CIS Control 16: Application Software Security

Manage the security life cycle of in-house developed, hosted, or acquired software to prevent, detect, and remediate security weaknesses before they can impact the enterprise.

Why is this CIS Control Critical?

Applications provide a human-friendly interface to allow users to access and manage data in a way that is aligned with business functions. They also minimize the need for users to deal directly with complex (and potentially error-prone) system functions, like logging into a database to insert or modify files. Enterprises use applications to manage their most sensitive data and control access to system resources. Therefore, an attacker can use the application itself to compromise the data instead of an elaborate network and system hacking sequence that attempts to bypass network security controls and sensors. This is why protecting user credentials (specifically application credentials) defined in CIS Control 6 is so important.

Lacking credentials and application flaws are the attack vectors of choice. However, today's applications are developed, operated, and maintained in a highly complex, diverse, and dynamic environment. Applications run on multiple platforms: web, mobile, cloud, etc., with application architectures that are more complex than legacy client-server or database-web server structures. Development life cycles have become shorter, transitioning from months or years in long waterfall methodologies to DevOps cycles with frequent code updates. Also, applications are rarely created from scratch and are often "assembled" from a complex mix of development frameworks, libraries, existing code, and new code. There are also modern and evolving data protection regulations dealing with user privacy. These may require compliance with regional or sector-specific data protection requirements.

These factors make traditional approaches to security, like control (of processes, code sources, run-time environment, etc.), inspection, and testing are much more challenging. Also, the risk that an application vulnerability introduces might not be understood except in a specific operational setting or context.

Application vulnerabilities can be present for many reasons: insecure design, insecure infrastructure, coding mistakes, weak authentication, and failure to test for unusual or unexpected conditions. Attackers can exploit specific vulnerabilities, including buffer overflows, exposure to Structured Query Language (SQL) injection, cross-site scripting, cross-site request forgery, and click-jacking of code to gain access to sensitive data or take control over vulnerable assets within the infrastructure as a launching point for further attacks.

Applications and websites can also be used to harvest credentials data or attempt to install malware onto the users who access them.

Finally, it is now more common to acquire Software as a Service (SaaS) platforms, where software is developed and managed entirely through a third party. These might be hosted anywhere in the world. This brings challenges to enterprises that need to know what risks they are accepting with using these platforms, and they often do not have visibility into the development and application security practices of these platforms. Some of these SaaS platforms allow for customizing of their interfaces and databases. Enterprises that extend these applications should follow this CIS Control, similar to if they were doing ground-up customer development.

16.1: Establish and Maintain a Secure Application Development Process

Establish and maintain a secure application development process. In the process, address such items as: secure application design standards, secure coding practices, developer training, vulnerability management, security of third-party code, and application security testing procedures. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.

Asset Type	Security Function	Implementation Groups
Documentation	Govern	2, 3

Dependencies

- None

Inputs

1. GV49: Secure Application Development Process
2. Date of last update or review of the secure application development process

Operations

1. Determine whether Input 1 exists within the enterprise
    1. If Input 1 exists, M1 = 1
    2. If Input 1 does not exist, M1 = 0
2. Review Input 1 and determine whether it includes, at a minimum, the following components: secure application design standards, secure coding practices, developer training, vulnerability management, security of third-party code, and application security testing procedures
    1. For each component included in the process, assign a value of 1. Sum all values (M2)
3. Compare Input 2 to the current date and capture the timeframe in months (M3)

Measures

- M1 = Output of Operation 1
- M2 = Count of components included in the process
- M3 = Timeframe in months since last review or update

Metrics

- If M1 is 0, this Safeguard receives a failing score. The other metrics don't apply.
- If M3 is greater than twelve months, then this Safeguard is measured at a 0 and receives a failing score. The other metrics don't apply.

Completeness

Metric	**The percentage of components included in the secure application development process**
Calculation	`M2 / 6`

16.2: Establish and Maintain a Process to Accept and Address Software Vulnerabilities

Establish and maintain a process to accept and address reports of software vulnerabilities, including providing a means for external entities to report. The process includes such items as: a vulnerability handling policy that identifies the reporting process, responsible party for handling vulnerability reports, and a process for intake, assignment, remediation, and remediation testing. As part of the process, use a vulnerability tracking system that includes severity ratings and metrics for measuring timing for identification, analysis, and remediation of vulnerabilities. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.

Third-party application developers need to consider this an externally-facing policy that helps to set expectations for outside stakeholders.

Asset Type	Security Function	Implementation Groups
Documentation	Govern	2, 3

Dependencies

- None

Inputs

1. `GV48`: Process to Accept and Address Software Vulnerabilities
2. Date of last update or review of process

Operations

1. Determine whether Input 1 exists within the enterprise
    1. If Input 1 exists, M1 = 1
    2. If Input 1 does not exist, M1 = 0
2. Review Input 1 `GV48` and determine whether it includes, at a minimum, the following components: a reporting process, a responsible party for handling vulnerability reports, a process for intake, assignment, remediation, remediation testing, and a vulnerability tracking system
    1. For each component included in the process, assign a value of 1. Sum all values. (M2)
3. Compare Input 2 to the current date and capture the timeframe in months (M3)

Measures

- M1 = Output of Operation 1
- M2 = Count of components included in the process
- M3 = Timeframe in months since last review or update

Metrics

- If M1 is 0, this Safeguard receives a failing score. The other metrics don't apply.
- If M3 is greater than twelve months, then this Safeguard is measured at a 0 and receives a failing score. The other metrics don't apply.

Completeness

Metric	**The percentage of components included in the secure application development process**
Calculation	`M2 / 7`

16.3: Perform Root Cause Analysis on Security Vulnerabilities

Perform root cause analysis on security vulnerabilities. When reviewing vulnerabilities, root cause analysis is the task of evaluating underlying issues that create vulnerabilities in code, and allows development teams to move beyond just fixing individual vulnerabilities as they arise.

Asset Type	Security Function	Implementation Groups
Software	Protect	2, 3

Dependencies

- Safeguard 16.2: Establish and Maintain a Process to Accept and Address Software Vulnerabilities

Inputs

1. Root Cause Analysis Process
2. Vulnerabilities addressed over the last twelve months

Operations

1. Determine whether Input 1 exists within the enterprise
    1. If Input 1 exists, M1 = 1
    2. If Input 1 does not exist, M1 = 0

2. Review Input 1 and determine whether it includes, at a minimum, the following components: categorization of vulnerabilities, guidance for how lessons learned are incorporated into the development process

    1. For each component included in the process, assign a value of 1. Sum all values. (M2)

3. For each vulnerability addressed over the last twelve months, assess whether the root cause analysis process was followed

    1. Identify and enumerate vulnerabilities for which the process was followed (M4)
    2. Identify and enumerate vulnerabilities for which the process was not followed (M5)

Measures

- M1 = Output of Operation 1
- M2 = Count of components included in the process
- M3 = Count of Input 2
- M4 = Count of vulnerabilities for which root cause analysis was conducted
- M5 = Count of vulnerabilities for which root cause analysis was not conducted

Metrics

- If M1 is 0, this Safeguard receives a failing score. The other metrics don't apply.

Completeness of Process

Metric	**The percentage of components included in the secure application development process**
Calculation	`M2 / 2`

Compliance

Metric	**The percentage of vulnerabilities for which root cause analysis was conducted**
Calculation	`M4 / M3`

16.4: Establish and Manage an Inventory of Third-Party Software Components

Establish and manage an updated inventory of third-party components used in development, often referred to as a "bill of materials," as well as components slated for future use. This inventory is to include any risks that each third-party component could pose. Evaluate the list at least monthly to

identify any changes or updates to these components, and validate that the component is still supported.

Asset Type	Security Function	Implementation Groups
Software	Identfy	2, 3

Dependencies

- Safeguard 2.1: Establish and Maintain a Software Inventory

Inputs

1. `GV47`: Inventory of Third-Party Software Components
2. Date of last review or update of the inventory

Operations

1. Determine whether Input 1 exists within the enterprise
    1. If Input 1 exists, M1 = 1
    2. If Input 1 does not exist, M1 = 0
2. Use Input 1 and determine whether each software component listed includes, at a minimum, the following information: risk associated with components, whether the component is supported
    1. Identify and enumerate software components with complete information (M3)
    2. Identify and enumerate software components with missing information (M4)
3. Compare the date of Input 2 to the current date and capture the timeframe in days (M5)

Measures

- M1 = Output of Operation 1
- M2 = Count of Input 1
- M3 = Count of software components with complete information
- M4 = Count of software components with missing information
- M5 = Timeframe since the last review or update of the inventory

Metrics

- If M1 is 0, this Safeguard receives a failing score. The other metrics don't apply.
- If M5 is greater than twelve months, then this Safeguard is measured at a 0 and receives a failing score. The other metrics don't apply.

Completeness of Inventory

Metric	**The percent of components included in the secure application development process**
Calculation	`M3 / M2`

16.5: Use Up-to-Date and Trusted Third-Party Software Components

Use up-to-date and trusted third-party software components. When possible, choose established and proven frameworks and libraries that provide adequate security. Acquire these components from trusted sources or evaluate the software for vulnerabilities before use.

Asset Type	Security Function	Implementation Groups
Software	Protect	2, 3

Dependencies

- Safeguard 16.4: Establish and Manage an Inventory of Third-Party Software Components

Inputs

1. `GV47`: Inventory of Third-Party Software Components

Operations

1. For each software component in Input 1 `GV47`, determine whether the latest component is being used

    1. Identify and enumerate software components that are up-to-date (M2)
    2. Identify and enumerate software components that are not up-to-date (M3)

2. For each software component identified in Operation 1.1, determine whether they are explicitly trusted by the enterprise

    1. Identify and enumerate software components that are trusted by the enterprise (M4)

Measures

- M1 = Count of Input 1
- M2 = Count of software components that are up-to-date
- M3 = Count of software components that are not up-to-date
- M4 = Count of software components that are up-to-date and trusted

Metrics

Compliance

Metric	The percentage of up-to-date and trusted software components
Calculation	`M4 / M1`

16.6: Establish and Maintain a Severity Rating System and Process for Application Vulnerabilities

Establish and maintain a severity rating system and process for application vulnerabilities that facilitates prioritizing the order in which discovered vulnerabilities are fixed. This process includes setting a minimum level of security acceptability for releasing code or applications. Severity ratings bring a systematic way of triaging vulnerabilities that improves risk management and helps ensure the most severe bugs are fixed first. Review and update the system and process annually.

Asset Type	Security Function	Implementation Groups
Documentation	Govern	2, 3

Dependencies

- Safeguard 16.2: Establish and Maintain a Process to Accept and Address Software Vulnerabilities

Inputs

1. `GV48`: Process to Accept and Address Software Vulnerabilities
2. Date of last update or review of the severity rating system and process

Operations

1. Using Input 1 `GV48` determine whether the enterprise has a severity rating system and process for application vulnerabilities
    1. If the system and process exist, M1 = 1
    2. If the system and process do not exist, M1 = 0
2. Review Input 1 `GV48` and determine whether it includes, at a minimum, the following components: guidance for prioritizing the order vulnerabilities are fixed, level of security acceptability for releasing code or applications
    1. For each component included in the process, assign a value of 1. Sum all values. (M2)
3. Compare Input 2 to the current date and capture the timeframe in months (M3)

Measures

- M1 = Output of Operation 1
- M2 = Count of components included in the process
- M3 = Timeframe in months since last review or update

Metrics

- If M1 is 0, this Safeguard receives a failing score. The other metrics don't apply.
- If M3 is greater than twelve months, then this Safeguard is measured at a 0 and receives a failing score. The other metrics don't apply.

Completeness

Metric	The percentage of components included in the secure application development process
Calculation	`M2 / 2`

16.7: Use Standard Hardening Configuration Templates for Application Infrastructure

Use standard, industry-recommended hardening configuration templates for application infrastructure components. This includes underlying servers, databases, and web servers, and applies to cloud containers, Platform as a Service (PaaS) components, and SaaS components. Do not allow in-house developed software to weaken configuration hardening.

Asset Type	Security Function	Implementation Groups
Software	Protect	2, 3

Dependencies

- Safeguard 4.1: Establish and Maintain a Secure Configuration Process
- Safeguard 4.2: Establish and Maintain a Secure Configuration Process for Network Infrastructure

Inputs

1. `GV1`: Enterprise Asset Inventory
2. `GV37`: Network infrastructure configuration standards

Operations

1. Use Input 1 `GV1` to identify and enumerate application infrastructure components `GV50` (M1)

2. For each infrastructure component identified in Operation 1, check configurations using Input 2 `GV37` and determine if they meet industry-recommended hardening configuration

standards

    1. Identify and enumerate infrastructure components that meet industry standards (M2)
    2. Identify and enumerate infrastructure components that do not meet industry standards (M3)

Measures

- M1 = Count of application infrastructure components
- M2 = Count of components that meet industry standards
- M3 = Count of components that do not meet industry standards

Metrics

Compliance

Metric	The percentage of application infrastructure components that meet industry configuration standards
Calculation	`M2 / M1`

16.8: Separate Production and Non-Production Systems

Maintain separate environments for production and non-production systems.

Asset Type	Security Function	Implementation Groups
Network	Protect	2, 3

Dependencies

- None

Inputs

1. `GV1`: Enterprise Asset Inventory

Operations

1. Use Input 1 `GV1` to identify and enumerate production systems (M1)

2. For each production system identified in Operation 1, use Input 1 `GV1` to identify if at least one non-production system exists for the system

    1. Identify and enumerate production systems with at least one non-production system (M2)

2. Identify and enumerate production systems without a non-production system (M3)

Measures

- M1 = Count of production systems
- M2 = Count of production systems with a non-production system to complement
- M3 = Count of production systems without a non-production system to complement

Metrics

Coverage

Metric	The percentage of non-production systems with an existing production system
Calculation	`M2 / M1`

16.9: Train Developers in Application Security Concepts and Secure Coding

Ensure that all software development personnel receive training in writing secure code for their specific development environment and responsibilities. Training can include general security principles and application security standard practices. Conduct training at least annually and design in a way to promote security within the development team, and build a culture of security among the developers.

Asset Type	Security Function	Implementation Groups
Users	Protect	2, 3

Dependencies

- None

Inputs

1. List of software developing personnel with assigned roles and development environments
2. List of required courses for each role and development environment
3. Date of last training course

Operations

1. For each individual in Input 1, determine whether they have taken the applicable courses per role and environment

    1. Identify and enumerate personnel that have completed the appropriate courses (M2)
    2. Identify and enumerate personnel that have not completed the appropriate courses (M3)

2. For each individual who has completed the appropriate courses, compare the date of the last training from Input 3 to the current date and capture the timeframe in months

    1. Identify and enumerate personnel that have completed all appropriate training within twelve months or less (M4)
    2. Identify and enumerate personnel that have not completed all appropriate training within twelve months or less (M5)

Measures

- M1 = Count of software-developing personnel
- M2 = Count of software developing personnel with completed courses
- M3 = Count of software developing personnel without completed courses
- M4 = Count of software developing personnel with training in scope
- M5 = Count of software developing personnel with training out of scope

Metrics

Compliance

Metric	**The percentage of software development personnel with all appropriate training courses in scope**
Calculation	`M4 / M1`

16.10: Apply Secure Design Principles in Application Architectures

Apply secure design principles in application architectures. Secure design principles include the concept of least privilege and enforcing mediation to validate every operation that the user makes, promoting the concept of "never trusting user input." Examples include ensuring that explicit error checking is performed and documented for all input, including for size, data type, and acceptable ranges or formats. Secure design also means minimizing the application infrastructure attack surface, such as turning off unprotected ports and services, removing unnecessary programs and files, and renaming or removing default accounts.

Asset Type	Security Function	Implementation Groups
Software	Protect	2, 3

Dependencies

- Safeguard 16.1: Establish and Maintain a Secure Application Development Process

Inputs

1. `GV49`: Secure Application Development Process
2. `GV50`: Application Infrastructure Components

Operations

1. Use Input 1 `GV49` to determine whether the process outlines a secure software framework that includes secure design principles

    1. If the framework exists, M1 = 1
    2. If the framework does not exist, M1 = 0

2. For each application infrastructure component in Input 2 `GV50`, determine whether the secure design principles were applied per the framework

    1. Identify and enumerate application infrastructure components where design principles are applied (M3)
    2. Identify and enumerate application infrastructure components where design principles are not applied (M4)

Measures

- M1 = Output of Operation 1
- M2 = Count of Input 2 `GV50`
- M3 = Count of applications infrastructure components with design principles applied
- M4 = Count of applications infrastructure components without design principles applied

Metrics

- If M1 is 0, this Safeguard receives a failing score. The other metrics don't apply.

Compliance

Metric	**The percentage of applications infrastructure components where design principles were applied**
Calculation	`M3 / M2`

16.11: Leverage Vetted Modules or Services for Application Security Components

Leverage vetted modules or services for application security components, such as identity management, encryption, auditing, and logging. Using platform features in critical security functions will reduce developers' workload and minimize the likelihood of design or implementation errors. Modern operating systems provide effective mechanisms for identification, authentication, and authorization and make those mechanisms available to applications. Use only standardized, currently accepted, and extensively reviewed encryption algorithms. Operating systems also provide mechanisms to create and maintain secure audit logs.

Asset Type	Security Function	Implementation Groups
Software	Protect	2, 3

Dependencies

- Safeguard 2.1: Establish and Maintain a Software Inventory

Inputs

1. `GV5`: Authorized Software Inventory

Operations

1. Use Input 1 `GV5` to identify and enumerate application security components (M1)
2. For each application security component identified in Operation 1, determine whether custom code exists
    1. Identify and enumerate components that contain custom code (M2)
    2. Identify and enumerate components that do not contain custom code (M3)
3. For each application security component identified in Operation 2.1, determine whether vetted modules or services exist
    1. Identify and enumerate components for which vetted modules or services exist (M4)
    2. Identify and enumerate components for which vetted modules or services do not exist (M5)

Measures

- M1 = Count of application security components
- M2 = Count of application security components containing custom code
- M3 = Count of application security components not containing custom code
- M4 = Count of application security components containing custom code and vetted modules or services do exist
- M5 = Count of application security components containing custom code and vetted modules or services do not exist

Metrics

Compliance

Metric	**The percentage of application security components using vetted modules or services when available**
Calculation	`(M3 + M5) / M1`

16.12: Implement Code-Level Security Checks

Apply static and dynamic analysis tools within the application life cycle to verify that secure coding practices are being followed.

Asset Type	Security Function	Implementation Groups
Software	Protect	3

Dependencies

- Safeguard 2.1: Establish and Maintain a Software Inventory

Inputs

1. GV5: Authorized Software Inventory

Operations

1. Use Input 1 GV5 to identify and enumerate in-house developed software (M1)
2. Use Input 1 GV5 to identify static analysis tools
3. For each software identified in Operation 1, determine if it is verified by a static tool identified in Operation 2
    1. Identify and enumerate software verified by a static tool (M2)
    2. Identify and enumerate software not verified by a static tool (M3)
4. Use Input 1 GV5 to identify dynamic analysis tools
5. For each software identified in Operation 1, determine if it is verified by a dynamic tool identified in Operation 4
    1. Identify and enumerate software verified by a dynamic tool (M4)
    2. Identify and enumerate software not verified by a dynamic tool (M5)

Measures

- M1 = Count of in-house developed software
- M2 = Count of in-house developed software verified by a static analysis tool
- M3 = Count of in-house developed software not verified by a static analysis tool
- M4 = Count of in-house developed software verified by a dynamic analysis tool
- M5 = Count of in-house developed software not verified by a dynamic analysis tool

Metrics

Static Analysis Tool Coverage

Metric	**The percentage of in-house developed software verified by a static analysis tool**
Calculation	`M2 / M1`

Dynamic Analysis Tool Coverage

Metric	**The percentage of in-house developed software verified by a dynamic analysis tool**
Calculation	`M4 / M1`

16.13: Conduct Application Penetration Testing

Conduct application penetration testing. For critical applications, authenticated penetration testing is better suited to finding business logic vulnerabilities than code scanning and automated security testing. Penetration testing relies on the skill of the tester to manually manipulate an application as an authenticated and unauthenticated user.

Asset Type	Security Function	Implementation Groups
Software	Detect	3

Dependencies

- Safeguard 2.1: Establish and Maintain a Software Inventory

Inputs

1. `GV5`: Authorized Software Inventory
2. Application Penetration Process for enterprise

Operations

1. Determine whether Input 2 exists for the enterprise

    1. If the process exists, M1 = 1
    2. If the process does not exist, M1 = 0

2. Use Input 1 `GV5` to identify and enumerate all applications within the enterprise (M2)

3. For each application identified in Operation 2, determine whether an unauthenticated penetration test has been conducted per the process outlined in Input 2

    1. Identify and enumerate applications that have been tested (M3)
    2. Identify and enumerate applications that have not been tested (M4)

4. Use the output of Operation 2 to identify and enumerate critical applications within the list of applications (M5)

5. For each application identified in Operation 4, determine whether an authenticated penetration test has been conducted per the process outlined in Input 2

    1. Identify and enumerate applications that have been tested (M6)
    2. Identify and enumerate applications that have not been tested (M7)

Measures

- M1 = Output of Operation 1
- M2 = Count of applications within the enterprise
- M3 = Count of applications that have undergone unauthenticated penetration testing per enterprise's process
- M4 = Count of applications that have not undergone unauthenticated penetration testing per enterprise's process
- M5 = Count of critical applications
- M6 = Count of critical applications that have undergone authenticated penetration testing per enterprise's process
- M7 = Count of critical applications that have not undergone authenticated penetration testing per enterprise's process

Metrics

- If M1 is 0, this Safeguard receives a failing score. The other metrics don't apply.

Unauthenticated Penetration Testing Coverage

Metric	**The percentage of applications that underwent unauthenticated penetration testing per enterprise's process**
Calculation	`M3 / M2`

Authenticated Penetration Testing Coverage

Metric	**The percentage of critical applications that underwent authenticated penetration testing per enterprise's process**
Calculation	`M6 / M5`

16.14: Conduct Threat Modeling

Conduct threat modeling. Threat modeling is the process of identifying and addressing application security design flaws within a design, before code is created. It is conducted through specially trained individuals who evaluate the application design and gauge security risks for each entry point and access level. The goal is to map out the application, architecture, and infrastructure in a structured way to understand its weaknesses.

Asset Type	Security Function	Implementation Groups
Software	Protect	3

Dependencies

- Safeguard 2.1: Establish and Maintain a Software Inventory

Inputs

1. `GV5`: Authorized Software Inventory
2. Threat Modeling Process for the Enterprise

Operations

1. Determine whether Input 2 exists for the enterprise
    1. If the process exists, M1 = 1
    2. If the process does not exist, M1 = 0
2. Use Input 1 `GV5` to identify and enumerate all in-house developed applications (M2)
3. For each application identified in Operation 2, determine whether the threat modeling process was followed
    1. Identify and enumerate applications for which threat modeling was conducted (M3)
    2. Identify and enumerate applications for which threat modeling was not conducted (M4)

Measures

- M1 = Output of Operation 1
- M2 = Count of in-house developed applications
- M3 = Count of in-house developed applications that underwent threat modeling
- M4 = Count of in-house developed applications that did not undergo threat modeling

Metrics

- If M1 is 0, this Safeguard receives a failing score. The other metrics don't apply.

Compliance

Metric	**The percentage of in-house developed applications that underwent threat modeling**
Calculation	`M3 / M2`