Secure Development and Cyber Security

This page contains important information regarding secure (software) development standards and preparing for a cyber security testing by the Cyber Security One Red Team (GE Digital, Israel)

Software Product Security and Secure Development Standard

The link below is a site with resources that explains the Secure (Software) Development Lifecycle (SDLC) and it contains the important Secure Development Standard

https://spscoe.ge.com/

Software Development Lifecycle (SDLC) V-Model

The figure below is taken from GE Aviation's Product Security Procedure. It should be used as a reference of how cyber security risk assessments and solutions should work from a requirement definition to a tested solution.

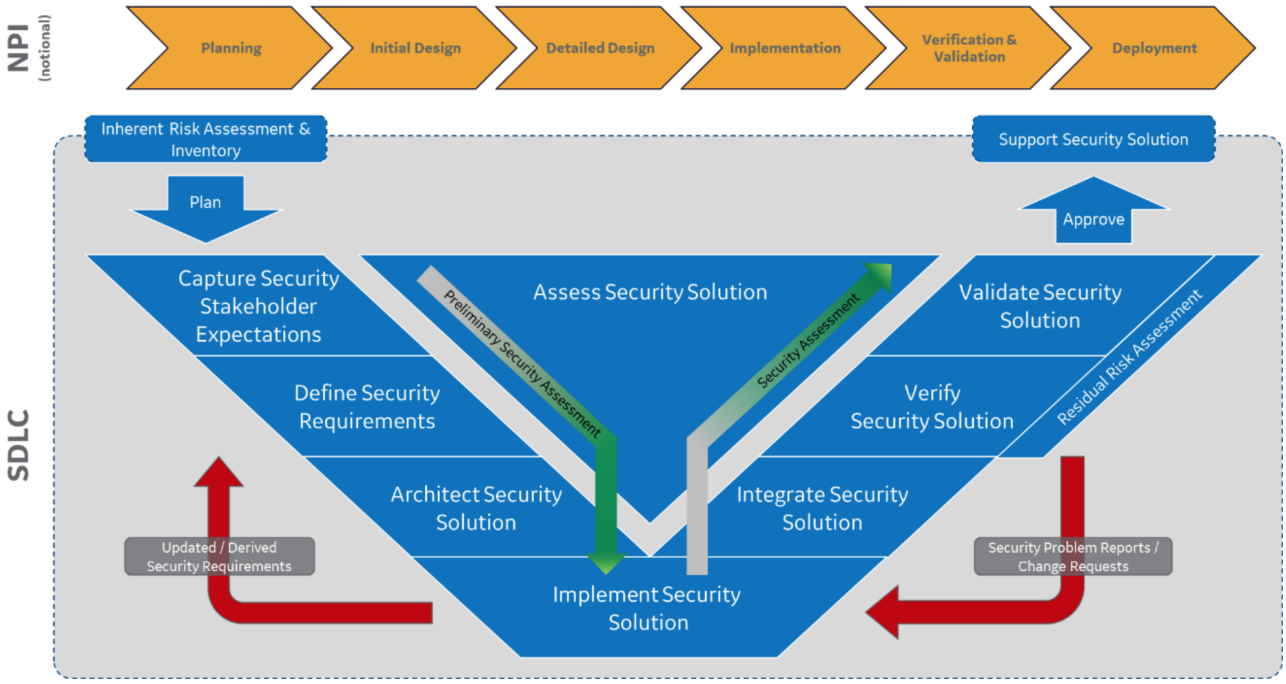

Figure 2 - SDLC and a generalized NPI process

Cyber Security One Red Team (GE Digital, Israel)

This template is a questionnaire to help the Cyber Security One Red Team better understand the product we are creating and in turn help us prepare for the One Red Team penetration testing required before our official release - CyberSecurity_Assessment_Questionnaire_Template 2018.docx

Working copy is on Box here... https://ge.box.com/s/84slrmyi8nmsb56dck3nmewd8707f5cj

For an initial One Red Team (ORT) review, the team must provide the following information:

1. General project release inforamtion:
    a. **Project Name**: GE Additive Build Prep Software
    b. **Project Go-Live Date**: March 31, 2019 (tentative)
    c. **Priority:** High
    d. **Priority rationalization:** Additive is a high priority for GE. This is a new product that is being created by implementing new, differentiating technology and integrating existing technology from more than one existing product. The team(s) vary in commercial software experience. We need help to align the teams regarding software security.
    e. **Technical Contact:** Bart Ingleston

f. **Security Champion:** Lars Bruns
g. **Requested Start Date for Pen Test:** January 31, 2019 (tentative)
h. **PentTest Environment Refresh Date:** February 11, 2019 (tentative)
i. **Code Freeze Date:** Feature Freeze, December 15, 2018 (tentative)
j. **Deploy to Dev Date**: Continuous, yet, still working on official dev deployment environment that you can test.

2. Fill out Cyber Security Assessment Questionnaire.
3. Send an email to OneRedTeam.Digital@ge.com with all the information above.

The One Red Team will schedule and initial review.

The review will provide information and insights that will help us implement our software product more securely and will help the team prepare for a One Red Team penetration test.

Notes from Cyber Security Red Team online session September 5, 2018

Work in progress

Session attendees: Jonathan (Yehonatan) Ariel, Idan Ben Ari, Bart Ingleston, Tom Citriniti, Mike Kinstrey

Preliminary security findings

1. **Lack of an authorization mechanism for the system endpoints**
   a. **Description:** Currently, the system REST endpoints only verify that every incoming request is authenticated, while they do not verify that the party in the possession of the token is indeed authorized to consume the endpoint.
   b. **Implications:**

Anyone who is able to obtain a valid token issued by the UAA, can consume any of the system endpoints. For example:

i. External users can access internal REST endpoints directly, and consume them.

ii. Any internal component (such as any local compute) can impersonate any other component in the system. This means that a malicious / infected component is not restricted in terms of the potential damage it can cause.

Notice: it is very probably that users / attackers be exposed to a token issued by the UAA, either by design / by conducting an attack (such as an “open redirect attack”, which we’ll discuss in a future meeting) or as a result of a faulty configuration of the UAA and its oAuth clients.

a. **Recommendation**:

i. Enforce a proper authorization mechanism, based on the least privilege principal, that will ensure that every party can only consume endpoints it is authorized to consume. The solution can be based either on a third party solution, or on the UAA infrastructure – setting proper claims for user and oAuth clients, so tokens will contain its necessary permissions.

ii. In addition, it is recommended to enforce proper network restrictions, that will prevent users from accessing internal endpoints. Notice that on its own this recommendation is not sufficient, however it can be used as an additional security layer.

1. **Lack of network restrictions / proper segmentation**
   a. **Description:** Any internal BuildPrep component has network access to all other internal components, allowing for a compromised component to interact with all other internal components, thus increase the overall attack surface.
   b. **Recommendation:** Implement strict network level controls and allow internal communication only between components that need to communicate and only on the required ports. This includes, but not limited to, the various local computes, the web server and the queues. Notice that all other communication should be blocked by default.
2. **Usage of insecure protocols**

   a. **Description:** Within the buildPrep network, some insecure protocols are in use. Specifically, the local computes interact with internal services on top of insecure channels such as HTTP.
   b. **Implications:** An internal attacker may carry on several types of attacks, such as

      i. Stealing sensitive technical information, both technical data (such as access tokens) and IP.

      ii. Tampering with the information sent on the channel.

   a. **Recommendation:**

      i. Ensure that secure protocols (such as HTTPS) are in use for all internal as well as external communication. This includes, but not limited to:

1. Communication channels used by the local computes to communicate with the services

2. Communication channels with the queues (should be TLS-enabled)

1. P**rint Job submission security not designed yet**

c. **Description:** The mechanism responsible for sending print jobs to the printers is not designed yet. This one is not a “security vulnerability” per-say as it was not designed yet, but we wanted to point this one out, because this is a sensitive and error-prone interface.

d. **Recommendation:**

a. The “job submission service” should establish trust with the printer before sending out print jobs (preferably using certificates). If OPCUA is supported by the printers, then consider using its built-it certificate based encryption to perform this task.

b. Ensure the communication is done on top of a secure channel.

c. Ensure credentials used by each printer is unique (no usage of identical passwords, keys, secrets, etc).

Suggested developments tools tool to improve development process security:

We suggest that you guys use the following security tools as part of the development process:

1. *Findbugs* configured with *FindSecurityBugs* –
   a. These tools can statically identify many security bugs in Java code. They can be easily added to common IDES as plugins, as well as embedded to the Build process.
2. OWASP Dependency Check - a utility that identifies project dependencies (vulnerable Jar files) and checks if there are any known, publicly disclosed, vulnerabilities. We recommend to run this on a regular basis, in order to identify 3rd-party security problems that might arise in the future.

Additional suggestions of action items to be taken the BuildPrep team:

1. Identify all of the places in which credentials are in used, and ensure that:
   a. All of the credentials are generated per installation, and unique per instance. In order to deal with cross-sites attacks (i.e., client A attacking client B), We need to ensure that default credentials, even complex ones, are not in use.
   b. All of the credentials adhere to a strong password policy.

Some examples of entries of this list would be:

i. Users

ii. oAuth clients

iii. Datasets

iv. Management interfaces of various components (web, databases, queues, etc.)

v. Infrastructure (VPN/Router/SSH) etc.

1. Gather hardening guides for all of the relevant technologies, and make sure to embed the hardening process as part of the installation process. We need to ensure that every component (be it Linux / Postgres / UAA any other component) is set up in a secure way, which includes:

   i. Using up-to-date versions, that are not vulnerable to known attacks (CVEs)

   ii. Configured in a secure way (getting rid of default users , redundant functionality, etc)

   iii. Configured according to the least privileged principle (for example, accounts that run the services are not system / root account).

*We’ll be able to help in gathering the guides once we get the software repository (see next bullet).

1. Enhance the user-facing web interface with relevant HTTP security headers. This is a quick win, as these headers, which are served as “security tips” to the browsers, can help mitigating various kind of attacks.

Requests for information in preparation for the next meetings:

a. It will be very helpful if we could get the following, or some of it, prior to the next meeting:

   i. A complete list of software repository and their versions. Which includes:

      1. Frameworks (such as Spring / Spring boot)
      2. Servers (as as nginx)
      3. Databases and queues (such as Postgres)
      4. OS

   ii. Example of pairs of actual HTTP requests + responses, for the connections below. This can be captured by any proxy (such as *Burp)* – please include the requests and the responses in their entirety (including all of the headers)

1. User -> web server
2. Web server -> project service
3. Project service -> web service (or vice versa)
4. Local computes (any of them) -> project service
5. Any other pair that we might missed out

iii. Example of content of queue entries (for one or more local compute).

iv. Configuration files of the main components (such as the web server)

Suggested topics for next meetings

1. oAuth flow + UAA integration (grant types, authorization, clients, redirect, etc)
2. Web security (such as OWASP top 10)
3. PKI (digital certificates)
4. Licensing architecture and flows
5. Communication with printer (the “Print Job Submission” service)
6. Encrypting CAD designs
7. Discussing the functionality of the web based Java services (Workflow-svc,project-svc and materials-svc)
8. Discussing the functionality of the local computes
9. The process of setting up a new site
10. VM / Container hardening
11. CAD tool
12. Using IDS / IPS

Notes from Rapid Security Assessment sessions with Red Team in Herzliya, Israel October 2 - October 4, 2018

Work in progress

Session attendees throughout the week: Bart Ingleston, David Smith, Amit Kaplan, Michael Reizelman, Dor Dali-Levy, Tom Citriniti, Mike Kinstrey, Gregg Steuben

Threat Models

We will not focus security features, we will apply security design principles and implementations to the threats.

Focus on the Threats

Build Prep top threats (10/2/2018)

- Stealing design and data
- Tamper with parameters and design
- Digital Rights (e.g. instancing)

REST API

- Apply 2-way SSL for both external and internal service requests
  - For external request, obtained and apply a trusted Certificate Authority issued certificate (.cer).
  - For internal requests (e.g. call between Nucleus platform services), use self-signed certificates.
  - IMPORTANT: for each deployment of build prep, generate new certificates. Do NOT re-use the same certificates across different deployments of the platform.
  - Authentication using OAuth2 and JWT (via UAA), create a separate client and secret that can be used to obtain tokens that is different than the client used for obtaining tokens from the web app or CAD tool. The scopes on this “services” client can have different scope and possibly do more.
  - Validate API parameters. Update the description on Confluence to include specific tests.
    - File naming – use REGEX to make sure the file name does not start with “..” (Path Traversing attach)
    - Apply access controls to the API endpoints. Certain endpoints should only be executable by authorized users. We can accomplish this by using better defined Spring Security Config filters (in the implementation) and defining more scopes in our UAA.
      - Example: GET request with scope of “read”
      - Example: POST request with scope of “write”
      - Example: PUT, PATCH, DELETE only by scope of “edit”
      - Example: certain endpoints regardless of HTTP method, has to have a scope of “admin”. Endpoints generated by Spring Boot Web Starter, e.g. /error, or by Srping Boot Actuator, e.g /health
      - No matter the language, use a security framework, use the latest version, and subscript to updates and alerts

OWASP Top 10

Read and know the OWASP top 10. These are known attacks and best practices

- Create API trace logging. Grab user and log the action being performed. This will help with debugging and auditing.

OAuth2, JWT, and UAA

For the web app, we need to implement the OAuth2 user authentication flow.

Use UAA to get a “code” first on the user’s behalf.

“Code” will be redirected to web app server (app.js) and the code will be used to obtain a valid bearer token (JWT).

The token is then captured by the app server (app.js) and it can be associated with the user in the user session and never sent back to the web browser.

Implement a reverse proxy where the token is added to all request header.

See the Predix Web App starter implemention.

Docker

NOTE: get pitches/material from Dor

Read Docker Security section in Docker official documentation

Docker does not come with security features. Docker, by default, doesn’t apply any restrictions on the container’s default user’s (root) access rights to commands or the running host operating system, but does provide ways to config containers to apply restrictions, create users with lower access rights, the number of processes a container can start, and whitelist the commands the user can execute.

Docker container practices

- Create a new user group and a new user in that group and make that user the default user in the container.
- Use a tool that help configure the container. Tools can be specific to the host OS
    - Debian, Ubuntu: AppAmour
    - Red Hat, CentOS: Selinux
    - gVisor: Google middleware that implements common commands for the continer without having the OS run those commands, e.g. file open

Kubernetes for Container Orchestration

NOTE: get pitches/material from Ami

Kubernetes documentation is lacking on configuration settings for their app descriptors.

Kubernetes, similar to Docker, by default, doesn’t apply any restrictions on Deployments, Pods, Ingresses. Need to set these restrictions on deployment of apps.