Secure Design and Development Cybersecurity Capability Maturity Model (SD2-C2M2): Next-Generation Cyber Resilience by Design

Sri Nikhil Gupta Gourisetti, Scott Mix, Michael Mylrea, Christopher Bonebrake, Md Touhiduzzaman
Electricity Infrastructure – Cybersecurity *in* Energy and Environment Directorate
Pacific Northwest National Laboratory
Richland, WA, USA
{SriNikhil.Gourisetti|Scott.Mix|Michael.Milrea|Christopher.Bonebrake|Md.Touhiduzzaman}@pnnl.gov

ABSTRACT

The Secure Design and Development Cybersecurity Capability Maturity Model (SD2-C2M2) provides a browser-based tool that allows hardware and software developers to assess the maturity level of their design and development processes, allows management to determine desired maturity levels in seven domains, and allows developers to monitor process maturity improvements against management goals. The tool can be used by commercial developers as well as internal development organizations. This paper provides an overview of the tool and domains and presents a hypothetical case study of the tool to reduce software buffer overflow attack vulnerabilities by improving the software development process.

KEYWORDS

C2M2, SD2-C2M2, cybersecurity, maturity model, operational technology, industrial control systems, energy delivery system, software development, hardware development, system design

ACM Reference format:

Sri Nikhil Gupta Gourisetti, Scott Mix, Michael Mylrea, Christopher Bonebrake, Md Touhiduzzaman. 2019. Secure Design and Development Cybersecurity Capability Maturity Model (SD2-C2M2): Next-Generation Cyber Resilience by Design. In *Proceedings of ACM Northwest Cybersecurity Symposium. ACM, Richland, WA, USA.*

1 Introduction

Securing the supply chain of critical control components in modern electricity control systems is herculean task, because vendors rush critical product to market without considering cybersecurity as part of their design and deployment criteria. Currently, no widely available, quantifiable, and repeatable method can evaluate the cybersecurity of energy delivery system (EDS) components throughout their entire lifecycle. Such a methodology is necessary to enable vendor improvement, manufacturer improvement, and end-user requirements in product purchasing. In response, researchers at Pacific Northwest National Laboratory (PNNL) developed the Secure Design and Development Cybersecurity Capability Maturity Model (SD2-C2M2) to enable developers, producers, and integrators of industrial control systems, also known as Operational Technology (OT). SD2-C2M2 is developed following the core functionalities from the electric sector C2M2 (ES-C2M2) [1] and buildings cybersecurity framework (BCF) [2] methodologies. This novel and timely Cybersecurity Capability Maturity Model enables OT stakeholders to include cybersecurity in the design, development, manufacture, testing, and maintenance of critical operational technology.

This study was conducted at the Pacific Northwest National Laboratory, which is operated for the U. S. Department of Energy by the Battelle Memorial Institute under Contract DE-AC05-75RL01830.

By developing an easy-to-use tool for secure design and development of OT, cybersecurity best practices can be adopted, and processes can be assessed against desired cybersecurity maturity levels to produce more secure products. Initial internal research at PNNL has compiled a comprehensive set of best practices that are applicable to the lifecycles of software, firmware, hardware, and to human factors (e.g., training and environments). This data was used to create a graphical interface toolset that allows a user to select the subset of the best practices. The user can use the GUI application to evaluate the organization's maturity in the areas of interest. For example, an EDS vendor could choose to explore areas involving design and creation of devices and software and an owner/operator might choose the areas involving use, maintenance, and end-of-life tasks. The vendor or owner/operator could then select criteria (i.e., not implemented, informally implemented, documented, or formally implemented) in a range of detailed areas. For an initial investigation, results might be compared to the expectations of upper management; later results could show growth from the initial baseline. Those best practices were used to develop a toolset that has a graphical user interface (GUI). The GUI allows the user to perform the assessments and has features such as a results report generator, advanced data analytics, etc. Those features will allow the user to perform in-depth analysis of the results acquired from the assessment. For this paper, our objective is to provide an overview of the SD2-C2M2 methodology, software application, and demonstrate its efficacy through a use case.

2 SD2-C2M2 Architecture

SD2-C2M2 is a tool that developers and integrators of critical OT can use to assess and improve their design and development practices and procedures based on a set of best practices. By facilitating implementation of cybersecurity best practices, the tool can then compare the maturity levels against a set of management-derived requirements to determine the areas of interest that require improvements to be made.

The tool facilitates the secure development process through seven major domains covering *Background & Foundation*, *Design*, *Build*, *Test*, *Integrate*, *Deploy*, and *Lifecycle & End-of-Life*. These domains are generally implemented chronologically, but a domain can be revisited later in the development process, if necessary. For example, if serious errors are found during the Test process, it may be necessary to revisit the Build process; or, if new features are requested during the Deploy process, a major product revision may require going all the way back to the Background & Foundation process to develop a new set of requirements that must be designed, built, and tested. SD2-C2M2 also has a built-in feature to save an assessment and compare against future assessments. This feature helps the operators to determine where improvements have been made, and whether the improvements have caused an increase in the maturity level indicator for those improvements.

2.1 Domain Definitions

The seven domains used by the SD2-C2M2 tool follow the typical hardware or software design lifecycle. The domains (described below) allow the tool to logically group the best practices and allow responses to be given by different subject matter expert groups.

- **Background & Foundation:** considers practices and procedures that serve as foundation processes. These areas include developer training; understanding the environments in which the products will or could be deployed; processes for gathering and documenting requirements in accordance with which the products will be designed, built, and tested; understanding and using the tools that the developers will be using, and understanding the security considerations of working with third-party suppliers and vendors.
- **Design:** considers the processes used to specify how products will be built, based on requirements and other factors. In addition to hardware and software design considerations, these factors include human factors (e.g., usability), failure mode analysis, selection of programming languages, system (as opposed to component) design, security considerations, and designing for testability and maintenance of the final product.
- **Build:** considers practices that are used to turn the design into a product that can be delivered to customers. These practices include hardware construction and software development, as well as managing changes, and considering the impacts of third-party suppliers.
- **Test:** considers the processes used to test the developed and built products against the specified requirements. The Test domain considers testing of the hardware and software components.
- **Integrate:** considers the processes used to integrate hardware and software components into a system. These processes and procedures include integration and assembly procedures, configuration actions performed at the factory, system-level testing, customer-witnessed factory testing, and preparing the system for shipment to the customer.
- **Deploy:** considers the processes and procedures used by the customer to configure and use the delivered system. These processes and procedures include additional testing of the system in its ultimate location, training of the customer in the use of the system, and using the documentation provided to the customer with the system.
- **Lifecycle & End-of-Life:** considers the processes used by the customer to operate and maintain the delivered system. Maintenance procedures include customer-performed as well as factory-performed maintenance. The domain also includes end-of-life actions to dispose of the system and information contained in it when the system is no longer in service.

2.2 Maturity Definitions

The SD2-C2M2 tool is used to assess practices and procedures used for secure design and development of systems. The tool focuses on assessing whether procedures and processes exist and determining their level of formality. All the questions in the tool elicit four possible states:

- **Not Implemented:** The specific practice or procedure has not been implemented.
- **Informally Implemented:** The specific practice or procedure is implemented, but not in a formal or consistent manner. Developers may be aware of the practice or procedure and may implement it at varying levels. No oversight exists to determine whether the practice is being performed. Processes may be implemented in an *ad hoc* manner based on “tribal knowledge base” or suggested by senior designers/mentors.
- **Documented:** The specific practice/procedure is documented with expectations by management that it be followed, but it is not necessarily being followed. If it is followed, it may not be followed completely or consistently across projects or between developers. There are no procedures in place to determine whether the procedure is being followed.
- **Formally Implemented:** The specific practices/procedures are being consistently followed as documented in all cases. Oversight is in place (e.g., automated reviews, peer reviews, sign-off, supervisory oversight) to ensure that the procedure is being followed, and that deviations from expected performance are corrected. Procedures may be reviewed periodically to determine whether improvements can or need to be made to them.

For example, in the *Background & Foundation* domain, for the *Developer Training & Certification* subdomain, the following would be applied:

- **Not Implemented**: No training is expected or required.
- **Informally Implemented**: Tribal/institutional knowledge, example code comments.
- **Documented**: Coding procedures with no follow-on that determines they are implemented, no training other than "here's the document - follow it."
- **Formally Implemented**: Classroom training (e.g., 4 to 8 hours), with refresher training, code review sign-off looking for specific coding errors, automated code inspection tool.

The SD2-C2M2 tool could also be used to determine the efficacy of procedures used on a specific project. For example, if formal procedures exist for an area, but they are not followed during the development of a particular product (for example, because they are outdated, create perceived inefficiencies, or are deemed unimportant by line management or supervisors), the lack of conformance with the procedures indicates a gap in the implementation of the procedures, not necessarily a shortcoming of the procedures themselves. Conformance with the established procedures needs to be investigated to determine the root cause of the gap and a remedy to address the gap (for example, to modify the procedures to make them relevant or efficient, or to address supervisory attention to the documented procedures).

The Maturity Indicator Level (MIL) descriptions are documented in more detail in the Electricity Subsector Cybersecurity Capability Maturity Model (ES-C2M2) document [3]. Four aspects of the MILs are important for understanding and applying SD2-C2M2:

1. The MILs apply independently to each domain. As a result, an organization using the model may be operating at different MIL ratings for different domains. For example, an

organization could be operating at MIL1 in one domain, MIL2 in another domain, and MIL3 in a third domain.

2. The MILs are cumulative within each domain; to earn a MIL in a given domain, an organization must perform all of the practices in that level and its predecessor level(s). For example, an organization must perform all of the domain practices in MIL1 and MIL2 to achieve MIL2 in the domain. Similarly, the organization would have to perform all practices in MIL1, MIL2, and MIL3 to achieve MIL3.
3. Establishing a target MIL for each domain is an effective strategy for using the model to guide cybersecurity program improvement. Organizations should become familiar with the practices in the model prior to determining target MILs. Gap analysis activities and improvement efforts should then focus on achieving those target levels.
4. The performance of best practices and MIL achievement need to align with business objectives and the organization's cybersecurity strategy. Striving to achieve the highest MIL in all domains may not be optimal. Companies should evaluate the costs of achieving a specific MIL against potential benefits. However, the model was developed so that all companies, regardless of size, should be able to achieve MIL1 across all domains.

Additional information about the MILs is available in the ES-C2M2 document [3] (See Section 5.2 on pages 15-17 in [3]).

3 SD2-C2M2 Software Application and Design

The SD2-C2M2 software was built from the ground up with a focus on ease of use and functionality. Some of the built-in applications include Management Priorities, SD2-C2M2 Core, and Comparative Evaluation. Some of the main design features of the webtool include a pie summary, timeline graphs, and tabular plots; a save/load feature to retain assessment on-premises; and the ability to generate a .pdf report at the end of an assessment. The objective of this section is to discuss the above features in detail.

3.1 Design and Features

Management Priorities: The management priorities sub-application will allow management to define their goals across all the domains. An illustration of the sub-application is shown in Figure 1. According to the illustration, management would like to reach MIL1 for the *Background & Foundation* domain; MIL2 for *Design*, *Test*, and *Deploy* domains; MIL3 for *Build* and *Integrate* domains. During the core assessment process, the management priorities will be reflected in the process. Therefore, this sub-application acts like a bridge between management expectations and the practices of engineering and development teams.

SD2-C2M2 Core: This contains more than 700 best practices related to secure design and development processes that are tailored to technical and non-technical/management aspects of an organization. The best practices are divided and grouped over 7 domains that are further divided into 35 subdomains, as shown in Figure 2. The CORE framework questionnaire is inspired by an internal research project at PNNL under which more than 16 researchers and technical experts with over 200 years of combined experience in cybersecurity and hardware and software design, development, and deployment contributed to its development.

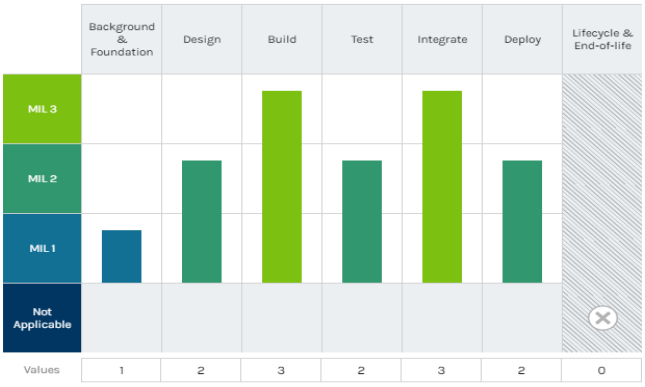

Figure 1: Illustration of the Management Priorities application

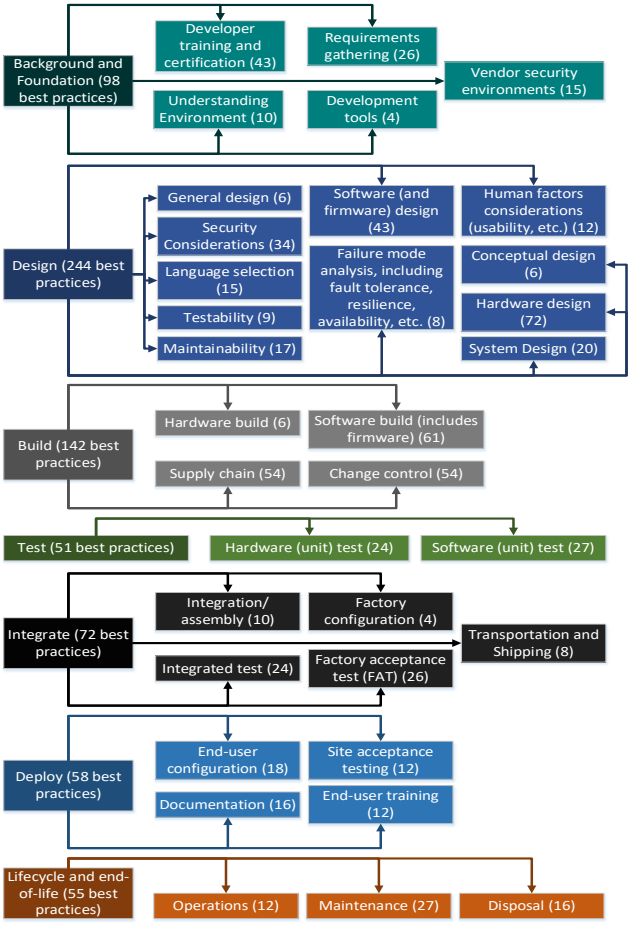

Figure 2: Domains and subdomains of the *Core* application

Comparative Evaluation: Estimating the progress of adopting best practices is not trivial, especially when the process involves a manual comparison of more than 700 best practices. Therefore, the comparative evaluation sub-application automates the comparison process between various assessments from the past and the current assessment. This sub-application has built-in data analytics to provide extensive analysis of the findings. An illustrative comparison of the nine assessments is shown in Figure 3.

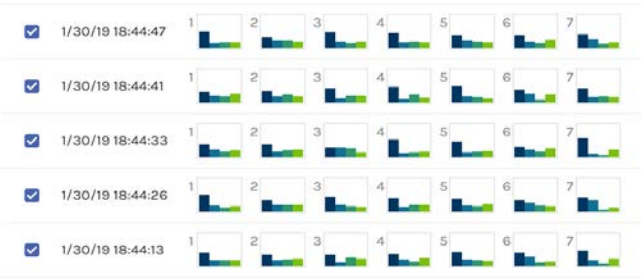

Figure 3: *Comparative evaluation* sub-application illustration

Other Software Features: The SD2-C2M2 software application is a sophisticated tool that hosts a plethora of user-friendly features. Because SD2-C2M2 is designed with security in mind, total control to host and secure the data is given to the end user. Therefore, SD2-C2M2 is built without any backend, making it a very light software application. However, several features let users save the findings and avoid redundancy, including the following:

- *Cached Progress:* Responses to the assessment questions are saved in the browser cache. The user can use this feature to complete the assessment over multiple sessions instead of doing it in one sitting.
- *Load/Save Progress:* The assessment progress can be saved to a file, which facilitates comparison over time. In a medium to large organization, software and hardware design processes are often spread across multiple teams. This feature will let the users finish their portion of the assessment and share it with the other teams to complete the remaining portion of the assessment. The teams are not forced to do the assessment together, which may be impractical to expect in a large organization. The *load* feature lets the users download the assessment, which can also be used in the comparative evaluation sub-application.
- *Export PDF Report:* At the end of an assessment, the SD2-C2M2 webtool generates a report with interactive graphics and data visualizations in the web portal. The report can also be exported as a .pdf file for portability and archiving. The HTML version of the report is dynamic and interactive and lets the user navigate between the results and various sub-tools on the fly.

3.2 Data Visualizations

Pie summary: The pie summary shown in Figure 4 provides a brief overview of the evaluation results. The pies are organized by domain and MIL. By default, the pie summary is depicted as follows: MIL2 pie includes the responses to both MIL1 and MIL2 questions. Similarly, MIL3 pie includes the responses to MIL1, MIL2, and MIL3 questions. The tool also generates a second version of this summary that does not combine lower MILs (independent pie summaries). In the SD2-C2M2, MIL1 indicates that the initial practices performed may be in *ad hoc* manner; MIL2 indicates that the practices are documented, stakeholders are involved, and adequate resources are provided and used; MIL3 indicates that the procedures and systems are reviewed in conformance and are guided with policies. MIL3 also emphasizes strict access controls, roles, and responsibilities.

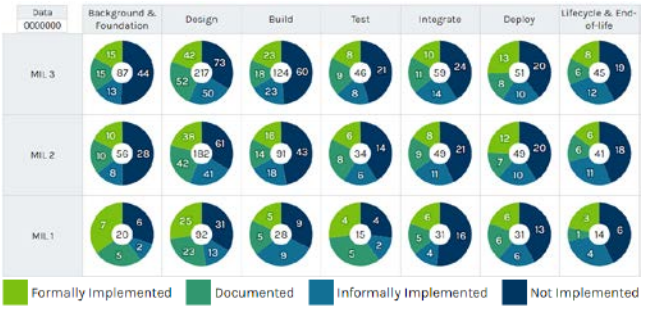

Figure 4: Illustrative pie summary

Timeline graph and tabular plot: This graph, shown in Figure 5, is generated from the comparative analysis application. When the user imports multiple assessment files, this sub-application generates two main visualizations: (1) a line plot with a "total implementation" line of maturity of all domains (the Y-axis for this graph is "percentage implemented"), and (2) a tabular bar chart plot that shows the overall differences between the loaded assessments, as shown in Figure 3.

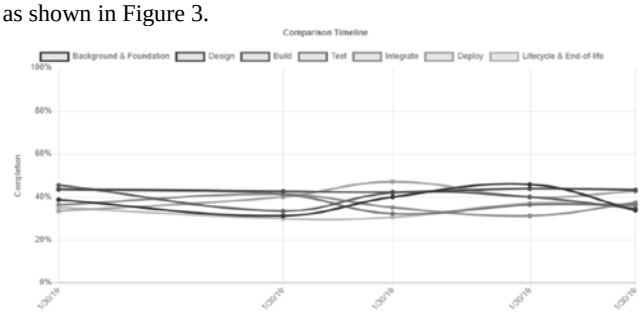

Figure 5: Illustrative timeline graph

4 SD2-C2M2 Use Case: Buffer Overflow Attack

The objective of this section is to provide a brief overview of the buffer overflow attack and demonstrate the efficacy of the SD2-C2M2 methodology and software application by testing it on a case study to demonstrate how to adopt best practices to defend against buffer overflow attacks. This section begins with an enumeration of various common weakness enumerations (CWE) that are related to buffer overflow attack. Then, we demonstrate the use of SD2-C2M2 to mitigate those CWEs.

4.1 Significance of Buffer Overflow Attack

Buffer overflows are one of the most common software vulnerabilities whose exploitation can result in a severe impact on the software program. The "SANS/CWE Top 25 vulnerabilities" [4] report identifies three common weakness enumerations (CWEs) [5] that are directly related to buffer overflow (see Table 1).

Table 1 CWE that are related to buffer overflow attacks

CWE	Rank	Description	Consequence
120	3	Buffer copy without checking size of input (classic buffer overflow)	Unverified input leads to buffer overflow that can cause the application to crash or put the program into an infinite loop.
190	24	Integer overflow or wraparound	Wraparound results in buffer overflow, which results in

			memory corruption and undefined behavior of the program.
131	20	Incorrect calculation of buffer size	Incorrect calculation leads to an out-of-bound read or write that results in an application crash or exposure of sensitive data.
798	7	Use of hard-coded credentials	Loss of credentials can lead to access to the software.
129	>25	Improper validation of array index	Untrusted array index leads to potential modification of memory sequences.
789	>25	Uncontrolled memory allocation	Uncontrolled memory allocation leads to out-of-memory issues and application crash.
191	>25	Integer underflow	Trigger buffer overflow by executing arbitrary code.
805	>25	Buffer access with incorrect length	Incorrect length puts the program in an infinite loop that leads to application crash.
119	>25	Improper restriction of operations within the bounds of a memory buffer	Execute unauthorized code lead to buffer overflow.

Based on the above analysis, it is evident that multiple CWEs are related to buffer overflows. Therefore, we chose to demonstrate the use of SD2-C2M2 to prevent buffer overflow attacks on an illustrative software system.

4.2 An Overview of a Buffer Overflow Attack

A buffer overflow attack takes control of the user input and places data in the buffer that is beyond the buffer's allocated size [6]. This causes data to be unexpectedly overwritten, leading to a system crash or facilitating injection of malicious code by an attacker.

Based on the review of a buffer overflow attack, the following are the identified security gaps: (1) input checking needs to be carefully monitored and validated; (2) file access and user permissions must be kept in mind while programming; (3) password and authentication should not be hard-coded; (4) program code auditing practice should be mandatory, (5) during design, the data segment of buffer space should be placed in a non-executable zone,; (6) patch updates should be mandatory; and (7) buffer size limitations need to be enforced [7]. Using the above overview, a fictitious facility is used as for a case study in the next section.

4.3 Case Study

The objective of this section is to demonstrate the usability and efficacy of the SD2-C2M2 software application and the best practices that are inherent to SD2-C2M2. To achieve that purpose, an illustrative case is developed based on the analysis detailed in various SANS reports about buffer overflow attack [6] [7] [8].

4.3.1 Attack Definition

A group of adversaries called *AttackBuffers* excels at identifying the buffer overflow vulnerabilities in the hard drive of a computer system server located in an organization's facilities. The adversary group also excels at identifying the exact buffer size of the software program of that server and takes control of the buffer. *AttackBuffers* then creates a specially crafted save file and stores it on the hard drive. The buffer is overflowed by the save file causing the server to crash, resulting in significant damage to the organization.

4.3.1 Illustrative new establishment: MIL0 Organization

Initially, the organization *NewOrgSoft* started at MIL0 with no best practices in place to defend against buffer overflow attacks. Therefore, the overall maturity of the organization looks like the illustration in Figure 6. In this state, the *NewOrgSoft* is vulnerable to attacks from *AttackBuffers*. The cybersecurity team at *NewOrgSoft* made it an absolute priority to implement and improve the cybersecurity best practices progressively based on the timeline shown in Table 2. Throughout the next sections, all the best practices adopted to reach a particular MIL are discussed. Note that the CWEs shown in the previous section are evaluated throughout each maturity phase and are given one of the four ratings: Not Implemented (NI), Partially Implemented (PI), Documented (D), and Formally Implemented (FI). As shown in Table 3, none of the CWEs are addressed at MIL0 state.

Table 2 Cybersecurity improvement timeline

Dates	Organization State	Target Maturity
D1: m1/d1/y1	Establishment	MIL0
D2: D1 + 6 months	Basic Security	MIL1
D3: D2 + 6 months	Medium Security	MIL2
D4: D3 + 6 months	Advanced Security	MIL3

Table 3 CWE evaluation for MIL0 at date D1

CWE	Status
CWE-120	NI
CWE-190	NI
CWE-131	NI
CWE 798	NI
CWE-129	NI
CWE-789	NI
CWE-805	NI
CWE-119	NI

4.3.2 MIL1 Best Practices Related to Buffer Overflow Attack

NewOrgSoft set a target to reach MIL1 before 6 months from the start of improvements (i.e., by date D2 in Table 2). Out of the more than 700 best practices, the following MIL1 best practices are related to buffer overflow. Therefore, the practice sets (PSs) below need to be fully implemented to have an efficient first line of defense against buffer overflow attacks and to achieve MIL1 status:

- ***PS1:*** Software developers are required to undergo formal training for the relevant programming languages and security best practices.
- ***PS2:*** A formal software functional and non-functional requirement gathering process that follows recognized standards should be enforced.
- ***PS3:*** Vulnerability disclosure procedures and breach notification procedures should be in place and periodically updated.
- ***PS4:*** With the focus on designing the software for integrity, the design process should include failure mode analysis and measures to handle out-of-bounds logical parameters.
- ***PS5:*** All the software interfaces between the components should follow recognized standards and be formally documented. Software language selection should be

considered during the design considering the requirement to ensure that the software is tolerant to input error.

- ***PS6:*** The software should be designed with defense-in-depth concepts, and the testability of software components and the assembled system should be built into the design. Periodically, the software test libraries should be updated to reflect special cases and conditions that trigger "bug-fix" modifications.
- ***PS7:*** For effective testing purposes, the software test plans and test libraries should include abnormal tests and test sets.
- ***PS8:*** The software should be developed with coding techniques to validate the input.
- ***PS9:*** Software testing procedures should include regression testing when the components are changed to confirm all problem reports preventing shipping have been resolved.
- ***PS10:*** Site acceptance tests that include validation of the software should be conducted using customer data and environments. Problem report resolutions should be delivered at the end of applying and testing the site acceptance tests. All patches should be tested prior to software release to customers and customer documentation/guides should include instructions for reporting bugs.

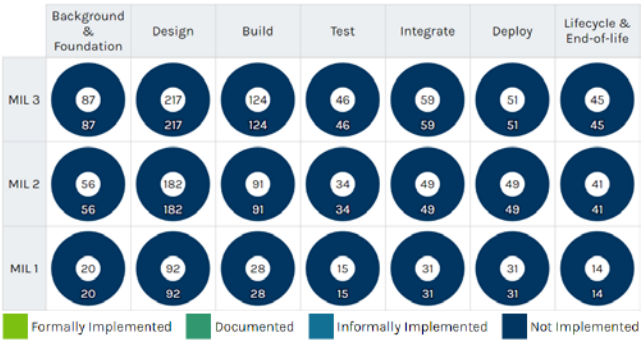

Figure 6: MIL0 maturity of the organization at date D1

Upon achieving the FI status for all the above-listed best practices, the distribution and overall maturity is as shown in Figure 7. It can be observed in Figure 7 that the numbers in the *green* portion of the pies are unchanged across MIL1, 2, and 3. This is because none of the MIL2 and MIL3 best practices are FI. Because MIL1 is a subset of MIL2 and MIL2 is a subset of MIL3, the number of best practices that are FI remains unchanged by date D2. Based on the maturity shown in Figure 7, improvements across each domain are shown in Table 4, and the CWEs addressed because of the improvements are shown in Table 5. Note that the best practices that are FI are only related to buffer overflow attack. Therefore, in regard to buffer overflow attacks, *NewOrgSoft* is at MIL1. However, as shown in Figure 7, not all best practices from MIL1 are FI. Therefore, the overall grade of *NewOrgSoft* is still MIL0.

Table 4 Cybersecurity improvements achieved by date D2

Domain	Total	FI	% Improve
Background and Foundation	87	6	~7%
Design	217	13	~6%
Build	124	1	~0.9%
Test	46	1	~2%
Integrate	59	5	~0.7%
Deploy	51	3	~6%
Lifecycle and End-of-Life	45	1	~2%

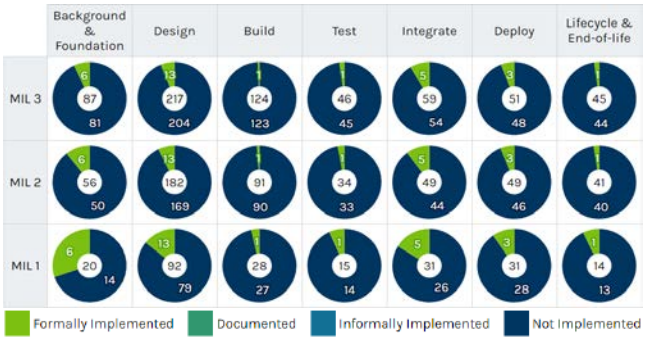

Figure 7: MIL1 maturity of the organization at date D2

Table 5 CWE evaluation for MIL1 at date D2

CWE	Status	Related PS
CWE-120	PI	1, 5, 8
CWE-190	PI	4, 5, 6, 8
CWE-131	D	6, 8, 9
CWE 798	PI	1, 4, 6
CWE-129	D	5,8
CWE-789	FI	8
CWE-805	PI	1, 5, 8
CWE-119	PI	1, 5, 8

4.3.3 MIL2 Best Practices Related to Buffer Overflow Attack

By successfully reaching the target MIL1 state by date D2, *NewOrgSoft* set a new target to reach MIL2 before 6 months from the date D2 (i.e., by date D3 in Table 2). To achieve MIL2 with respect to buffer overflow attacks, in addition to the previously stated best practices, the following MIL2 best practices are required to achieve the status of *FI*.

- ***PS11:*** Software developers are required to undergo formal training on development tools, environments, and local development practices/tools. They should be trained in technical and secure coding concepts. The training topics should also include cybersecurity topics such as vulnerability analysis, programming-language-based security, and source code analysis techniques. Cybersecurity training material should be updated upon making any significant change in the software environment and the developers should be retrained using the updated material.
- ***PS12:*** The software requirements gathering process should identify support requirements and the software should be designed with graceful degradation.
- ***PS13:*** The design process should consider the security of the software system and include vulnerability analysis procedures, procedures to securely interface with external systems, and considerations for fault-tolerant designs.
- ***PS14:*** The software development programming language should consider and evaluate risks inherent in the selected language.
- ***PS15:*** The software should be designed to handle conflicting or misleading inputs. For example: conflicting temperature readings for the same process element.

- ***PS16:*** The software security assessment process should include evaluation of the interfaces between software components.
- ***PS17:*** The software should be built using coding techniques to practice defense in depth through secure coding practices such as regular source code reviews and automated scanning of source code.
- ***PS18:*** The software should be built using well-defined data structures and should be able to take advantage of built-in programming language features.
- ***PS19:*** All the source code should be stored in a secure repository and strict access controls should be imposed to only allow authorized users to read, write, and execute.
- ***PS20:*** All the third-party libraries or open-source software modules that are used to achieve end-product development should be procured from reliable sources. Those libraries and modules should be scanned and analyzed for vulnerabilities.
- ***PS21:*** All software updates should be regression tested and the test procedures should include input limit (out-of-bounds) testing. The test procedures should also include active scanning, run-time verification tests, performance analysis, stress testing, and software vulnerability testing.
- ***PS22:*** Software test libraries should be updated to reflect special cases/conditions that trigger "bug-fix" modifications.
- ***PS23:*** Factory acceptance testing (FAT) should include vulnerability scanning.
- ***PS24:*** Regression tests should be performed to validate patch installation. Software patch documentation should contain a list of resolved issues.
- ***PS25:*** End-user training should include integration of the software with other systems (both hardware and software) and methods for monitoring the security of the software.

Upon achieving the FI status for all the above-listed best practices, the distribution and overall maturity is as shown in Figure 8. It can be observed in Figure 8 that the numbers in the *green* portion of the pies are different for MIL1 and MIL2 but remained unchanged for MIL3. This is because not all of the MIL3 best practices are FI. Based on the implementation shown in Figure 8, improvements across each domain are shown in Table 6 and the CWEs addressed because of the improvements are shown in Table 7. Note that the best practices that are FI are only related to buffer overflow attack. Therefore, with respect to buffer overflow attacks, the organization is at MIL2. However, it can be seen in Figure 8 that not all the best practices from MIL1 and MIL2 are FI. Therefore, the overall grade of *NewOrgSoft* still remains at MIL0.

Table 6 Cybersecurity improvements achieved by date D3

Domain	Total	FI	% Improve
Background and Foundation	87	18	~21%
Design	217	25	~12%
Build	124	13	~11%
Test	46	6	~13%
Integrate	59	11	~19%
Deploy	51	6	~12%
Lifecycle and End-of-Life	45	2	~4%

Table 7 CWE evaluation for MIL2 at date D3

CWE	Status	Related PS
CWE-120	D	11, 14, 15, 17, 18, 19
CWE-190	FI	14, 15, 17, 18
CWE-131	FI	15, 17
CWE 798	D	11, 13
CWE-129	FI	14, 21
CWE-789	FI	*FI in MIL1*
CWE-120	D	11, 14, 15, 17, 18, 19
CWE-120	D	11, 14, 15, 17, 18, 19

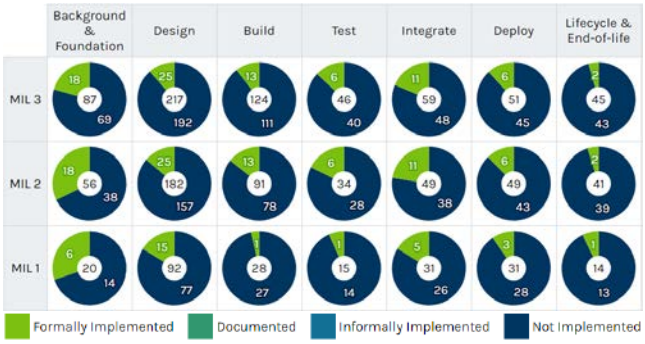

Figure 8: MIL2 maturity of the organization at date D3

4.3.4 MIL3 Best Practices Related to Buffer Overflow Attack

By successfully reaching the target MIL2 state by date D3, *NewOrgSoft* set a new target to reach MIL3 before 6 months from the date D3 (i.e., by date D4 in Table 2). To achieve MIL3 with respect to buffer overflow attacks, in addition to the previously stated best practices (MIL1 and MIL2), the following MIL3 best practices are required to achieve the state of *FI*. Achieving MIL3 indicates that *NewOrgSoft* has holistic defensive systems in place to protect and defend against the attacks from *AttackBuffers*.

- ***PS26:*** In-depth cybersecurity training course modules should be in place for the software development and testing tasks. Cybersecurity training topics should also include source code analysis tools.
- ***PS27:*** Software developers should receive annual refresher training in secure coding concepts.
- ***PS28:*** The software design process should consider misuse mitigation during the development process.
- ***PS29:*** The software should be designed with fault resilience (component restart). The programming language selection should consider issues raised in ISO/IEC 24772 [9].
- ***PS30:*** The software should be designed to prevent the spread of faults and the software assessment process should include additional security attributes.
- ***PS31:*** The software designs should be *red-teamed* to detect unanticipated design vulnerabilities and the designs should include methods to determine if the software is *under attack*. The design process should include an attack surface analysis.
- ***PS32:*** The software should be built with defensive coding techniques and all the third-party libraries or open-source software modules that are used to achieve the end-product development should be scanned with static and dynamic software analysis tools for malicious code.
- ***PS33:*** The software test procedures should include stress testing and input *fuzzing* testing.

The overall maturity of the facility after achieving the FI status for all the above-listed best practices is shown in Figure 9. As shown in Figure 9, the numbers in the *green* portion of the pies are different for MIL1, MIL2, and MIL3. This is because at this point, all the best practices related to buffer overflow defense under MIL1, 2, and 3 are FI. However, for the Test, Deploy, and Lifecycle & End-of-Life domains, no change is observed between the MIL2 and MIL3 statuses (see Figure 8 and Figure 9). This indicates that those three domains do not have any MIL3 best practices related to buffer overflow attacks. Based on the maturity shown in Figure 9, improvements across each domain are shown in Table 8 and the CWEs addressed because of the improvements are shown in Table 9. Note that the best practices that are FI are only related to buffer overflow attack. Therefore, with respect to buffer overflow attacks, the organization is at MIL3. However, the overall grade of *NewOrgSoft* still remains at MIL0.

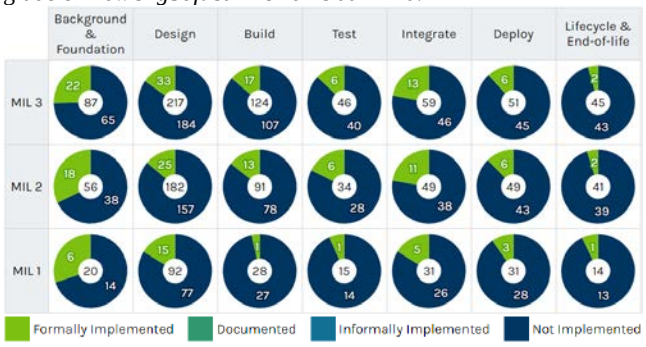

Figure 9: MIL3 maturity of the organization at date D4

Table 8 Cybersecurity improvements achieved by date D4

Domain	Total	FI	% Improve
Background and Foundation	87	22	~25%
Design	217	33	~15%
Build	124	17	~14%
Test	46	6	~13%
Integrate	59	13	~22%
Deploy	51	6	~12%
Lifecycle & End-of-Life	45	2	~4%

Table 9 CWE evaluation for MIL3 at date D4

CWE	Status	Related PS
CWE-120	FI	27, 29, 33
CWE-190	FI	*FI in MIL 2*
CWE-131	FI	*FI in MIL 2*
CWE- 798	FI	28, 29, 30, 31
CWE-129	FI	*FI in MIL 2*
CWE-789	FI	*FI in MIL 1*
CWE-120	FI	27, 29, 33
CWE-120	FI	27, 29, 33

4.3.5 Comparative analysis of MIL1, MIL2, and MIL3

Adoption of best practices is resource and time constrained. Therefore, an organization should target to reach MIL1 as the first objective. Then, the organization should continue progressing to reach MIL2 and finally MIL3. Figure 10 to Figure 16 show an in-depth comparative analysis across all the domains between all the fictitious organizations (all the green is related to buffer overflow and the blue is not related to buffer overflow).

As shown in Figure 10, for the domain *Background & Foundation*, large concentration of the best practices related to buffer overflow are in the *developer and training* subdomain followed by *vendor security environments* and *requirements gathering*. The subdomains *understanding environments* and *development tools* may be of less significance to buffer overflow. It is also evident in Figure 10 that ~27% of the best practices related to buffer overflow belongs to MIL1, ~55% to MIL2, and only ~18% to MIL3. A very different concentration is observed in the *Design* domain.

Figure 10: Background & Foundation

For the Design domain (see Figure 11), most best practices are in security considerations followed by system design, software (and firmware) design, testability, failure mode analysis, language selection, conceptual design, human factors considerations, and maintainability. The subdomains general design, product uses, and hardware design may not be significant to buffer overflow. According to Figure 11, ~39% of the best practices related to buffer overflow belong to MIL1, ~36% to MIL2, and ~24% to MIL3.

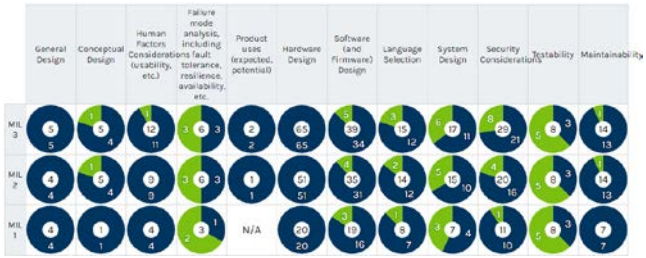

Figure 11: Design – (a) – (c): MIL1 – 3

In the Build domain shown in Figure 12, a large concentration of best practices is in *software build* followed by *supply chain* and *change control*. The subdomain *hardware build* may not be of any significance to buffer overflow. According to Figure 12, ~6% of the best practices related to buffer overflow belong to MIL1, ~71% to MIL2, and ~24% to MIL3.

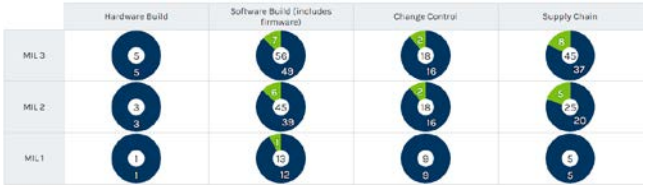

Figure 12: Build – (a) – (c): MIL1 – 3

The Test domain has only two subdomains. As shown in Figure 13, all the best practices related to buffer overflow attack/defense are in the *software (unit) test* subdomain. According to Figure 13, ~17% of the best practices related to buffer overflow belong to MIL1, ~83% to MIL2, and 0% to MIL3.

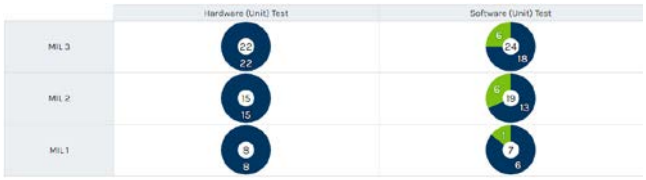

Figure 13: Test – (a) – (c): MIL1 – 3

In case of the Integrate domain (Figure 14), the only subdomains that are relevant to buffer overflow attack/defense are *integrated test* and *FAT*. According to Figure 14, ~39% of the best practices related to buffer overflow belong to MIL1, ~46% to MIL2, and 15% to MIL3.

Figure 14: Integrate – (a) – (c): MIL1 – 3

According to Figure 15, the best practices related to buffer overflow attack/defense are distributed across the *site acceptance testing*, *end-user training*, and *documentation* subdomains. The subdomain *end-user configuration* is not significant in buffer overflow analysis. According to Figure 15, 50% of the best practices related to buffer overflow belong to MIL1, 50% to MIL2, and 0% to MIL3.

Figure 15: Deploy – (a) – (c): MIL1 – 3

In the final domain Lifecycle and End-of-Life analysis shown in Figure 16, the only subdomain relevant to buffer overflow attack/defense is *maintenance*. According to Figure 16, 50% of the best practices related to buffer overflow belong to MIL1, 50% to MIL2, and 0% to MIL3.

Figure 16: Lifecycle and End-of-Life – (a) – (c): MIL1 – 3

5 Conclusion

In the absence of a clear value proposition, vendors of critical OT and EDSs will continue to rush to market critical technology—software and hardware—that is vulnerable to cyber-attacks. SD2-C2M2 provides an easy-to-use tool that facilitates the adoption of cybersecurity in the design and deployment process. Including cybersecurity in the process of developing these systems will help reduce the attack surface of critical systems in U.S. critical energy infrastructure. The case study presented suggests implementing this tool can prevent a host of cyber vulnerabilities in critical OT. The future work focuses on testing the tool at an EDS vendor that designs and builds software and hardware. Based on the feedback provided by the vendor subject matter experts, the SD2-C2M2 best practices and software tool will be updated in anticipation of public release.

REFERENCES

[1] C. Glantz, S. Somasundaram, M. Mylrea, R. Underhill and A. Nicholls, "Evaluating the Maturity of Cybersecurity Programs for Building Control Systems," Technical Report, Pacific Northwest National Laboratory, 2016.

[2] S. N. G. Gourisetti, M. Mylrea, E. Gervais and S. Bhadra, "Multi-Scenario Use Case based Demonstration of Buildings Cybersecurity Framework Webtool," in *IEEE Symposium on Computational Intelligence Applications in Smart Grid*, USA, 2017.

[3] U. S. Department of Energy, "Electricity Subsector Cybersecurity Capability Maturity Model (ES-C2M2), Version 1.1," February 2014. [Online]. Available: https://www.energy.gov/sites/prod/files/2014/02/f7/ES-C2M2-v1-1-Feb2014.pdf. [Accessed 7 February 2019].

[4] B. Martin, M. Brown, A. Paller and D. Kirby, "2011 CWE/SANS Top 25 Most Dangerous Software Errors," MITRE, 2011.

[5] Mitre, "Common Weakness Enumeration," Mitre, [Online]. Available: http://cwe.mitre.org/.

[6] M. E. Donaldson, "Inside the Buffer Overflow Attack: Mechanism, Method, and Prevention," SANS, 2002.

[7] J. Deckard, "Defeating Overflow Attack," SANS Institute, 2004.

[8] J. Nelißen, "Buffer Overflows for Dummies," SANS, 2002.

[9] ISO/IEC, "TR 24772, Information technology - Programming languages - Guidance to avoiding vulnerabilities in programming languages through language selection and use," ISO/IEC, 2010.