Course Overview and Objectives

Narrations

This course introduces you to the business and regulatory imperatives for secure software development.

It presents models, standards, and guidelines that can help you understand security issues and improve the security posture of your applications.

And, it describes how to integrate secure development practices into the software development lifecycle.

After completing this course, you will be able to identify major factors in the business, regulatory, and threat landscapes that drive the demand for software security.

You will also be able to identify important security models, standards, and guidelines that provide a strategic roadmap for application security.
In addition, you will be able to identify secure development practices and principles that correspond to each phase of the software development lifecycle.

On Screen Text

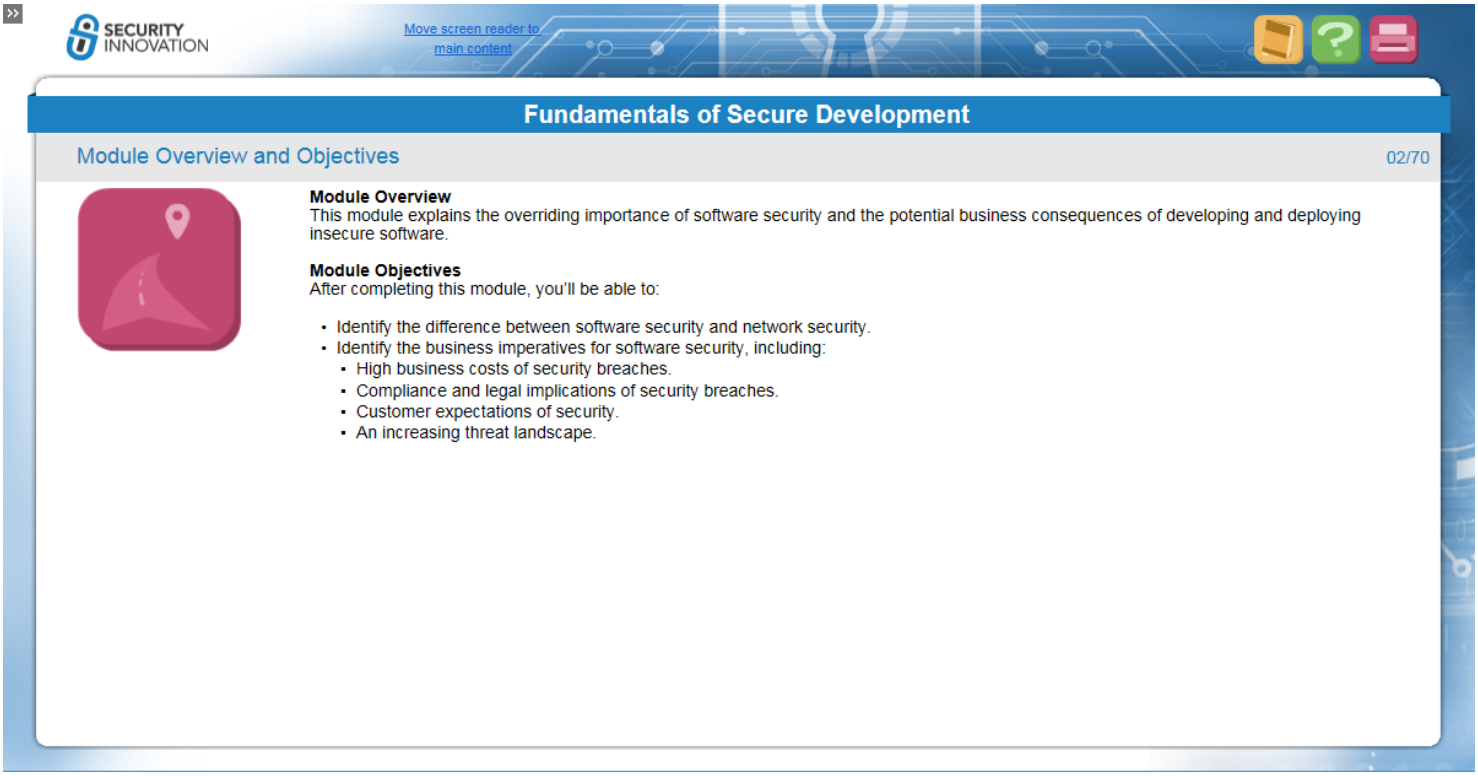
Course Overview and Objectives

This course introduces you to the business and regulatory imperatives for secure software development. It presents models, standards, and guidelines that can help you understand security issues and improve the security posture of your applications. And, it describes how to integrate secure development practices into the software development lifecycle.

After completing this course, you will be able to:

- Identify major factors in the business, regulatory, and threat landscapes that drive the demand for software security.
- Identify important security models, standards, and guidelines that provide a strategic roadmap for application security.
- Identify secure development practices and principles that correspond to each phase of the software development lifecycle.

Module Overview and Objectives



Narrations

This module explains the overriding importance of software security for your organization, and the potential business consequences of developing and deploying insecure software.

After completing this module, you'll be able to identify the difference between software security and network security.

In addition, you'll be able to identify the business imperatives for software security, including the high

business costs of security breaches, the compliance and legal implications of security breaches,

customer expectations of security, and an increasing threat landscape.

On Screen Text

Module Overview and Objectives

Module Overview
This module explains the overriding importance of software security and the potential business consequences of developing and deploying insecure software.

Module Objectives

After completing this module, you'll be able to:

- Identify the difference between software security and network security.
- Identify the business imperatives for software security, including:
    - High business costs of security breaches.
    - Compliance and legal implications of security breaches.
    - Customer expectations of security.
    - An increasing threat landscape.

What is Software Security?

Narrations

Let's begin by looking at the definition of software security.

Software security focuses on protecting the information and resources made available by applications running on computer systems.

Software security is often confused with network security.

Network security focuses on protecting the IT infrastructure that controls the flow of data between network systems.

Network security controls, such as firewalls and intrusion detection systems, are often insufficient for software security.

Software security attacks often occur at higher layers of operation not visible to network security controls. Also, applications may implement proprietary protocols not recognized by network security controls.

On Screen Text

What is Software Security?

- Software security focuses on protecting information and resources made accessible by applications running on computer systems.

- Software security is often confused with network security. Network security focuses on the protection of an organization's IT infrastructure—the servers and related equipment that control the flow of data between networked systems.
- Network security controls are typically insufficient to mitigate software security risks.
- Software security attacks often occur at higher layers of operation, not visible to network security controls.

The Security Imperative

Narrations

Let's look at some of the key reasons that organizations need to develop software that is more secure. They include high business costs of security breaches, compliance and legal implications of security breaches, customer expectations of security, and an increasing threat landscape.

On Screen Text

The Security Imperative

Key reasons why organizations must develop and deploy secure software include:

- High business costs of security breaches
- Compliance and legal implications of security breaches
- Customer expectations of security
- An increasing threat landscape

The Business Costs of Software Security Breaches

Narration

An organization that develops or deploys insecure software and suffers a security breach is exposed to various risks that have high direct and indirect costs.

For example, when an application is hacked, an organization can suffer the effects of downtime, as well as the expense related to forensics and restoring operations.

Financial claims and settlements can raise costs, as can compliance penalties.

Loss of investor and customer confidence might lead to lost sales and damage to the

company's reputation. According to a study by McAfee security, the global annual cost of

cybercrime is over $400 billion, making it clear that organizations must develop and

deploy more secure software to reduce their overall financial exposure.

On Screen Text

The Business Costs of Software Security Breaches

Software security breaches expose organizations to various risks that result in the following direct and indirect costs:

- Downtime and other operational losses if an application is hacked
- Financial claims and settlements

- Compliance fines and penalties
- Loss of investor and customer confidence
- Lost sales
- Damage to reputation

*Source: *Net Losses: Estimating the Global Cost of Cybercrime,* June 2014, McAfee, Intel Security

Compliance and Legal Implications

Narration

As cybercrime becomes more widespread and creates more burdens on society, regulators are responding by placing more emphasis on software security and protecting sensitive customer data.

For example, Massachusetts has passed data protection laws requiring businesses to protect the sensitive information of residents. Other states are undertaking similar efforts. Even if an organization is not required by local regulations to develop secure software, they might still need to do so in order to conduct business in other states.

It is difficult, if not impossible, to ignore software security without incurring significant fines or loss of business.

On Screen Text

Compliance and Legal Implications

Federal and state regulations mandate software security and require protection of sensitive customer data.

Even if an organization is not required by local regulations to develop secure software, they might still need to do so to conduct business in other states.

It is difficult, if not impossible, to ignore software security without incurring significant fines or loss of business.

Customers Expect Secure Solutions

Narrations

Regulations, legislation, and media have made businesses and professionals more aware of the need for secure software and services.

As a result, today's customers use security as part of their decision-making process when selecting a solution or a partner.

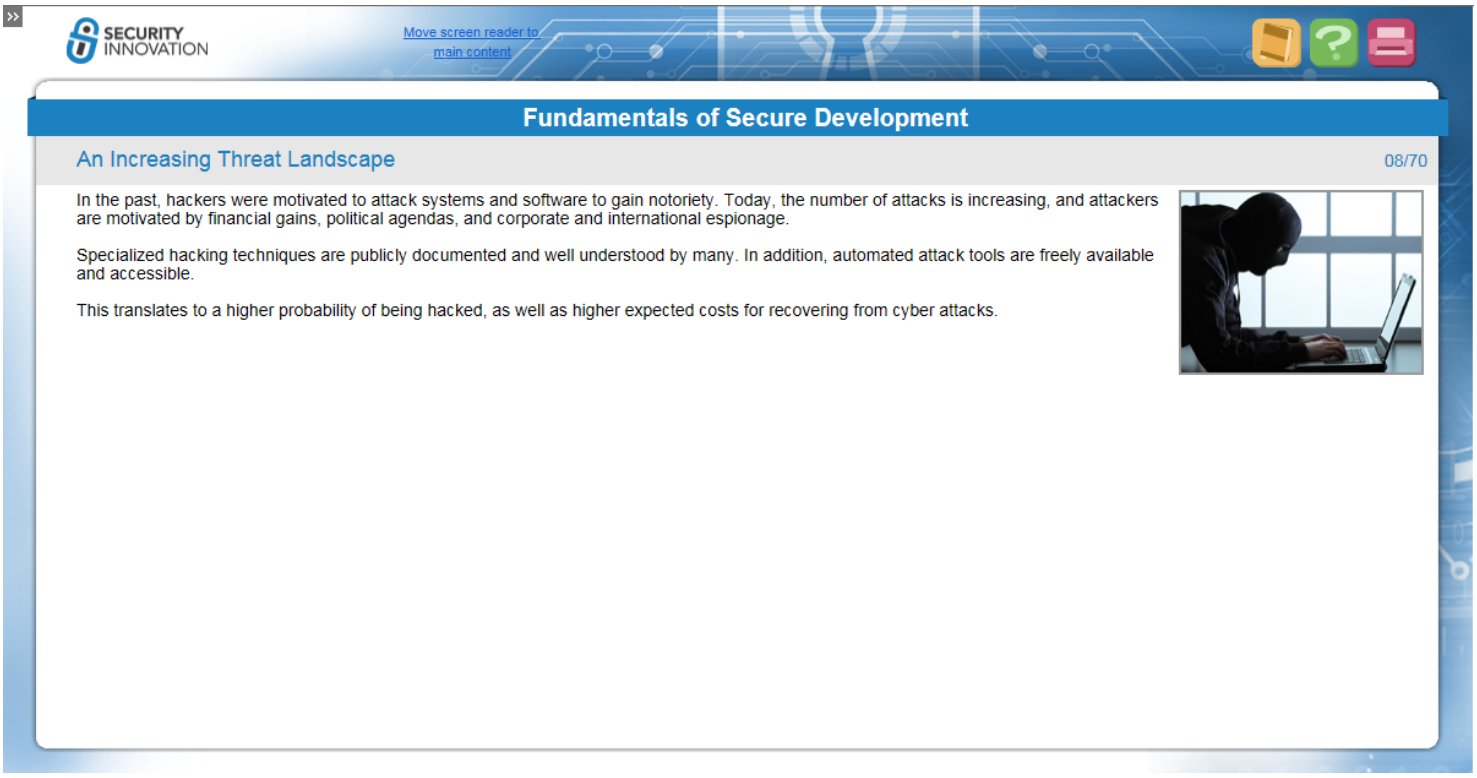
Failure to develop and deliver secure software and services could damage your organization's reputation, and drive business to your competitors.

On Screen Text

Customers Expect Secure Solutions

- Regulatory and legislative measures and the media have increased business awareness of the need for secure software and services.
- Failure to develop and deliver secure software and services could damage your organization's reputation, and drive business to your competitors.

An Increasing Threat Landscape



Narrations

In the past, hackers were motivated to attack systems and software for reputational gains amongst their peers. Today, that has dramatically changed. The number and frequency of attacks has increased, and attackers are primarily motivated by significant financial gains, political agendas, and corporate and international espionage.

Specialized hacking techniques that used to be known only among a few are now publicly documented, and well understood by many.

To make matters worse, even amateurs can use off-the-shelf tools and canned scripts to make up for a lack of skill.

Automated attack tools are freely available and accessible.

All these factors increase the likelihood that organizations will get hacked, leading to higher business costs for recovering from cyber attacks.

On Screen Text

An Increasing Threat Landscape

In the past, hackers were motivated to attack systems and software to gain notoriety. Today, the number of attacks is increasing, and attackers are motivated by financial gains, political agendas, and corporate and international espionage.

Specialized hacking techniques are publicly documented and well understood by many. In addition, automated attack tools are freely available and accessible.

This translates to a higher probability of being hacked, as well as higher expected costs for recovering from cyber attacks.

Knowledge Check

On Screen Text

Knowledge Check

Network security controls such as firewalls and intrusion detection systems are sufficient to address software security risks because software must operate on top of networks.

- True
- False

Module Summary

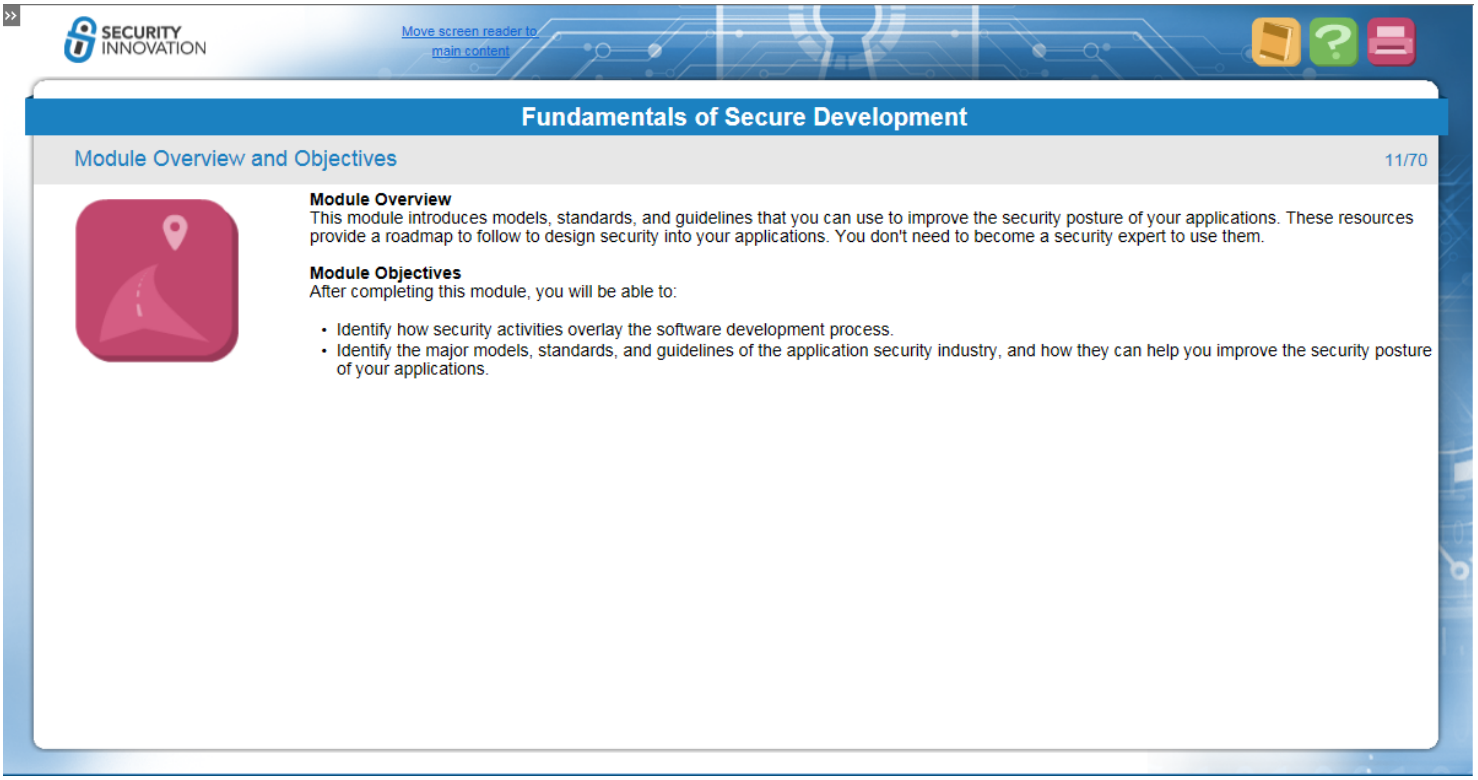
On Screen Text

Module Summary

In this module, you learned how software security differs from network security.

You also learned the key reasons why organizations must develop secure software, including high business costs of security breaches, compliance and legal implications of security breaches, customer expectations of security, and a changing threat landscape.

Module Overview and Objectives



Narrations

This module introduces models, standards, and guidelines that you can use to improve the security posture of your applications.

These resources provide a roadmap to follow to design security into your applications.

You don’t need to become a security expert to use them.

After completing this module, you will be able to identify how security activities

overlay the software development process. You will also be able to identify the

major models, standards, and guidelines of the application security industry, and

how they can help you improve the security posture of your applications.

On Screen Text

Module Overview and Objectives

Module Overview

This module introduces models, standards, and guidelines that you can use to improve the security posture of your applications. These resources provide a roadmap to follow to design security into your applications. You don't need to become a security expert to use them.

Module Objectives

After completing this module, you will be able to:

- Identify how security activities overlay the software development process.
- Identify the major models, standards, and guidelines of the application security industry, and how they can help you improve the security posture of your applications.

Overview of Secure Development Models Standards and Guidelines

Narrations

As a developer, you might not have the time, resources, or interest to become a security expert.

Yet, you have a responsibility to your organization and customers to develop secure code.

This module introduces you to secure development models, standards, and guidelines that provide you with a structure for reducing risk from application security vulnerabilities.

They alleviate the need for you to become a security expert, and provide an efficient way to maximize application security.

As you learn about secure development, keep these important points in mind:

First, hacker methods and attacks are continuously evolving, and models, standards, and guidelines are updated regularly to address changing threat landscapes.

So, it is important that you stay up-to-date with application security trends to ensure that your application is resistant to current risks and attacks.

Second, be aware that the secure development practices listed do not make your application "hacker proof" or free from all vulnerabilities.

Models and guidelines highlight the key risks and attacks to your application at a given point in time, but your application can undoubtedly be exposed to new risks.

On Screen Text

Overview of Secure Development Models, Standards, and Guidelines

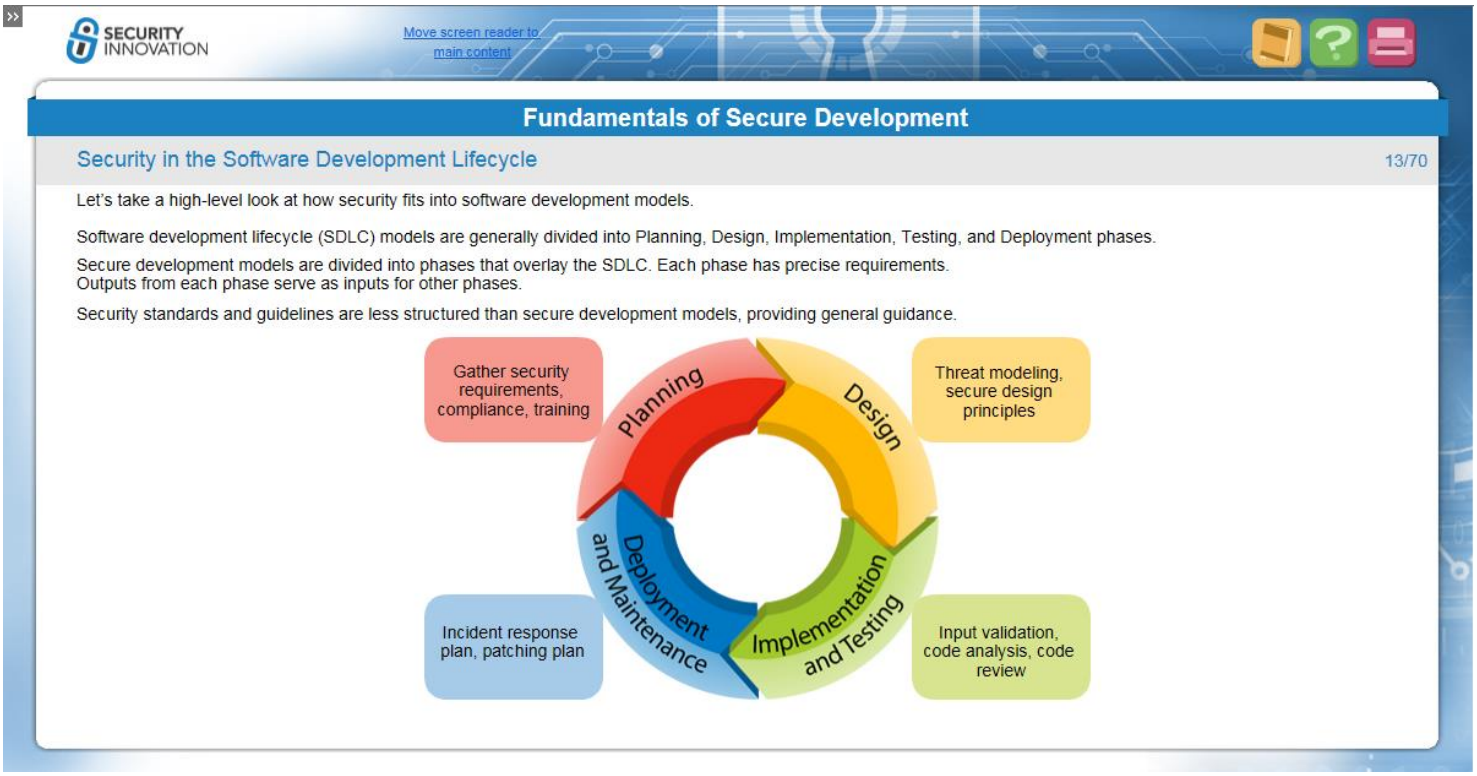
This module introduces you to the following secure development models, standards, and guidelines that provide a fast track to improving application security:

- Open Web Application Security Project (OWASP) Top 10
- Microsoft Security Development Lifecycle (SDL)
- CWE/SANS Top 25 Most Dangerous Software Errors
- Payment Application Data Security Standard (PA-DSS)
- Cloud Security Alliance (CSA) Notorious Nine Report
- Cloud Security Alliance (CSA) Top 10 Big Data Security and Privacy Challenges
- OpenSAMM - Software Assurance Maturity Model (SAMM)
- Building Security In Maturity Model (BSIMM)
- Comprehensive Lightweight Application Security Process (CLASP)
- The Common Criteria for Information Technology Security Evaluation (CC)

Important points about secure development include the following:

- Hacker methods and attacks are constantly evolving, as are the security techniques to address them.
- These security models, standards, and guidelines greatly improve your application security posture, but they do not make your application “hacker proof”.

Security in the Software Development Lifecycle



Narrations

Software development models such as Waterfall, Spiral, and Agile are generally divided into Planning, Design, Implementation, Testing, and Deployment phases.

This is sometimes called the software development lifecycle (SDLC).

Secure development models are similarly divided into phases that overlay your existing software development model.

Each phase describes the tasks that must be executed in each software development phase to minimize your application's exposure risk.

Secure development models are precise in their requirements.

The outputs from each phase serve as input for other phases in the model.

In comparison, standards and guidelines are less structured, providing an overview of

quality baselines to achieve, vulnerabilities to be aware of, and coding patterns to follow

or avoid.

On Screen Text

Security in the Software Development Lifecycle

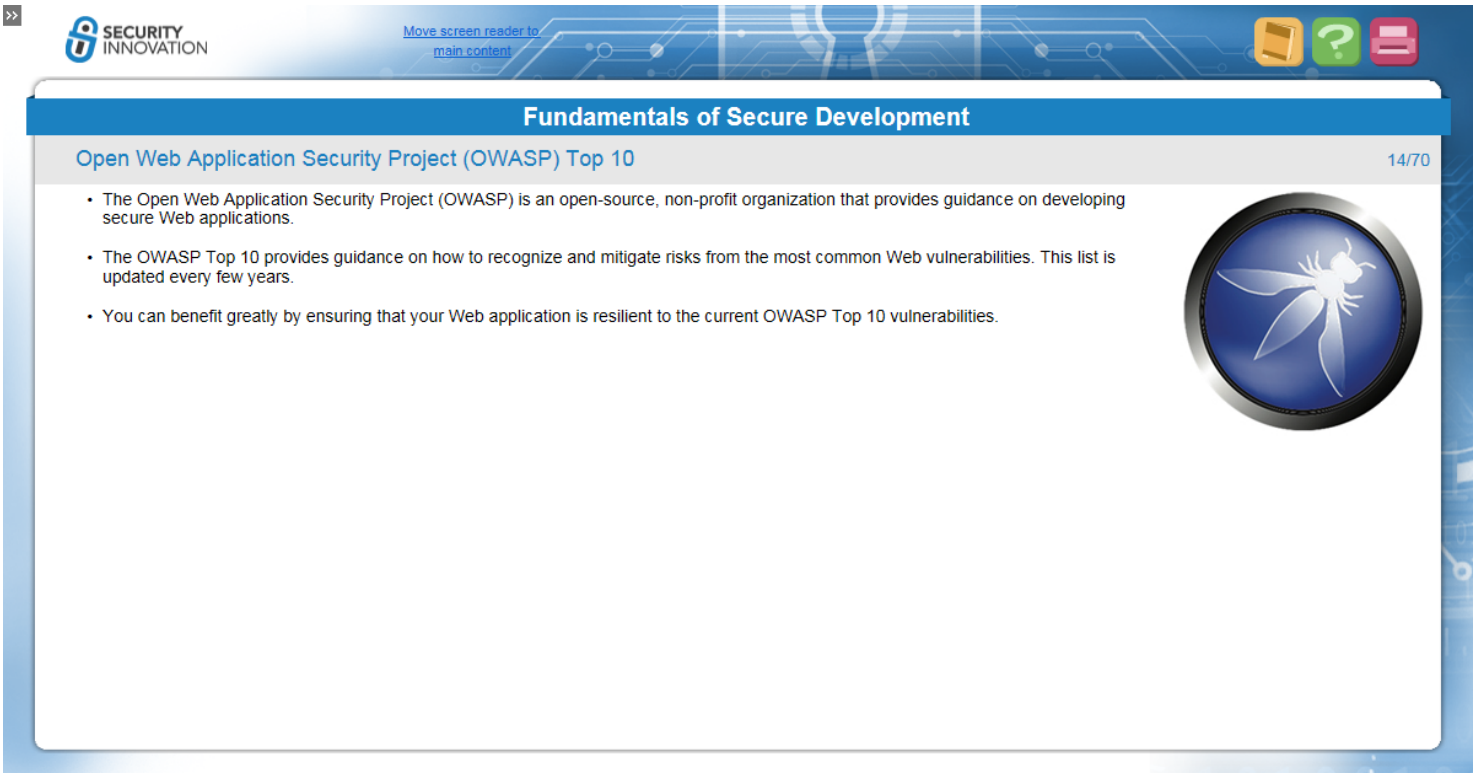
Let's take a high-level look at how security fits into software development models.

Software development lifecycle (SDLC) models are generally divided into Planning, Design, Implementation, Testing, and Deployment phases.

Secure development models are divided into phases that overlay the SDLC. Each phase has precise requirements. Outputs from each phase serve as inputs for other phases.

Security standards and guidelines are less structured than secure development models, providing general guidance.

Open Web Application Security Project (OWASP) Top 10



Narrations

The Open Web Application Security Project, or OWASP, is an open-source,

non-profit organization that provides guidance on secure Web software

development, testing frameworks, and tools.

Perhaps the most well-known of their offerings is the OWASP Top 10 list, which describes the most common

Web application attacks and vulnerabilities, along with example code vulnerabilities, example attacks, and

guidance on mitigating the risk.

OWASP Top 10 is updated every few years to help you focus on vulnerabilities that pose the most current risk.

As a developer, you benefit greatly by ensuring that your Web application is resilient to the current OWASP Top 10 vulnerabilities.

However, as with any security model, standard, or guideline, following OWASP Top 10 does not necessarily eliminate all possible risks.

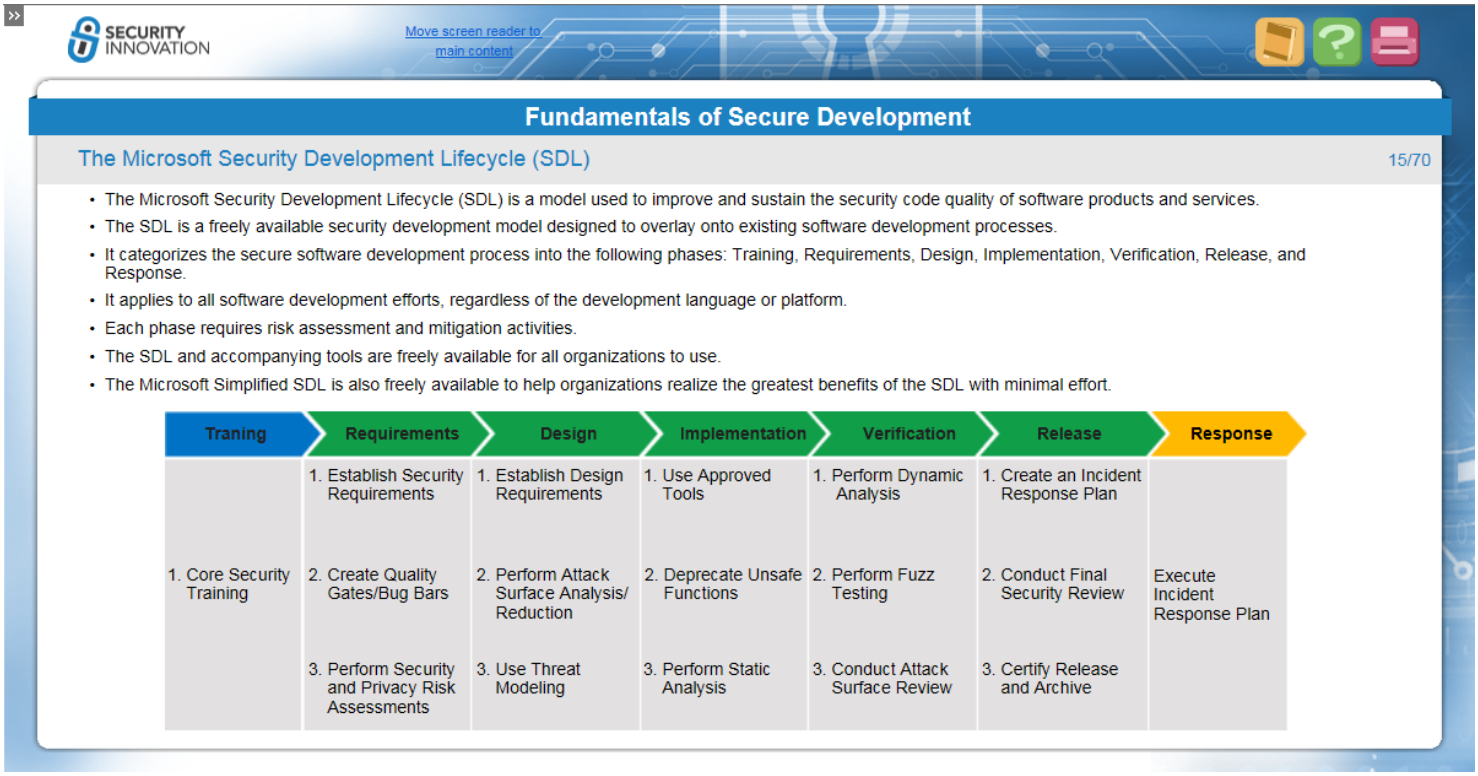
On Screen Text

Open Web Application Security Project (OWASP) Top 10

- The Open Web Application Security Project (OWASP) is an open-source, non-profit organization that provides guidance on developing secure Web applications.
- The OWASP Top 10 provides guidance on how to recognize and mitigate risks from the most common Web vulnerabilities. This list is updated every few years.

You can benefit greatly by ensuring that your Web application is resilient to the current OWASP Top 10 vulnerabilities.

The Microsoft Security Development Lifecycle (SDL)



Narrations

Microsoft developed the Security Development Lifecycle (SDL) to improve and sustain the overall security code quality across all of its products, services, and line-of-business applications.

The SDL is a freely available security development model designed to overlay onto existing software development processes.

At its core, it categorizes the secure software development process into the following phases:

Training, Requirements, Design, Implementation, Verification, Release, and Response.

The Microsoft SDL yields security benefits to any software development team developing in any language on any platform.

In each SDL phase, development teams must perform risk assessment and mitigation activities.

For example, in the training phase, the SDL requires that each software development team member in a technical role attend at least one security training class each year.

In the Implementation phase, it requires that developers use static code analysis tools, which will be discussed later in this course.

The SDL requires participation of the entire software development staff, including program managers, developers, and test teams.

This contrasts with other models, standards, and guidelines, which tend to focus strictly on the developer aspects of developing secure software.

Microsoft also publishes a set of tools to help software development teams implement and

execute the SDL process more efficiently. Example tools include a threat-modeling tool

that helps teams identify threats early in the Design phase, and a fault-injection tool to aid

with security testing efforts in the Verification phase.

Microsoft has also published the Simplified SDL, a lightweight framework version of the SDL available as a spreadsheet,

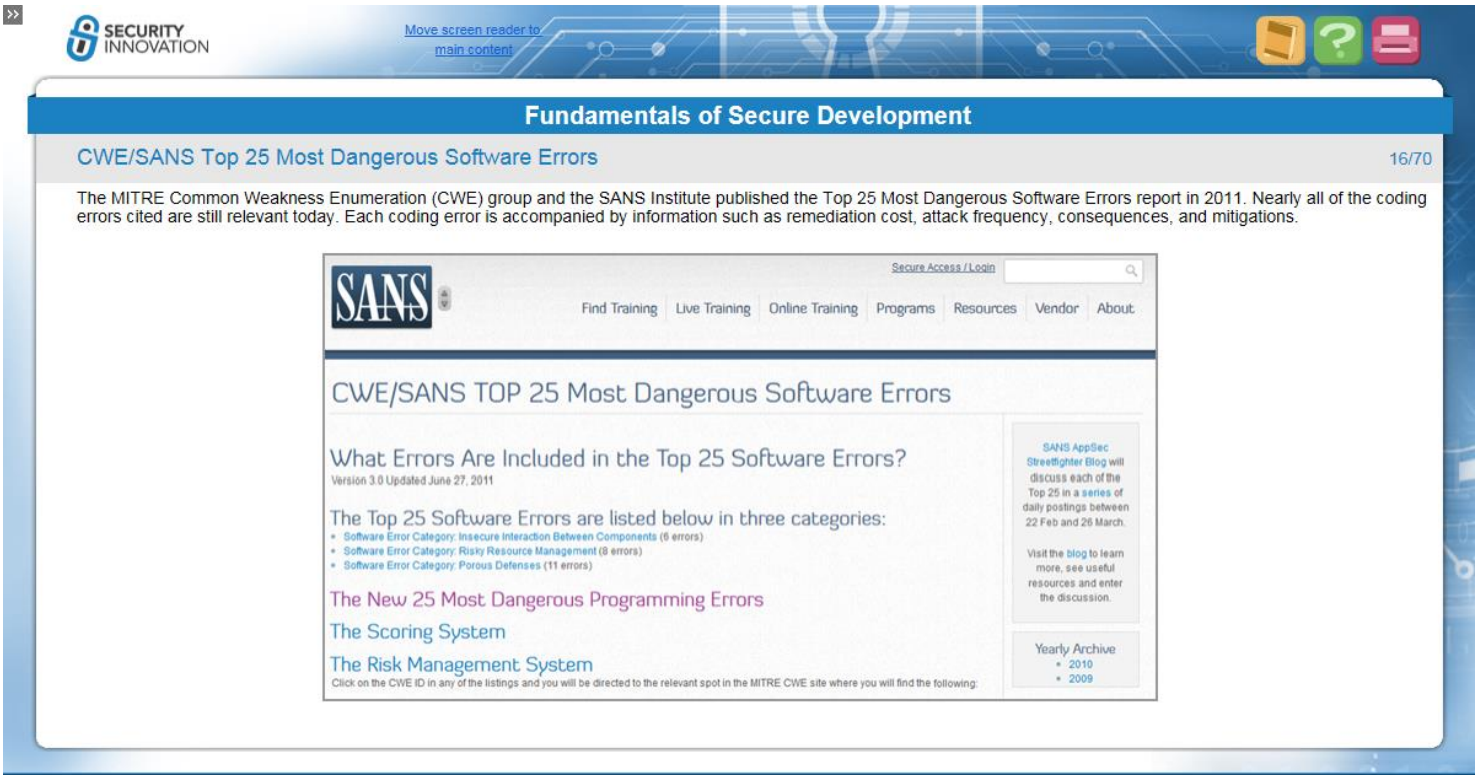
to help organizations realize the greatest benefits of the SDL with minimal effort.

On Screen Text

The Microsoft Security Development Lifecycle (SDL)

- The Microsoft Security Development Lifecycle (SDL) is a model used to improve and sustain the security code quality of software products and services.
- The SDL is a freely available security development model designed to overlay onto existing software development processes.
- It categorizes the secure software development process into the following phases: Training, Requirements, Design, Implementation, Verification, Release, and Response.
- It applies to all software development efforts, regardless of the development language or platform.
- Each phase requires risk assessment and mitigation activities.
- The SDL and accompanying tools are freely available for all organizations to use.
- The Microsoft Simplified SDL is also freely available to help organizations realize the greatest benefits of the SDL with minimal effort.

CWE/SANS Top 25 Most Dangerous Software Errors



Narrations

The MITRE Common Weakness Enumeration (CWE) group and 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.

Although the last report was published in 2011, nearly all of the weaknesses cited are still relevant today.

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

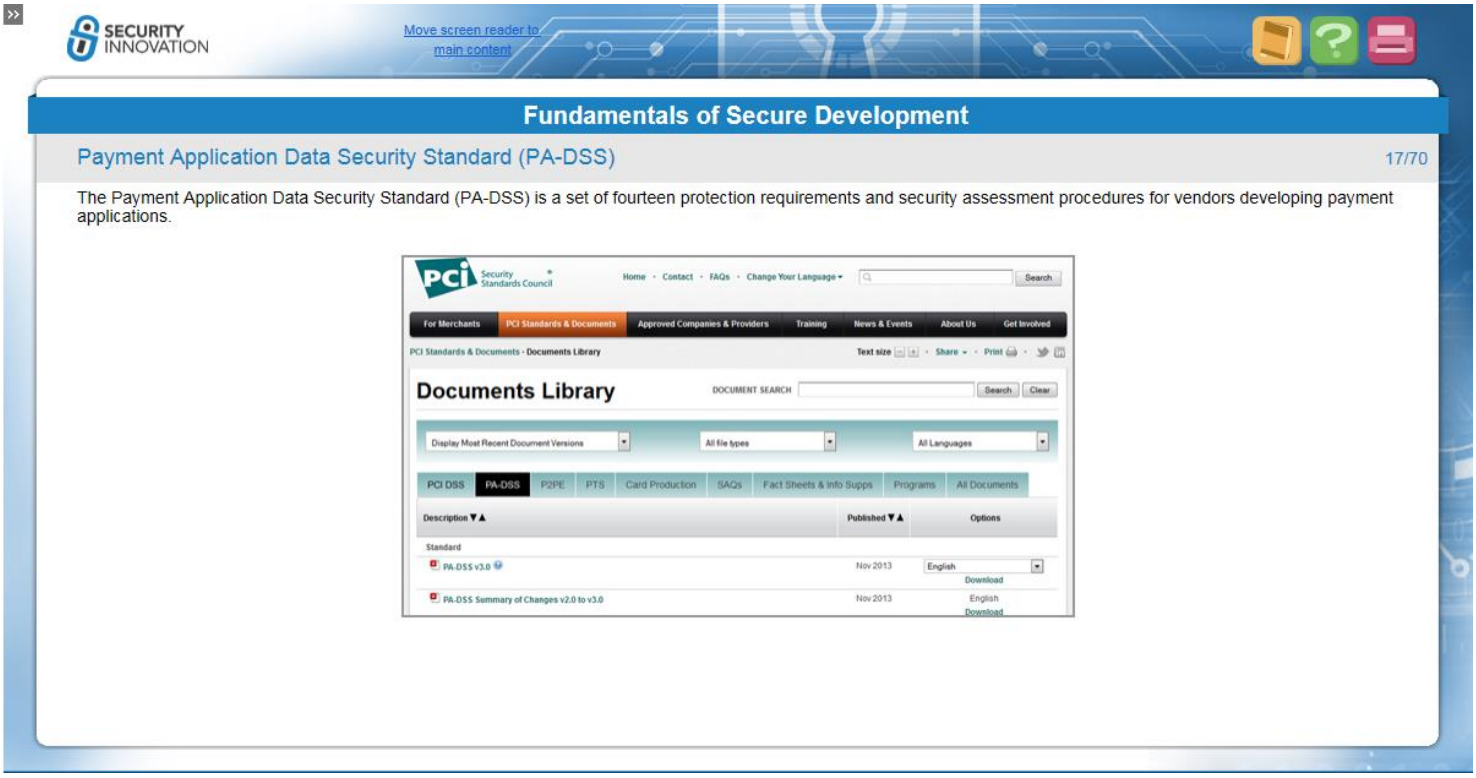
On Screen Text

CWE/SANS Top 25 Most Dangerous Software Errors

The MITRE Common Weakness Enumeration (CWE) group and the SANS Institute published the Top 25 Most Dangerous

Software Errors report in 2011. Nearly all of the coding errors cited are still relevant today. Each coding error is accompanied by information such as remediation cost, attack frequency, consequences, and mitigations.

Payment Application Data Security Standard (PA-DSS)



Narrations

In addition to the security and privacy requirements specified in the Payment Card Industry Data Security Standard (PCI DSS),

the Payment Card Industry Security Standards Council also created a Payment Application Data Security Standard (PA-DSS) for software vendors who develop payment applications.

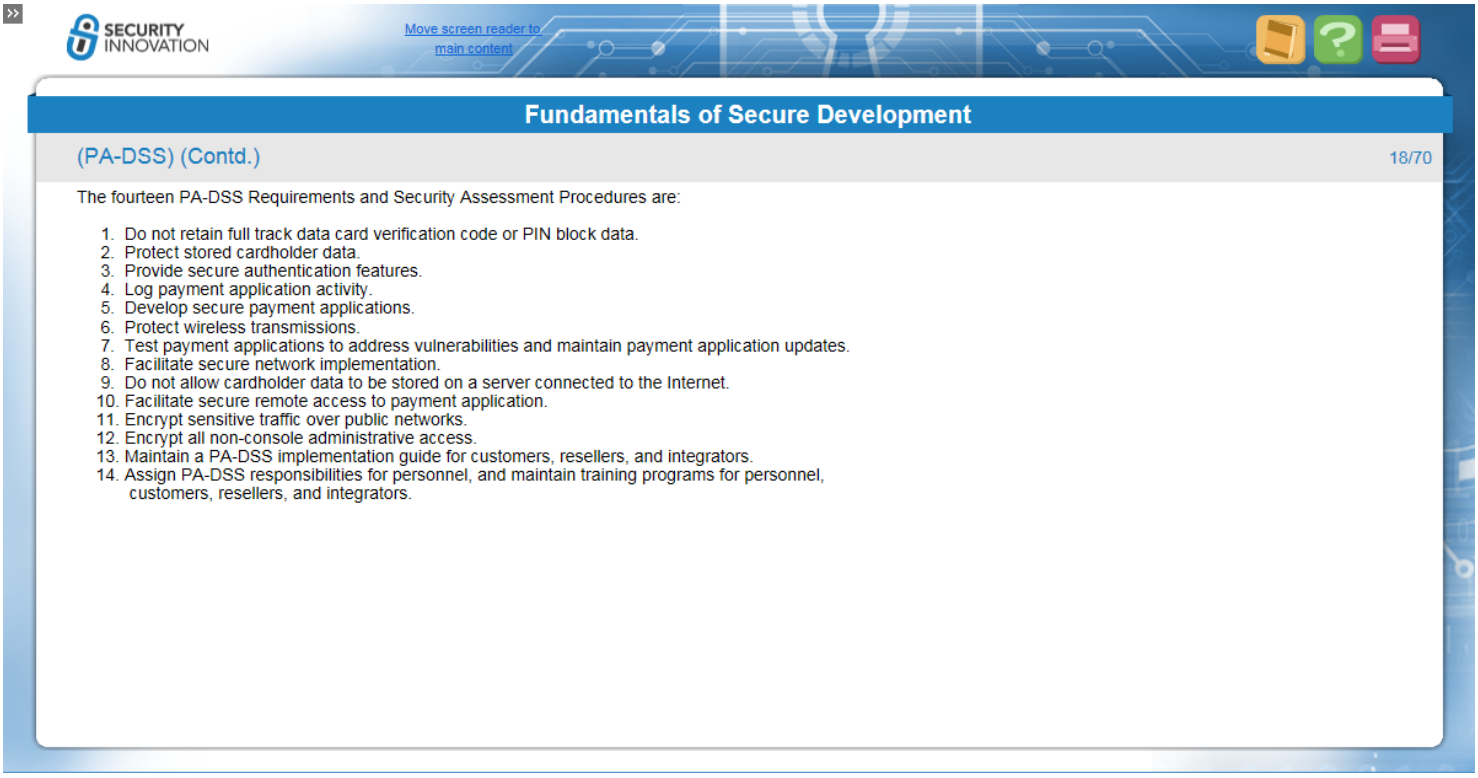
The PA DSS has fourteen requirements, which are listed on the next screen.

On Screen Text

Payment Application Data Security Standard (PA-DSS)

The Payment Application Data Security Standard (PA-DSS) is a set of fourteen protection requirements and security assessment procedures for vendors developing payment applications.

(PA-DSS) (Contd.)



Narrations

The fourteen PA-DSS requirements are listed here.

On Screen Text

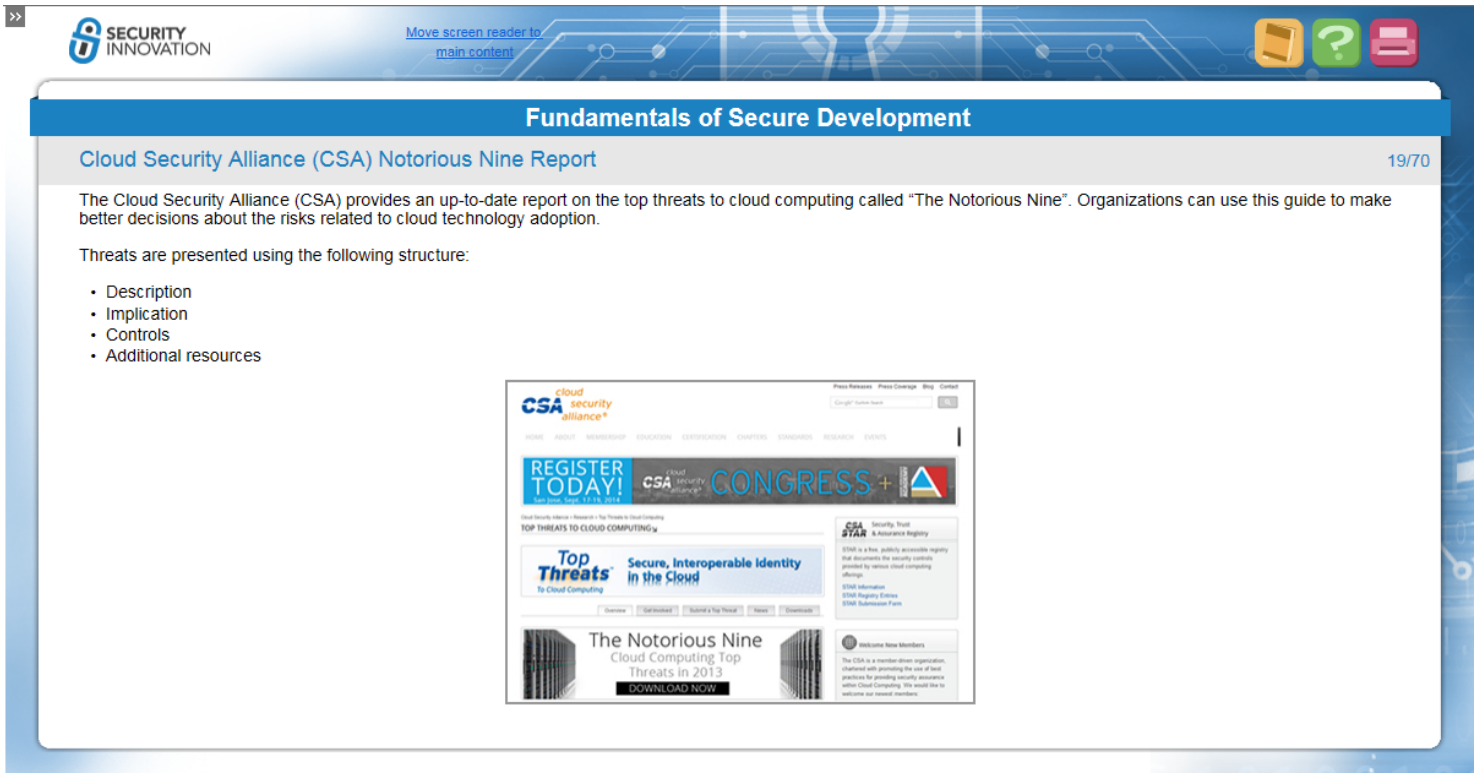
(PA-DSS) (Contd.)

The fourteen PA-DSS Requirements and Security Assessment Procedures are:

1. Do not retain full track data card verification code or PIN block data.
2. Protect stored cardholder data.
3. Provide secure authentication features.
4. Log payment application activity.
5. Develop secure payment applications.
6. Protect wireless transmissions.
7. Test payment applications to address vulnerabilities and maintain payment application updates.
8. Facilitate secure network implementation.
9. Do not allow cardholder data to be stored on a server connected to the Internet.
10. Facilitate secure remote access to payment application.
11. Encrypt sensitive traffic over public networks.
12. Encrypt all non-console administrative access.
13. Maintain a PA-DSS implementation guide for customers, resellers, and integrators.

Assign PA-DSS responsibilities for personnel, and maintain training programs for personnel, customers, resellers, and integrators.

Cloud Security Alliance (CSA) Notorious Nine Report



Narrations

The Cloud Security Alliance publishes the Notorious Nine, which is an up-to-date report on the top nine threats in cloud computing based on expert consensus and survey.

Organizations can use this guide to make better decisions about the risks related to cloud technology adoption.

For each threat, this report describes threat implications, controls to reduce risk, and external links to additional resources. Many, but not all, of the Notorious Nine have direct implications for application security, and development teams writing applications for cloud deployment can benefit greatly by reading this report for the latest cloud computing threats.

On Screen Text

Cloud Security Alliance (CSA) Notorious Nine Report

The Cloud Security Alliance (CSA) provides an up-to-date report on the top threats to cloud computing called "The Notorious Nine". Organizations can use this guide to make better decisions about the risks related to cloud technology adoption.

Threats are presented using the following structure:

- Description
- Implication
- Controls
- Additional resources

Cloud Security Alliance (CSA) Top 10 Big Data Security and Privacy Challenges

Narrations

In 2012, the Cloud Security Alliance published the Top 10 Big Data Security

and Privacy Challenges report, which discusses issues important to

application developers.

The report is freely available and can be downloaded directly from the CSA Website.

On Screen Text

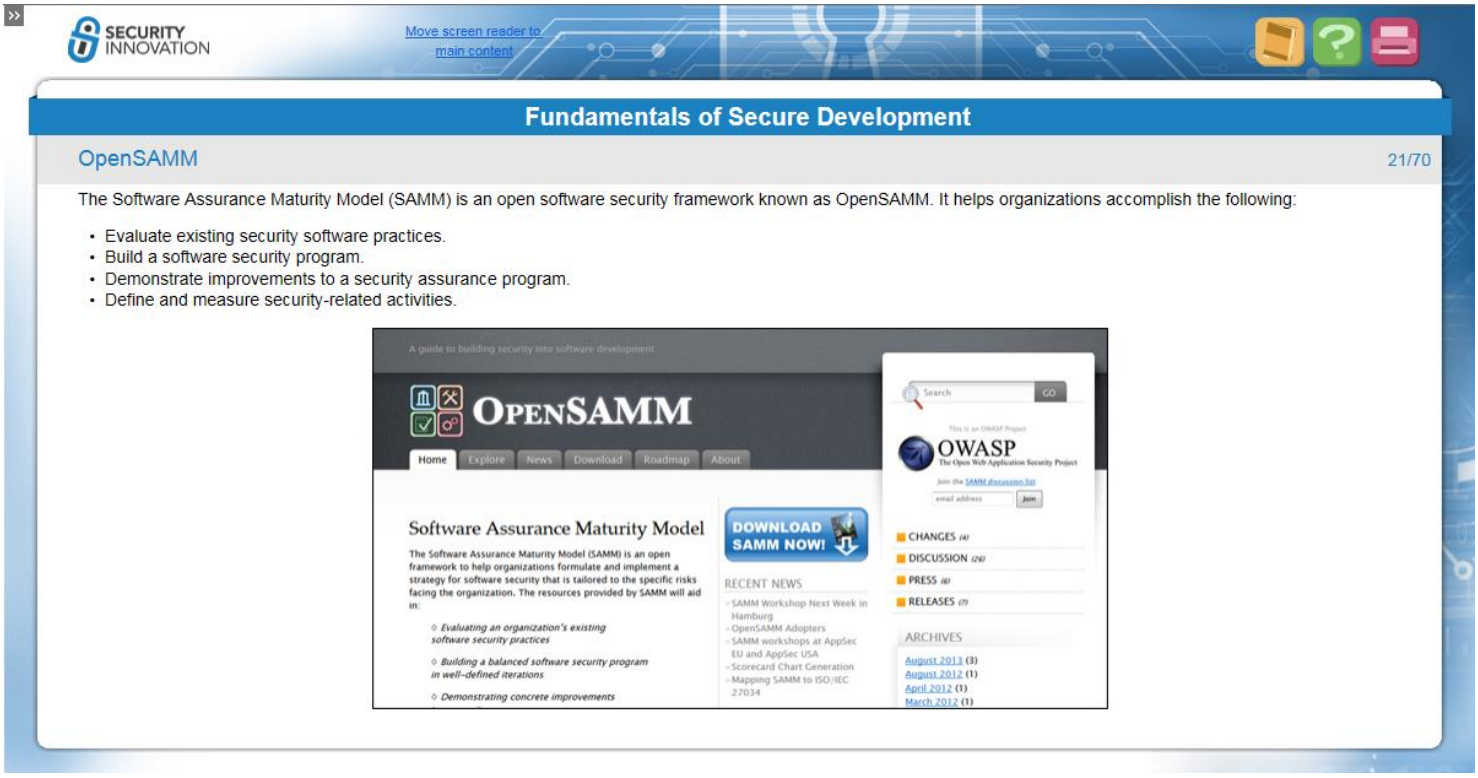
Cloud Security Alliance (CSA) Top 10 Big Data Security and Privacy Challenges

In 2012, the CSA published the Top 10 Big Data Security and Privacy Challenges report. It covers the following issues important to application developers:

1. Secure computations in distributed programming frameworks
2. Security best practices for non-relational data stores
3. Secure data storage and transactions logs
4. Endpoint input validation/filtering
5. Real-time security monitoring
6. Scalable and composable privacy-preserving data mining and analytics
7. Cryptographically enforced data-centric security
8. Granular access control

9. Granular audits
10. Data provenance

OpenSAMM



Narrations

The Software Assurance Maturity Model (SAMM) is an open software security framework, commonly called OpenSAMM.

It 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.

On Screen Text

OpenSAMM

The Software Assurance Maturity Model (SAMM) is an open software security framework known as OpenSAMM. It helps organizations accomplish the following:

- Evaluate existing security software practices.
- Build a software security program.
- Demonstrate improvements to a security assurance program.

Define and measure security-related activities.

Building Security In Maturity Model (BSIMM)

Narrations

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 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.

The Governance domain helps organizations manage and measure software activities related to internal strategy and metrics, compliance and policy, and training.

The Intelligence domain provides proactive security guidance and describes organizational threat assessment and modeling.

The SSDL domain discusses essential software security practices to integrate into your

own software development process. The Deployment domain provides practices that

are more aligned with traditional network security and software maintenance, such as

penetration testing, configuration, and vulnerability management.

On Screen Text

Building Security In Maturity Model (BSIMM)

The Building Security in Maturity Model (BSIMM) analyzes and shares the security best practices of real-world companies such as Adobe, Microsoft, and PayPal.

BSIMM provides a free software security framework (SSF) that development teams can follow:

Comprehensive Lightweight Application Security Process (CLASP)

Narrations

The Comprehensive Lightweight Application Security Process, or CLASP, is another product of OWASP. CLASP offers a well-defined approach to handling security issues and concerns in the beginning of the software development life cycle. Organizations can benefit from CLASP resources such as a list of key security best practices, fundamental security goals and principles, activities to improve your secure software development process, secure software engineering roadmaps, and downloadable documents for development teams.

CLASP is an inactive OWASP project, but organizations can still benefit by learning about its security resources, development processes, and documentation.

On Screen Text

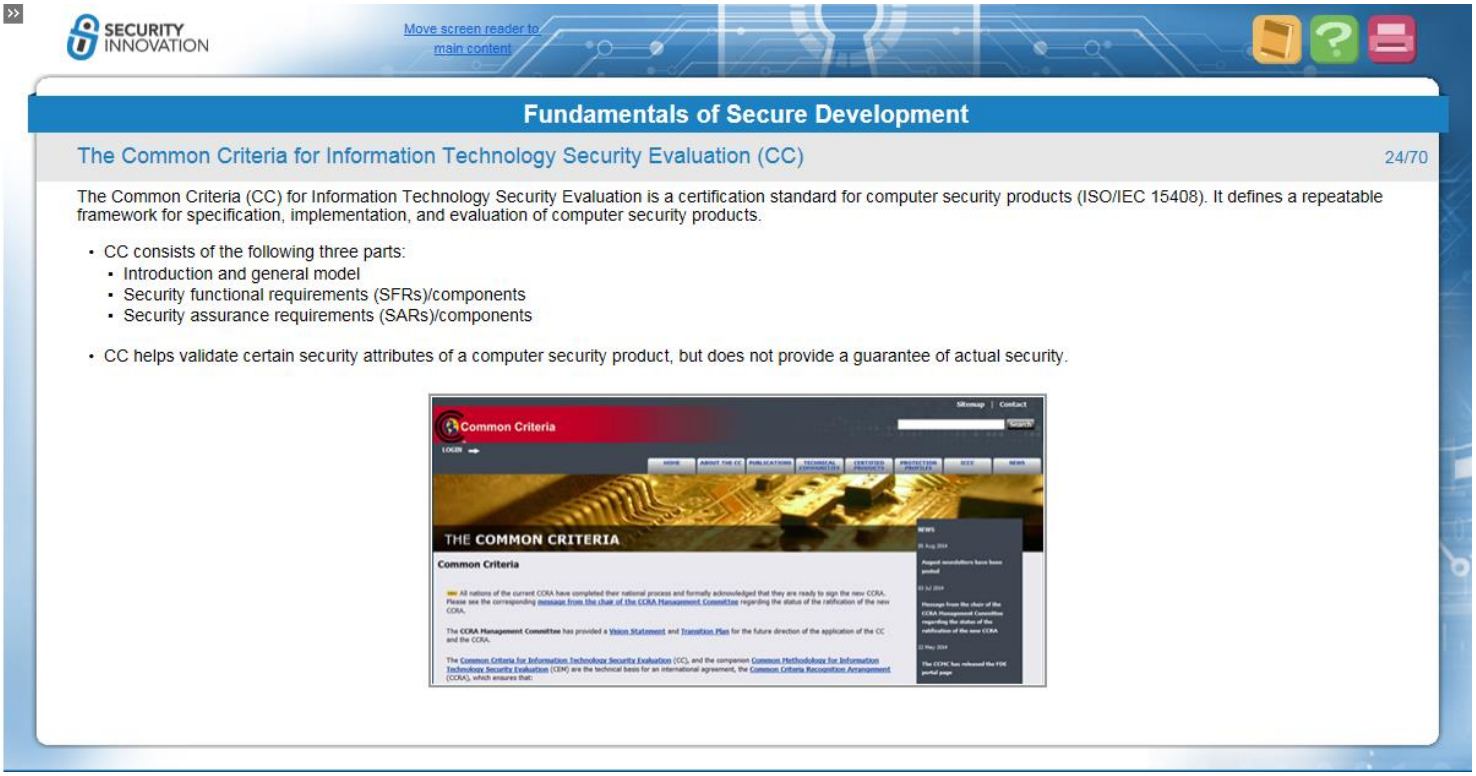
Comprehensive Lightweight Application Security Process (CLASP)

The Comprehensive Lightweight Application Security Process (CLASP) is an OWASP project. It defines a process to handle security issues and concerns early in the software development lifecycle. CLASP provides the following key resources:

- Security best practices
- Fundamental security goals and principles
- Activities to improve your secure software development process
- Secure software engineering roadmaps
- A downloadable book and searchable vulnerability checklist for use by development teams

Note: CLASP is an inactive OWASP project, but organizations can still benefit by learning about its security resources, development processes, and documentation. For more information on CLASP, visit the CLASP homepage.

The Common Criteria for Information Technology Security Evaluation (CC)



Narrations

The Common Criteria for Information Technology Security Evaluation (often abbreviated as CC) defines an international standard and repeatable framework for the specification, implementation, and evaluation of a computer security solution.

High assurance customers such as Federal Government agencies use the Common Criteria as a basis for evaluation.

The Common Criteria publication is provided in three parts.

The first includes an introduction to basic concepts and principles, how to express IT security objectives, and a general evaluation model.

The second part presents a standard way of expressing security requirements as a set of security functional requirements or components.

The third part presents a set of assurance components to describe a standard way of expressing security assurance requirements.

Note that the Common Criteria standard does not guarantee security.

Instead, it provides a mechanism to describe and validate security attributes of a product in a rigorous, standard, and repeatable manner.

On Screen Text

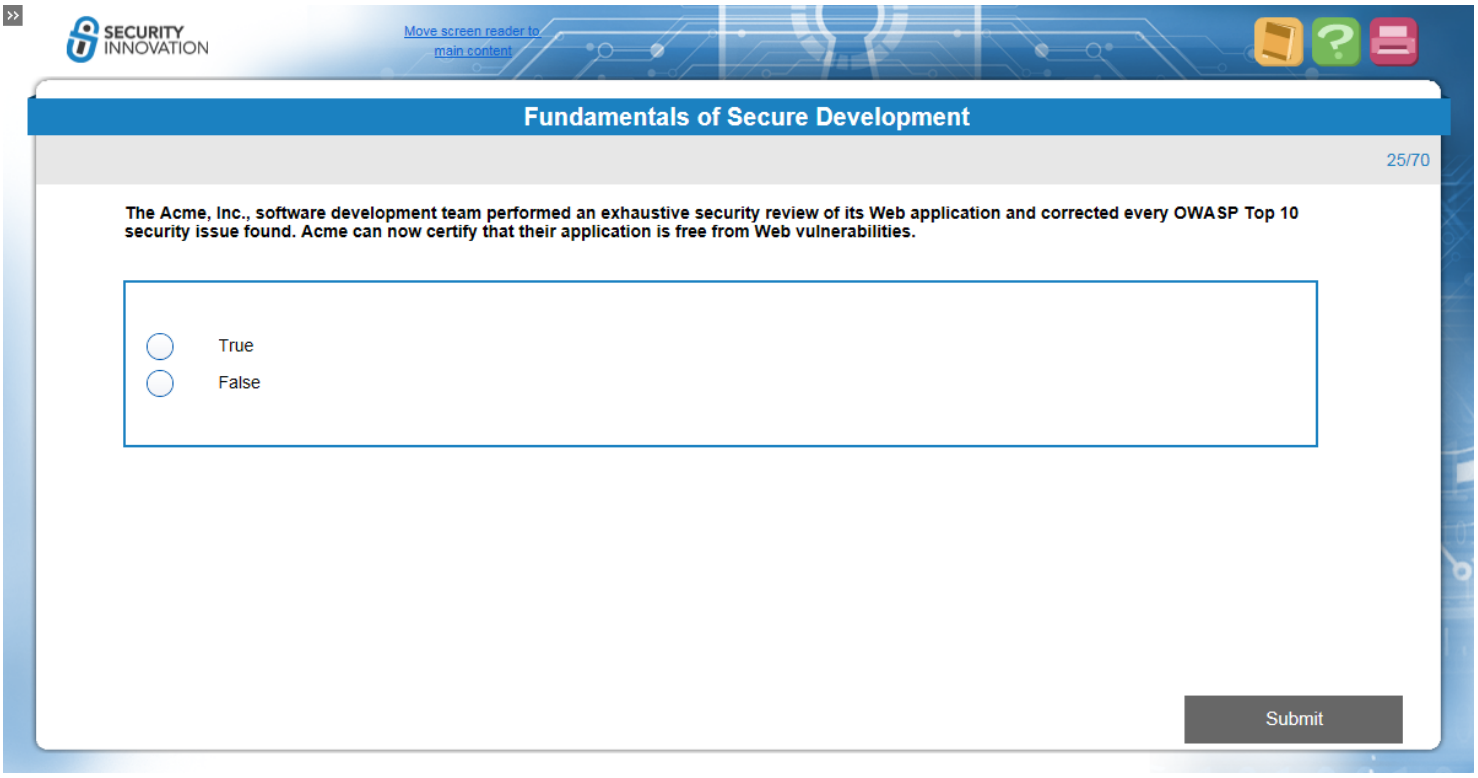
The Common Criteria for Information Technology Security Evaluation (CC)

The Common Criteria (CC) for Information Technology Security Evaluation is a certification standard for computer security products (ISO/IEC 15408). It defines a repeatable framework for specification, implementation, and evaluation of computer security products.

- CC consists of the following three parts:
  - Introduction and general model
  - Security functional requirements (SFRs)/components
  - Security assurance requirements (SARs)/components

CC helps validate certain security attributes of a computer security product, but does not provide a guarantee of actual security.

Knowledge Check



On Screen Text

Knowledge Check

The Acme, Inc., software development team performed an exhaustive security review of its Web application and corrected every OWASP Top 10 security issue found. Acme can now certify that their application is free from Web vulnerabilities.

- True
- False

Knowledge Check

On Screen Text

Knowledge Check

Which statement below is incorrect regarding the Microsoft Security Development Lifecycle (SDL)?

- It is a secure development process.
- It applies only to Microsoft development technologies.
- Its framework and tools are freely available.
- It helps sustain the security of products and services.

Module Summary

On Screen Text

Module Summary

In this module, you learned about the following secure development models, standards, and guidelines that help you efficiently improve your application security posture:

- Open Web Application Security Project (OWASP) Top 10
- Microsoft Security Development Lifecycle (SDL)
- CWE/SANS Top 25 Most Dangerous Software Errors
- Payment Application Data Security Standard (PA-DSS)
- Cloud Security Alliance (CSA) Notorious Nine Report
- Cloud Security Alliance (CSA) Top 10 Big Data Security and Privacy Challenges
- OpenSAMM - Software Assurance Maturity Model (SAMM)
- Building Security In Maturity Model (BSIMM)
- Comprehensive Lightweight Application Security Process (CLASP)
- The Common Criteria for Information Technology Security Evaluation (CC)

You learned that attacks and the techniques used to address them are constantly evolving, which requires you to stay current on the resources listed. You also learned that using these resources can greatly improve your application security posture, but it cannot make an application hacker proof or provide a guarantee of security.

Module Overview and Objectives

Narrations

In this module, we will discuss how to integrate secure application development practices into the four phases of the software development lifecycle.

We'll look at the specific security activities that correspond to each phase.

After completing this module, you will be able to identify security activities that correspond to each phase of the software development lifecycle.

You will also be able to identify key application security principles and secure coding principles.

On Screen Text

Module Overview and Objectives

Module Overview

In this module, we will discuss how to integrate secure application development practices into the four phases of the software development lifecycle. We will look at specific security activities that correspond to each phase.

Module Objectives

After completing this module, you will be able to:

- Identify security activities that correspond to each phase of the software development lifecycle.
- Identify key application security principles and secure coding principles.

Integrating Security in the Software Development Process

Narrations

We'll now look at security in the context of the general software development process At each phase, we will look at key risk identification and mitigation techniques that you can apply to your own software development process to improve the security of your applications.

On Screen Text

Integrating Security in the Software Development Process

Security must be integrated into each phase of the software development process.

Security in the Planning Phase

Narrations

Every software security project begins with the Planning phase.

In this phase, software development teams gather security requirements for the software that they are designing and implementing. In this phase, we will look at legal security requirements, customer security requirements, customer security requirements, and the appropriate security training for your software development team.

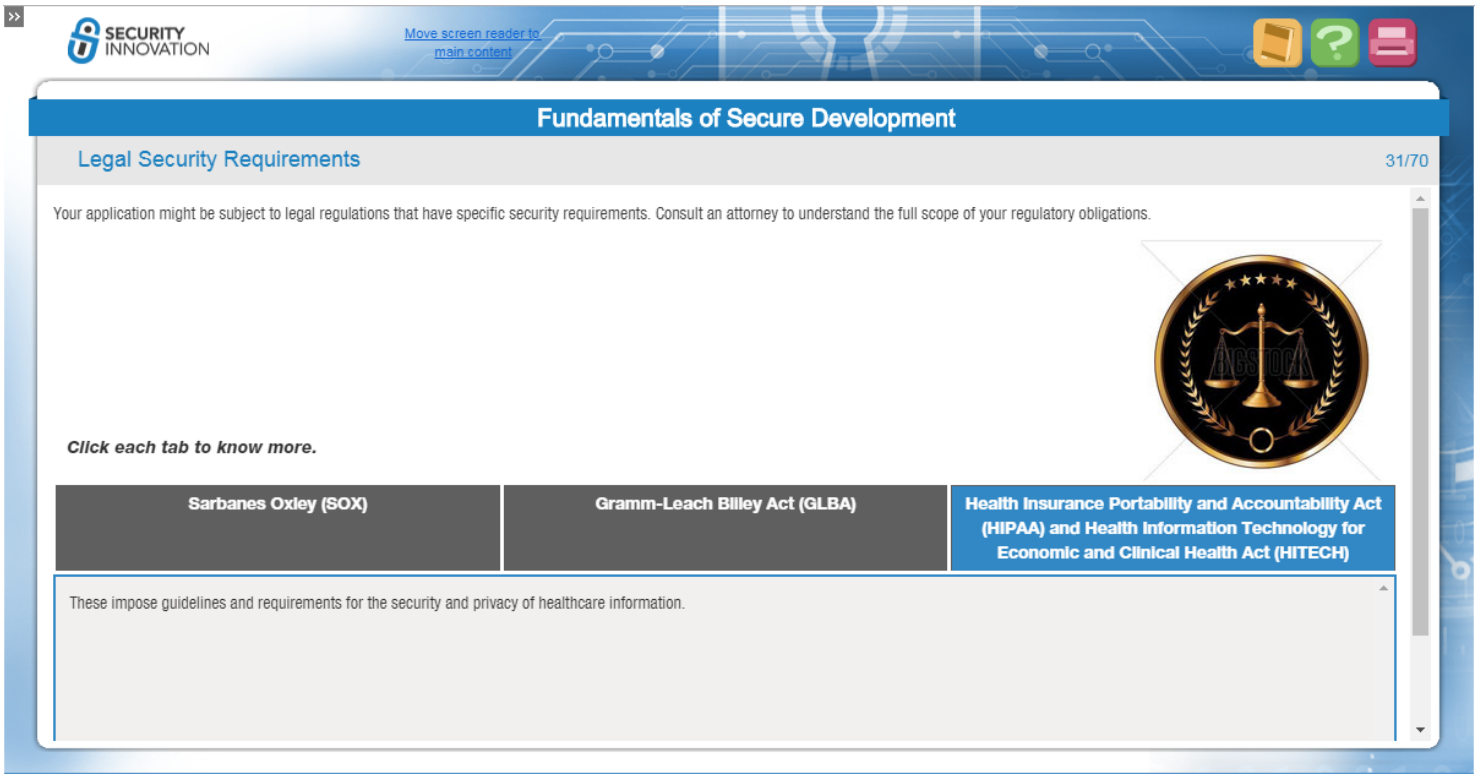
On Screen Text

Security in the Planning Phase

Security in the Planning phase includes the following:

- Legal security requirements
- Customer security requirements
- Training

Legal Security Requirements



Narrations

When gathering security requirements, you need to know if your application must comply with any US federal or state regulations, which often have explicit security requirements.

In addition, your application may be subject to regulations in other countries. Because laws,

regulations, and standards are not easily interpreted, it is recommended that you or your

organization consult an attorney to understand the full scope of your application's security

obligations.

You can use a spreadsheet to help track information related to legislative requirements.

For example, is your application going to work with financial data, take credit card data, or handle health care information?

Will it transact with the European Union, or with other countries?

Examples of security-related legislation in the US are shown here. Click each option to see examples of US security-related laws and regulations.

On Screen Text

Legal Security Requirements

Your application might be subject to legal regulations that have specific security requirements. Consult an attorney to understand the full scope of your regulatory obligations.

Click each option to see examples of US security laws and regulations.

Sarbanes Oxley (SOX)

Implemented in the wake of the Enron disaster, this law imposes severe penalties on publicly traded companies for exposing or falsifying financial data.

Gramm-Leach Bliley Act (GLBA)

This law includes provisions to protect consumers' personal financial information held by financial institutions and also imposes rules on safeguards.

Health Insurance Portability and Accountability Act (HIPAA) and Health Information Technology for Economic and Clinical Health Act (HITECH)

These impose guidelines and requirements for the security and privacy of healthcare information.

Legal Security Requirements (Cont.)

Narrations

On Screen Text

Legal Security Requirements (Cont.)

Payment Card Industry Data Security Standard (PCI-DSS) and Payment Application Data Security Standard (PADSS)

These comprise a comprehensive set of requirements for secure handling of payment card data.

California Senate Bill 1386 (SB 1386)

This law requires companies to send notices to California residents if personal data is exposed.

The Massachusetts Data Protection Law (201 CMR 17.00)

This law obligates businesses that handle or store personal information of any Massachusetts' resident to protect the privacy of that information.

Customer Security Requirements and Technical Requirements

Narrations

In addition to legal requirements, your customers might have specific security and technical requirements that you need to understand.

Ask questions about assets to be protected, compliance, quality of service, and data confidentiality requirements.

Microsoft has developed a template that can help your team develop a security and privacy questionnaire, which can be found on the Microsoft SDL Requirements homepage.

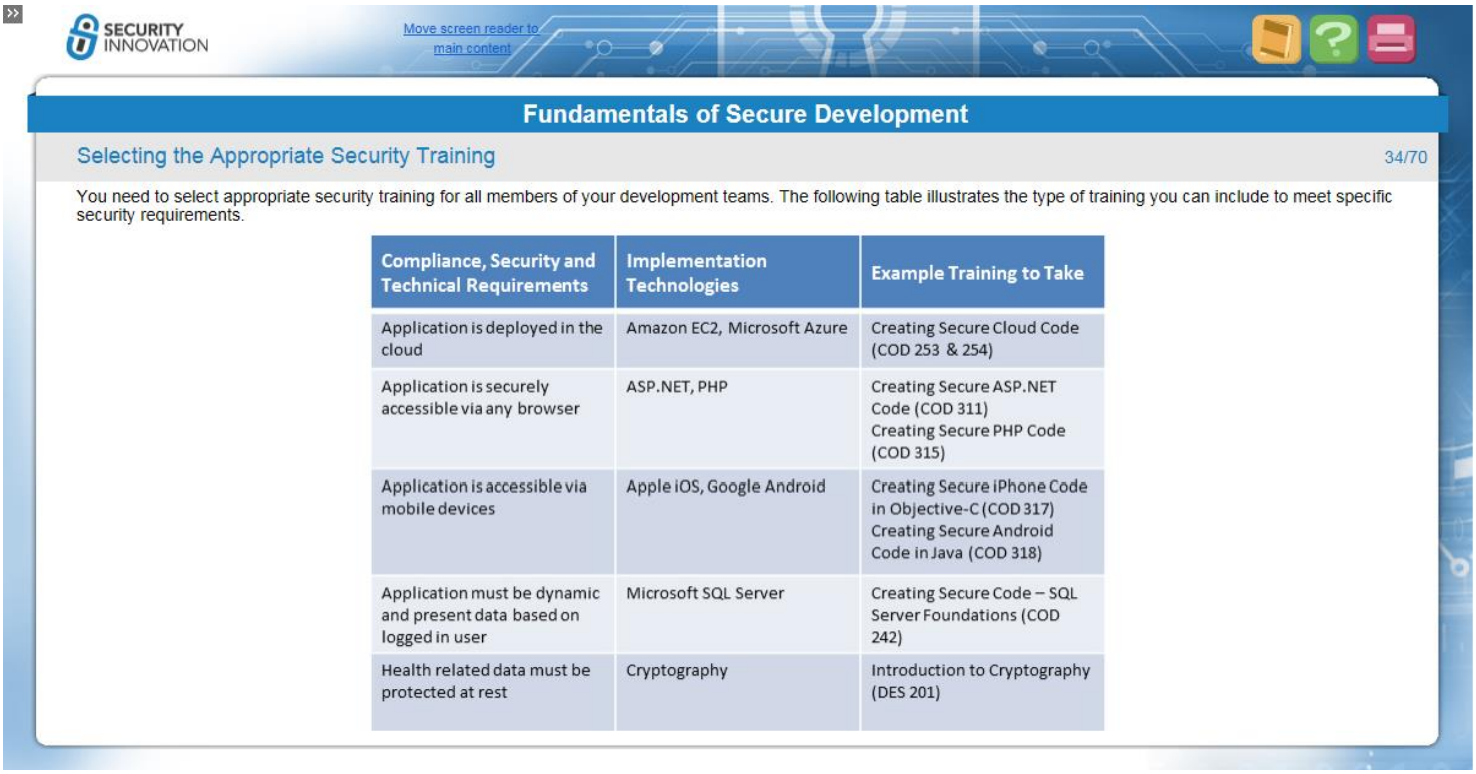
On Screen Text

Customer Security Requirements and Technical Requirements

As you gather customer security and technical requirements for your application, ask the following questions:

- What specific assets need to be protected?
- What are your compliance requirements?
- What are your quality of service requirements?
- What data is considered confidential?

Selecting the Appropriate Security Training



Narrations

Secure application development requires that each member of the software development team is well trained in security threats and mitigations.

But how do you select a specific training curriculum?

You should select training based on the technologies you

plan to use. After you have gathered legal and customer

requirements, you should have an idea of the

technologies you will use to meet those requirements.

For example, let's look at developing a cloud-based healthcare application that is

accessible via a Web browser or a mobile device. In the example, the first column of the

table shows that the application must be deployed in the cloud, be securely accessible via

any browser, and be accessible via mobile devices.

In addition, the application must be dynamic and must present data based on the user logged in, and health-related data must be protected at rest.

Some of the technologies you would select in order to meet these requirements are shown in the second column of the table. As shown in the third column, once you know the technologies you will use to develop your application, you can select appropriate training for security issues related to those technologies.

On Screen Text

Selecting the Appropriate Security Training

You need to select appropriate security training for all members of your development teams. The following table illustrates the type of training you can include to meet specific security requirements.

Security in the Design Phase

Narrations

The Design phase follows the Planning phase.

Early risk mitigation in the Design phase reduces the security cost and effort needed later in the lifecycle.

For the Design phase, we'll first discuss threat modeling, which helps you anticipate security risks and concerns.

Then, we'll look at overriding application security principles that serve as the foundation for designing and developing a secure application.

These principles include attack surface reduction, secure defaults, least privilege, defense in depth, compartmentalization, and policy compliance.

Applying these principles in the Design phase provides a jumpstart on security in the next phases of application development and deployment.

On Screen Text

Security in the Design Phase

- Threat modeling helps you anticipate security risks and concerns.
- The following application security principles provide a foundation for designing and developing a secure application:

- Attack surface reduction
- Secure defaults
- Least privilege
- Defense in depth
- Compartmentalization
- Policy compliance

Threat Modeling

Narrations

Design phase threat modeling is used to identify and evaluate potential application security issues.

There are three main approaches to threat modeling.

The first is attacker-centric, in which you anticipate what an attacker might do, and derive risks based on that perspective.

The second is software-centric, in which you identify potential attacks against each element of the software design.

The third approach is asset-centric, in which you examine the assets managed by an application,

such as sensitive information, or intellectual property, and derive risks based on potential threats

to those assets.

On Screen Text

Threat Modeling

Threat modeling is a structured approach for identifying potential risks to your application.

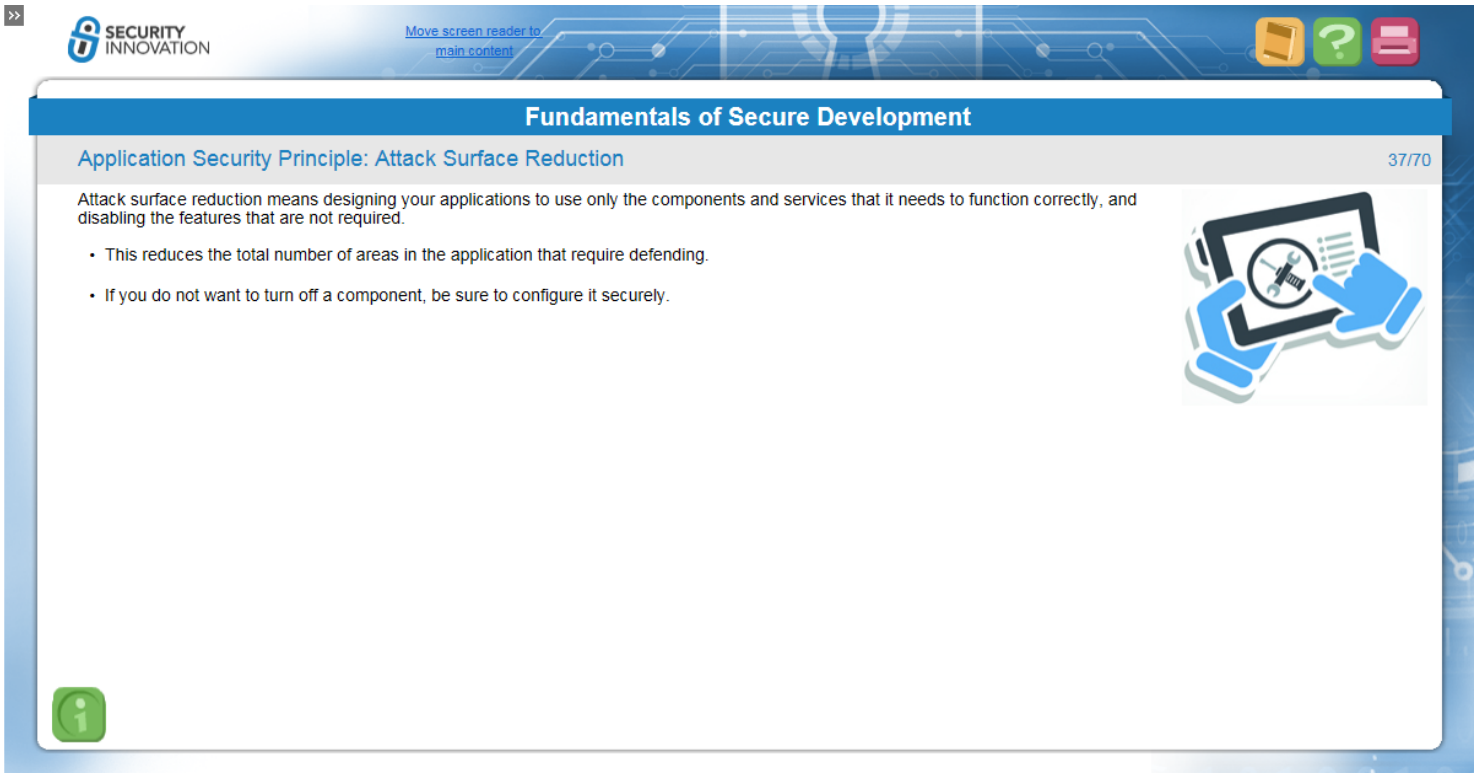
Following are the three general approaches to threat modeling:

- Attacker-centric
- Software-centric

- Asset-centric

For more information on threat modeling, see the Team Mentor article Create a Threat Model, and take the Security Innovation course “How to Create an Application Threat Model” (ENG 301).

Application Security Principle: Attack Surface Reduction



Narrations

Let's discuss the first application security principle, attack surface reduction.

To reduce your application's attack surface, design your application to use only the components and services needed for correct operation, and disable all other features. This reduces the total number of areas in your application that require defending.

Remember, you must defend all possible ways to compromise a system, but attackers only have to find one weakness that was overlooked. If you do not want to turn off certain features, you can reduce the attack surface by using more secure configurations of those features, rather than their insecure defaults. This leads to the next application security principle: secure defaults.

On Screen Text

Application Security Principle: Attack Surface Reduction

Attack surface reduction means designing your applications to use only the components and services that it needs to function correctly, and disabling the features that are not required.

- This reduces the total number of areas in the application that require defending.
- If you do not want to turn off a component, be sure to configure it securely.

Application Security Principle: Secure Defaults

Narrations

Secure defaults means do not leave insecure settings in place for the components and services of your application.

Instead, design your application to use the most secure settings by default.

Users are often unaware of certain features, or they do not know how to configure them to the most secure settings.

Designing an application with secure defaults reduces the likelihood that an attacker can exploit insecure settings left in place by users.

For example, communicate with encrypted communications instead of unprotected TCP.

It might not entirely eliminate attacks against your applications, but it removes one attack vector.

On Screen Text

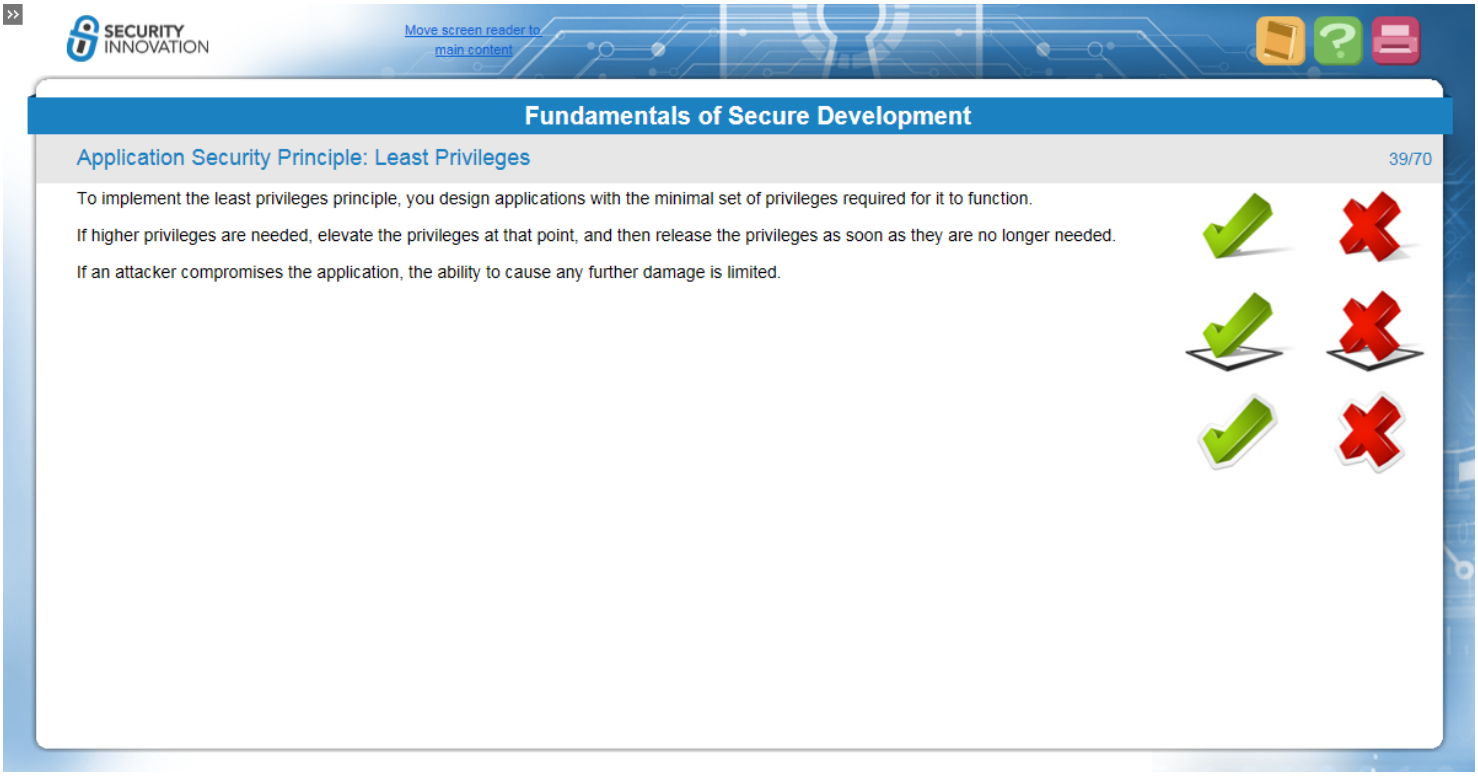
Application Security Principle: Secure Defaults

Secure defaults means designing applications in such a way that the most secure settings are used by default.

Attackers often exploit common default configuration settings that are forgotten or weak. Apply the principle of secure defaults to mitigate this risk.

For example, communicate using encrypted communications instead of unprotected TCP.

Application Security Principle: Least Privileges



Narrations

In the least privileges principle, begin with the idea that all software can and will be compromised by a malicious user.

To reduce the impact of a compromise, applications should be designed using the minimal set of privileges required to function correctly.

If higher privileges are needed, elevate privileges at that point and then release those elevated privileges as soon as they are no longer needed.

By applying the least privileges principle, you limit the potential damage that can be caused by a malicious user who compromises the application.

On Screen Text

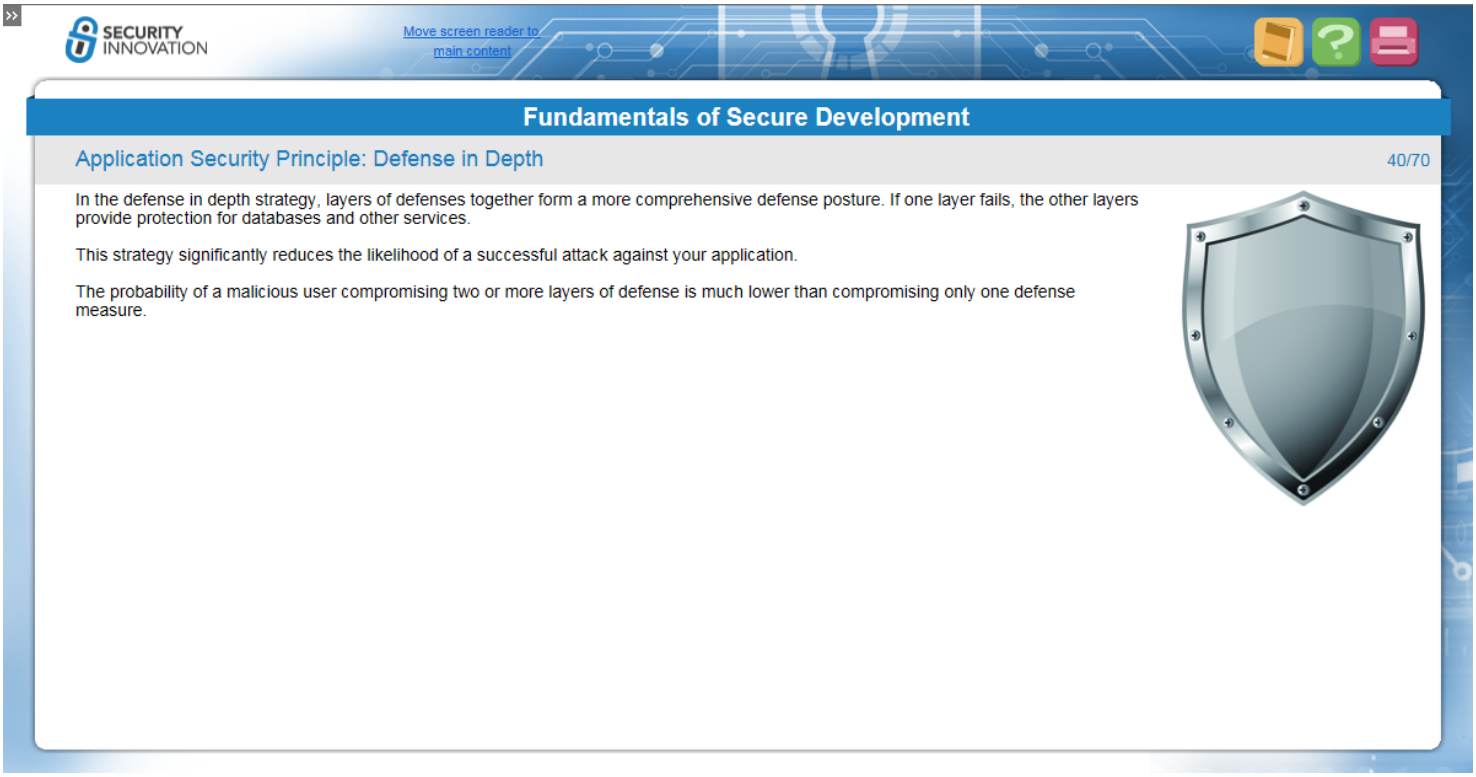
Application Security Principle: Least Privileges

To implement the least privileges principle, you design applications with the minimal set of privileges required for it to function.

If higher privileges are needed, elevate the privileges at that point, and then release the privileges as soon as they are no longer needed.

If an attacker compromises the application, the ability to cause any further damage is limited.

Application Security Principle: Defense in Depth



Narrations

At some point, all systems and defenses will fail. To mitigate this risk, implement the defense in depth strategy.

Layer a series of defenses to form a more comprehensive defense posture.

If one layer of defense fails, the other layers continue to provide protection.

Layering defenses reduces the chance of a successful attack.

It is much more difficult for an attacker to successfully compromise two or more layers of defense than to compromise only one.

On Screen Text

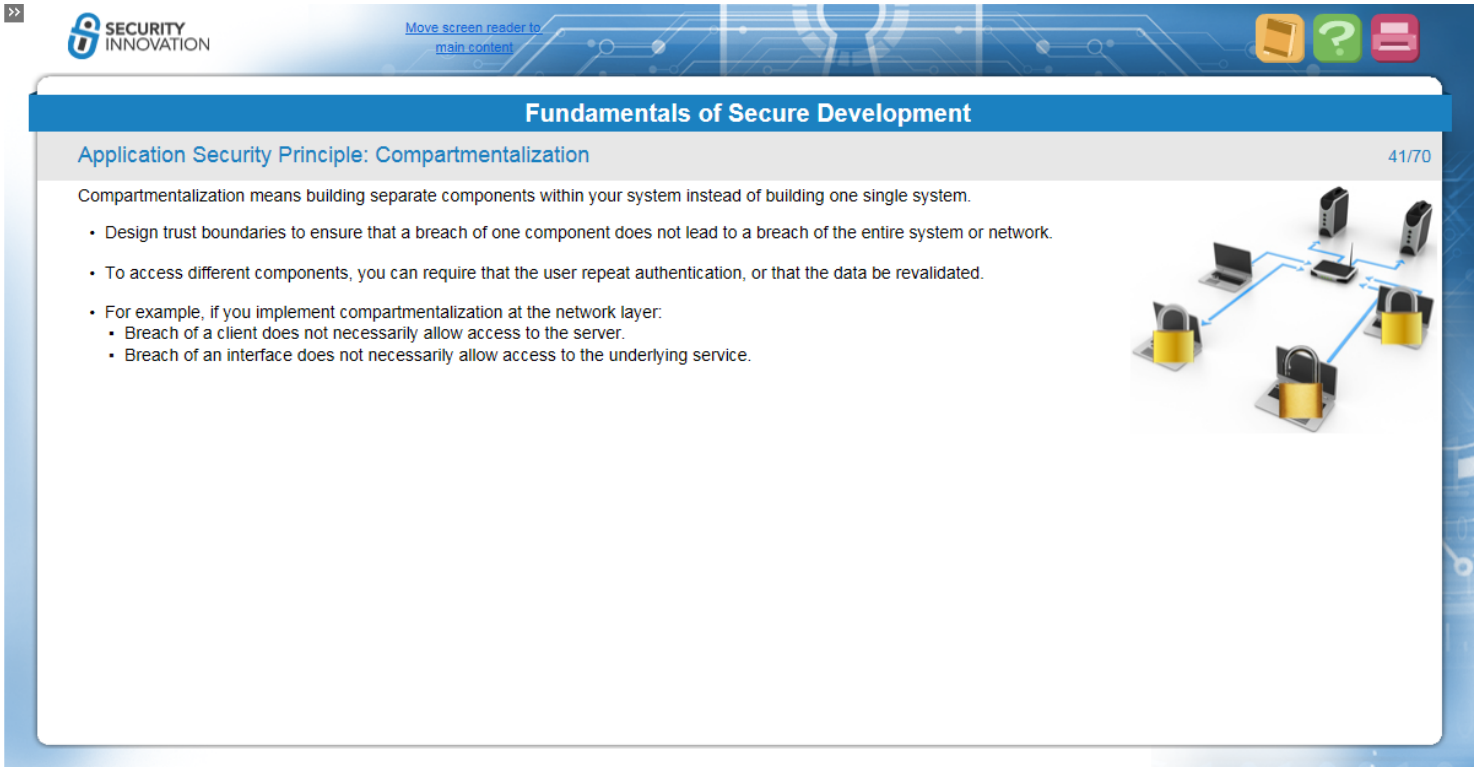
Application Security Principle: Defense in Depth

In the defense in depth strategy, layers of defenses together form a more comprehensive defense posture. If one layer fails, the other layers provide protection for databases and other services.

This strategy significantly reduces the likelihood of a successful attack against your application.

The probability of a malicious user compromising two or more layers of defense is much lower than compromising only one defense measure.

Application Security Principle: Compartmentalization



Narrations

Compartmentalization is similar to defense in depth and least privilege.

Instead of building one system, you build separate components in your application.

This way, you can design trust boundaries to isolate internal components from one another.

To access different components, you can require re-authentication, or that data be re-validated.

Compartmentalization helps ensure that a breach of one component does not lead to a breach of the entire system or network.

For example, if you implement compartmentalization at the network layer, an attacker who succeeds in breaching a client will not necessarily be able to gain access to the server.

Similarly, an attacker who breaches the interface might not necessarily gain access to the underlying service.

On Screen Text

Application Security Principle: Compartmentalization

Compartmentalization means building separate components within your system instead of building one single system.

- Design trust boundaries to ensure that a breach of one component does not lead to a breach of the entire system or network.

- To access different components, you can require that the user repeat authentication, or that the data be revalidated.
- For example, if you implement compartmentalization at the network layer:
    - Breach of a client does not necessarily allow access to the server.
    - Breach of an interface does not necessarily allow access to the underlying service.

Application Security Principle: Policy Compliance

Narrations

Implementing policy compliance means designing your application in a way that enables your customers to comply with legislated security requirements.

As we learned earlier, US federal and state governments have legislated security requirements with

regulations such as Sarbanes Oxley, Gramm-Leach Bliley, HIPAA, HITECH, PCI-DSS, PA-DSS, California

Senate Bill 1386, and the Massachusetts Data Protection Law.

On Screen Text

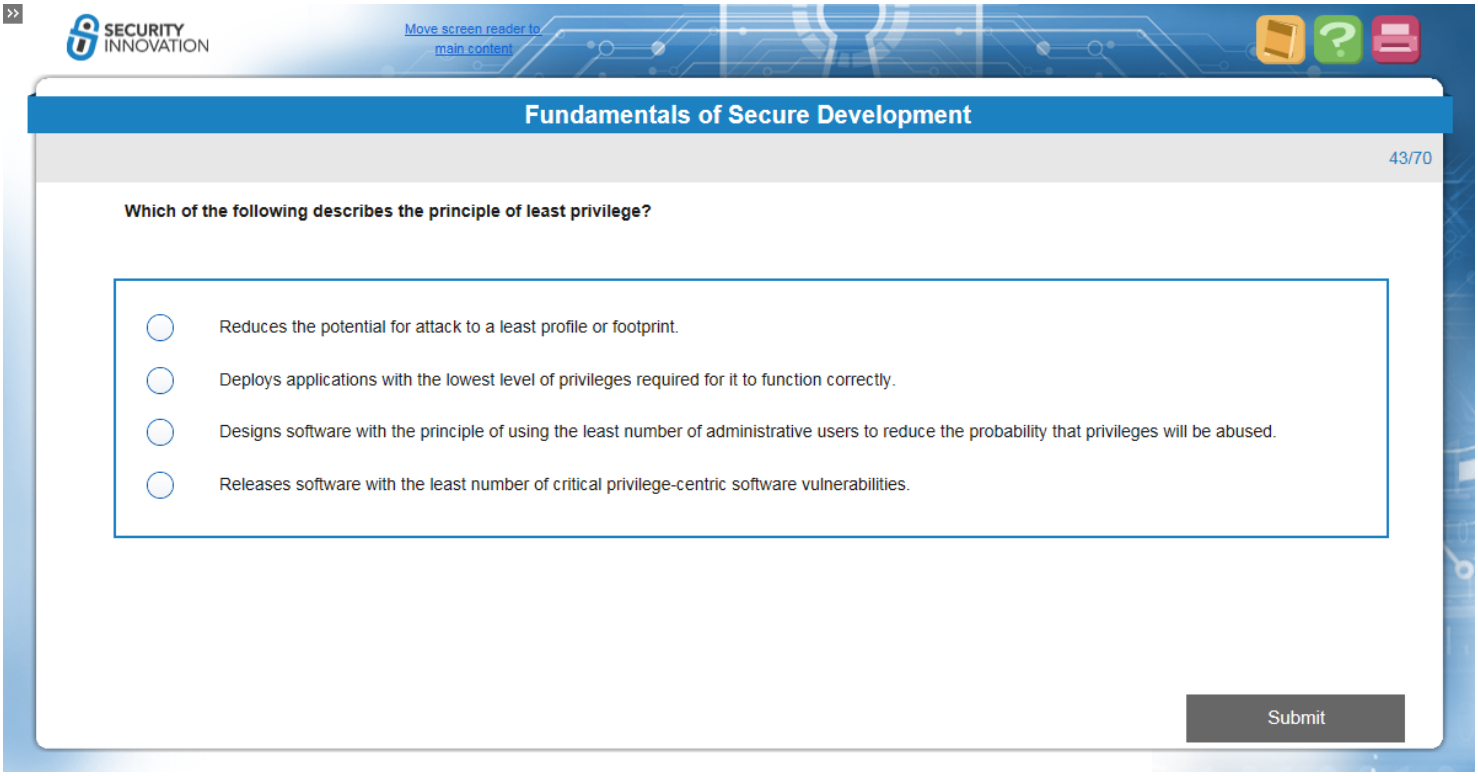
Application Security Principle: Policy Compliance

Implementing policy compliance means designing your application in a way that enables your customers to comply with legislated security requirements, such as:

- Sarbanes Oxley (SOX)
- California Senate Bill 1386 (SB 1386)
- Gramm-Leach Bliley Act (GLBA)
- Health Insurance Portability and Accountability Act (HIPAA)
- Health Information Technology for Economic and Clinical Health Act (HITECH)
- Payment Card Industry Data Security Standard (PCI-DSS)
- Payment Application Data Security Standard (PA-DSS)

- Massachusetts Data Protection Law

Knowledge Check



On Screen Text

Which of the following describes the principle of least privilege?

- Reduces the potential for attack to a least profile or footprint.
- Deploys applications with the lowest level of privileges required for it to function correctly.
- Releases software with the least number of critical privilege-centric software vulnerabilities.
- Designs software with the principle of using the least number of administrative users to reduce the probability that privileges will be abused.

Security in the Implementation and Testing Phases

Narrations

We've discussed security activities for the Planning and Design phases.

Now, let's look at the Implementation and Testing phases.
During the Implementation and Testing phases, developers implement features and testers verify the accuracy of the code. To help ensure the security of your application, apply secure coding principles when implementing code, perform security code analysis and security code review to identify vulnerabilities in source code, and use fault injection, vulnerability scanning, and penetration testing to identify runtime vulnerabilities.

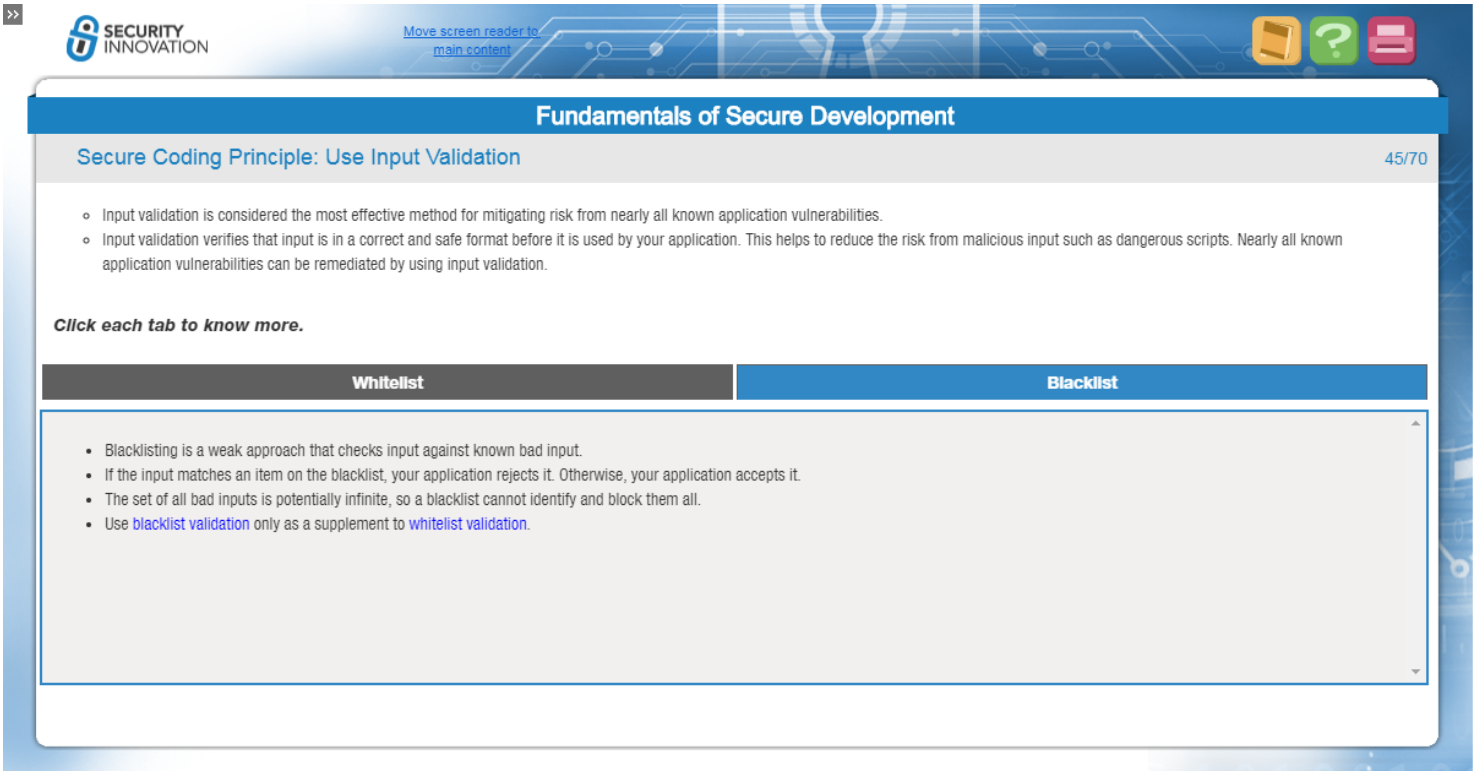
Let's take a look at these activities to learn about their uses and benefits.

On Screen Text

Security in the Implementation and Testing Phases

- Apply secure coding principles:
  - Use input validation.
  - Use security libraries.
  - Leverage language, compiler, and platform protection.
- Perform security code analysis and review:
  - Static code analysis
  - Binary analysis
  - Manual security code review
- Use fault injection.
- Use vulnerability scanning.
- Use penetration testing.

Secure Coding Principle: Use Input Validation



Narrations

Input validation is a critical component of any application security strategy.

When properly implemented, it is considered the most effective method for mitigating risk from nearly all known application vulnerabilities.

Input validation verifies that input is in a correct and safe format before it

is used by your application. This helps to reduce the risk from malicious

input, such as scripts, that could be used to gain access to data, to gain

control of some or all of your application, or to otherwise interfere with

or crash your system.

As a general rule, validate input from all untrusted sources. The two main approaches to input validation are whitelist and blacklist validation.

“Whitelisting” validates all input against a list of acceptable input formats, types, ranges, and lengths.

If the input matches one of the acceptable formats on the whitelist, your application can accept it.

Otherwise, your application rejects it. This is a very robust form of input validation—always use whitelisting.

“Blacklisting,” on the other hand, validates all input against a blacklist of known bad input.

If the input matches an item on the blacklist, your application rejects it.

Otherwise, your application accepts it. (A blacklist is analogous to a no-fly list at the airport.

If a passenger's name is on the no-fly list, he or she is not allowed on the plane.

If the passenger does not appear on the list, then he or she is allowed to fly.)

The inherent weakness of blacklisting is obvious.

The set of all bad inputs is potentially infinite, so a blacklist cannot identify and block them all–some could get through.

Use blacklist validation only as a supplement to whitelist validation.

On Screen Text

Secure Coding Principle: Use Input Validation

- Input validation is considered the most effective method for mitigating risk from nearly all known application vulnerabilities.
- Input validation verifies that input is in a correct and safe format before it is used by your application. This helps to reduce the risk from malicious input such as dangerous scripts. Nearly all known application vulnerabilities can be remediated by using input validation.

Click each tab to know more.

Whitelist

- Whitelisting is a robust approach that checks inputs against known good formats, type, ranges, and lengths.
- If the input matches one of the acceptable formats on the whitelist, your application accepts it. Otherwise, your application rejects it.

Blacklist

- Blacklisting is a weak approach that checks input against known bad input.
- If the input matches an item on the blacklist, your application rejects it. Otherwise, your application accepts it. The set of all bad inputs is potentially infinite, so a blacklist cannot identify and block them all.
- Use blacklist validation only as a supplement to whitelist validation.

Input Validation Tips

Narrations

Be sure to use input validation libraries that are well tailored to your application's needs.

Input validation is complex because the structure of input, such as names, dates, and addresses, can vary widely.

It takes skill and experience to develop and test input validation routines that will suit an application.

Always perform validation on the server side for client-server applications.

The client side is too vulnerable to be relied on for security; it can be bypassed or manipulated by an attacker.

You can still validate on the client for performance, and as a supplementary security measure.

If input passes both client and server validation checks, it is likely safe.

If it passes the client check but fails on the server side, an attack could be occurring.

Finally, use regular expressions to implement a whitelist input validation strategy in your application.

Regular expressions can be used to describe valid input patterns and to verify that untrusted inputs match those patterns.

On Screen Text

Input Validation Tips

Following are some input validation tips:

- Use input validation routines and libraries that are well tailored to meet the specific needs of your application.
- Perform validation on the server side for client-server applications.

Use regular expressions to implement whitelist input validation in your application.

Secure Coding Principle: Use Standard Security Libraries

Narrations

Use standard security libraries when available. This helps ensure correctness of security mitigations and controls.

It also reduces your overall workload because the work has already been done by security experts.

For encryption, always use a standard cryptographic library. Nearly every programming language has built-in cryptographic implementations.

Never attempt to develop your own cryptography, which could expose your application to undue risk.

Cryptographic algorithms are difficult to implement correctly and require many rounds of review and scrutiny by experts before being certified as reasonably correct.

For defenses against attacks such as cross site scripting, many standard libraries already exist, such as the standard encoding library that Microsoft has built into the .NET Framework.

Microsoft has also released a stronger version called the Web Protection Library.

For database applications, you can use the Anti-SQL Injection library developed by the security experts at Security Innovation and IronBox to simplify database protection strategies.

On Screen Text

Secure Coding Principle: Use Standard Security Libraries

Use standard security libraries whenever they are available.

- For encryption, always use a standard cryptographic library. Never attempt to develop your own, which could expose your application to undue risk.
- For defenses against attacks such as cross site scripting, use Microsoft's standard encoding library built into the .NET Framework.
- For database applications, you can use the Anti-SQL Injection library developed by Security Innovation and IronBox to simplify database protection strategies.

Secure Coding Principle: Use Language Compiler and Platform Protection

Narrations

Programming languages and technology platforms often have built-in security features that can reduce your secure application development workload.

For example, Microsoft ASP.NET has a built-in mechanism called Request Validation that detects certain cross-site scripting attacks.

And, when developing in C or C++ for Windows, you can use the compiler's /GS option to add limited run-time protection against stack-based buffer overflows, a very serious vulnerability that affects applications written in native or unmanaged languages.

Be sure you understand the full security capabilities of the language and platform on which you are developing your secure application, and leverage those capabilities whenever possible.

On Screen Text

Secure Coding Principle: Use Language, Compiler and Platform Protection

Most programming languages and platforms have built-in security features to defend against common application security attacks.

For example:

- Microsoft ASP.NET has built-in features to defend against cross-site scripting attacks.
- The Windows C/C++ compiler has built-in protection against stack-based buffer overflows and common elevation of privilege attacks.

Be sure you understand the security capabilities of the development language and platform you are using, and leverage those capabilities as much as possible.

Code Analysis and Code Review

Narrations

Let's now discuss code analysis and code review.

Code analysis is performed with software tools that inspect compiled and non-compiled source code for known defects.

For example, code analysis tools can be used to detect instances of known SQL injection patterns in your C and C++ code.

Code analysis can be categorized into static analysis and binary analysis.

Code review is the manual inspection of application source code by a security analyst.

Both code analysis and code review are important components of any secure

application development strategy. They allow you to detect and correct

vulnerabilities in legacy applications, and to find vulnerabilities in current

applications early in the development lifecycle, when the cost of fixing them is

low.

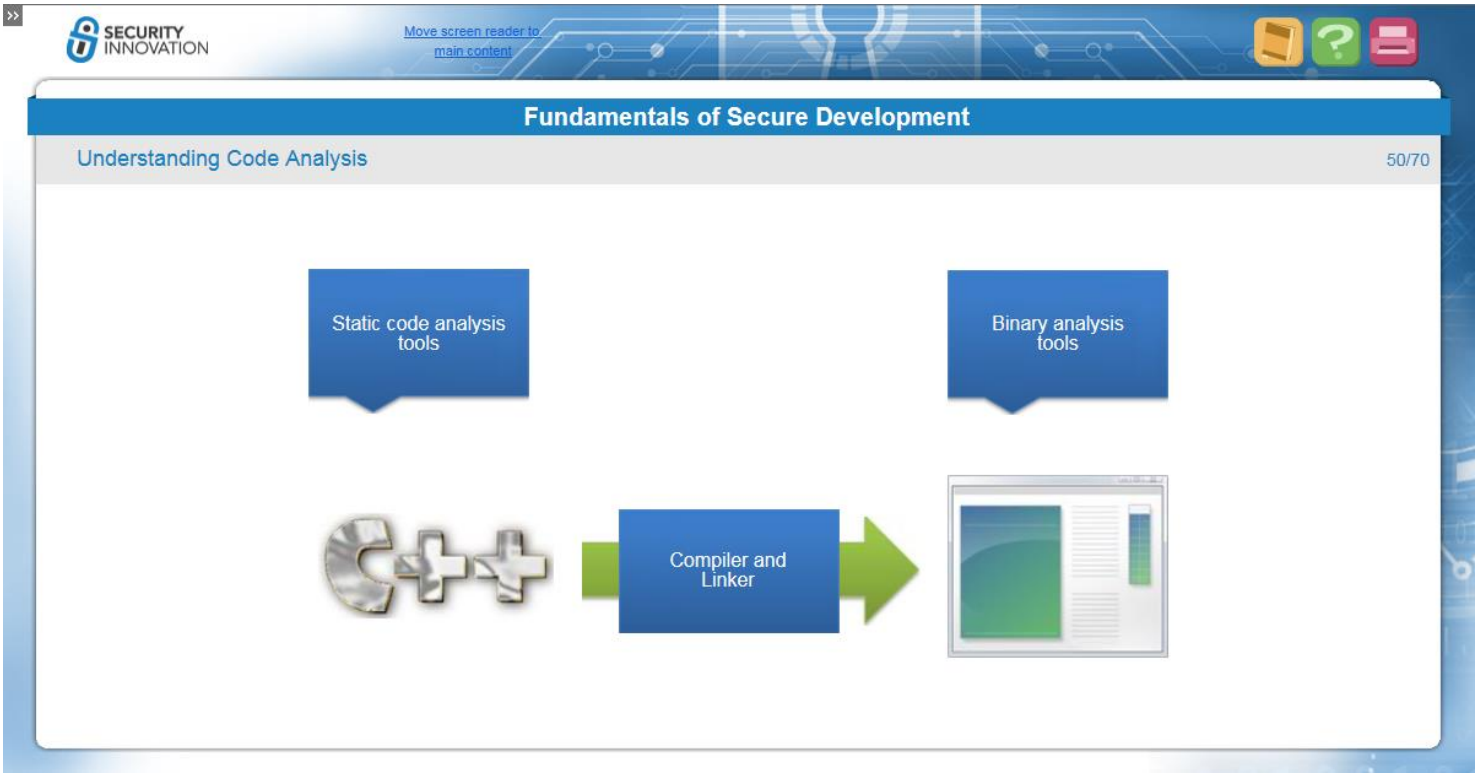
On Screen Text

Code Analysis and Code Review

- **Code analysis** is performed with software tools that inspect compiled and non-compiled application code for known defects. There are two categories of code analysis:
  - Static analysis
- Binary analysis
- **Code review** is the manual inspection of application source code for known defects.
- Code analysis and code review allow you to:
  - Detect and correct vulnerabilities in legacy applications.

Detect vulnerabilities in current applications early in the development lifecycle when the cost of fixing them is low.

Understanding Code Analysis



Narrations

Let’s focus on static and binary code analysis. To understand the difference between the two, consider the application source code lifecycle.

All applications are expressed as source code. These are the human readable instructions, such as Java code, that implement software.

A compiler and linker convert the source code into machine code, often called an application binary.

Static code analysis tools analyze source code, while binary analysis tools analyze application binaries.

Each has specific benefits and limitations.

On Screen Text

Understanding Code Analysis

Static Analysis: Pros and Cons

Narrations

Let's take a look at some of the benefits and limitations of static analysis tools.

Compared to manual code reviewers, static analysis tools can scale easily to analyze large amounts of code and are objective and unbiased.

Compared to binary analysis tools, static analysis tools are generally more mature and tend to find more vulnerabilities.

In addition, issues identified by static analysis tools are easier to debug because they work with human-readable source code instead of compiled code.

Static analysis tools do have limitations, though.

They are language specific, so organizations that use many programming languages

might require many static analysis tools. In addition, they can produce false positives

by flagging items that are not vulnerable, and produce false negatives by missing

actual vulnerabilities, which can degrade confidence in these tools.

Finally, static analysis tools can only find implementation flaws, revealing specific coding patterns that lead to vulnerabilities.

They cannot necessarily detect flaws related to the business logic of the application.

For example, a static analysis tool might not detect if an application incorrectly assigns excessive rights to a user, or if it contains a bug in which certain operations are performed contrary to the intended design.

On Screen Text

Static Analysis: Pros and Cons

The pros and cons of static analysis tools are as follows:

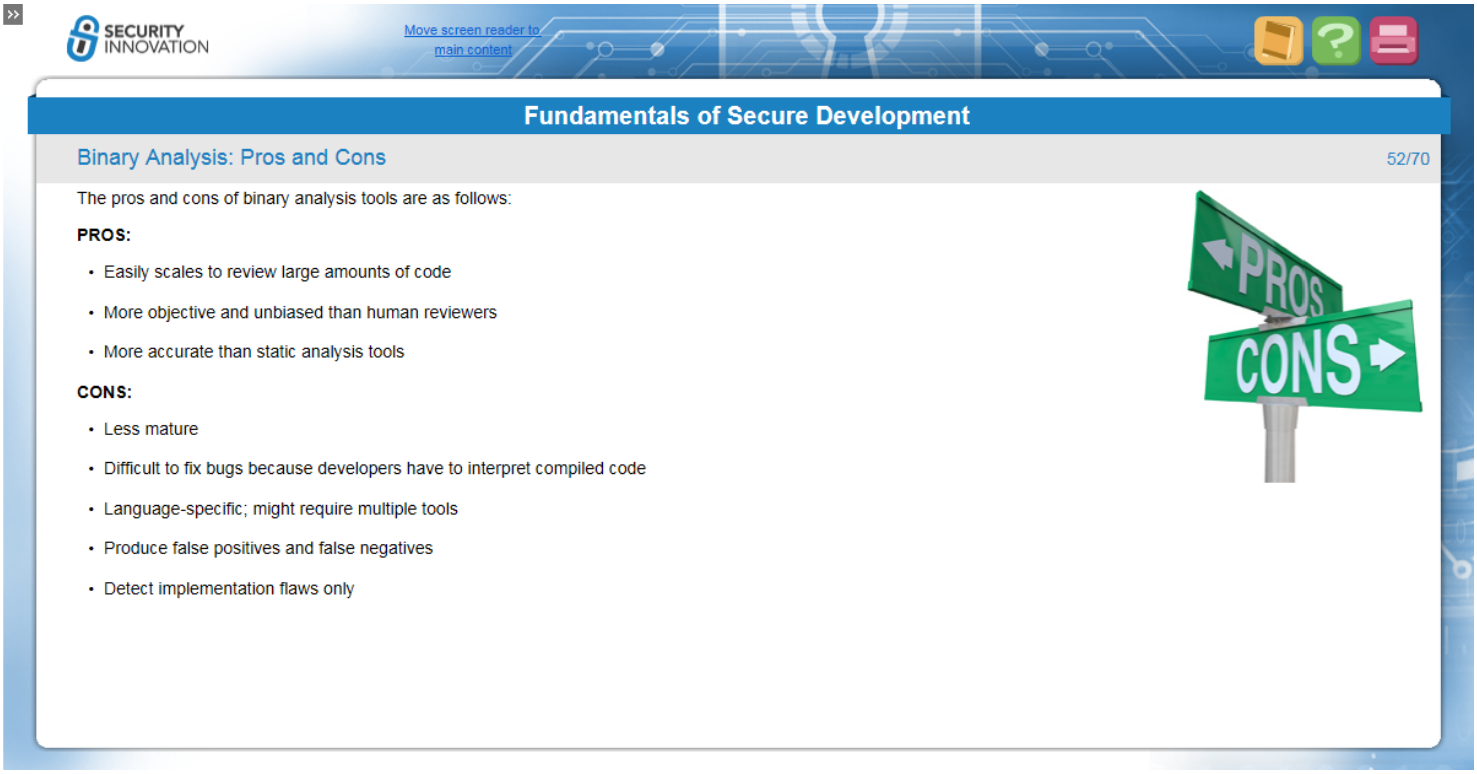
PROS:

- Easily scales to review large amounts of code
- Objective and unbiased
- Mature technology
- Easier to debug identified issues

CONS:

- Language specific; might require multiple tools
- Produces false positives and false negatives
- Detects implementation flaws only
- Cannot detect business logic flaws

Binary Analysis: Pros and Cons



Narrations

Now let's take a look at some pros and cons of binary analysis tools.

Like static analysis tools, binary analysis tools can scale to

review large amounts of code, and are more objective and

unbiased than human reviewers.

Binary analysis tools are more accurate than static analysis tools because they

operate on the actual code that is executed. (Static analysis tools operate on source

code, which might be optimized and rearranged by compilers.) However, binary

analysis tools do have limitations.

They are far less mature compared to static analysis tools, so the number of vulnerabilities that they can detect is limited.

In addition, the bugs identified by binary analysis tools are difficult to fix because developers have to interpret compiled code.

Finally, binary analysis tools suffer from some of the same limitations as static analysis tools.

They are language-specific, they produce false positives and false negatives, and they can only detect implementation flaws.

On Screen Text

Binary Analysis: Pros and Cons

The pros and cons of binary analysis tools are as follows:

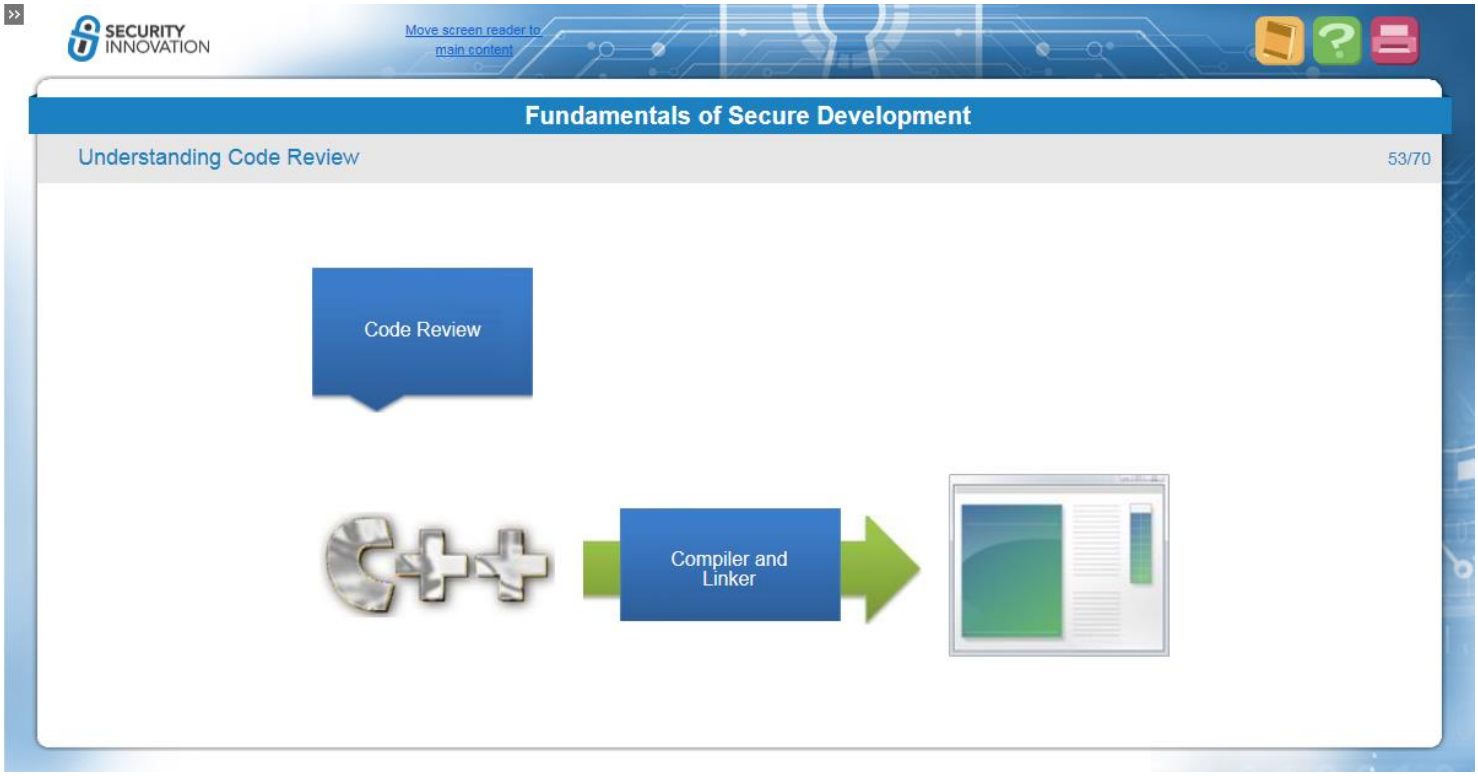
PROS:

- Easily scales to review large amounts of code
- More objective and unbiased than human reviewers
- More accurate than static analysis tools

CONS:

- Less mature
- Difficult to fix bugs because developers have to interpret compiled code
- Language-specific; might require multiple tools
- Produce false positives and false negatives
- Detect implementation flaws only

Understanding Code Review



Narrations

Let’s review the application compilation lifecycle to understand code review.

Recall that applications are first expressed as source code. Then, the source code is processed by a compiler and linker to produce binary code.

Code review, like static code analysis tools, utilizes human-readable source code to detect implementation flaws.

However, the analysis is performed by a human and not a software tool.

On Screen Text

Understanding Code Review

Code Review: Pros and Cons

Narrations

Code review has both benefits and limitations.

Compared to code analysis, manual code reviews generally produce fewer false negatives and false positives.

And human code reviewers can find not only implementation flaws, but also flaws in design and in business logic.

Although manual code review provides some benefits, it does have certain limitations.

Manual code review is very time consuming and does not scale well.

Reviewers can become fatigued from examining large amounts of code, which diminishes their ability to accurately identify vulnerabilities.

In addition, different reviewers have different areas of expertise, so vulnerabilities might be overlooked depending on who reviews the code.

On Screen Text

Code Review: Pros and Cons

Code review has the following benefits and limitations:

PROS:

- Has fewer false positives and false negatives
- Can identify implementation, design, and business logic flaws

CONS:

- Time consuming and does not scale well
- Affected by fatigue and human error
- Biased and not objective

Security Code Review Resources

Narrations

You need not be a security expert to perform a security code review. Although security expertise certainly helps, several resources can help you perform your own code reviews. For example, the OWASP Code Review Project has produced the OWASP Code Review Guide, which explains how to review code for a large number of vulnerability types.

On Screen Text

Security Code Review Resources

Several resources are available to help you perform a security code review. For example, the OWASP Code Review Project has produced the OWASP Code Review Guide, which explains how to review code for a large number of vulnerability types.

For more information, visit the Team Mentor article How to Perform a Security Code Review, and take the Security Innovation course “How to Perform a Security Code Review” (ENG 312).

Fault Injection

Narrations

You can augment your security testing with a technique called fault

injection, whereby faults in the code paths are introduced into an

application to see how the application responds.

Fault injection is commonly used to identify vulnerabilities that may have been missed by other secure testing techniques.

Two fault injection techniques are commonly used.

In compile-time injection, faults are injected directly into the source code to simulate error conditions and to see how the application responds.

In run-time injection, faults are injected into the application while it is running to see how the application responds.

Faults can be injected in a variety of ways, such as by corrupting physical memory, modifying network packets, or through common inputs like user interfaces and API calls.

Fuzz testing or fuzzing is a common fault injection technique used to discover vulnerabilities in applications.

Fuzzing typically uses an automated tool that provides invalid or random input to an application and monitors how the application behaves.

A crash or an unhandled exception can often provide clues about more serious vulnerabilities in code, such as buffer overflows.

On Screen Text

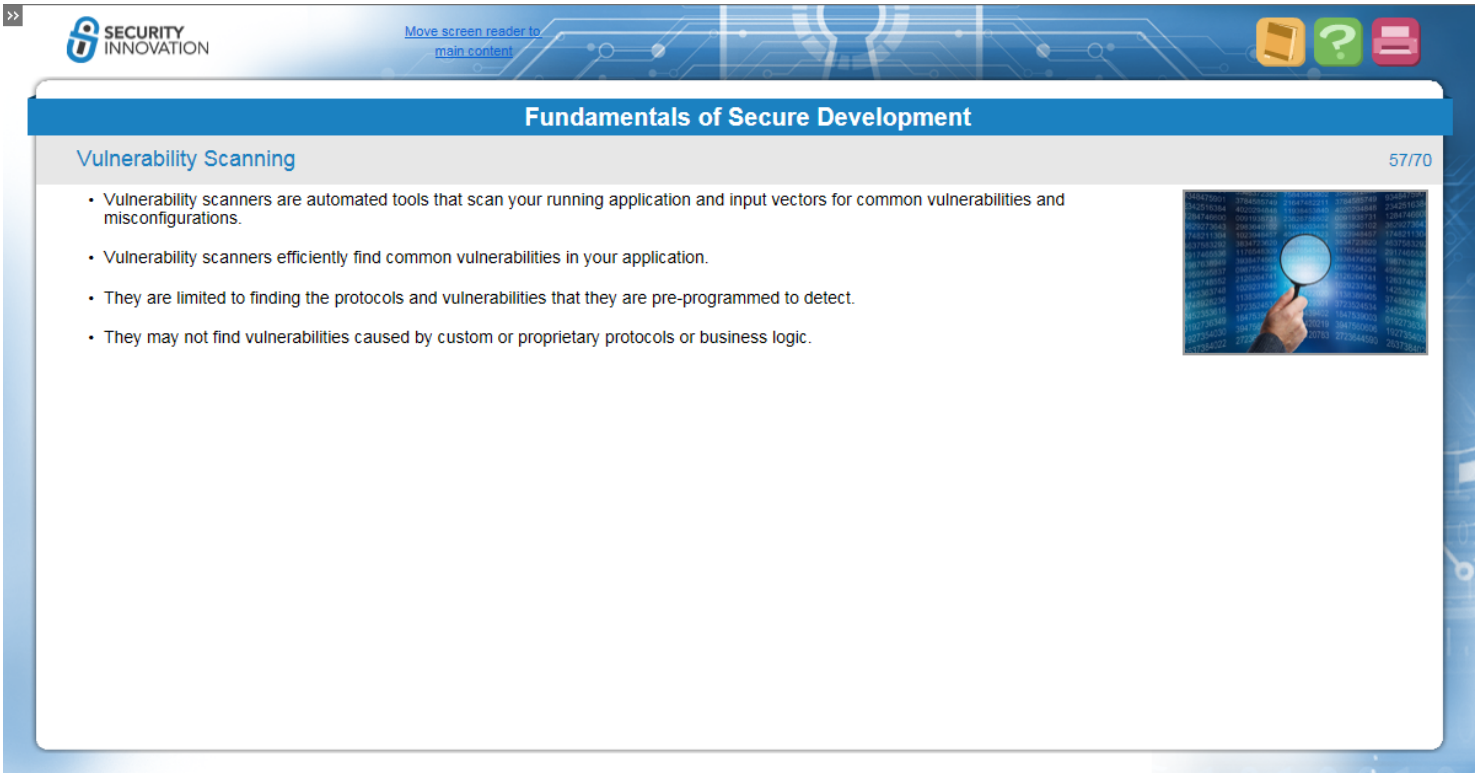
Fault Injection

- In fault injection, faults in the code paths are introduced into an application to see how it responds.
- The two common fault injection techniques are:

- Compile-time: Faults are injected directly into source code.
- Run-time: Faults are injected while the application is running.

- Fuzzing: A common fault injection technique called fuzzing is used to discover vulnerabilities in applications. Fuzzing uses an automated tool to send a series of invalid or random inputs to an application and monitors how the application responds to those inputs.

Vulnerability Scanning



Narrations

Vulnerability scanners are automated tools that can help you check your application for common application vulnerabilities and misconfigurations. Vulnerability scanning differs from code analysis in that the scanning tool operates on running applications and input vectors, such as listening ports and APIs, instead of source code or compiled binaries.

Several regulations and standards require applications to undergo regular vulnerability scanning to maintain compliance. Although vulnerability scanners can efficiently find common application vulnerabilities and misconfigurations, they are limited to the protocols and vulnerabilities that they are pre-programmed to detect.

They might not find vulnerabilities caused by custom or proprietary protocols or business logic. As a result, the vulnerability scanner might report that your application is free from vulnerability, when in fact it is not, creating a false sense of security.

Still, vulnerability scanning is an effective means of identifying common unmitigated risks in your applications.

On Screen Text

Vulnerability Scanning

- Vulnerability scanners are automated tools that scan your running application and input vectors for common vulnerabilities and misconfigurations.
- Vulnerability scanners efficiently find common vulnerabilities in your application.
- They are limited to finding the protocols and vulnerabilities that they are pre-programmed to detect.
- They may not find vulnerabilities caused by custom or proprietary protocols or business logic.

Penetration Testing

Narrations

In penetration testing, you attack an application to identify and exploit its vulnerabilities.

Regular penetration tests are often legally required to maintain regulatory compliance.

Penetration tests differ from vulnerability scans.

Penetration tests are performed by actual security experts who use both custom and off-the-shelf tools and techniques.

Unlike vulnerability scanners, penetration testers can adapt to

custom protocols and business logic. However, because

penetration testers are human, they cannot scale to the same

degree as automated vulnerability scanners, and their accuracy

can be affected by fatigue, limitations in expertise, or lack of

esources.

Still, penetration testing is an effective way to identify vulnerabilities in your applications.

On Screen Text

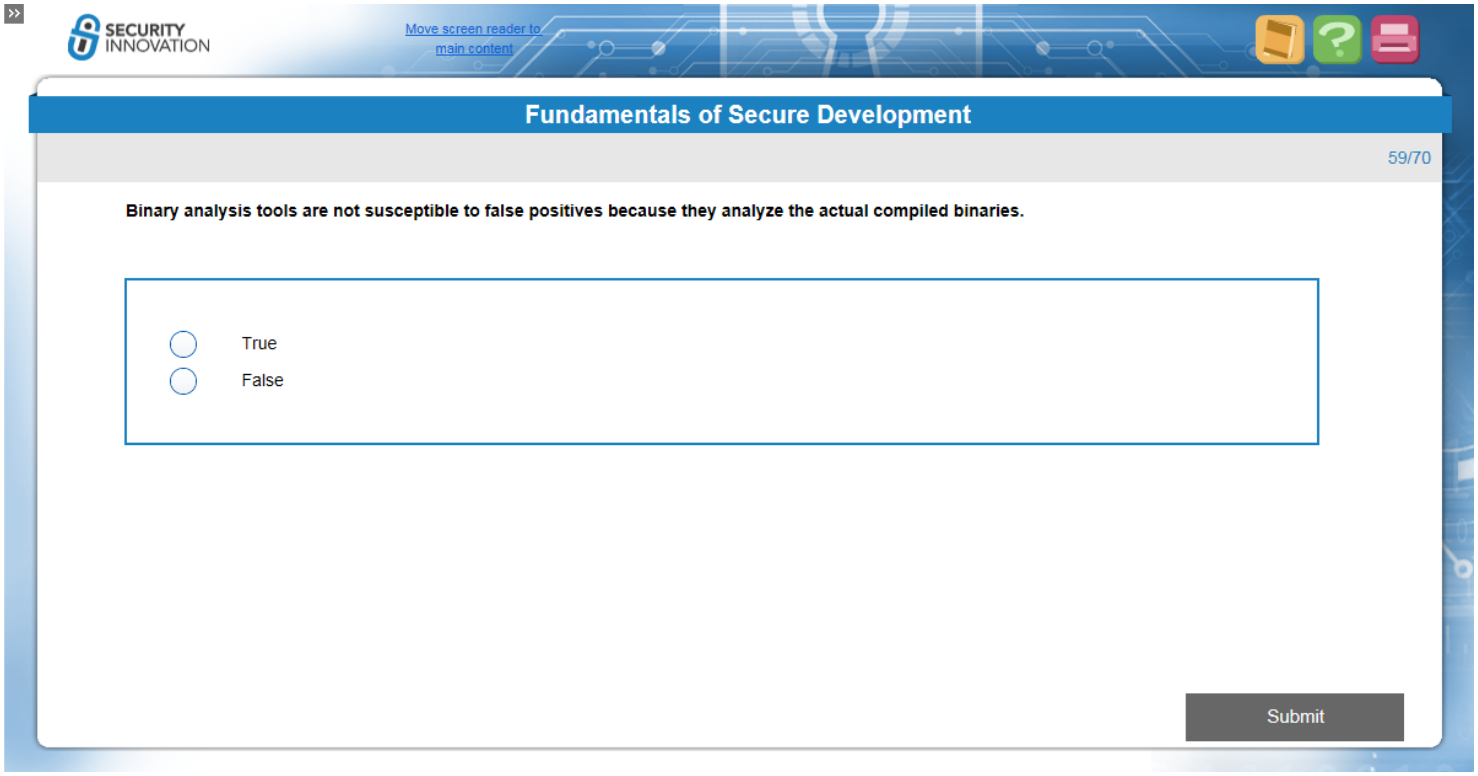
Penetration Testing

In penetration testing, you attack an application to identify and exploit its vulnerabilities.

- Penetration testers can adapt to custom protocols and business logic.
- They cannot scale to the same degree as automated vulnerability scanners.

Accuracy can be affected by fatigue, limitations in expertise, or lack of resources.

Knowledge Check



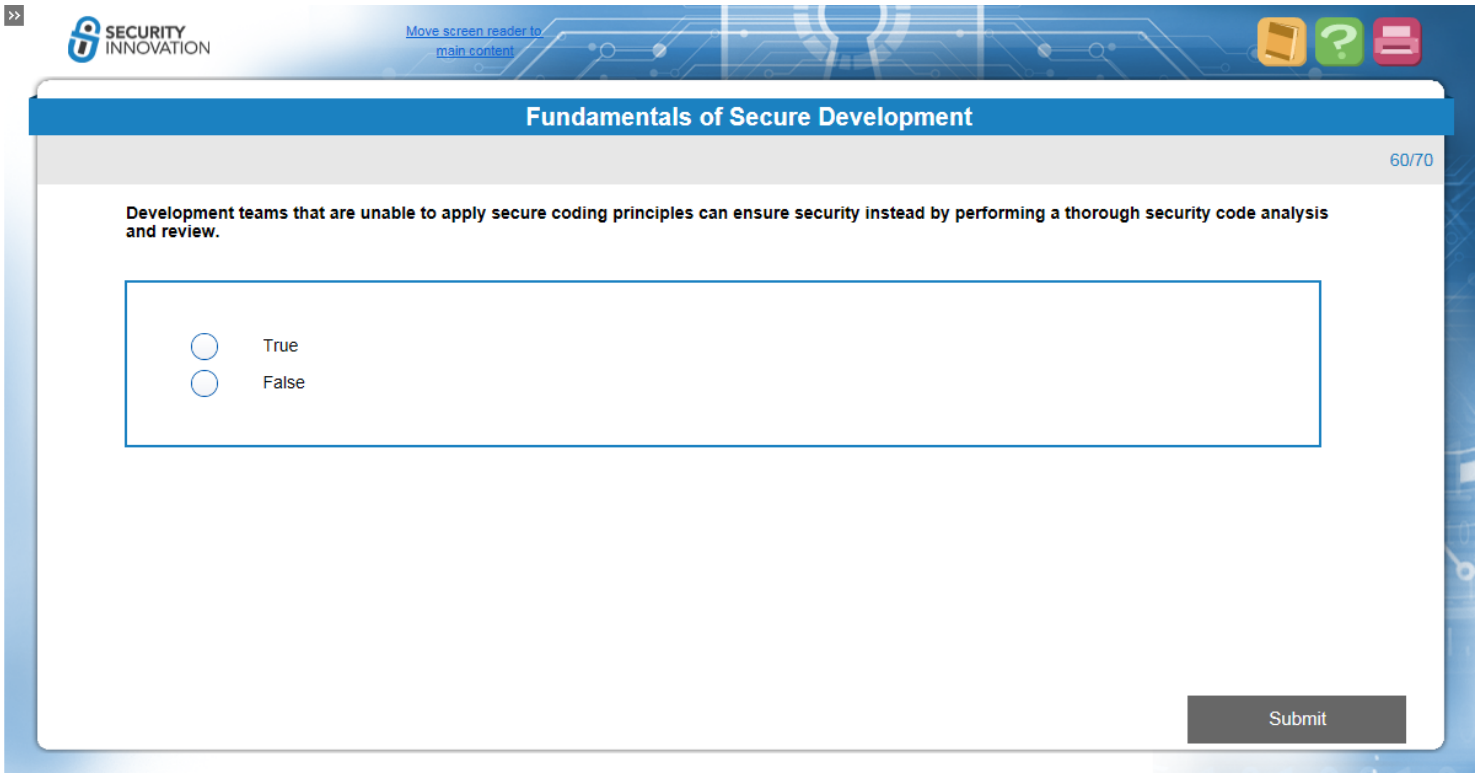
On Screen Text

Knowledge Check

Binary analysis tools are not susceptible to false positives because they analyze the actual compiled binaries.

- True
- False

Knowledge Check



On Screen Text

Knowledge Check

Development teams that are unable to apply secure coding principles can ensure security instead by performing a thorough security code analysis and review.

- True
- False

Security in the Deployment and Maintenance Phase

Narrations

The final phase of the software development process is the Deployment and Maintenance phase.

Secure deployment concepts that apply to this phase include least privilege deployment, incident response plans, and patching plans.

On Screen Text

Security in the Deployment and Maintenance Phase

Key secure deployment concepts for the Deployment and Maintenance phase include the following:

- Least privilege deployment
- Incident response plan
- Patching plan

Least Privilege Deployment

Narrations

Let’s take a look at least privilege deployment.

As you learned in the Design phase, least privilege can be used to design applications so that they use the least amount of privileges needed for the application to function.

You can also leverage the principle of least privilege in the Deployment phase.

Deploy applications using the minimal privileges needed for the application to function.

This way, if an attacker compromises a user account, the attacker’s privileges and ability to inflict damage are limited.

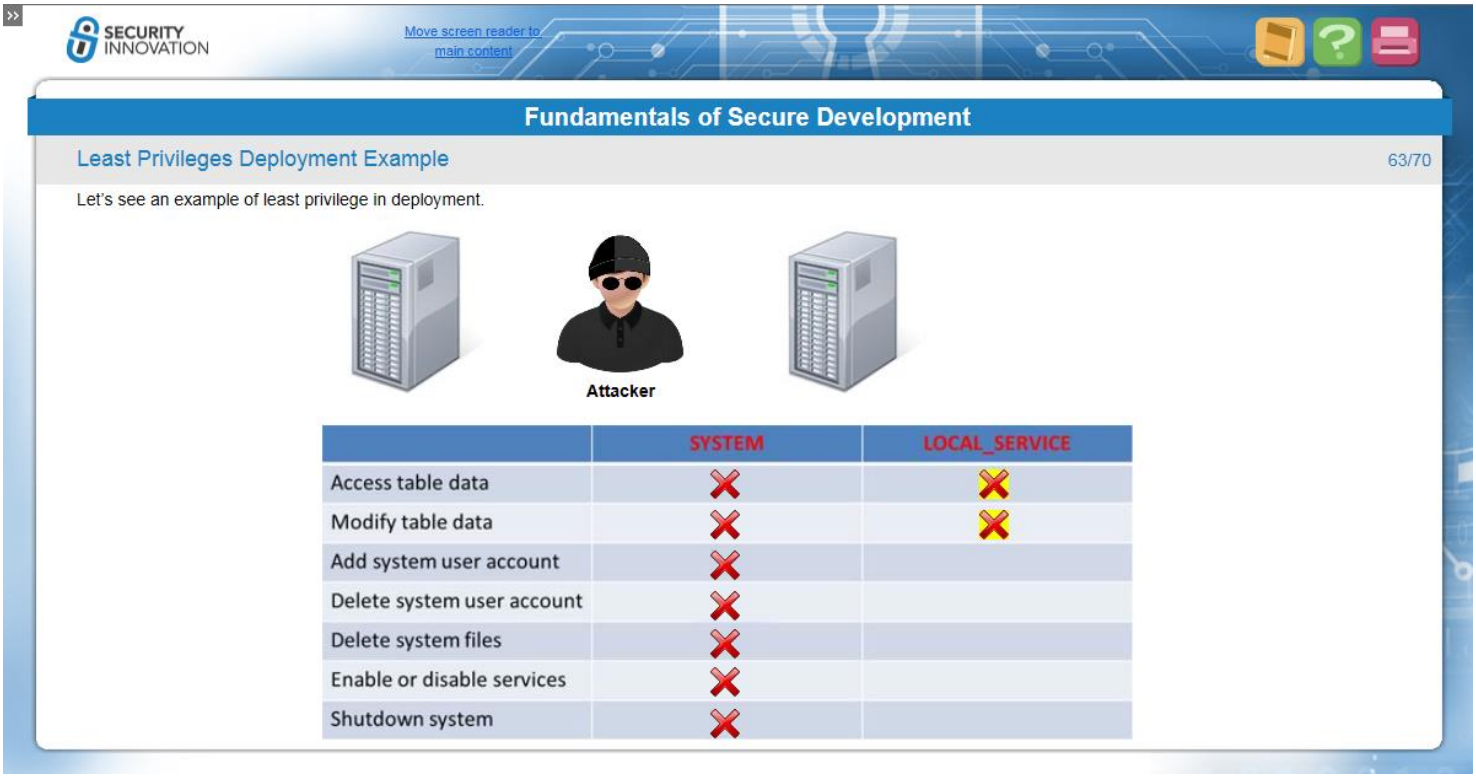
On Screen Text

Least Privilege Deployment

Apply the principle of least privilege in the Deployment phase by deploying your applications with the minimal set of privileges needed for the application to run correctly.

This limits the potential damage if an attacker hacks your application.

Least Privileges Deployment Example



Narrations

Let's see an example of least privilege in deployment.

Using the principle of least privilege, you deployed a Windows database application as a lower privileged service such as LOCAL SERVICE and not SYSTEM.

An attacker then compromised your application.

As this chart illustrates, if the database application was running as the

SYSTEM account, the attacker could potentially do anything they

wanted because the SYSTEM account has no restrictions. However,

because the application was deployed using the lower privileged LOCAL

SERVICE account, the attacker is limited to the privileges of the

application, thereby limiting the potential for further damage.

On Screen Text

Least Privileges Deployment Example

Let’s see an example of least privilege in deployment.

Security Incident Response Plans

Narrations

As part of a sound application security strategy, you must have a security incident response plan in place prior to deployment.

This governs your organization's response to security and privacy incidents so that they are handled as effectively and efficiently as possible.

The plan should include a list of incident response team members, their emergency contact information, and their roles and responsibilities. In addition, it should include a policy for handling internal and external communications about security incidents, an incident validation and patch development process, and provisions to ensure compliance with applicable laws and regulations.

For more guidance on incident response planning for all development languages and platforms, see Microsoft's SDL Privacy Escalation Response Framework.

On Screen Text

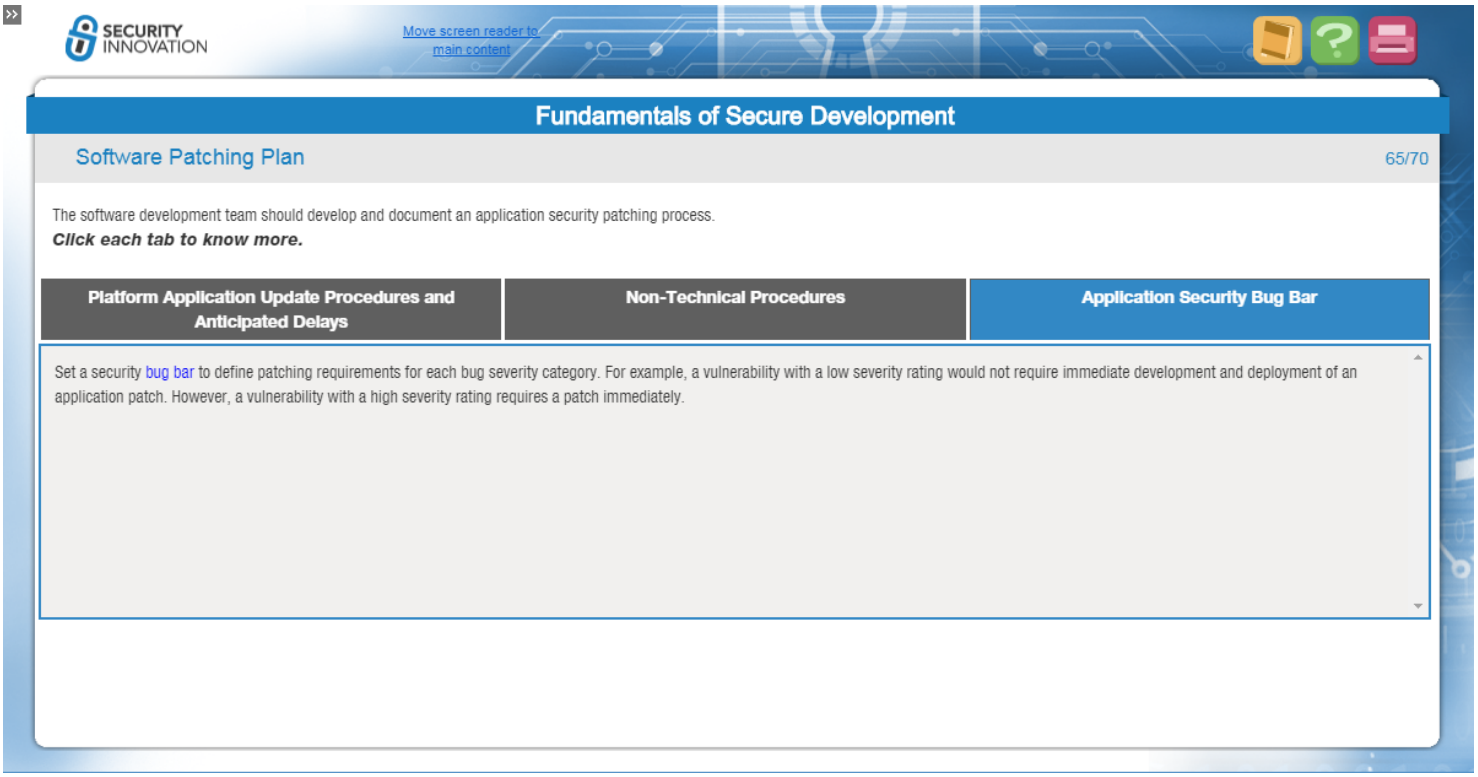
Security Incident Response Plans

To help your organization handle security and privacy incidents, have a security incident response plan in place before deployment. Items to consider in your plan include:

- A list of incident response team members and emergency contact information.
- A description of roles and responsibilities for the incident response team members.
- An internal and external communication policy for security incidents.
- A process for validating incidents and developing patches.

Provisions to ensure compliance with applicable regulations.

Software Patching Plan



Narrations

The software development team should develop and document an application security patching process.

Click each tab to learn about items to include in the security patching process document.

On Screen Text

Software Patching Plan

The software development team should develop and document an application security patching process.

Click each tab to learn about items to include in the security patching process document.

Platform Application Update Procedures and Anticipated Delays:

Document the process to deploy security patches and updates for each supported platform. Include information about anticipated delays.

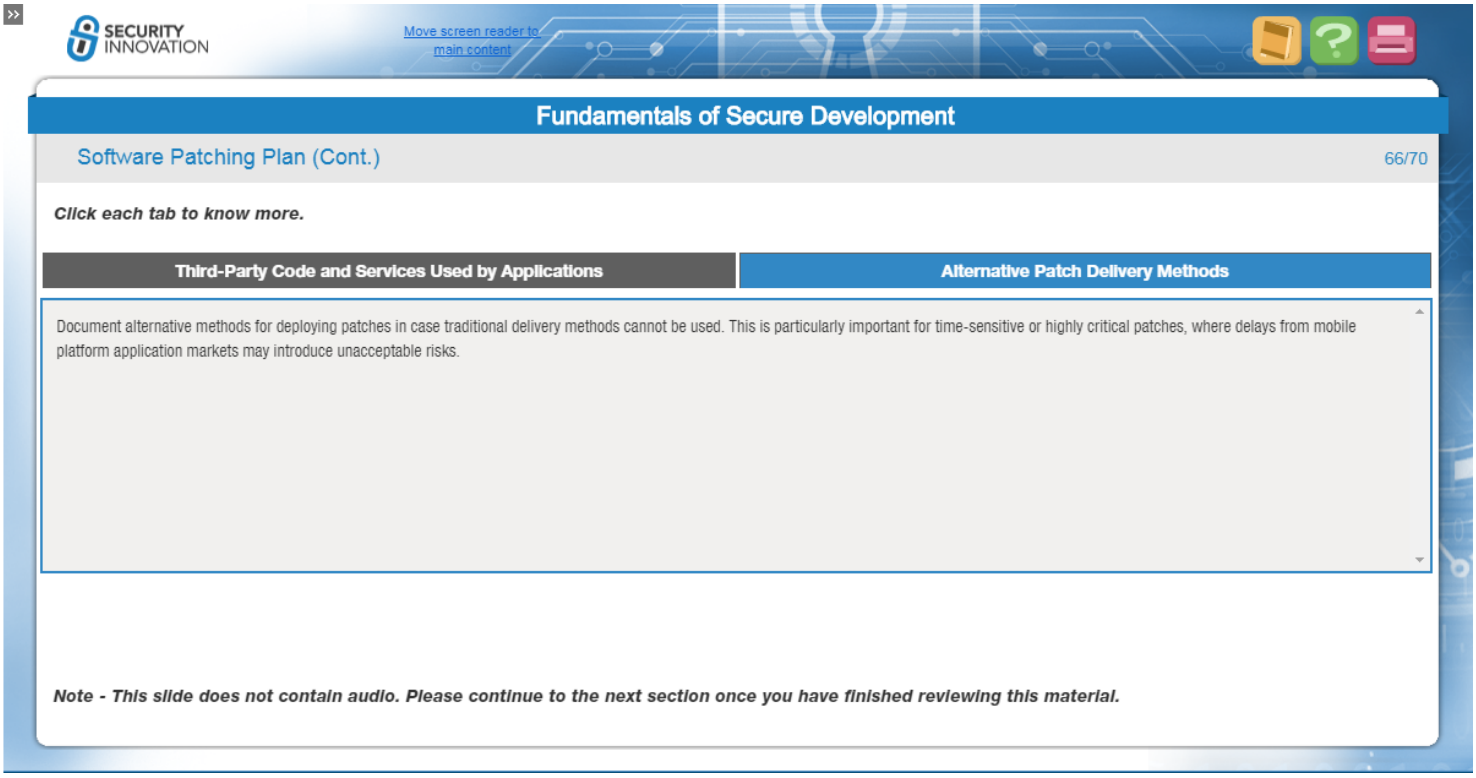
Non-Technical Procedures:

Document non-technical procedures related to security patching, such as how to notify customers, and how to handle an extended delay during the patch approval process.

Application Security Bug Bar:

Set a security bug bar to define patching requirements for each bug severity category. For example, a vulnerability with a low severity rating would not require immediate development and deployment of an application patch. However, a vulnerability with a high severity rating requires a patch immediately.

Software Patching Plan (Cont.)



On Screen Text

Software Patching Plan

Third-Party Code and Services Used by Applications:

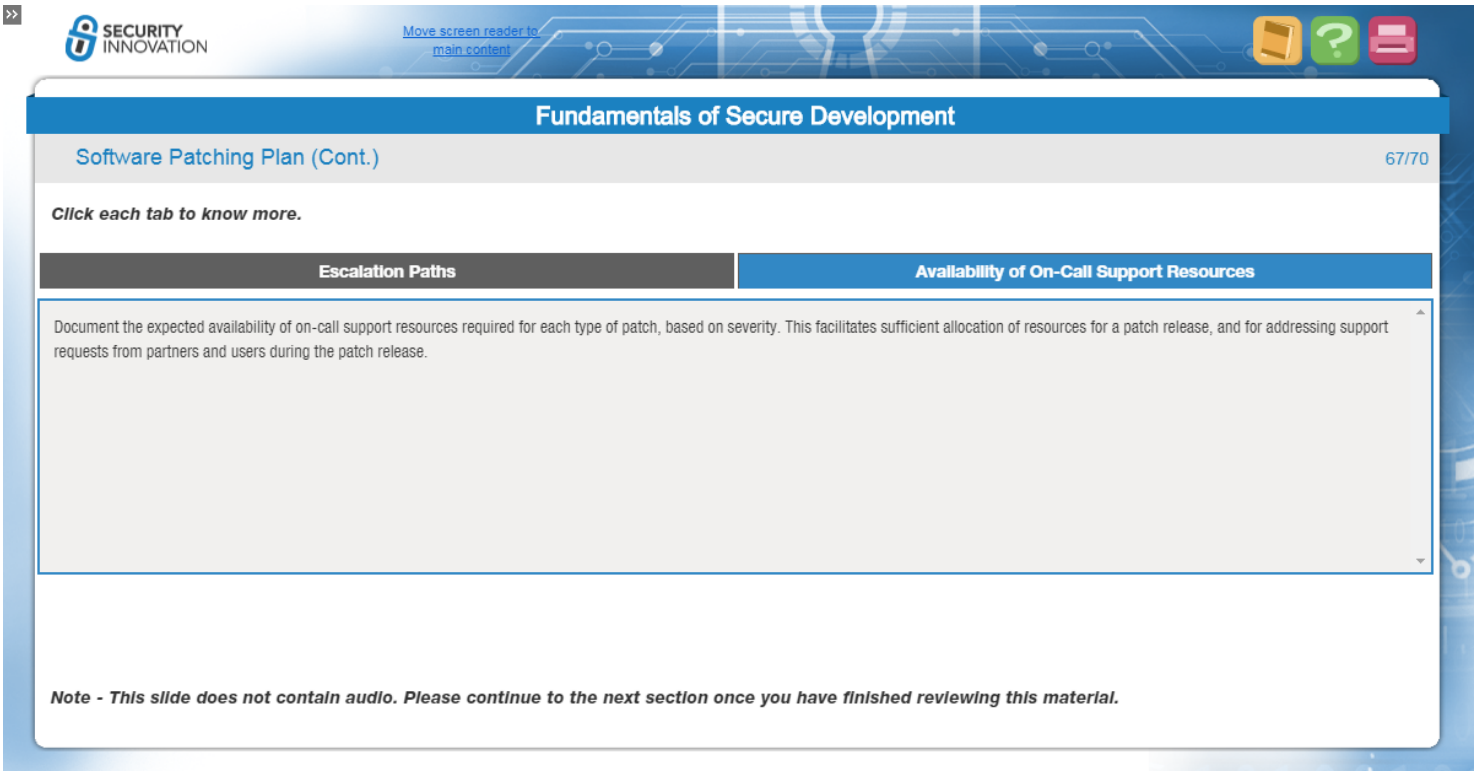
Document all third-party code, libraries, and services used by an application.

Document procedures for deploying patches for these components.

Alternative Patch Delivery Methods:

Document alternative methods for deploying patches in case traditional delivery methods cannot be used. This is particularly important for time-sensitive or highly critical patches, where delays from mobile platform application markets may introduce unacceptable risks.

Software Patching Plan (Cont.)



On Screen Text

Software Patching Plan (Cont.)

Escalation Paths:

Document patch support and escalation paths, including contact information and procedures for escalation.

Availability of On-Call Support Resources:

Document the expected availability of on-call support resources required for each type of patch, based on severity. This facilitates sufficient allocation of resources for a patch release, and for addressing support requests from partners and users during the patch release.

Knowledge Check

On Screen Text

Knowledge Check

A least-privilege software deployment is the process of:

- Deploying software with minimal admin accounts.
- Configuring software so that the least number of features are enabled.
- Configuring software to run as services with minimal security privileges.
- Deploying software so that it depends on the least number of third-party components

Slide 70 - Module Summary

On Screen Text

Module Summary

Security in the Planning Phase

In this topic, you learned about the need to gather legal security requirements and customer security requirements, and how to select the proper security training for your development team.

Security in the Design Phase

In this topic, you learned about threat modeling as well as the application security principles to apply during the Design phase, such as least privilege, attack surface reduction, defense in depth, compartmentalization, and policy compliance.

Security in the Implementation and Testing Phase

In this topic, you learned about the key secure coding principles of using input validation, using well known cryptographic libraries and security libraries, and leveraging built-in language, compiler, and platform protection.

You also learned about code analysis and code review, fault injection, vulnerability scanning, and penetration testing.

Security in the Deployment and Maintenance Phase

In this topic, you learned about least privilege deployment, developing a security incident response plan, and developing a software patching plan.

Thank You

On Screen Text

Thank You

This concludes the **Fundamentals of Secure Development** course. Thank you.

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