presented by

insyde®

UEFI Goes to Washington

UEFI Fall 2023 Developers Conference & Plugfest

October 9-12, 2023

Presented by Tim Lewis, CTO of Insyde Software

Agenda

- Why Does The U.S. Government Care About UEFI?
- U.S. Government Initiatives That Affect UEFI Firmware
  - SBOMs
- Call to Action

Why Does The U.S. Government Care?

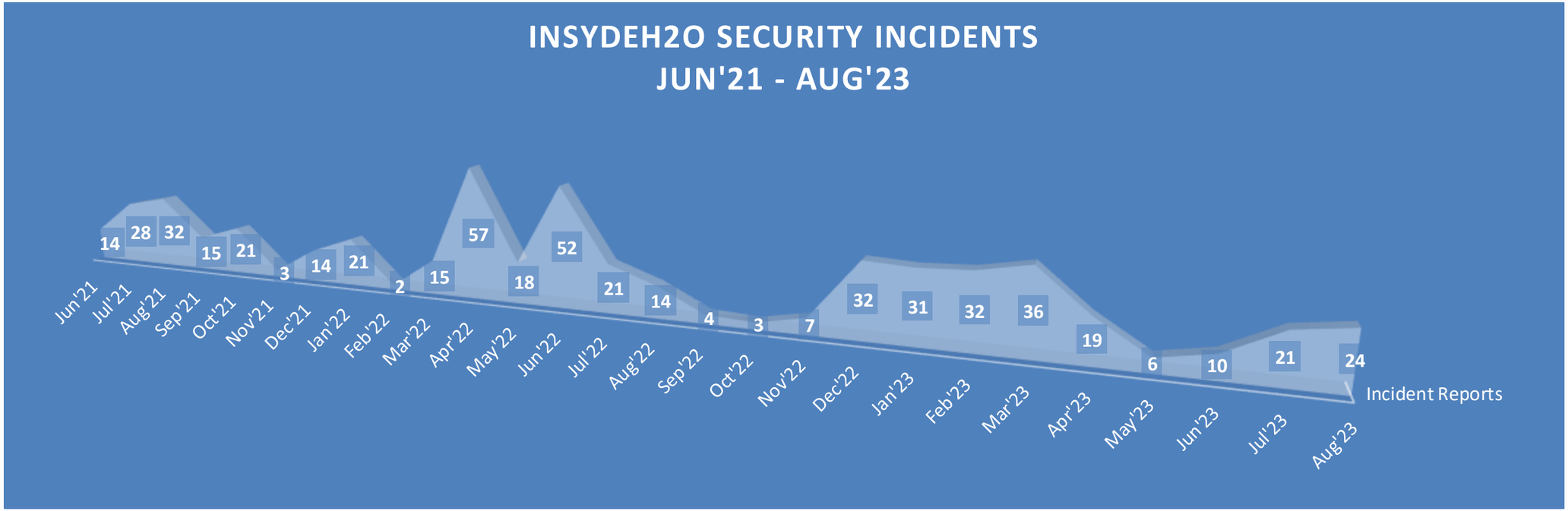
"Firmware presents a large and ever-expanding attack surface…Securing the firmware layer is often overlooked, but it is a single point of failure in devices and is one of the stealthiest methods in which an attacker can compromise devices at scale. Over the past few years, hackers have increasingly targeted firmware to launch devastating attacks."

Assessment of the Critical Supply Chains Supporting the U.S. Information and Communications Technology Infrastructure, U.S. Departments of Commerce and U.S. Department of Homeland Security, February 23, 2022, page 41, https://www.dhs.gov/sites/default/files/2022-02/ICT%20Supply%20Chain%20Report_0.pdf

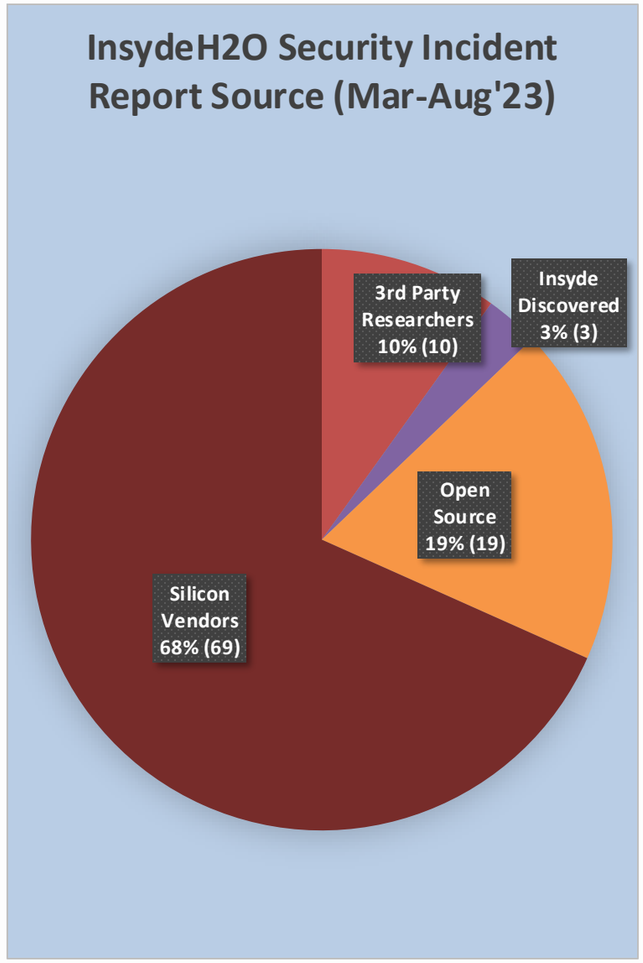
UEFI Firmware Security Incidents

Insyde tracked 544 total security incident reports (SIRs) for UEFI firmware over 2+ year period (20.44/mo).

- Twice-annual peaks are primarily due to silicon vendor security bulletins.

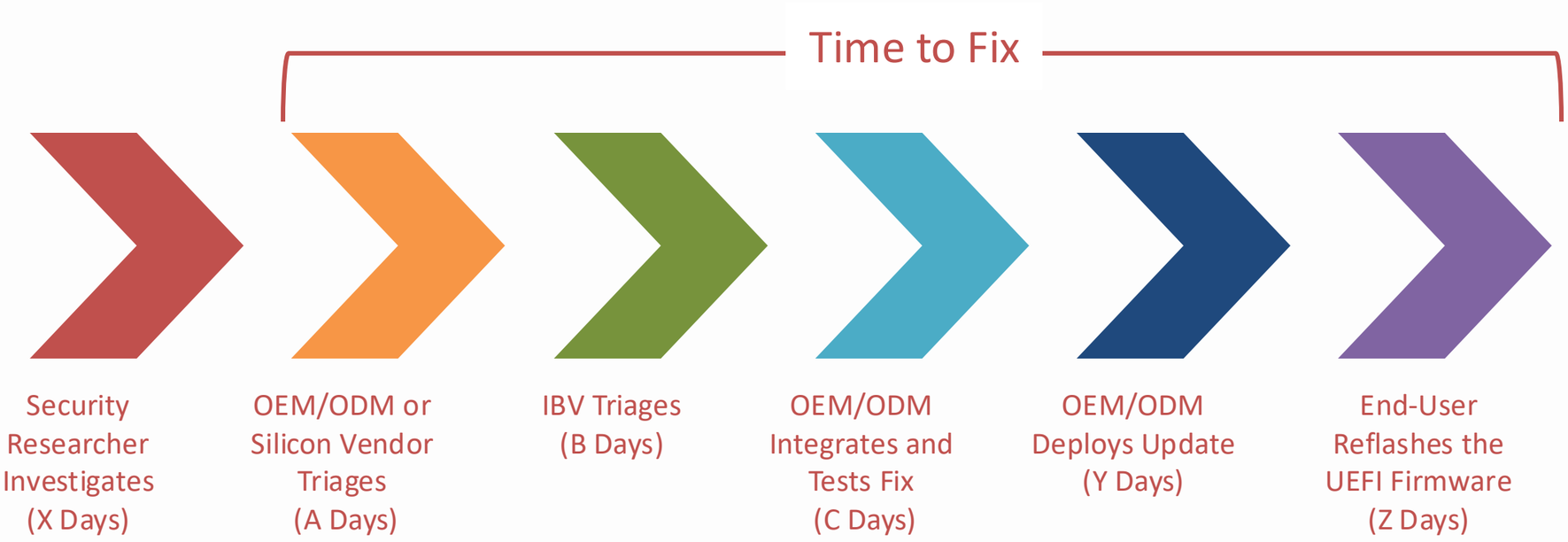


Security Trends – March to August 2023



- Silicon vendor firmware security bugs continue to dominate the day-to-day security issues.
- Bad: Security researchers have found the EDK2 networking stack, and it looks like there will be more.
- Good: 1 out of 13 Insyde issues due to ODM/OEM Feature code, which means Insyde's core fixes have broader reach.
  - UEFI variable issues more likely in chipset and OEM code.
  - Therefore, less attractive to malware authors and security researchers because # of units affected per issue is lower
- Good: Trend toward UEFI variable issues, not SMM issues.

Time To Fix



- Security researcher initial reports typically want disclosure (A+B+C+Y+Z) at 90 days. We can only measure (A+B+C) currently.
- Insyde leaves 30 days for C.
- Insyde is encouraging OEM/ODM to shorten A as much as possible.
- Insyde A+B+C rate is 6.25% at 90 days and 50% at 120 days.

U.S. Government Interest In Securing UEFI

U.S. Government Steps Toward Securing UEFI Deployments

- CISA “A Call to Action: Bolster UEFI Cybersecurity Now”
- NIST SP 800-218 Attestations

CISA's "Bolster UEFI Security Now" Blog

- "Attackers have a clear value proposition for targeting UEFI software" because UEFI subversion can provide malicious software the ability to persist through:
  - System reboot – the malware survives basic defensive actions such as turning the device off and on again.
  - Operating system reinstallation—Malware that persists through reinstallation can evade this standard incident response practice.
  - Partial physical part replacement—A device infected with this level of persistent malware basically needs to be thrown away rather than repaired.

Adapted from https://www.cisa.gov/news-events/news/call-action-bolster-uefi-cybersecurity-now

CISA's "Bolster UEFI Security Now" Blog

- **Cybersecurity & Infrastructure Security Agency** (CISA) charts path to improved UEFI firmware security:
  - Enable system owners to able to audit, manage, and update UEFI components.
  - Enable collecting, analyzing, and responding to event logs that identify UEFI-related activities (e.g., changes, updates, add/remove components).
  - Use secure development environments and adopt software development best practices
  - Adopt uninterrupted and reliable update capabilities
  - Expand adoption of best practices for PSIRT operations.

Adapted from https://www.cisa.gov/news-events/news/call-action-bolster-uefi-cybersecurity-now

End-User & Security Response

End-User & Security Response

- Ultimately, platform security depends on the end user or IT administrator, who is least knowledgeable about firmware security or how to fix it.
- Except in the limited case of flash tampering, most security technologies in UEFI firmware are content with stopping attacks from stealing secrets or compromising integrity **not** responding and repairing.

End-User & Security Response

- What should UEFI firmware do when it detects an active security threat?
- UEFI firmware's answer in most cases: hang the system.
- But end users don't react to "hang" with productive responses – reset, turn off, etc.
- If there is an active security threat, how should UEFI firmware react, report and respond?
  - Insyde's FACT and HP's SureStart and some non-BIOS solutions try to give options: reflash, restart, report to admin, crash dump, etc.

Events	Default EDK2, CPU or Chipset Behavior
BIOS flash modification	Force flash update or hang
Illegal SMM CPU save state access	Hang
SMM call-out	Hang
SMM non-SMRAM modification	Hang
Heap overflow/underflow	Hang
Stack overflow/underflow	Hang
Null-pointer	Undefined or hang
SMRAM modification	Hang
Driver corrupts stack	Undefined
Driver integer overflow/underflow	Undefined
Driver uses uninitialized data.	Undefined
Driver private data structure corruption	Undefined
Driver TPL inversion	Undefined
PCIe Reported Security Events	Ignored

NIST SP 800-218 Attestation

NIST SP 800-218

- Executive Order (EO) 14028 – May 2021
  - NIST SP 800-218 Secure Software Development Framework (SSDF) – Feb 2022
  - OMB (M-22-28 & M-23-16) requires all federal agencies to comply with NIST guidance – Sep 2022
    - No later than September 13, 2023, for all software, "agencies shall collect attestation letters not posted publicly by software providers for all software subject to the requirements of this memorandum."
    - All Federal critical software must comply with NIST guidance – Jun 2023
    - All Federal 3rd party software must comply with NIST guidance – Sep 2023

- Requirements:
  - A self-attestation that the product was built in conformance with NIST's SSDF.
  - On request, a Software Bill of Materials (SBOM) for the product.
  - On request, other artifacts substantiating SSDF conformance, e.g., output of vulnerability scanners, software provenance metadata, etc.
  - On request, evidence of participation in a Vulnerability Disclosure Program.

https://www.nist.gov/itl/executive-order-14028-improving-nations-cybersecurity
https://www.whitehouse.gov/wp-content/uploads/2022/09/M-22-18.pdf
https://www.whitehouse.gov/wp-content/uploads/2023/06/M-23-16-Update-to-M-22-18-Enhancing-Software-Security.pdf

Key Points To NIST SP 800-218

- **PS.3.2:** Collect, safeguard, maintain, and share provenance data for all components of each software release (e.g., in a software bill of materials [**SBOM**]).

- **PW.4.4:** Verify that acquired commercial, open-source, and all other third-party software components comply with the requirements, as defined by the organization, throughout their life cycles. **[Supply Chain]**

- **RV.1.1:** Gather information from software acquirers, users, and public sources on potential vulnerabilities in the software and third-party components that the software uses and investigate all credible reports. **[PSIRT]**

NIST SP 800-218 Attestation Form

The software is developed and built in secure environments. Those environments are secured by the following actions, at a minimum:

1. Separating and protecting each environment involved in developing and building software;

   Regularly logging, monitoring, and auditing trust relationships used for authorization and access: (i) to any software development and build environments; and (ii) among components within each environment;

   Enforcing multi-factor authentication and conditional access across the environments relevant to developing and building software in a manner that minimized security risk;

   Taking consistent and reasonable steps to document as well as minimize use or inclusion of software products that create undue risk within the environments used to develop and build software;

   Encrypting sensitive data, such as credentials, to the extent practicable and based on risk;

   Implementing defensive cyber security practices, including continuous monitoring of operations and alerts and, as necessary, responding to suspected and confirmed cyber incidents;

NIST SP 800-218 Attestation Form

2. The software producer has made a good-faith effort to maintain trusted source code supply chains by: (a) Employing automated tools or comparable processes; and (b) Establishing a process that includes reasonable steps to address the security of third-party components and manage related vulnerabilities;

3. The software producer employs automated tools or comparable processes in a good-faith effort to maintain trusted source code supply chains;

4. The software producer maintains provenance data for internal and third-party code incorporated into the software;

5. The software producer employs automated tools or comparable processes that check for security vulnerabilities. In addition: (a) The software producer ensures these processes operate on an ongoing basis and, at a minimum, prior to product, version, or update releases; and (b) The software producer has a policy or process to address discovered security vulnerabilities prior to product release; and (c) The software producer operates a vulnerability disclosure program and accepts, reviews, and addresses disclosed software vulnerabilities in a timely fashion.

Software Bill of Materials (SBOM)

Implementation Details

- During build, SPDX files for each package and source file are created from source files, augmented with license data from license files (if present), dependencies from external libraries and the hash of the source file and placed in the project build directory.

- Concatenated together into the project directory.

- Tools can verify that the contents of the SPDX files match the source files.

Sample SPDX

```
SPDXVersion: SPDX-1.2
DataLicense: CC0-1.0
DocumentNamespace: https://www.insyde.com/spdxdocs/FatPkg-SPDX-1.2-00e96079-9043-522d-bc22-bbccf48d6de3Document
Name: FatPkg
SPDXSPDXID: SPDXRef-DOCUMENT
DocumentComment: <text>FatPkg Document</text>

Creator: Organization: Insyde Software Corp. (https://www.insyde.com/)
Created: 2022-04-21T03:19:03Z

Relationship: SPDXRef-DOCUMENT CONTAINS SPDXRef-Package-FatPkg
RelationshipComment: <text>Document contains package FatPkg</text>

PackageName: FatPkg
SPDXID: SPDXRef-Package-FatPkg
PackageVersion: 1.0
PackageDownloadLocation: http://svn.insyde.com/
PackageSupplier: Organization: Insyde Software Corp. (https://www.insyde.com/)
PackageVerificationCode: df94ca698b3e3ffa862b3cd5ac3d9568dfda10cd
PackageLicenseDeclared: BSD-2-Clause
PackageLicenseConcluded: BSD-2-Clause
PackageLicenseInfoFromFiles: BSD-2-Clause
PackageCopyrightText: <text>Copyright (c) 2012 - 2022, Insyde Software Corp. All Rights Reserved.</text>

FileName: \EDK2\FatPkg\EnhancedFatDxe\Data.c
SPDXID: SPDXRef-Data-srcFileType: SOURCE
FileChecksum: SHA1: d0cc1c226b572507bb3aafb5b4ef363fa0579404
LicenseConcluded: BSD-2-Clause
LicenseInfoInFile: BSD-2-Clause
FileCopyrightText: <text>Copyright (c) 2005 - 2013, Intel Corporation. All rights reserved.</text>
```

Implementation Details

- If SWID XML files are present, they are updated by the build tools with the hash of the module.
- If SWID XML files are not present, they are generated from for each package by build tools, augmented with data from SPDX files (if present) and the hash of the module.
- Converted to coSWID, header added and copied to output directory and, optionally, inserted into special SBOM region in the final firmware image.
- Usable by BMC
  - Validate image or image vs. coSWID SBOM file and monitor

Sample Module SWID XML SBOM

```
<?xml version="1.0" ?><SoftwareIdentity lang="En" name="VariableRuntimeDxe"
  tagId="CBD2E4D5-7068-4FF5-B462-9822B4AD8D60" version="1.0"
  xmlns="http://standards.iso.org/iso/19770/-2/2015/schema.xsd">
  <Entity name="Insyde Software Corp." regid="insyde.com"
    role="tagCreator softwareCreator"/>
  <Meta colloquialVersion="8BA32620EDAB7B861ACEE5861E494244E72D6683"/>
</SoftwareIdentity>
```

Possible Future SBOM Efforts

- Optimize for size: Use compression, don’t include licensing information.
- Add library dependencies (for all libraries or only external libraries) for SWID
- Create a validation tool to verify module hash matches those in SBOM.

Call to Action

- Update your Secure Development Lifecycle (SDL) based on NIST guidelines in order that you can provide attestation to your customers.

- Prepare a strategy that gives end users and IT department more information about the firmware contents and gives them ways to respond to security incidents.

- Get your SBOMs ready for (at least) the entire BIOS and the end-user tools. UEFI SBOM Team working on minimum standards.

Links

- https://www.nist.gov/system/files/documents/2022/02/04/software-supply-chain-security-guidance-under-EO-14028-section-4e.pdf
- https://www.whitehouse.gov/wp-content/uploads/2022/09/M-22-18.pdf
- https://www.whitehouse.gov/wp-content/uploads/2023/06/M-23-16-Update-to-M-22-18-Enhancing-Software-Security.pdf
- https://www.ntia.doc.gov/files/ntia/publications/sbom_minimum_elements_report.pdf
- https://www.nist.gov/itl/executive-order-improving-nations-cybersecurity/critical-software-definition-explanatory
- https://www.cisa.gov/sites/default/files/2023-04/secure-software-self-attestation_common-form_508.pdf

Thanks for attending the UEFI Fall 2023 Developers Conference & Plugfest

For more information on UEFI Forum and UEFI Specifications, visit http://www.uefi.org

presented by

Backup

NIST SP 800-218 Checklist (PO.1)

Define Security Requirements for Software Development

Ensure that security requirements for software development are known at all times so that they can be taken into account throughout the SDLC and duplication of effort can be minimized because the requirements information can be collected once and shared.

This includes requirements from internal sources (e.g., the organization's policies, business objectives, and risk management strategy) and external sources (e.g., applicable laws and regulations).

PO.1.1: Identify and document all security requirements for the organization's software development infrastructures and processes, and maintain the requirements over time.

PO.1.2: Identify and document all security requirements for organization-developed software to meet, and maintain the requirements over time.

PO.1.3: Communicate requirements to all third parties who will provide commercial software components to the organization for reuse by the organization's own software.

NIST SP 800-218 Checklist (PO.2)

Implement Roles and Responsibilities

Ensure that everyone inside and outside of the organization involved in the SDLC is prepared to perform their SDLC-related roles and responsibilities throughout the SDLC.

PO.2.1: Create new roles and alter responsibilities for existing roles as needed to encompass all parts of the SDLC.

PO.2.2: Provide role-based training for all personnel with responsibilities that contribute to secure development.

PO.2.3: Obtain upper management or authorizing official commitment to secure development, and convey that commitment to all with development-related roles and responsibilities.

NIST SP 800-218 Checklist (PO.3)

Implement Supporting Toolchains

Use automation to reduce human effort and improve the accuracy, reproducibility, usability, and comprehensiveness of security practices throughout the SDLC, as well as provide a way to document and demonstrate the use of these practices.

Toolchains and tools may be used at different levels of the organization, such as organization-wide or project-specific, and may address a particular part of the SDLC, like a build pipeline.

PO.3.1: Specify which tools or tool types must or should be included in each toolchain to mitigate identified risks, as well as how the toolchain components are to be integrated with each other.

PO.3.2: Follow recommended security practices to deploy, operate, and maintain tools and toolchains.

PO.3.3: Configure tools to generate artifacts of their support of secure software development practices as defined by the organization.

NIST SP 800-218 Checklist (PO.4)

Define and Use Criteria for Software Security Checks

Help ensure that the software resulting from the SDLC meets the organization's expectations by defining and using criteria for checking the software's security during development.

PO.4.1: Define criteria for software security checks and track throughout the SDLC.

PO.4.2: Implement processes, mechanisms, etc. to gather and safeguard the necessary information in support of the criteria.

NIST SP 800-218 Checklist (PO.5)

Implement and Maintain Secure Environments for Software Development

Ensure that all components of the environments for software development are strongly protected from internal and external threats to prevent compromises of the environments or the software being developed or maintained within them.

Examples of environments for software development include development, build, test, and distribution environments.

PO.5.1: Separate and protect each environment involved in software development.

PO.5.2: Secure and harden development endpoints (i.e., endpoints for software designers, developers, testers, builders, etc.) to perform development-related tasks using a risk-based approach.

NIST SP 800-218 Checklist (PS.1)

Protect All Forms of Code from Unauthorized Access and Tampering

Help prevent unauthorized changes to code, both inadvertent and intentional, which could circumvent or negate the intended security characteristics of the software.

For code that is not intended to be publicly accessible, this helps prevent theft of the software and may make it more difficult or time-consuming for attackers to find vulnerabilities in the software.

PS.1.1: Store all forms of code – including source code, executable code, and configuration-as-code – based on the principle of least privilege so that only authorized personnel, tools, services, etc. have access.

NIST SP 800-218 Checklist (PS.2)

Provide a Mechanism for Verifying Software Release Integrity

Help software acquirers ensure that the software they acquire is legitimate and has not been tampered with.

PS.2.1: Make software integrity verification information available to software acquirers.

NIST SP 800-218 Checklist (PS.3)

Archive and Protect Each Software Release

Preserve software releases in order to help identify, analyze, and eliminate vulnerabilities discovered in the software after release.

PS.3.1: Securely archive the necessary files and supporting data (e.g., integrity verification information, provenance data) to be retained for each software release.

PS.3.2: *Collect, safeguard, maintain, and share provenance data for all components of each software release (e.g., in a software bill of materials [SBOM]).*

NIST SP 800-218 Checklist (PW.1)

Design Software to Meet Security Requirements and Mitigate Security Risks

Identify and evaluate the security requirements for the software; determine what security risks the software is likely to face during operation and how the software's design and architecture should mitigate those risks; and justify any cases where risk-based analysis indicates that security requirements should be relaxed or waived.

Addressing security requirements and risks during software design (secure by design) is key for improving software security and also helps improve development efficiency.

PW.1.1: Use forms of risk modeling – such as threat modeling, attack modeling, or attack surface mapping – to help assess the security risk for the software.

PW.1.2: Track and maintain the software's security requirements, risks, and design decisions.

PW.1.3: Where appropriate, build in support for using standardized security features and services (e.g., enabling software to integrate with existing log management, identity management, access control, and vulnerability management systems) instead of creating proprietary implementations of security features and services.

NIST SP 800-218 Checklist (PW.2)

Review the Software Design to Verify Compliance with Security Requirements and Risk Information

Help ensure that the software will meet the security requirements and satisfactorily address the identified risk information.

PW.2.1: Have 1) a qualified person (or people) who were not involved with the design and/or 2) automated processes instantiated in the toolchain review the software design to confirm and enforce that it meets all of the security requirements and satisfactorily addresses the identified risk information.

NIST SP 800-218 Checklist (PW.4)

Reuse Existing, Well-Secured Software When Feasible Instead of Duplicating Functionality

Lower the costs of software development, expedite software development, and decrease the likelihood of introducing additional security vulnerabilities into the software by reusing software modules and services that have already had their security posture checked.

This is particularly important for software that implements security functionality, such as cryptographic modules and protocols.

PW.4.1: Acquire and maintain well-secured software components (e.g., software libraries, modules, middleware, frameworks) from commercial, open-source, and other third-party developers for use by the organization's software.

PW.4.2: Create and maintain well-secured software components in-house following SDLC processes to meet common internal software development needs that cannot be better met by third-party software components.

PW.4.4: Verify that acquired commercial, open-source, and all other third-party software components comply with the requirements, as defined by the organization, throughout their life cycles.

NIST SP 800-218 Checklist (PW.5)

Create Source Code by Adhering to Secure Coding Practices

Decrease the number of security vulnerabilities in the software and reduce costs by minimizing vulnerabilities introduced during source code creation that meet or exceed organization-defined vulnerability severity criteria.

PW.5.1: Follow all secure coding practices that are appropriate to the development languages and environment to meet the organization's requirements.

NIST SP 800-218 Checklist (PW.6)

Configure the Compilation, Interpreter, and Build Processes to Improve Executable Security

Decrease the number of security vulnerabilities in the software and reduce costs by eliminating vulnerabilities before testing occurs.

PW.6.1: Use compiler, interpreter, and build tools that offer features to improve executable security.

PW.6.2: Determine which compiler, interpreter, and build tool features should be used and how each should be configured, then implement and use the approved configurations.

NIST SP 800-218 Checklist (PW.7)

Review and/or Analyze Human-Readable Code to Identify Vulnerabilities and Verify Compliance with Security Requirements

Help identify vulnerabilities so that they can be corrected before the software is released to prevent exploitation.

Using automated methods lowers the effort and resources needed to detect vulnerabilities. Human-readable code includes source code, scripts, and any other form of code that an organization deems human-readable.

PW.7.1: Determine whether code review (a person looks directly at the code to find issues) and/or code analysis (tools are used to find issues in code, either in a fully automated way or in conjunction with a person) should be used, as defined by the organization.

PW.7.2: Perform the code review and/or code analysis based on the organization's secure coding standards, and record and triage all discovered issues and recommended remediations in the development team's workflow or issue tracking system.

NIST SP 800-218 Checklist (PW.8)

Test Executable Code to Identify Vulnerabilities and Verify Compliance with Security Requirements

Help identify vulnerabilities so that they can be corrected before the software is released in order to prevent exploitation.

Using automated methods lowers the effort and resources needed to detect vulnerabilities and improves traceability and repeatability.

Executable code includes binaries, directly executed bytecode and source code, and any other form of code that an organization deems executable.

PW.8.1: Determine whether executable code testing should be performed to find vulnerabilities not identified by previous reviews, analysis, or testing and, if so, which types of testing should be used.

PW.8.2: Scope the testing, design the tests, perform the testing, and document the results, including recording and triaging all discovered issues and recommended remediations in the development team's workflow or issue tracking system.

NIST SP 800-218 Checklist (PW.9)

Configure Software to Have Secure Settings by Default

Help improve the security of the software at the time of installation to reduce the likelihood of the software being deployed with weak security settings, putting it at greater risk of compromise.

PW.9.1: Define a secure baseline by determining how to configure each setting that has an effect on security or a security-related setting so that the default settings are secure and do not weaken the security functions provided by the platform, network infrastructure, or services.

PW.9.2: Implement the default settings (or groups of default settings, if applicable), and document each setting for software administrators.

NIST SP 800-218 (RV.1)

Identify and Confirm Vulnerabilities on an Ongoing Basis

Help ensure that vulnerabilities are identified more quickly so that they can be remediated more quickly in accordance with risk, reducing the window of opportunity for attackers.

RV.1.1: Gather information from software acquirers, users, and public sources on potential vulnerabilities in the software and third-party components that the software uses, and investigate all credible reports.

RV.1.2: Review, analyze, and/or test the software's code to identify or confirm the presence of previously undetected vulnerabilities.

RV.1.3: Have a policy that addresses vulnerability disclosure and remediation, and implement the roles, responsibilities, and processes needed to support that policy.

NIST SP 800-218 (RV.2)

Assess, Prioritize, and Remediate Vulnerabilities

Help ensure that vulnerabilities are remediated in accordance with risk to reduce the window of opportunity for attackers.

RV.2.1: Analyze each vulnerability to gather sufficient information about risk to plan its remediation or other risk response.

RV.2.2: Plan and implement risk responses for vulnerabilities.

NIST SP 800-218 (RV.3)

Analyze Vulnerabilities to Identify Their Root Causes

Help reduce the frequency of vulnerabilities in the future.

RV.3.1: Analyze identified vulnerabilities to determine their root causes.

RV.3.2: Analyze the root causes over time to identify patterns, such as a particular secure coding practice not being followed consistently.

RV.3.3: Review the software for similar vulnerabilities to eradicate a class of vulnerabilities, and proactively fix them rather than waiting for external reports.

RV.3.4: Review the SDLC process, and update it if appropriate to prevent (or reduce the likelihood of) the root cause recurring in updates to the software or in new software that is created.