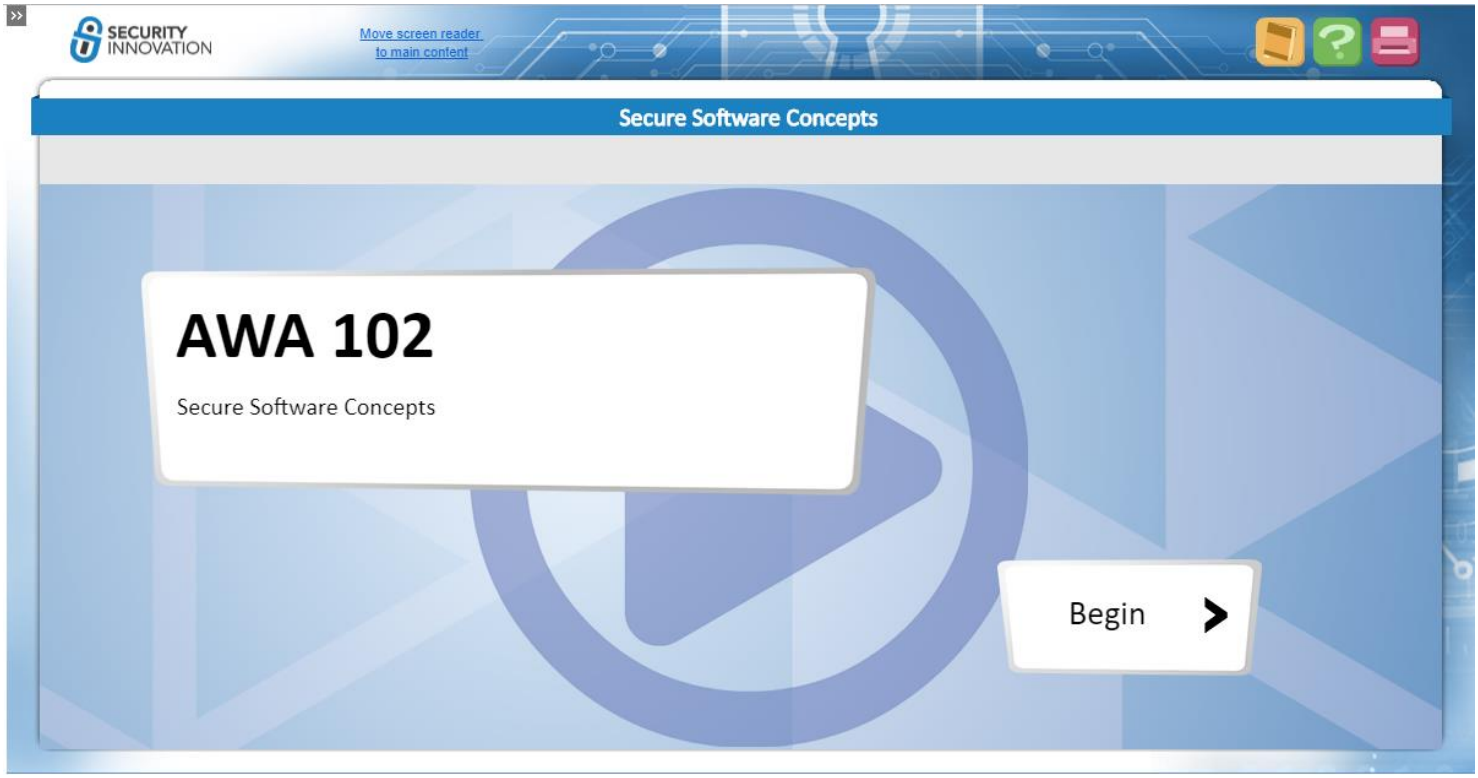



**Narration**

**On Screen Text**

**AWA 102**
Secure Software Concepts

## Course Overview and Objectives


### Narration

This course provides a high-level overview of secure software concepts for applications, including application security, security standards, secure development methodologies, and security best practices.

When you have completed this course, you will be able to describe the current threat landscape and identify several common security vulnerabilities. You will also be able to list several resources for evaluating and mitigating the most common application security risks.

You will be able to identify security-related tasks for each stage in a secure software development lifecycle, and list resources for implementing a security strategy based on your organization's actual risk profile, and leveraging other organization's experiences with secure development practices. Finally, you will be able to describe how to apply several security best practices to harden your security stance.

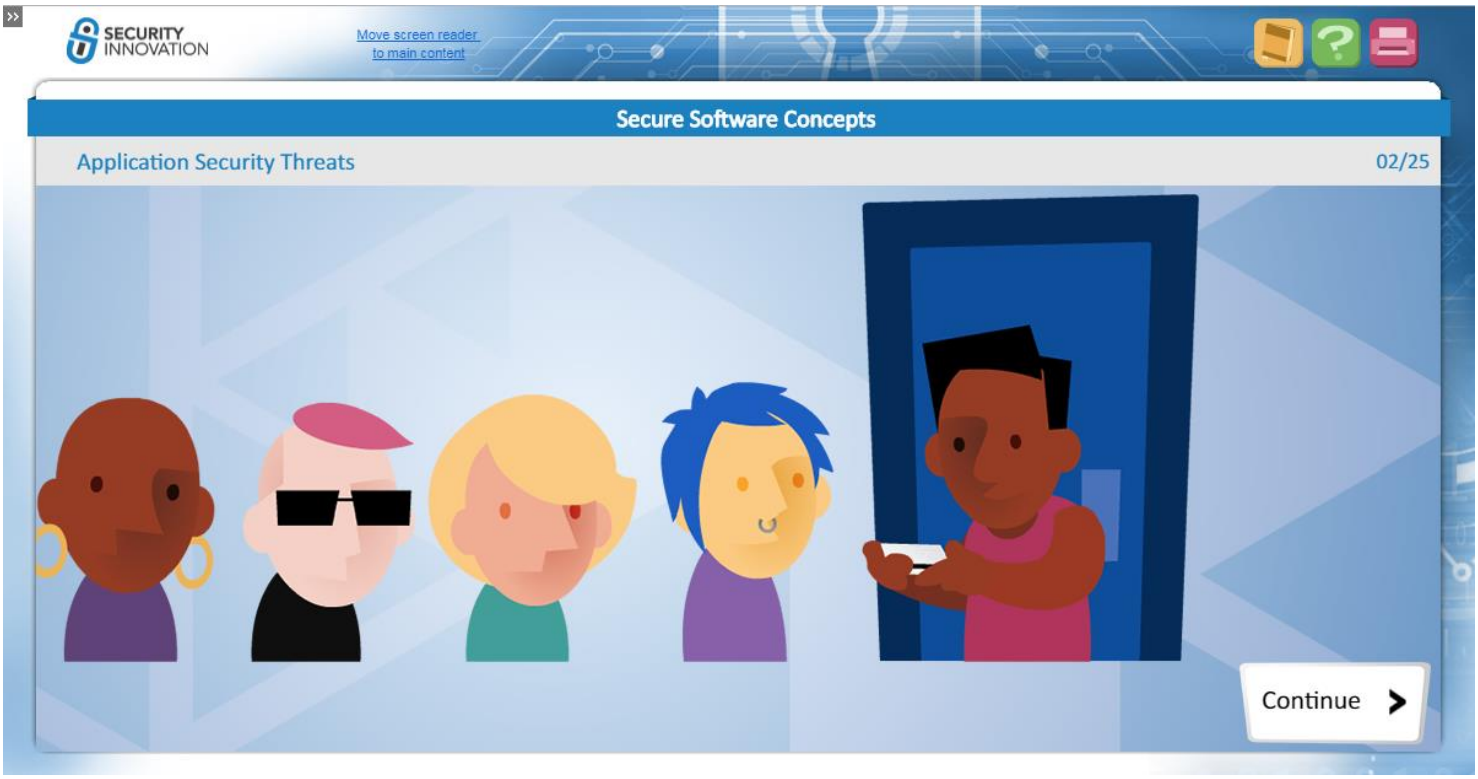
### On Screen Text

Course Overview and Objectives

**When you have completed this course, you will be able to**:

- Describe the current threat landscape
- Identify several common security vulnerabilities
- List several resources for evaluating and mitigating common security risks
- Identify security-related tasks for each stage in a secure software development lifecycle
- List resources for implementing a security strategy
- Describe how to apply several security best practices

## Application Security Threats




### Narration

Despite our greatest efforts to address the most common security threats, we now face increasingly sophisticated and targeted attacks. Even with our own applications secure, there is an additional attack surface from third-party libraries and external cloud services.

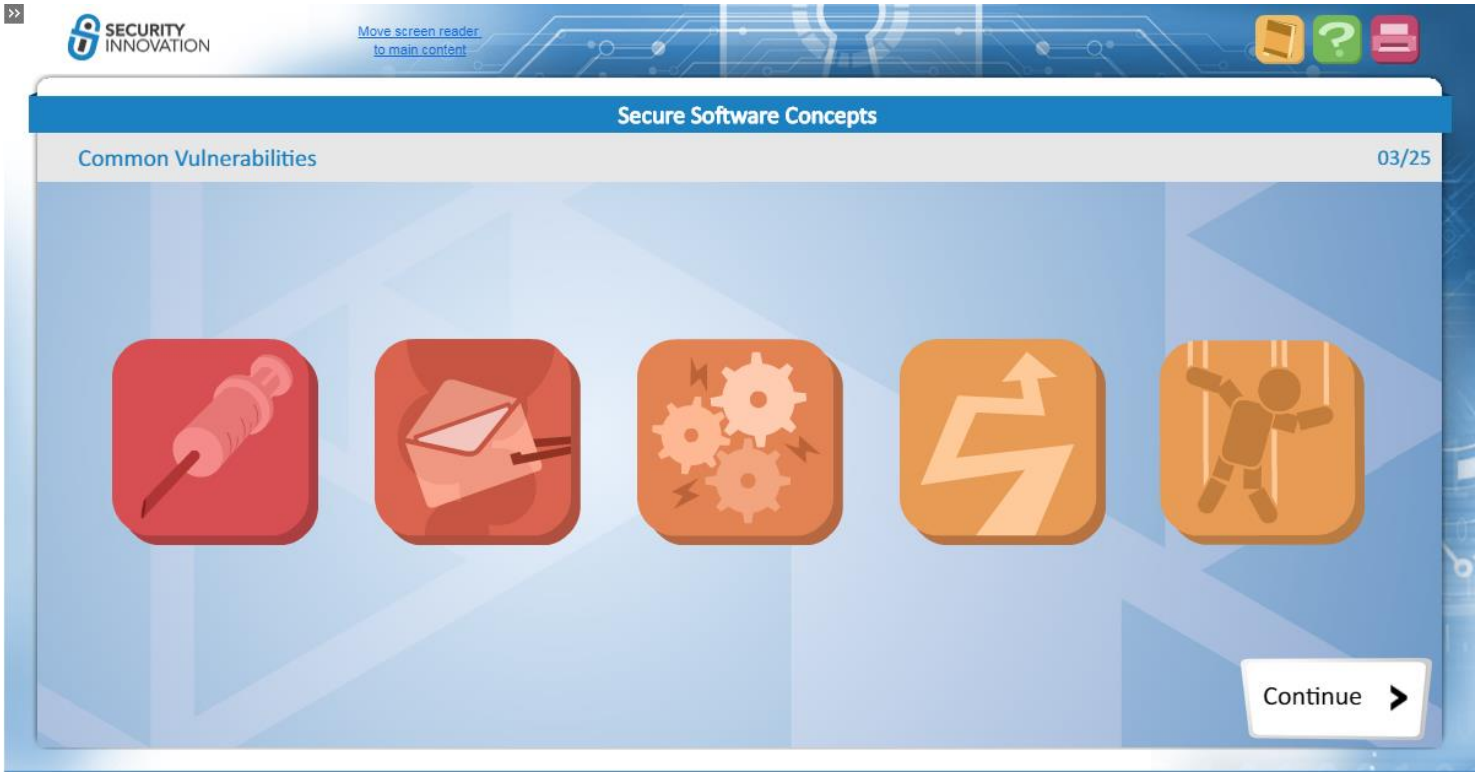
The current threat landscape not only includes individual hackers, but also coordinated attacks from state-sponsored groups, organized crime, botnets, ransomware, and distributed attacks from Internet of Things (IoT) devices and mobile malware.

Furthermore, there is increasing pressure from regulatory bodies, potential litigation, and greater customer expectations, all demanding the highest security.

### On Screen Text

Application Security Threats

## Common Vulnerabilities




### Narration

Software security vulnerabilities are code defects that are exploitable and reproducible. Software security is the process of identifying and eliminating these defects.

Some of the most common vulnerabilities found in applications are:

Injection vulnerabilities – Failing to properly filter user input, thus leading to injection of arbitrary database, shell, or other commands.

Exposing sensitive data – Failing to properly encrypt or otherwise secure sensitive system and customer information.

Misconfigured security settings – Failing to properly configure a system or platform with the most secure options available.

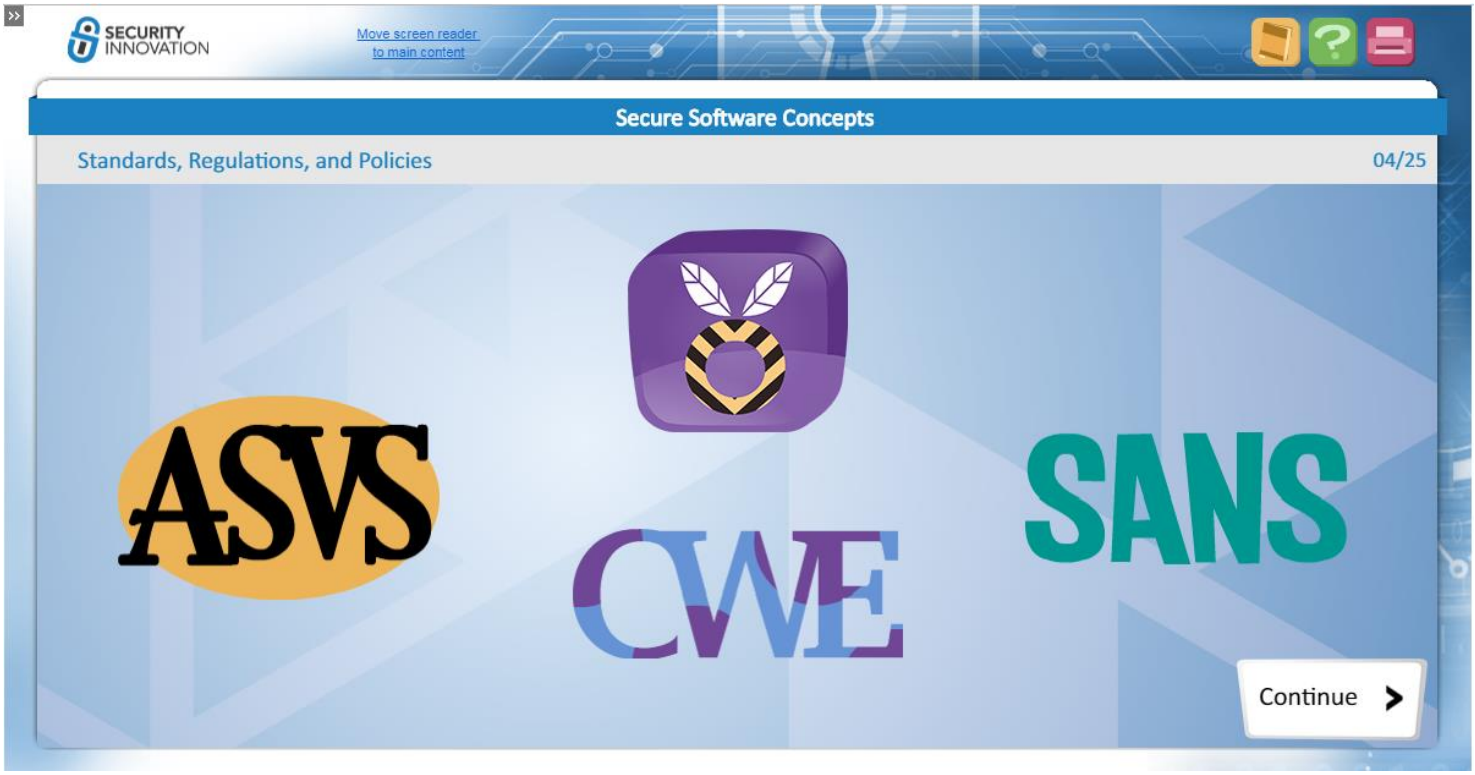
Cross-site scripting - Failure to encode untrusted data for the correct output context.

And, cross-site request forgery – Failing to properly handle web requests to ensure users did intend to make sensitive changes or perform sensitive actions.

### On Screen Text

Common Vulnerabilities

### Standards, Regulations, and Policies




#### Narration

Whether it be healthcare, finance, e-commerce, or government entities, many organizations fall under the domain of one or more regulatory bodies. As custodians of sensitive user data, standards and regulatory bodies ensure organizations comply with minimum standards and practices.

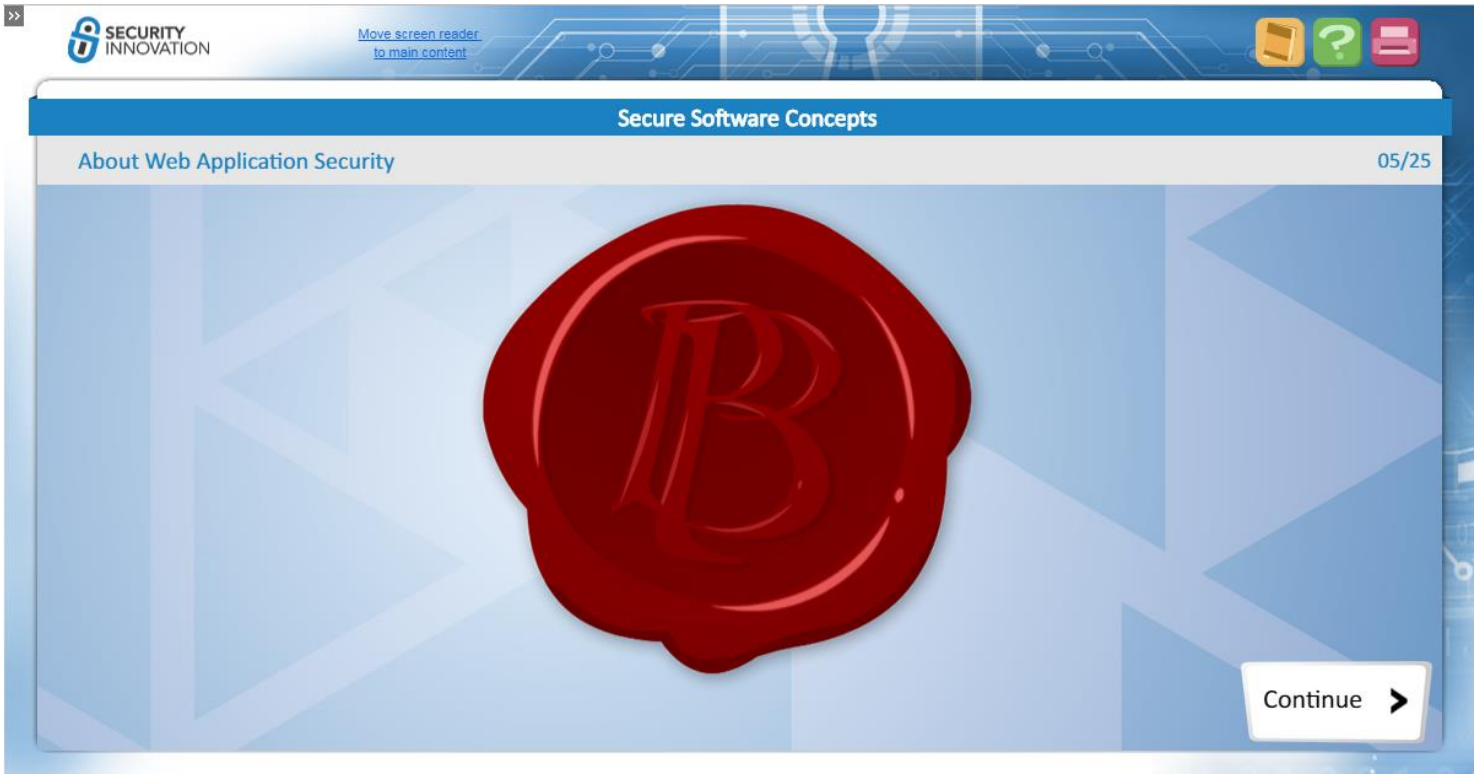
Even without external regulations, it is important to implement widely-accepted application security guidelines in your organization. Secure coding best practices recommended by the Open Web Application Security Project (OWASP) Top 10, OWASP Application Security Verification Standard (ASVS), and the Common Weakness Enumeration (CWE)/SysAdmin, Audit, Network, Security (SANS) Institute Top 25 all provide guidance for the most common security threats.

As you design and build web applications, it is vital that you integrate adherence to security policies, standards, and regulations into the entire development lifecycle.

#### On Screen Text

## Standards, Regulations, and Policies

## About Web Application Security




### Narration

As more organizations move their operations online and increasingly rely on web-based applications and services, the urgency of web application security becomes increasingly apparent. For those charged with managing and mitigating the risks of web applications, it can be a considerable task to balance the business, legal, and financial requirements of the organization.

Web applications are a primary target for hackers because they are the most visible targets and most likely to contain serious vulnerabilities; they are typically connected to database and other back-end servers that could be a path into the internal network; and, attacks can be easily automated to identify common weaknesses.

Application security includes designing software on a solid foundation of standards and best practices, using structured development frameworks, and implementing secure development methodologies. The failure to secure web applications can quickly escalate to a breach of an organization’s entire network.

### On Screen Text

About Web Application Security

### OWASP Top 10


#### Narration

The Open Web Application Security Project (OWASP) Top 10 list describes the most common web application attacks and vulnerabilities, along with example code vulnerabilities, example attacks, and guidance on mitigating risks.

The list is compiled based on a consensus of information security professionals and by analyzing actual attack data. Each item on the Top 10 list includes analysis of the risks, and the exploitability, prevalence, and detectability of the attack. The guidance also includes example scenarios and steps to prevent exposure to the vulnerability.

OWASP updates the Top 10 list every few years to help you focus on vulnerabilities that pose the most current risks. However, as with any security model, standard, or guideline, following the OWASP Top 10 does not necessarily eliminate all possible risks.

#### On Screen Text

## OWASP Top 10

1. Injection

2. Broken Authentication + Session Management

3. Sensitive Data Exposure

4. XML External Entities

5. Broken Access Control

6. Security Misconfiguration

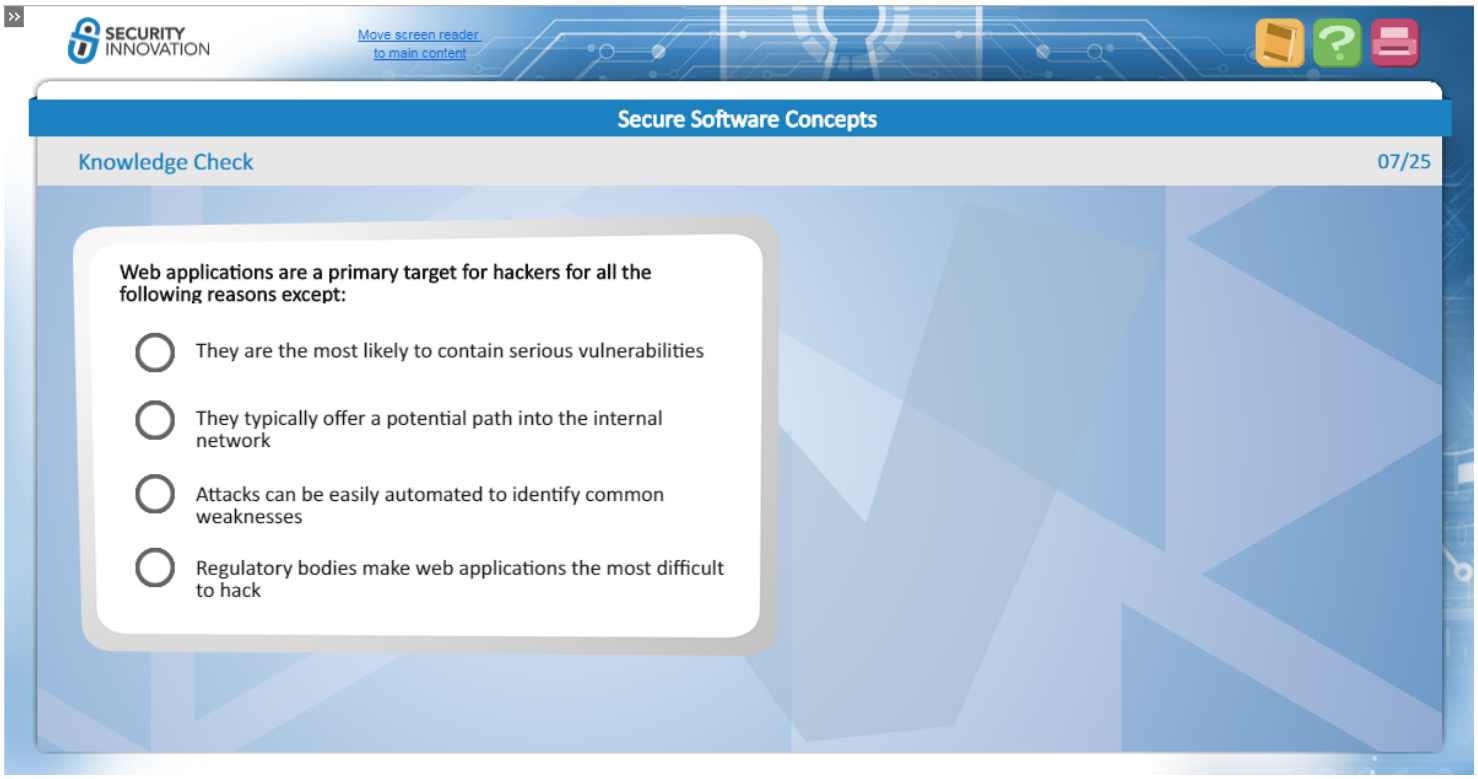
7. Cross Site Scripting

8. Insecure Deserialization

9. Using Components with Known Vulnerabilities

10. Insufficient Logging + Monitoring

### Knowledge Check




### Narration

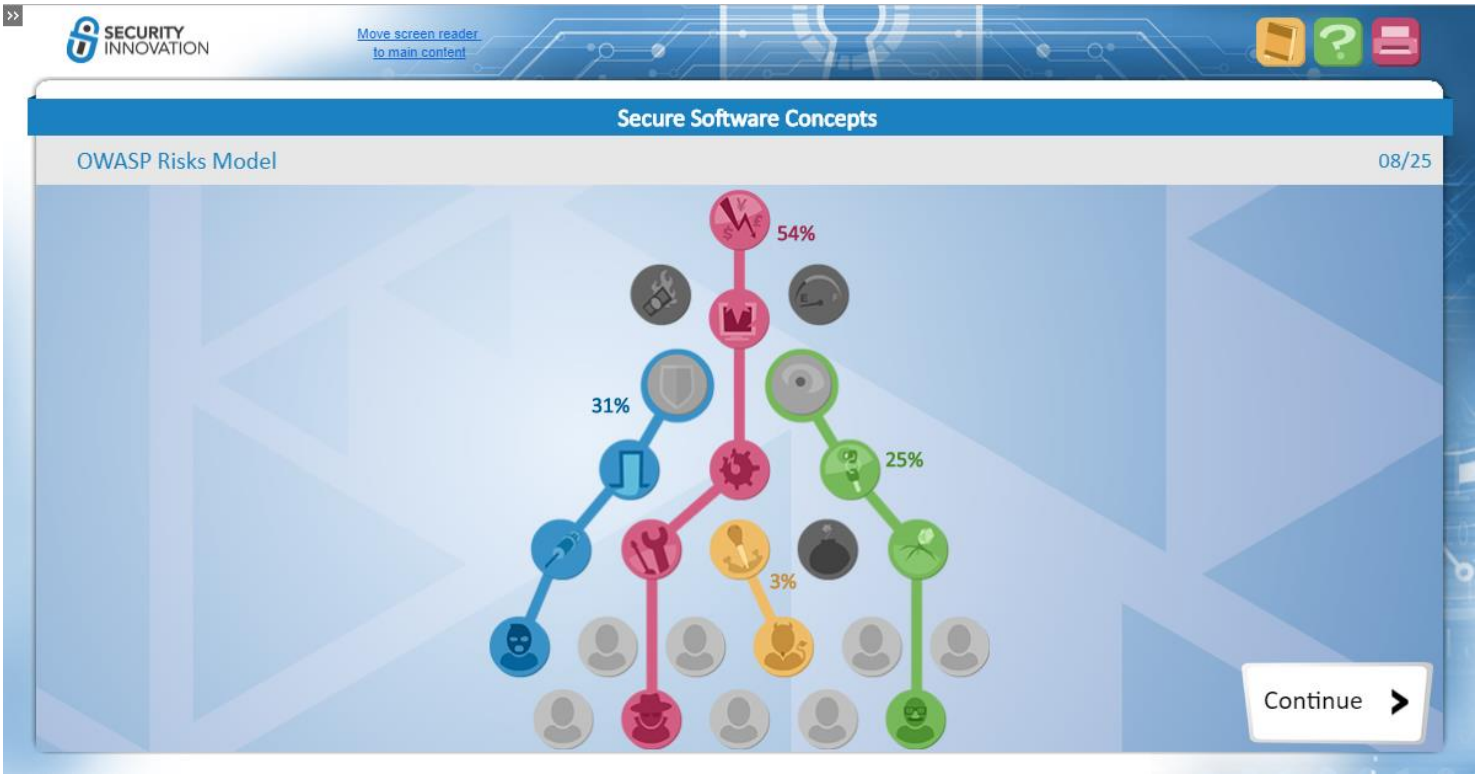
### On Screen Text

## Knowledge Check

**Web applications are a primary target for hackers for all the following reasons except:**

They are the most likely to contain serious vulnerabilities
They typically offer a potential path into the internal network
Attacks can be easily automated to identify common weaknesses
Regulatory bodies make web applications the most difficult to hack

## OWASP Risks Model




### Narration

For the OWASP Top 10, risk is a measurement of the likelihood that any particular attack path—from threat agent to security controls—will have a significant impact on your organization.

There may be many different attackers, attack vectors, security weaknesses, and security controls that exist in any web application. Any number of attackers—or threat agents—might employ different attack vectors looking for security weaknesses. A single security weakness coupled with insufficient security controls might lead to a security compromise. A security compromise could have a serious impact on both technical and business operations.

To determine risk, an organization must understand who the threat agents are and take into consideration the attack vectors, security weaknesses, and associated security controls for an application.

Finally, you must assess the likelihood of a particular compromise having a significant impact on your technical or business operations. Throughout the OWASP Top 10 guide, you will see risk ratings for each vulnerability.

### On Screen Text

OWASP Risks Model

**31%**

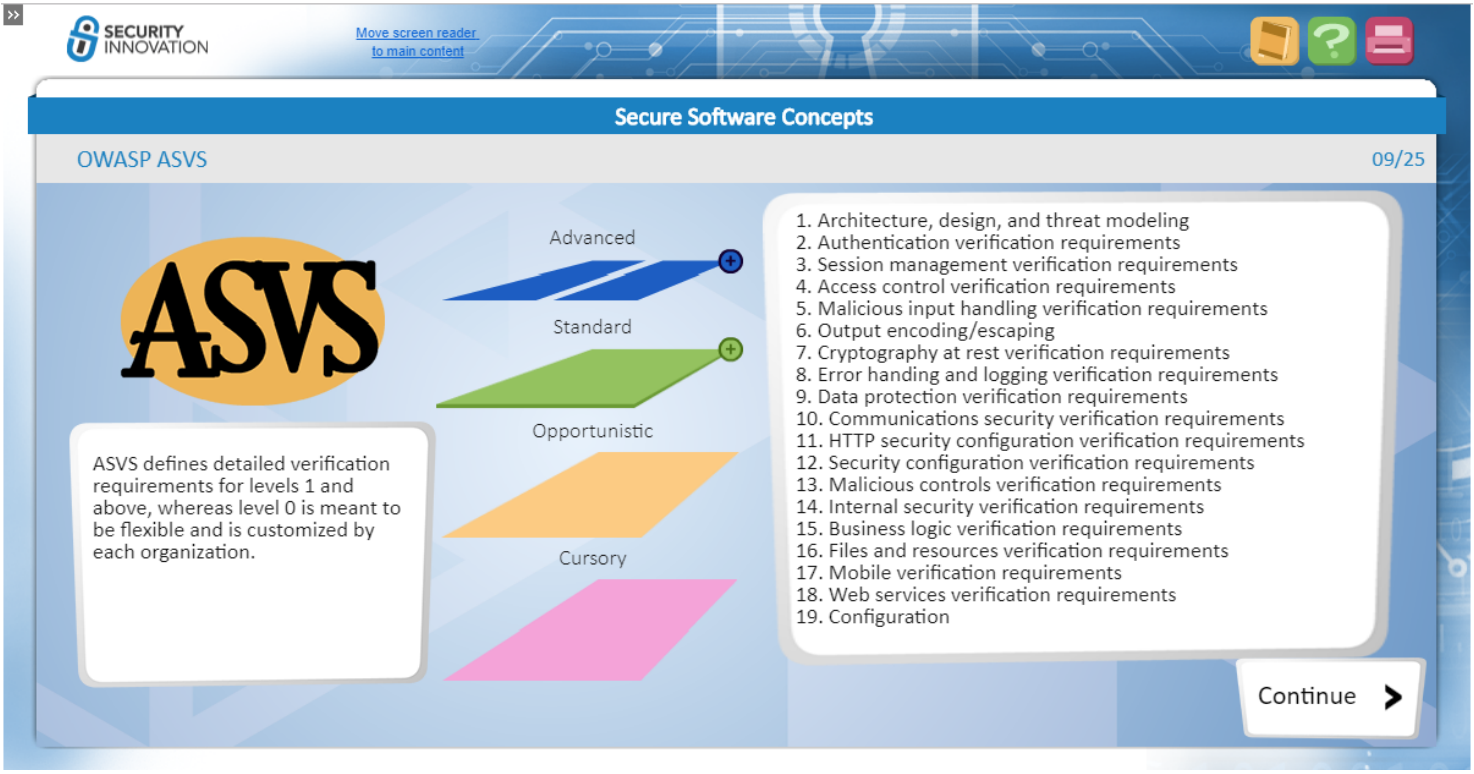
## Secure Software Concepts

**54%**

**25%**

**3%**

## OWASP ASVS




### Narration

The OWASP Application Security Verification Standard (ASVS) project provides a standard for testing web application security, with a list of verification requirements in nineteen key areas. Whereas the OWASP Top 10 provides a focused list of the most common security issues, the ASVS provides a comprehensive guide for addressing most common application security risks.

ASVS defines four levels of verification. Each level provides for a more in-depth set of requirements than the previous level. Applications that meet these requirements are considered an ASVS Level N application, where N is the level of compliance met. You can use the ASVS standard as both guidance and as a metric for measuring and comparing the security of an application.

### On Screen Text

## OWASP ASVS

ASVS defines detailed verification requirements for levels 1 and above, whereas level 0 is meant to be flexible and is customized by each organization.

Advanced

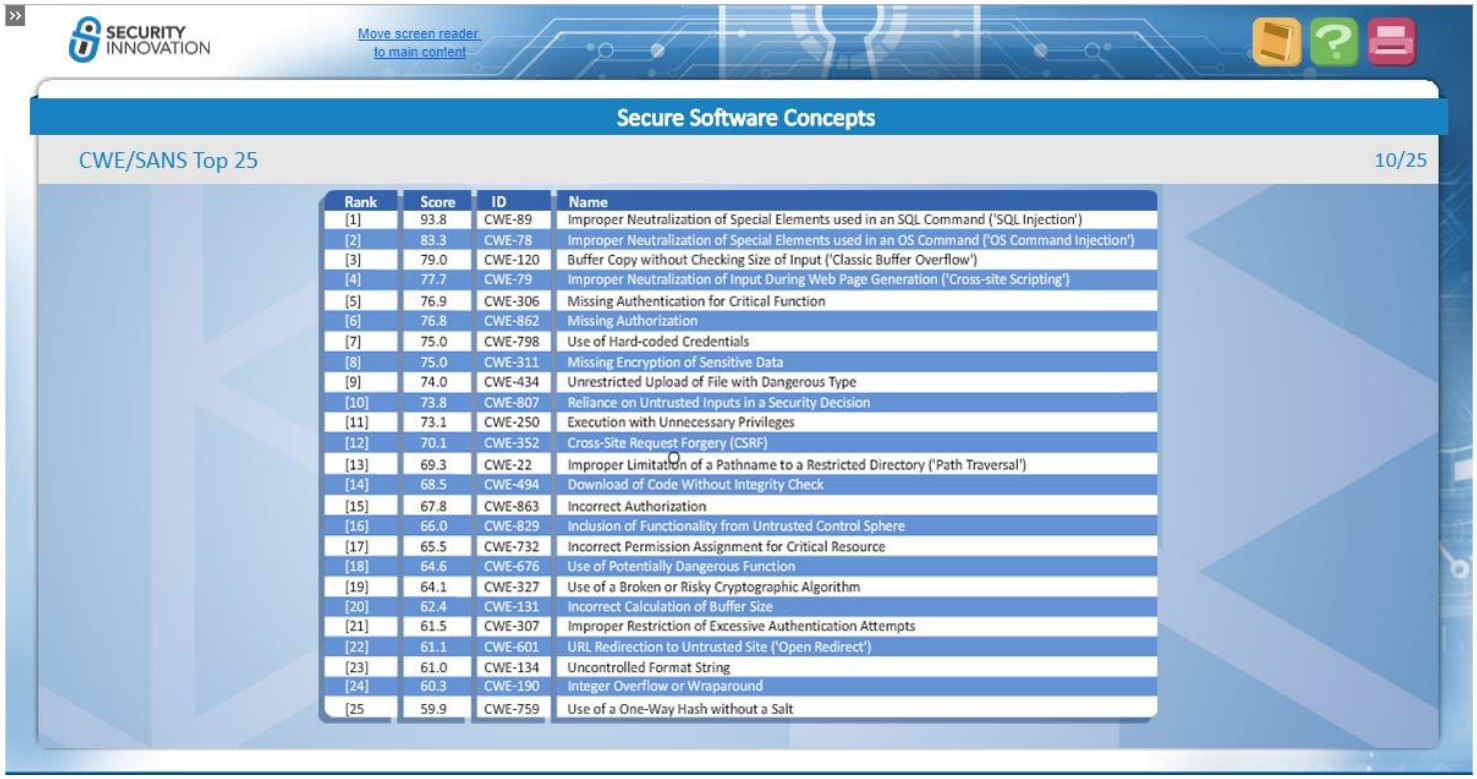
Standard

Opportunistic

Cursory

1. Architecture, design, and threat modeling
2. Authentication verification requirements
3. Session management verification requirements
4. Access control verification requirements
5. Malicious input handling verification requirements
6. Output encoding/escaping
7. Cryptography at rest verification requirements
8. Error handing and logging verification requirements
9. Data protection verification requirements
10. Communications security verification requirements
11. HTTP security configuration verification requirements
12. Security configuration verification requirements
13. Malicious controls verification requirements
14. Internal security verification requirements
15. Business logic verification requirements
16. Files and resources verification requirements
17. Mobile verification requirements
18. Web services verification requirements
19. Configuration

## CWE/SANS Top 25




| Rank | Score | ID | Name |
|---|---|---|---|
| [1] | 93.8 | CWE-89 | Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') |
| [2] | 83.3 | CWE-78 | Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') |
| [3] | 79.0 | CWE-120 | Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') |
| [4] | 77.7 | CWE-79 | Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') |
| [5] | 76.9 | CWE-306 | Missing Authentication for Critical Function |
| [6] | 76.8 | CWE-862 | Missing Authorization |
| [7] | 75.0 | CWE-798 | Use of Hard-coded Credentials |
| [8] | 75.0 | CWE-311 | Missing Encryption of Sensitive Data |
| [9] | 74.0 | CWE-434 | Unrestricted Upload of File with Dangerous Type |
| [10] | 73.8 | CWE-807 | Reliance on Untrusted Inputs in a Security Decision |
| [11] | 73.1 | CWE-250 | Execution with Unnecessary Privileges |
| [12] | 70.1 | CWE-352 | Cross-Site Request Forgery (CSRF) |
| [13] | 69.3 | CWE-22 | Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') |
| [14] | 68.5 | CWE-494 | Download of Code Without Integrity Check |
| [15] | 67.8 | CWE-863 | Incorrect Authorization |
| [16] | 66.0 | CWE-829 | Inclusion of Functionality from Untrusted Control Sphere |
| [17] | 65.5 | CWE-732 | Incorrect Permission Assignment for Critical Resource |
| [18] | 64.6 | CWE-676 | Use of Potentially Dangerous Function |
| [19] | 64.1 | CWE-327 | Use of a Broken or Risky Cryptographic Algorithm |
| [20] | 62.4 | CWE-131 | Incorrect Calculation of Buffer Size |
| [21] | 61.5 | CWE-307 | Improper Restriction of Excessive Authentication Attempts |
| [22] | 61.1 | CWE-601 | URL Redirection to Untrusted Site ('Open Redirect') |
| [23] | 61.0 | CWE-134 | Uncontrolled Format String |
| [24] | 60.3 | CWE-190 | Integer Overflow or Wraparound |
| [25 | 59.9 | CWE-759 | Use of a One-Way Hash without a Salt |

### Narration

The MITRE Common Weakness Enumeration (CWE) group and SysAdmin, Audit, Network, Security (SANS) Institute published a report on the top 25 most dangerous software errors. The list is intended to help developers understand, recognize, and prevent common security coding errors that could lead to serious vulnerabilities in their applications.

Each of the CWE/SANS Top 25 errors is accompanied by a wealth of information such as remediation cost, attack frequency, consequences, and mitigation techniques.

Although the last joint report was published in 2011, nearly all the weaknesses cited are still relevant today. Additionally, the community-developed formal list of over 700 common weaknesses continues to be updated as an ongoing effort sponsored by the U.S. Department of Homeland Security.

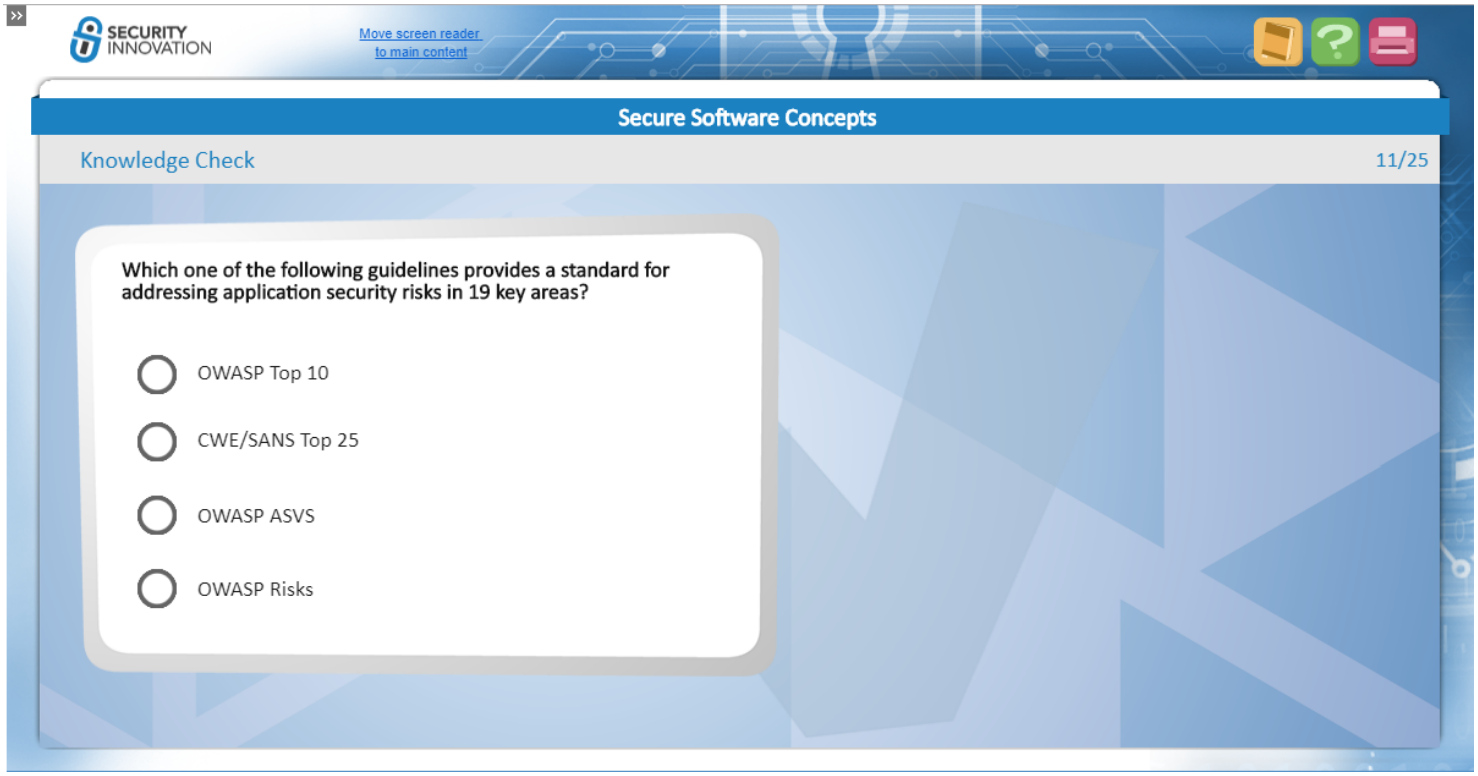
### On Screen Text

CWE/SANS Top 25

| Rank | Score | ID | Name |
|---|---|---|---|
| **[1]** | 93.8 | CWE-89 | Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') |
| **[2]** | 83.3 | CWE-78 | Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') |
| **[3]** | 79.0 | CWE-120 | Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') |
| **[4]** | 77.7 | CWE-79 | Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') |
| **[5]** | 76.9 | CWE-306 | Missing Authentication for Critical Function |
| **[6]** | 76.8 | CWE-862 | Missing Authorization |
| **[7]** | 75.0 | CWE-798 | Use of Hard-coded Credentials |
| **[8]** | 75.0 | CWE-311 | Missing Encryption of Sensitive Data |
| **[9]** | 74.0 | CWE-434 | Unrestricted Upload of File with Dangerous Type |
| **[10]** | 73.8 | CWE-807 | Reliance on Untrusted Inputs in a Security Decision |
| **[11]** | 73.1 | CWE-250 | Execution with Unnecessary Privileges |
| **[12]** | 70.1 | CWE-352 | Cross-Site Request Forgery (CSRF) |
| **[13]** | 69.3 | CWE-22 | Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') |
| **[14]** | 68.5 | CWE-494 | Download of Code Without Integrity Check |
| **[15]** | 67.8 | CWE-863 | Incorrect Authorization |
| **[16]** | 66.0 | CWE-829 | Inclusion of Functionality from Untrusted Control Sphere |
| **[17]** | 65.5 | CWE-732 | Incorrect Permission Assignment for Critical Resource |
| **[18]** | 64.6 | CWE-676 | Use of Potentially Dangerous Function |
| **[19]** | 64.1 | CWE-327 | Use of a Broken or Risky Cryptographic Algorithm |
| **[20]** | 62.4 | CWE-131 | Incorrect Calculation of Buffer Size |
| **[21]** | 61.5 | CWE-307 | Improper Restriction of Excessive Authentication Attempts |
| **[22]** | 61.1 | CWE-601 | URL Redirection to Untrusted Site ('Open Redirect') |
| **[23]** | 61.0 | CWE-134 | Uncontrolled Format String |

| Rank | Score | ID | Name |
|---|---|---|---|
| **[24]** | 60.3 | CWE-190 | Integer Overflow or Wraparound |
| **[25]** | 59.9 | CWE-759 | Use of a One-Way Hash without a Salt |

## Knowledge Check




### Narration

### On Screen Text

Knowledge Check

**Which one of the following guidelines provides a standard for addressing application security risks in 19 key areas?**

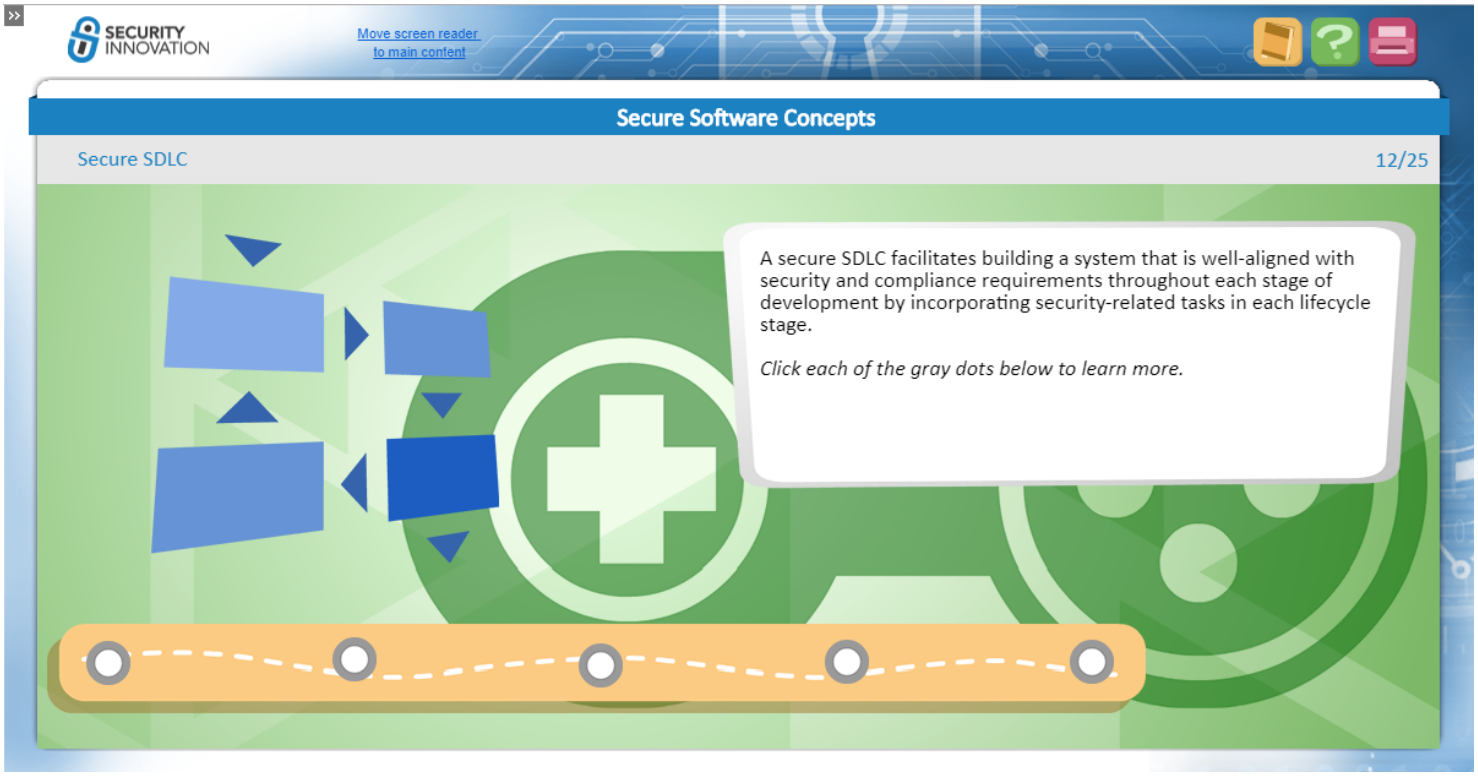
OWASP Top 10

CWE/SANS Top 25

OWASP ASVS

OWASP Risks

## Secure SDLC




### Narration

A secure software development lifecycle—or SDLC—ensures that proper security controls and safeguards are an intrinsic part of an application's development. By treating security as a design requirement, it becomes ingrained into every component of the system.

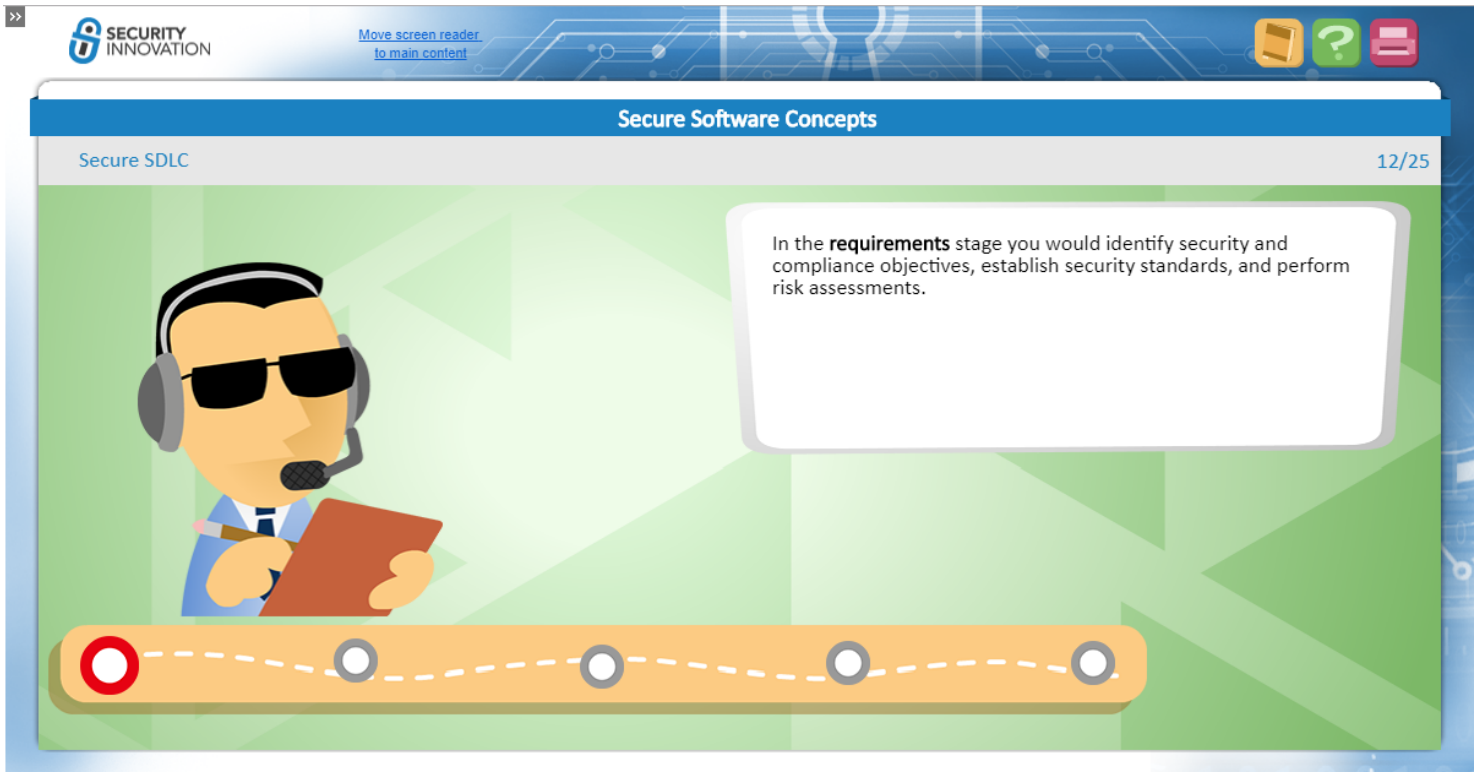
An SDLC facilitates building a system that is well-aligned with security and compliance requirements throughout each stage of development by incorporating security-related tasks in each lifecycle stage. Let's look at an example.

### On Screen Text

## Secure SDLC

A secure SDLC facilitates building a system that is well-aligned with security and compliance requirements throughout each stage of development by incorporating security-related tasks in each lifecycle stage.
*Click each of the gray dots below to learn more.*

## Secure SDLC




### Narration

In the requirements stage you would identify security and compliance objectives, establish security standards, and perform risk assessments.

### On Screen Text

Secure SDLC

In the **requirements** stage you would identify security and compliance objectives, establish security standards, and perform risk assessments.

## Secure SDLC


### Narration

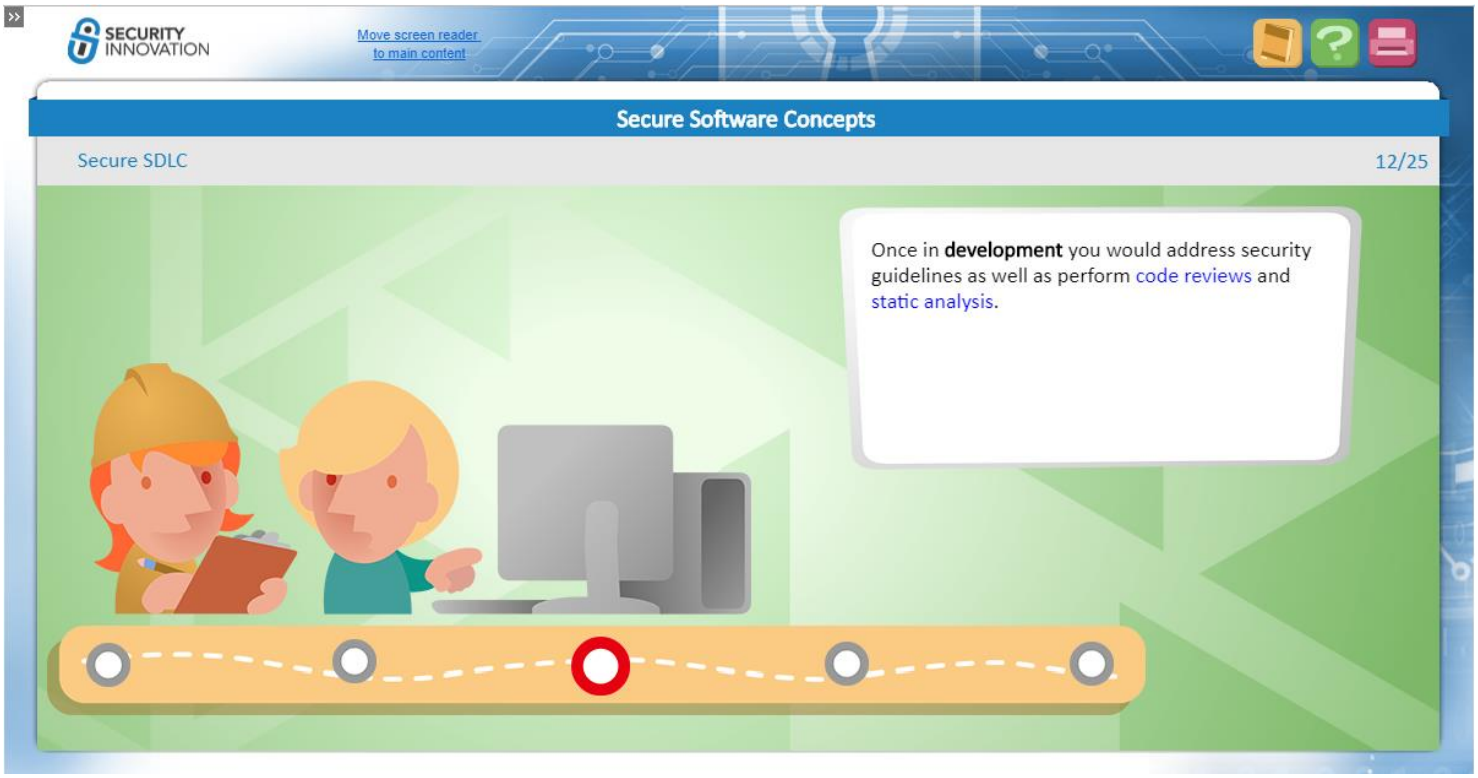
During design you would analyze the application's attack surface, perform threat modeling, and document the application's security architecture.

### On Screen Text

Secure SDLC

During **design** you would analyze the application's attack surface, perform threat modeling, and document the application's security architecture.

### Secure SDLC




### Narration

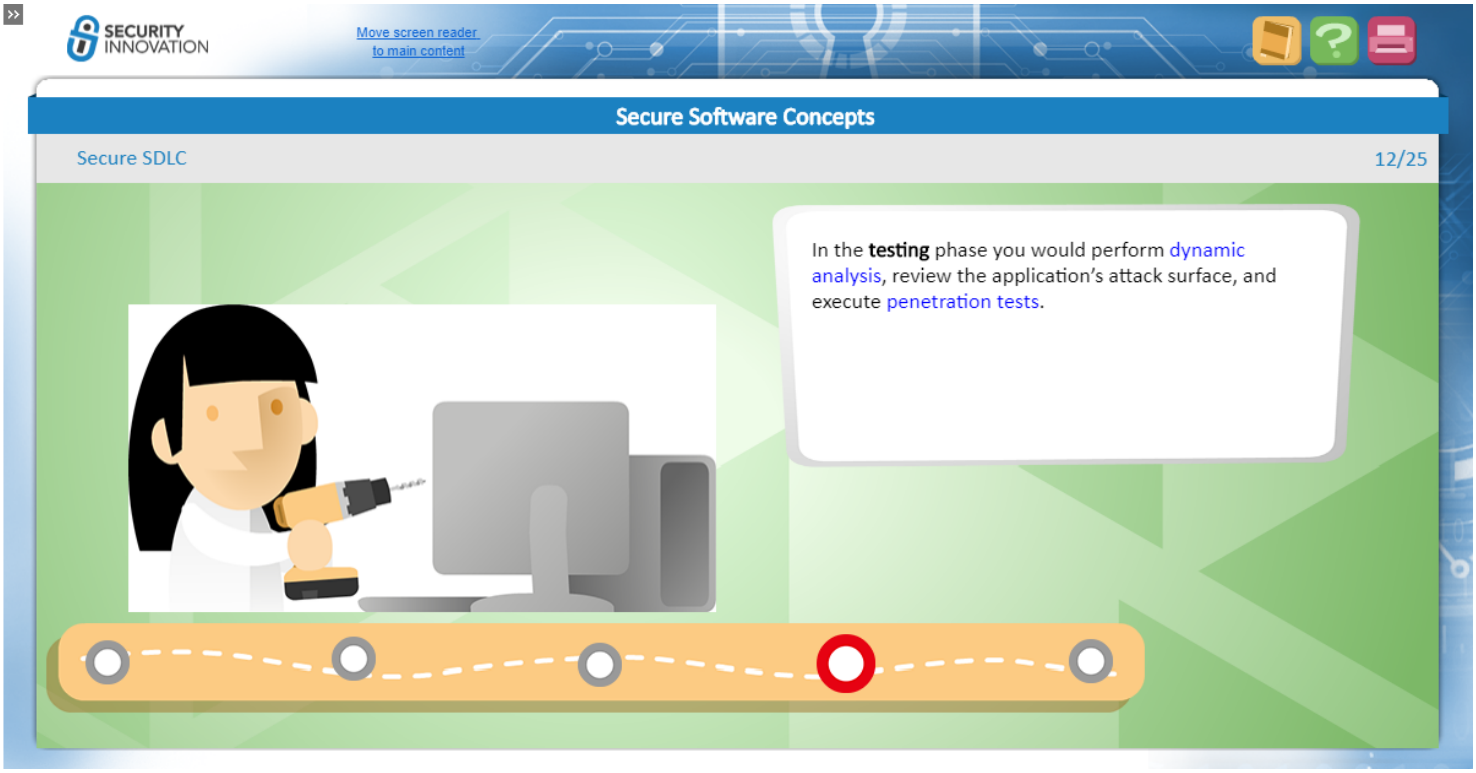
Once in development you would address security guidelines as well as perform code reviews and static analysis.

### On Screen Text

## Secure SDLC

Once in **development** you would address security guidelines as well as perform code reviews and static analysis.

## Secure SDLC




### Narration

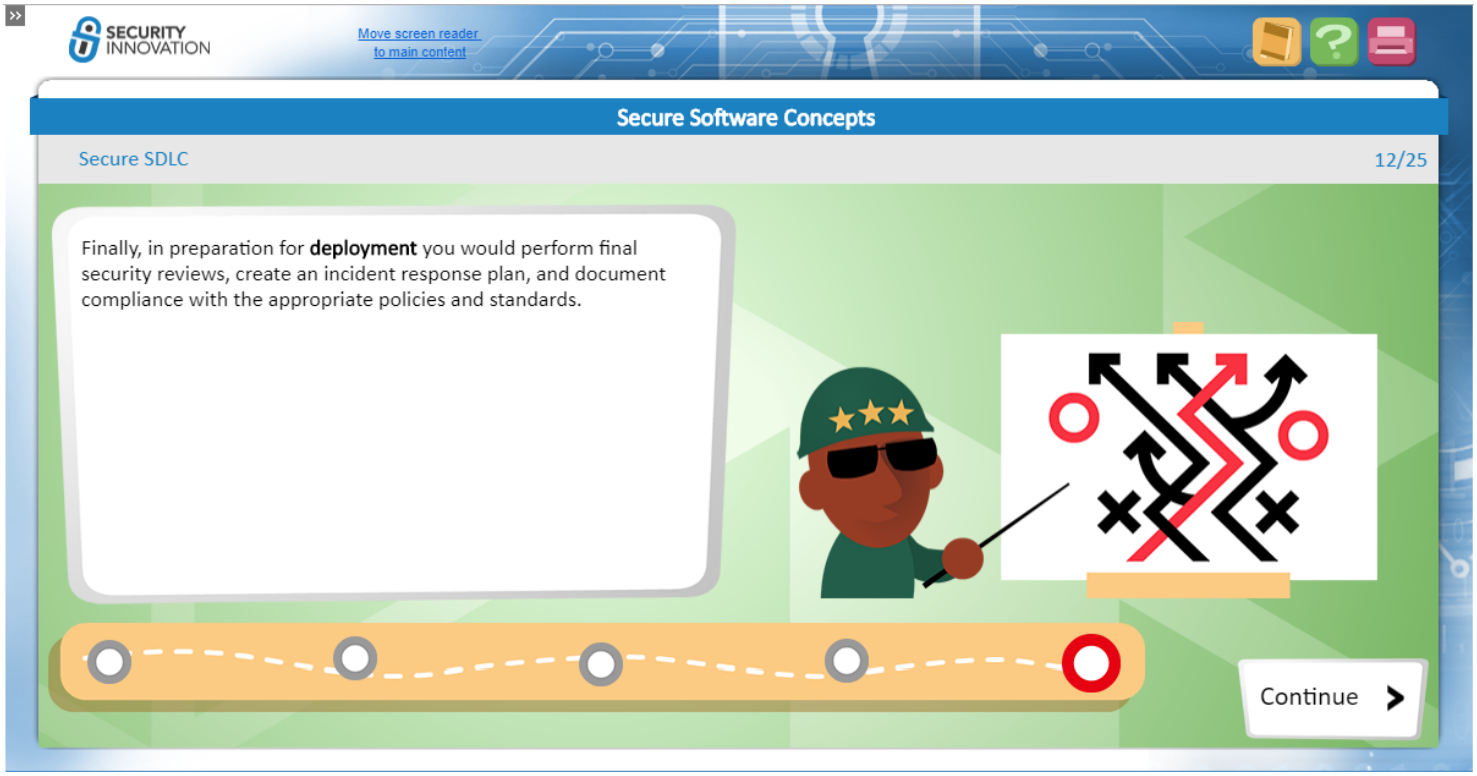
In the testing phase you would perform dynamic analysis, review the application's attack surface, and execute penetration tests.

### On Screen Text

Secure SDLC

In the **testing** phase you would perform dynamic analysis, review the application's attack surface, and execute penetration tests.

### Secure SDLC




### Narration

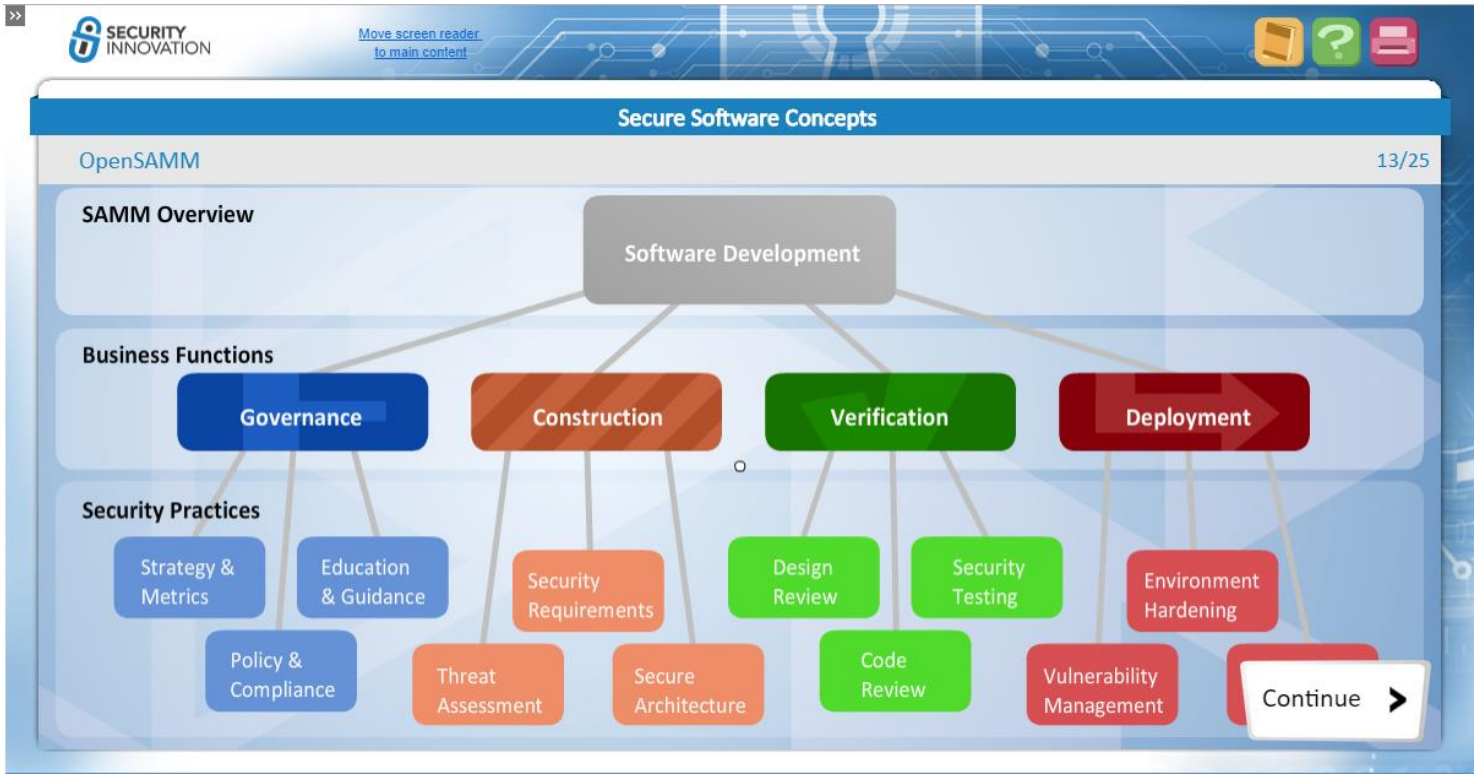
Finally, in preparation for deployment you would perform final security reviews, create an incident response plan, and document compliance with the appropriate policies and standards.

### On Screen Text

## Secure SDLC

Finally, in preparation for **deployment** you would perform final security reviews, create an incident response plan, and document compliance with the appropriate policies and standards.

## OpenSAMM




### Narration

The Open Software Assurance Maturity Model (OpenSAMM) is an open software security framework that helps organizations devise and implement a software security strategy that is aligned to its actual risks.

The OpenSAMM project provides guidance to help organizations evaluate existing software security practices, build a software security program, demonstrate improvements to a security assurance program, and define and measure security-related activities.

OpenSAMM is designed to help evaluate an organization's current software development security practices, build a well-balanced software security program, produce demonstrable improvements in software security management, and define and measure all software security tasks within an organization.

### On Screen Text

OpenSAMM

## BSIMM


### Narration

The Building Security In Maturity Model (BSIMM) is a study of real-world software security initiatives from companies such as Adobe, Microsoft, PayPal, and many others.

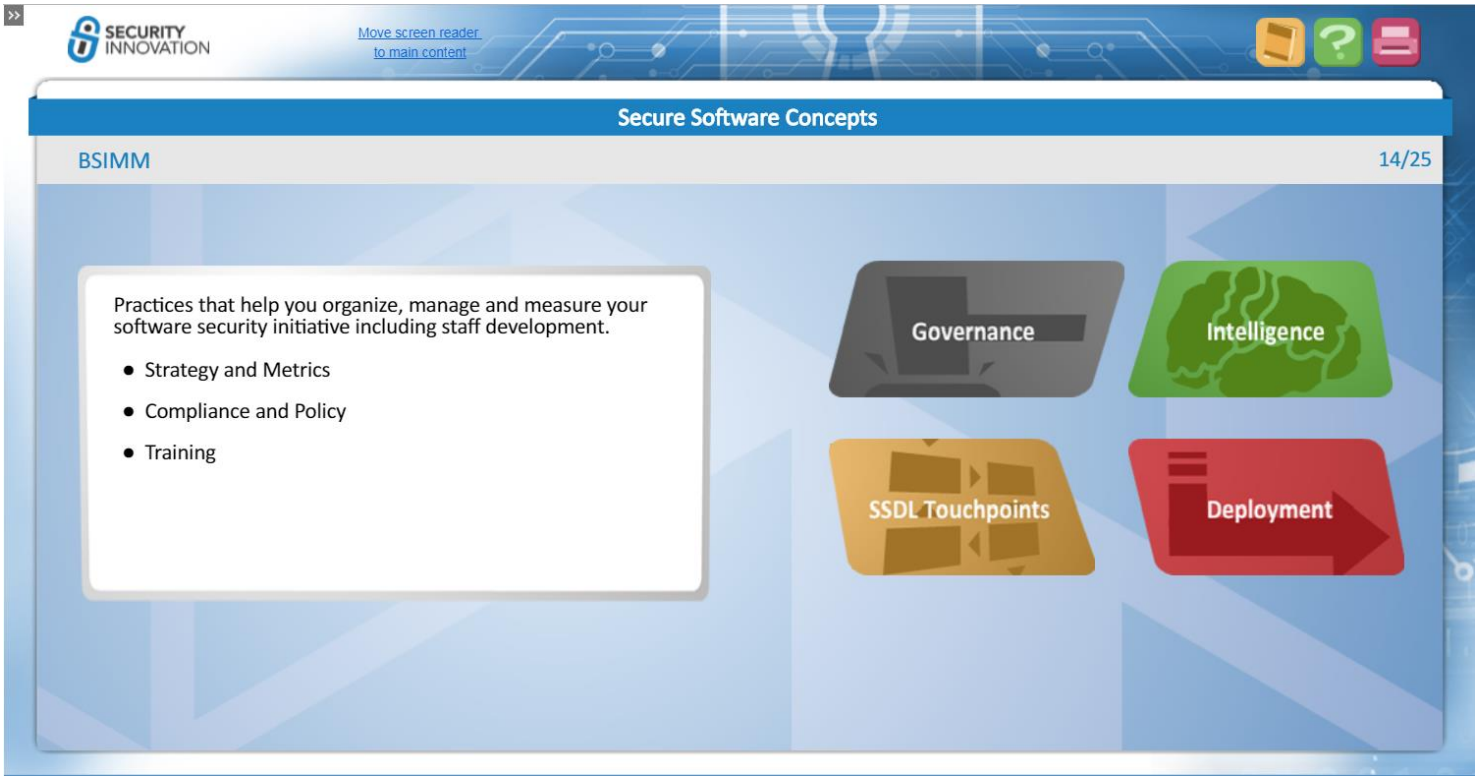
It does not represent a specific security development model, but instead provides an aggregate of the experiences and lessons learned by the participating organizations.

BSIMM describes over 100 activities that your organization can implement. Those activities are organized in a free software security framework (SSF) that defines 12 practices organized into four domains: Governance, Intelligence, Secure Software Development Lifecycle (SSDL), and Deployment.

### On Screen Text

BSIMM

## BSIMM




### Narration

Practices that help you organize, manage, and measure your software security initiative including staff development include strategy and metrics, compliance and policy, and training.

### On Screen Text

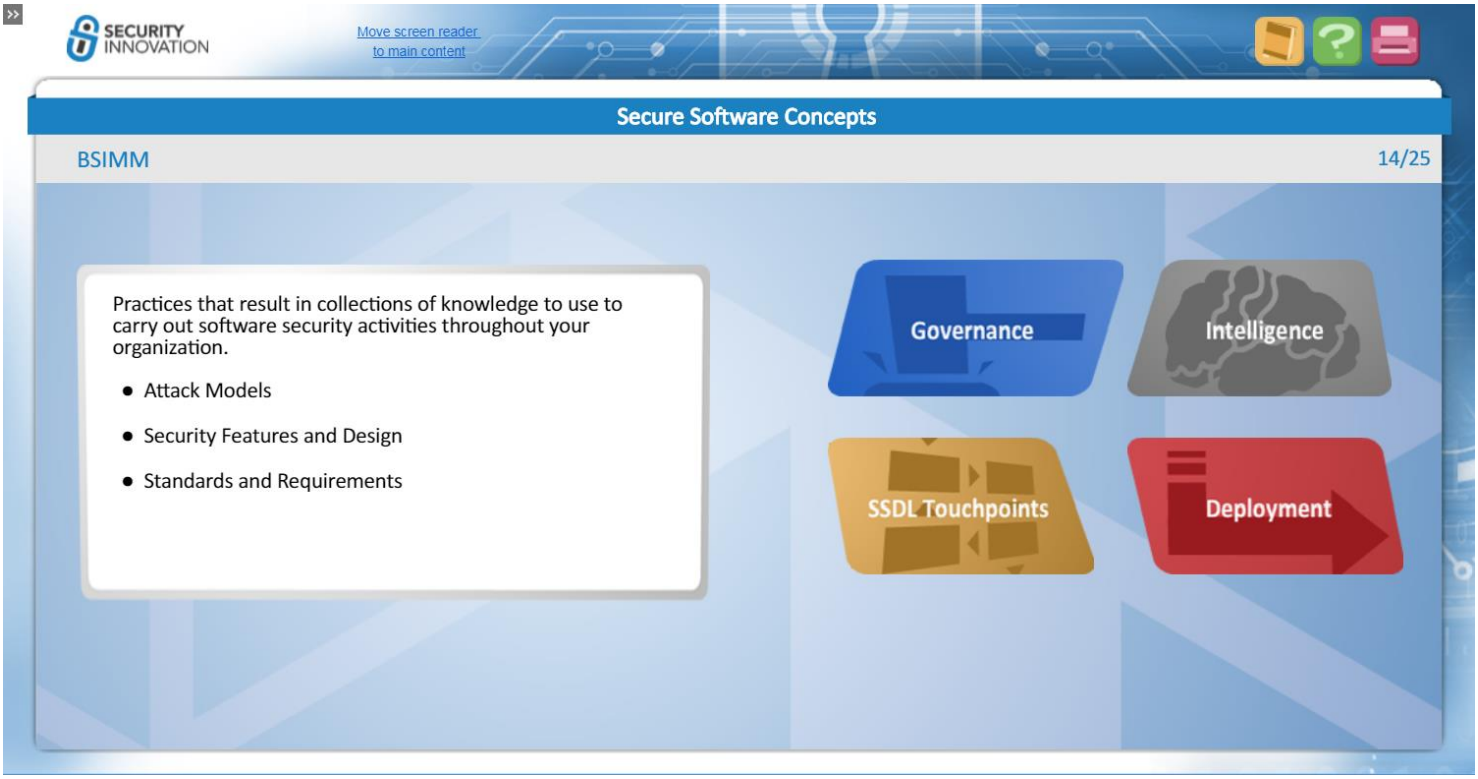
# BSIMM

### Governance

Practices that help you organize, manage and measure your software security initiative including staff development.

- Strategy and Metrics
- Compliance and Policy
- Training

## BSIMM




### Narration

Practices that result in collections of knowledge to use to carry out software security activities throughout your organization include attack models, security features and design, and standards and requirements.

### On Screen Text

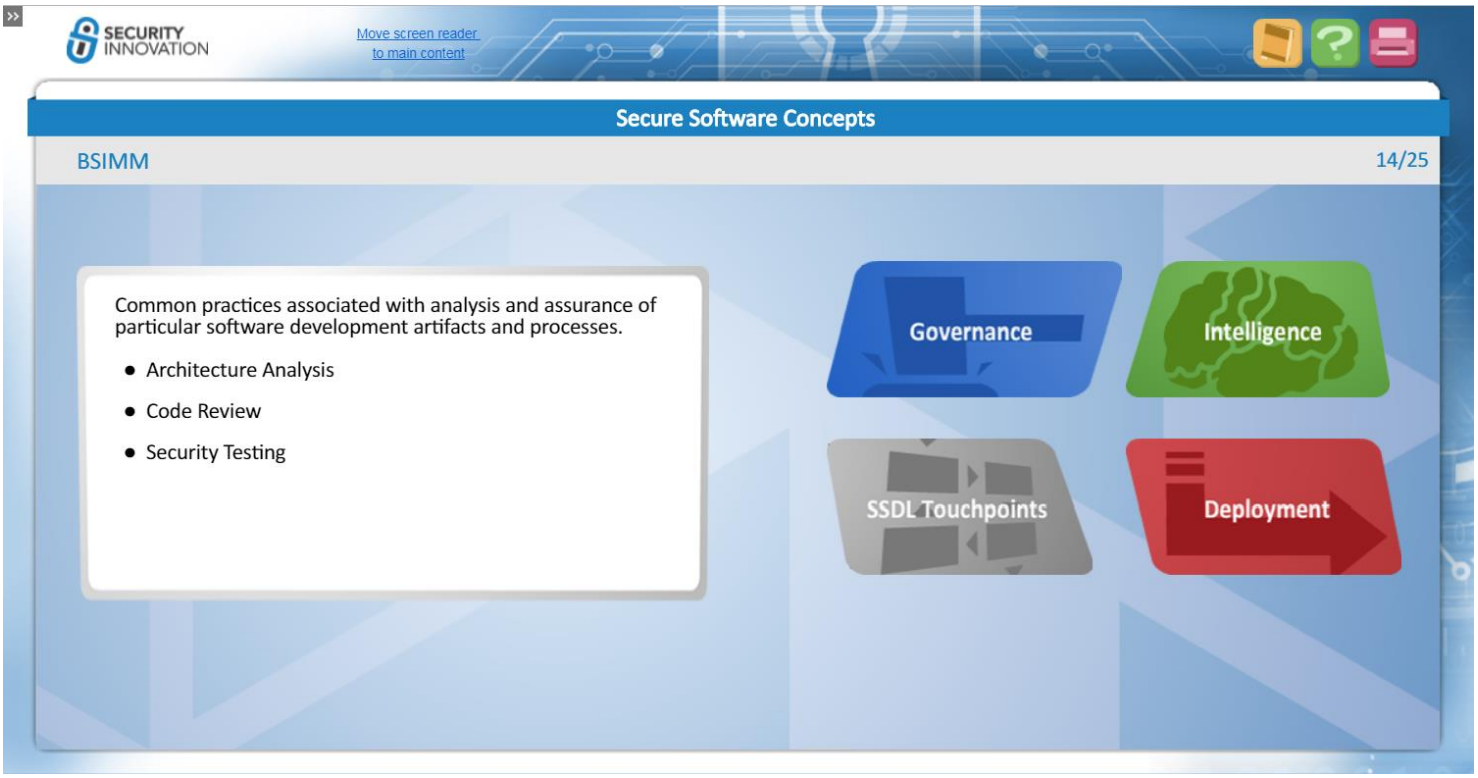
BSIMM

**Intelligence**

Practices that result in collections of knowledge to use to carry out software security activities throughout your organization.

- Attack Models
- Security Features and Design
- Standards and Requirements

## BSIMM




### Narration

Common practices associated with analysis and assurance of particular software development artifacts and processes include architecture analysis, code review and security testing.

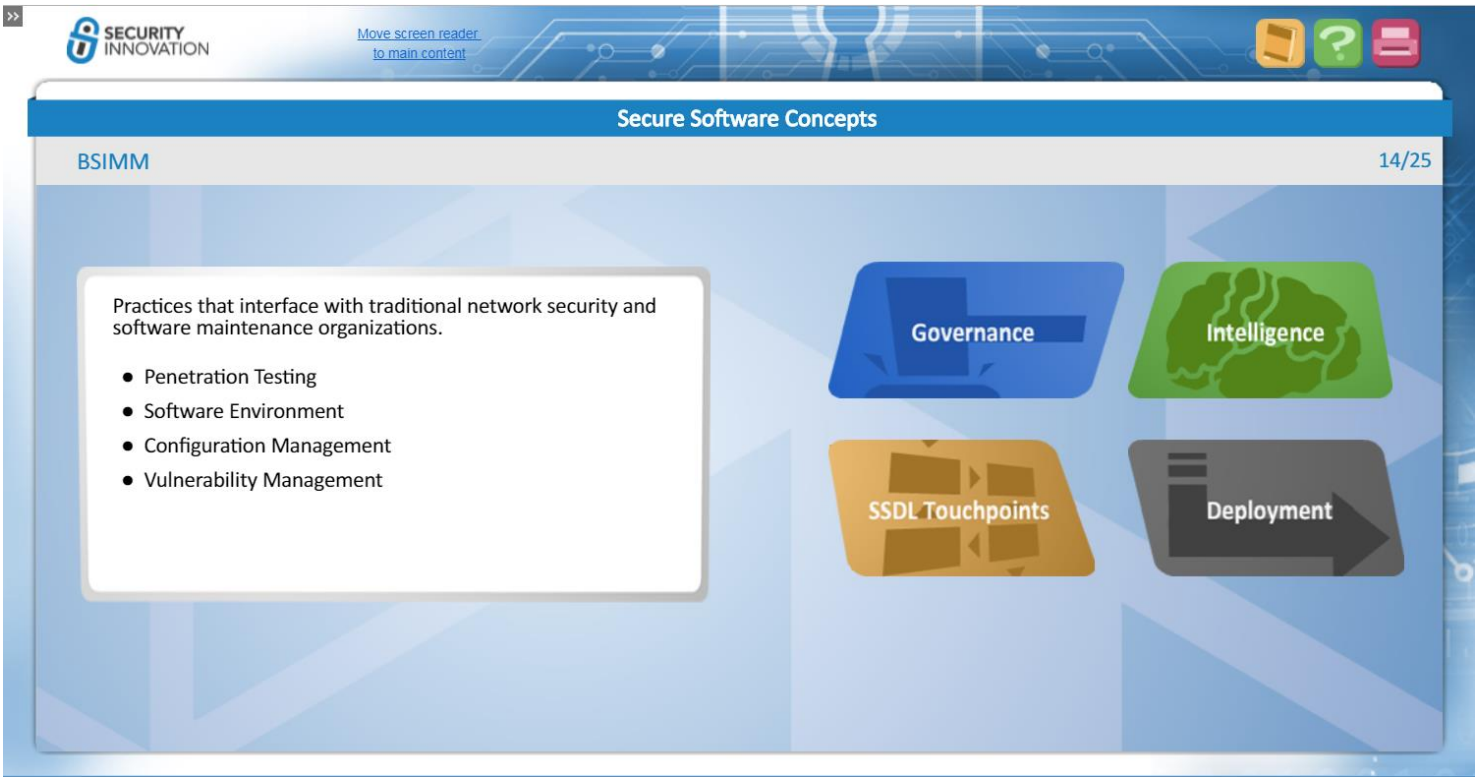
### On Screen Text

BSIMM

**SSDL Touchpoints**
Common practices associated with analysis and assurance of particular software development artifacts and processes.

- Architecture Analysis
- Code Review
- Security Testing

## BSIMM




### Narration

Practices that interface with traditional network security and software maintenance organizations include penetration testing, software environment config management, and vulnerability management.

### On Screen Text

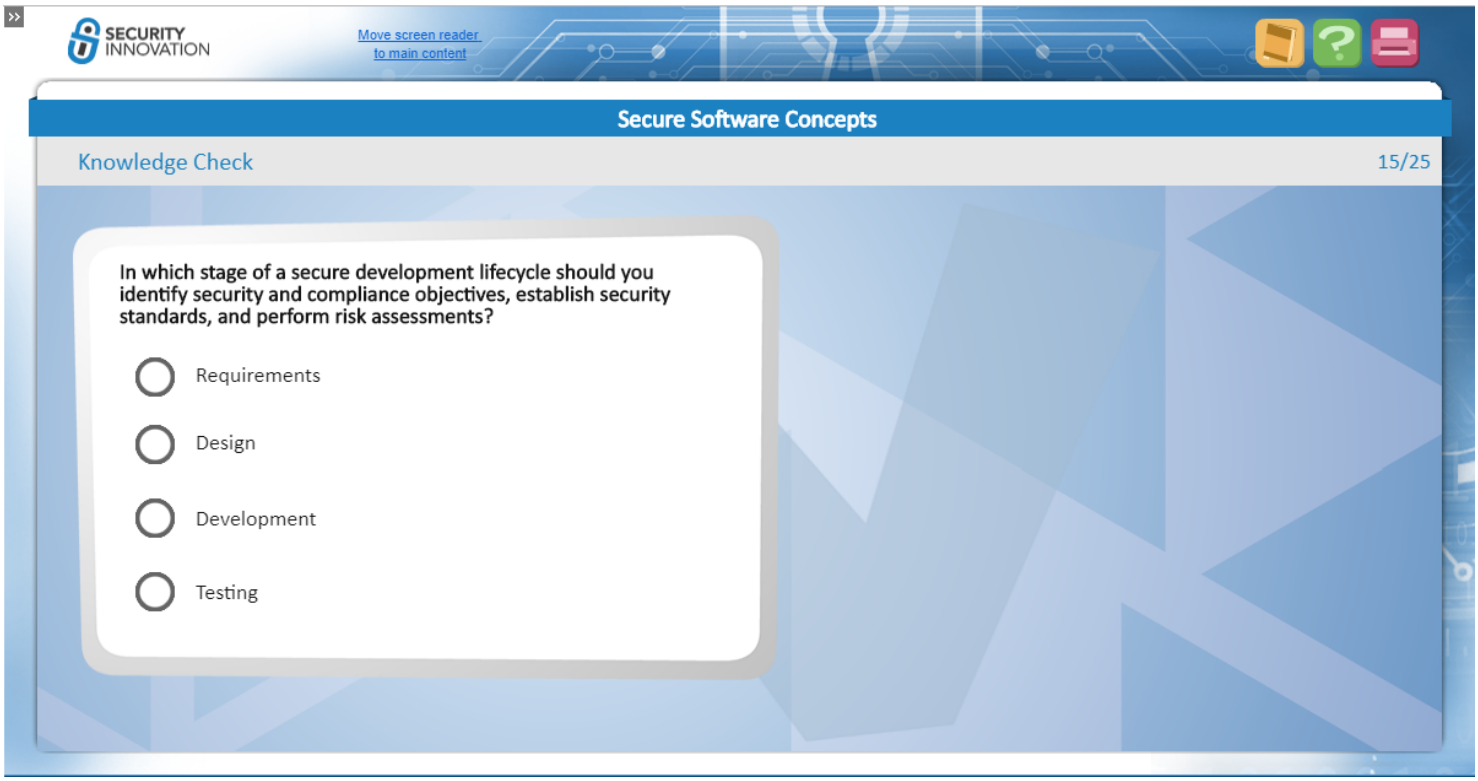
BSIMM

**Deployment**

Practices that interface with traditional network security and software maintenance organizations.

- Penetration Testing
- Software Environment
- Configuration Management
- Vulnerability Management

## Knowledge Check




**Narration**
**On Screen Text**

### Knowledge Check

In which stage of a secure development lifecycle should you identify security and compliance objectives, establish security standards, and perform risk assessments?

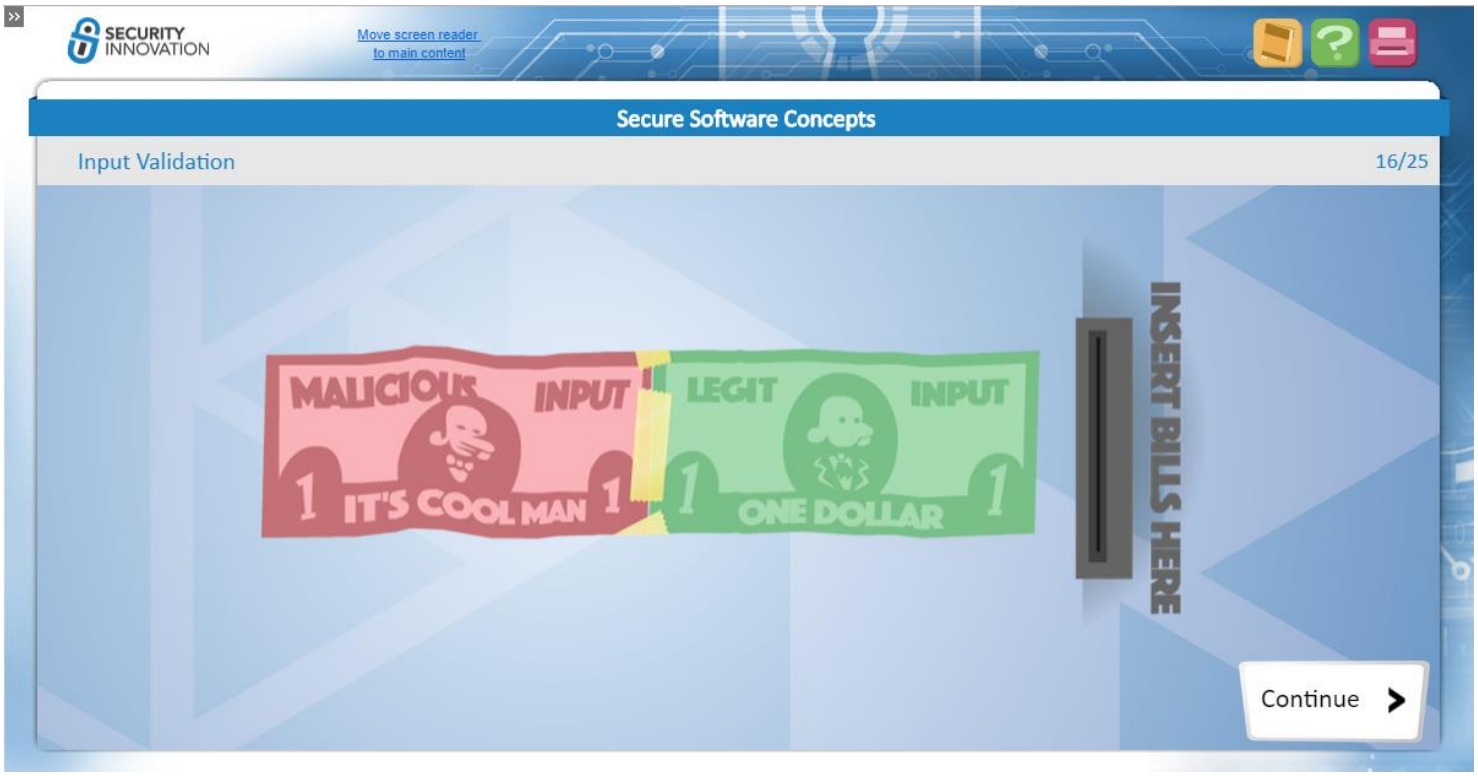
Requirements

Design

Development

Testing

### Input Validation



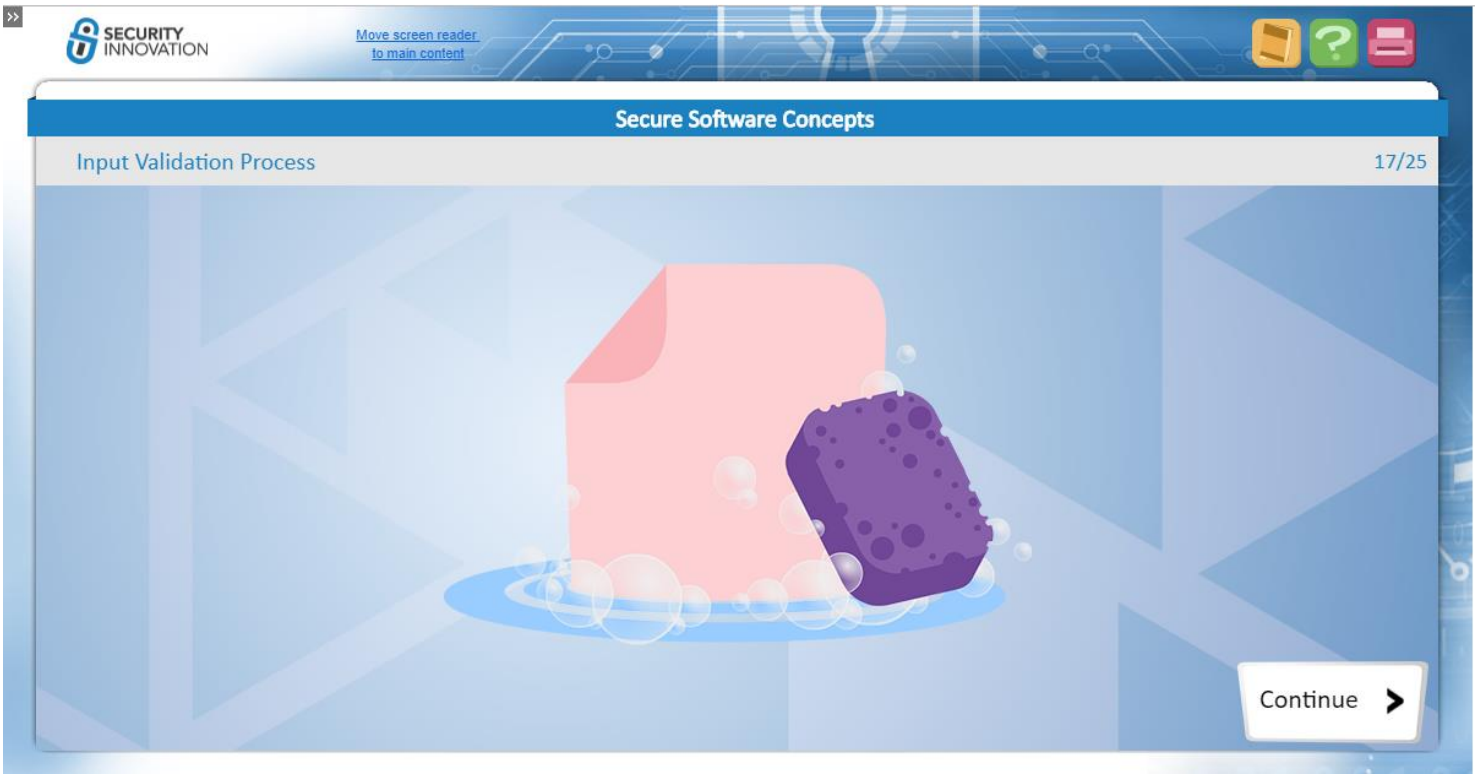

### Narration

As you learn about application security, one practice you will hear about often is input validation, primarily because it plays such a vital role in preventing security flaws.

### On Screen Text

## Input Validation

### Input Validation Process




### Narration

Input validation occurs in multiple steps:

First, there is client-side validation, which is there mostly to assist users in data entry and to improve the user experience, for example, by ensuring that a phone number does not contain letters or other invalid characters.

After the server receives the data from the web browser, there should then be a normalization (or canonicalization) process. This step helps ensure that the data is formatted consistently for validation. This step would include counteracting any attempts to obscure an attack, such as removing encoded characters.

With data normalized, the next step would be to validate it using a whitelist approach. Rather than blocking known bad data— a blacklist approach— you would only allow data based on acceptable ranges, formats, data types, and lengths. For example, a financial transaction should only contain numbers, decimals, commas, and currency symbols.

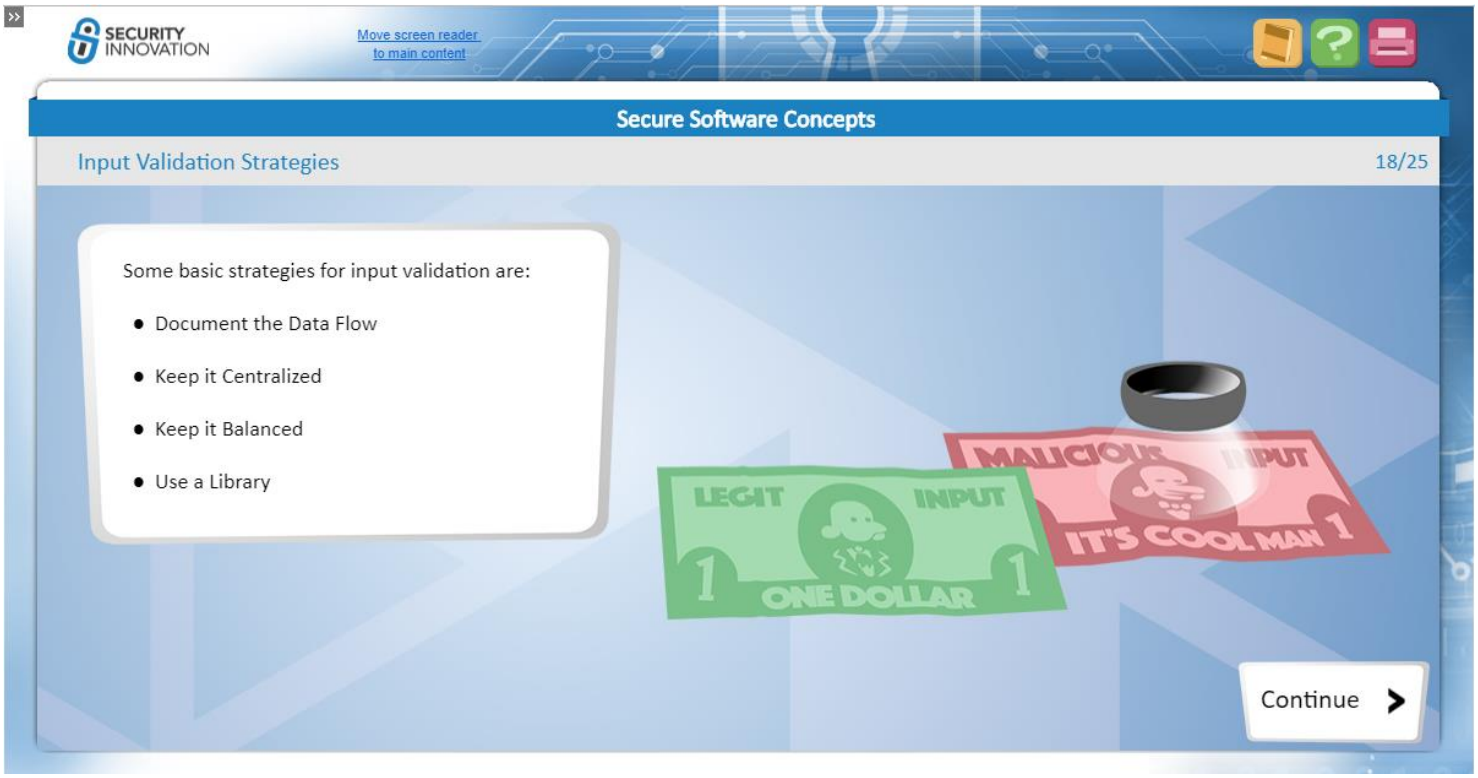
Optionally, the application might perform an additional step to strip off any data not needed, such as hyphens, parenthesis, commas, etc.

And finally, if necessary, the application might perform any character encoding required for data storage.

### On Screen Text

## Input Validation Process

## Input Validation Strategies




### Narration

Input validation isn't simply checking the values for each input; doing it right requires a comprehensive and consistent strategy across the entire application.

Some basic strategies for input validation are:

Document the Data Flow – Dissect the application to identify trust boundaries, data flows, entry points, and exit points. The key here is to know what data enters your application, where it enters, and where it originates.

Keep it Centralized – Centralizing data validation code provides a valuable chokepoint for data entering the application, and it also allows for consistent and easy-to-audit validation code.

Keep it Balanced – While validating data is important, it should not detract from the user experience. Some data can be automatically sanitized—such as removing dashes from a phone number—rather than rejected for not matching a specific format.

Use a Library – Many application frameworks and common libraries have robust data-validation functions built in. Make use of these features when possible as they have often been well-tested in a variety of scenarios.

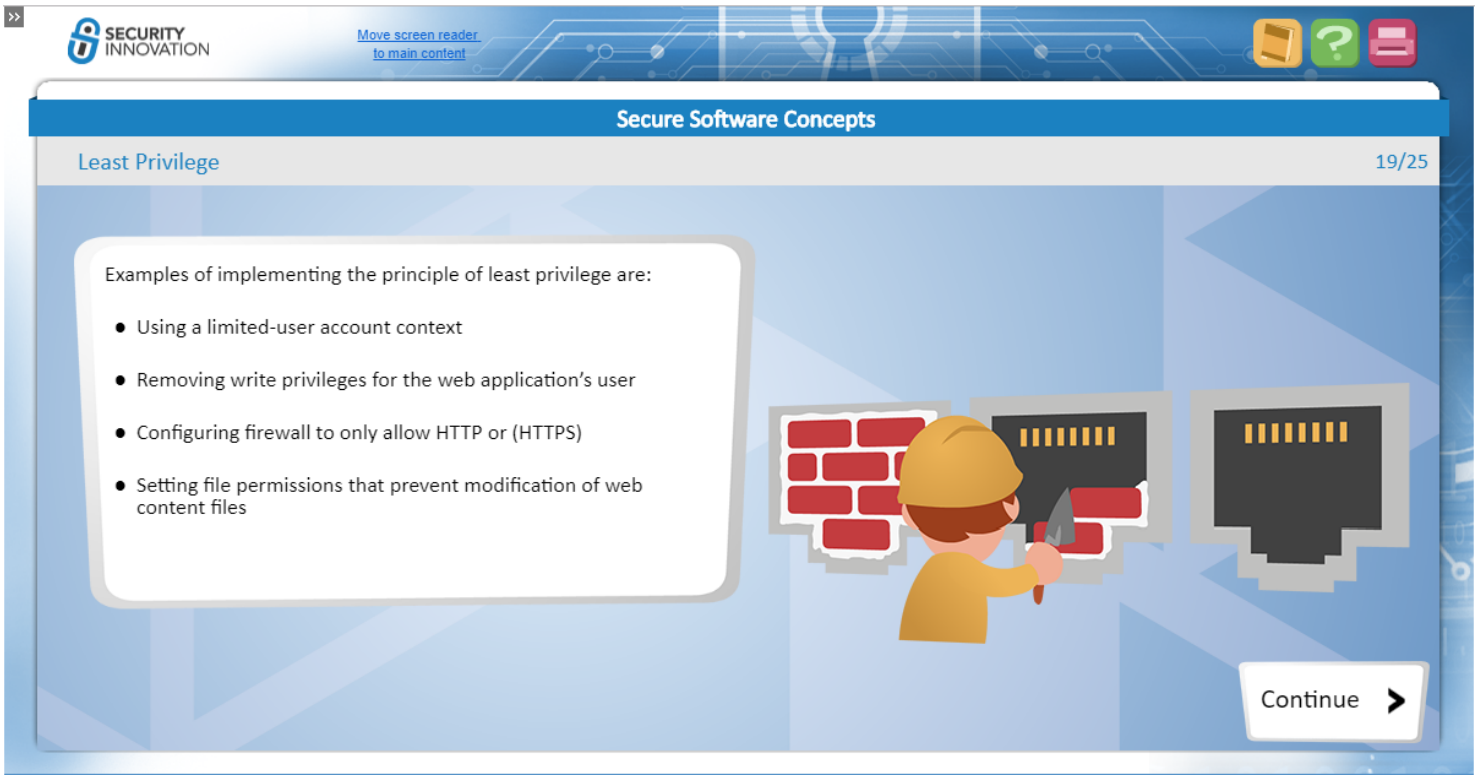
**On Screen Text**

## Input Validation Strategies

Some basic strategies for input validation are:

- Document the Data Flow
- Keep it Centralized
- Keep it Balanced
- Use a Library

## Least Privilege




### Narration

The least privileges principle begins with the idea that all software can and will eventually be compromised by a malicious user. To reduce the impact of a compromise, we must design applications using the minimal set of privileges required to function correctly. This principle applies across all layers of the application, including the underlying operating system, databases, and cloud accounts.

The principle of least privilege accomplishes two things. First, it reduces the attack surface and thus lessens the likelihood of successful attacks. Second, it limits capabilities after a successful attack and thus makes it more difficult to execute lateral attacks.

Examples of implementing the principle of least privilege are: Using a limited-user account context for running the web, database, and application server software; removing write privileges for the web application's user if an application doesn't need to write to a database table; configuring a firewall to only allow a web server to communicate via Hypertext Transfer Protocol (HTTP) or Hypertext Transfer Protocol Secure (HTTPS); and, setting file permissions that prevent the web server platform from modifying web content files.

### On Screen Text

Least Privilege

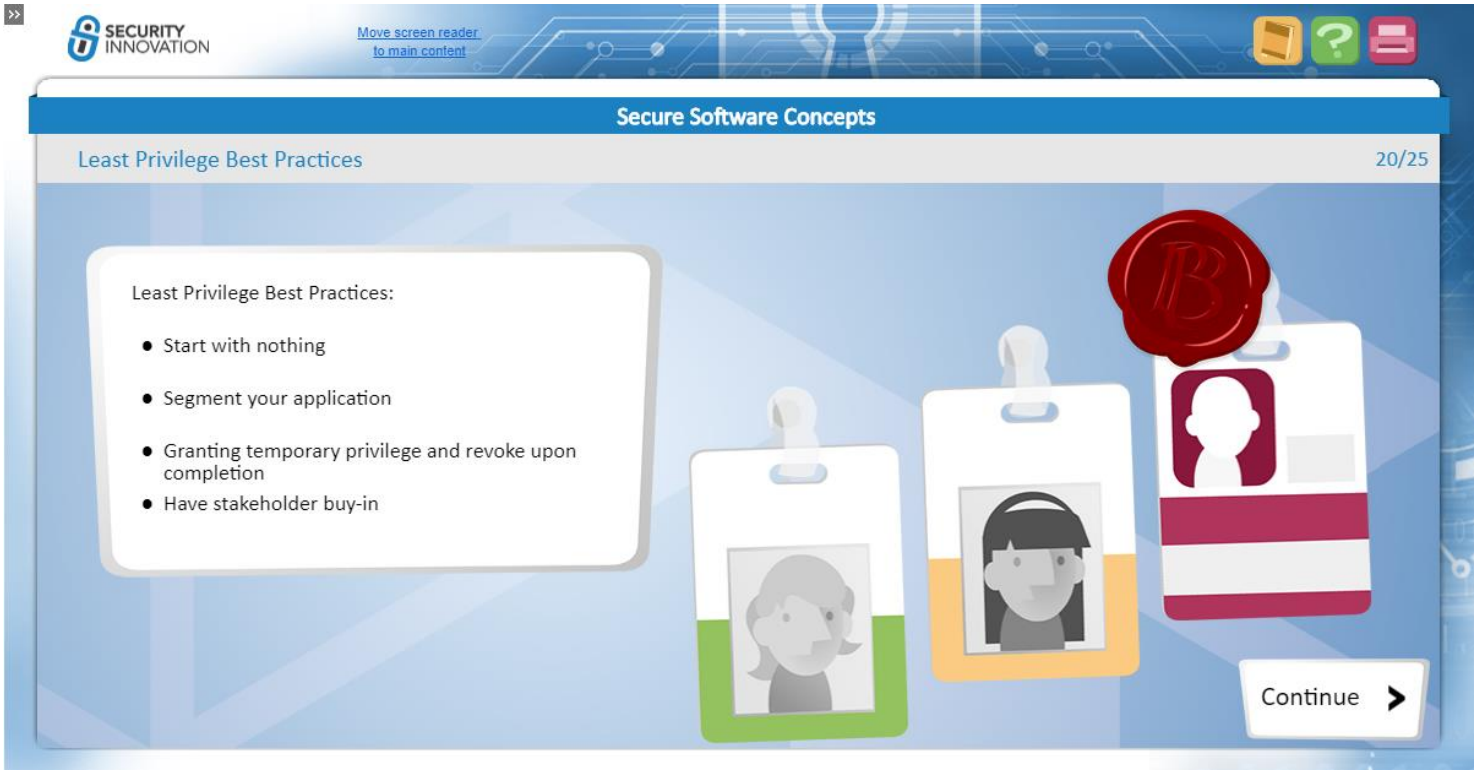
The principle of least privilege accomplishes two things:

- Reduces the attack surface and thus lessens the likelihood of successful attacks.
- Limits capabilities after a successful attack and thus makes it more difficult to execute lateral attacks.

Examples of implementing the principle of least privilege are:

- Using a limited-user account context
- Removing write privileges for the web application’s user
- Configuring firewall to only allow HTTP or (HTTPS)
- Setting file permissions that prevent modification of web content files

### Least Privilege Best Practices




**Narration**

As you implement the principle of least privilege security into your application, keep in mind the following tips:

Start with nothing, treating every user or process as having no privileges and only granting those privileges necessary to perform each action.

Segment your application so that you can easily implement a role-based approach to security.

If necessary to give a user or process higher privileges for one action, consider granting that privilege only temporarily and revoking it as soon as possible after completion of the action.

Finally, because least privilege can limit what certain users can do, make sure you have stakeholder buy-in before introducing restrictions.

**On Screen Text**

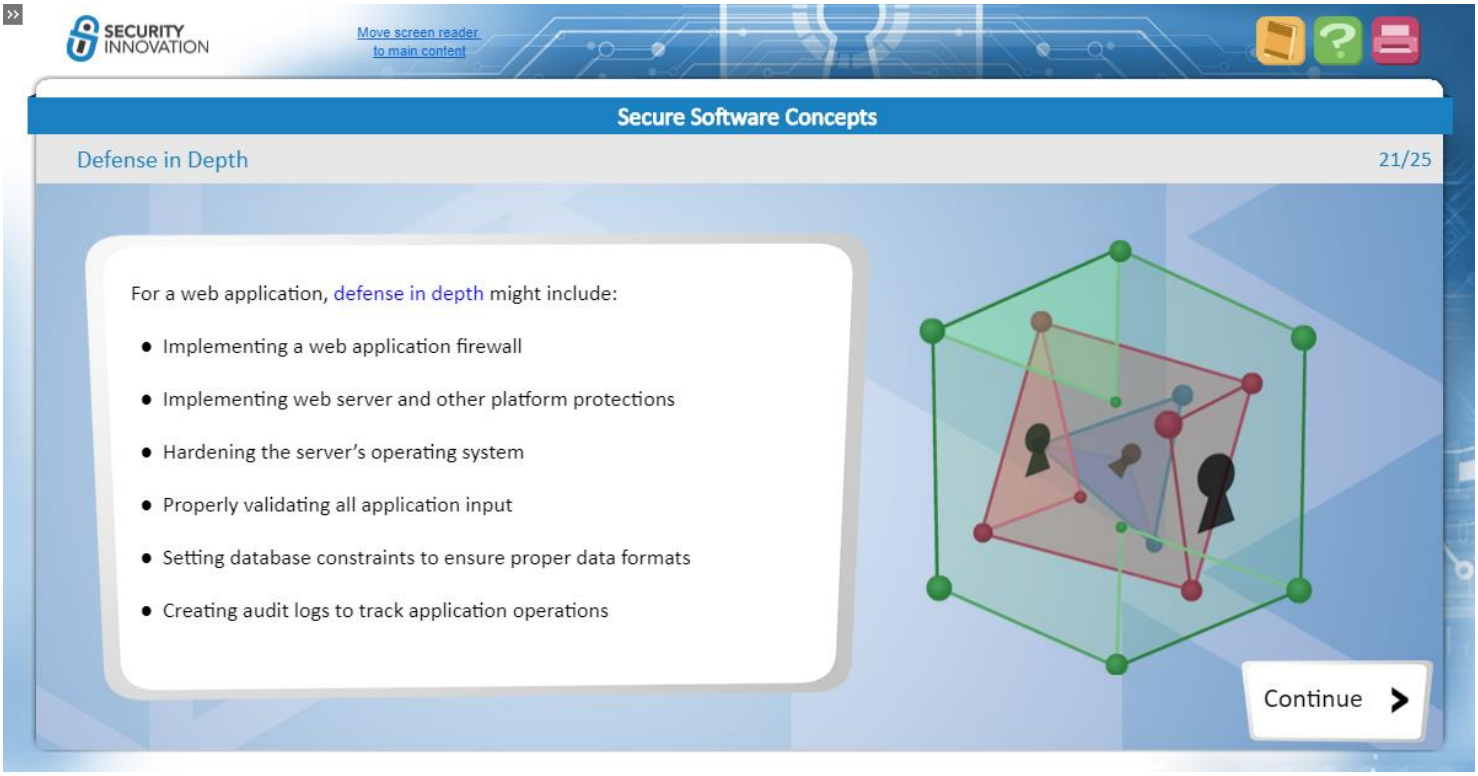
## Least Privilege Best Practices

Least Privilege Best Practices:

- Start with nothing
- Segment your application

- Granting temporary privilege and revoke upon completion
- Have stakeholder buy-in

## Defense in Depth




### Narration

Another important security principle is defense in depth—in short creating multiple layers of security defenses to protect from failures in any one layer.

Even the most sophisticated security techniques are prone to failure. With multiple layers of defense, however, each layer increases the difficulty and effort required to compromise the application. Defense in depth ensures that all but the most sophisticated and determined attacks will fail. Defense in depth not only provides redundancy, but it can also address unexpected attack vectors, such as internal threats or physical attacks.

For a web application, defense in depth might include: implementing a web application firewall; implementing web server and other platform protections; hardening the server's operating system; properly validating all application input; setting database constraints to ensure proper data formats; and, creating audit logs to track application operations.

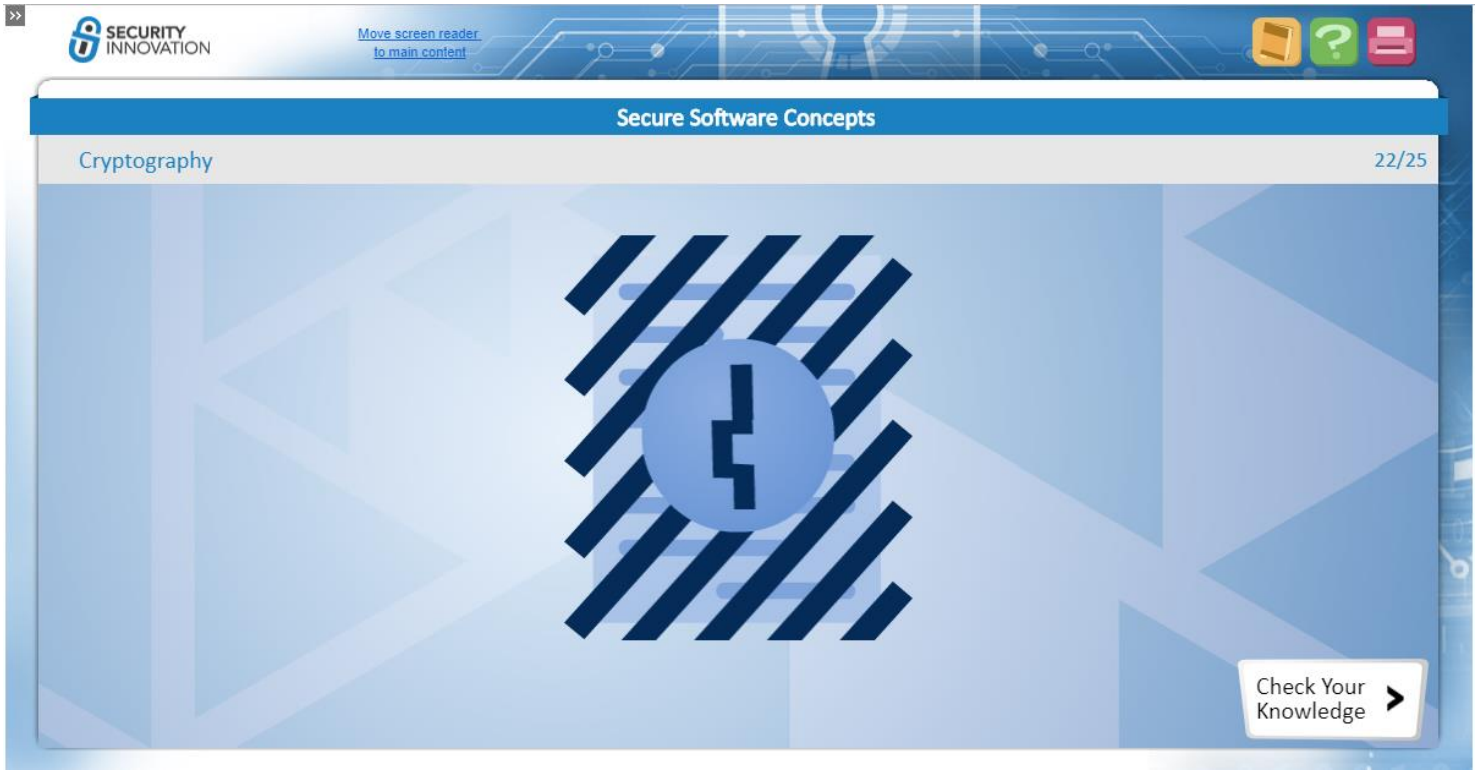
### On Screen Text

Defense in Depth

For a web application, defense in depth might include:

- Implementing a web application firewall
- Implementing web server and other platform protections
- Hardening the server's operating system
- Properly validating all application input
- Setting database constraints to ensure proper data formats
- Creating audit logs to track application operations

## Cryptography




### Narration

Protecting private user information means encrypting data both in transit and on disk. Cryptography is a key security principle, but it isn't just about encryption. Cryptography also provides for authentication, confidentiality, and integrity. Authentication is the verification of a claim. That claim could be the assertion that a person is who they say they are or that software is what it claims to be. Authentication can also apply to web sites, allowing you to ensure that you are connecting to the proper host.

Confidentiality is the assurance that communications between two parties are private and cannot be intercepted by any third party.

Integrity is proof that data has not changed in transit from one location to another or from one medium to another. Public key cryptography provides a means for authentication through the use of public key certificates, confidentiality with asymmetric encryption, and integrity by means of digital signatures.

Cryptography in general is mature and appropriately secure when implemented with well-established libraries and following the proper best practices.

### On Screen Text

Cryptography

### Knowledge Check


### Narration

### On Screen Text

## Knowledge Check

**Which one of the following is the correct order of steps for input validation?**

Whitelist validation, encoding, canonicalization

Client-side validation, blacklist validation, encoding

Encoding, canonicalization, client-side validation

Canonicalization, whitelist validation, encoding

## Course Summary

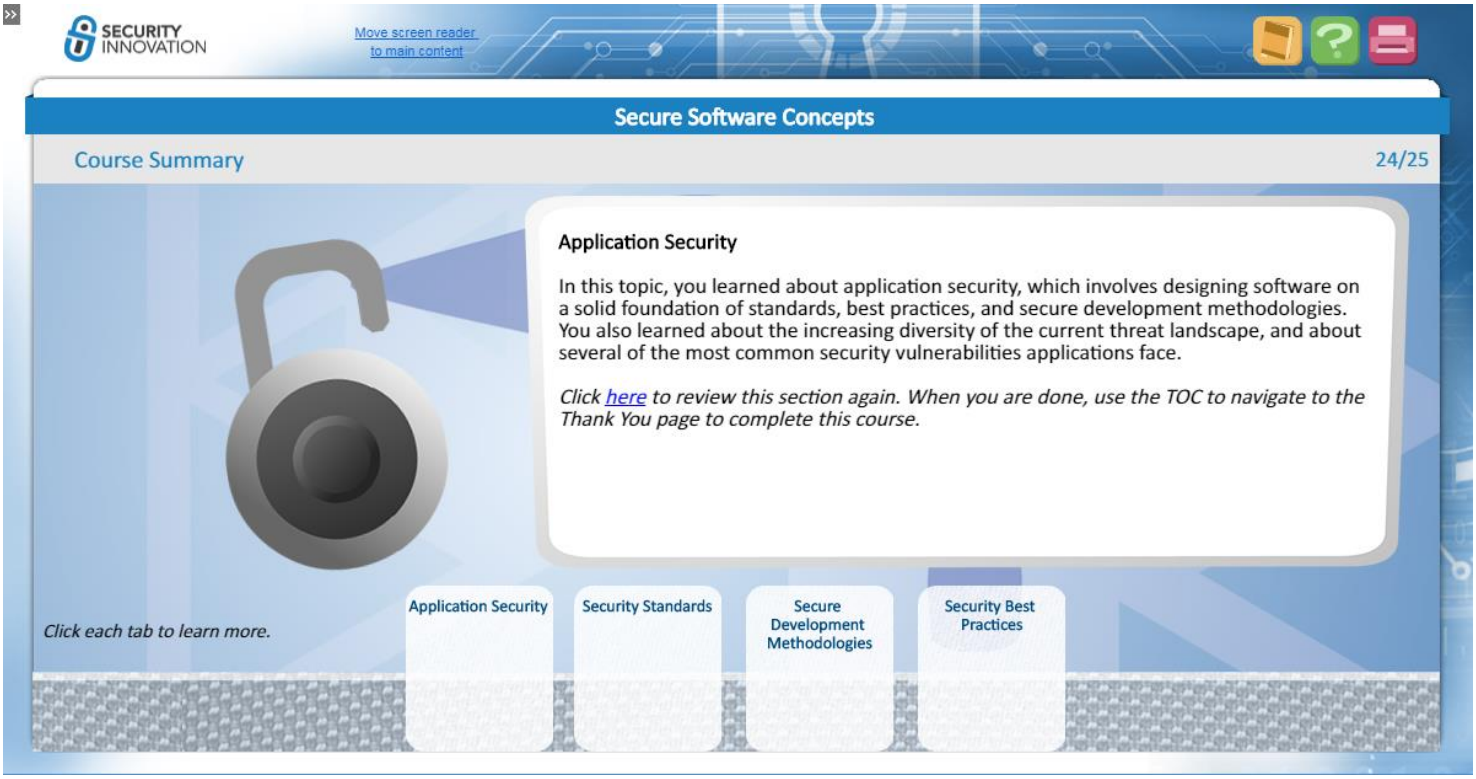

### Narration

### On Screen Text

## Course Summary

*Click each tab to learn more.*

### Course Summary




### Narration

In this topic, you learned about application security, which involves designing software on a solid foundation of standards, best practices, and secure development methodologies. You also learned about the increasing diversity of the current threat landscape, and about several of the most common security vulnerabilities applications face.

*Click here to review this section again. When you are done, use the TOC to navigate to the Thank You page to complete this course*.

### On Screen Text

## Course Summary

### Application Security

In this topic, you learned about application security, which involves designing software on a solid foundation of standards, best practices, and secure development methodologies. You also learned about the increasing diversity of the current threat landscape, and about several of the most common security vulnerabilities applications face.

*Click here to review this section again. When you are done, use the TOC to navigate to the Thank You page to complete this course*.

*Click each tab to learn more.*

### Course Summary


### Narration

In this topic, you learned that security policies, standards, and regulations should be integrated into the entire development lifecycle to inform secure coding best practices. This topic also discussed the OWASP Top 10 and ASVS projects and the CWE/SANS Top 25 list for evaluating and mitigating risks from the most common application security vulnerabilities.

*Click here to review this section again. When you are done, use the TOC to navigate to the Thank You page to complete this course*.

### On Screen Text

## Course Summary

### Security Standards

In this topic, you learned that security policies, standards, and regulations should be integrated into the entire development lifecycle to inform secure coding best practices. This topic also discussed the OWASP Top 10 and ASVS projects and the CWE/SANS Top 25 list for evaluating and mitigating risks from the most common application security vulnerabilities.

*Click here to review this section again. When you are done, use the TOC to navigate to the Thank You page to complete this course*.

*Click each tab to learn more.*

### Course Summary


### Narration

This topic provided examples of security-related tasks you might perform at each stage of a secure software development lifecycle. It also described OpenSAMM, a framework which can help you implement a security strategy aligned to the actual risks faced by your organization, and BSIMM, an aggregate report of the lessons learned by major corporations in their experiences implementing secure development practices.

*Click here to review this section again. When you are done, use the TOC to navigate to the Thank You page to complete this course.*

### On Screen Text

## Course Summary

### Secure Development Methodologies

This topic provided examples of security-related tasks you might perform at each stage of a secure software development lifecycle. It also described OpenSAMM, a framework which can help you implement a security strategy aligned to the actual risks faced by your organization, and BSIMM, an aggregate report of the lessons learned by major corporations in their experiences implementing secure development practices.

*Click here to review this section again. When you are done, use the TOC to navigate to the Thank You page to complete this course.*

*Click each tab to learn more.*

### Course Summary


### Narration

In this topic you learned about four security best practices: input validation, which is the first line of defense against many attacks; least privilege, which minimizes the potential impact of successful attacks; defense in depth, which strengthens your security position; and cryptography, which when applied properly can protect sensitive data and provide assurances of confidentiality, integrity, and authenticity.

*Click here to review this section again. When you are done, use the TOC to navigate to the Thank You page to complete this course*.

### On Screen Text

## Course Summary

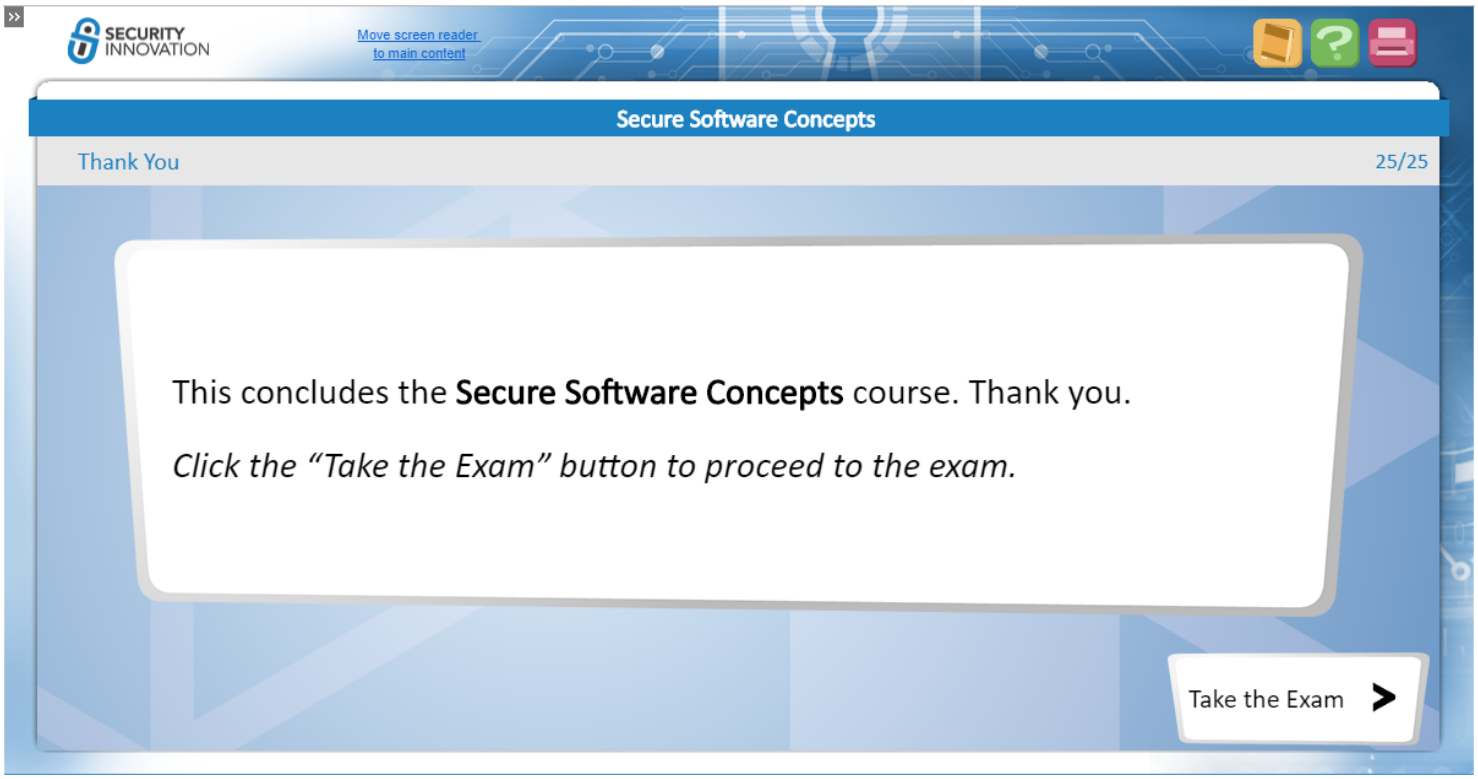
### Security Best Practices

In this topic you learned about four security best practices: input validation, which is the first line of defense against many attacks; least privilege, which minimizes the potential impact of successful attacks; defense in depth, which strengthens your security position; and cryptography, which when applied properly can protect sensitive data and provide assurances of confidentiality, integrity, and authenticity.

*Click here to review this section again. When you are done, use the TOC to navigate to the Thank You page to complete this course*.

*Click each tab to learn more.*

**Thank You**




**Narration**

**On Screen Text**

## Thank You

This concludes the **Secure Software Concepts** course. Thank you.

*Click the "Take the Exam" button to proceed to the exam.*