TechNote

Static Analysis and Daml Applications

Contents

1 Executive Summary

Note: This is expected to be read alongside the Digital Asset Security Posture document that describes in detail the Information Security program of the firm and our product and development controls. This is available on https://digitalasset.com/security

Security Testing is an important process during the development of Daml Applications and the Daml products themselves. This TechNote describes Digital Asset approach to source code scanning and the respective responsibilities of Digital Asset and Daml customers / developers with a focus on Static Analysis Security Testing (SAST).

We discuss the following:

- Security of Digital Asset products: Daml language, Daml Connect and Daml Drivers and how it relates to Static Analysis (SAST)
- Customers (and/or partners) developing Daml Solutions, including Daml models, along with automation or “triggers” for handling events, and front-end or other interface / interconnect components.

1.1 Digital Asset Products

For our products, Digital Asset specifically chose languages and technologies that reduce the opportunity for our developers to introduce vulnerabilities into our products and we additionally perform a variety of formal methods analysis, threat modelling and security tests against our products during our design and development lifecycle.

The technology choices include using functional, strongly static typed languages like Haskell and Scala, and application frameworks to protect against common vulnerabilities like OWASP Top 10 SQL Injection, Buffer Overflow, etc. Therefore most of the traditional SAST checks are done at the build compilation stage rather than as a post-processing step. For additional validation, we perform SAST checks against our released products using Veracode and other equivalent tools.

1.2 Customer Daml Applications

Customers developing their own Daml Applications, using a variety of technologies to build front and back end components, can use traditional SAST Tools to scan components written in languages like Java, Javascript, Node, Python, etc.

Daml itself protects against many traditional vulnerabilities through two main main mechanisms: language properties and developer tools. On the language properties front Daml’s ledger model, language features and restrictions, reduced boilerplate code, and the inability for developers to do I/O, network or file access or memory handling, protect against most traditional weaknesses and attack surfaces. On the developer tooling front, Digital Asset provides as part of daml: Connect SDK, some additional tools (Daml Studio, DLint, Daml Script)

to validate Daml models as they are developed, and as part of Continuous Integration (CI), and to look for code quality and Daml language best practice recommendations.

Daml also provides Daml Script to allow business logic to be validated and tested as part of the development process.

2 Background

2.1 Static Analysis Security Testing

Static Analysis Security Testing (SAST) is one of several techniques to look for vulnerabilities in applications as they are developed. SAST is generally a source code based scan, where security tools are used to look for coding patterns that are vulnerable to exploitation. For web applications, the OWASP organization has put together the OWASP Top 10 list of the most common issues that developers may face in writing secure code. These include: **Injection Attacks** - parsing specially crafted strings to try to circumvent processing in backend systems (SQL, NoSQL, LDAP, etc) and obtain unauthorized access; **Broken Authentication** - mishandling authentication and session management allowing attackers to take over identity of another user or elevate privileges; **Sensitive Data Exposure** - Applications not properly controlling access to sensitive or personal data, allowing for fraud or identity theft; **Broken Access Control** - data and command access restrictions are not properly implemented

The full list of OWASP Top is available here: https://owasp.org/www-project-top-ten/

SAST scanning is frequently used because of the nature of specific programming languages. It is too easy for developers to mishandle processing of data or poorly implement security, and leave their systems open to attack. However, unless optimised and their rulesets maintained, SAST tools generally produce a low signal-to-noise feedback requiring a lot of manual review and validation and ongoing optimization.

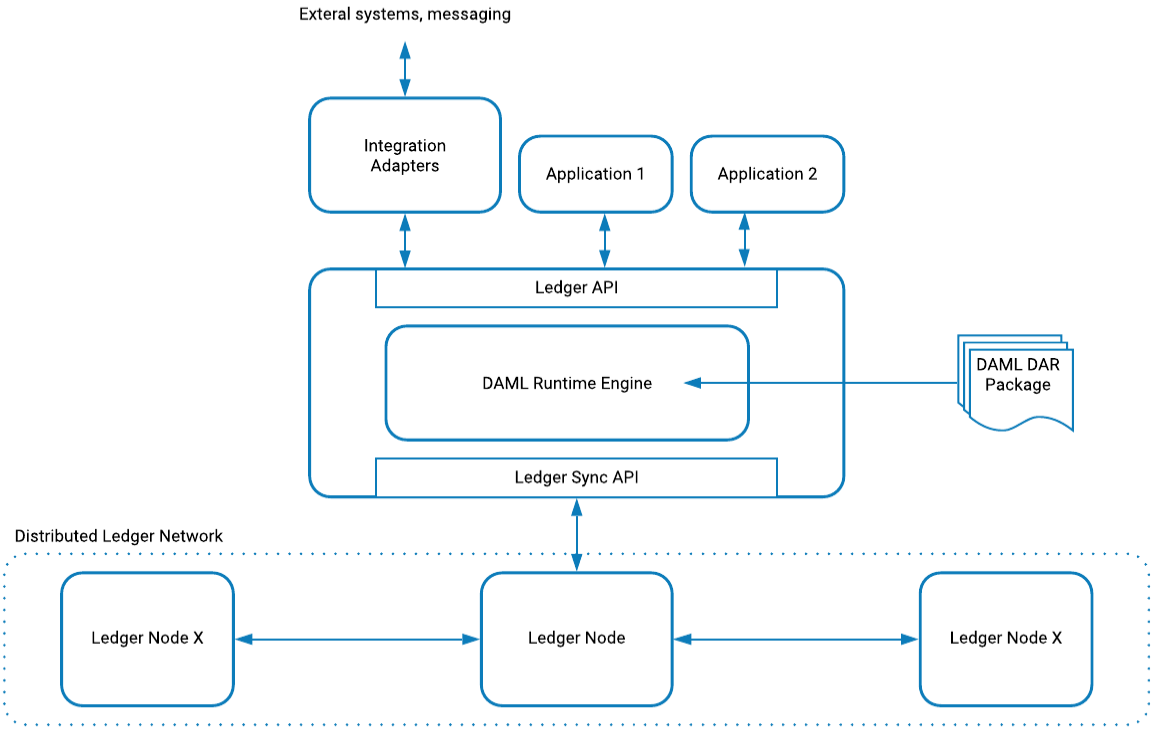
2.2 Components of an Application

A Daml Application frequently consists of:

- the data model and workflows described in Daml, and Daml based event handlers, a.k.a. Triggers, and scenario based testing through Daml Script.
- a collection of additional components or programs to interface with external users and systems via front-ends, messaging or other connectors. These are usually written in other programming languages, like Java, Javascript, C, etc

The front-end, triggers and connectors move the state of the Daml workflow forward based on events - a user submitting a request, a message containing instructions, etc.

Therefore a Daml application typically contains a mixture of traditional development languages (Java, Python, Node JS, Javascript, Typescript, etc) and the Daml source code compiled and packaged into DARs (Daml Archives).



In the following sections we look at:

- How does Daml (the language) protect against a wide variety of traditional sources of code vulnerabilities?
- Digital Asset position on static analysis and the security of its products.
- What static testing should be done for the Daml Application by developers.

3 The Daml Language & Static Analysis

Questions we are often asked include: *"How do I scan my Daml code through a SAST Tool?", "How can my Security Team see the results of the SAST scanner in our CI/CD pipeline?"*

The open-source Daml language is a purpose built Domain Specific Language (DSL) with the specific focus on allowing developers to focus on describing their business processes, data model, workflow actions and associated privileges with minimal overhead, abstracting away much of the underlying boilerplate code and specifics of the underlying infrastructure, persistence mechanisms (database or DLT), the identity management systems and other traditional enterprise application concerns.

Daml takes many of the key features of lambda-calculus and strongly static typed languages like Haskell, and excludes most of the traditional sources of vulnerabilities - I/O access , network or file access, memory handling, authentication, etc. It is not possible to do these types of actions with the Daml code. Additionally, a large amount of boilerplate code, often required for languages like Java and Python and other smart contract languages, is no longer necessary and consequently no longer a source of mistakes.

Daml (the “surface” language, i.e. what a developer sees) is compiled into an intermediate format (DAML-LF) using the Daml compiler. Both Daml itself and Daml-LF have a strong type system and restricted side-effects.

Daml also explicitly exposes an authorization model for access to data and actions on Daml templates. This makes it harder for developers to forget to enforce access permissions.

Daml has a formally specified and published domain model (https://docs.daml.com/concepts/ledger-model/index.html), which has been presented at leading formal methods conferences, and a formally specified and published type system

These features mean that much of the checking of the Daml code is done during development by the compiler and therefore no longer requires post-processing by a SAST scanner. A Daml program will simply not compile if type errors are made or if code does not adhere to the Daml Domain Model.

3.1 Daml Static Analysis (Daml Studio, DLint)

Daml Studio is the user-friendly developer environment for Daml Templates and Packages, and is provided as part of the Daml Connect SDK. It is based on the open source Microsoft VS Code IDE and provides Daml-specific extensions for workflow development and testing. Daml Studio performs live checks against the code being developed, and the scenario testing capability of Daml Script allows the developer to test the model as part of local development.

Also part of the Daml Studio, Digital Asset provides a consistently enhanced and configurable static analysis tool and linter - DLint - for the Daml Language. Daml DLint builds on the capabilities of the Haskell equivalent – HLint – a tool that has been in use and with ongoing upgrades for 15+ years. DLint runs alongside the developer as they code their business workflow logic and scenarios, and it highlights a variety of best practices, including:

- DA Recommended Daml Language Best Practices.
- Lambda, Monad, and Recursion handling recommendations.
- “Code smells” - In computer programming, a code smell is any characteristic in the source code of a program that possibly indicates a deeper problem.
- Reducing code duplication and better structure for templates.
- Static detection of potential runtime errors.

DLint can be run as a standalone tool as part of the security and quality checks of a Continuous Integration (CI) build pipeline.

Digital Asset continues to enhance this capability as best practices, potential language traps, and recommendations evolve. Digital Asset also works with our partners to provide reference application examples, highlighting greatest efficiencies, best practices, and optimized transaction patterns.

4 SAST Testing of Digital Asset Products

Another question we receive is *"How does Digital Asset perform SAST scanning with their products?"*

Digital Asset ensures the security of our products through a variety of mechanisms. These include:

- Formal analysis and review of the language and protocols
- Choice of technologies used to implement our products
- Static Analysis scans using Veracode SAST, Haskell HLint and Daml DLint
- Suite of security tests run during development and build
- Security testing and third party audits and penetration tests
- Open-source of core Daml engine code for external review
- Build pipeline security controls and artifact provenance

Digital Asset selected technologies to implement the Daml language and runtime components that inherently make it easier for our developers to ensure the security of the products. These include:

- Strongly static typed languages, including Haskell and Scala
- Google grpc and protobuf (typed wire protocol and serialization frameworks that reduce need for developers to write this)
- Database access libraries which prevent SQL injection, such as doobie, anorm, and slick

The Daml Drivers and other runtime components (for example Daml Connect: JSON API Server, Auth Middleware, etc) are written in Scala, and run on the Java Virtual Machine (JVM). We further restrict the allowable features of the Scala language through compiler extensions like WartRemover.

As part of Digital Asset's Secure Software Development Life Cycle (S-SDLC), Digital Asset runs Veracode SAST JVM bytecode scanner against production releases of the code

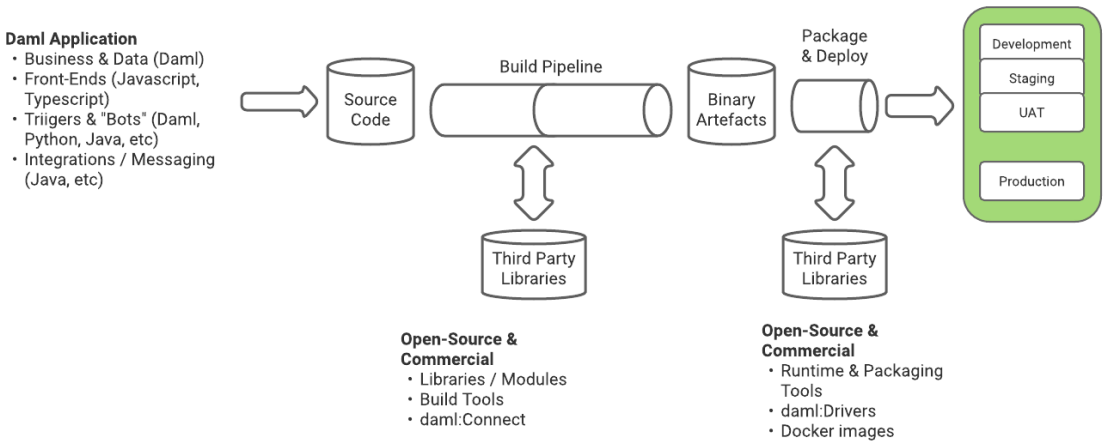
Daml Language and Connect/Driver Community Editions are open source under the Apache2 license. As such all code, including the CI pipeline configuration, can be audited at https://github.com/digital-asset/daml . The compiler and the Daml language are available for external researchers to review and analyse.

5 Security Testing for Customer Daml Applications

"What does a customer need to consider when developing a full Daml Application?"

A full Daml Application will consist of more than just Daml code and it is still recommended to perform static analysis scans against the non-Daml code. We would recommend that customers follow industry best practices around:

- Static analysis (SAST) of non-Daml code through scanners[1], like CodeDx, Coverity, Veracode, eslint (Javascript, Typescript, etc), Fortify
- Software Composition Analysis (SCA) of all dependent libraries and modules and other packaging and runtime components (e.g. Docker containers, Kubernetes config, cloud controls).
- Dynamic Analysis (DAST) and Penetration Tests of the final business application
- Functional and non-functional testing of the business requirements and flows, including approvals and oversight.
- Implement regular testing as part of your CI/CD pipeline ("Shift Left")
- Validate all I/O and data handling for components that interact with a Daml Ledger.



[1] Digital Asset does not specify a particular vendor or product for this. Your company may have specific preferences or requirements for the types of tests they do.