IDS WG Meeting Minutes
March 3, 2022

This IDS WG Meeting was started at approximately 3:00 pm ET on March 3, 2022.

Attendees

Graydon Dodson	Lexmark
Erin Huber	Xerox
Smith Kennedy	HP
Jeremy Leber	Lexmark
Alan Sukert	
Bill Wagner	TIC
Steve Young	Canon

Agenda Items

1. The topics to be covered during this meeting were:
   - Review of the HCD iTC Meetings since our last HCD iTC Meeting on 2/3/222
   - Presentation by Al Sukert on NIST SP 800-218, Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities
   - Round Table
2. Meeting began by stating the PWG Anti-Trust Policy which can be found at https://www.pwg.org/chair/membership_docs/pwg-antitrust- policy.pdf and the PWG Intellectual Property Policy which can be found at https://www.pwg.org/chair/membership_docs/pwg-ip-policy.pdf.
3. Al then provided a summary of what was covered at the two HCD iTC Meetings (2/21/22 and 2/28/22) since the last IDS Workgroup meeting on 2/17/22.
   - The most important item was that the 2nd Public Draft of the HCD Supporting Document (SD) – Version 0.98 dated 2/24/2022 – was released for public review on Tuesday March 1st. Comments against this draft are due to the HCD iTC by April 15th.

     Al went through the list of the major changes that were included in this 2nd Public Draft of the HCD SD. The major changes were:
     - Added a Test Assurance Activity for SFR **FPT_TST_EXT Extended: TSF testing** where one was not present in previous drafts.
     - Moved the Assurance Activities for the following:
     - All of the Audit Log related SFRs to under "Security Audit (FAU)" rather than because they are all mandatory SFRs
     - SFR **FCS_CKM.1/AKG Cryptographic Key Generation (for asymmetric keys)** to **Chapter 3. Evaluation Activities for Conditionally Mandatory Requirements** as required by NIAP Technical Decision TD 0074
     - SFR **FCS_CKM.2 Cryptographic Key Establishment** to **Chapter 3. Evaluation Activities for Conditionally Mandatory Requirements** because it refers to the conditional requirement SFR **FCS_CKM.1.1/AKG Cryptographic Key Generation (for asymmetric keys)**
     - Added ISO/IEC 11770-6:2016 to the list of references an evaluator shall verify the approved derivation mode and key expansion algorithm for in the TSS Assurance Activity for SFR **FCS_KDF_EXT.1 Extended: Cryptographic Key Derivation.**
     - Corrected an incorrect CEM paragraph reference in **Section 6.2.1. Basic Functional Specification (ADV_FSP.1)** *Table 2. Mapping of ADV_FSP.1 CEM Work Units to Evaluation Activities*.
     - Corrected several unreachable URLs in Appendix C: Public Vulnerability Sources.

- Removed redundant Operator User Guidance Evaluation Activities related to the evaluator ensuring that the Operational guidance contains instructions for configuring any cryptographic engine associated with the evaluated configuration of the TOE and providing a warning to the administrator that use of other cryptographic engines was not evaluated nor tested during the CC evaluation of the TOE.
- Corrected two incorrect paragraph references in **Section 6.6.1. Vulnerability Survey (AVA_VAN.1)** *Table 3. Mapping of AVA_VAN.1 CEM Work Units to Evaluation Activities*.
- Added the missing content of the Evaluation Activity (Documentation) and Evaluation Activity sections under **Section 6.6.1. Vulnerability Survey (AVA_VAN.1)**.
- Implemented the significant updates to the Assurance Activities (mostly in the Test Assurance Activities) requested by ITSCC (the Korean Common Criteria Scheme) for the following SFRs:
  - **FCS_CKM.1/SKG Cryptographic key generation (Symmetric Keys)**
  - **FCS_COP.1/DataEncryption Cryptographic Operation (Data Encryption/Decryption)**
  - **FCS_COP.1/SigGen Cryptographic Operation (Signature Generation and Verification)**
  - **FCS_RBG_EXT.1 Extended: Cryptographic Operation (Random Bit Generation)**
  - **FCS_COP.1/StorageEncryption Cryptographic operation (Data Encryption/Decryption)**
  - **FCS_COP.1/KeyWrap Cryptographic operation (Key Wrapping)**
  - **FCS_CKM.1/AKG Cryptographic Key Generation (for asymmetric keys)**
  - **FCS_KDF_EXT.1 Extended: Cryptographic Key Derivation**

- Otherwise, the main accomplishment over the last two HCD iTC meetings was that the iTC finished addressing all of the comments received against the 2nd Public Draft of the HCD cPP. That means that the HCD cPP author Brian Volkoff can start working of the Final Draft of the HD cPP. That means that baring any major changes due to the conditions spelled out at the Feb 9th IDS Face-to-Face Session, the HCD iTC is on track to have the Final Draft of the HCD cPP ready for public review at the beginning of April as planned.

4. Al then went through a presentation he put together on NIST SP 800-218, Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities; [SP 800-218, Secure Software Development Framework (SSDF) Version 1.1 | CSRC (nist.gov)]() provides a link to NIST SP 800-218. The presentation slides are located at https://ftp.pwg.org/pub/pwg/ids/Presentation/SSDF.pdf. This NIST SP is a direct result of the Cybersecurity Executive Order 2021-14028 issued May 12, 2021.

   One of the subjects of this Executive Order was "*Enhancing Software Supply Chain Security*" which included the following request:

   Within 90 days of publication of the preliminary guidelines pursuant to subsection (c) of this section, the Secretary of Commerce acting through the Director of NIST, in consultation with the heads of such agencies as the Director of NIST deems appropriate, shall issue guidance identifying practices that enhance the security of the software supply chain. Such guidance may incorporate the guidelines published pursuant to subsections (c) and (i) of this section. Such guidance shall include standards, procedures, or criteria regarding:
   (i) secure software development environments, including such actions as:
   (A) using administratively separate build environments;
   (B) auditing trust relationships;
   (C) establishing multi-factor, risk-based authentication and conditional access across the enterprise;

(D) documenting and minimizing dependencies on enterprise products that are part of the environments used to develop, build, and edit software;
(E) employing encryption for data; and
(F) monitoring operations and alerts and responding to attempted and actual cyber incidents;

It was this request that led to the development of NIST SP 800-218.

The Secure Software Development Framework (SSDF) describes a set of high-level practices based on established standards, guidance, and secure software development practice documents; it focuses on the outcomes of the practices rather than on the tools, techniques, and mechanisms to do so. Its goal is to identify secure software development practices but not prescribe how to implement them.

The SSDF is divided into four groups:

- **Prepare the Organization (PO):** Organizations should ensure that their people, processes, and technology are prepared to perform secure software development at the organization level.
- **Protect the Software (PS):** Organizations should protect all components of their software from tampering and unauthorized access.
- **Produce Well-Secured Software (PW):** Organizations should produce well-secured software with minimal security vulnerabilities in its releases.
- **Respond to Vulnerabilities (RV):** Organizations should identify residual vulnerabilities in their software releases and respond appropriately to address those vulnerabilities and prevent similar ones from occurring in the future.

Each of these groups is divided into practices, and each practice contains the following elements:

- **Practice:** The name of the practice and a unique identifier, followed by a brief explanation of what the practice is and why it is beneficial
- **Tasks:** One or more actions that may be needed to perform a practice
- **Notional Implementation Examples:** One or more notional examples of types of tools, processes, or other methods that could be used to help implement a task. No examples or combination of examples are required, and the stated examples are not the only feasible options. Some examples may not be applicable to certain organizations and situations.
- **References:** Pointers to one or more established secure development practice documents and their mappings to a particular task. Not all references will apply to all instances of software development

The practices in each group are as follows:

<u>Prepare the Organization (PO)</u>

- **Define Security Requirements for Software Development (PO.1)**: 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)
- **Implement Roles and Responsibilities (PO.2)**: 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
- **Implement Supporting Toolchains (PO.3)**: 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

- **Define and Use Criteria for Software Security Checks (PO.4)**: 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
- **Implement and Maintain Secure Environments for Software Development (PO.5)**: 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

<u>Protect the Software (PS)</u>

- **Protect All Forms of Code from Unauthorized Access and Tampering (PS.1)**: 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
- **Provide a Mechanism for Verifying Software Release Integrity (PS.2)**: Help software acquirers ensure that the software they acquire is legitimate and has not been tampered with
- **Archive and Protect Each Software Release (PS.3)**: Preserve software releases in order to help identify, analyze, and eliminate vulnerabilities discovered in the software after release

<u>Produce Well-Secured Software (PW)</u>

- **Reuse Existing, Well-Secured Software When Feasible Instead of Duplicating Functionality (PW.4)**: 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
- **Create Source Code by Adhering to Secure Coding Practices (PW.5)**: 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
- **Configure the Compilation, Interpreter, and Build Processes to Improve Executable Security (PW.6)**: Decrease the number of security vulnerabilities in the software and reduce costs by eliminating vulnerabilities before testing occurs
- **Review and/or Analyze Human-Readable Code to Identify Vulnerabilities and Verify Compliance with Security Requirements (PW.7)**: 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
- **Test Executable Code to Identify Vulnerabilities and Verify Compliance with Security Requirements (PW.8)**: 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
- **Configure Software to Have Secure Settings by Default (PW.9)**: 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

<u>Respond to Vulnerabilities (RV)</u>

- **Identify and Confirm Vulnerabilities on an Ongoing Basis (RV.1)**: 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

- **Assess, Prioritize, and Remediate Vulnerabilities (RV.2)**: Help ensure that vulnerabilities are remediated in accordance with risk to reduce the window of opportunity for attackers
- **Analyze Vulnerabilities to Identify Their Root Causes (RV.3)**: Help reduce the frequency of vulnerabilities in the future

The actual document includes a table that traces each of these practices back to a subsection in the Executive Order, and it turns out that all of the practices trace back to the subsection referenced above.

The presentation does list the Tasks for each of the practices. Al highlighted a few of them:

- **PO.1.1**: Identify and document all security requirements for the organization's software development infrastructures and processes, and maintain the requirements over time and **PO.1.2**: Identify and document all security requirements for organization-developed software to meet, and maintain the requirements over time – this shows the important of not only identifying the software requirements but identifying the requirements for the software development environment.
- **PO.3.2**: Follow recommended security practices to deploy, operate, and maintain tools and toolchains – Tools and toolchains are an integral part of generating software builds so it is important that they be properly deployed, operated and maintained
- **PO.4.1**: Define criteria for software security checks and track throughout the SDLC – it will be interesting to see how this one gets implemented because it is not clear how or if this has been done in the past
- **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 – It will be interesting to see how the "endpoints" are defined for this task
- **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 – Al was glad they included the principle of "least privilege" in accessing source code
- **PS.2.1**: Make software integrity verification information available to software acquirers – This same principle of "software integrity verification" was included in the HCD cPP with respect to the integrity verification of the software chains and Roots of Trust
- **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]) – SBOMs are another area of focus in the Executive Order
- **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 – nice to see inclusion of risk modeling as a task
- **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 – very important that a non-designed not review the designs so you don't have "the fox guarding the hen house"
- **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 – It is important that developer maintain a catalog of both in-house and 3rd Party components to facilitate reuse
- **PW.5.1**: Follow all secure coding practices that are appropriate to the development languages and environment to meet the organization's requirements – this is one task that too many

developers don’t follow and should. Too many errors and vulnerabilities are introduced because developers either do not have coding standards or do not follow the coding standards they have

- **PW.6.1**: Use compiler, interpreter, and build tools that offer features to improve executable security and **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 – we had an interesting discussion on these two tasks about how compliers are used and misused with users not making the proper settings or leaving all the settings on while doing continuous integration.
- **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 – Al commented that when he first started working at GE his job was to review Fortran code for violations of coding standards; code reviews are a very time-consuming and difficult job. It is important that automatic code checkers be used for this task.
- **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 – it is not aa question of whether here; there have been significant advances in automatic testing and analysis to aid in finding vulnerabilities
- **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 – this is something many customers are now asking for; they want a baseline of configuration settings for each of the machines that get from a vendor that will ensure each machine is secretly set up.
- **RV.1.2**: Review, analyze, and/or test the software’s code to identify or confirm the presence of previously undetected vulnerabilities – this is obviously something that has been and will continue to be stressed given the fact that attackers are getting smarter, better funded and their tools are getting better and more sophisticated.
- **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 – this is a very important but often neglected step; any SDLC must be evergreen and continuously improved through this type of review.

5. There was no Round Table discussion at today’s meeting.
6. **Actions:** None

Next Steps

- The next IDS WG Meeting will be March 17, 2022 at 3:00P ET / 12:00N PT. Main topics will be review of the 2/21 and 2/28 HCD iTC Meetings, if Ira can attend a deeper dive into the 2/8/22 update of the HCD Software Guidelines (otherwise a special topic to be determined) and round table.