TECHNICAL WHITEPAPER

Secure Development Lifecycle

August 2016

PREDIX

Secure Development Lifecycle

Contents

Secure Development Lifecycle Overview	4
Secure Development Lifecycle Tracks	5
Developer Security Training	6
Design/Architecture Review	7
Threat Modeling	15
Security User Stories/Security Requirements	17
Automated Dynamic Application Security Testing (DAST)	19
Automated Static Application Security Testing (SAST)	20
Open Source Software (OSS) Vulnerability Assessment	21
Penetration Testing/Assessment	22

Secure Development Lifecycle Overview

Purpose

This framework establishes Secure Development Lifecycle (SDL) guidelines for Customers, Partners/ISVs, and Developers. The framework establishes a set of requirements and direction for product safety, quality and reliability, with the goal of reducing security risk exposure for GE Digital – Predix platform and its ecosystem of products and services.

Scope

This framework applies to all software products and services developed by Customers, Partners/ISVs, and Developers. It also applies to related integration efforts involving commercial or open source software. Customers, Partners/ISVs, and Developers must have resources available to them in order to follow the SDL prior to publishing an application or service within Predix or as a Catalog Tile. Predix is working on making subscription based services available to support and facilitate this process.

Secure Development Lifecycle Tracks

The Predix Secure Development Lifecycle follows traditional "SDL for Agile" frameworks, with a few notable exceptions to gear it more towards development for the Industrial Internet. Not every task in the Agile process is represented as an SDL track in this procedure, but every track corresponds to a set of activities in the Agile process.

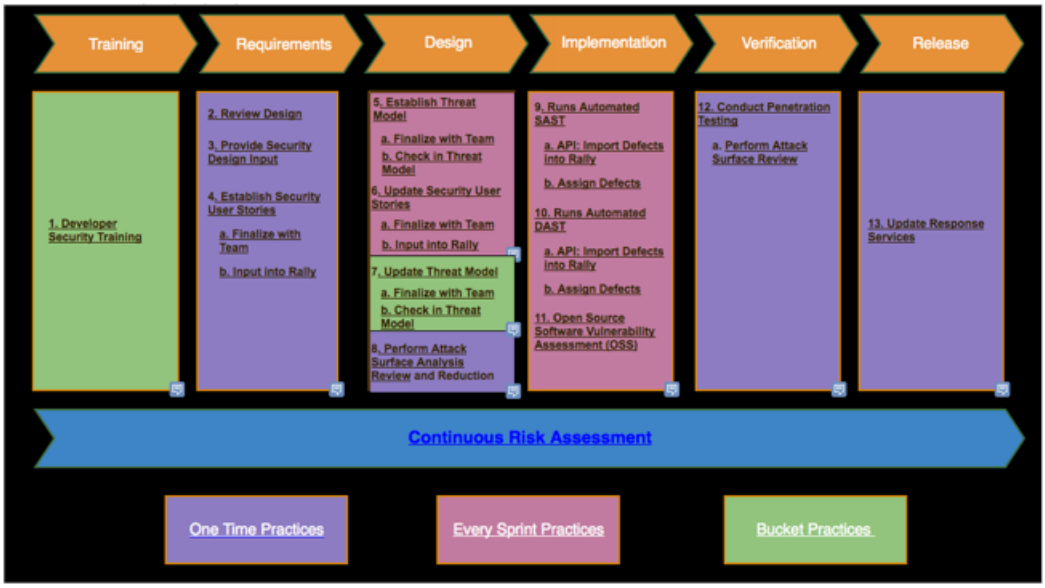

Figure 1: SDL Tracks

The following Tracks are integral to SDL implementation, and each is explained in greater detail in focused sections further in this online artifact.

- Developer Security Training – Ongoing courses provided to developers in order to improve their understanding of techniques for identifying and mitigating security vulnerabilities. Training will focus on topics including threat modeling, DAST testing, and coding techniques to prevent common defects such as SQL injection.
- Design/Architecture Review – A collaborative effort between the ISV Customer Development/Engineering teams and their own product Security group to assess and develop application or service design patterns that mitigate risk to the platform and associated applications and services
- Threat Modeling – a structured approach for analyzing the security of an application, with special consideration for boundaries between logical system components, which often communicate across one or more networks.
- Security User Stories / Security Requirements – a description of functional and non-functional attributes of a software product and its environment which must be in place to prevent security vulnerabilities. Security user stories/requirements are written in the style of a functional user story/requirement.
- Automated Dynamic Application Security Testing (DAST) - a process of testing an application or software product in an operating state, implemented by a web application security scanner
- Automated Static Application Security Testing (SAST) – a process of testing an application or software product in a non-operating state, analyzing the source code for common security vulnerabilities
- Red Team Penetration Testing – Hands-on security testing of a runtime system. This sort of testing uncovers more complex security flaws that may not be caught by DAST or SAST tools.

Note: Traditional "SDL for Agile" does not require automated DAST or SAST, but the general industry trend is toward automation, both to promote consistent usage and to improve efficiency. This is critical for developing on Predix and Industrial Internet applications and services.

Developer Security Training

Developer security training is foundational to all the security tracks highlighted in this procedure. Without this training, together with experience and a security mindset, it will not be possible to do threat modeling, write accurate security user stories, or evaluate SAST / DAST results.

There are a number of paid and free resources available to Customers, Partners/ISVs, and Developers. Enterprises typically have their own SLD curriculums or training courses. Others can subscribe to online services while everyone can benefit from free training provided by such organizations as SAFECode:

https://training.safecode.org/

Design/Architecture Review

An application or service developed by a Customer, Partner/ISV, or Developer must produce architecture and design that has taken into account security considerations to help improve the overall Predix security posture so that everyone can benefit from it. The cost and effort of retrofitting security after development are too high. An architecture and design review helps Development and Engineering teams validate the security-related design features of their application or service before starting the development phase. This allows ISV/Customers to identify and fix potential vulnerabilities before they can be exploited and before the fix requires a substantial reengineering effort.

The architecture and design review process analyzes the architecture and design from a Cyber Security perspective. Ideally the design will commence with assistance from the ISV/Customer product security teams or other experts at the concept phase. If design artifacts already exist, it should facilitate this process nonetheless. Despite the comprehensive nature of the design documentation, the Development and Engineering teams must be able to decompose the application and be able to identify key items, including trust boundaries, data flow, entry points, and privileged code. The physical deployment configuration of the application must also be known. Special attention must be given to the design approaches that have been adapted for those areas that most commonly exhibit vulnerabilities. The guidance provided here will help ISV/Customer develop and secure products and services for consumption within Predix.

For further information on this subject, please refer to below table:

- General Design Principles for Secure Software Development
- Securing Web Application
- Secure Session Management
- Transport Layer Protection
- Securing Password
- Mobile Application Security

Table 1: General Design Principles for Secure Software Development

General Principle	Key Practices	Benefits	Examples and Practices
Minimize the number of high-consequence targets	Principle of least privilege	Minimizes the number of actors in the system that are granted high levels of privilege and the amount of time any actor can hold onto its privileges.	In a traditional Web portal application, the end user is only allowed to read, post content, and enter data into HTML forms, while the Webmaster has all the permissions.
	Separation of privileges, duties, and roles	Ensures that no single entity (human or software) should have all the privileges required to modify, delete, or destroy the system, components and resources.	Developers should have access to the development and test code/systems; however, they should not have access to the production system. If developers have access to the production system, they could make unauthorized edits that could lead to a broken application or add malicious code for their personal gain. The code needs to go through the appropriate approvals and testing before being deployed

General Principle	Key Practices	Benefits	Examples and Practices
			into production. On the other hand, administrators should be able to deploy the package into production, but should not have the ability to edit the code.
	Separation of domains	Separation of domains makes separation of roles and privileges easier to implement.	Database administrators should not have control over business logic and the application administrator should not have control over the database.
Don't expose vulnerable or high-consequence components	Keep program data, executables, and configuration data separated	Reduces the likelihood that an attacker who gains access to program data will easily locate and gain access to program executables or control/configuration data.	On Unix or Linux systems, the chroot "jail" feature of the standard operating system access controls can be configured to create an isolated execution area for software, thus serving the same purpose as a Java or Perl "sandbox."
	Segregate trusted entities from untrusted entities	Reduces the exposure of the software's high-consequence functions from its high-risk functions, which can be susceptible to attacks.	Java and Perl's sandboxing and .NET's Code Access Security mechanism in its Common Language Runtime (CLR) assigns a level privilege to executables contained within it. This privilege level should be the minimal needed by the function(s) to perform its normal expected operation. If any anomalies occur, the sandbox/CLR will generate an exception and an exception handler will prevent the executable from performing the unexpected operation.
	Minimize the number of entry and exit points	Reduces the attack surface.	Firewalls provide a single point of contact (called a chokepoint) that allows the administrator control of traffic coming into or out of the network. Like a firewall, strive for one entry point into any software entity (function, process, module component) and ideally one exit point.

General Principle	Key Practices	Benefits	Examples and Practices
	Assume environment data is not trustworthy	Reduces the exposure of the software to potentially malicious execution environment components or attacker-intercepted and modified environment data.	Java Platform, Enterprise Edition (Java EE) components run within "contexts" (e.g. System Context, Login Context, Session Context, Naming and Directory Context, etc.) that can be relied on to provide trustworthy environment data at runtime to Java programs.
	Use only trusted interfaces to environment resources	This practice reduces the exposure of the data passed between the software and its environment.	Application-level programs should call only other application-layer programs, middleware, or explicit APIs to system resources. Applications should not use APIs intended for human users rather than software nor rely on a system-level tool (versus an application-level tool) to filter/modify the output.
Deny attackers the means to compromise	Simplify the design	This practice ensures that all attacker actions are observed and recorded, contributing to the ability to recognize and isolate/block the source of attack patterns.	Enforcing accountability with the combination of auditing and non-repudiation measures. Auditing amounts to security-focused event logging to record all security-relevant actions performed by the actor while interacting with the system. Audits are after-the-fact and often can be labor intensive; Security-Enhanced Linux (SELinux) can be used to enforce data access using security labels. Non-repudiation measures, most often a digital signature, bind proof the identity of the actor responsible for modifying the data.
	Hold all actors accountable	This practice ensures that all attacker actions are observed and recorded, contributing to the ability to recognize and isolate/block the source of attack patterns.	Enforcing accountability with the combination of auditing and non-repudiation measures. Auditing amounts to security-focused event logging to record all security-relevant actions performed by the actor while interacting with the system.

General Principle	Key Practices	Benefits	Examples and Practices
			Audits are after-the-fact and often can be labor intensive; Security-Enhanced Linux (SELinux) can be used to enforce data access using security labels. Non-repudiation measures, most often a digital signature, bind proof the identity of the actor responsible for modifying the data.
	Timing, synchronization, and sequencing should be simplified to avoid issues.	Modeling and documenting timing, synchronization, and sequencing issues will reduce the likelihood of race conditions, order dependencies, synchronization problems, and deadlocks.	Whenever possible, make all individual transactions atomic, use multiphase commits for data "writes," use hierarchical locking to prevent simultaneous execution of processes, and reduce time pressures on system processing.
	Make secure states easy to enter and vulnerable states difficult to enter	This practice reduces the likelihood that the software will be allowed to inadvertently enter a vulnerable state.	Software should always begin and end its execution in a secure state.
	Design for controllability	This practice makes it easier to detect attack paths, and disengage the software from its interactions with attackers. Caution should be taken when using this approach since it can open up a whole range of new attack vectors.	To increase the software controllability, design the software to have the ability to self-monitor and limit resource usage, provide exception handling, error handling, anomaly handling, and provide feedback that enables all assumptions and models to be validated before decisions are taken.
	Design for secure failure	Reduces the likelihood that a failure in the software will leave it vulnerable to attack.	Implement watchdog timers to check for the "I'm alive" signals from processes and use exception handling logic to correct actions before a failure can occur.

Securing Web Application—OWASP's Code Review Guide Recommendations

General Principle	Recommendations
Authentication	• Ensure all internal and external connections (user and entity) go through an appropriate and adequate form of authentication. • Ensure all pages enforce the requirement for authentication. • Pass authentication credentials or sensitive information only via HTTP "POST" method, do not accept HTTP "GET" method. • Ensure authentication credentials do no traverse "the wire" in clear text form.
Authorization	• Ensure application has clearly defined the user types and the rights of said users. • Grant only those authorities necessary to perform a given role. • Ensure the authorization mechanisms work properly, fail securely, and cannot be circumvented. • Do not expose privileged accounts and operations externally.
Cookie Management	• Ensure that unauthorized activities cannot take place via cookie manipulation. • Encrypt the entire cookie if it contains sensitive data. • Ensure secure flag is set to prevent accidental transmission over "the wire" in a non-secure manner. The secure flag dictates that the cookie should only be sent over secure means, such as Secure Sockets Layer (SSL). • Do not store private information on cookies. If required, only store what is necessary.
Data/Input Validation	All external inputs should be examined and validated.
Error Handling / Information Leakage	• Ensure the application fails in a secure manner. • Ensure resources are released if an error occurs. • Do not expose system errors to the user.
Logging/Auditing	• Ensure the payload being logged is of a defined maximum length and the logging mechanism enforces that length. • Log both successful and unsuccessful authentication attempts. • Log access to sensitive data files. • Log privilege escalations made in the application. • Do not log sensitive information.
Cryptography	• Ensure the application is implementing known good cryptographic methods. • Do not transmit sensitive data in the clear, internally or externally. • Do not develop custom cryptography.

Session Management

Session Management Issue	Recommendations
Attacker guessing the user's Session ID	Session IDs should be created with the same standards as passwords. This means that the Session ID should be of considerable length and complexity. There should not be any noticeable pattern in the Session IDs that could be used to predict the next ID to be issued.
Attacker stealing the user's Session ID	Session IDs, like all sensitive data, should be transmitted by secure means (such as HTTPS) and stored in a secure location (not publically readable).
Attacker setting a user's Session ID (session fixation)	The application should check that all Session IDs being used were originally distributed by the application server.

Transport Layer Protection

Transport Layer Protection	Recommendations
TLS	• Require TLS for all sensitive pages. Non-TLS requests to these pages should be redirected to the TLS page. • Configure your TLS provider to only support strong (for example, FIPS 140-2 compliant) algorithms. • Backend and other connections should also use TLS or other encryption technologies.
Cookies	Set the 'secure' flag on all sensitive cookies. This will prevent the browser from sending any cookie with the 'secure' flag enabled to any 'HTTP' connections.
Certificates	Ensure your certificate is valid, not expired, not revoked, and matches all domain used by the site.

Securing Password

Password Security	Recommendations
Password Complexity	Enforce password complexity requirements established by policy or regulations. Authentication credentials should be sufficient to withstand attacks that are typical of the threats in the deployed environment. An example of password complexity is requiring alphabetic as well as numeric and/or special characters in the password.
Password Minimum Length	Enforce a minimum length requirement for the password, as established by policy or regulations. OWASP recommends at least eight characters, but sixteen characters or the use of multi-word pass phrases will provide a better solution.
Password Change	Enforce password changes based on requirements established in policy or regulations. Critical systems may require more frequent password changes.

Password Security	Recommendations
Password Reuse	Prevent password re-use. Passwords should be at least one day old before they can be changed.
Failed Login	Disable the account after a certain number of failed login attempts.
Error Messages	Display generic error messages when a user types in an incorrect username or password.
Password Storage	Store passwords in the database as salted hash values.

Mobile Application Security

Password Security	Recommendations
Insecure Data Storage	Store only what is absolutely required; never use public storage areas (e.g. SD cards); leverage secure containers and platform provided file encryption APIs; and do not grant files world readable or world writeable permissions.
Weak Server Side Controls	Understand the additional risks mobile applications can introduce into existing architectures and use the wealth of knowledge already out there (for example, OWASP Web Top 10, Cloud Top 10, Cheat Sheets, Development Guides).
Insufficient Transport Layer Protection	Ensure all sensitive data leaving the device is encrypted; this includes data over carrier networks, Wi-Fi, etc.
Client Side Injection	Sanitize or escape untrusted data before rendering or executing it; use prepared statements for database calls (concatenation is a bad practice); and minimize the sensitive native capabilities tied to hybrid web functionality.
Poor Authorization and Authentication	Contextual information can enhance the authentication process but only as a part of a multi- factor authentication; never use device ID or subscriber ID as a sole authenticator; and authenticate all API calls to paid resources.
Improper Session Handling	Do not use device identifier as a session token. Make users re-authenticate every so often and ensure that tokens can be revoked quickly in the event of a stolen/lost device.
Security Decisions via Untrusted Inputs	Check caller's permissions at input boundaries; prompt the user for additional authorization before allowing consummation of paid resources; when permission checks cannot be performed, ensure additional steps are required to launch sensitive actions.
Side Channel Data Leakage	Understand what third-party libraries in your application are doing with the user data; never log credentials, PII, or other sensitive data to system logs; remove sensitive data before screenshots are taken; before releasing apps, debug them to observe files created, written to, or modified in any way; and test your application across as many platform versions as possible.

Password Security	Recommendations
Broken Cryptography	Encoding, obfuscation, and serialization is not considered encryption. Preventions Tips: Do not store the key with the encrypted data; use what your platform already provides; and do not develop in-house cryptography.
Sensitive Information Disclosure	Do not store the private API keys in the client; keep proprietary and sensitive business logic on the server; and never hardcode the password.

Threat Modeling

Threat Modeling is a structured approach for analyzing the security of a product. Threat Modeling evaluates the various attack surfaces of the product and identifies the vulnerabilities in design, thereby helping the software architect to mitigate the effects of the threat to the system.

Why Should We Do Threat Modeling?

Threat Modeling identifies vulnerabilities in the design phase of software development, making concerns easier and cost-effective to resolve. Also, Threat Modeling ensures that products are developed with built-in security from the beginning.

Development teams are responsible for the security of applications they create and maintain. Each team, in consultation with Cyber Security, keeps a Threat Model of its application, operating environment, and data flow boundaries.

The Threat Model should be reviewed and updated by the ISV/Customer Development/Engineering teams and their product security team(s) at least once prior to each Release.

In order to provide visibility and the ability to audit, Threat Models should be posted in a central repository location, with Role-based Access Control (RBAC).

The benefits of continuous threat modeling include:

- Identify security gaps early on and tackle these in the design phase, when they are cheapest and quickest to address
- Reduce the number of serious, complex defects uncovered during security testing
- Provide visibility across a project, clarifying the need for planning and development efforts that address security
- Provide for a visual representation of system security components, data flows and boundaries
- Maintain an up-to-date risk profile
- Raise awareness across development teams so that security becomes a daily priority, alongside functional development and deployment tasks.

How Should Threat Modeling Be Implemented?

The Threat Modeling process can be classified in three steps:

1. Decompose the System
2. Determine and Rank Threats
3. Determine Countermeasures and Mitigation

1. Decompose the System

   The Threat Modeling process begins with understanding the system and the way it interacts with external entities. Decomposing the system involves:

   - Creating use-cases to understand how the application is used
   - Identifying entry points to see where a potential attacker could interact with the application
   - Identifying assets i.e. items/areas that the attacker would be interested in
   - Identifying trust levels which represent the access rights that the application will grant to external entities.

   The ISV/Customer Development or Engineering team can document the above information in the Threat Model besides using it to produce Data Flow Diagrams (DFDs) for the application. The DFDs show the different paths through the system, highlighting the access privilege boundaries.

2. Determine and Rank Threats

Threats are identified using a threat categorization methodology. A threat categorization such as STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege) can be used to define threat categories such as:

- Auditing & Logging
- Authentication
- Authorization
- Configuration Management
- Data Protection in Storage and Transit
- Data Validation
- Exception Management.

3. Determine Countermeasures and Mitigation

   A lack of protection against a threat might indicate a vulnerability whose risk exposure could be mitigated by implementing a countermeasure. Such countermeasures can be identified using threat-countermeasure mapping lists. Once threats are assigned a risk-rank, sorting them from the highest to lowest risk is possible. Risk-ranking also simplifies prioritizing the mitigation effort and threat response by applying the identified countermeasures.

Are there any tools available for Threat Modeling?

Predix Cyber Security recommends using the Microsoft Threat Modeling Tool 2016, which is available as a free download:

https://www.microsoft.com/en-us/download/details.aspx?id=49168

You need a Windows Machine/Windows VM for installing this tool.

When Should I Develop Threat Models?

Whenever an application or service is being developed within Predix or as a Catalog Tile, create a threat model during the design phase. Whenever a new feature is developed on an existing application or service, the current threat model should be updated to incorporate/eliminate attack surfaces based on the feature being developed. Threat Modeling is a living design artifact for a product. It evolves with the development of a product.

Where do I store the developed threat model?

Predix Cyber Security recommends that you store the Threat Model artifact in a safe and secure location for your consumption and from where it can be made available for Predix Cyber Security team member(s) who may want to review the artifact to meet certain Catalog Tile Service Security Review criteria. Every threat model version should be committed with an appropriate commit-message, to identify the reason for the revision and the release in which the revised design will be implemented.

Security User Stories/Security Requirements

Each application or service being built within Predix or as a Catalog Time must go through some requirements gathering in order to facilitate review of functional security stories and creation of the corresponding security user stories. For instance, if a product calls for an authentication component, then the security user stories for this component would focus on aspects such as account lockout, number of allowed failed login attempts, and password complexity (this is not a comprehensive list).

Third parties review security user stories within their development team and/or SDL partner, makes corrections as appropriate, and then enters these stories into the appropriate issue tracking tool. Security stories are updated as functional requirements change, are added, or are removed.

Note that that there are two broad categories of security user story:

1. Evil user story – Focused on how the hacker will exploit a vulnerability to compromise the security of a system, this sort of story is told from the hacker's point of view.
2. Nonfunctional security user story – Augmenting the functional definition of a product, this sort of story provides security guidance.

The following sample set of security user stories was provided to an actual project team and was useful in securing their product prior to public rollout. Some stories have been removed and others have been modified so as not to reference the original project directly.

This set includes both nonfunctional security user stories and so-called "evil" user stories.

	Scenario	Evil Story	Consequence	Notes
1	Sample Project instances available for public download present a SHA-2 checksum.	As an interested outside party downloading a Sample Project .ova file from Company XYZ, I can validate the SHA-2 hash associated with that .ova file to check authenticity.	1. Valid SHA-2 checksum associated with the given ova available for download is presented on the public download page. 2. MD5 and SHA-1 checksums are not presented because these hash algorithms are no longer collision-resistant.	*Calculating Checksum* Example: $ sha256sum Sample Project3.3.5.ova 47585bd6222948ee6b138 166ec2812092d46a6abbc97 6028799a57egare002d4c
2	Sample Project default password complies with Company XYZ password composition rules.	As an unauthorized user / hacker, I know that the default Sample Project predix user password is "predix," enabling me to access an unattended customer Sample Project instance that has been customized by its owner and where	1. All Sample Project accounts are deployed with default passwords that comply with the Company XYZ password composition rules.	Refer to: Company_XYZ_ AccessManagement_Policy

	Scenario	Evil Story	Consequence	Notes
		sensitive information has been stored.		
3	Account credentials are not cached in Firefox or elsewhere.	As a hacker, I download the public version of Sample Project and open gmail in Firefox. I notice that the credentials to companyxyz, mail to: predixadoption@gmail.com are cached in Firefox. I log in and access a security token posted by another Sample Project user. I send malicious e-mail to other Sample Project users, resulting in information and account compromise.	1. Review of new Sample Project image verifies that "Remember passwords for sites" is unchecked in Firefox and that no account credentials have been cached. 2. phpPgAdmin credentials are no longer cached in Firefox (which led to a hyperion database containing traffic camera information, probably unrelated to Sample Project).	

Automated Dynamic Application Security Testing (DAST)

ISVs, Customers and Developers working with their product security teams or SDL partner, should configure DAST scan profiles in their CI/CD build tool (ex. Jenkins) and in the application security scanner referenced by their build tool. This will help automate the scanning process and streamline DevOps, should the third party be using a DevOps approach to development.

However, if a stand-alone SaaS based DAST is being used, the scanning tool must meet Predix Cyber Security requirements which are currently met by the following SaaS based DAST tool:

Tinfoil Security:*https://www.tinfoilsecurity.com/*

Automated Static Application Security Testing (SAST)

ISVs, Customers and Developers working with their product security teams should configure SAST scan profiles in their CI/CD build tool (ex. Jenkins) and in the SAST scanner their built tool references. This will help automate the scanning process and streamline DevOps, should the ISV/Cusatomer be using a DevOps approach to development.

However, if a stand-alone SAST is being used, the scanning tool must meet Predix Cyber Security requirements. Products that can facilitate your SAST efforts are as follows, but please note Predix Cyber Security does not endorse any of the technologies.

https://www.checkmarx.com/technology/static-code-analysis-sca/

http://www8.hp.com/us/en/software-solutions/static-code-analysis-sast/

http://www.veracode.com/products/binary-static-analysis-sast/

https://scan.coverity.com/

Open Source Software (OSS) Vulnerability Assessment

Fixing software vulnerabilities is a crucial defense against exploitation and potential breach. With Open Source Software, the vulnerability is further magnified, hence the need for OSS assessment and the use of OSS Security Assessment tools.

OSS Security Assessment process involves tools that:

- Identify open source throughout the product code base
- Map known vulnerabilities to the open source in use by the product
- Recommend remediation for identified outstanding vulnerabilities in open-source components

Some SAST products also provide OSS scanning capabilities. Products that can facilitate your OSS efforts are as follows, but please note that Predix Cyber Security does not endorse any of the technologies.
https://www.checkmarx.com/technology/static-code-analysis-sca/

http://www.veracode.com/products/binary-static-analysis-sast/

https://www.blackducksoftware.com/solutions/application-security

Penetration Testing/Assessment

Even with all of the above steps followed closely, there is still a possibility of a defect or vulnerability making it to the final gate prior to production. To that end, pen testing provides a final validation that secure code is going into production. All Customers, Partners/ISVs, and Developers are being recommended to undergo this level of testing prior to final release to production. At the discretion of Predix Cyber Security, some environments within Predix may require a final pen test to be conducted by Predix Cyber Security Red Team.

The assessment method of pen test is defined by the main attack vectors and test scenarios to be carried out based on the information provided by the requestors. The goal of pen test is not to have comprehensive coverage of code scanning, or finding all security defects that the technology has, but rather focus on areas where there may be higher business risks. For example the scenarios may target to identify security defects that may expose critical data, compromise credentials, or have potential reputational impact to the business. A pen-tester will use various scanning tools as well as manual testing to identify security vulnerabilities that are high business impact. The length of each pen test varies based on the size of scope of the technology and the resources assigned.