FRSecure CISSP Mentor Program

2023

Class #11 – Domain 8

Software Development Security

Ron Woerner, CISSP®

President of Something

CISSP® MENTOR PROGRAM – SESSION ELEVEN

FRSECURE CISSP MENTOR PROGRAM LIVE STREAM

THANK YOU!

Quick housekeeping reminder.

- The online/live chat that's provided while live streaming on YouTube is for constructive, respectful, and relevant (about course content) discussion **ONLY**.
- At **NO TIME** is the online chat permitted to be used for disrespectful, offensive, obscene, indecent, or profane remarks or content.
- Please do not comment about controversial subjects, and please **NO DISCUSSION OF POLITICS OR RELIGION**.
- Failure to abide by the rules may result in disabling chat for you.
- **DO NOT share or post copywritten materials. (pdf of book)**

GETTING GOING…

Managing Risk!

Study Tips:

- Study in small amounts frequently (20-30 min)
- Flash card and practice test apps help
- Take naps after heavy topics (aka Security Models)
- Write things down, say them out loud
- Use the Slack Channels
- Exercise or get fresh air in between study sessions

Let's get going!

GETTING GOING…

Great job last week! We're through Domain 7 (Security Operations)

- Shout Out to Chris and Ron for a great class!
- Ready for more?
  - We are close to the finish
  - USE the Discord to connect with others
  - Ask questions in #questions channel
- Check-in.
- How many have read Domain 8?

Let's get going!

Glossaries & Resources

- CISA Resources: https://www.cisa.gov/resources-tools/resources
- NIST CSRC Glossary: https://csrc.nist.gov/glossary
- Committee on National Security Systems (CNSS) Glossary, CNSSI No. 4009, April 6, 2015 - https://rmf.org/wp-content/uploads/2017/10/CNSSI-4009.pdf

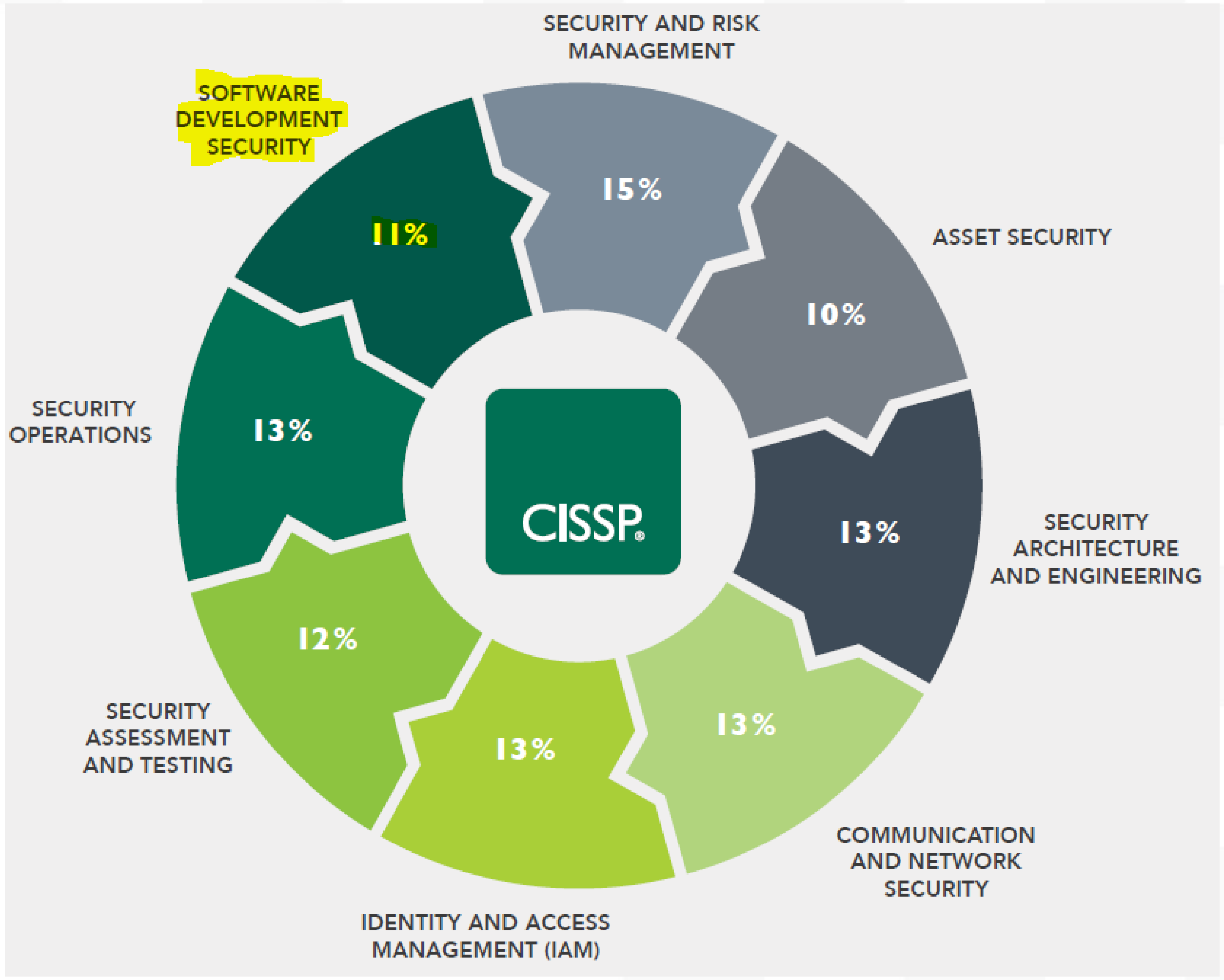
Lots of links to share tonight. Stay tuned to the end for the list.

CISSP® MENTOR PROGRAM – SESSION ELEVEN

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

CISSP **Exam Overview**

Caution!
Concepts overlap between domains.

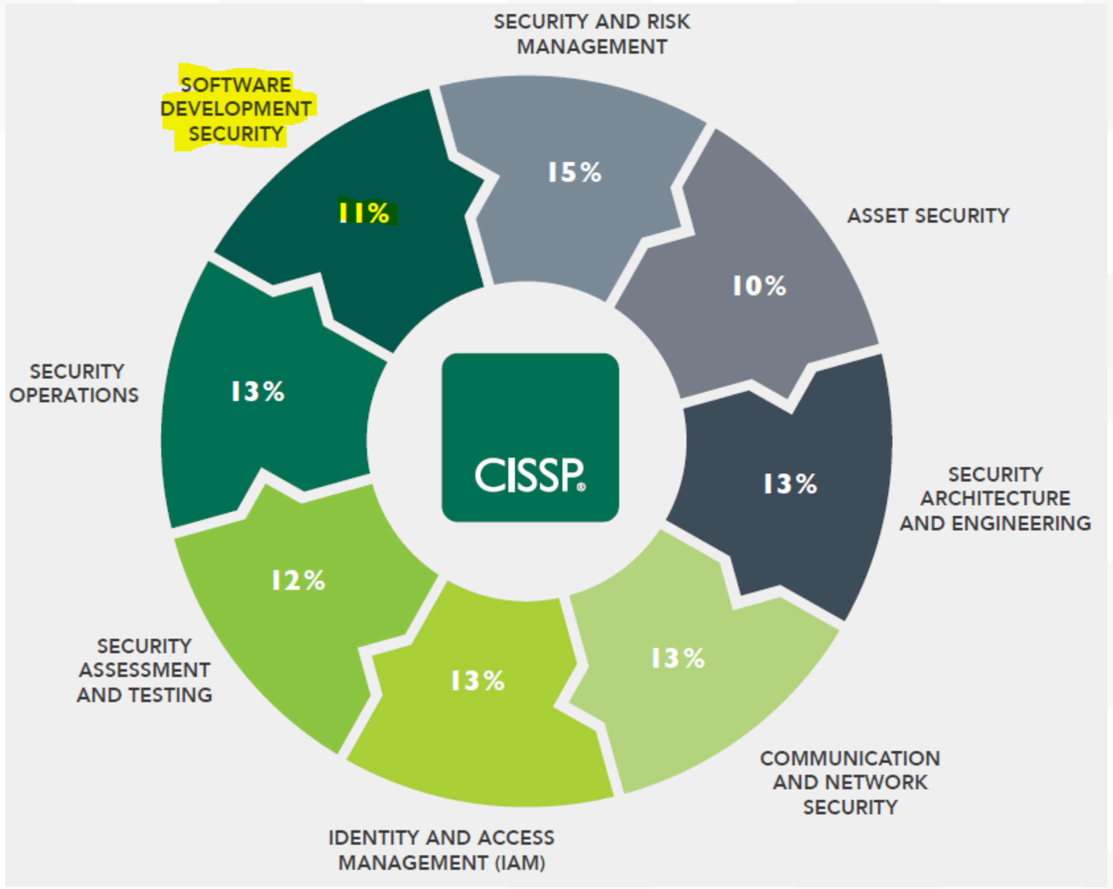


https://www.isc2.org/-/media/ISC2/Certifications/Ultimate-Guides/UltimateGuideCISSP-Web.ashx

CISSP® MENTOR PROGRAM – SESSION TEN

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY– OVERVIEW

- Understand and integrate security in the Software Development Life Cycle (SDLC)
- Identify and apply security controls in software development ecosystems
- Assess the effectiveness of software security
- Assess security impact of acquired software
- Define and apply secure coding guidelines and standards



https://www.isc2.org/Certifications/cissp/Certification-Exam-Outline

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Software Development Security is

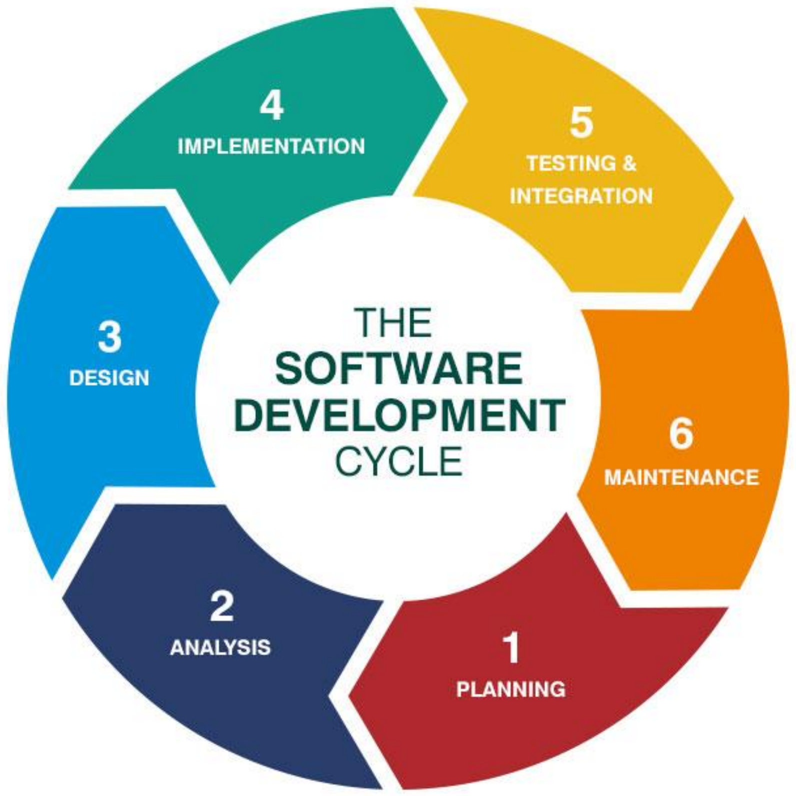
Design and organization of the components, processes, services, and controls appropriate to reduce the security risks associated with a system to an acceptable level.

Security Engineering Is

Implementation of that design

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Development Methodologies



CISSP® MENTOR PROGRAM – SESSION ELEVEN

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Development Methodologies

SECURING THE SOFTWARE SUPPLY CHAIN

RECOMMENDED PRACTICES GUIDE FOR DEVELOPERS

https://www.cisa.gov/sites/default/files/publications/ESF_SECURING_THE_SOFTWARE_SUPPLY_CHAIN_DEVELOPERS.PDF

OFFICE OF THE DIRECTOR OF NATIONAL INTELLIGENCE
UNITED STATES OF AMERICA
NATIONAL SECURITY AGENCY
CYBERSECURITY COLLABORATION CENTER
CYBERSECURITY & INFRASTRUCTURE SECURITY AGENCY
CISA
CSCC
DIB Sector Coordinating Council
National Defense ISAC
INFORMATION TECHNOLOGY
IT SCC
SECTOR COORDINATING COUNCIL

CISSP® MENTOR PROGRAM – SESSION ELEVEN

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Assess Security Impact of Acquired Software

COTS

SECURING THE SOFTWARE SUPPLY CHAIN

RECOMMENDED PRACTICES GUIDE FOR

CUSTOMERS

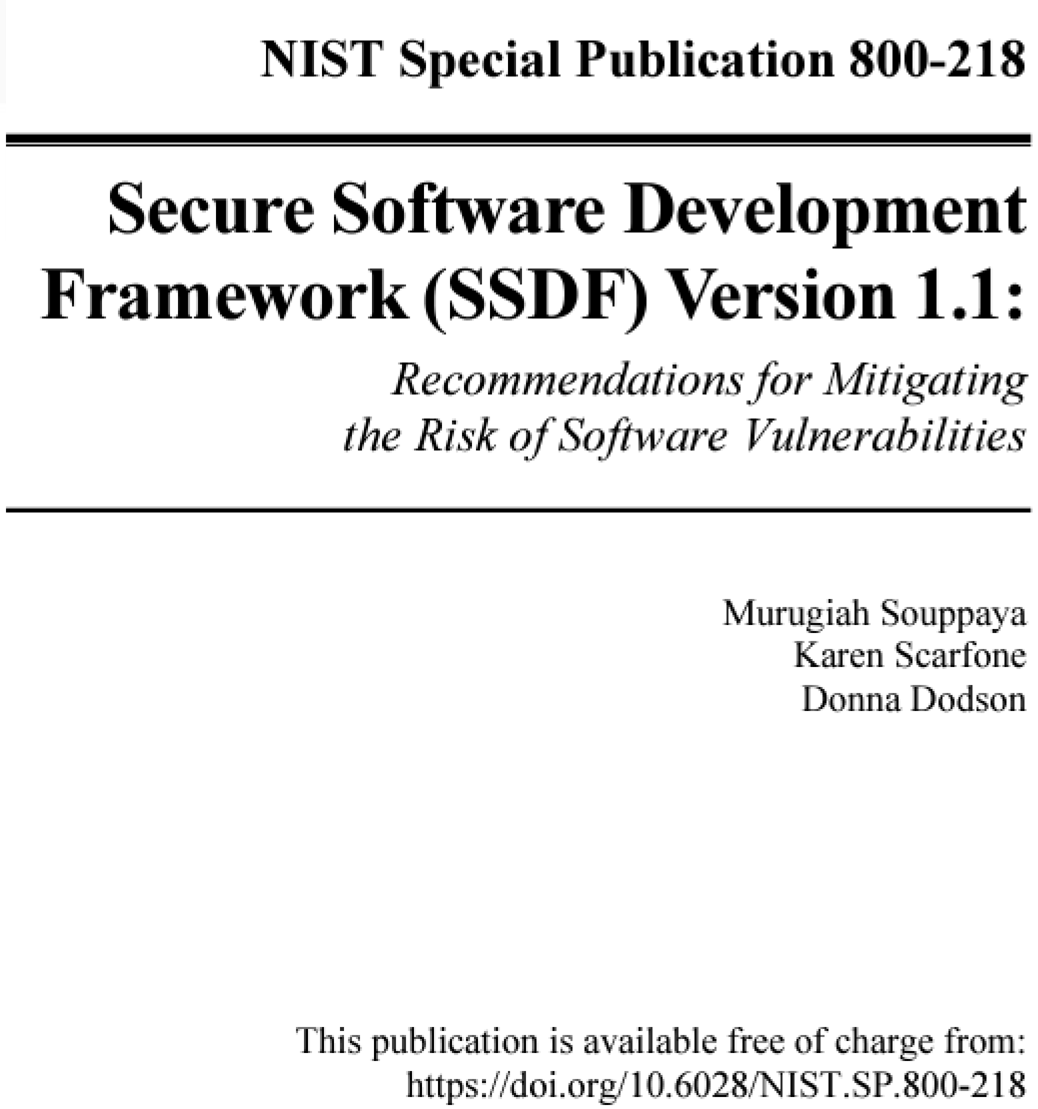
https://www.cisa.gov/sites/default/files/2023-01/ESF_SECURING_THE_SOFTWARE_SUPPLY_CHAIN_CUSTOMER.PDF

CISSP® MENTOR PROGRAM – SESSION ELEVEN

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Development Methodologies



NIST Special Publication 800-218

Secure Software Development Framework (SSDF) Version 1.1:

Recommendations for Mitigating the Risk of Software Vulnerabilities

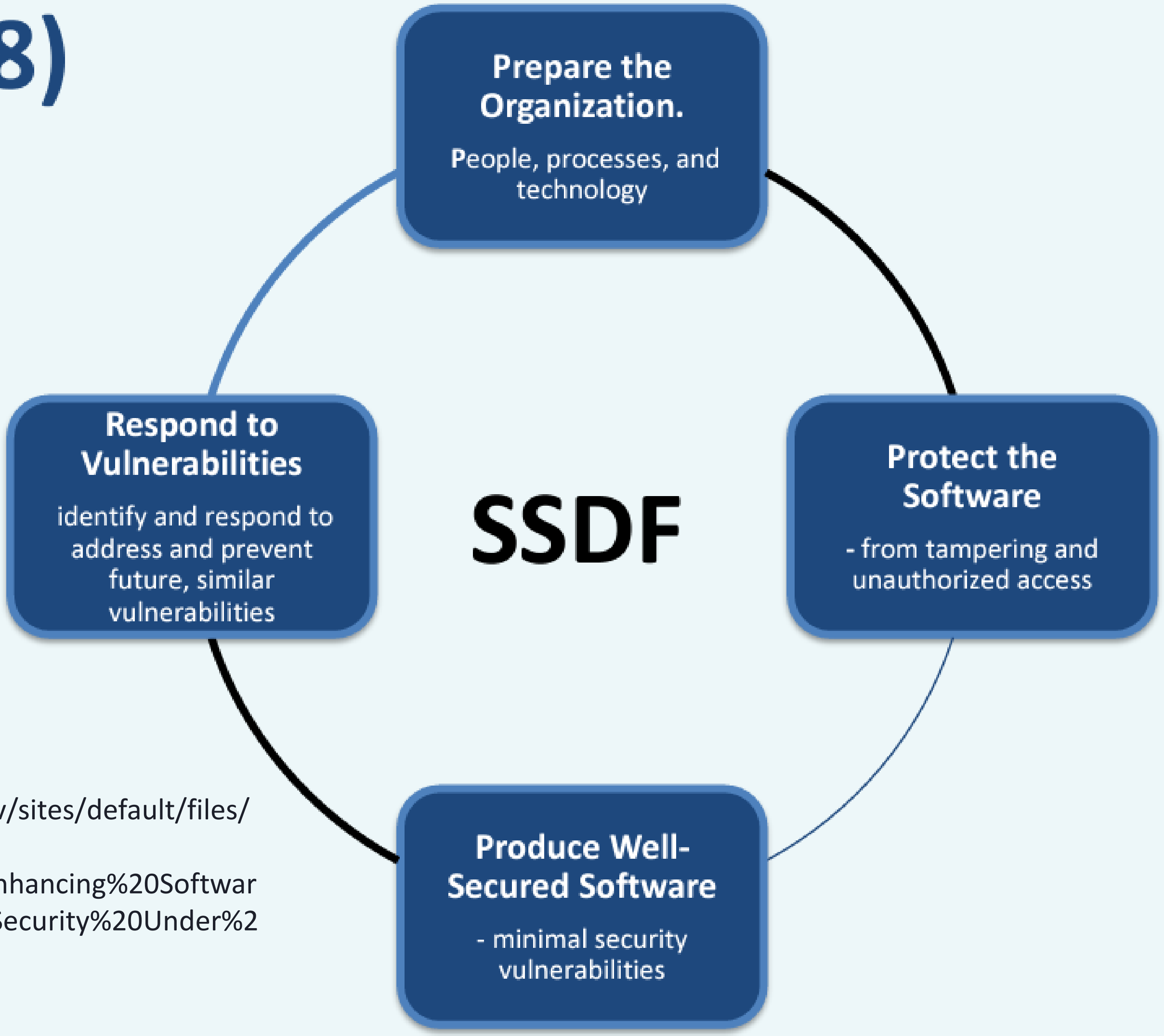
Murugiah Souppaya
Karen Scarfone
Donna Dodson

This publication is available free of charge from:
https://doi.org/10.6028/NIST.SP.800-218

https://csrc.nist.gov/publications/detail/sp/800-218/final

Secure Software Development Framework version 1.1 (NIST SP 800-218)

- Helps with adopting a risk management approach
  - document secure software development practices today
  - define future target practices.
- Provides a common lexicon & taxonomy
- Focus is on OUTCOMES not the "how"
- Leverages existing practices from established standards & guidance
- Broadly applicable to IT, IoT, OT


Prepare the Organization.
People, processes, and technology
Protect the Software
- from tampering and unauthorized access
Produce Well-Secured Software
- minimal security vulnerabilities
Respond to Vulnerabilities
identify and respond to address and prevent future, similar vulnerabilities
SSDF

https://www.nccoe.nist.gov/sites/default/files/2022-09/Guidelines%20for%20Enhancing%20Software%20Supply%20Chain%20Security%20Under%20EO%2014028.pdf

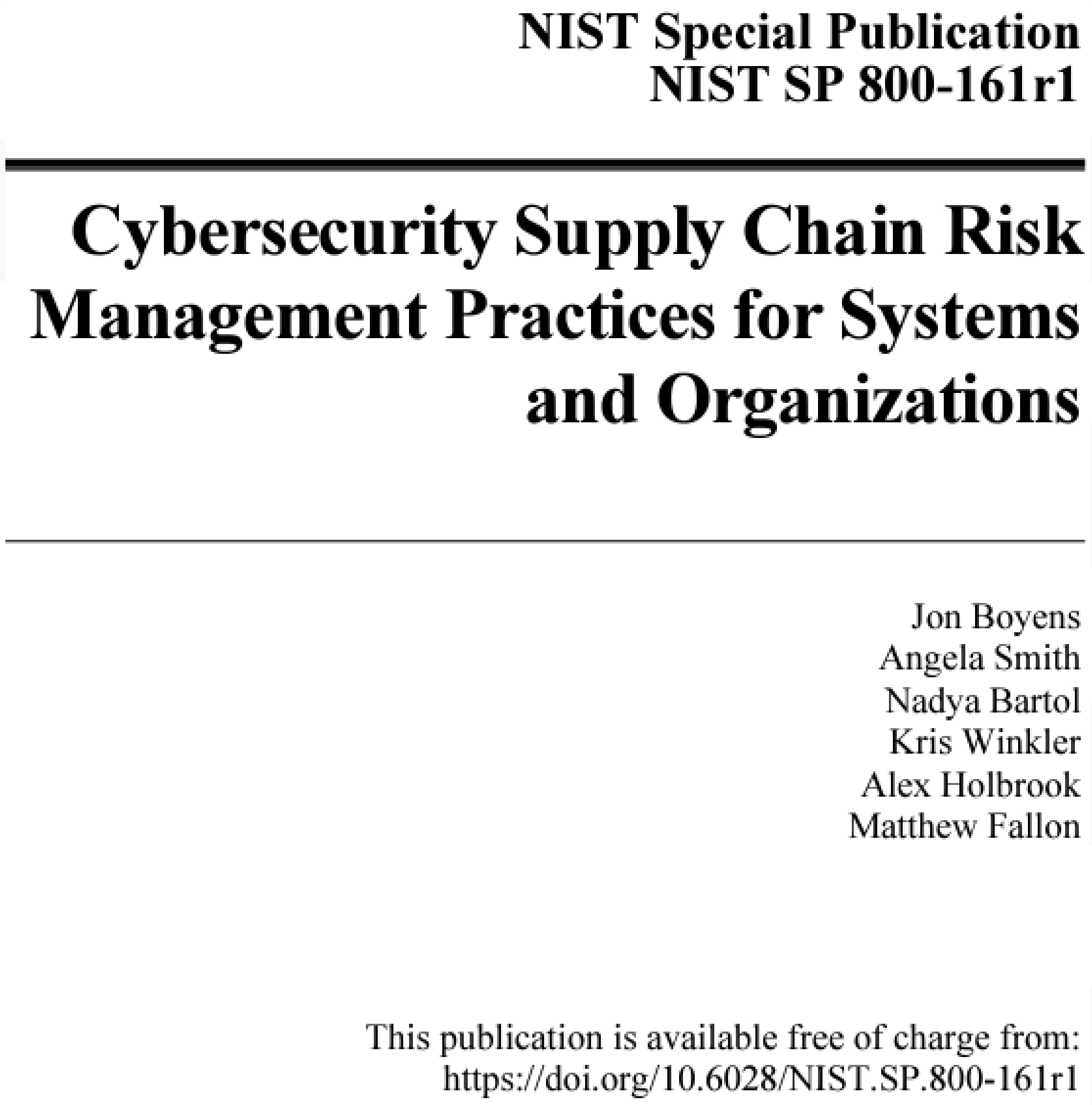
OMB M-22-18:** *Enhancing the Security of the Software Supply Chain through Secure Software Development Practices

CISSP® MENTOR PROGRAM – SESSION ELEVEN

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Development Methodologies


NIST Special Publication
NIST SP 800-161r1

Cybersecurity Supply Chain Risk Management Practices for Systems and Organizations

Jon Boyens
Angela Smith
Nadya Bartol
Kris Winkler
Alex Holbrook
Matthew Fallon

This publication is available free of charge from:
https://doi.org/10.6028/NIST.SP.800-161r1

https://csrc.nist.gov/publications/detail/sp/800-161/rev-1/final

CISSP® MENTOR PROGRAM – SESSION ELEVEN

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Development Methodologies

https://safecode.org/

https://safecode.org/wp-content/uploads/2018/03/SAFECode_Fundamental_Practices_for_Secure_Software_Development_March_2018.pdf

Fundamental Practices for Secure Software Development

Essential Elements of a Secure Development Lifecycle Program

Third Edition

March 2018

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Development Methodologies

Initiation – The business need and case for a system is expressed, requirements are documented, and resources are allocated.

Development – Activities are conducted to design the system and underlying architecture needed to meet requirements; then the system is created by developing custom code, purchasing or integrating external code, or otherwise constructing the system.

Deployment and delivery – The system is placed into the operating environment and made ready for use, which will include testing to ensure the system is fit for purpose and meets requirements.

Operations and maintenance – Most operations and maintenance activities are outside the scope of development work, but the SDLC can be iterated as user needs evolve, so crucial processes like change management may kick off another round of development activities.

Disposal - Activities in this phase are almost entirely the purview of the SLC, but some development may be required to perform activities such as archiving or transitioning data to a replacement system.

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Development Methodologies (Waterfall)

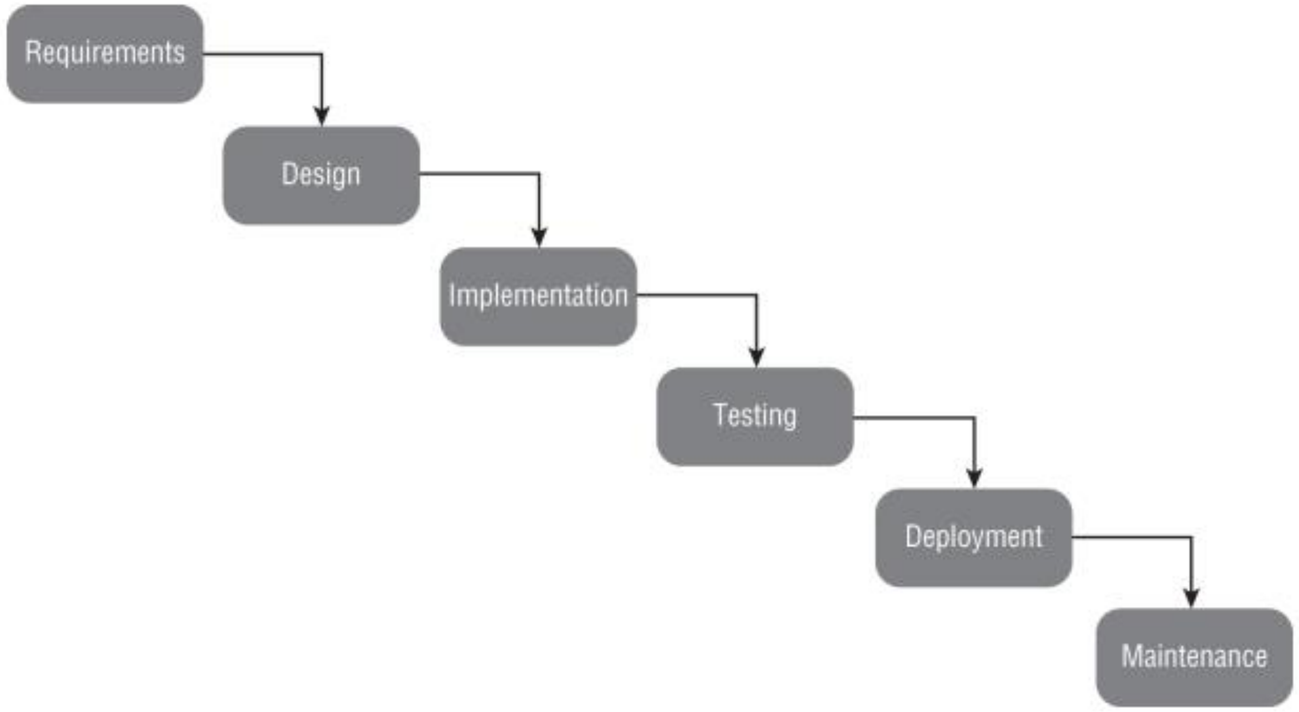
One of the oldest and has fallen out of favor in many organizations due to its inflexibility and difficulty in meeting the complex requirements of modern information systems.

Phases of a typical Waterfall development project include

- **Requirements**
- **Design**
- **Implementation**
- **Testing**
- **Deployment**
- **Maintenance**

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Development Methodologies (Waterfall)



Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Development Methodologies (Waterfall)

- Requires that **all tasks be completed** before the project progresses to the next phase.
- **Not designed to be iterative**, which presents difficulties when requirements change.
- To progress to the design phase, all requirements must be gathered, documented, and agreed upon.
- This rigor can be a security benefit, as the Waterfall method **enforces strict control processes** such as requirements gathering and testing.
- The rigidity of the Waterfall model makes it useful for projects where it is possible to know all requirements up front and where **no changes** will be required once development has started.
- **Rapidly evolving security requirements**, which may necessitate major changes to system functionality, are particularly **underserved by this model**.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Development Methodologies (Agile)

Created in 2001

12 principals in the Agile Manifesto

Not a development methodology but are instead a set of guiding ideas

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Development Methodologies (Agile)

- Software is **released quickly** with basic functionality; then the development process iterates to include additional functionality, avoiding large all-at-once releases in favor of smaller, more frequent releases.
- Each iteration will also take into account **lessons learned** from customers (e.g., changed requirements) as well as lessons learned by the development team.
- **All stakeholders** should be involved at a minimum during requirements gathering and testing to ensure the system needs are documented and tests are conducted to verify they are met.
- Changes to requirements, including **evolving security needs, are well supported by Agile methodologies**, since they are designed to re-assess requirements frequently and incorporate those in the iterative development.

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Development Methodologies (Agile)

Testing approaches

- High value on speed and efficient use of resources, which gives rise to the use of **integrated testing** to support the delivery of functional software.
- First, testing is on a **smaller, more manageable target**, so fixes are generally easier. Second, the need to be **responsive to changing requirements** necessitates robust testing to ensure newly delivered functionality does not introduce bugs, flaws, or vulnerabilities.
- Agile methodologies leads to a **strong focus on automation**.
- **Security testing** benefits from automation as well, as critical controls like software code reviews and vulnerability scanning can be **conducted automatically** and more frequently.

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Development Methodologies (Agile)

Extreme Programming (XP) – Introduces the use of integrated teams including developers, customers, and managers to drive the delivery of high-value software features. User stories are documented to capture what users are trying to achieve as well as the acceptance criteria of software that implements those features, and the stories are prioritized based on value.

XP utilizes a concept known as **pair programming**, which pairs developers. The developers take turns writing code and offering advice/input, with the goal of achieving higher quality code by providing extra oversight and knowledge to draw upon.

Refactoring code can mean many things, but it is essentially a way of removing obsolete, redundant, or unneeded code to improve software's functionality.

Developers may have made decisions in the past that were sound but are no longer relevant as the system and requirements evolved.

Refactoring can be used to ensure systems do not contain unnecessary junk code or functions known as cruft.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Development Methodologies (Agile)

Test-Driven Development (TDD) – is driven by the use of test cases: first a test is written, then it is run. If it fails, code is written or refactored as needed to make the test succeed; the ultimate goal is to **ensure that all tests pass**.

TDD is especially **effective for existing systems** where modifications or updates must be implemented with a minimum of complexity, such as implementing code-level security controls like fixing a buffer overflow or injection vulnerability.

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Development Methodologies (Agile)

Test-Driven Development (TDD) – Activities are conducted to design the system and underlying architecture needed to meet requirements; then the system is created by developing custom code, purchasing or integrating external code, or otherwise constructing the system.

Scrum– The system is placed into the operating environment and made ready for use, which will include testing to ensure the system is fit for purpose and meets requirements.

Operations and maintenance – Most operations and maintenance activities are outside the scope of development work, but the SDLC can be iterated as user needs evolve, so crucial processes like change management may kick off another round of development activities.

Disposal - Activities in this phase are almost entirely the purview of the SLC, but some development may be required to perform activities such as archiving or transitioning data to a replacement system.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Development Methodologies (Agile)

Scrum - lightweight set of team members continuously iterating development work to achieve the overall product goal.

Scrum master - role is to remove impediments to the team's success. These include tasks such as coaching team members on self- organizing work, gathering input from key stakeholders, and facilitating team events.

Product owner - voice of the customer, the product owner is responsible for ensuring developers understand the goals their software is being designed to meet. The product owner also owns the product backlog, where all work to be done is broken down into work units and prioritized, such as new features and bug fixes.

Development team - broad team that includes traditional developers who write code, architects who design system infrastructure, data scientists or analysts, and, crucially to security, testers and Quality Assurance (QA) staff. Security testing should be done for all development work, so many code- and application-level security controls will be delegated to members of the development team.

Backlogs - Product backlog contains **all requirements** the product owner has identified, along with prioritization based on value to the customer. **Sprint backlog** is a **smaller list** of requirements drawn from the product backlog that, in the team's estimation, can be delivered in the time allotted.

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Development Methodologies (Agile)

Key Scrum Activities.

Sprint- time-boxed set of work to be done on the project

Sprint planning- certain backlog requirements will be selected for the sprint, and the team agrees upon the work they feel can be achieved in the given time period

Daily Scrum- meet daily to discuss progress, blocking issues, and the plan for the day's activities. According to the Agile Manifesto, these meetings are best conducted face to face and are often known as standup meetings.

Sprint review and retrospective - Two key events happen at the end of a sprint.

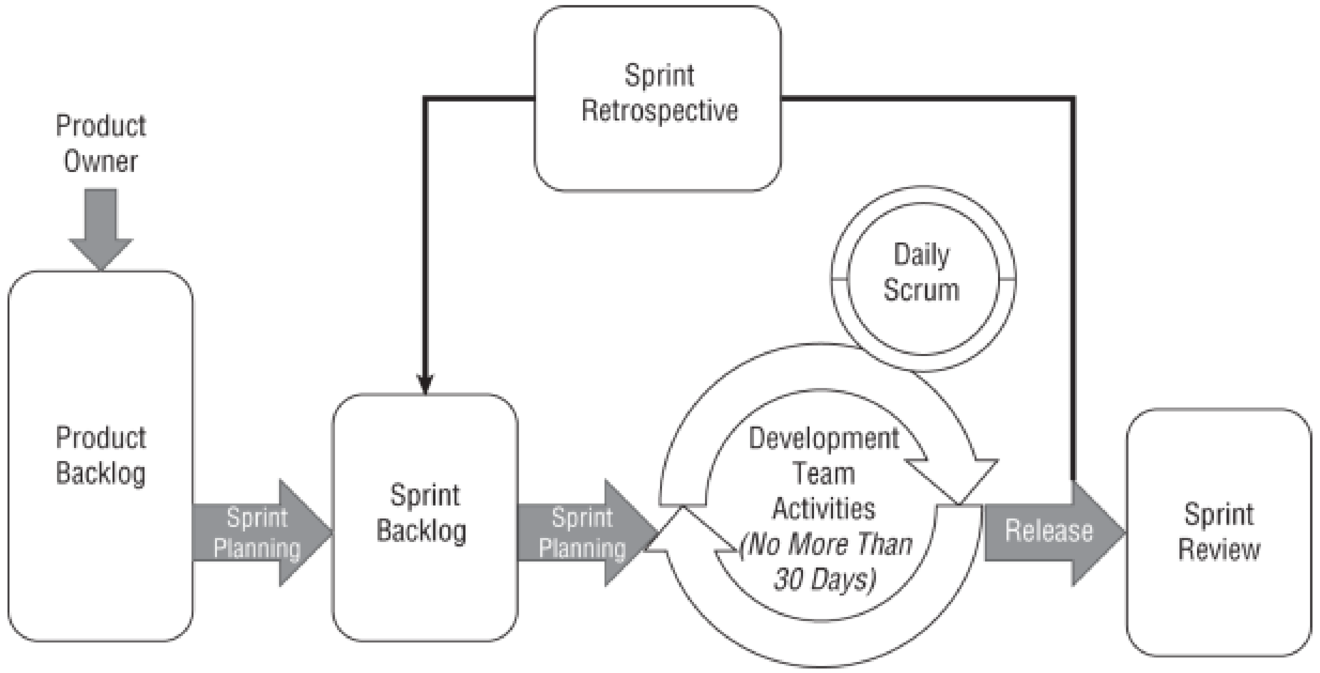
The **sprint review** identifies work completed and includes a demo to the customer, supporting the goal of integrating users and developers.

A **sprint retrospective** is an opportunity for the development team to perform continuous improvement by reflecting on successes, failures, and opportunities based on the recent sprint.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Development Methodologies (Agile)



DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

DevOps

Integration of the traditionally disparate functional areas of development and operations, with the goal of better supporting the organization's information system resources.

Include the use of **automation, frequent testing**, and an emphasis on frequent software integration and deployment.

Software delivery is sped up by following an assembly line approach, known as a **pipeline**, with a focus on **building quality** in.

QA is the third member of the DevOps team in addition to development and operations and is responsible for implementing many aspects of the **cross-functional communication**.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

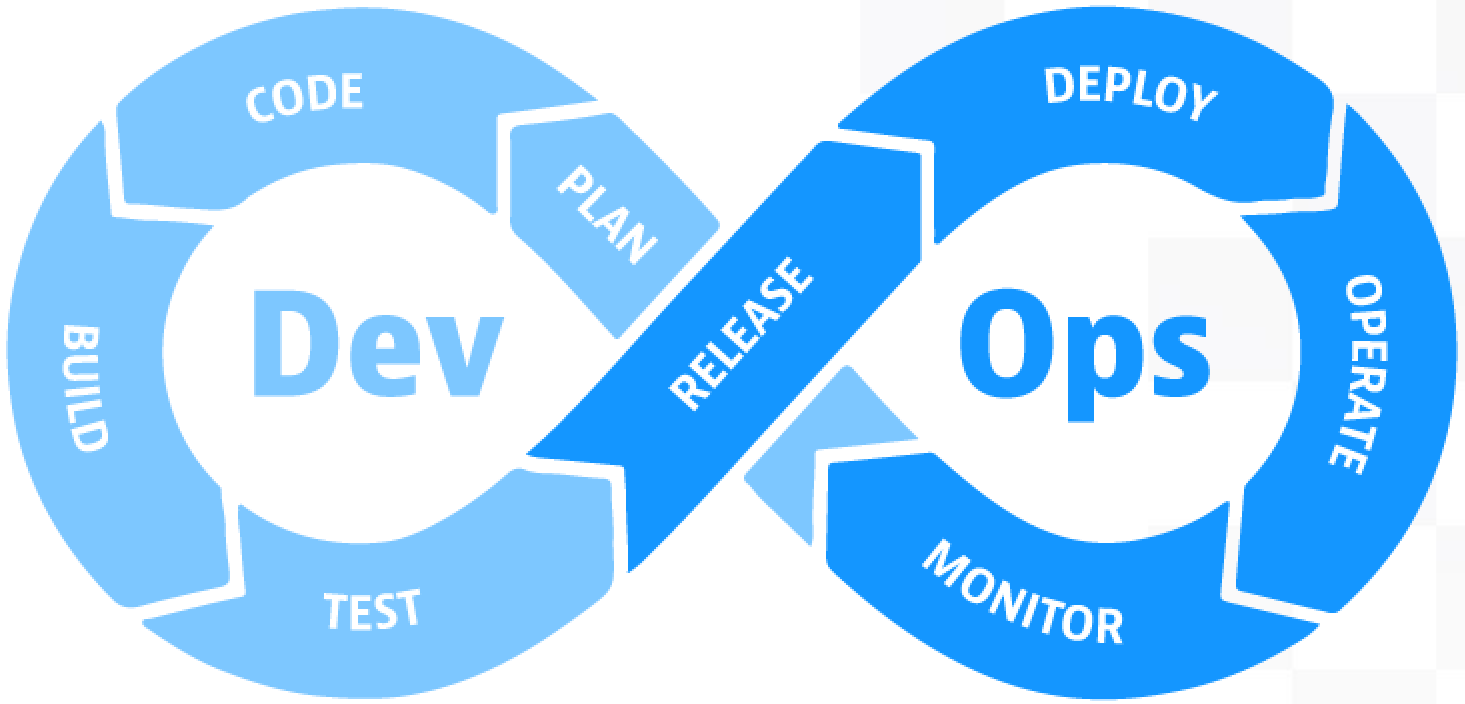
DevOps

- Members of the DevOps team are likely to have **privileged access**, such as being able to make changes in production environments, create new users and permissions, and access log data.
- Additional monitoring is likely justified, as are controls like multifactor authentication (**MFA**) and additional training for these staff members.
- Principles of least privilege and separation of duties must still be implemented.
- **Permissions specific only to their primary role**, rather than having access to all abilities granted to the DevOps team.
- Automation plays a crucial role, such as the use of an automated code deployment tool that systematically validates that all testing and other requirements have been met before pushing code to production.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

DevOps



DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

DevSecOps

DevSecOps Manifesto

- **Leaning in -** over Always Saying "No"
- **Data & Security Science -** over Fear, Uncertainty and Doubt
- **Open Contribution & Collaboration -** over Security-Only Requirements
- **Consumable Security Services with APIs -** over Mandated Security Controls & Paperwork
- **Business Driven Security Scores -** over Rubber Stamp Security
- **Red & Blue Team Exploit Testing -** over Relying on Scans & Theoretical Vulnerabilities
- **24x7 Proactive Security Monitoring -** over Reacting after being Informed of an Incident
- **Shared Threat Intelligence -** over Keeping Info to Ourselves
- **Compliance Operations -** over Clipboards & Checklists

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

DevSecOps

The goal of DevSecOps is to favor **iteration over perfection**, where a workable security solution delivered today and improved tomorrow is preferred over a perfect solution delivered years from now — by which time the system may have suffered attacks or exploits!

Integrating security and compliance activities and data into development processes.

DevSecOps practitioner, such as a **CISSP**, performs **red team testing** to determine the exact nature of a vulnerability.

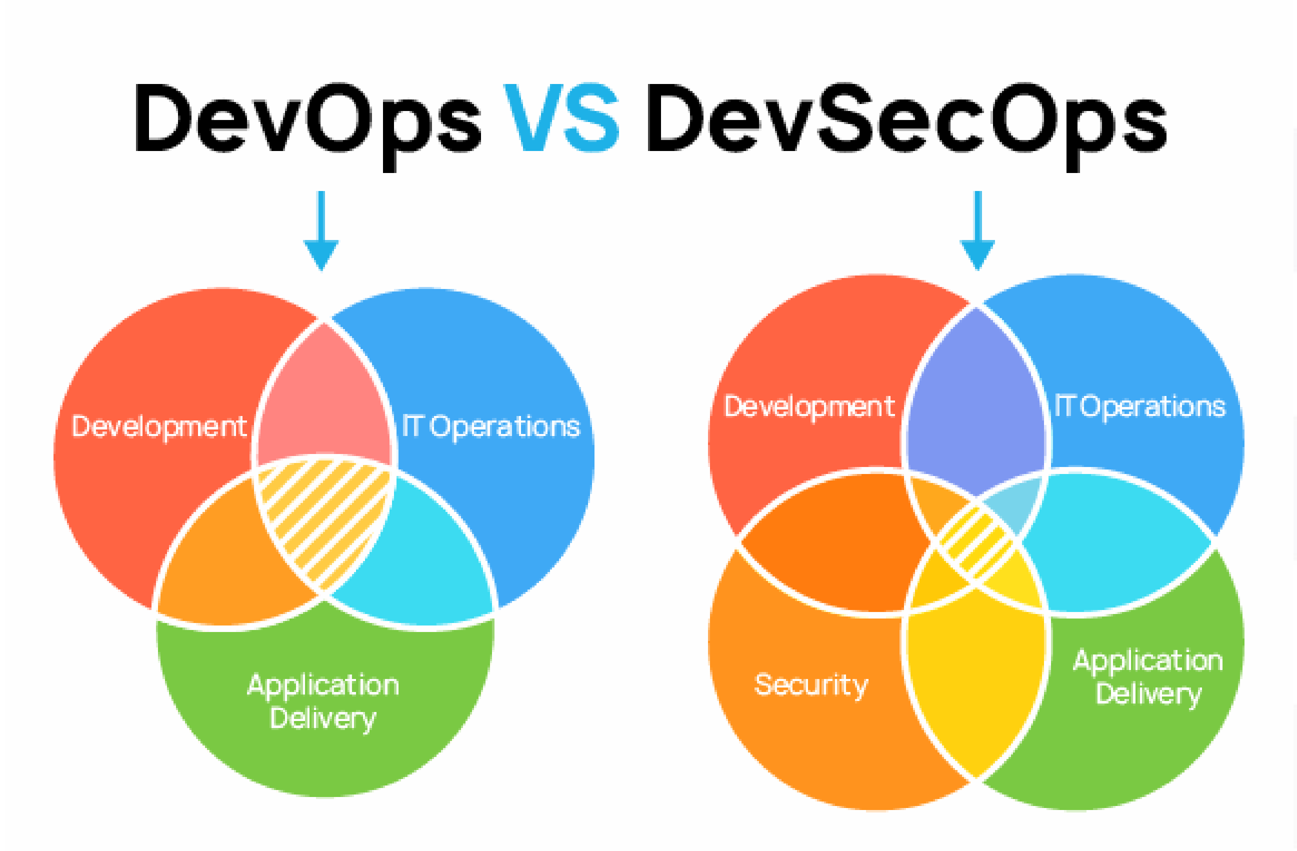
Ensure security is built into the process an organization uses to develop and operate its information systems.

CISSP® MENTOR PROGRAM – SESSION ELEVEN

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

DevSecOps



Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Other Methodologies

Modified Waterfall - First, it may be possible to proceed to the next phase before all current phase activities are finished. Second, it is possible to step back a phase and iterate work if necessary. If the developers realize that architectural assumptions made during design are off in some way, the project can step back one phase, update the system design to address the deficiencies, then progress to implementation once again.

Spiral - Designed to be executed in a repetitive series, and it places a heavy focus on risk assessment, analysis, and evaluation. Large, complex, and costly projects are a common use for this model, as it emphasizes risk control over time via repeated evaluation of project risks, costs, and benefits. The process iterates through four phases:

1.Determine objectives, alternatives, and risks

2. Evaluate alternatives and resolve risks

3. Development and testing

4. Plan next phase

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Other Methodologies

Prototype- simplified version of a system or app is built and released for feedback and review. A subsequent round of development is done based on the feedback and then released for additional review, and so on, until the stakeholder's needs have been met.

Cleanroom - assumes that flaws cannot be fixed once development is complete. It focuses exclusively on defect prevention. The ultimate goal is to produce software that meets a defined level of reliability, sometimes known as zero-defect, with a focus on robust design and implementation rather than testing and remediation.

Usually implemented with another development model such as Waterfall and demands a focus on gathering statistical data regarding flaws and control measures.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Maturity Models

- Provide a way of **measuring an organization's progress on continuous improvement** in a discipline, such as security management or software development.
- Business maturity models are often known as **capability maturity models** and are usually specific to a particular organizational discipline like software development.
- **Provide a baseline to measure the organization's current state** of ability, define a desired goal state, and provide systemic recommendations for improving the capability.
- Process improvement, and in general, an organization is **more mature when its processes are more formal, repeatable, and well-managed rather than chaotic.**

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Maturity Models (Common Maturity Model Components)

Domains - the processes or business aspects that the model is designed to **help measure** and are usually common activities such as measurement and analysis.

Levels - typically **ranging from 1 to 5**. Levels are arranged in a scale showing the organization's increasing capabilities and may describe individual process areas or the overall collection of processes described by the maturity model.

Criteria - to achieve a specific level of maturity, an organization **must meet the criteria for that level**. Logically, an organization cannot measure an ad hoc or undefined process, so that practice is not introduced until the organization matures fundamental practices.

Targets - describes the organization's **desired maturity level.** Just as security controls entail a cost benefit analysis of risk reduction, so too must the choice to improve processes lead to a defined organizational benefit.

While increased process maturity **reduces the risk** of failed or mismanaged projects, it also **introduces cost and complexity**; different organizations will choose different levels of desired maturity.

Target can be used to perform a **gap analysis and prioritize activities** needed to close identified gaps.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Maturity Models (Capability Maturity Model)

One of the earliest maturity models to be widely adopted, the Capability Maturity Model (**CMM**), was developed jointly by the U.S. Department of Defense and the Software Engineering Institute (**SEI**) at Carnegie Mellon University.

It arose from the **need to measure and improve** the burgeoning field of software development in the 1980s, as increasing processing power and decreasing costs led to widespread adoption of computer systems.

Development projects were often poorly estimated, controlled, and managed due to computer science being a nascent field at the time

SEI formalized a set of best practices common across successful projects. The initial purpose of the model was to evaluate the capabilities of government contractors to deliver projects on time and on budget, but other development and business processes began to adopt the maturity model mindset.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Maturity Models (Capability Maturity Model Integration)

CMMI Institute run by **ISACA** (https://cmmiinstitute.com/cmmi) and has been adapted to address Agile development as well as new areas of business concern including development, services, supplier management, and people (workforce management and professional development), as well as an emerging Cybermaturity program designed to apply maturity model concepts to governance and reporting of cybersecurity.

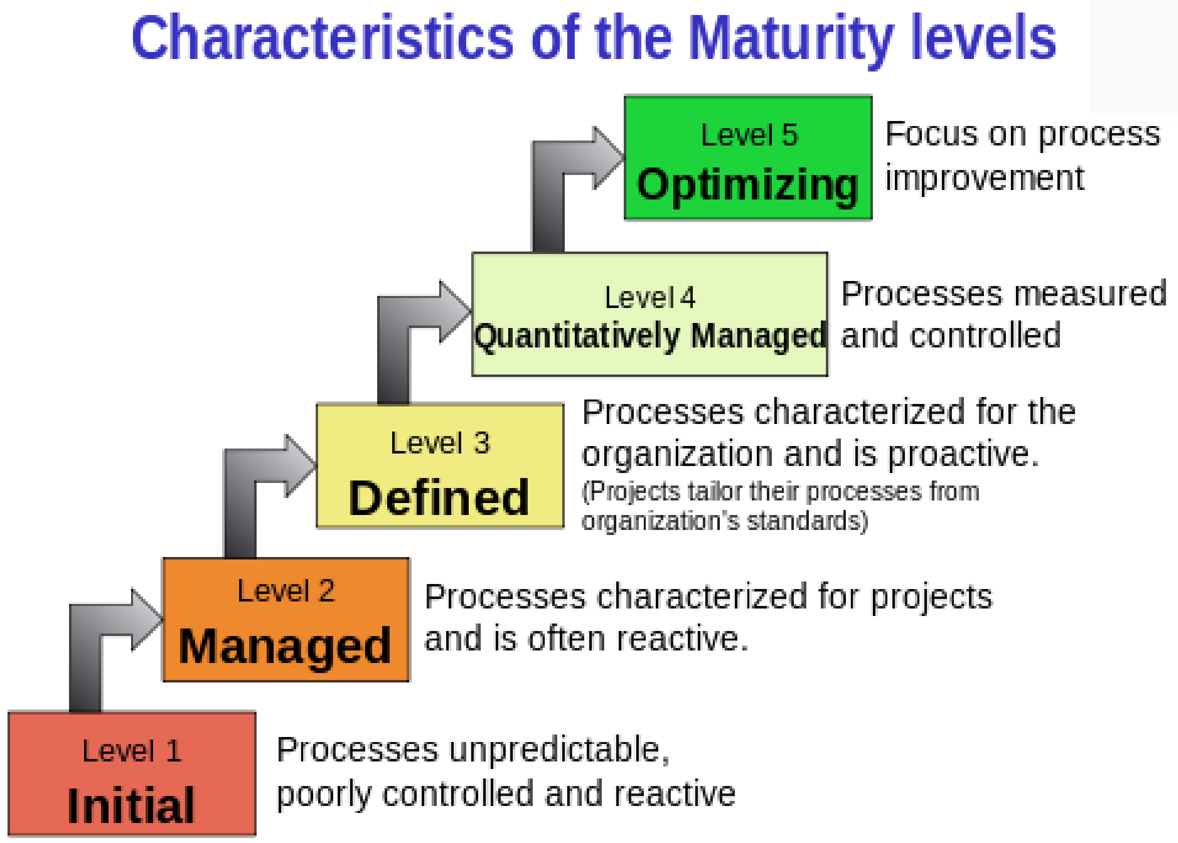
CMMI models are broken down into domains called process areas and five levels of maturity. Some process areas include increasing requirements at higher maturity, while others apply only at or above certain levels.

CISSP® MENTOR PROGRAM – SESSION ELEVEN

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Maturity Models (Capability Maturity Model Integration)


Characteristics of the Maturity levels
Level 5
Optimizing
Focus on process improvement
Level 4
Quantitatively Managed
Processes measured and controlled
Level 3
Defined
Processes characterized for the organization and is proactive.
(Projects tailor their processes from organization's standards)
Level 2
Managed
Processes characterized for projects and is often reactive.
Level 1
Initial
Processes unpredictable, poorly controlled and reactive

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Maturity Models (Capability Maturity Model Integration)

Initial - Processes are **unpredictable** and largely reactive. This is typical of an organization largely performing only ad hoc processes.

Managed - Processes are characterized or documented for projects and are often **reactive**. Certain projects may be well-managed and repeatable, but that success is not leveraged across the entire organization.

Defined - Processes are characterized for the organization and are **proactive**. An organization at level 3 has documented processes shared across various projects/efforts and takes proactive steps to ensure projects are successful, such as project risk management.

Quantitatively managed - Processes are measured and **controlled** by the measurements. This is an organization collecting and responding to metrics.

Optimizing - The organization focuses on **process improvement**. An organization at level 5 has documented processes, proactively controls them using lessons learned, and has a robust oversight function to gather metrics and manage activities.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Maturity Models (Software Assurance Maturity Model)

The *Software Assurance Maturity Model* (**SAMM**) is maintained by the Open Web Application Security Project (**OWASP**), located at https://owaspsamm.org/

SAMM is a prescriptive framework for implementing a software security program and focuses on **integrating security activities into an existing SDLC.**

It provides an action **plan for assessing the current state**, identifying a target state, and implementing necessary improvements across the SDLC including updates to processes, people, knowledge, and tools.

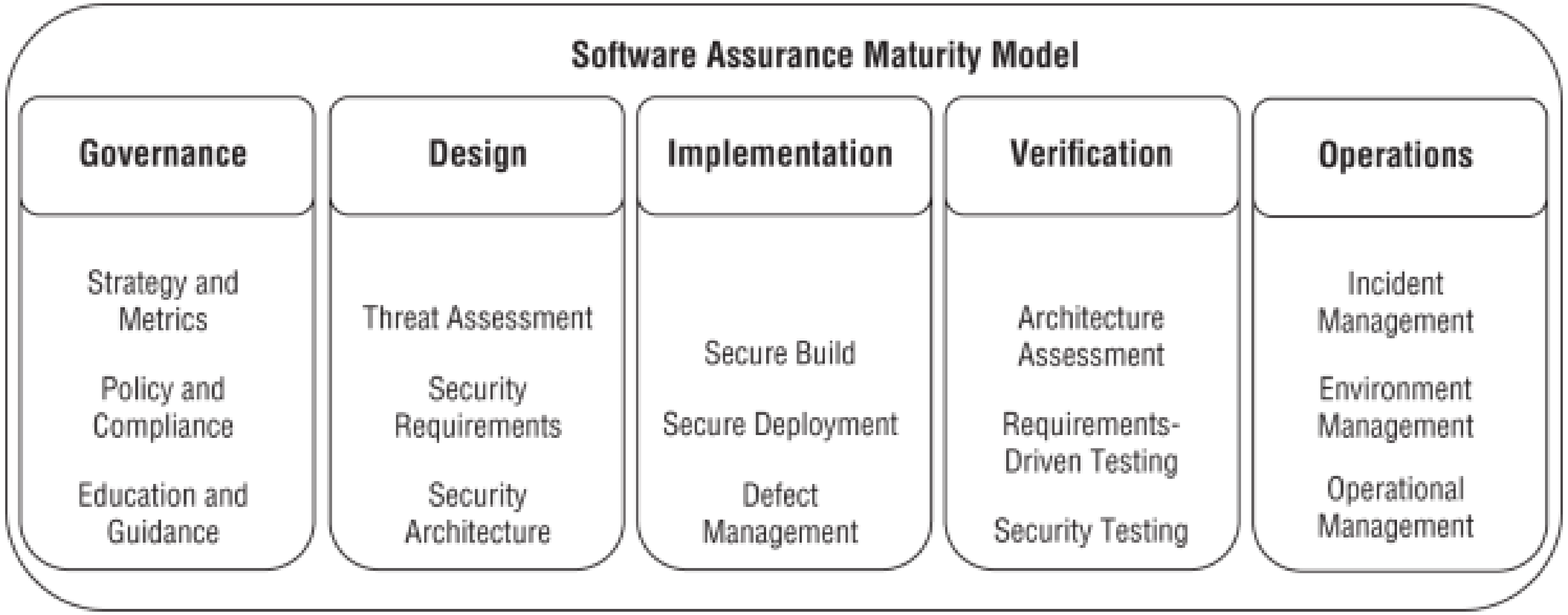
CISSP® MENTOR PROGRAM – SESSION ELEVEN

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Maturity Models (Software Assurance Maturity Model)

SAMM domains and practices



DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Maturity Models (Software Assurance Maturity Model)

Within each set of practices, there are **streams of activity**, which often represent different responsibilities.

In Security Testing Stream A, for example, automated security testing tools, which can be managed by a QA function, are required, While in Stream B, manual security testing of high-risk components is required.

This testing is a specialized function that requires application security or penetration testing experience. Each security practice is also represented in **three levels of maturity**.

For example, the Security Testing maturity levels are as follows:

1. Perform security testing (both manual and tool based) to discover security defects.

2. Make security testing during development more complete and efficient through automation complemented with regular manual security penetration tests.

3. Embed security testing as part of the development and deployment processes.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Maturity Models (Building Security-In Maturity Model)

Maturity model focused on determining the current state of secure software creation capabilities, identifying improvement opportunities, and prioritizing efforts to improve secure software development.

By determining which practices are actually effective, as determined through interviews with practicing software security professionals.

These software security initiatives (SSIs) are ordered based on their observed frequency in the interviews.

SSIs that are observed more frequently are considered to be widely adopted practices, while less-frequently observed practices may be either new concepts or performed only by organizations with a very high maturity.

https://www.synopsys.com/software-integrity/software-security-services/bsimm-maturity-model.html

CISSP® MENTOR PROGRAM – SESSION ELEVEN

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Maturity Models (Building Security-In Maturity Model)

Successful implementation of BSIMM relies upon an internal function dedicated to software security known as the software security group (SSG).

The BSIMM sets out **121** activities across four domains of practice:

Governance

Intelligence

Software Security Development Lifecycle (SSDL)

Deployment.

The SSG is responsible or accountable for implementing and maturing the organization's chosen SSIs. As with all maturity models, not every organization must be at the same level, but BSIMM is a useful source of empirically based guidance for organizations to measure and improve the software security.

CISSP® MENTOR PROGRAM – SESSION ELEVEN

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Maturity Models (Building Security-In Maturity Model)

https://www.synopsys.com/blogs/software-security/bsimm-trends-and-recommendations/

CISSP® MENTOR PROGRAM – SESSION ELEVEN

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Maturity Models

https://owaspsamm.org/blog/2020/10/29/comparing-bsimm-and-samm/

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Maturity Models (Cybersecurity Maturity Model Certification)

Published by the U.S. Department of Defense. Although broadly related to cybersecurity including but not limited to software development security, CMMC is designed for contractors who are handling sensitive information, known as controlled unclassified information (CUI), on behalf of government clients.

The maturity levels in CMMC are similar to CMMI, ranging from **level 1, Performed** (the activities associated with basic cyber hygiene are in place and functioning at the organization), up to **level 5, Optimizing** (an advanced, continuously optimized, and proactive cybersecurity program is in place).

These maturity levels assess the organization's implementation of processes, capabilities, and practices, and are organized into domains that will look familiar to practitioners experienced in the NIST SP 800-53 control families

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Maturity Models (Cybersecurity Maturity Model Certification)

- Access Control (AC)
- Asset Management (AM)
- Audit and Accountability (AU)
- Awareness and Training (AT)
- Configuration Management (CM)
- Identification and Authentication (IA)
- Incident Response (IR)
- Maintenance (MA)
- Media Protection (MP)

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Maturity Models (Cybersecurity Maturity Model Certification)

- Personnel Security (PS)
- Physical Protection (PE)
- Recover (RE)
- Risk Management (RM)
- Security Assessment (CA)
- Situational Awareness (SA)
- Systems and Communication Protection (SC)
- System and Information Integrity (SI)

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Operation and Maintenance

Operation and maintenance, often written **O&M**, is the phase when systems are in live use and running in what is known as a **production environment**, meaning they are **actively being used by end users**.

Testing, verification, and validation activities mark the end of development, and **the transition to O&M is often known as promotion or deployment**. To keep software both running and secure, there are a number of critical tasks that must be performed during this phase.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Operation and Maintenance

Continuity – Resilience of systems in the face of adverse conditions, as well as the ability to recover when an interruption does occur, are critical. Activities for business continuity, disaster recovery, and cyber resilience all occur during the O&M phase, such as generating system backups.

Monitoring and incident response – Live production systems are where a majority of security incidents occur, monitoring capability will be focused on this environment. When an incident is detected, the process of investigation and recovery typically focuses on remediating the issue and restoring production to normal.

Vulnerability and patch management – Scanning systems for flaws and remediating them is a significant, ongoing security task. Some flaws may cause an iteration of the SDLC, especially for custom software, though routine patch deployment for third-party software is the bulk of this work.

Access control – While controlling access to development and test environments is important, most systems typically have far more users than developers, and sensitive data is not usually used in testing or development. Procedures to provision, review, and deprovision access will primarily focus on operational systems.

Change and configuration management – System level change and configuration management is discussed in Chapter 7, "Security Operations," while the specifics of managing changes to source code are discussed separately in this chapter.

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Change Management

- One key aspect of managing security for many systems is to maintain them in a **known, secure state**, which is the function of configuration management. However, change is necessary, and so processes by which changes can be requested, reviewed, implemented, and tested are also necessary, which is the **goal of change management**.
- Change management is typically **not under the purview of the security department**, but security **practitioners do need to actively participate** in the process.
- Changes must be **reviewed for potential security impacts**, testing plans must be documented to ensure security is not adversely affected, and **security processes may be triggered in the event a change goes poorly.**
- Change management processes may be implemented on a **number of different levels**, with corresponding levels of rigor and sets of processes.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Change Management

- At the organization level, changes such as operating system (OS) replacements would require significant budget and resource planning.
- At a source-code level, developers may follow secure coding standards like commenting code and performing peer reviews before code is checked into a repository, which acts as the configuration and change management system for the organization's source code.
- The **code is maintained in a known good state**, and approved methods of changing it such as pull requests and commits must follow rules to ensure the integrity of the source code is not lost by unapproved developer changes.
- At some point, code may also reach a state of completion after which no changes are permitted, in a process known as a freeze.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Change Management

- Formal change management body, and common names include some variation on change "something" board, like change management, change control, or change advisory (**CMB**, **CCB**, or **CAB**).
- The composition of a CMB typically includes **permanent essential members** from IT and security teams, as well as ad hoc members like finance or HR for changes with impacts on or input required from those departments.
- The **CMB is responsible for shepherding the change management process**, which, much like the name of the group, will be unique across different organizations.
- A high-level change management process is outlined here, along with corresponding security practices recommended at each phase.

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Change Management

Request - Many change processes involve the submission of a ticket or other formal request for a change; the advantage of a ticketing system is the ability to capture all details related to the change in a single location. The request must capture full details of the change as well as anticipated resource requirements and impacts.

Analysis and approval - Once documented, the change is reviewed by the CMB to identify all anticipated requirements and impacts of the change. This follows a set of documented criteria, which should include reviewing any security impacts like introducing new risks or adverse effects on existing security controls.

Change development - Once a change is approved, the responsible parties must develop a plan to actually execute the change, such as scheduling time to perform needed IT tasks, gathering necessary resources, and purchasing or developing the changed system components. These actions must follow the documented and approved change request.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Change Management

Implementation - With a plan developed, the change can be implemented. As with the development phase, all approved change procedures must be followed. In the event a change is not successful, back-out or rollback procedures are required, and the creation of such plans and procedures is typically a requirement for change approval.

Testing - The newly changed system must be tested to ensure the changes are working as expected, existing functions were not broken, and the security controls are still operating as intended. Depending on the organization and the magnitude of the change, this testing may include highly formal activities like certification and accreditation.

Postmortem - There are a number of post-change activities typically required. Documentation of the changes performed must be completed, as the changed system is the new known state, which will be maintained by configuration management. Newly implemented features may require user training, and any lessons learned should be documented for continuous process improvement. This phase may also be called an after-action report, lessons learned, or retrospective, but the goal of identifying improvement opportunities is the same.

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Emergency Change Management

- Delay is an implicit part of change management processes, mainly because time is required to gather information and resources needed. There will be **situations when this delay is unacceptable**, such as a change needed to address a major outage or newly discovered critical vulnerability.
- Emergency changes, unlike normal changes, are implemented either without approval or with very limited approval. Rather than requiring review by the entire CMB, a **single member may grant approval**, or IT personnel may be authorized to implement the change without any prior approval.
- **Testing and verification processes are similarly streamlined.**
- Emergency changes do not imply a total lack of process or procedures, but rather a **modification designed for speed.**
- **Documentation is still required**, and many organizations perform retroactive change review and approval after the change has been deployed to ensure any unintended consequences or unforeseen impacts are understood, documented, and appropriately handled.
- After-the-fact documentation is required to support this review, and **testing should include regression tests or security assessments like vulnerability scanning and pen testing, depending on the change**

CISSP® MENTOR PROGRAM – SESSION ELEVEN

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Understand and Integrate Security in the Software Development Life Cycle (SDLC)

Integrated Product Team

Integrated product teams (**IPTs**) are **collections of multidisciplinary individuals** with a collective responsibility for delivering a product or process, similar to the combinations seen in DevOps, DevSecOps, and Agile software development.

The **goal of IPT is to either assemble or make readily available all resources with a role, skills, or knowledge relevant to a project**, and by assembling these resources, traditional delays or impediments can be removed.

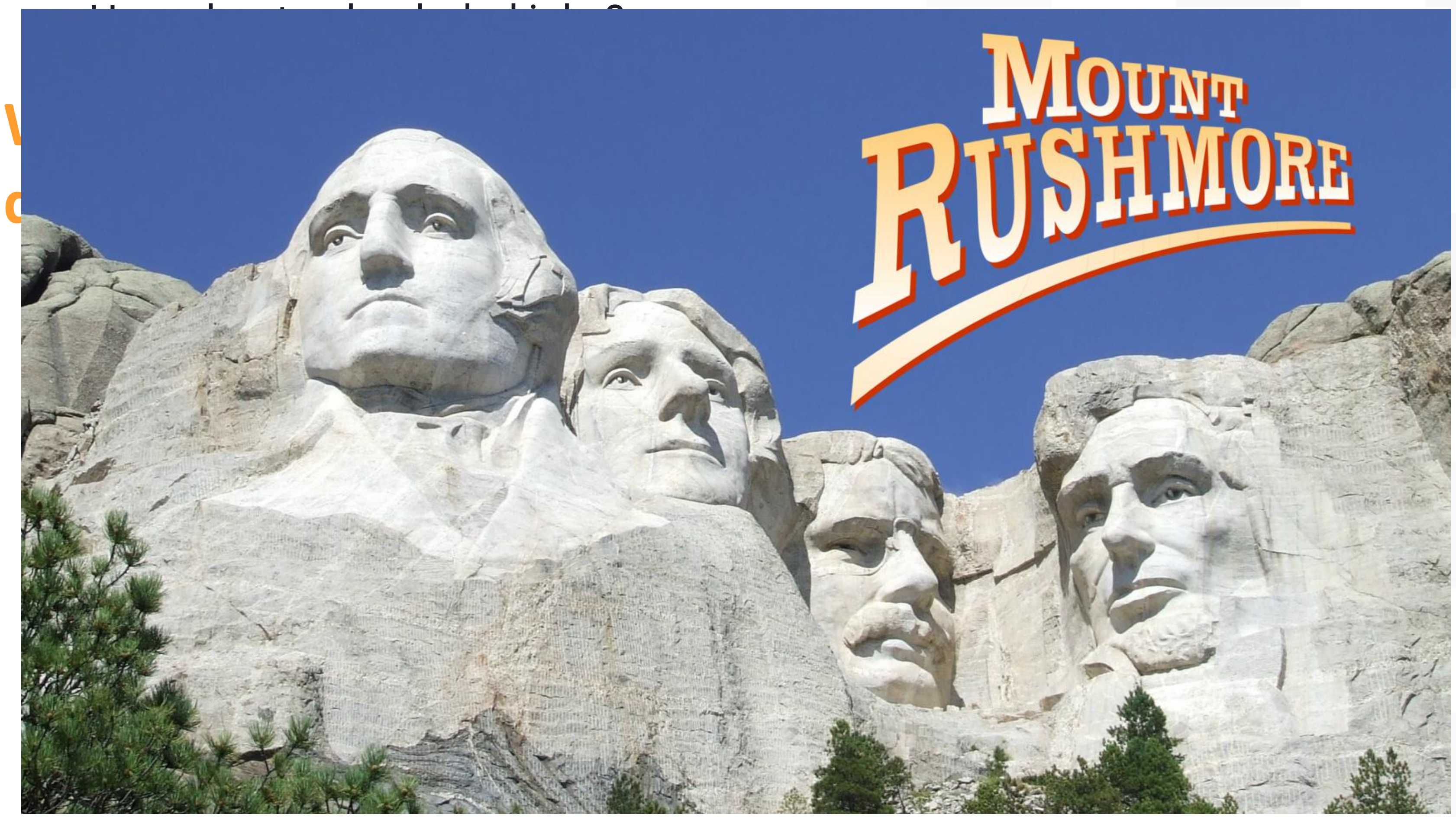
The goal of integrated product and process development (**IPPD**) is to **combine both product and process design to ensure a product, often an information system, has a holistic set of requirements** used to design both systems and business processes.

IPPD relies on extensive modeling and testing of both system and process prototypes and on the use of IPTs. These teams combine stakeholders from the user community and developers with the goal of deepening the understanding of what users need from the system and any technical constraints that need to be overcome.

CISSP® MENTOR PROGRAM – SESSION ELEVEN

DAD JOKE

Before we get too deep into this.



HAHAHAHA

Moving on…

Identify and Apply Security Controls in Software Development Ecosystems

Intro

Security practitioners need to **understand the criticality of software security** in an organization, as well as the foundational role filled by the software and tools used by developers when creating information systems.

An **overview of programming is important to understand** the various risks and associated security controls that must be addressed in software development.

Particularly **important to understand are the concepts underpinning modern programming** languages, the tools and environments in which they run, and the processes utilized by developers to write, test, and deploy code needed for modern information systems.

A number of **security tools are also designed to be integrated with and provide data** to these developers, including a host of application security testing tools that are increasingly automated and provide near-continuous feedback to improve the security of software.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Identify and Apply Security Controls in Software Development Ecosystems

Programming Languages

- Programming languages represent a set of instructions used to build programs that are executed by a computer, typically with a goal of processing information.
- These languages consist of instructions and a particular way of writing these instructions (known as syntax), as well as methods to manipulate data. These are combined to create algorithms, or specific sets of instructions, to achieve a data processing goal.
- Just as we use sentences, paragraphs, and punctuation to convey meaning in human languages, programming languages also have logical structures designed to allow computers to understand them, execute the instructions, and control the flow of data.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Identify and Apply Security Controls in Software Development Ecosystems

Programming Languages

- Compiled languages are translated into machine-readable form before being run in a process known as compiling. The developer uses a program called a complier to perform this task, which takes the written program as input and produces machine-executable code, often called a binary, as an output.
- Compiled programs can be run only by the system type for which they were compiled and are often optimized for that specific system type; for example, a program compiled to run on the Windows OS will not function if you try to run it on macOS or Linux.
- Compiled programs do not expose their source code to the end user, so if code is a valuable intellectual property asset, a compiled language is the best way to distribute it while preserving confidentiality. C# and Swift are examples of compiled languages.

Identify and Apply Security Controls in Software Development Ecosystems

Programming Languages

- Interpreted languages are translated into machine readable format when they are run, also known as on the fly. This makes interpreted code more portable across system types, as users on various platforms can install an interpreter specific to their system so the developer does not need to compile system-specific versions.

- There are downsides to interpreted languages, and security can be particularly troublesome. Code is distributed as written or in an intermediate form, which is in human-readable form, so interpreted languages may expose sensitive intellectual property.

- Due to the interpretation process, there is additional operational overhead that makes interpreted programs slower than a compiled program, though the speed of modern computing systems makes this a negligible concern in many cases. JavaScript and Python are examples of interpreted languages.

Identify and Apply Security Controls in Software Development Ecosystems

Programming Languages

- Security concerns with mobile code include its dynamic nature — since it is loaded at runtime, there is the possibility an attacker could modify the code that is requested when a user runs the application. This means previously tested code may not be loaded, but instead an attacker's malicious code is loaded and executed.

- Database systems also have languages used to define their data structures, interact with stored data, and manage the database itself. Structured query language (SQL) is one example, which includes sublanguages for interacting with a database management system (DBMS). This includes the data definition language (DDL) for creating databases and tables, as well as data manipulation language (DML) for performing actions such as querying or inserting new records. The data control language (DCL) is primarily concerned with access control for data stored in the database, which makes it critical for implementing security

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Identify and Apply Security Controls in Software Development Ecosystems

Programming Languages (Type Checking and Type Safe Languages)

Static type checking is performed by a compiler to verify if the program's functionality matches type constraints for data inputs. For example, the expression price + tax can be successfully evaluated if both input variables are integer types storing values that represent currency. However, if the program has declared an integer and a text value of "yes" or "no" to define if sales tax applies to the item, attempting to add those two pieces of data isn't possible.

Type checking can identify such programming flaws.

Dynamic type checking checks the values stored in a program's variables as it is running to ensure data matches the expected type. Languages that implement these features are known as type-safe, and they support important security goals related to integrity of data.

Strongly typed languages require all variable type declarations to be checked at compile time, while a loosely or weakly typed language allows for dynamic type checking when an application is run.

*The requirements used to determine the use of a strong or weak language may be tied to data integrity or system availability goals, since applications written a strongly typed language should not perform functions with invalid data, resulting in errors. However, code written in a strongly typed language may be less reusable for different functions, leading to additional development work required.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Identify and Apply Security Controls in Software Development Ecosystems

Programming Languages (Programing Paradigms)

- Declarative programming expresses the logic of a task without describing how it should be executed; in these languages, a system must interpret the logic and execute appropriate tasks.
- Imperative paradigm of programming uses statements or commands that change a program's state, similar to issuing imperative commands in human languages like "stand up" or "study for the CISSP." Although this kind of direct control could be a security challenge, modern imperative programming often focuses on the use of procedures, sometimes known as subroutines or functions, which provide for greater control over how the program operates.
- A standardized subroutine for accessing data in a filesystem can be tested for access control flaws; each time that subroutine is reused, there is assurance that it will implement access controls in a consistent manner.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Identify and Apply Security Controls in Software Development Ecosystems

Programming Languages (Programing Paradigms)

- Procedural programming is related to the imperative paradigm and is based on the concept of a program calling procedures, which are similar to routines or subroutines. Procedures are a set of steps to be executed and allow these steps to be logically grouped and called dynamically when needed.

- Block structured programming languages utilize bocks as an organizational unit; the scope of program elements like variables and functions is restricted to the block to prevent conflicts with other elements elsewhere in the code.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Identify and Apply Security Controls in Software Development Ecosystems

Programming Languages (Programing Paradigms)

- Object-oriented programming (OOP) is a paradigm that focuses on objects rather than actions in programming. Instead of writing specific commands to gather input data and perform procedures on it, OOP treats both data and functions as objects, known as classes, which can be linked together through defined interactions.

- An object is anything that has a state that can be tracked, like data pending a mathematical transformation to render it unreadable (the process of encryption). Code objects are self-contained modules of functions that are “called” to perform their designated function.

- There are a number of security concepts related to OOP that are important, many of which relate to the abstraction inherent in viewing data and functions as objects that communicate in predefined ways.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Identify and Apply Security Controls in Software Development Ecosystems

Programming Languages (Programing Paradigms)

Encapsulation - This is also known as data hiding and is effectively a way of isolating data from being accidentally mishandled. Direct access to a class can be denied to external objects, thereby restricting access to only approved functions known as methods.

Inheritance - Classes of data objects can have subclasses that inherit some or all of the main class's characteristics, such as access restrictions. For example, the Employee data class might have a Salary subclass, which implements the same methods for manipulating the data but additional role-based access control restrictions. This subclass can be reused as a building block, which speeds development.

Polymorphism - This term refers to the multiple (poly) forms (morphs) an object may take when being created or instantiated. Instantiating a new object from an existing object typically duplicates the attributes and methods of the existing object, but data of a different type may cause security concerns because the new data type requires different methods.

Polyinstantiation - This term means making multiple (poly) copies (instances) of an object, typically with the goal of supporting different levels of confidentiality and integrity for that object. Operating systems may create multiple instances of a shared resource space, like virtual memory, to prevent different processes from reading or writing over each other, or databases may create a copy of records for each classification level, omitting information above that classification level. This prevents users at a lower level from even knowing that more-sensitive information exists, reducing the likelihood of inference attacks.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Identify and Apply Security Controls in Software Development Ecosystems

Programming Languages (Markup and Scripting Languages)

Hypertext Markup Language (HTML) - Specifies layout or display elements of text delivered to a web browser.

Cascading Style Sheets (CSS) - Allows definition for how a web document should be styled for display, including fonts, color, and layout/spacing of elements on the page.

JavaScript - Provides interactive applications inside a web browser on a client system, often for displaying and manipulating data on the user's screen. JavaScript may require access to privileged functions on user machines such as local file access, which is a major security concern since the code comes from an untrusted party across the internet.

Python - A high-level language that can be used for a variety of uses ranging from small personal programs to large web applications. As an interpreted language, it can run on a variety of platforms and interact with data in other applications, such as a local script written by a single user to automate repetitive tasks like cleaning up data in a spreadsheet, or a distributed web app that communicates with other systems and user browsers via Application Programming Interface (API) calls.

CISSP® MENTOR PROGRAM – SESSION ELEVEN

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Identify and Apply Security Controls in Software Development Ecosystems

Libraries

- Software libraries are prewritten repositories of common functions, code, classes, scripts, procedures, or other software elements.
- They allow developers to more quickly and easily integrate functionality into programs without the need to write code from scratch.

- Many platforms like operating systems, social media networks, and cloud computing services include them in what is known as a software development kit (SDK), which speeds the task of developers building programs that integrate these platform's capabilities.
- Reuse speeds development, but it also offers increased dependability of software since these shared functions are likely to be extensively tested and improved by the community of developers and systems implementing the library.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Identify and Apply Security Controls in Software Development Ecosystems

Libraries

- Many libraries also offer compliance-focused features, such as user interface components that are designed to meet accessibility requirements, or regulated functions such as credit card processing that can be implemented without the need to address Payment Card Industry Data Security Standard (PCI DSS) compliance.
- Libraries typically implement a number of standard features, including subroutines, global variables, and class definitions. Depending on the library, additional capabilities may also be offered, such as prebuilt algorithms for common tasks like encryption, as well as data structures like lists, tables, and arrays.
- The use of libraries in the organization should be governed by a number of requirements, including the need for validated libraries that meet proven security requirements.
- A library containing vulnerable or malicious code poses a risk to the entire application relying on it.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Identify and Apply Security Controls in Software Development Ecosystems

Toolsets

Source code editors - which provide developers the ability to both write and check their code. These are similar to text editors, but include development-specific features like syntax checkers to identify incorrectly written code, library integrations that can automatically suggest appropriate functions, and even prebuilt templates of common functionality that developers can leverage to speed up their work.

Code repositories and revision control tools - provide centralized storage and integrity protection for code. Centralized storage supports multiple developers working simultaneously, while version control ensures that developers are not able to interfere with one another's work by locking code that is already open for edits and maintaining revision history with timestamps and details of the user who made changes.

Debuggers and bug databases - help developers find and fix issues with their code. Some work by highlighting known issues or identifiable flaws, such as function calls with incorrect inputs, while others can provide line-by-line execution of a program to identify where a particular error is occurring.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Identify and Apply Security Controls in Software Development Ecosystems

Toolsets

Compilers - designed to create platform-specific application binaries that consist of machine language that can be run on a specific architecture. Many programming languages can be compiled to run on a variety of platforms, allowing developers to write a single program and then use an appropriate compiler for systems they want to support.

Testing tools - typically include static code analysis and unit testing tools. Static code analysis reviews the underlying code of a program without actually running the program itself; the goal is to identify problems like improper coding that could lead to buffer overflow conditions. Data flow analysis may also be performed by these tools, where possible values that a function could generate are reviewed to ensure they meet expected parameters.

Toolsets in use at the organization should be governed by security practices like configuration management, patch management, and audits.

Identify and Apply Security Controls in Software Development Ecosystems

Integrated Development Environment

- An integrated development environment (IDE) is a combination of tools in a single environment, like a desktop program or web application, which supports the work of developers.
- Capabilities frequently incorporated in an IDE include a source code editor, debugging tools, build automation tools like compilers, testing tools, and deployment automation like integrations with container orchestration and deployment platforms including Docker and Kubernetes.
- As with toolsets, the security of the IDE should be a focus of the security program as well.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Identify and Apply Security Controls in Software Development Ecosystems

Runtime

- Runtime is the collection of all hardware and software required to actually run an application.
- For many systems, this will be quite complex, as hardware may contain its own software (known as firmware) for proper operation, device drivers may be required to allow apps to communicate with the hardware, and an OS will need to coordinate actions and mediate access to the hardware.
- For high-security applications, secure hardware such as a trusted computing base (TCB) may be required. A TCB comprises all hardware, software, and firmware responsible for enforcing security and serves the function of a reference monitor to control access and enforce security policies within the system.

*The traditional model of a runtime many people visualize consists of a desktop computer (or laptop) running a standard OS like macOS or Windows. Trends like Internet of Things (IoT) and edge computing now mean that the collection of hardware and software needed to perform computing functions is much broader.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Identify and Apply Security Controls in Software Development Ecosystems

Continuous Integration and Continuous Delivery (CI/CD)

- Continuous integration (CI) is the practice of continuously merging developer's code to a shared main location, often known as a repository.

- Integrating code more frequently reduces the cost and time required to do large-scale reconciliation and integration activities like debugging any issues caused by the newly written or modified code.

- A code repository is an implementation of configuration management that enforces rules on how items enter and leave the controlled state.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Identify and Apply Security Controls in Software Development Ecosystems

Continuous Integration and Continuous Delivery (CI/CD)

- Automated security testing can also be combined with these integration activities.
- Continuous delivery (CD) is a process that delivers code automatically to an environment other than live production or requires some manual steps to deploy to a production environment.
- This is often used in software development where a testing environment with manual testing processes is required, such as systems processing regulated or sensitive data, or complex systems where automation may not be possible.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Identify and Apply Security Controls in Software Development Ecosystems

Continuous Integration and Continuous Delivery (CI/CD)

- A closely-related and somewhat confusing term is continuous deployment (also referred to with the acronym CD), which extends the concept of CI by taking the integrated code and deploying it to the production environment automatically with no manual intervention needed — unless an automated step fails.
- Achieving continuous deployment demands a high degree of maturity in an organization's automated QA and testing processes, to ensure that code is thoroughly tested and passes before being deployed.
- Automated functional, integration, and regression testing are often required to ensure the newly developed code does not break the system or introduces bugs.

Identify and Apply Security Controls in Software Development Ecosystems

Continuous Integration and Continuous Delivery (CI/CD)

- The combination of tools and processes to implement CI/CD is often known as a pipeline, and this pipeline includes all the steps and requirements to deliver functional software to the production environment. This will include manual steps like writing code and addressing any errors that arise, as well as automated steps like running tests and scans. Packaging the code, such as building containers, and deploying it are the final stages of the pipeline.

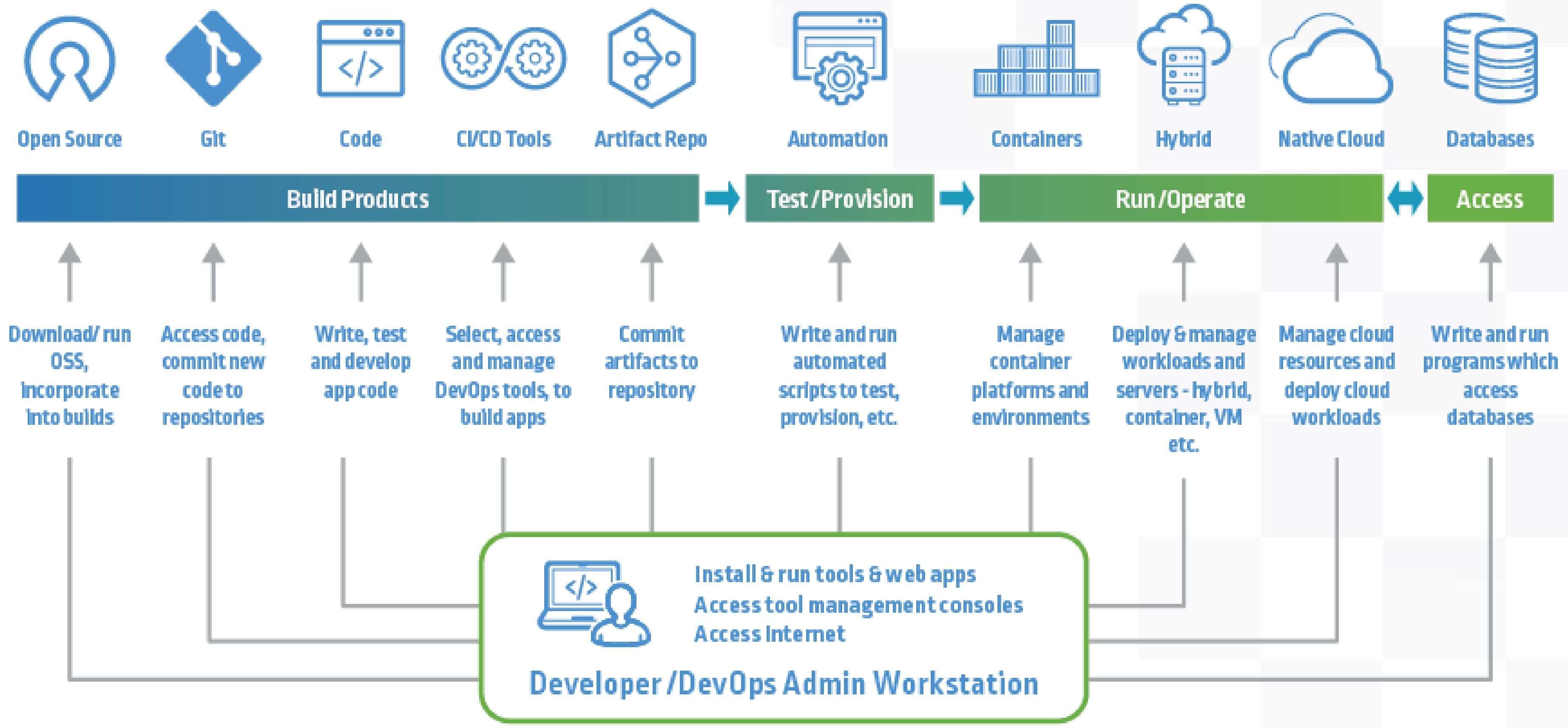
- Each system and step in the pipeline is a chance to implement security controls like integrity verification, encryption to protect confidentiality, and manual or automated testing and reviews.

CISSP® MENTOR PROGRAM – SESSION ELEVEN

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Identify and Apply Security Controls in Software Development Ecosystems

Continuous Integration and Continuous Delivery (CI/CD)


Open Source
Git
Code
CI/CD Tools
Artifact Repo
Automation
Containers
Hybrid
Native Cloud
Databases
Build Products
Test/Provision
Run/Operate
Access
Download/run OSS, incorporate into builds
Access code, commit new code to repositories
Write, test and develop app code
Select, access and manage DevOps tools, to build apps
Commit artifacts to repository
Write and run automated scripts to test, provision, etc.
Manage container platforms and environments
Deploy & manage workloads and servers - hybrid, container, VM etc.
Manage cloud resources and deploy cloud workloads
Write and run programs which access databases
Install & run tools & web apps
Access tool management consoles
Access Internet
Developer/DevOps Admin Workstation

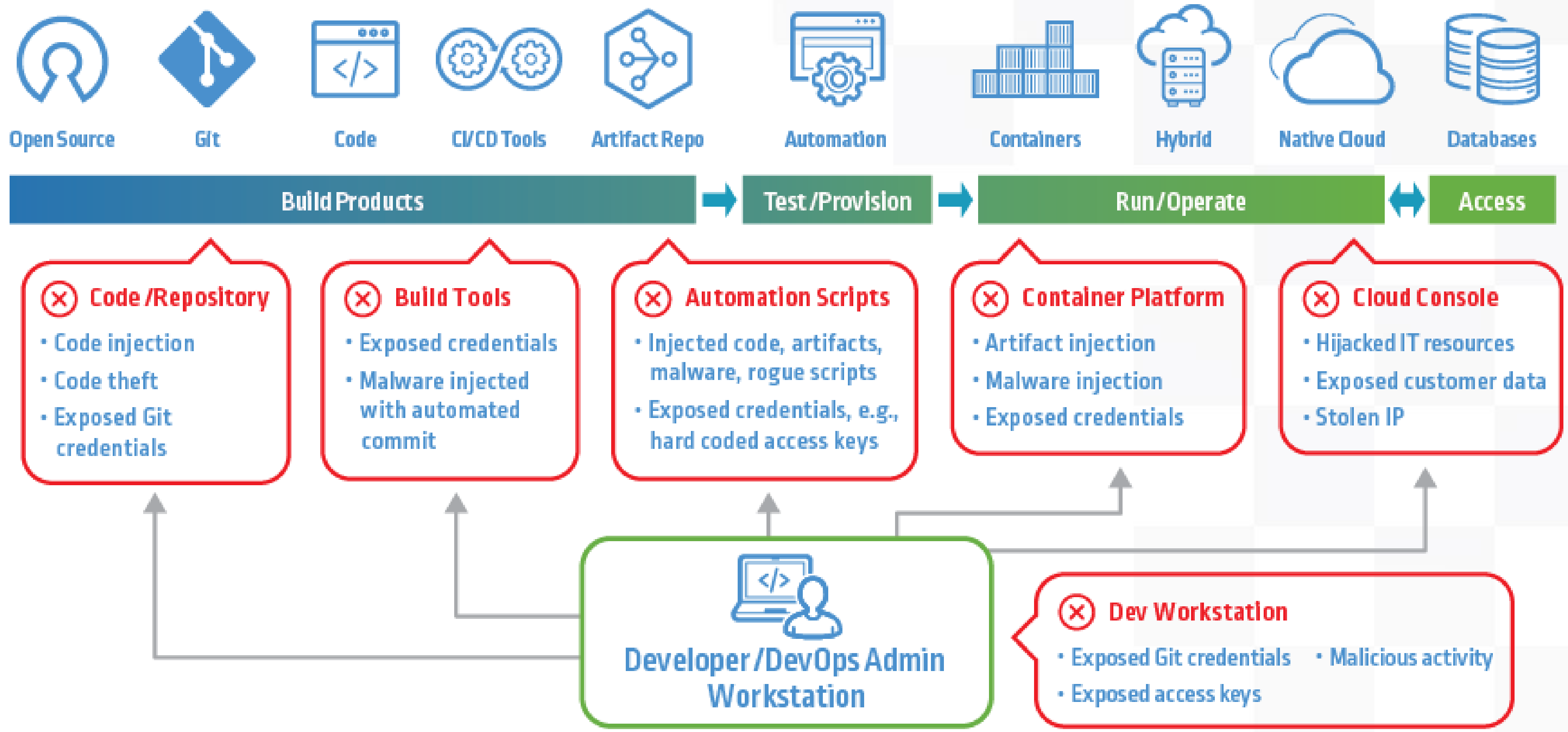
https://www.cyberark.com/what-is/ci-cd-pipeline/

CISSP® MENTOR PROGRAM – SESSION ELEVEN

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Identify and Apply Security Controls in Software Development Ecosystems

Continuous Integration and Continuous Delivery (CI/CD)



https://www.cyberark.com/what-is/ci-cd-pipeline/

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Identify and Apply Security Controls in Software Development Ecosystems

Emphasis on Testing in CI/CD Pipelines

- CI/CD relies heavily on automation to achieve speed. Rather than manually building the computer hardware, OSs, and enabling applications like web servers.
- CI/CD pipeline will likely take advantage of virtualized or cloud computing where these activities can be automated and orchestrated using definition files. These definition files specify the required computing resources that need to be provisioned when an application is deployed, and this practice is called infrastructure as code (IaC).
- Configuration management, testing, and auditing are much easier since definition files exist.
- A CISSP can achieve a great deal of security impact by implementing small, automated, and repeatable security checks into this automated pipeline.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Identify and Apply Security Controls in Software Development Ecosystems

Emphasis on Testing in CI/CD Pipelines

- Application security (AppSec) is another important automated testing function supporting CI/CD pipelines.
- Unit, functional, and regression testing should all be included in automated testing activities, as well as security-specific testing like vulnerability scanning.
- DevOps and security practitioners can identify ways to design an AppSec program using Agile principles, with the goal of integrating security within a CI/CD pipeline to ensure security requirements are met without impeding the agility of the organization.
- Automated testing tools can shift the balance of work away from the security team and break down an important barrier that often leads to issues.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Identify and Apply Security Controls in Software Development Ecosystems

Security Orchestration, Automation, and Response (SOAR)

- Primary goal is to speed the time to detect and respond to security incidents, and it achieves this by integrating disparate data sources and systems to coordinate automated actions.

- SOAR is often confused with security information and event management (SIEM), but SIEM tools simply ingest data to support response activities by correlating data and generating alerts for human responders.

- The difference is similar to intrusion detection versus intrusion prevention: intrusion detection systems (IDSs) generate an alert, while intrusion prevention systems (IPSs) take action automatically to stop the intrusion.

CISSP® MENTOR PROGRAM – SESSION ELEVEN

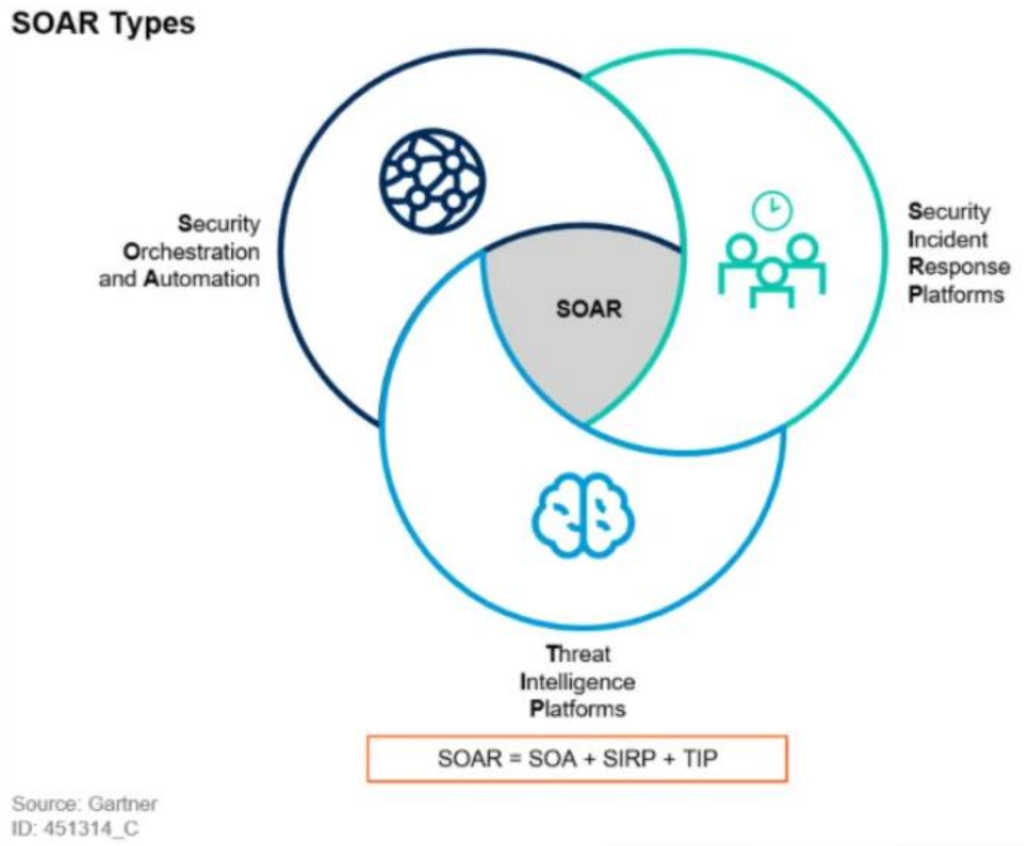
DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Identify and Apply Security Controls in Software Development Ecosystems

Security Orchestration, Automation, and Response (SOAR)

Gartner, Solution Path for Security in the Public Cloud, Figure 13, January 2020 Gartner, Solution Path for Security in the Public Cloud, Figure 13, January 2020

https://insights.sei.cmu.edu/blog/benefits-and-challenges-of-soar-platforms/


SOAR Types
Security Orchestration and Automation
Security Incident Response Platforms
SOAR
Threat Intelligence Platforms
SOAR = SOA + SIRP + TIP
Source: Gartner
ID: 451314_C

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Identify and Apply Security Controls in Software Development Ecosystems

Security Orchestration, Automation, and Response (SOAR)

Orchestration - This is where SIEM and SOAR have some overlap, in that they both integrate data from disparate systems into a single data set. However, SOAR goes beyond logs and includes data from security tools like user and entity behavior analytics (UEBA), IDS/IPS, external threat intelligence services, and domain-specific solutions like email anti-phishing tools, anti-malware, application security scanners, etc.

Alerts on suspicious activity from a SIEM will be an input to a SOAR platform, relying on the SIEM's ability to correlate data across networks or systems. The use of APIs is critical for data ingestion as well as exposing functionality on the target systems to enable automation of responses.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Identify and Apply Security Controls in Software Development Ecosystems

Security Orchestration, Automation, and Response (SOAR)

Automation - Automation follows a set of machine-executable rules, often collected in a playbook or other set of instructions designed to respond to specific triggers like malware detection.

The advantage of SOAR over legacy methods is the speed with which it executes. Speed is critical since this is a countermeasure activated after a risk event has occurred, and the goal is to reduce the duration and impact of the incident.

Automating actions like forcing a password change after a user's workstation has been impacted by malware, isolating the affected machine from the network, quarantining or deleting the email from other users' inboxes, and opening an incident response case with relevant details prepopulated saves time and reduces the window of opportunity for the attack to do damage.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Identify and Apply Security Controls in Software Development Ecosystems

Security Orchestration, Automation, and Response (SOAR)

Response - SOAR evolves incident response from the legacy model of detecting and manually responding to a coordinated, automated response in which human expertise, knowledge, and talent are utilized in the most efficient manner possible.

Information systems can receive and act on data faster than human analysts; even a fully staffed, 24/7 security operations center (SOC) will exhibit slower response times compared to an automated system with visibility into and automation abilities across the infrastructure.

SOAR doesn't seek to completely remove human intervention in incident response. Many of the platforms include incident response case-management tools, with the aim of consolidating information needed for decision making and providing automated chains of actions that can initiated by the responder.

CISSP® MENTOR PROGRAM – SESSION ELEVEN

DAD JOKE

Before we get too

How about a dumb dad

Every time I take m
the ducks try to bit

HAHAHAHA

Moving on…

Identify and Apply Security Controls in Software Development Ecosystems

Software Configuration Management (SCM)

- **Software configuration management (SCM)** is the implementation of configuration and change management practices to support secure software development. Its main purpose is to provide integrity for key software resources like source code and program resources like requirements documentation.
- These items placed under configuration management are known as **configuration items (CIs)**. Each CI must be uniquely identified, and changes to CIs must follow a rigorous change process to ensure the changes are coordinated and do not introduce flaws or break functionality.
- **Baselines** represent a stable set of **immutable configuration items** that collectively provide a functional system — note that this term is also used in the context of security controls, where a baseline represents the minimum set of required controls.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Identify and Apply Security Controls in Software Development Ecosystems

Software Configuration Management (SCM)

- SCM's chief goal is to **promote visibility and control** over the state of and changes to all the elements in a software system.
- Changes can introduce bugs or flaws that negatively impact fundamental security concepts of **confidentiality, integrity, availability, nonrepudiation, and authenticity (CIANA),** and it is imperative to preserve the integrity of system elements like source code, requirements documentation, and settings.
- Uncontrolled changes can introduce vulnerabilities or have other unintended consequences.

Identify and Apply Security Controls in Software Development Ecosystems

Code Repositories

- Repositories, or repos as they are commonly known, are a cornerstone of modern software development and provide key features needed to support multi-user teams developing software.
- They provide a centralized storage location and support collaborative work by allowing multiple users to access and make changes, as well as implementing restrictions such as branching, which allow a user to pull code and make nondestructive changes isolated from the main code.
- When complete, the changes can be tested and merged back into the main code.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Identify and Apply Security Controls in Software Development Ecosystems

Code Repositories (Protecting Source Code)

Confidentiality – Repos provide access controls in the form of authorization. Features to protect confidentiality may include restrictions preventing copying of data to local computers or portable media devices.

Integrity – Most code repositories implement some form of configuration management and version control for software. Version control systems that allow rollback to a previous version can also be used as a countermeasure to unintended or undesirable changes.

Availability – The centralized nature of a repo can be a single point of failure. Having a single system hosting all data also makes it easier to back up and restore the system, as a robust backup schedule for that system guarantees that all the required data is available when needed.

Nonrepudiation – Access control mechanisms implemented in repositories — such as unique user IDs, audit trails, and version history — can be used to identify and hold users accountable for changes they made to source code.

Authenticity - Data is authentic if it can be proven that no corruption or unwanted alteration has occurred, and source code repositories can implement this using version control to roll back unwanted changes and using backups to support integrity.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Identify and Apply Security Controls in Software Development Ecosystems

Code Repositories (Protecting the Repository)

Data – Tools like data loss prevention (DLP) and intrusion detection should be deployed to monitor data and activity on repository systems and hosts.

Communication and network – Proper security measures to control and monitor access to the repository must be in place. Due to the remotely accessible nature and increased trend of distributed systems, it is particularly important to secure remote network access by implementing protocols like HTTPS and Transport Layer Security (TLS), or secure access solutions like a VPN or proxy service.

Access control – Principles of minimum necessary and least privilege must be considered, and access control procedures such as request approvals and routine reviews should also be implemented.

Backup and availability – In all cases, the configurations should support the organization's availability goals by ensuring software code is accessible when needed.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Identify and Apply Security Controls in Software Development Ecosystems

Application Security Testing

Static application security testing (SAST) - SAST evaluates source code and other nonrunning application elements like compiled binaries. Testing is performed in development or testing environments, which means there is no impact to live production environments.

SAST disadvantages - SAST tools can typically understand only one programming language, so different tools will be required if multiple languages are in use. Difficult for non developers to access due to their IDE integration, meaning security practitioners who aren't developers are excluded. Complex vulnerabilities due to interfaces between code and underlying system elements like OS exploits cannot be detected.

Dynamic application security testing (DAST) - They can typically run against any application with an interface or API regardless of underlying language and are not as tightly integrated with an IDE, which means access is easier for non developers. automated DAST scan is initiated when new code is pushed to a testing environment, and any findings are automatically pushed into developer tickets or task assignments.

DAST disadvantages - Due to their possible existence outside the developer's toolset, there may be organizational challenges related to getting information to the correct developer. Some DAST tools are designed for continuous monitoring of production environments as a detective measure; this may cause performance issues and, more importantly, allows software with flaws to enter production environments where it could be exploited until discovered.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Identify and Apply Security Controls in Software Development Ecosystems

Application Security Testing

Interactive application security testing (IAST) - Combines elements of SAST, DAST, and penetration testing, often using complex algorithms and machine learning to analyze source code and correlate vulnerabilities discovered during dynamic testing.

IAST disadvantages - Since IAST incorporates source-code analysis like SAST, it is also language-dependent, meaning some languages may not be supported and the dynamic elements of IAST may cause performance issues if run in a production environment.

Runtime application self-protection (RASP) -Executes alongside the application as it is run; RASP is also less of a testing tool but is often incorporated into overall application security to complement SAST and DAST. A RASP security tool integrates with an application and analyzes the program's execution to spot unusual or unexpected behavior, and then it takes corrective action.

RASP disadvantages - Like any automated reactive technology, false positive RASP hits can lead to DOS, much like an IPS shutting down unexpected but legitimate traffic. Similar to IAST, as a relatively recent technology, RASP will suffer from compatibility and a gap in skills to configure and manage the tools.

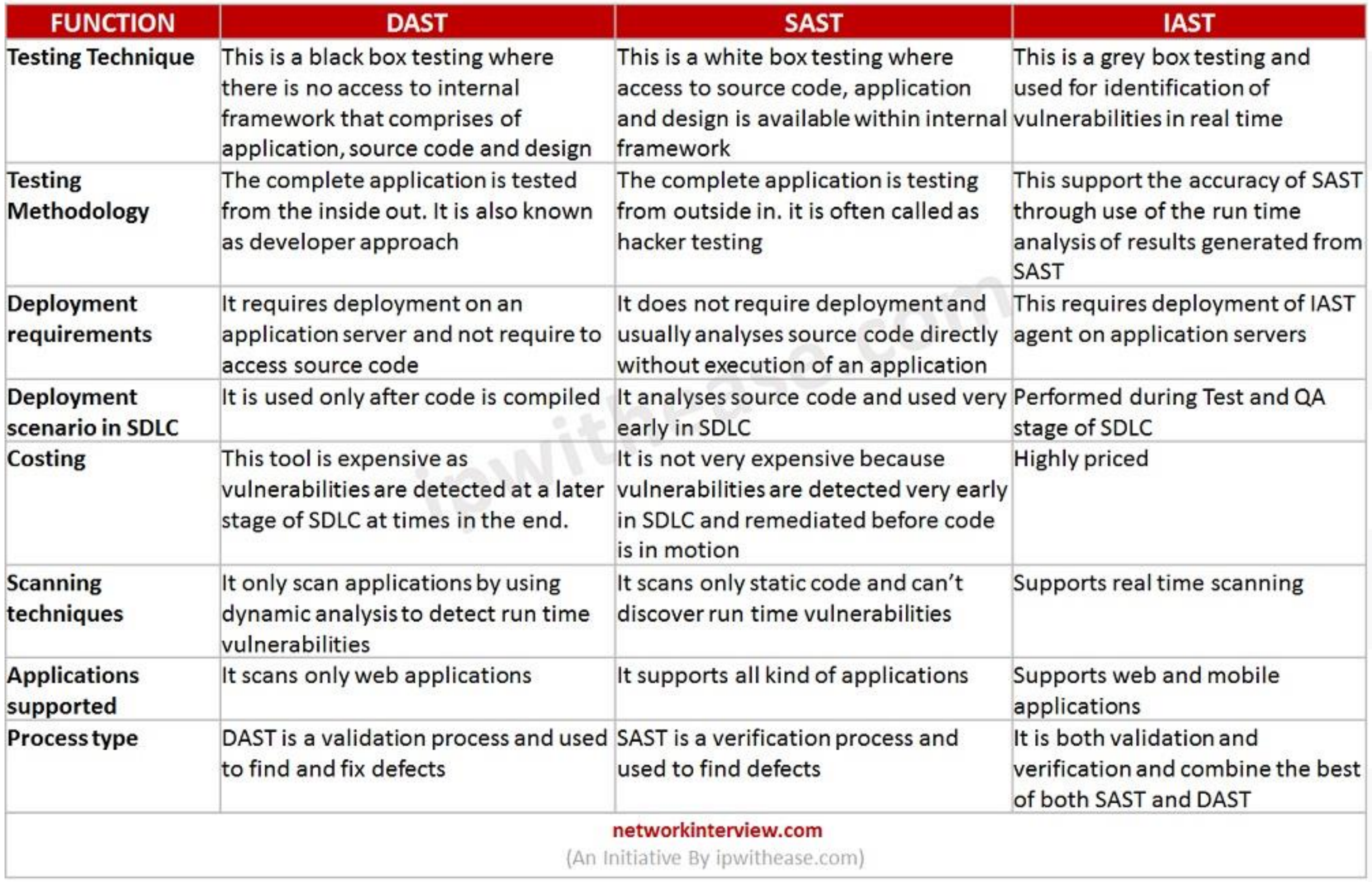
CISSP® MENTOR PROGRAM – SESSION ELEVEN

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Identify and Apply Security Controls in Software Development Ecosystems

Application Security Testing

https://networkinterview.com/dast-sast-iast-security-testing/



FUNCTION	DAST	SAST	IAST
Testing Technique	This is a black box testing where there is no access to internal framework that comprises of application, source code and design	This is a white box testing where access to source code, application and design is available within internal framework	This is a grey box testing and used for identification of vulnerabilities in real time
Testing Methodology	The complete application is tested from the inside out. It is also known as developer approach	The complete application is testing from outside in. it is often called as hacker testing	This support the accuracy of SAST through use of the run time analysis of results generated from SAST
Deployment requirements	It requires deployment on an application server and not require to access source code	It does not require deployment and usually analyses source code directly without execution of an application	This requires deployment of IAST agent on application servers
Deployment scenario in SDLC	It is used only after code is compiled	It analyses source code and used very early in SDLC	Performed during Test and QA stage of SDLC
Costing	This tool is expensive as vulnerabilities are detected at a later stage of SDLC at times in the end.	It is not very expensive because vulnerabilities are detected very early in SDLC and remediated before code is in motion	Highly priced
Scanning techniques	It only scan applications by using dynamic analysis to detect run time vulnerabilities	It scans only static code and can't discover run time vulnerabilities	Supports real time scanning
Applications supported	It scans only web applications	It supports all kind of applications	Supports web and mobile applications
Process type	DAST is a validation process and used to find and fix defects	SAST is a verification process and used to find defects	It is both validation and verification and combine the best of both SAST and DAST

networkinterview.com
(An Initiative By ipwithease.com)

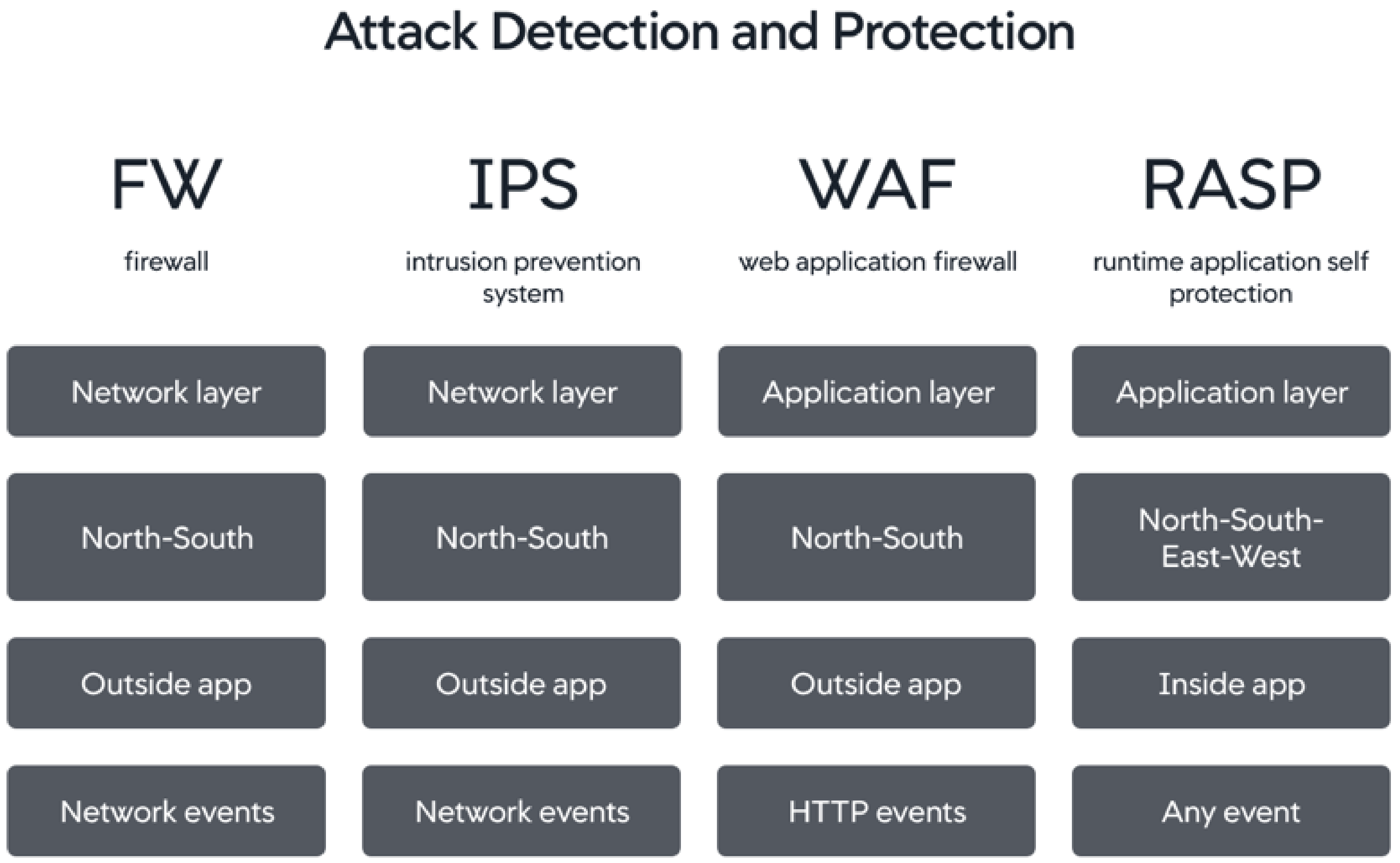
CISSP® MENTOR PROGRAM – SESSION ELEVEN

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Identify and Apply Security Controls in Software Development Ecosystems

Application Security Testing

https://www.appdynamics.com/learn/how-rasp-protects-apps



Attack Detection and Protection

FW	IPS	WAF	RASP
firewall	intrusion prevention system	web application firewall	runtime application self protection
Network layer	Network layer	Application layer	Application layer
North-South	North-South	North-South	North-South-East-West
Outside app	Outside app	Outside app	Inside app
Network events	Network events	HTTP events	Any event

CISSP® MENTOR PROGRAM – SESSION ELEVEN

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Assess the Effectiveness of Software Security

Intro

Software security must be measured to gauge its effectiveness and drive improvement over time.

Security Assessment and Testing." Logging and auditing changes to software environments, not just production but also development environments, provides critical data needed to verify the integrity of software powering critical information systems.

While a common practice across all domains of information security, requires specific practices for software security to ensure adequate treatment of software's unique risks.

These include risks to applications and information systems, as well as to software development tools and the source code itself, both of which are valuable assets.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Assess the Effectiveness of Software Security

Audit and Logging of Changes

A standard Windows 10 workstation used during a regular workday can easily generate tens of thousands of logged events; reviewing each of these manually is a virtually impossible task.

Most events change the state of the system in some way; for example, a server restarting means the system is not available for a period of time.

This could be caused by an unexpected power failure or by a new deployment of code that adds user-demanded functionality.

Keeping track of these changes and providing visibility into which are normal and which are not is crucial for diagnosing issues and for remediating events that rise to the level of security incidents.

Assess the Effectiveness of Software Security

Audit and Logging of Changes

Auditing and logging are different, though often confused, terms. They are often used interchangeably, such as in audit logs, even though each term actually has its own definition.

Assess the Effectiveness of Software Security

Audit and Logging of Changes

Auditing – **Official or systematic inspection** of something such as an account, documents, or practices. In software environments, audits focus on the implementation and **effectiveness of software security controls**, and they often rely on an official standard of comparison such as the ISO, NIST, or OWASP security frameworks.

Logs themselves may be subject to audit to identify instances of behavior that go against organization requirements, such as users attempting to access unauthorized resources.

Logging – merely the **recording of events**. In a software system, logs are generated by all system events, like users signing into an application, and the logs should capture sufficient detail to definitely identify critical information about the event. This includes what the event was, who/what initiated the action, when it occurred (often called a timestamp), and other metadata about the event like a criticality level. Some events will be categorized as incidents if they violate security controls or cause a degradation of the system.

Assess the Effectiveness of Software Security

Audit and Logging of Changes

Accountability – ability to identify who or what is responsible for events that occur and is a key element of identity and access management. Sufficient data is obviously required; if log files do not record usernames or timestamps, it will be impossible to identify which user took a particular action at any given time.

Nonrepudiation - indisputable proof of a unique individual having performed a specific action.

- This concept is often confusing because it is a negative — repudiating an action means refusing to accept responsibility.
- Nonrepudiation, therefore, means having sufficient proof to prevent a user from denying that they took the action.
- Not all log data will support nonrepudiation; for example, physical access logs collected at a main door are not sufficient to prove a particular employee was in a specific location within the building, unless access controls create log data as the employee moves about the facility.

Assess the Effectiveness of Software Security

Applications of Logging and Auditing

- Log generation and routine audits are both **reactive** and **detective** controls.
- Logs are generated after an action has occurred, so they do not work to prevent a risk being realized, though they support monitoring activities that detect when something has gone wrong.
- **Logs** are **only** made **useful** when audit and monitoring processes **make use of them**; much like policies, they are useless if documented but never read or used.

- **SIEM** tools ingest log data as a primary input and then perform monitoring activities on the collected data to look for suspicious or unwanted behavior.
- SIEMs typically perform this monitoring on a continuous and near real-time basis, and alerts generated are key inputs to the **detection phase of incident response** procedures.

Assess the Effectiveness of Software Security

Applications of Logging and Auditing

- Audits are a more formal and structured process of reviewing activity against an expected baseline, often as a way to demonstrate compliance with a legal, regulatory, or other compliance burden.
- Assessments are a more informal process of review and are often conducted to satisfy internal oversight goals rather than external compliance requirements.
- Log data is crucial for audits, as it presents a record of system activity over a period of time, which confirms if an organization is functioning in accordance with documented governance.

Assess the Effectiveness of Software Security

Applications of Logging and Auditing

- *Source code management (SCM)* is a particularly important application of logging and auditing for software security.
- Logs can be useful for operational troubleshooting if a particular piece of code is causing errors and can also be used to audit compliance by identifying code changes that might diverge from identified requirements.

- Logging in operational environments provides insight into the activities being performed by users and processes while a system is in active use.
- Logs are often used for access reviews or audits, as well as investigations of both operational and security incidents.

Assess the Effectiveness of Software Security

Applications of Logging and Auditing

Change management - Logs of system changes like new app installations or changes to access permissions can be used to verify whether changes made correspond to approved change requests.

Security incidents - Live production environments, especially those with public- or external-facing elements like web servers, will see a variety of suspicious and malicious activity. Logging and monitoring these actions are crucial tasks for security practitioners and often form a key pillar of security analysts' and incident responders' job functions.

Performance management - Although not an obvious security task, availability is a component of security. Ensuring systems are running, reachable, and able to support the required user load can be achieved by monitoring various metrics from log files, such as CPU utilization, memory consumption, and network bandwidth usage.

Cloud environment changes - When consuming cloud services, most organizations will lose physical and some logical visibility over parts of their IT infrastructure. Rather than deploying network sensors to detect problems, web pages or APIs with status information will provide vital logs and monitoring data, such as when a service began to experience degradation or an outage and when it is restored.

Assess the Effectiveness of Software Security

Applications of Logging and Auditing

- There are **definitive resources** for **logging** and **monitoring** practices as well as **audit** standards.

- **NIST SP 800-92**, "Guide to Computer Security Log Management," and the **OWASP Logging Cheat Sheet** cover basics like policy requirements and best practices for implementing logging functions.

- Many **compliance frameworks mandate** certain types of logs or require that logs generally contain sufficient data to support accountability.
- These compliance frameworks, such as **PCI DSS**, also provide **auditing standards** for assessing the implementation and operating effectiveness of security controls including logging.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Assess the Effectiveness of Software Security

Software Development Auditing

Requirements phase – Audit artifacts from this phase will mostly be documentation such as requirements analysis documents and the documented requirements themselves. Key activities to audit include reviews like **privacy impact assessment (PIA)** and data classification or system categorization. Ensuring complete requirements and documented understanding is critical to designing and building a system with adequate protection mechanisms for data.

Design phase – During design, alternatives will be evaluated against requirements, such as cloud versus on-premises hosting for a system. Key activities to be audited include the formal risk assessment and decision processes utilized to determine which of the proposed solution alternatives adequately meet the stated requirements, and formal acceptance of the residual risk.

Assess the Effectiveness of Software Security

Software Development Auditing

Implementation phase – Hands-on activities like writing code, testing system functionality, and security tests like code reviews or penetration tests should all be in scope for the audit program. Testing is also an important audit method used to demonstrate that processes and technical controls respond as expected, like trying actions known to violate separation of duties policies and ideally observing the actions being blocked.

Verification phase – Audits of testing activity are important to verify oversight functions are implemented and effective. These should include checks for adequately designed test cases or data, coverage of testing, execution of all required testing, and successfully passing required tests before system deployment.

Operation and maintenance phase - bulk of auditing occurs here. Systems in this phase may also be subject to more external audits, as many laws and regulations require auditing on live production systems. Many audit frameworks now incorporate live system audits as well as SDLC auditing, due to the key role that secure software development plays in overall system security.

Assess the Effectiveness of Software Security

Risk Analysis and Mitigation

- Software development and security requires ongoing assessment and mitigation of risks, so these concepts must be applied across all SDLC phases and software environments including development and end user or live production.
- Goals are still the same, chiefly identifying assets to protect, likelihood and impact of various vulnerabilities and related threats, and the most cost effective ways to manage the risks.
- Software risks require unique analysis approaches, and the first step is determining targets for analysis. Aspects of software development environments and practices that should be considered when performing this risk assessment include the following

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Assess the Effectiveness of Software Security

Risk Analysis and Mitigation

Data risks - Information systems store, transmit, and process information that requires protection, so the systems themselves are often at risk due to their role in handling data.

Security practitioners should prioritize based on classification levels; systems handling the most sensitive or highly regulated data are likely to cause the most impact in case of a breach.

That doesn't mean lower-sensitivity systems can be ignored, however, and the possibility of an adversary compromising a lower-classified system and pivoting to a higher one must also be considered.

New technology risk - Information systems running standardized commercial software with well-known configuration and support needs (and with proper support and maintenance) are usually less risky than emerging technologies.

A lack of knowledge and expertise increases the likelihood of a misconfiguration when deploying systems based on new or emerging technology, and their novelty can mean security options are not yet available.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Assess the Effectiveness of Software Security

Risk Analysis and Mitigation

System changes - Organizational needs evolve, new technologies become available, and old technologies cease to be supported. System changes can introduce new risks and significantly alter existing risks, though in some cases, changes may reduce or eliminate risks as well.

Code risks - Software is code, and unintended or malicious changes introduce serious risk to all elements of security. Improperly handled changes can introduce bugs or flaws rendering a system unavailable, while the integrity and authenticity of data processed by a system can be compromised by unapproved code changes.

SDLC activities like code reviews, change management, testing, and vulnerability management should all be designed to detect and correct these risks. Code is also a source of confidentiality risk, as improperly programmed and configured systems will not implement the desired levels of data confidentiality.

Assess the Effectiveness of Software Security

Risk Assessment and Treatment

Planning - The organization's approach to system development can introduce operational risks like schedule or budget concerns. If security is broadly planned as one day of testing before a system launches, it is unlikely the security effort will be effective, or findings will be adequately remediated.

Requirements - New system features, like exposing systems to access from the internet or adding new features like credit card processing, introduce risks such as new attack vectors or regulatory compliance burdens.

Design - The chosen design for a system may increase or decrease various risks like remote access, lack of features to support needed security controls, or cloud vendor lock-in.

Development - While being developed, all the security controls discussed in this chapter must be addressed to mitigate software development risks. This includes appropriate use of developer toolsets, proper use of secure coding guidelines and standards, and testing to ensure the software being developed meets both functional and security requirements.

Operations and maintenance - The vast majority of risks against a system impact it during live use by end users. The routine assessment of risk and a continuous monitoring program will be crucial to measuring these risks and the effectiveness of chosen mitigations.

Assess the Effectiveness of Software Security

Mitigations Strategies

When discussing security control selection for software development, it is important to note that implementation of these controls must be integrated into existing SDLC processes for the systems they are designed to protect.

Some costs to be measured when performing the analysis include direct costs like software or product purchases, professional services support, and operating costs like support contracts. Indirect costs may be more difficult to define or measure discretely due to the commingling of many factors in a development project.

Assess the Effectiveness of Software Security

Mitigations Strategies

- Measuring the benefits of security controls can be similarly difficult, especially if the benefits are a negative or hypothetical.
- Qualitative impact measurement can help in these situations; for example, a "minor" incident is unlikely to result in a business-ending fine, while a "catastrophic" incident could cause the business to become insolvent.
- Security practitioners will often find that they need to justify security controls based on non security benefits as well, such as enhanced usability of a cloud-based system for a mobile or remote workforce.
- In many cases, the end result of the cost-benefit analysis will be a presentation to management of various alternatives to mitigate, transfer, and ultimately accept residual risk, so the costs as well as any benefits resulting from controls must be clearly documented and communicated.

Assess the Effectiveness of Software Security

Mitigations Strategies

- Measuring the benefits of security controls can be similarly difficult, especially if the benefits are a negative or hypothetical.
- Qualitative impact measurement can help in these situations; for example, a "minor" incident is unlikely to result in a business-ending fine, while a "catastrophic" incident could cause the business to become insolvent.
- Security practitioners will often find that they need to justify security controls based on non security benefits as well, such as enhanced usability of a cloud-based system for a mobile or remote workforce.
- In many cases, the end result of the cost-benefit analysis will be a presentation to management of various alternatives to mitigate, transfer, and ultimately accept residual risk, so the costs as well as any benefits resulting from controls must be clearly documented and communicated.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Assess Security Impact of Acquired Software

Commercial Off-The-Shelf (COTS)

- With regard to security, acquired software is more complicated to assess and control than custom-built software.
- The ability to proactively control software development processes or make software fixes to address vulnerabilities typically does not exist when the organization does not control the development processes.
- This loss of control may seem like an unacceptable security risk, but it is balanced against the similarly steep costs of trying to develop skills and resources needed to build complex information systems.
- Acquiring software from the appropriate source can speed the achievement of critical business objectives, and compensating security controls should be chosen to mitigate risk to an acceptable level.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Assess Security Impact of Acquired Software

Commercial Off-The-Shelf (COTS)

- Most software vendors do not provide any visibility, though some share audit and assurance reports like code scanning or a certification like ISO 27001, which provides assurance over the vendor's security program, or ISO 27034, which provides assurance over the vendor's application security controls.
- The US *Cybersecurity & Infrastructure Security Agency* (CISA) maintains a best practices guide for managing COTS software security: (www.us-cert.cisa.gov/bsi/articles/bestpractices/legacy-systems/security-considerations-in-managingcots-software). [NOTE: OLD LINK]
- **New links:**
  - **https://www.cisa.gov/sites/default/files/publications/ESF_SECURING_THE_SOFTWARE_SUPPLY_CHAIN_DEVELOPERS.PDF**
  - **https://www.cisa.gov/sites/default/files/2023-01/ESF_SECURING_THE_SOFTWARE_SUPPLY_CHAIN_CUSTOMER.PDF**

CISSP® MENTOR PROGRAM – SESSION ELEVEN

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

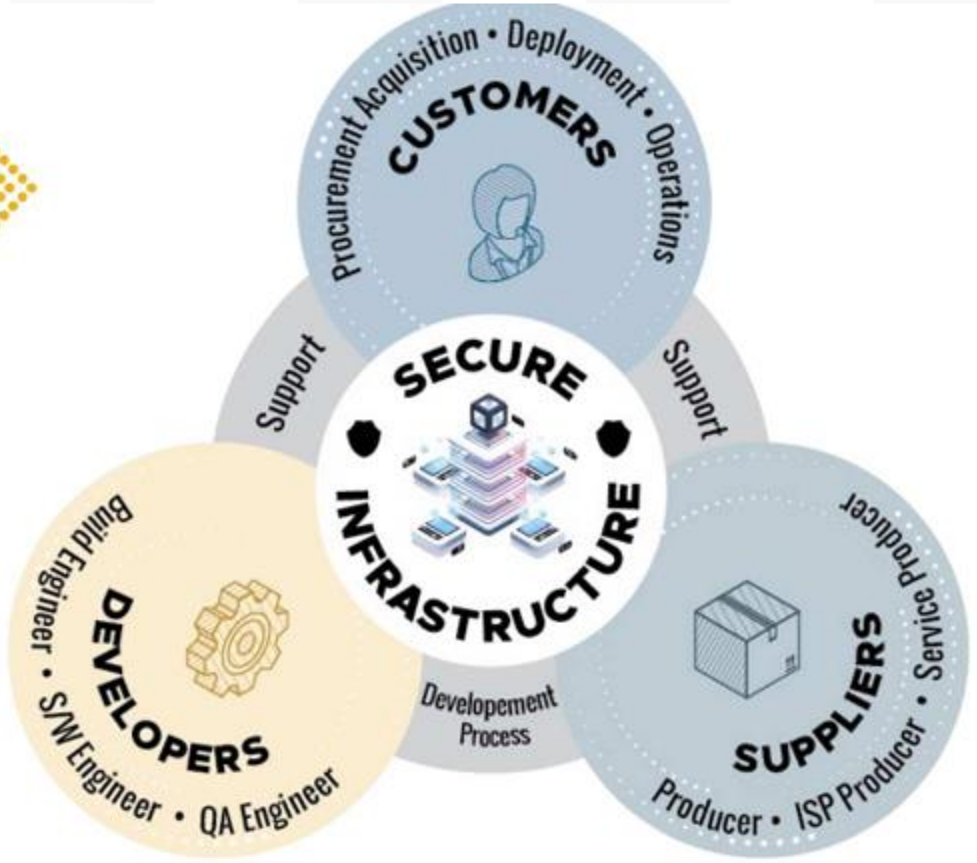
SECURING THE SOFTWARE SUPPLY CHAIN

CUSTOMERS

User Organizations (i.e. the Customer) follow a series of key phases in the acquisition, deployment, and operation of software products.

Customer teams specify to and rely on vendors for providing key artifacts (e.g. SBOM) and mechanisms to verify the software product, its security properties, and attest to the SDLC security processes and procedures.

Processes Involved in the Key Phases


CUSTOMERS
Procurement Acquisition • Deployment • Operations
Support
Support
SECURE INFRASTRUCTURE
DEVELOPERS
Build Engineer • S/W Engineer • QA Engineer
Developement Process
SUPPLIERS
Producer • ISP Producer • Service Producer

https://www.cisa.gov/sites/default/files/2023-01/esf_securing_the_software_supply_chain_customer_slicksheet.pdf

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Assess Security Impact of Acquired Software

COTS

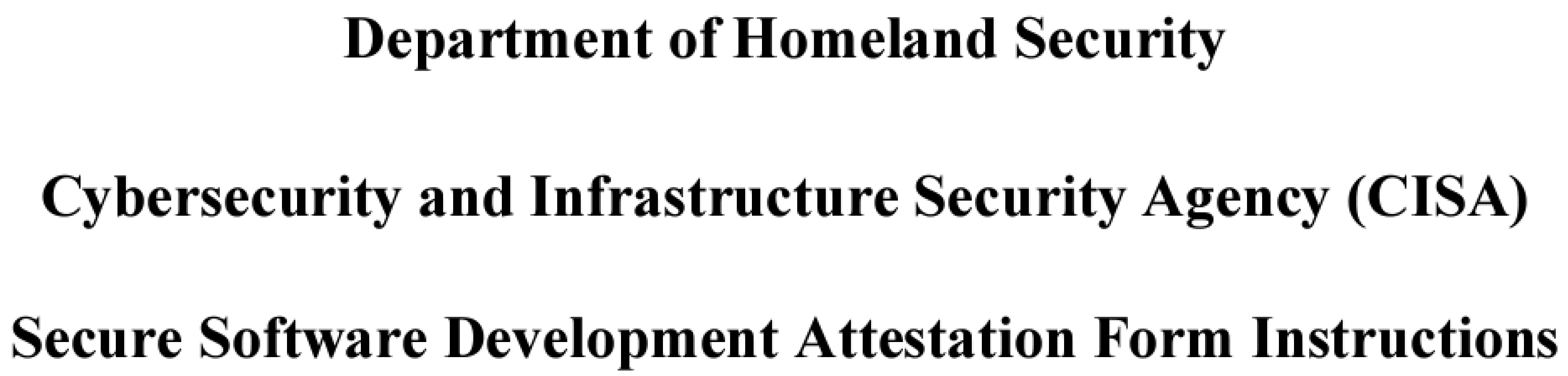
SECURING THE SOFTWARE SUPPLY CHAIN

RECOMMENDED PRACTICES GUIDE FOR CUSTOMERS

https://www.cisa.gov/sites/default/files/2023-01/ESF_SECURING_THE_SOFTWARE_SUPPLY_CHAIN_CUSTOMER.PDF

CISSP® MENTOR PROGRAM – SESSION ELEVEN

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY



Department of Homeland Security

Cybersecurity and Infrastructure Security Agency (CISA)

Secure Software Development Attestation Form Instructions

https://www.cisa.gov/sites/default/files/2023-04/secure-software-self-attestation_common-form_508.pdf

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Assess Security Impact of Acquired Software

Commercial Off-The-Shelf (COTS)

Identifying why COTS is risky - Well-known and widely used software such as Microsoft Windows and Internet Information Services are also well known by threat actors. Wide adoption of these systems increases the profitability of exploiting them since the pool of potential targets is bigger, and the software developer typically has limited liability in the event a flaw in their software leads to a security incident.

Common methods of attack - COTS software is often subject to malicious modification or interference like remote code execution (RCE) flaws or DOS attacks. The black-box nature of the source code also means the COTS software may contain unwanted malware or functionality, which the organization is unable to identify through conventional means like SAST

Standard risk assessment and mitigation strategies - Organizations using COTS software must have adequate risk management practices adapted to their use cases. This includes compiling an accurate inventory of COTS software and components and identifying sources of information for vulnerabilities, such as vendor disclosures, penetration testing, and routine security reviews. Defect prevention strategies during the SDLC are not an option, so mitigation must focus on corrective and compensating controls such as intrusion detection systems and network access controls.

Assess Security Impact of Acquired Software

Open Source

Makes its source code freely available. OSS is typically developed by a community for the benefit of all, allowing organizations to share and collaborate to improve the software while offering the benefit of source- code visibility.

Unlike COTS, all organizations using OSS can perform code testing to verify security, bugs, or unwanted functionality, and they can share fixes made to address security flaws back to the software project for the benefit of the community.

The community-driven approach of OSS has arguable security implications. On the one hand, the open nature means that bugs or flaws should eventually be caught, and the software is generally freely available for use. On the other hand, the complexity of OSS projects, the technical skills required to use the software, and the lack of dedicated support can make them less attractive than COTS alternatives. When evaluating OSS, there are a number of key concerns

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Assess Security Impact of Acquired Software

Open Source

Performing sufficient evaluation - Relying on testing and evaluation performed by untrusted outside parties can also lead to testing with insufficient coverage, differing objectives, or even results that are outdated or inaccurate if not performed by qualified testers.

Simply relying on the wisdom of the community or an administrator is insufficient.

Validate source code integrity - OSS repositories are typically configured for public visibility and contributions, meaning anybody can add, modify, or delete source code. There have been multiple instances where OSS projects had malicious code inserted, as well as instances of prepackaged OSS being made available from inauthentic sources. In both cases, unsuspecting users who downloaded and deployed the software got unwanted malware in addition to the legitimate software. Always ensure you review the changes made to OSS to ensure they are expected and properly documented, and only download OSS from the legitimate, official repositories.

Look for commercial support - OSS is typically licensed for free use and distribution, which means organizations are also left to find their own support. Companies such as Red Hat have sprung up by offering support for popular OSS like Linux, and the dedicated resources of such an arrangement have beneficial impacts on the overall quality and security of the OSS.

Assess Security Impact of Acquired Software

Third-Party

- Third-party software is developed under contract by a firm outside your organization's direct control, and the resulting software is typically for your organization's exclusive use.
- The use of a third party to develop software is a decision that should be made after a deliberate consideration of the costs and benefits.
- The availability of the source code itself is another important aspect of third-party software security to consider. If the development organization goes out of business, the contracting organization may find itself without support for critical systems; code escrow is a contractual arrangement often used to prevent this eventuality.
- The developing organization deposits a copy of the source code with a trusted third party, called an escrow agent.

*The contracting organization can get access to the code under limited circumstances, such as the developer going out of business or breaching the contract, which enables continued support and development of the system.

Assess Security Impact of Acquired Software

Managed Services (SaaS, IaaS, PaaS)

- Direct financial incentives like reducing costs for unneeded hardware or software capacity, as well as indirect incentives like secure remote accessibility to enable distributed teams, have combined to push many organizations to consume information systems as managed services.
- Billed via metered consumption, which is often a payas-you-use model.
- Widely agreed upon definitions of the various cloud service models are found in NIST SP 800-145, “The NIST Definition of Cloud Computing,” (Chapter 3)
- Emerging cloud services must be considered with respect to software security, including serverless and microservices architectures, which combine elements of managed software, infrastructure, and platform to enable faster deployment of code.
- *The choice of cloud services impacts the consumer's software security assessment abilities and responsibilities.

Assess Security Impact of Acquired Software

Assessing Shared Responsibilities

IaaS- The consumer will have the most flexibility to deploy software and therefore the most ability to perform software assessments. IaaS offers building blocks of information system functionality, so assessment software security assessment will incorporate COTS, OSS, and third-party software as appropriate to the type of software the consumer deploys in the IaaS environment.

PaaS- Some options are removed from the consumer's control, so compensating processes like reviewing audit reports and SLAs for software assurance are needed.

SaaS- Offers consumers the fewest options for directly assessing the security of software. However, given the shared nature of SaaS providers, there are often audit reports and robust control documentation available regarding the CSP's software development, testing, and hosting security practices.

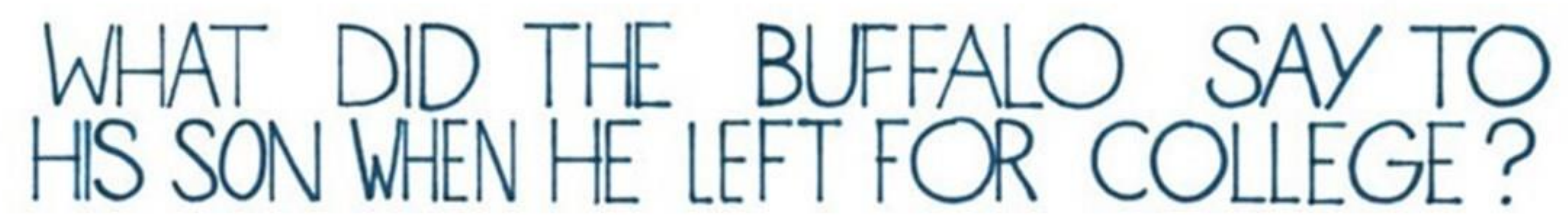
Assess Security Impact of Acquired Software

Audits and Assurance

- As with all third-party and supply-chain security concerns, assurance is a key concept for managed software services.
- Direct observation and control are not available, because the organization is not responsible for software development, so a trusted third party is often involved to perform an impartial review in the form of an audit.
- There are multiple security and compliance frameworks related to cloud computing, and these are covered in detail in Chapter 6.
- The downside, obviously, is that the consumers give up some measure of direct control over their risks.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Dad Joke Time



WHAT DID THE BUFFALO SAY TO HIS SON WHEN HE LEFT FOR COLLEGE?

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Define and Apply Secure Coding Guidelines and Standards

Intro

- Software developers should be provided with guidelines and standards to enable them to write code that achieves the organization's strategic objectives, particularly the requirement to safeguard confidentiality, integrity, and availability of systems and data.
- There is significant commonality for secure coding practices across industries and organizations, which has led groups like OWASP to design coding guidelines and standards for a variety of languages, software libraries, and application environments like web and mobile.
- Documented guidelines and standards can be used to implement multiple control actions. They can prevent bad programmer behaviors like developing code according to preference or familiarity rather than organizational standards.
- **Administrative and technical controls**

Define and Apply Secure Coding Guidelines and Standards

Security Weaknesses and Vulnerabilities at the Source-Code Level

Weaknesses and vulnerabilities are closely related, but there is a nuanced difference between the two terms.

- Vulnerabilities exist due to the presence of a software weakness such as a bug or flaw in the system's source code, which may result from errors or incorrect design choices made when architecting, developing, and implementing the system.
- Weaknesses can exist in a system with no practicable method of exploiting them, in which case they are not vulnerabilities; weaknesses can also impact non security aspects of a system, such as a poor database design that causes slow search performance.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Define and Apply Secure Coding Guidelines and Standards

Security Weaknesses and Vulnerabilities at the Source-Code Level

- A vulnerability, by contrast, is the way in which a weakness could be exploited in a software system.
- Non security weaknesses obviously are not exploitable, and some insecure source-code-level weaknesses may be compensated for by other parts of the overall system, such as an OS memory manager that prevents buffer overflow conditions otherwise allowed by poorly written source code.
- This is one of the challenges presented in software security testing: if a source code weakness is not exploitable, the cost-benefit analysis of fixing the weakness may not support the effort and resources needed to fix it.

Define and Apply Secure Coding Guidelines and Standards

Security Weaknesses and Vulnerabilities at the Source-Code Level

- Vulnerabilities are typically tracked by the software in which they exist, and more specifically by the affected version, allowing organizations to determine if a vulnerability affects their systems. Weaknesses and vulnerabilities are cataloged and scored using common systems to aid in identifying, communicating, and testing for them. This tracking and scoring are useful for prioritizing fixes like patches or the deployment of compensating controls.

Define and Apply Secure Coding Guidelines and Standards

Common Weakness Enumeration (CWE)

- The **Common Weakness Enumeration (CWE)** is maintained by the MITRE Corporation in partnership with various U.S. government agencies including the Department of Homeland Security.

- This community-developed list identifies common weaknesses in hardware, software source code, and implementations of software features that can lead to exploitable vulnerabilities. Each CWE is given a unique identifier such as CWE-561: Dead Code

https://cwe.mitre.org/

Define and Apply Secure Coding Guidelines and Standards

Common Weakness Enumeration

Description, Extended Description - Details of the weakness and its effects. The description for CWE-561 is: “The software contains dead code, which can never be executed.” This is code that, when executed, skips over certain portions or never calls those functions, thereby preventing it from ever being run.

Relationships - Other types of related weaknesses, such as the high-level category of Bad Coding Practices.

Modes of Introduction, Applicable Platforms - This identifies when this weakness may be introduced to software by SDLC phase, which is useful when designing security controls to deploy in the SDLC. Applicable Platforms identifies what languages or systems are susceptible. CWE-561 arises during the implementation phase and impacts all programming languages.

Common Consequences, Demonstrative Examples - These sections are self-explanatory. Consequences of dead code include the likelihood of unintended program behavior, since the function expected of the dead code will never be performed. Demonstrative examples help software developers by showing example code demonstrating the weakness.

Define and Apply Secure Coding Guidelines and Standards

Common Weakness Enumeration

Observed Examples - If the weakness has led to an actual exploit, the relevant Common Vulnerabilities and Exposures (CVE) reference is included. CVEs are discussed in detail in the following section.

Potential Mitigations, Detection Methods - These categories are most useful for security practitioners, as they provide actionable guidance to detect and mitigate the weaknesses, such as static code analysis to detect dead code, and mitigation tactics. In the case of CWE561, the mitigation is deceptively simple: remove the dead code. However, if the dead code had an intended purpose, new code that does not exhibit the same weakness is required.

Miscellaneous metadata - CWEs contain additional metadata like ordinality, taxonomy, references, and version history of the weakness.

Define and Apply Secure Coding Guidelines and Standards

Common Weakness Scoring System

The Common Weakness Scoring System (**CWSS**) is an attempt to score weaknesses and determine a priority for addressing them based on the **score**.

This can help **identify high-risk weaknesses** that should be addressed first given limited time and may also be used to identify low-risk weaknesses that will not be addressed due to the costs required and small benefit achieved.

The CWSS score comprises an evaluation of several factors, which are organized into three groups: **Base Finding, Attack Surface**, and **Environmental**. Each factor has values to account for uncertainty inherent in complex software projects, as well as the evolution over time of how much information is available about a weakness.

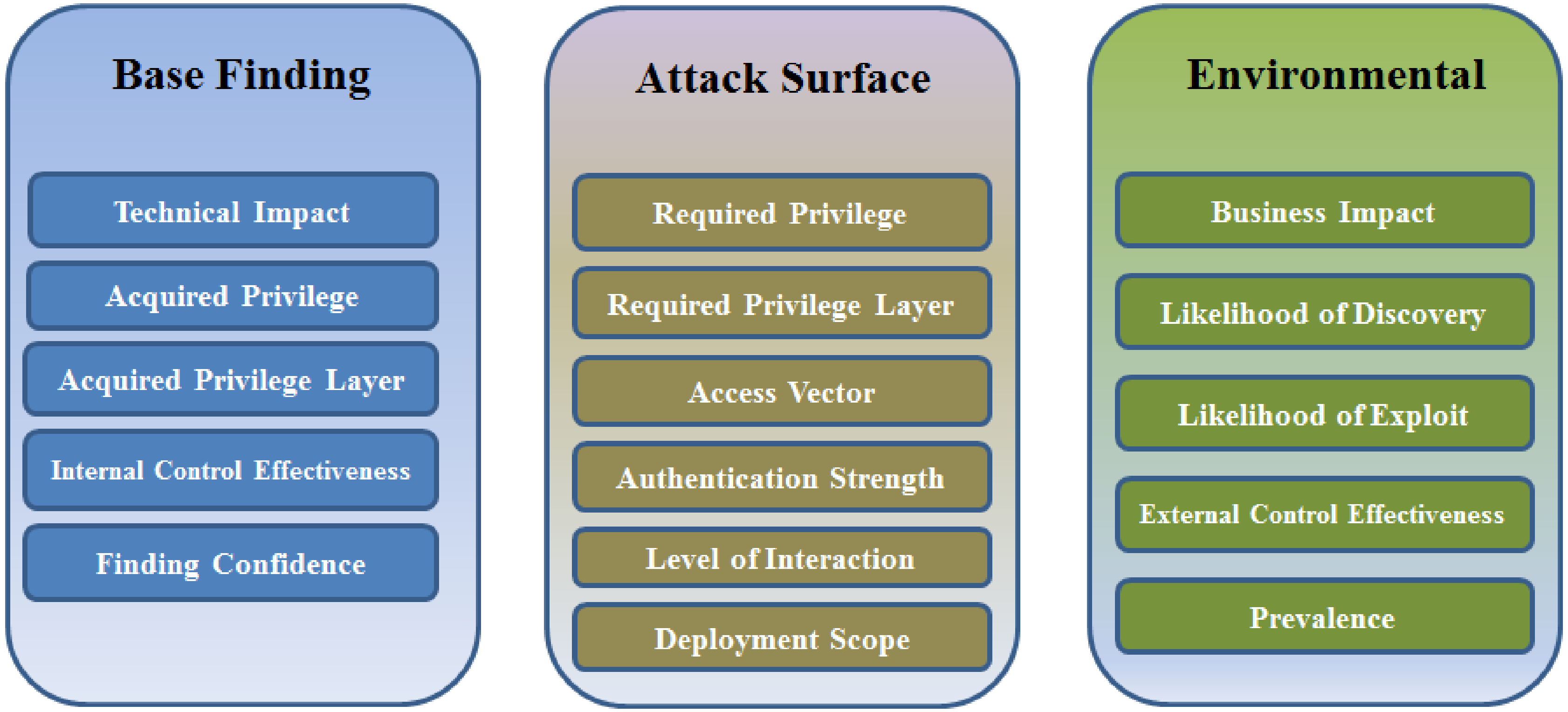
https://cwe.mitre.org/cwss/cwss_v1.0.1.html

CISSP® MENTOR PROGRAM – SESSION ELEVEN

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Define and Apply Secure Coding Guidelines and Standards

Common Weakness Scoring System



https://cwe.mitre.org/cwss/cwss_v1.0.1.html

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Define and Apply Secure Coding Guidelines and Standards

Common Weakness Scoring System

Base Finding - This helps quantify results of a weakness being exploited. Metrics include the following: **Technical Impact** (**TI**), **Acquired Privilege** (**AP**), **Acquired Privilege Layer** (**AL**), **Internal Control Effectiveness** (**IC**), and **Finding Confidence** (**FC**).

Attack Surface - This provides an explanation of the factors needed to exploit the weakness. Metrics include **Required Privilege** (**RP**), **Required Privilege Layer** (**RL**), **Access Vector** (**AV**), **Authentication Strength** (**AS**), **Level of Interaction** (**IN**), and **Deployment Scope** (**SC**).

Environmental - This describes the impact and likelihood of exploiting the weakness. Metrics include **Business Impact** (**BI**), **Likelihood of Discovery** (**DI**), **Likelihood of Exploit** (**EX**), **External Control Effectiveness** (**EC**), and **Prevalence** (**P**).

Define and Apply Secure Coding Guidelines and Standards

Common Vulnerabilities and Exposures (CVE)

- Similar to CWE, the Common Vulnerabilities and Exposures (CVE) provides a consistent way to not only identify but also describe common cybersecurity vulnerabilities.

- CVEs are often tied back to CWEs. For example, the “goto” fail vulnerability in Apple's macOS and iOS (CVE-2014-1266) resulted from poor coding practices that lead to code never being executed, which is an example of the previously discussed CWE-561, Dead Code.

https://cve.mitre.org/

Define and Apply Secure Coding Guidelines and Standards

Common Vulnerabilities and Exposures

- The CVE ID consists of two key pieces of information: the year in which the vulnerability was registered on the CVE list and a numeric ID.
- CVE records include a description of the vulnerability with details like impacted software and versions, any versions that are not impacted, risks related to the vulnerability, and workarounds or fixes.
- References are also provided, and they often consist of vendor-issued material like knowledge base articles or patching information, and links to relevant material about the discovery of the vulnerability.

*The full list of all CVEs reported, which goes back to 1999, is available at cve.mitre.org/index.html.

Define and Apply Secure Coding Guidelines and Standards

Common Vulnerabilities Scoring System

- The Common Vulnerability Scoring System (CVSS) provides that information, similar to CWSS. Currently in version 3.1, CVSS provides context related to the impact and severity of each CVE. They are accessible in the National Vulnerability Database (NVD) maintained by NIST at nvd.nist.gov/vuln-metrics/cvss.

- CVSS scores measure three categories of information related to each vulnerability, which are used to calculate three sub-scores that together comprise the overall CVSS score.

- A CVSS vector is also created, which provides a text-based reference to the underlying vulnerability measurements, such as CVSS:3.1/AV:A/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N. This is made up of the scores on each metric

https://nvd.nist.gov/vuln-metrics/cvss

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Define and Apply Secure Coding Guidelines and Standards

Common Vulnerabilities Scoring System

Base Score - metrics are related to the exploitability and impact of a particular vulnerability and, once established, do not typically change.

Exploitability metrics include the **Attack Vector** (**AV**), **Attack Complexity** (**AC**), **Privileges Required** (**PR**), **User Interaction** (**UI**), and **Scope** (**S**). Impact metrics cover the standard security triad of **Confidentiality** (**C**), **Integrity** (**I**), and **Availability** (**A**).

Temporal Score - metrics capture how the risk of a particular vulnerability changes over time, which means this number also changes over time

Metrics comprising the temporal score include **Exploit Code Maturity** (**E**), **Remediation Level** (**RL**), and **Report Confidence** (**RC**).

https://nvd.nist.gov/vuln-metrics/cvss

CISSP® MENTOR PROGRAM – SESSION ELEVEN

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Define and Apply Secure Coding Guidelines and Standards

Common Vulnerabilities Scoring System

Environmental Score - metrics capture how a particular vulnerability impacts an individual organization. A highly exploitable, high-impact vulnerability sounds drastic, but if an organization's only system with that vulnerability is in a secure physical location and not connected to a network, the patching process is not an emergency that should interrupt all other work. Environmental score metrics modify metrics from the other categories and are subdivided into three groups

- **Exploitability -** includes Attack Vector (MAV), Attack Complexity (MAC), Privileges Required (MPR), User Interaction (MUI), and Scope (MS).
- **Impact -** measures the specific effect of the vulnerability against the organization's CIA triad, including Confidentiality Impact (MC), Integrity Impact (MI), and Availability Impact (MA).
- **Impact Sub-score Modification -** modifies the Base Score impact metrics to tailor them to the organization's specific CIA requirements, including Confidentiality Requirement (CR), Integrity Requirement (IR), and Availability Requirement (AR).

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Define and Apply Secure Coding Guidelines and Standards

Common Vulnerabilities Scoring System

CVSS scores range from 0 to 10, with higher numbers denoting a more critical vulnerability that should be prioritized for fixing.

A score of **0** is categorized **None** and is mainly informational. A score between 0.1 and 3.9 is Low severity, between 4.0 and 6.9 is Medium severity, between 7.0 and 8.9 is High severity, and a score between 9.0 and **10** is **Critical**.

Security practitioners can monitor the NVD and use the CVSS calculators to determine the environmental scores of specific vulnerabilities to prioritize remediation activities.

Define and Apply Secure Coding Guidelines and Standards

OWASP Top 10

- The Open Web Application Security Project (OWASP) is a community-driven effort that originally started to share knowledge and develop best practices around web application development.
- It has since expanded in scope to include other application types, such as mobile apps, driven mainly by the increased prevalence of applications that rely on system components or services accessed over the internet and the wide variety of languages that are used for both web and traditional desktop apps.
- One of OWASP's most visible projects is the Top Ten, which is a compilation of the 10 most commonly seen vulnerabilities in web applications

CISSP® MENTOR PROGRAM – SESSION ELEVEN

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Define and Apply Secure Coding Guidelines and Standards

OWASP Top 10

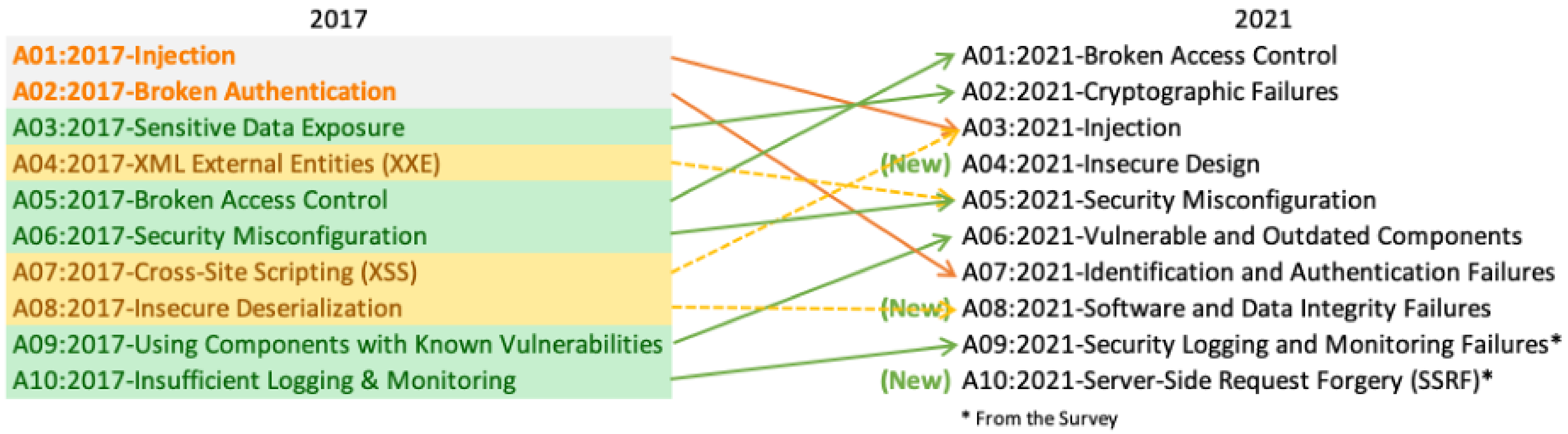
- The Open Web Application Security Project (OWASP) is a community-driven effort that originally started to share knowledge and develop best practices around web application development.
- It has since expanded in scope to include other application types, such as mobile apps, driven mainly by the increased prevalence of applications that rely on system components or services accessed over the internet and the wide variety of languages that are used for both web and traditional desktop apps.
- One of OWASP's most visible projects is the Top Ten, which is a compilation of the 10 most commonly seen vulnerabilities in web applications

https://owasp.org/www-project-top-ten/

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Define and Apply Secure Coding Guidelines and Standards

OWASP Top 10


2017
A01:2017-Injection
A02:2017-Broken Authentication
A03:2017-Sensitive Data Exposure
A04:2017-XML External Entities (XXE)
A05:2017-Broken Access Control
A06:2017-Security Misconfiguration
A07:2017-Cross-Site Scripting (XSS)
A08:2017-Insecure Deserialization
A09:2017-Using Components with Known Vulnerabilities
A10:2017-Insufficient Logging & Monitoring
2021
A01:2021-Broken Access Control
A02:2021-Cryptographic Failures
A03:2021-Injection
(New) A04:2021-Insecure Design
A05:2021-Security Misconfiguration
A06:2021-Vulnerable and Outdated Components
A07:2021-Identification and Authentication Failures
(New) A08:2021-Software and Data Integrity Failures
A09:2021-Security Logging and Monitoring Failures*
(New) A10:2021-Server-Side Request Forgery (SSRF)*
* From the Survey

https://owasp.org/www-project-top-ten/

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Define and Apply Secure Coding Guidelines and Standards

OWASP Cheat Sheets

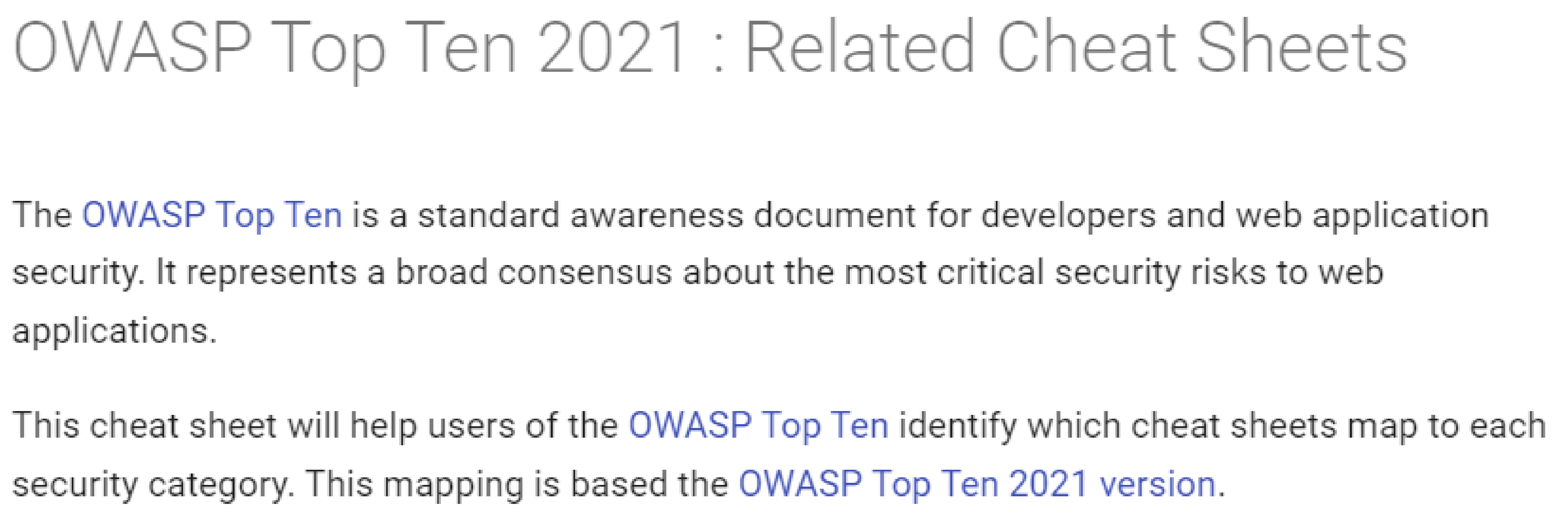
https://cheatsheetseries.owasp.org/

CISSP® MENTOR PROGRAM – SESSION ELEVEN

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Define and Apply Secure Coding Guidelines and Standards

OWASP Top 10



OWASP Top Ten 2021 : Related Cheat Sheets

The OWASP Top Ten is a standard awareness document for developers and web application security. It represents a broad consensus about the most critical security risks to web applications.

This cheat sheet will help users of the OWASP Top Ten identify which cheat sheets map to each security category. This mapping is based the OWASP Top Ten 2021 version.

https://cheatsheetseries.owasp.org/IndexTopTen.html

Define and Apply Secure Coding Guidelines and Standards

Software Composition Analysis

- One emerging area of security concern is the increasing use of open-source components, libraries, and modules in modern programs. These show up even in custom developed applications, and their use is directly related to Top 10 item A9:2017-Using Components with Known Vulnerabilities.

- These components are often known as free and open-source software (FOSS) dependencies and bring obvious benefits such as faster development and increased reliability if well-known components are used. FOSS means that these software components can be used at no charge, and the code is open for inspection or modification.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Define and Apply Secure Coding Guidelines and Standards

Software Composition Analysis

- Software composition analysis (SCA) is a security practice specifically designed to address FOSS component vulnerabilities and their impact on software.
- Open-source projects often publish security notices or include release notes or documentation addressing remediated vulnerabilities with updates, which an SCA tool can use to continuously identify vulnerabilities in an application using that component.
- DevOps and DevSecOps practices can also be utilized to address these flaws, such as CI/CD pipelines that incorporate automatic checks and incorporate the latest update dependencies when building an application.

Define and Apply Secure Coding Guidelines and Standards

Security of Application Programming Interfaces (APIs)

- An **API** provides **standardized access** to the features of a system without the need to understand its inner workings.
- Features that are accessible through an API are often referred to as being exposed via API, meaning they can be called using standardized methods such as HTTP requests.
- **Representation state transfer or REST APIs** expose functions using URLs similar to web applications and accept commands using common HTTP verbs like GET and POST for retrieving or sending data to the API.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Define and Apply Secure Coding Guidelines and Standards

Security of Application Programming Interfaces

- APIs are a form of modular software development, and modern application architectures often rely on APIs to send data between functions, modules, or tiers of complex systems.

- There are other forms of APIs as well, including internal software APIs in modern OSs for access to generic system functions like creating interactive windows in a user interface, **simple object access protocol** (**SOAP**) APIs designed for exchanging messages between system components, and RPC APIs that are used for executing functions across nodes in distributed systems.

- Most cloud services are built on an API foundation, with tasks such as cloud environment administrative, data handling, and services like data storage buckets and machine learning models exposed to users via URLs, commonly known as API endpoints.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Define and Apply Secure Coding Guidelines and Standards

Security of Application Programming Interfaces

- OWASP publishes a number of freely available resources on API security, including a cheat sheet for REST APIs (https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html) and an API Security Top 10 list detailing common vulnerabilities and issues facing API security (https://owasp.org/www-project-api-security/).

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Define and Apply Secure Coding Guidelines and Standards

Authentication and Access Control

Basic authentication - utilizes a simple authentication scheme that is part of the HTTP protocol and sends a username:password string encoded in base64. Since this is not a form of encryption, this form of authentication method should never be used alone. Stronger authentication methods are preferred, but if basic authentication must be used, the communication channel requires protection, such as TLS encryption.

Key-based authentication - serves dual purposes. A secret, shared symmetric key is used to encrypt data being passed. If the recipient can decrypt the data with a secret key associated with a unique identity, then the sender's identity can be authenticated, and the data is also protected in transit. Many applications with APIs allow users to log into a secure web application to retrieve the API key needed for encryption and authentication, providing an out-of-bound channel for key distribution.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Define and Apply Secure Coding Guidelines and Standards

Authentication and Access Control

Certificate-based authentication - as the name implies, relies on authenticating an identity using digital certificates. This can be one-way, in which only one party provides a certificate asserting an identity, or mutual authentication where both parties exchange certificates and validate them. This obviously requires a Public Key Infrastructure (PKI) in place, meaning this may not be a valid choice for all situations.

Federated and single sign-on - authentication relies on another organization's identity and access management (IDAM) tools to control access. Technologies like security assertion markup language (SAML) and Oauth can be used to federate, or combine, IDAM capabilities across organizations. API security can rely on these federations to delegate access control decisions; rather than the API provider enforcing access controls, consumers grant permissions to their users and then pass relevant authentication and access control information via SAML statements or Oauth assertions.

Define and Apply Secure Coding Guidelines and Standards

Input Validation and Sanitization

- Whenever data is supplied by an outside system or user, it should be considered untrusted and treated appropriately.
- This can avoid unwanted application behavior like code injection attacks and may also be implemented as a control for data integrity.
- The API layer of an application is a key control point as it accepts and processes untrusted data supplied by external users or systems.
- Input validation checks should be implemented at the API layer to validate the incoming data matches expectations. Syntactic validation enforces correct syntax of data.
- Performing syntactic and semantic validation can help avoid data integrity issues and unwanted system behavior due to unexpected values, while semantic validation can also help identify unwanted content like SQL statements in a field such as Last_Name where a simple text string is expected

Define and Apply Secure Coding Guidelines and Standards

Input Validation and Sanitization

- Performing syntactic and semantic validation can help avoid data integrity issues and unwanted system behavior due to unexpected values, while semantic validation can also help identify unwanted content like SQL statements in a field such as Last_Name where a simple text string is expected.
- Unfortunately, semantic validation is extremely difficult, as the various combinations of human language make it tough to write the rules needed to check.
- For example, the ' character has special meaning in SQL and could be an unwanted character in a name field, but rejecting it would cause common names like O'Brian to be rejected. As a result, escaping content is considered a better safeguard against injection attacks, while semantic validation is generally more useful for data integrity.
- The OWASP Input Validation Cheat Sheet provides a comprehensive resource on validation strategies: cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_S heet.html.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Define and Apply Secure Coding Guidelines and Standards

Protection of Resources

- APIs must be designed with the same level of rigor applied to user access management to ensure the functionality exposed cannot be exploited to attack the system's confidentiality, integrity, or availability.

- APIs are typically designed for system-to system communication or for privileged access, and the possibility of their use in an attack is not considered during threat modeling.

CISSP® MENTOR PROGRAM – SESSION ELEVEN

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Define and Apply Secure Coding Guidelines and Standards

Protection Communications

- Web applications and other communications implement protections like **TLS** for encrypting data in transit, and API access should be no different.
- Symmetric key-based authentication not only can verify a user's identity but also provide security of data in transit, as can certificate-based authentication where a public key can be used either for encryption of data or for a session key.
- The use of standard web protocols is a benefit of REST APIs, as they utilize HTTPS with little developer effort.
- The level of communications security required should of course be driven by the sensitivity level of the data.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Define and Apply Secure Coding Guidelines and Standards

Cryptography

Confidentiality - both symmetric and asymmetric keys can be used, depending on the system requirements. A key design choice that must be made is when to encrypt data, as it can be encrypted prior to transport or utilize secure communications channels.

Integrity - Hash functions should be applied whenever the integrity of data being accepted through an API needs to be validated.

Authenticity - Proving the authenticity of data often relies on proving that the parties in communication are the legitimate sender and recipient. Strategies such as Hashed Message Authentication Code (HMAC) can be used to verify that the communications partners are still the intended parties.

Nonrepudiation - Proving the source of data or the identity of a system actor rests on identifying the holder of keying material.

Define and Apply Secure Coding Guidelines and Standards

Cryptography

Access control - Access to encrypted data relies on decrypting it, so by extension, controlling access to the key controls access to the data.

- In the case of symmetric keys, this involves implementing secure distribution and management mechanisms for the encryption keys. For asymmetric encryption, it involves the choice of which key is used to encrypt data.
- Using a sender's private key will allow anybody to decrypt the data with the corresponding public key, which proves authenticity but provides no confidentiality.
- Using a recipient's public key to encrypt data provides confidentiality because only the recipient's private key can decrypt it, but without proof of the sender's identity since the public key is widely available.

Define and Apply Secure Coding Guidelines and Standards

Security Logging, Monitoring, and Alerting

- System actions performed via API must be part of a logging and monitoring strategy to verify that system usage via the API meets security requirements.
- The events to be logged should be determined by the system's requirements for both operational and security monitoring.
- Logs are made meaningful by reviewing them, so they must be either reviewed by a person or ingested into a log monitoring tool like a SIEM that automates the process, depending on the organization's monitoring strategy and resources.
- Events that violate security constraints or indicate operational issues should generate alerts to invoke processes like incident response to investigate.
- Where possible, log data from multiple sources should be correlated to provide investigators with a holistic view of the incident, like network device logs showing traffic, application logs showing API access and use, and even, possibly, external data such as known malicious IP addresses that can be used to determine whether a security incident is a malicious attack.

Define and Apply Secure Coding Guidelines and Standards

Security Testing APIs

- As with all system functionality, APIs must be tested to verify they meet security, performance, and user requirements.
- All the testing methods and considerations presented in Domain 6 apply to APIs, such as designing a testing strategy with appropriate depth and coverage as indicated by the API's risk profile.
- An internal API used to exchange financial data between departments without any public access should not be scheduled for a pentest before undergoing functional testing to ensure proper data integrity, since the risk to integrity is greater than the risk of breaching confidentiality.

Define and Apply Secure Coding Guidelines and Standards

Security Testing APIs

- There are a number of API security tools available to design and test APIs.
- The choice of a tool and associated testing strategy should be driven by the technologies in use, such as SOAP or REST, the sensitivity level of data being handled by the API, and the features exposed by the API.
- OWASP also has a dedicated API security project, which contains resources such as a Security Top 10 and planned cheat sheet for API security, available at https://owasp.org/www-project-api-security/

Define and Apply Secure Coding Guidelines and Standards

Secure Coding Practices

- Secure coding practices comprise the entirety of actions taken during software and systems development processes. Nonexistent, disorganized, or poorly understood organizational practices tend to produce software and systems with flaws or bugs, which in turn become weaknesses and vulnerabilities.

- Standardized secure coding practices that are well-documented, disseminated to all stakeholders, and applied effectively throughout the organization can increase the quality of development projects.

- This leads to software and system development processes that seek to minimize flaws and bugs, as well as testing and monitoring processes designed to catch and fix vulnerabilities as soon as possible.

Define and Apply Secure Coding Guidelines and Standards

Standards for Secure Software Development

- The OWASP project contains a multitude of resources for secure coding practices. The Cheat Sheet Series (https://cheatsheetseries.owasp.org/) is an entirely library of guidance on secure practices, such as the following:
  - Implementing authentication and access controls
  - Preventing common vulnerabilities and attacks like XSS and injection
  - Architectural security for technologies like microservices, REST APIs, and SAML
  - Technology-specific security guidance including XML, HTML5, Docker, and Kubernetes

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Define and Apply Secure Coding Guidelines and Standards

Standards for Secure Software Development

- A technology-neutral OWASP Secure Coding Practices Quick Reference Guide (owasp.org/www-pdfarchive/OWASP_SCP_Quick_Reference_Guide_v2.pdf) is also available and provides recommendations and best practices across a set of common development activities.
- Although primarily written for web and mobile apps, the guidance is broadly applicable to almost any application and system architecture.
- Each category contains key focus areas for secure coding practices, and these practices can be used to guide the development and implementation of organization-specific practices tailored to your unique technology stack, programming languages, and security requirements.

https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/

https://github.com/OWASP/www-project-secure-coding-practices-quick-reference-guide

Define and Apply Secure Coding Guidelines and Standards

Standards for Secure Software Development

There are 13 categories, each of which contain a checklist of common practices, covering the following:

- Input validation
- Output encoding
- Authentication and password management
- Session management
- Access control
- Cryptographic practices
- Error handling and logging
- Communication security
- System configuration
- Database security
- File management
- Memory management
- General coding practices

https://www.securecoding.com/blog/owasp-secure-coding-checklist/

Define and Apply Secure Coding Guidelines and Standards

Standards for Secure Software Development

- The Carnegie Mellon University Software Engineering Institute (CMU SEI) publishes guides covering both secure coding practices as well as secure coding standards.
- The Top 10 Secure Coding Practices, which actually contains two bonus practices, is a very high-level set of principles for implementing secure coding like using input validation and following the “keep it simple” rule.
- The Coding Standards are a set of publications designed to provide standards for secure software development in specific languages including C/C++, Android, and Perl.
- While not entirely focused on security, they do cover best practices for each language and include some security-specific guidance.

*These documents and others related to secure coding can be found at the CMU SEI CERT Secure Coding wiki at https://wiki.sei.cmu.edu/confluence/display/seccode/SEI+CERT+Coding+Standards.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Define and Apply Secure Coding Guidelines and Standards

Education and Culture

- Security culture begins with clear and effective policies and other documentation that provides clear guidance to members of the organization on their security roles and responsibilities.
- This is the essence of building a security culture, in which all stakeholders understand, support, and actively carry out security efforts, and it is essential that this is widely adopted across the organization, since cybersecurity is a broad discipline underpinning any process that relies on data and information systems.

- While the documentation should be clear enough for broad consumption, additional documentation like procedures, guidelines, standards, and checklists must be created to provide specific guidance to different stakeholders.
- One crucial element of a security culture is the set of principles it is built on, and the foundational role played by software development means a number of key principles must be considered in order to create a successful security culture. These principles include the following:

Define and Apply Secure Coding Guidelines and Standards

Education and Culture

Secure by design - This principle largely stems from a paradigm shift as the field of software engineering matured and decisions that led to insecure technologies like DNS and SNMP were examined in light of modern information security risks. Rather than attempting to retrofit security after deployment, this principle espouses practices that require the inclusion of security at the outset of a system: in the design phase. Anticipating that the system will come under attack drives design and implementation decisions that provide inherent security abilities, rather than relying on more expensive and error-prone compensating controls.

Secure by default - This principle is a collection of best practices for delivering more secure software, such as the inclusion of layered security controls and conservative default settings designed for security. Many data breaches share the underlying root cause of improperly secured cloud storage services exposing data to the public. In a secure by default configuration, those services should be private by default and require extra steps be taken to turn on public access.

CISSP® MENTOR PROGRAM – SESSION ELEVEN

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Define and Apply Secure Coding Guidelines and Standards

Education and Culture

DevSecOps - Although not strictly a security principle, one of the foundational ideas of DevSecOps is to shift security “to the left” — in other words, to push security activities earlier in the software development lifecycle rather than developing security strategies only after a system is built and deployed. Implementing DevSecOps in an organization is, fundamentally, an opportunity to build a culture of security and deliver higher quality software.

Build security in - SAMM and BSIMM were discussed earlier in this chapter and represent a set of practices and principles that can be used to drive enterprise change to a culture of security. The maturity model supports this by providing a convenient measure of the organization's current state, metrics to track progress toward a desired future state, and a set of practices to implement to mature the secure software development capability.

Define and Apply Secure Coding Guidelines and Standards

Software-Defined Security

The world of technology is rapidly evolving to software defined replacements for systems that previously required a combination of physical hardware and software. For example, **software-defined networking (SDN)** abstracts physical networking gear and software configurations into purely software-configurable virtual networks that can be reconfigured with far less effort than their hardware-based equivalents. Cloud services also make use of software defined infrastructure in the form of virtualization, with common infrastructure elements such as server and networking hardware replaced by more flexible software only equivalents.

Software-defined security, sometimes abbreviated **SDS or SDSec**, mirrors this trend to create virtualized, software-controlled security infrastructure. It extends concepts like virtualized infrastructure configuration via definition files, as well as leveraging SDN to provide easy reconfiguration of network functions, routes, and protection mechanisms. Provisioning, monitoring, and management of security functions can be extensively automated and controlled via software configuration, which replaces previously manual, human-driven processes. Deploying SDS represents an evolution of security architecture to support more targeted and flexible controls. It also accounts for new application and system architectures that do away with traditional designs due to novel services available in cloud computing.

CISSP® MENTOR PROGRAM – SESSION ELEVEN

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Define and Apply Secure Coding Guidelines and Standards

Software-Defined Security

Consider, for example, a legacy application that comprises three tiers: a web server, a business logic server, and a database server. Each has traditional security tools installed like anti-malware, file integrity monitoring, IDS/IPS, etc., and is hosted in a data center with traditional perimeter controls like a firewall. A modern application, by contrast, may comprise a web server delivering a browser-based application to users, whose actions create API calls to a set of microservices for specific tasks like processing data, which in turn involves API calls to a cloud-hosted database. All of these system elements are also geographically disbursed and configured for high availability, so user requests may be processed in data centers all over the world with different hardware and software configurations. There is no traditional perimeter, and the business logic and database cannot be monitored with traditional tools. The microservices don't exist on a traditional server, and the cloud-hosted database server is not under the purview of the organization's security program.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Define and Apply Secure Coding Guidelines and Standards

Software-Defined Security

Enhanced availability - Implementing security controls requires an element of determinism; for example, deploying a firewall requires the practitioner to know data will be flowing between two points, and the firewall is placed in the middle. If a different communications channel is used, the firewall's protection is rendered moot. Dynamic and geographically distributed cloud applications make this task difficult with traditional security, but an SDS paradigm can be used to ensure security controls follow the data. Software definition files are easily portable, so it is possible to ensure a new virtual server will have the same configuration and protections regardless of its location. Increased security tool flexibility allows organizations to take advantage of high availability in cloud environments.

Infrastructure neutrality - Security controls often require administrators with system-specific configuration and administrative knowledge. SDS allows for a generic definition to be interpreted as appropriate for underlying system and for the process to be automated. For example, an encryption setting of “AES with minimum 256-bit key length” can be interpreted by a virtualization server and appropriately applied to all new servers regardless of the host OS being deployed. This is similar to the interpreted code paradigm, which allows for write-once, run-anywhere portability. SDS definitions also support increased configuration reliability and consistent application of security policy, similar to using standard configuration files rather than relying on manual build processes.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Define and Apply Secure Coding Guidelines and Standards

Software-Defined Security

Micro-targeting - Security tools are often blunt and lack adequate granularity for enforcing controls, which requires trade-offs or compensating controls. This is especially apparent in network segmentation, which is covered in Domain 4, "Communication and Network Security," and it is a problem solved by the microsegmentation that SDNs enable. Rather than requiring a balance between protection of individual hosts and network administration overhead, individual hosts automatically receive appropriate firewall rules applied based on purpose-driven templates. Other security tool deployment and configurations can be similarly automated, such as IDS with rules appropriate for the server's function, or encryption configuration based on the classification label applied to the system. This granularity avoids trade-offs between management overhead and insufficient granularity of controls.

Centralized management -Many security tools are platform- or technology-specific, and they tend to be siloed along these lines. SDS enables centralized management and oversight by abstracting security control details away from the infrastructure and associated management tools. SDS enables security practitioners to see a consolidated view of access control policies and control implementations, while system administrators are still able to perform their tasks in system-specific tools like Active Directory or an LDAP

Define and Apply Secure Coding Guidelines and Standards

Software-Defined Security

Standardization, integrations, and automation - Standardization and integrations are key to enabling the features of SDS discussed here. Definition files are written in standard formats such as YAML, which alternatively stands for Yet Another Markup Language or the recursive YAML Ain't Markup Language. These can be interpreted by various cloud providers, systems, or platforms to enforce the desired security configurations appropriate to the infrastructure. This standardization, coupled with integrations between infrastructure and security components like cloud services and SOAR tools, enables automated security control deployment, enforcement, and reconfiguration.

Dynamic response - Security controls can proactively reduce the likelihood of a risk or reactively reduce the impact of a risk after it is realized. SDS incorporates and extends automated capabilities from SOAR to enable faster response times. Restricting the time an attacker has to move laterally from a compromised server to other hosts reduces their chances of successful malicious activity. Dynamic response leverages the ease of reconfiguring virtualized software infrastructure to speed up the response, and the use of orchestration and automation capabilities means intelligence from an incident can be used to automatically deploy proactive mitigations to other hosts and systems. Tools with these capabilities are often described as self-healing, though it is important to note that security practitioners must still maintain vigilance and perform some actions manually.

DOMAIN 8: SOFTWARE DEVELOPMENT SECURITY

Define and Apply Secure Coding Guidelines and Standards

Software Bill of Materials (SBOM)

A "software bill of materials" (SBOM) has emerged as a key building block in software security and software supply chain risk management. A SBOM is a nested inventory, a list of ingredients that make up software components. The SBOM work has advanced since 2018 as a collaborative community effort, driven by National Telecommunications and Information Administration's (NTIA) multistakeholder process.

CISA will advance the SBOM work by facilitating community engagement, development, and progress, with a focus on scaling and operationalization, as well as tools, new technologies, and new use cases. This website will also be a nexus for the broader set of SBOM resources across the digital ecosystem and around the world.

An SBOM-related concept is the Vulnerability Exploitability eXchange (VEX). A VEX document is an attestation, a form of a security advisory that indicates whether a product or products are affected by a known vulnerability or vulnerabilities.

http://ntia.gov/SBOM
https://www.cisa.gov/sbom

**NOT IN THE BOOK
NEED TO KNOW!!!**

SESSION 11 - FIN
YOU MADE IT!

Domain 8 is done WHOOT HECK YA!! YALL!

Domain 8 can be a challenge because it's so dense.

Next Session - Review and Practice exams– Everyone

- Identification and classification Information and Assets
- Asset handling requirements
- Provision and inventory
- Management
- Roles
- Data Lifecyle and controls

CISSP® MENTOR PROGRAM – SESSION ELEVEN

SESSION 11 - FIN

Domain 8 Links

- https://www.cisa.gov/sites/default/files/2023-01/esf_securing_the_software_supply_chain_customer_slicksheet.pdf
- https://www.cisa.gov/sites/default/files/publications/ESF_SECURING_THE_SOFTWARE_SUPPLY_CHAIN_DEVELOPERS.PDF
- https://www.cisa.gov/sites/default/files/2023-01/ESF_SECURING_THE_SOFTWARE_SUPPLY_CHAIN_CUSTOMER.PDF
- https://www.nccoe.nist.gov/sites/default/files/2022-09/Guidelines%20for%20Enhancing%20Software%20Supply%20Chain%20Security%20Under%20EO%2014028.pdf
- https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-218.pdf
- https://i.blackhat.com/us-18/Thu-August-9/us-18-Lipner-SDL-For-The-Rest-Of-Us.pdf
- https://safecode.org/
- https://owaspsamm.org/
- https://owaspsamm.org/blog/2020/10/29/comparing-bsimm-and-samm/
- https://www.synopsys.com/software-integrity/software-security-services/bsimm-maturity-model.html
- https://www.synopsys.com/software-integrity/engage/bsimm/bsimm13

CISSP® MENTOR PROGRAM – SESSION ELEVEN

SESSION 11 - FIN

Domain 8 Links

- https://www.cyberark.com/what-is/ci-cd-pipeline/
- https://insights.sei.cmu.edu/blog/benefits-and-challenges-of-soar-platforms/
- https://csrc.nist.gov/publications/detail/sp/800-92/final
- https://cwe.mitre.org/
- https://cwe.mitre.org/cwss/cwss_v1.0.1.html
- https://cve.mitre.org/
- https://nvd.nist.gov/vuln-metrics/cvss
- https://owasp.org/
- https://owasp.org/www-project-top-ten/
- https://cheatsheetseries.owasp.org/
- https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html
- https://owasp.org/www-project-api-security/
- https://github.com/OWASP/www-project-secure-coding-practices-quick-reference-guide
- https://www.securecoding.com/blog/owasp-secure-coding-checklist/
- https://wiki.sei.cmu.edu/confluence/display/seccode/SEI+CERT+Coding+Standards
- http://ntia.gov/SBOM
- https://www.cisa.gov/sbom

CISSP® MENTOR PROGRAM – SESSION ELEVEN

SESSION 11 - FIN
YOU M

Domain 8

Domain

Next Se e

- Ident sets
- Asset
- Provi
- Mana
- Roles
- Data

FRSecure CISSP Mentor Program

2023

Class #11 – Domain 8

Software Development Security

Ron Woerner, CISSP®
President of Something