Dependency Upgrades

10x Investigation Recommendations

September 1, 2020

Alexander (AJ) Stein -- alexander.stein@gsa.gov

Mike Stern -- michael.stern@gsa.gov

Background

The idea for the Dependencies Upgrade project came from a submission by Aidan Feldman, and the Census xD team completed the Phase 1 project.

> *Without timely action, software dependencies and vulnerabilities can lead to major data breaches. Currently, many software applications either rely upon a manual process for updating or require individual set-up of automation for code repositories. TTS will investigate whether new or existing tools can conduct automated scanning of software dependencies across government open-source repositories, and whether there is the potential for a centralized security scanning service offering.*

The subject of this report is a particularly deep subject matter especially for readers not familiar with software engineering practices. The subject of security and dependencies is also broad and open to assumptions even for those with familiarity in the problem space. We believe it's worth spending the first part of the report in defining the subject and narrowing the scope of the conversation.

Software and Dependencies

All software is built from other software.

Software, for the purpose of this report, is defined as custom developed applications used to:

- serve interactive web applications and websites
- perform business functions: batch processing, data management, and transactional updates

Commercial software, while still software, is not part of the scope of the report, this project does NOT focus on software procured for usage on laptops/desktops or mobile devices. Nor does the report consider Software as a Service (SaaS) that is built commercially, and used by government entities. The report includes discussion about SaaS, but in the context of the solution not the problem space.

When speaking about dependencies, we will be referring to the parts of custom developed software that **other** people have written which are combined with custom software to help fulfill the intended goals of the user.

Custom software is composed of:

- code that is written for the project (sometimes referred to as source code)
- Included dependencies (code written by others)

These are the general step to make software available for use:

1. Write source code

2. Include dependencies
3. Build the software (combines steps one and two)
4. Deploy the software

All software is versioned, both the source that the author writes as well as the dependencies being used. In order to include a dependency, the author declares the combination of the software name, and its version. This dependency declaration allows the software to be built. Included in the building process is the step of combining the software the author wrote, with the software someone else has written.

There are other types of dependencies that software needs to run:

- hardware (memory and hard drives),
- the operating system (linux, windows),
- firmware (interacts between hardware and operating system)
- middleware (like databases or messaging queues),
- external dependencies (cloud provided services, APIs, authentication platforms).

None of these types of dependencies are the focus of this report & remain out of scope for the project

Security, Software, and Dependencies

All software is inherently flawed, or at least at risk of being exploited from its original intention. Broadly defined, these flaws or ability to exploit are referred to as vulnerabilities. Uncorrected vulnerabilities have real consequences: leaks of private information, misuse of federal assets, financial exposure or loss of trust in government services.

Security, in the context of this report, is any action taken to reduce the risk of the inherent flaws or potential misuse of the running software. The mission and methods of security is sometimes at odds with the explicit goals of the software. Distilled to the most extreme version of this: the only software that is perfectly secure is that software that is not run.

Tooling

Through the use and inspection of software, flaws and risk of exploitation are discovered. This discovery leads to the creation of new versions of both software and the dependencies the software relies on. The discovery is often identified and reported to standards bodies. The reports enable different tooling to identify impact and how far the risk spreads. Security tooling focuses on the discovery, categorization, assessment of severity, and communication of software vulnerabilities.

The dependencies that are used within an application factor heavily into the software development risk calculation. Many third-party dependencies, most often open-source libraries, have published disclosures regarding known vulnerabilities. These vulnerability

disclosures are tracked and maintained in centralized repositories that provide insight to software development organizations into their applications' overall risk profile.

Process

There is no unified way to take the knowledge of vulnerability and identify it in all the impacted software. Even if there was, there still remains the need to apply the required change, rebuild the software, verify the desired change with no impact to functionality and deploy it. The activity associated with the above will be referenced as a process for the purposes of this report.

The combination of tooling and process is applied with the goal of reducing the risk of custom developed software, while balancing the sometimes conflicting goals of the software and its users.

How this Relates to Federally Developed Software

The recommendation for the Phase One project is located in the TTS Google Drive. The premise behind the pitch ties into the idea that all software is inherently risky. In order to deploy custom software developed using federal resources, the Authority to Operate (ATO) must be granted. There are security compliance controls required to gain ATO that are associated with dependency management and vulnerability scanning (NIST 800-53 (Rev. 4) RA-5).

The requirements for software development security compliance differ by agency. For applications that qualify as Low and Moderate systems for GSA as Lightweight Authority to Operate (LATO), the Vulnerability Scanning (RA-5) is a required control. In addition to the NIST standard, there is further guidance provided by FedRamp for Dependency Scanning. 18F's "Before you Ship" guide describes this LATO designation for applications that are:

1. Low or Moderate impact level, **and**
2. Built using agile methods **and**
3. Deployed to cloud infrastructure which has already received an ATO (such as AWS, Azure, or cloud.gov).

The RA-5 control remains the responsibility of the **application** team, even when part of an LATO and deployed to cloud infrastructure with ATO. This control is declared in the respective customer responsibility matrix (CRM) for each of the cloud service providers.

The Goal

The goal of this 10x project is to investigate methods to help development organizations fulfill the responsibility of this control. For additional information regarding the problem space, we created a primer document to contextualize dependency management as part of an organization's information security program. The Sonatype *State of the Software Supply Chain* (registration and download required) also includes examples and explanations of the relevance for dependency management and threat detection.

Recommendation

We recommend **NOT** moving forward with a Phase 3 for the Dependencies Upgrades project. Effective and timely dependency management comes down to two factors: tooling and process. The research findings have not provided compelling evidence for a "one size fits all" or cross agency approach for tooling or process. Software development organizations will have specific individualized needs for both their tooling and processes to successfully mitigate the risk of outdated software dependencies.

A cross agency shared service or centralized implementation has merit and could be successfully implemented in organizations with high application deployment concentrations. This deployment point often comes in a code repository, a centralized build and deploy pipeline, or a shared infrastructure, such as a Platform as a Service (PaaS). Tooling exists that provides cross-language scanning support, flexible deployment, and informative reporting. Organizations must commit to both tooling and process for such a service to succeed. We have seen examples of this type of commitment, but at the time of this report, there is not enough support to recommend 10x involvement in a Phase 3 implementation of a shared service.

How We Arrived at This Recommendation

Our research confirmed that, despite similarities amongst software development organizations in diverse federal agencies, relevant stakeholders have differing opinions on tooling and used in dependency management processes. The Phase 2 10x team garnered insight by:

1. Interviewed stakeholders (see *Appendix* for details) on their usage of currently available tooling.
2. Built a prototype to support further exploration of the solution space. The prototype explored building a dependency upgrade management service for cloud.gov tenants. The prototype was demonstrated to the cloud.gov team. The intent of choosing cloud.gov was to assess the potential of centralized tooling and the ease of customization of the tool to suit individual tenant workflows.

Through the feedback received during interviews and the presentation of the prototype we were able to validate our hypothesis: even for groups unified by a common problem, providing a standardized, streamlined solution does not provide alignment that will scale to accommodate the diversity of these teams' workflow needs. The cloud.gov team further validated this hypothesis by confirming that this type of solution would not be viable as a shared service for its tenants. This further validated the hypothesis. Despite having a common problem, and tenants in an extremely homogeneous environment, there is enough diversity in process to pose significant adoption challenges even if a shared service were available.

Findings

Tooling is available both in common commercial and open-source offerings. There are several free to use products that have optional premium tiers that extend the freemium model. We considered the different types of tooling and believe that most are viable depending upon the software development organizations' needs.

The success of a tool choice is entirely dependent on the process adopted by the software development organization. For detection and reporting tools to be effective, there must be someone responsible for reviewing findings and taking action to reduce the risk associated with any positive findings. In addition to having a defined process for reviewing and triaging findings, the organization must be confident in testing and deploying updated software. Every development team we interviewed affirmed their concerns over regressions in the functionality of the software introduced by dependency upgrades. Even for tools that automate the upgrade of dependencies, staffing is required to verify that these changes do not impact the software's clients upon successful deployment. The *Demand for Options* section further discusses this topic.

It was a goal of the Phase 2 exploration to prototype a potential solution. As we explored various organizations' needs, we realized that a generalized approach would face significant adoption challenges, even if we provided a solution that went above and beyond guidance in the form of documentation. Due to the difficulties associated with adoption, the 10x team shifted focus to finding an area of high concentration with the potential for a more significant impact on multiple development organizations. TTS currently uses scanning tools that operate on source-code repositories. Tooling built into GitHub and GitLab are explored further in the *Availability of Tools* section below. The 10x team focused its dependency scanning prototyping efforts on minimizing developer effort by leveraging their ongoing use of cloud.gov as a preferred PaaS.

TTS Solutions' PaaS, cloud.gov, became an ideal target of high impact for various reasons. We further explore these reasons when discussing the cloud.gov prototype discussion in the *Centralization of Tooling* section. After sharing the prototype and discussing the potential for further development, the cloud.gov product management notified the 10x team that

dependency scanning was not part of the immediate roadmap for further investigation. We concluded the Phase 2 prototype work based on that decision. The 18F GitHub organization hosts the prototype code repositories, and the potential for future activity is described in the *Next Steps* section.

QSMO and Shared Services

The Quality Service Management Offices (QSMO) program is an emerging area that will impact dependency upgrade guidance as these offices refine the scope of their mission. The Office of Management and Budget (OMB) has identified specific functional areas that cut across the federal government and have the potential for shared services. OMB is also designating specific agencies to stand up QSMOs for select mission support functions. When mapping the functionality of dependency upgrade scanning, the closest relevant QSMO is the Cyber Quality Services Management Office**.** The Department of Homeland Security and the Cybersecurity and Infrastructure Security Agency's (CISA) organization are the QSMO. We consulted DHS representatives regarding this project and confirmed this work is not relevant to the QSMO mission or their workstreams. They consider dependency management a pre-deployment development activity, and thereby, out of scope. Given that dependency upgrade risk ultimately maps to cyber-awareness, the Cyber QSMO may become more relevant as it further defines mission and scope.

Demand for Options

Overall, the interview subjects' consensus is that the 10x proposal would require a significant differentiator to distinguish itself from existing market options. We hypothesized that one significant differentiator, having surveyed a broad range of existing solutions, is to link dependency state in development environments more closely to actual deployments. This can be done by integrating with the deployment pipeline as efficiently as possible. The ability to have near real-time visibility into actual deployed dependencies and not just versions in a repository, would have value with discovering a new critical vulnerability and better tracking mitigation. The general appeal to using deployed artifacts is avoiding drift of source to deployed applications. It also benefits from having a fully built transitive dependency graph, rather than just interrogating the declared dependencies as part of the source.

We authored a Dependency Management Problem Spaces and Solutions document that analyzed the demand for options and the competitive marketplace. There is a further breakdown of the market in the Cloud.gov Tenant Interviews section below.

Success is Process and Budget Dependent

When calculating a project's security risk, it is essential to properly estimate operations and maintenance burden for the period when active development ends. Key to that estimate is considering dependency management and related security maintenance. When attention is no longer focused on active development, the risk for dependencies going out of date

remains. Operations and maintenance activities require ongoing funding, to “keep the lights on” for the project when deployed. Maintenance and operational budgets should include monitoring and reporting dependencies, making upgrades when necessary, and also account for deployment and verification to ensure regression free updates. Budgeting resources, with engineers and finances, is a function of the software development organization. The organization of engineering activities is separate from the investment in the primary deliverable, but applies regardless of team size. As simple as this nuance is, many organizations fail to understand their own structure and culture and its direct relationship to ongoing operations and maintenance.

Effectively handling operations and maintenance tasks, among them dependency upgrades, motivates how stakeholders classically describe, for lack of a better term, their own organizational “maturity.” This framing is inherently static and flawed. Even contemporary, government-backed IT governance frameworks define these choices as dynamic and open to an organization’s intentional planning and unplanned choices. In that vein, we analyzed dependency upgrades and organizational structure in terms of implementation tiers. This implementation tier concept in the NIST Cybersecurity Framework (CSF) provides a gradual continuum with four discrete steps. Although not absolute, there are common parts of security operations, and dependency management in particular, that are informed by implementation tiers. As noted above, and visualized in Figure 2 in the Appendix, the CSF supposes that the four steps with the continuum of implementation tiers are ***Partial, Informed, Repeatable, and Adaptive***. Along this continuum, teams organize development work, define standards and practices that are communicated internally and externally to partner organizations. All of the parts make software development whole, and the sum of these choices dictate the level of discipline in processes like dependency upgrades. Each organization chooses:

- whether to use commercial-off-the-shelf tooling, build tooling, or a hybrid of both
- the amount of governance that goes into the resulting workflow
- how that governance scales

With this rubric in mind, we interviewed a variety of stakeholders in the government to assess their respective implementation tier, how it impacts their choices from available tooling and the difficulties around process management that came with them (see the *Tenant Interviews* section).

One interview subject suggested the concept of insourcing product ownership goes a far way in solving the problem of unplanned operations and maintenance costs. Product ownership as a function left strictly to government employees, provides continuity and stability instead of contract staff. If a system supports mission activities, involving someone with mission accountability is critical. This individual needs to have responsibility and authority to define the process and ensure staffing in place for ongoing dependency maintenance, verification, and testing. Application security is part of the mission and should not be separated from it.

A second theme emerged during interviews related to setting accountability for contracted development or procured software using contractual language. An 18F consulting lead recommended writing contracts that included Quality Assurance Surveillance Plan (QASP) language that defines keeping dependencies up to date and vulnerabilities remediated as part of the overall quality of developed software. Additionally, a different TTS interview subject suggested contractual language that requires SBOM declarations as part of procured software delivery.

Throughout the interviews, the following became clear: at the operational level, technical leads make do with the best tooling they can afford, spanning aggregating information across individual tools manually as part of operations rotations. At the strategic level, procurement and acquisition officers are restructuring contract requirements to require dependency management as an aspect of quality for continually delivered software, not a gate to prevent deployment. These approaches signal aspirations for development teams with the goal of more robust tooling and processes.

Availability of Tools

There is no shortage of tooling to manage dependencies, with varying capabilities, and approaches. Some tools only report what libraries need upgrades; others automatically submit code changes to fix them. Some are small, self-contained utility programs that can be run on laptops or isolated servers; others exclusively focus on reading source code from specific cloud services and APIs. Some in the latter category also integrate with an ecosystem of source code management tools the vendor provides. Given the lengthy and technical nature of the findings, we leave the discussion of tooling to the Appendix, as broken down into subsections: *Traditional Component Analysis and Management, Standards Organizations Tooling, Commodity SaaS Platforms, Containerization and Artifact Scanning, Static vs. Dynamic Scanning, Standardization of Tooling and Data Interchange, Operational Level Concerns, Security Compliance Ecosystems*.

Prototyping of Vulnerability Scanning infrastructure

The 10x team chose the open-source dependency scanning tool *Vuls* for the prototype. This tool has a modular architecture that allows for various deployment topologies and flexibility in its scanning and reporting modes.The modular architecture is composed of several components that can be assembled via API. The modules are:

- Vulnerability Databases (multiple sources)
- Vulnerability Scanner
    - Multiple language and dependency management systems
    - Customizable infrastructure and software scanning modes
- Vulnerability Reporting (multiple deployment models)
- Vulnerability reporting UI

The prototype implementation chosen relied upon Local Scan Mode. This mode was selected because it required no agent running locally, simply network connectivity and permissions to

the artifact buildpack and to the reporting server. This mode demonstrates the ability to run scans without elevated or special privileges for cloud.gov PaaS tenants. Initially, the reporting server and vulnerability database were intentionally hosted outside of the cloud.gov environment to illustrate the flexibility of the architecture. Hosting outside of cloud.gov also demonstrated the option of using a centralized reporting server or isolated server hosted in different infrastructure, allowing integration with userbases in cloud.gov, CloudFoundry environments, or more unique deployment environments.

The primary focus of this 10x project is dependency management and scanning, but *Vuls* also allows for scanning of the operating system (OS) level and system packages, giving additional context and options. For the prototype, we chose to use a non-OS level scanning option. Please see the *Next Steps* section below for links to the repositories created to support the prototype effort.

The prototype's chosen architecture from the options that vuls.io allows;
https://github.com/18F/10x-dux-vuls-eval/blob/master/docs/ARCHITECTURE.md

Figure 1. Data flow and sequence diagram

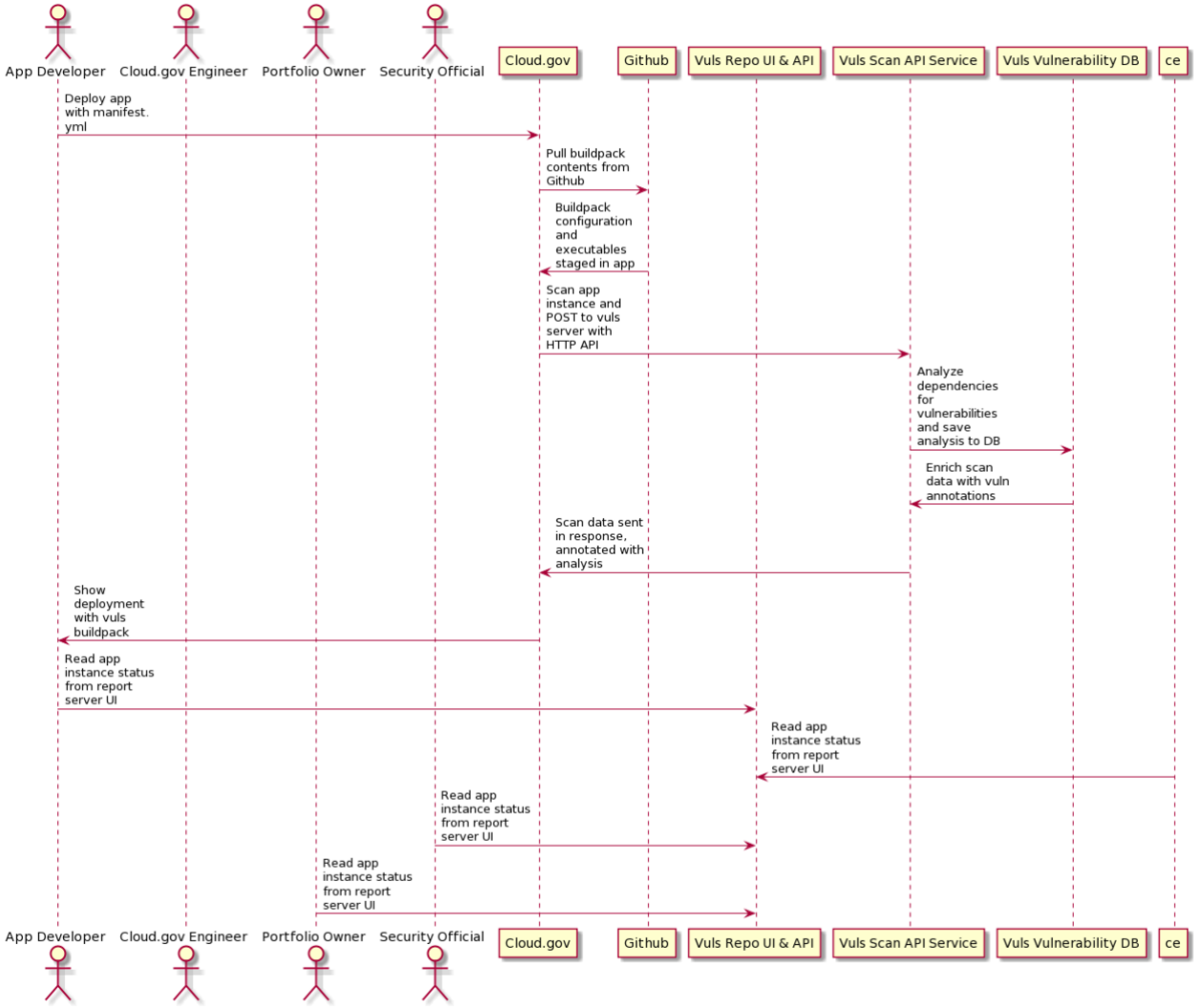


Figure 1: image demonstrating the interaction between actors and logical components of the vuls prototype

Centralization of Tooling

Background:

Midway through the project, we met with members of the cloud.gov team, including the product owner, a platform operator, and a consulting engineer, to discuss the Dependency Upgrades project. The goal of this meeting was to present on the *Vuls* prototype and pitch for building out a dependency scanner as an opt-in service for use by cloud.gov tenants. The *Vuls* prototype was built using native tooling in the form of Cloud Foundry sidecar buildpacks. After the 10x briefing, the cloud.gov team reviewed internally. Subsequent to that review the team recommended that 10x utilize a portion of the remaining budget to validate and assess

customer needs and validate if there is a real customer need that is not being addressed by existing tools.

Part of the concern has been the competitive, crowded landscape for dependency scanning tools. The cloud.gov team wanted to ensure market differentiation that would justify further investment in building out the service in cloud.gov.

Approach

To accomplish this, we subsequently took two different approaches:

- Interview three tenants of cloud.gov, representing different segments of cloud.gov tenants
- Present the idea to the TTS Security Compliance Guild at their monthly meeting. The outcome of that presentation and discussion is summarized below as well

Cloud.gov Tenant Interviews

The cloud.gov team provided the three tenants we spoke with. These interviews were scheduled as half-hour meetings and were conversational. We used the original pitch deck from the meeting with cloud.gov as a framework for the conversation. We introduced the problem, the research conducted, the proposed solution, and some additional insight on 'how we got here'. After ensuring clarity, the interview subject was asked: "Is this something that you would find useful for your development organization?".

The results effectively represented three levels of utility of the idea. All three respondents validated the idea in concept but found different utility for their current or future development needs. The three responses are as follows:

- The "Advocate" would find immediate use in current and future development efforts.
- The "Curious" was interested in the concept, but thought that the significant challenges are not merely in detection, but in reporting, communication, and most importantly, program level funding for ongoing maintenance of mission-critical services.
- The "Doubtful" was not interested in the proposed solution for their use: established process and tooling already accomplish their goals. The Doubtful has previous experience that shows the addition of tooling and automation muddies the water by providing multiple sets of truths which cause their existing process to fragment.

Security Compliance Guild Presentation

As part of the cloud.gov internal discussion, Peter Burkholder asked the #g-security-compliance channel the 10x idea "Does [the 10x Dependency Upgrades project concept] still have value to folks in 2020 or has the availability of Snyk and Dependabot made this less of an issue?". The ensuing slack discussion led to an invitation to present at the guild meeting to elaborate on the idea & the progress so far.

Discussion points after presentation & demo:

There were several follow on actions noted after the discussion. Issues have been created in the appropriate repository for any follow up actions that were not accomplished as part of the Phase 2 closeout. See the GitHub project/issue board as well as the following Next Steps section below for full details.

Next Steps

We leave behind several repositories that allow for the continuation of the work if priorities change in cloud.gov roadmap, or implemented by individual development organizations. A GitHub project board that tracks issues across repositories was created as well.

Prototype Repos:

- Dependency Upgrades eXample app (DUX) on github.com
- Vuls reporting server infrastructure as code on github.com
- Dependency Upgrades vuls.io buildpack on github.com
- A fork of the Vuls API server on github.com
- A fork of the official Vuls web frontend on github.com

Using our ongoing prototyping experience and user interviews, we will prioritize and enhance the prototype through the following objectives until the conclusion of the project.

Upcoming Prototype Development Tasks:

- Enhancement: Complete conversion of multi-buildpack to a modern sidecar buildpack
- Enhancement: Adapt deployment from vanilla AWS to inside a cloud.gov space
- Enhancement: Utilize cloud service brokers for RDS database
- Enhancement: Prototype exporting dependency data from vuls to a vulnerability manager server like DefectDojo
- Enhancement: Design and prototype transformation of dependency data from vuls or DefectDojo exports in Heimdall Data Format to map 800-53 control compliance
- Documentation: Update deployment documentation for others to deploy their own instances of the prototype as desired

Advocacy for Platform Extensions

As part of the exploration with the security compliance guild, there were several ideas that spawned from the prototype demonstration. Both quotes relate to security compliance with the same goal as the prototype: reduce application risk and satisfy security controls using Cloud Foundry tooling:

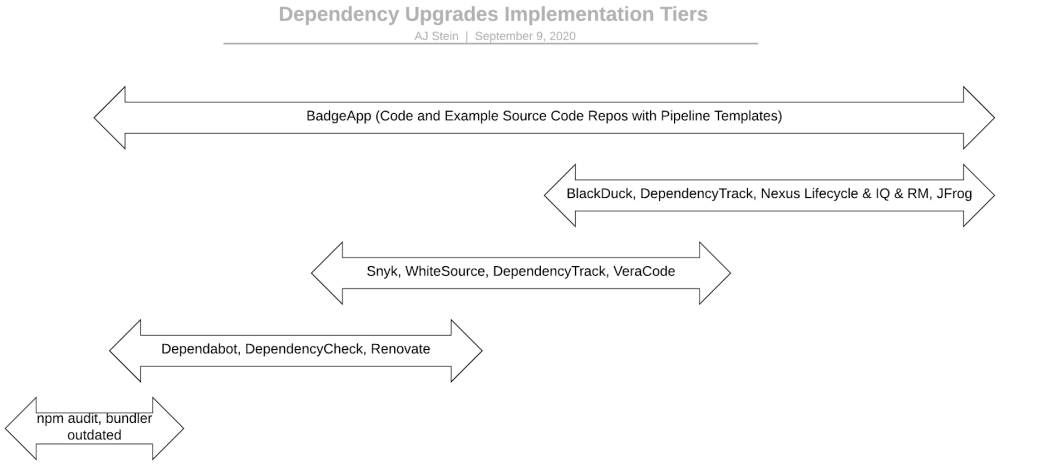
- *"I think there's a lot of compliance tooling potential if the pattern of sidecars that deploy local stuff in the container, but also take configuration from a bound service independent of the app deployment rhythm, and potentially report into a central service."*
- "*You could have a few different sidecars, each handing an individual NIST control, and those results can be gathered/reported centrally in something like DefectDojo."*

Appendix

Resources:

- Project artifacts:
  - Dependency Management and You
  - Dependency Upgrade Problem Spaces and Solutions
- Prototype Related
  - Vuls.io
  - VulsRepo (reporting UI
  - Cloud.gov
    - Operator notification event stream Issue request
  - Buildpacks CloudFoundry lifecycle
    - Multi Buildpacks
    - Sidecar Buildpacks
      - ClamAV Antivirus scanning while running
      - Monit tampering detection while the application is running
    - OSBAPI Cloud Service Brokers
    - CloudFoundry Cloud Service Brokers
  - Integration with Defect Dojo for reporting
- Fedramp and Controls
  - Fedramp Quick guide
  - Vulnerability Scanning supplemental guidance
  - 18F Before you Ship guide
  - Guidance for compliance
  - TTS/18F guidance on Dependency Analysis
- Security Process example for fec.gov
- Dependency Management process for TTS

Figure 2. Process and tooling adoption matrix



	Partial	Informed	Repeatable	Adaptive
Who:	Small, unstructured developer organization	Small developer organization; multiple developers or teams; limited hierarchy	Medium to large development organization; multiple teams with some hierarchy or structure; large enough developer to developer communication no longer efficient	Large development organization; likely hierarchical or spread out into agency component divisions
What:	One or few development projects	Few development projects	Multiple development projects; moderate or significant development load	Many development projects, often interconnected or dependent on each other;
How:	Completely reactive due to lack of options	Reactive, but often proactive	Almost equally proactive and reactive	More proactive than reactive
When:	At the discretion of developer(s) in the organization who volunteer	Periodically during the development lifecycle, but optional and not reviewed by team or development organization	Periodically required the development lifecycle; team and development organization review and collect metadata and metrics about dependencies when possible, but not an organizational priority	Frequently during the development lifecycle a requires; development organization requires developer(s) and team(s) to review metadata and metrics about dependencies and report them at required interviews
Information Sources:	Manual, unstructured review by volunteer: common "word of mouth" sources for developer news; third party reports from source code audit; third party vulnerability report based on system running developer software	Low-friction workflows and free tools; often use tools built into package managers of the language and runtime the developers chose; sometimes free or freemium third-party tooling	Workflows that limit or prevent changes to source code or system deployment for dependencies that present significant risk; use of language runtime tooling, but frequrntly strong use of free or freemium tools to track dependencies across source code repositories, teams, developer organization(s), or the whole agency; dependency tracking tools used to measure patching timelines and efficacy	Centralized workflows and toolings that present developers their dependencies and their use relative to different developers, teams, developer orgs, and potentially the agency; often, but not always, uses a centralized package manager tool with software component analysis (SCA) tools so developers use only authorized safe dependencies or request others to review including new packages; tools allow for deduplication of effort and reporting on false positives for internal and external consumers
Centralized Tooling:	None	Some consistency in tools, but limited or no centralized management of common tools, limited or no policy	Some centralized tooling, but not complete	Comprehensive centralized tooling driven by stakeholders and developers, mandated by policy
Internal Alignment:	Little or no alignment with the agency mission or cybersecurity directives	Somewhat aligned with agency objectives; developers choose to review dependencies on schedule or on a common trigger condition	Good alignment with agency mission and cybersecurity objectives; developers review dependencies with schedules and workflows based on these objectives	Very strong alignment with the agency mission or cybersecurity directives; development organization works with agency leadership for periodic review and adjustment of schedules and workflows with these objectives
External Alignment:	Little or no communication of dependencies with external stakeholders about dependencies used or software they make others use as dependencies	Ad-hoc communication of dependencies with external stakeholders about dependencies	Structured but infrequent communication of dependencies with external stakeholders about dependencies	Development organization and agency have well-defined strategy for external stakeholder communication, network of stakeholders, and actively build an information sharing network on dependencies themselves, workflows, tools, and practices

Figure 2: Matrix to assist organizations in assessing their implementation tier, it factors in tooling and process management

Interview Subjects:

1. Cloud.gov Compliance Lead
2. Consulting engineer at 18F
3. Tech lead at Federal Elections Commission-- fec.gov
4. Engineering supervisor at 18F
5. Technology Director: TTS, GSA
6. Engineer on data.gov, former Product Manager of cloud.gov
7. Product Manager of cloud.gov
8. Product Manager of code.gov
9. Two Developers/Operators of Federalist
10. Director of Security Operations at GSA
11. Member of Joint Advisory Board GSA TR, Continuous Monitoring reviewer for cloud.gov
12. Director of Cybersecurity Initiatives at National Telecommunications and Information Administration in the US Department of Commerce.
13. DevOps and Security Compliance lead for cloud.gov
14. 18F Developer and open-source Advocate
15. TTS Tech Portfolio team member

Tooling Discussion

Traditional Component Analysis and Management

Vendors with complex, large-scale development lifecycle platforms describe their dependency management functionality as part of Software Composition Analysis (SCA) and Software Component Management (SCM). Synopsis Blackduck, JFrog, Veracode, and Sonatype Nexus ecosystems classify themselves as SCA and SCM platforms. These tools are multi-faceted and target large, often hierarchical user bases associated with enterprise deployments. Although incredibly powerful as a class, the design and defaults align with traditional enterprise secure development processes that do not favor development in the open and often intentionally prevent direct access to the upstream dependency repositories package managers use. Each ecosystem's culture and practices are different, but historically, these SCA and SCM platforms' target audience was not those with DevOps first cultures. In this model, there must be a singular source of truth, an inventory of approved packages and versions. All developers are restricted to using that inventory, where restrictions are made effective by either selectively proxying or completely blocking direct access to upstream repositories package managers used to upgrade dependencies. As a side effect, day-to-day development is slowed, even if these tools authorize versions without administrator approval, they are frequently several releases behind or completely prevent the addition of new open-source packages. Only recently has this approach become perceivably less favorable, as cloud-native vendors, more in tune with more public development practices, have taken shape.

Standards Organizations Tooling

The need for dependency upgrades in modern development is universal; multiple free or open-source commodity tools serve the smallest development projects' needs. Tools like OWASP Dependency-Check will scan a folder of source code, regardless if it is from a source control management tool like Git or just a folder with source code naively extracted into a folder. It uses available data from a variety of public databases to simply report outdated dependencies and applicable vulnerabilities. Minimalist utilities that avoid opinionated workflows and integrations like this allow flexibility favorable to the smallest of development teams and even large groups with mature DevOps practices. These focused solutions allow for integration to organization with custom or in-house requirements, and do so by focusing on the core problem of only reporting outdated versions.

Commodity SaaS Platforms

Commodity SaaS platforms for source code management and software development, like github.com and gitlab.com, have grown significantly with increased adoption, via closed or open-source, for the private or public sector development. These tools' success emerged after years of industry investment that spans the continuum of dependency management tools, from small utilities to full-featured SCA platforms. The new Commodity SaaS platforms, intentionally or not, brought new viewpoints and a different approach. As these platforms matured, they built marketplace functionality, evolving from API integrations with different vendors, to extending functionality *inside* their development workflows while increasing developer productivity.

Two relatively recent acquisitions have further validated this scanning model: Microsoft acquired Dependabot for inclusion in Github, and GitLab acquired Gemnasium for inclusion in its SaaS source code repository pipeline. Additionally, competitors created or refined external integrations, like Snyk and WhiteSource, which continued to operate independently, with modest free usage tiers and tiered pricing subscriptions for enterprise management and oversight features that customers often associate with the heavier-handed SCA platforms described above. Whether official "marketplace" platforms or third-party integrations, the ease of low-friction integration in the source control management system, where the developers have the most control, has been significant. Current TTS practices are no exception. Additionally, SaaS-based monitoring and analysis translate into almost immediate pull requests, where developers can merge changes into dependency manifests with an ever-shortening feedback loop.

Containerization and Artifact Scanning

The advancements in SaaS platforms, paired with the adoption of immutable deployments heralded by DevOps culture, have underpinned significant advancements in the dependency management landscape. Containerization has drastically improved the approachability and popularity of immutable deployments. Containers are composed of immutable layers, cryptographically hashed. Not only is each discrete change sequentially captured with a

globally unique hash, but the list of commands that generate the discrete change are captured as well. One or more layers will often include commands to load dependencies and then application, sealing the deployed artifact as a single entity. If a team needs a new copy of an application or its dependencies, they must derive a new image or rebuild it altogether. This unique image will have a new hash as an identifier, and developers can optionally associate that with a more meaningful tag. Industry leaders were not satisfied with the progress made in immutability, and sought to further extend this tooling with notarization. Notarization ensures the original publisher and their images, complete with their applications and their respective dependencies, use public key infrastructure. Standards authors focused on registry enhancements are intentionally baking into the newest revision of the notarization standard the ability to attest to SBOM metadata to declare all software and dependencies (see the *Standardization of Tooling* section for more details on the progression of SBOM). The present and future, of this mass adoption of build determinism, and the attestation process, significantly minimizes assumptions surrounding artifact analysis and enhances the ability to cross-reference the origin of image components as well, including dependencies. The compositional nature of containerized images allows application developers to build upon notarized layers, while tooling provides the ability to scan and tokenize their own resulting container. These advancements underlie two trends: This shift further left enables faster feedback in the development cycle to answer and potentially provide runtime analysis and confirm if specific vulnerable parts of the dependency are used in an exploitable way.

Static vs. Dynamic Scanning

Until recently, almost all of the tools used for dependency management rely on some form of static analysis: assessing the application without executing it. Nearly all common application runtimes are provided code from source control management tools like Git, including the manifest listing the installed dependencies. Not requiring a running application to complete the scan reduces complexity, but the method falls short in two dimensions. It does not provide evidence which version of code is actively deployed on the server. Nor does it provide an actionable indication that the scanned code actually uses the declared dependencies in a vulnerable way. The latter is incredibly difficult with static analysis alone, and only recently modern dependency management tools like Snyk are developing integrations to hook into the applications at runtime, using dynamic analysis to verify if security upgrades are urgent where actual exploitable behavior is confirmed. With more dependencies and more code at each software layer, this last factor is incredibly useful in prioritizing dependency management. Stakeholders we interviewed accept tooling risk ratings, as is, without adjusting for their own priorities. Merging all updates as soon as possible; exceptions are either not allowed or not considered, as the time required to analyze applicability or gauge risk outweighs the cost of deployment and testing. Tooling that reduces the effort required to analyze applicability and risk would be ideal, but is currently limited to Snyk. Barriers to Snyk adoption include a limited set of application runtime support, developers' requirement to include Snyk's runtime agent as an additional dependency, and configuring Snyk to make the data points viewable in the user interface.

Standardization of Tooling and Data Interchange

Whether open-source or proprietary, the almost universal need for dependency management has not led to a complete standardization of tools, leading to a revival of regulatory interests in the area of Supply Chain Risk Management. The regulatory pressure drives the software industry's interest in an exchange format for tools and developers alike, enabling reliable agnostic communication. This communication often happens via API, communicating the dependencies they manage reliably with standard formats.

Whether the format is standard-compliant or proprietary, industry adoption matters. One standard format that has seen significant traction, thanks to Github and cloud development platforms, in particular, is the Static Analysis Results Interchange Format (SARIF). This format is a generalized schema for transmitting any static analysis data, as long as it references locations in the source code pertaining to the finding. Prior to Github and other vendors integrating this standard to expand services and tool integrations with no or minimal development, SARIF was a standard from the OASIS organization. Even if its potential applications are vast, the inherent complexity of the standard is diminished. All the same, even less specialized than a standard directly focused on dependency upgrades, industry awareness is very recent, coinciding with Github and others accelerating mass adoption. In the case of a more applicable, relevant standard specific to dependency upgrades, no strong advocate has instigated mass adoption in quite the same way.

There is no shortage of tooling for dependency management, and similarly, no shortage of formats to express metadata and details about these dependencies. This tooling and format sprawl becomes onerous for teams, especially at scale. Advocacy for the Software Bill of Materials (SBOM) standard is a call to mitigate this risk. It is rare for all but the smallest development teams and organizations to have one source code repository, open-source or otherwise. Many free or low-cost tools present information only from a context of one singular repository at a time. Initially, governments and highly regulated industries focused on supply chain management risks, recently, both industry leaders and practitioners have aligned to include dependency management modeling as part of the larger supply chain risk.

Attempts to formalize standards has been made in the decades past, just not with the recent backing of public sector stakeholders, from the TTS supply chain risk management working group to the NTIA SBOM Working Group, and private sector partners with partnerships like those of the Cloudbees offshoot: Continuous Delivery Foundation and its sister organization, IT-CISQ. SBOM, even if revisiting a long-desired goal under the banner of a new name and enthusiasm, does pose immediate challenges. Many stakeholders are debating the multifaceted challenges of industry-wide adoption, some age-old, as experts compare and contrast old interchange formats like SWID against latecomers like SPDX and CycloneDX.

We interviewed vendor teams in the developer tooling and dependency management space. One product manager from Snyk confirmed that, even with the piqued interest from industry partners, the philosophical value of SBOM is well understood; the "one true format" has not

risen above the rest. Moreover, Snyk, and presumably other vendors, do not have a critical mass of customers demanding SBOM as a feature that will force their hand. A review of industry materials, commentary, open-source and proprietary tooling, suggests that others hold this view as well. Many tools and formats exist, the calls and buy-in for standardization grow stronger by the day, but the grassroots impact is limited. Many tools report dependency information, in many cases to specify upgrades needed, but all with their own quirks, formats, and resulting workflow preferences. Commercially, services with first-class SBOM export features exist for limited interoperability; GSA TTS indicated in late September they will pilot one such service, Ion Channel, pending GSA security approval. Dependency Track is one of few open-source applications to manage dependency upgrades with first-class SBOM functionality. Dependency Track's primary author created the CycloneDX standard and regularly chairs NTIA SBOM Workgroup meetings.

The public sector has the potential to accelerate change with its ability to enact regulatory requirements. Academia, in partnership with federal sponsors, has even built a marketplace of tools and samples, with the DHS-funded Software Assurance Marketplace (SWAMP). This effort is currently paused as it exhausted funding, but signals the type of public sector market forces that may drive vendor standardization. The progression of tooling shows we know how to manage upgrades, but as an industry, we are on the cusp of a grand unifying format and process for global communication of this information, at the operational and strategic level.

Operational Level Concerns

The lack of standardization plays out in several key contexts for secure development processes at the operational level. The core need is to be proactive: to understand and manage supply chain risk and perform software composition analysis, even if done under a different name. In siloed environments, operations and maintenance tasks have been disconnected from the rationale that drives them. This is often done in the name of efficiency: tools report, and operators take action based on the report. Enterprise organizations, in the quest for efficiency, do not always view this disconnectedness as a shortcoming. Lack of consistent tooling makes higher-risk situations that much harder. Organizations need to react to a vulnerability discovery while facing continual exploitation until patched. Data and tooling are reaching their maximum potential in both high-urgency incident response and more routine, day-to-day security engineering. Industry tools like Netflix Dispatch use acute events to direct teams' corrective action. Vulnerability management tools like OWASP's DefectDojo, and the MITRE Security Automation Framework normalize security reports into the Heimdall Data Format, allowing it to map to FISMA controls.

Security Compliance Ecosystems

Application dependency management spans multiple dimensions of tooling, development processes, and organizational requirements. The discussion is not limited to the narrow domain of just the dependency management software itself. Broadening the discussion to silo spanning options is important because dependency scanning tool selection is sometimes implicit when choosing a toolset for a broader purpose. Integrating different functional and

operational security compliance tooling becomes increasingly complex at larger and larger groups of developers in a company or agency. Intentionally or not, tool selection decisions are driven by unspoken roles of how organizations chose to build software and manage development teams. Organizations are often shy to spend budget to select a best of breed tool that overlaps functionality with a previously acquired technology, even if the “all in one” tool is mediocre compared to a purpose-built solution. The cost of integrating disparate tools results in selecting integrated suites.

Tooling Conclusion

The above discussion is consistent with our findings in this paper. Organizations of different shapes and sizes can upgrade their dependencies responsibly, and there is plenty of tooling to help accomplish this goal. However, surfacing successful vetted standards beyond their originating development organizations, is still a lofty goal, and many have an impact on the medium-to-long term future of this ambitious goal.