already have a relationship with vendor?	Y	N	Y	Y	Y	N	N	N	N
---------------------	--	---	---	---	---	---	---	---	---	---

Source	Requirement	Strict ?	Data Theorem	Now Secure	B D B A	Black Duck Polaris	Code QL	AppDo me	Mob SF (Full)	Mob SFs can	Ost orlab
Company Req	Provides a SAST scan of an app bundle	Y	Y	Y	Y	N	N	N	Y	N	Y
Company Req	Provides a DAST scan of an app bundle (can be limited scan depth)	Y	Y	Y	N	N	N	N	N	N	Y
Industry Std	Identify OWASP mobile top ten issues in general	Y	Y	Y	Y	Partial	Partial	Partial (prevention only)	Partial	Partial	Y
Industry Std	Identify if code is present to check for jailbroken device/check for official distribution	N	Partial	Y	N	N	N	Partial (prevention only)	Partial	N	Y
Industry Std	Identify known malware/bad patterns in our code, and in dependencies	Y	Y	Y	Y	Partial	Partial	N	Partial	Partial	Y
Industry Std	Identify any dangerous permissions requested	Y	Y	Y	Y	Partial	Partial	N	Y	Y	Y

Indus try Std	Identify secrets	Y	Y	Y	Y	Y	Y	N	Y	Parti al	Y
Indus try Std	Identify/list strings contained within app (reconnaissance)	N	N	N	Y	Y	Y	N	Y	N	Y
Indus try Std	Identifying endpoints to prevent inadvertent data transmission	Y	Y	Y	Y	Y	Y	Partial (prevent ion only)	Y	N	Y
Indus try Std	identify endpoints with known privacy/ security risks	Y	Y	Y	N	N	N	N	Partia l	N	Y
Indus try Std	Identify if code obfuscation is implemented (e.g. ProGuard implementati on)	Y	Y	N	N	N	N	Partial (prevent ion only)	Partia l	N	Y
Com pany Req	Provides a centralized dashboard for an executive summary	Y	Y	Y	Y	N	Y	Y	N	N	Y
Com pany Req	Allows easy flagging of false positives on individual apps, app group policies and system level	Y	Partia l	Y	Y	Y	Y	Y	N	Y	Y

Com pany Req	Integrates with existing build system (API for upload to scan)	Y	Y	Y	Y	Partial	Par tial	N	Partia l	Y	Y
Com pany Req	Integrates with existing ticketing systems (ServiceNow /Jira/Azure Dev Ops)	N	Y	Y	Y	Y	Y	N	N	Y	Y
Com pany Req	Supports common issues from applications created with Swift, Kotlin, Unity, React- Native, Java, .Net MAUI	Y	Y	Y	Y	Y	Y	N	Y	Y	Y
Com pany Req	Supports common issues from legacy applications created with Flutter	N	Y	Y	Y	Y	Y	N	Y	Y	Y
Com pany Req	Does not encourage long-term reliance on one given system in remediation advice	N	N	Y	Y	Y	Y	N	Y	Y	Y
Com pany Req	Provides remediation advice	Y	Y	Y	Y	Y	Y	N	N	N	Y

Notes on cost:

Cost for most services looks to depend on number of apps to be scanned.

Allowing free upload from build systems has usability benefits in that we don't have to create a project for a build to be scanned, but also has significant cost and administration impacts (for example a particular app might have several variants with bundle IDs changing from early builds to later builds, or

a particular app might be a white label app scanned as provided by a vendor with a different bundle ID, and each bundle change counts as a separate app, or there may be several development candidates that are abandoned, etc.) It is possible to reduce number of apps/administration/cost by limiting it on an process level so that each app must be related to a particular project, but this does limit the free development approach currently taken.

For those services which are dependent on app counts or scan counts for costs, controls could be implemented with:

- Requiring only release candidates to be scanned, using alternatives during development.
- Limiting bundle IDs, preventing both changes in bundle IDs and pre-release scans on .dev, .uat, etc. alternative bundle IDs.
- Removing CI/CD requirement and processing scan requests directly through masp.merck.com on a release.
- Changing the SSDLC baseline requirements to remove continuous production monitoring of app clients.
- Changing the SSDLC baseline requirements to reduce requirements for internal only applications.

Mitigations for process changes:

- Scanning only UAT builds or release candidates:
 - Best advice is early and frequent scanning, however majority of development scans are not reviewed by any teams, and those development builds are not changes to vulnerabilities.
 - UAT is performed on potential release candidates pointing to a different environment.
 - UAT builds are less frequent than development builds, but more frequent than release candidates.
 - Vulnerability scanning is technically part of acceptance testing.
 - Most UAT builds are expected to have at least some failures that would requires re-development.
 - Scanning on UAT would potentially be early enough in the development lifecycle to balance costs versus risk versus re-development.
- Coverage during development:
 - MobSFscan could be implemented during development to provide a balanced coverage of potential issues before a UAT candidate is created.
- Continuous production monitoring:
 - Anything released would have to be a release candidate before being released.
 - Release candidates are scanned therefore anything that is published will have been scanned.
 - The average age of releases/updates is about 1 year.
 - Therefore anything released would have had a scan within past year on average.

- An additional control at the end of each year is possible - has an app been published in past year? If not scan out of the release cycle process.

Requirements:

Below list includes strict and nice-to-have requirements

- Static scan of app bundle - can be done from source code or built bundle - preferred actual app. - defined by the percentage of our apps that we do not have source for
- Identifying 3d party library versions used and comparing against known CVEs.
- Identifying permissions used - allows ignoring of identified issues if related to a known app feature
- Identifying secrets - allows ignoring for known required "public" secrets.
- Inadvertent data transmission - identifying endpoints at least, identify those with known privacy risks
- Identifying strings contained within app (recon purposes outside of identifying secrets/endpoints)
- Malware - identify known bad patterns
- Code obfuscation - e.g is proguard implemented?
- Identify if code is present to check for jailbroken device/check for official distribution/Play Protect
- Identify any other OWASP top ten issues
- Dashboard / centralized system to see issues/statistics.
- Ability to easily flag false positives. NB *any* automated tool provides only reconnaissance of applications, triage of whether results are relevant always requires manual review and some knowledge of an individual application's core purpose, requirements, and features. Therefore an emphasis on the ease of administrating that tool in order to reduce false positives is encouraged.
- Integration with ServiceNow out of the box as designated tool by ITRMS for centralized vulnerabilities tracking

As only 49% of our apps have source code that we own, scanning of an app bundle is preferred. It is possible for us to take a split system approach if necessary.

If done with source code only approach would need to cover the below with minimal false positives:

- Kotlin
- Swift
- Java
- .Net MAUI
- React-Native
- Unity

Nice to have:

- Flutter

Must have (strict requirements):

OWASP Mobile Top 10 coverage.

Ability to easily administrate and tailor scans/results according to app features / target audience.

Advice provided must be unbiased (i.e. not pushing reliance on other security products from same company)

API for integration into CI/CD and automated reporting systems (NB we could work without this feature, but makes it easier to automate)

Candidates:

- [AppDome](#) (see page 33)
- [Black Duck / Polaris](#) (see page 34)
- [Black Duck Binary Analysis \("BDBA"\)](#) (see page 35)
- [CodeQL - GitHub](#) (see page 35)
- [Data Theorem](#) (see page 35)
- [Guard Square](#) (see page 36)
- [MobSF \(Full\)](#) (see page 36)
- [MobSFscan \(Lightweight source-code only version of MobSF\)](#) (see page 37)
- [NowSecure](#) (see page 112)
- [Ostorlab](#) (see page 112)

9.1 AppDome

9.1.1 Summary

AppDome offers a "wrapper" system that provides some controls to prevent some known vulnerabilities, however it does not scan for or remediate any vulnerabilities or provide any reconnaissance on the app bundle in itself.

The cost can vary between about \$50k per app to \$150k per app per year, with AppDome unable to give us an exact figure beyond that.

The process involves uploading an app bundle to AppDome's system, and then downloading a "protected" version that includes AppDome's proprietary framework which intercepts traffic between

the app and device/app and network, and prevents some attack vectors. It has customizable protection levels, but provides some features that can be configured to be against Apple's and Google's guidelines and may present some legal and privacy risk as implemented.

The AppDome system is akin to DataTheorem's Mobile Protect SDK which was previously trialled and failed due to not being development friendly.

AppDome's UI appears designed for small sets of apps, rather than large app volumes, and may not scale up well. It appears to be designed to be business user friendly rather than for developer or technical users, however our primary users are technical users.

Due to how the AppDome system is designed, once implemented use of AppDome would lock us into continuing to use AppDome until the enrolled applications are updated to remove it. Due to our update cycles this could present a large problem in being able to switch systems in future.

9.2 Black Duck / Polaris

9.2.1 Summary

Our existing Black Duck enterprise license includes Polaris, which can only scan source code, but can scan the technologies we code in. It would only cover 49% of our mobile applications. Potentially we could add in the Black Duck Binary Analysis license, which can handle app bundles, however this license was previously removed and will be covered separately.

The Black Duck Polaris service has export features, but training elements are only for internal developers (those with access to our SSO). Still potentially useful during development for those teams, however the majority of our apps are developed by teams who do not have access to our SSO.

The service has fix suggestions, but this is in beta only and relies on AI so might have privacy issues.

The service provides scan configuration based on particular policies, and options for scheduled testing on source code. This involves establishing groups and adding apps to the group (as opposed to selecting groups by app). This looked to be a little messy on the grouping during the demo, but given we don't have this option implemented at the moment then some quality is better than no quality. The configuration allows for scans at a pull request level which could be very useful during development.

Due to only providing coverage for 49% of our applications, Black Duck Polaris isn't a solution in itself, but could be a viable alternative tool during development to reduce the number of app bundles that need to be scanned.

9.3 Black Duck Binary Analysis ("BDBA")

9.3.1 Summary

This is an additional license option that is not included in our current Black Duck enterprise license. It offers app bundle scanning (as opposed to Polaris source code scanning). A demo is scheduled for 2025/3/10. The previous BDBA license was dropped in October 2024 due to a lack of use, which might cause complications. This might be a valid combination tool, or might provide full coverage. BDBA can scan binary files, and to some extent provide recon on app bundles, however it doesn't raise issues directly based on app bundles, only on the binary files contained within the app bundles. Some features may be overkill for our apps, while simultaneously not enough coverage for our apps as it is not originally designed for mobile app use. The UI is a little less usable than other options reviewed, but it provides grouping of policies to allow for better control of scanning the binaries, and establishing app teams. The cost for enterprise license for BDBA would be \$650k for a 3 year contract, on top of our current Black Duck system.

9.4 CodeQL - GitHub

9.4.1 Summary

CodeQL is an optional plug in to our GitHub instance, which can only scan source code. It cannot scan app bundles, so would only cover 49% of our apps. Although it is free for public repositories, there is a license cost to scan private repositories. Recommended by GitHub team to contact vendor to see a demo and to see the price range. GitHub themselves recommend NowSecure for our scenario.

CodeQL's main advantage is that it would integrate into our GitHub system, much the same way as other current security scans do. However it is possible for us to get the same feature for free using MobSFscan.

Account manager for GitHub is Sara Olson sara-skeddy@github.com³.

9.5 Data Theorem

9.5.1 Summary

Data Theorem is our current vendor, and therefore is a known service. Our current license costs \$625k pa for an unlimited number of apps, and this cost is heavily discounted. DataTheorem offers many features we don't currently use, but which are being explored. Our current contract expires in June 2026, with the possibility for early end to the contract in October 2025. Changing the number of apps/users will not affect the cost as previous negotiations were able to get us unlimited access.

³ <mailto:sara-skeddy@github.com>

Data Theorem offers combined SAST and DAST, production monitoring, etc. as desired. It has unexplored features that may meet the less strict requirements such as integration to other systems. There are significant UI issues with organizing the tool, such as the linking of apps and the management of false positives, however some of this comes from our allowing open access to development builds and our CI/CD system. Some of their remediation advice is not unbiased, and there are a number of false positives on every scan which makes information from dashboards inaccurate and hampers use.

As the main aim is to reduce costs, the recommendation is to stick with DataTheorem if other tools are more expensive while offering less coverage.

9.6 Guard Square

9.6.1 Summary

Guard Square was recommended to us by Gartner, we have not yet contacted them.

<https://www.guardsquare.com/what-is-mobile-application-security-testing>

9.7 MobSF (Full)

9.7.1 Summary

MobSF is a reconnaissance tool used by security researchers and developers, it does not provide full information or a dashboard, it merely provides a report on what it finds in a given app bundle. The tool itself is free/open source, but there is a cost for enterprise level support, and the tool does require us to run a server to host it for best use (possible to do on a per-app basis for free using docker, but docker does require a paid license to use). The tool was previously used on a limited basis before ITRMS AppSec was formed. MobSF is a recommended tool by various organizations, including OWASP.

The tool can be configured to output results to an API, which could be used to integrate it into our GitHub instance, or into additional dashboards.

The results require interpretation by a security researcher who knows the application and is not as simple as providing a list of issues that must/should/could be fixed along with remediation advice.

Recommendation is for MobSF to be a valid tool to use during development and to provide additional information, but not as a replacement for our current tool, and not for approving release candidates.

Example report (from 2021):



9.8 MobSFscan (Lightweight source-code only version of MobSF)

9.8.1 Summary

The MobSFscan tool is an off-shoot of the full MobSF tool, which can scan source code without the need for a server, but cannot scan app binaries. It is a security research tool which requires review to interpret if issues are relevant and covers less than the full MobSF scan. Potentially it is useful for teams during development, but it is not a substitute for release candidate scans. The tool is free/open source.

The tool can be configured to output results to our GitHub instance using SARIF, this would require working with the GitHub team to implement.

Recommendation is for this tool to be available to be used during development.

<https://github.com/MobSF/mobsfscan>

Bitrise test instance:

<https://app.bitrise.io/app/a847fd2a-4fe8-4e1c-b1e7-baea67cc36c2>

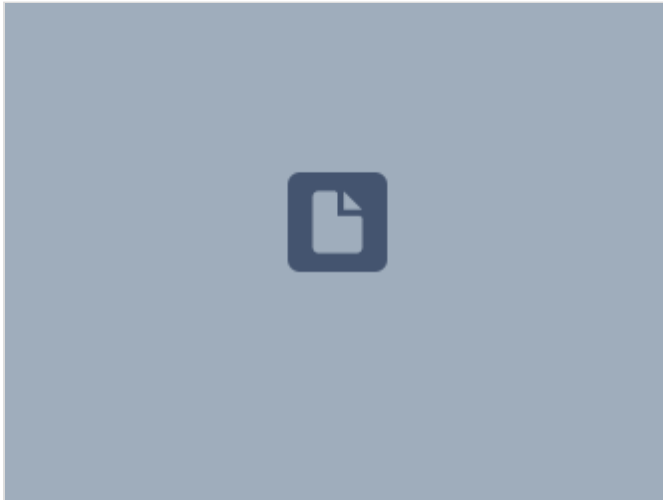
Example script:

```
echo "* MobSFscan - https://github.com/MobSF/mobsfscan"
echo "* Installing MobSFscan tool"
pip install mobsfscan
echo "* Running scan"
mobsfscan --sarif --type ios -o analysis-data.sarif $MOBSF_SCAN_PATH
echo "* Copying results to deploy directory"
cp analysis-data.sarif $BITRISE_DEPLOY_DIR/
```

NB integration with GitHub would require changes to allow external API access, or would require set up on each repository: <https://docs.github.com/en/code-security/code-scanning/integrating-with-code-scanning/uploading-a-sarif-file-to-github>

Results example:

```
{
  "runs": [
    {
      "tool": {
        "driver": {
          "name": "mobsfscan",
          "rules": [
            {
              "id": "ios_log",
              "name": "IosLog",
              "properties": {
                "tags": [
                  "security",
                  "external/cwe/cwe-532"
                ]
              }
            },
            {
              "id": "ios_insecure_random_no_generator",
              "name": "IosInsecureRandomNoGenerator",
              "properties": {
                "tags": [
                  "security",
                  "external/cwe/cwe-330"
                ]
              }
            },
            {
              "id": "ios_hardcoded_secret",
              "name": "IosHardcodedSecret",
              "properties": {
                "tags": [
                  "security",
                  "external/cwe/cwe-312"
                ]
              }
            },
            {
              "id": "ios_keychain_weak_accessibility_value",
              "name": "IosKeychainWeakAccessibilityValue",
              "properties": {
```



9.9 NowSecure

9.9.1 Summary

NowSecure's system offers features similar to Data Theorem, plus some more advanced features, with a more usable admin system and UI. It would cover our use cases and more. However it would cost more as the charge is \$5k pa per app, with an additional support cost of between 10-20% on top of that price. For coverage of all of our apps that are currently published this would work out at around \$800k pa, or if we limited the number of apps to only those publicly available costs could be reduced to around \$500k pa. One of the main features that is not included in that price is app store monitoring, which is included in our Data Theorem license.

NowSecure does everything we would want it to do, and it is recommended by both OWASP and GitHub, as well as by Gartner.

9.10 Ostorlab

9.10.1 Summary

Ostorlab provides all the features that DataTheorem provides, along with more advanced features. It has more usable and clearer admin controls and UI, and offers more customization features than DataTheorem. Some of the features may be beneficial in future, but are likely overly complex for our uses. We can control when scans run, the frequency of rescans, the application of policies to app groups, the grouping of apps, etc.

Ostorlab's pricing model is based on "app slots", where slots can be re-used by different apps, and each slot counts as a scan. It offers multiple scan options such as a fast scan during development, and a full scan for release candidates, as well as a privacy scan for covering legal requirements. The system allows for uploads from multiple sources, including API, GitHub, the App Store, the Play Store,

etc. The Ostorlab API offers every feature that is currently implemented in their web UI, so could easily be integrated into additional systems. Their API and other documents are public and well maintained for ease of reference.

The cost estimate varies depending on scanning strategy due to Ostorlab's pricing structure, where a "slot" is available for unlimited scans of one app per each month - if we scanned all our apps once a month then we'd need a slot for each app for each month of the year (so ~ 180 apps $\times$ 12 months == 2160 slots). After discussion within the mobile platform team, we would likely need between 500 to 700 slots per year, and we can optionally impose limits to scan release candidates only. This would still work out as cheaper than DataTheorem, but might work best if combined with development scans using MobSF to avoid surprises on scanning release candidates. If we needed additional slots then these can be purchased through the year. If a slot is not used in a year then it doesn't expire, but once activated it is locked to one app and must be used within the month (defined as the 28-31 days following first use of that slot for that app). Adjustments to our company policies and assigned bundle IDs would be needed to fully benefit from this pricing model.

The "slots" per-scan system is different from other pricing structures, so slots description and cost estimate documents are attached.

Minimum slots (180 scans == \$40k)



Workable (1000 scans == \$150k)



Better (2000 scans ==\$300k)

