

Course Overview and Objectives

The screenshot shows a course interface titled "Fundamentals of Application Security". At the top left is the "SECURITY INNOVATION" logo. A "Move screen reader to main content" link is visible. On the right are icons for a clipboard, a question mark, and a print function. The main content area has a blue header "Course Overview and Objectives" and a sub-header "01/60". The central content box is titled "COURSE OVERVIEW" in large green letters. Below it is a bulleted list of objectives:

- Identify the drivers for application security
- Understand fundamental concepts of application security risk management
- Understand the anatomy of an application attack and identify common forms of attack
- Identify the concepts of input validation as a primary risk mitigation technique
- Identify key security principles and best practices for developing secure applications

Narration

In this course, you will be able to identify the drivers for application security, understand the fundamental concepts of application security risk management, and understand the anatomy of an application attack and identify common forms of attack.

You will be also able to identify the concepts of input validation as a primary risk mitigation technique.

In addition, you will be able to identify the security principles and best practices for developing secure applications.

On Screen Text

Course Overview and Objectives

Module Overview and Objectives

The screenshot shows a web-based learning interface. At the top left is the "SECURITY INNOVATION" logo. To its right is a link "Move screen reader to main content". On the far right are three icons: a yellow square, a green circle with a question mark, and a red square with a minus sign. The title "Fundamentals of Application Security" is centered at the top in a blue header bar. Below it, a sub-header "Module Overview and Objectives" is on the left, and a page number "02/60" is on the right. The main content area has a dark background with the title "MODULE OVERVIEW" in large, bold, green letters. Below it is a bulleted list of objectives:

- Identify key attacker motives
- Important security risk management terms and concepts
- Key approaches for managing application security risk

Narration

This module will teach you application security fundamentals that will form the basic foundation for later modules and courses.

After completing this module you will be able to understand what application security is and understand the technical, business, and regulatory drivers for application security.

You will also be able to identify key attacker motives, important security risk management terms and concepts, and key approaches for managing application security risk.

On Screen Text

Module Overview and Objectives

What is Application Security?

The screenshot shows a presentation slide titled "Fundamentals of Application Security". The main heading on the slide is "WHAT IS APPLICATION SECURITY?" in large, bold, green letters. Below the heading is a graphic of a clock with the word "availability" written around its face. The slide is part of a larger presentation, as indicated by the navigation icons at the top and bottom.

Narration

Let's begin by gaining an understanding of what application security is. Application security simply refers to an application's ability to withstand malicious attack.

For example, how resilient is your application to unauthorized attempts to access customer credit card information? Or how well does your application resist attacks designed to shut it down?

In order to do this, applications are designed and developed against three key security goals: confidentiality, integrity and availability. These are sometimes referred to as the CIA triad.

The goal of confidentiality is to prevent unauthorized disclosure of or access to sensitive data managed by applications, such as personal and financial data.

The goal of integrity is to protect data from unauthorized modification or deletion.

And, the goal of availability is to ensure that the application and its services remain accessible to users.

On Screen Text

What is Application Security?

Why Developing Secure Applications Matters

The screenshot shows a presentation slide with a blue header bar containing the title 'Fundamentals of Application Security'. Below the header, there is a navigation bar with icons for back, forward, search, and other controls. The main content area has a dark background with white text. A single bullet point is displayed:

- **Addressing application security early in the software development lifecycle is much less expensive than fixing problems at later stages**

Narration

High-profile security breaches and the negative press that follows make application security an important business consideration.

Today, it is no longer a matter of if your application will get hacked, but rather when it will happen.

Lack of application security in today's threat landscape places your organization and its customers at risk for significant loss.

Here are some important reasons why you and your organization should be concerned about application security:

The first reason is loss of revenue. An application security breach can cause damage such as disruption of business operations, theft of proprietary data, and, in some cases, regulatory or state fines – all of which can have direct financial impact to your organization.

In addition, your brand can be damaged if customers no longer trust your ability to keep their data safe.

If a security breach involves customers' personal data that gets misused by criminals, this could cause further ill will towards your organization.

In general, customers today are better educated about security, and are demanding that the organizations they do business with meet high application security standards.

This creates competitive pressure: Organizations that don't meet the standard take the risk of losing business to competitors who can use the security of their applications as a market advantage.

Finally, it's important to note that your organization will realize cost savings if you address application security early in the software development lifecycle.

This significantly less expensive than the cost of addressing security after you have been hacked.

On Screen Text

Why Developing Secure Applications Matters

Fundamentals of Application Security

Regulatory, Standards, and State Requirements

The screenshot shows a web-based training interface. At the top left is the logo for 'SECURITY INNOVATION'. To its right is a link 'Move screen reader to main content'. The top navigation bar is blue with the title 'Fundamentals of Application Security'. Below the navigation bar, the page header reads 'Regulatory, Standards, and State Requirements' on the left and '05/60' on the right. The main content area features a large, dark grey rectangular background with a central graphic consisting of a green and blue gradient rounded square containing a white letter 'A' inside a white circle. The overall theme is cybersecurity, indicated by the circuit board pattern in the background and the security-related icons at the top right.

Narration

Depending on your industry, your organization may need to comply with strict regulations, standards, or state requirements regarding application security.

For example, the Payment Card Industry, or PCI, has many application security requirements that apply to all software that handles payment card data.

And, the healthcare and financial industries have their own regulations, standards, and guidelines that organizations must follow.

Failure to meet these requirements can lead to significant monetary fines and business penalties.

For very serious violations, companies could potentially be barred from conducting business altogether.

It is therefore important to become familiar with application security requirements relevant to your industry, in order to remain compliant.

On Screen Text

Regulatory, Standards, and State Requirements

Knowledge Check

The screenshot shows a web-based knowledge check interface. At the top left is the 'SECURITY INNOVATION' logo. To its right is a link 'Move screen reader to main content'. On the far right are three icons: a yellow book-like icon, a green question mark icon, and a red print icon. The main title 'Fundamentals of Application Security' is centered at the top in a blue header bar. Below it, a progress bar shows '06/60'. The main content area contains a question: 'A vulnerable application allows attackers to make unauthorized modifications to data managed by that application. Which security goal have the developers of this application failed to achieve?'. Below the question is a list of four options, each preceded by a blue circular radio button:

- Regulation
- Availability
- Integrity
- Confidentiality

In the bottom right corner of the main content area is a dark grey 'Submit' button.

Narration

On Screen Text

A vulnerable application allows attackers to make unauthorized modifications to data managed by that application.
Which security goal have the developers of this application failed to achieve:

Regulation

Availability

Integrity

Confidentiality

Fundamentals of Application Security

Knowledge Check

The screenshot shows a knowledge check interface. At the top left is the 'SECURITY INNOVATION' logo. To its right is a link 'Move screen reader to main content'. On the far right are three icons: a yellow book, a green question mark, and a red document. The main title 'Fundamentals of Application Security' is centered at the top. In the top right corner, it says '07/60'. Below the title, the question text is: 'It is less costly to address security later in the software development lifecycle when vulnerabilities are known, than to address it earlier when vulnerabilities are unknown.' To the left of the question is a legend: 'True' next to an empty blue circle, and 'False' next to a blue circle with a white dot. In the bottom right corner of the main area is a dark grey 'Submit' button.

Narration

On Screen Text

It is less costly to address security later in the software development lifecycle when vulnerabilities are known, than to address it earlier when vulnerabilities are unknown.

True

False

Understanding the Attacker

The screenshot shows a presentation slide titled "Fundamentals of Application Security". The slide has a blue header bar with the title and a navigation bar with icons for back, forward, and search. Below the header is a sub-header "Understanding the Attacker". The main content area features a dark background with four bullseye targets arranged in a cluster. Two targets are yellow and two are teal. The slide is numbered "08/60" in the top right corner.

Narration

Why do attackers target your applications? It's impossible to know the motives of every individual attacker, but here are some key reasons why attacks on applications are increasing:

Today's digital black market lets attackers convert stolen data into cash.

Attackers and organized crime gangs are highly incentivized to attack applications like yours that may contain valuable data.

For example, stolen credit card numbers could fetch as high as \$100 dollars per number on the black market.

Medical records and social security numbers can fetch even more because users can't simply cancel their medical records or social security number as they could do with a credit card.

A single application could house personal data on thousands, hundreds of thousands, or even millions of people, and could yield an incredible pay day for criminals.

In addition, it no longer takes a lot of skill to attack an application. Attackers have many publicly available tools that they can use to do the work for them.

Also, the number of targets has dramatically increased, and so has their availability online.

If an attacker is unable to compromise one application, there are plenty more targets to try.

Now that you have some understanding of attackers, let's look at how you can manage the application security risk they create.

On Screen Text

Understanding the Attacker

Security Risk Management Concepts and Terms

The screenshot shows a slide from a presentation titled "Fundamentals of Application Security". The title bar includes the "Fundamentals of Application Security" logo and navigation icons. The main content area is titled "Security Risk Management Concepts and Terms". A text box contains the following message: "Here are some important terms and concepts you should understand in order to manage your organization's application security risk." Below this message is a list of seven concepts, each preceded by a yellow circular bullet point:

- **Vulnerability**
- **Exploit**
- **Countermeasure**
- **Threat / Threat Agent**
- **Attack**
- **Asset**
- **Risk**

Narration

Listed here are some important terms and concepts you should understand in order to better manage your organization's application security risks.

Vulnerability:

A vulnerability is a weakness in your application that an attacker could potentially exploit.

This can be weakness in the code itself or perhaps even an application design flaw.

Exploit:

An exploit is a malicious application or a set of commands used to take advantage of vulnerabilities in your application.

The purpose of an exploit is to produce some unexpected or unintended behavior in your application,

such as exposing sensitive information or gaining unauthorized access to certain components.

Countermeasure:

A countermeasure is a step taken to address or reduce the potential for damage caused by a vulnerability in your application code.

Threat / Threat Agent:

A threat is an event that could exploit a vulnerability in your application code and cause some undesirable behavior.

Malicious hackers, or attackers, are often called threat agents because they exploit vulnerabilities in your application code to meet their objectives.

Attack:

An attack is an action that leverages one or more vulnerabilities to realize a threat.

For example, a malicious hacker may exploit a vulnerability in your authentication system to gain unauthorized access to an area of your application.

Asset:

An asset is a resource of value. For example, an asset to your organization may be a database of health or financial information of your customers.

Risk:

Finally, risk is the potential for loss or damage to an asset. Whenever a developer follows insecure coding practices,

Fundamentals of Application Security

they increase the risk that an attacker may be able to exploit the flawed code.

On Screen Text

Security Risk Management Concepts and Terms

Managing Application Security Risks

The screenshot shows a presentation slide with a blue header bar containing the title 'Fundamentals of Application Security'. Below the header, there is a navigation bar with icons for back, forward, search, and print. The main content area has a dark background with white text that reads: 'Mitigation is the most common and recommended way to address application security risks'. At the bottom right of the slide, there is a small circular logo with a gear and a key.

Narration

Now that you are familiar with key security risk management concepts and terms,

it is important to underscore the principle that no application can be completely secure or resilient to all attacks.

You can never completely eliminate the risk that your application may be hacked.

The key to great application security, then, is to correctly manage your risk.

There are four main approaches to managing risk: you can accept, avoid, transfer, or mitigate risk.

Accepting risk means that you take no proactive measures to address your risk and that you are willing to accept the full consequences of a threat to an asset.

This is sometimes a reasonable approach if the cost of addressing the risk greatly exceeds both the value of the asset you wish to protect, and the consequences should the risk be realized.

Usually, you should use this approach only as a last resort.

When accepting risk, you should always have a contingency plan, which is a set of actions you plan to take if the risk is realized.

Avoiding risk is the opposite of accepting risk. To avoid risk, you remove the exposure of an asset to the threat, or you remove the asset entirely.

For example, let's say your application manages customer credit card data, which certainly involves risk.

Your organization can choose to avoid this risk altogether by not handling credit card data at all.

This leads us to the next risk management technique called transference, which means you transfer the burden of the risk onto a third party.

For example, instead of handling card information yourself, you may elect to pay a third-party with expertise in the payment card industry to handle it for you,

thus transferring the risk to them. Insurance is a classic example of transferring risk.

By purchasing insurance against some undesirable event, you're transferring the risk to the insurance company for a fee, to decrease the overall loss should the risk be realized.

The final risk management approach is to mitigate risk, which means that you take proactive steps to reduce an asset's exposure to a risk.

Fundamentals of Application Security

For example, you can reduce the risk of an attacker guessing user account passwords by requiring all users to use strong passwords.

Or, to reduce the chances of an attacker intercepting sensitive data transmitted between your online application and users, you could protect the data in transit by using Transport Layer Security (TLS).

Mitigation is the most common and recommended way to address application security risks.

It should be your primary approach to managing application security risk.

On Screen Text

Managing Application Security Risks

Fundamentals of Application Security

Knowledge Check

The screenshot shows a knowledge check interface. At the top left is the 'SECURITY INNOVATION' logo. To its right is a link 'Move screen reader to main content'. On the far right are three icons: a yellow book-like icon, a green question mark icon, and a red document icon. The main title 'Fundamentals of Application Security' is centered at the top in a blue header bar. Below it, a status bar shows '11/60'. The main content area contains a text paragraph followed by a list of four options in a light blue box. A 'Submit' button is located at the bottom right of the list area.

A company has decided not to address a certain vulnerability in their application, because the cost of the mitigation is far greater than both the value of the asset they wish to protect, and the consequences if the vulnerability is exploited. This company has chosen to:

- Avoid the risk
- Mitigate the risk
- Transfer the risk
- Accept the risk

Submit

Narration

On Screen Text

A company has decided not to address a certain vulnerability in their application, because the cost of the mitigation is far greater than both the value of the asset they wish to protect, and the consequences if the vulnerability is exploited. This company has chosen to:

Avoid the risk

Mitigate the risk

Transfer the risk

Accept the risk

Knowledge Check

The screenshot shows a knowledge check interface. At the top left is the 'SECURITY INNOVATION' logo. A link 'Move screen reader to main content' is visible. The title 'Fundamentals of Application Security' is centered at the top. On the right, there are icons for a book, a question mark, and a document. The main content area contains the following text: 'A malicious script that guesses weak passwords to attempt to gain access to user accounts is an example of:'. Below this is a list of four options, each preceded by a radio button:

- A countermeasure
- A mitigation
- A risk
- An exploit

A 'Submit' button is located in the bottom right corner of the main content area.

Narration

On Screen Text

A malicious script that guesses weak passwords to attempt to gain access to user accounts is an example:

A countermeasure

A mitigation

A risk

An exploit

Module Summary

The screenshot shows a slide from a course titled "Fundamentals of Application Security". The slide has a blue header bar with the title. Below the header, there's a navigation bar with icons for back, forward, search, and print. The main content area has a dark background and features a section titled "UNDERSTANDING APPLICATION SECURITY" in green capital letters. The text discusses what application security is, fundamental goals of confidentiality, integrity, and availability, consequences of failing to address risks, influence of regulations, and reasons for rising attacks. It also mentions risk management terms and the four approaches: acceptance, avoidance, transference, and mitigation. At the bottom of the slide, a note states: "Note - This slide does not contain audio. Please continue to the next section once you have finished reviewing this material."

Narration

On Screen Text

Note – This slide does not contain audio. Please continue to the next section once you have finished reviewing this material

Module Summary

Module Overview and Objectives

The screenshot shows a web-based learning interface. At the top left is the "SECURITY INNOVATION" logo. To its right is a link "Move screen reader to main content". The top navigation bar is blue with the title "Fundamentals of Application Security". Below the navigation, a sub-header "Module Overview and Objectives" is followed by a page number "14/60". The main content area has a dark background with the title "MODULE OVERVIEW" in large, bold, green letters. Below the title is a bulleted list of objectives:

- Understand the anatomy of an application attack
- Understand how to use input validation as a primary application risk mitigation
- Understand key input validation approaches and pitfalls
- Identify common application attacks
- Understand key security principles and best practices for developing secure applications

Narration

This module will introduce you to secure application development principles.

You will learn about the anatomy of an application attack and primary countermeasures to mitigate the risk from those attacks.

You will get an introduction to the most common attacks and you will learn about key secure application development principles and best practices to simplify your secure development efforts.

After completing this module you will be able to: Understand the anatomy of an application attack, understand how to use input validation as a primary application risk mitigation, understand key input validation approaches and pitfalls, identify common application attacks, and understand key security principles and best practices for developing secure applications.

On Screen Text

Module Overview and Objectives

Anatomy of a Basic Application Attack

The screenshot shows a slide titled "INPUT VALIDATION" with the following text: "This process is referred to as Input Validation. Input validation helps verify that input is in a correct and safe format to be used by your application. While not a silver bullet, input validation can be used to help mitigate many common application vulnerabilities." The slide is part of a larger course titled "Fundamentals of Application Security" under the section "Anatomy of a Basic Application Attack". The top navigation bar includes links for "Move screen reader to main content", "Home", "Help", and "Print". The bottom right corner shows "15/60".

Narration

In the previous module, you learned about security risk management,

the need to protect assets, application vulnerabilities, and threats to software security.

Let's now see how these all work together during an attack. By understanding the basic attack anatomy,

you will build the knowledge you'll need to understand most application attacks. Let's begin.

Any method that allows data to be input to your application is considered an interface.

Common types of user interfaces include web forms, command line parameters, and APIs.

An attacker interacts with your application in order to perform nefarious actions.

To do this, attackers identify a vulnerability they can exploit to submit malicious data to the application in order to execute an unintended command or action,

for example, an attacker could supply a malicious data that would cause your application to execute a command, expose sensitive information, or even cause your application to crash making it unavailable to users.

As a developer, it is your job to implement counter measures to ensure your application accepts only valid data and blocks malicious data.

This process is referred to as input validation. Input validation helps verify that input is in a correct and safe format to be used by your application.

While not a silver bullet, input validation can be used to help mitigate many common application vulnerabilities.

What are the primary techniques for validating data? How do you determine if data is valid or not?

This leads to the next topic in our discussion, input validation techniques.

On Screen Text

Anatomy of a Basic Application Attack

Input Validation Techniques

The screenshot shows a presentation slide with a blue header bar containing the title 'Fundamentals of Application Security'. Below the header, there is a dark grey content area. In this area, the text 'There are two primary approaches to validating input:' is followed by a bulleted list. The list includes two main items: 'Whitelist input validation - allows known good input to pass' and 'Blacklist input validation - rejects known bad input'. Under the 'Whitelist' item, there is a secondary bullet point: 'Analyzes input for signs of invalid inputs or attacks'. The slide has a decorative background featuring a circuit board pattern and various icons.

There are two primary approaches to validating input:

- Whitelist input validation - allows known good input to pass
 - Type
 - Range
 - Format
 - Length
- Blacklist input validation - rejects known bad input
 - Analyzes input for signs of invalid inputs or attacks

Narration

The two main approaches to input validation are whitelist and blacklist validation.

Whitelisting allows known good input to pass, and rejects everything else.

It robustly validates input for type, range, format, and length.

Blacklisting rejects only known bad input, and allows everything else.

Its inherent weakness is that the set of all bad inputs is potentially infinite and cannot be completely detected,

whereas the whitelist approach models a finite set and is easier to implement and less prone to error.

Therefore, always use whitelist validation, and use blacklisting only as a supplementary measure.

On Screen Text

Input Validation Techniques

Whitelist Input Validation Example

Move screen reader to main content

Fundamentals of Application Security

Whitelist Input Validation Example 17/60

Simplified example:

Validating a credit card number using whitelist validation

- Type: Numerical characters
- Range: Zero (0) to nine (9)
- Format: Consecutive characters
- Length: Exactly 16 characters

✓ 1234567890112222
✓ 0011602587232456
✗ 0011602587232T56

Narration

Let's look at a simplified example of validating a credit card number using the whitelist validation approach.

As we learned previously, the whitelist approach first defines expected types, ranges, formats and lengths and validates input against that criteria.

In the example of a credit card, we would expect a type of all numerical characters.

The numerical characters would have a range from zero to nine.

The format and length of the input would be 16 consecutive characters.

Here's an example of a fictitious credit card number that matches our input validation type, range, format and length criteria.

Here's another example. Note that it violates our range requirement that the range of each character be from zero to nine.

On Screen Text

Whitelist Input Validation Example

Blacklist Input Validation Example

Move screen reader to main content

Fundamentals of Application Security

Blacklist Input Validation Example

18/60

SCHWARZ BANK

5105 1051 0510 5100
John Q. Cardholder 02/17

Blacklisted user input

- Jason, Jack, Jill, John, Bill
- 123, 456, 123456, ...
- Security, cloud, mobile, ...

✗ sEcurity

✗ 0x0011223344

✗ ;drop table master;--

Narration

To use the blacklist technique to validate a credit card, you must first define a list of inputs that are not valid credit card numbers.

In our example shown here, inputs like Jason, Jack, Jill, 123, or mobile are clearly not valid credit card numbers.

Any input that is not in our pre-defined blacklist is considered a valid input.

It is easy to see that an attacker could bypass this blacklist technique.

An attacker could submit invalid credit card input that is not on the blacklist,

and trick the input validation implementation into thinking a valid credit card input was submitted.

We could add these invalid inputs to our blacklist; however our blacklist set would still be incomplete.

In fact, it would be impossible to create a complete blacklist set because the set of invalid credit card numbers is infinite.

On Screen Text

Blacklist Input Validation Example

Knowledge Check

The screenshot shows a knowledge check interface. At the top left is the 'SECURITY INNOVATION' logo. A blue header bar contains the title 'Fundamentals of Application Security'. On the right side of the header, there are three icons: a yellow book, a green question mark, and a red document. Below the header, a message says 'Move screen reader to main content'. The main content area contains a question: 'When it comes to blacklist input validation, you should:' followed by four options in a list box. A 'Submit' button is located at the bottom right of the content area.

When it comes to blacklist input validation, you should:

- Use blacklist or whitelist validation, but not both
- Use blacklist in conjunction with whitelist, but never blacklist by itself
- Use blacklist in conjunction with whitelist, but never whitelist by itself
- Avoid this input validation technique altogether

Submit

Narration

On Screen Text

When it comes to blacklist input validation, you should:

Use blacklist or whitelist validation, but not both

Use blacklist in conjunction with whitelist, but never blacklist by itself

Use blacklist in conjunction with whitelist, but never whitelist by itself

Avoid this input validation technique altogether

Implementing Whitelist Input Validation with Regular Expressions

The screenshot shows a presentation slide with a blue header bar. The header contains the title 'Fundamentals of Application Security' and the subtitle 'Implementing Whitelist Input Validation with Regular Expressions'. On the right side of the header, there is a progress indicator showing '20/60'. Below the header is a dark gray content area. The content starts with a text block: 'Regular expressions provide a structured method for concise pattern matching and are useful for whitelist input validation implementation.' Below this text, there is a section titled 'To use regular expressions:' followed by a bulleted list:

- Define regular expressions that describe expected input data types, ranges, formats, and lengths where appropriate.
- Compare the input against the corresponding regular expression:
 - If it does not match the regular expression, reject the input.
 - If it matches the regular expression, accept the input.

Narration

Regular expressions provide a structured method for concise pattern matching that you can use to implement whitelist validation.

Several regular expression libraries are available, and many programming languages have built-in regular expression support.

For example, the .NET Framework and Java programming languages natively support regular expressions.

To use regular expressions, you first define regular expressions that describe expected input data types, ranges, formats, and lengths where appropriate.

You then compare the input received by your application against the corresponding regular expression.

If the input does not match the regular expression, it should be considered bad and rejected.

If the input matches the regular expression, then it is considered safe and can be used.

On Screen Text

Implementing Whitelist Input Validation with Regular Expressions

Whitelist Input Validation Example with Regular Expressions

The screenshot shows a presentation slide with the title 'Fundamentals of Application Security' at the top. Below the title, the sub-section 'Whitelist Input Validation Example with Regular Expressions' is displayed. On the left side of the slide, there is an image of a credit card from 'BLANCO BANK'. The card number is 5105 1051 0510 5100, and the expiration date is 02/17. The cardholder name is John Q. Cardholder. To the right of the card, the text 'Simplified example:' is followed by 'Validating a credit card number using whitelist validation with regular expression'. Below this text is a bulleted list:

- Type: Numerical characters
- Range: Zero (0) to nine (9)
- Format: Consecutive characters
- Length: Exactly 16 characters

At the bottom of the slide, there is a visual representation of the regular expression pattern: `^\d{16}\$`. The characters '^', '\d', '{16}', and '\$' are enclosed in green-bordered boxes.

Narration

Here's our credit card example using whitelist validation with regular expressions.

As indicated before, many modern programming languages have built-in support for regular expressions.

Here is a regular expression that implements our whitelist validation criteria.

The first criteria is that the type of the data must be all numerical characters.

A second criteria is that the characters must have a range from zero to nine.

This is implemented using the back-slash d regular expression fragment.

The last two criteria are that the format of the input must be consecutive characters and the length of the input must be exactly sixteen characters.

This is implemented using the highlighted regular expression fragments.

As you can see, it is fairly easy to implement whitelist validation with regular expressions.

However, use of regular expressions does have limitations, which will be discussed next.

On Screen Text

Whitelist Input Validation Example with Regular Expressions

Limitations of Using Regular Expressions

The screenshot shows a presentation slide with a blue header bar containing the title 'Fundamentals of Application Security'. Below the header, there is a sub-section title 'Limitations of Using Regular Expressions'. In the top left corner, there is a logo for 'SECURITY INNOVATION' and a link 'Move screen reader to main content'. On the right side of the slide, there are three small icons: a yellow square, a green circle with a question mark, and a red square. The main content area has a dark background with white text. It starts with a bullet point: 'Limitations of regular expressions include the following:' followed by three more bullet points:

- Regular expressions are not always ideal for validating ranges and length, especially with text that may be expressed in different character encodings.
- More complex data, such as an XML document or a PNG picture, can't be validated using regular expressions. These have to be validated through other means, specific to each type of data.
- Regular expressions are not the best way to validate numbers. Validate numbers using mathematical comparisons.

Narration

In practice, regular expressions are a useful way to implement white-list validation, and can be used to validate the format for most text input. However, they are not always ideal to validate range and length, especially with text that may be expressed in different character encodings.

Further, regular expressions are unable to easily validate more complex data like an XML document or an image file.

Finally, regular expressions are not the best way to validate numbers.

Numbers should be validated by using mathematical comparisons instead.

On Screen Text

Limitations of Using Regular Expressions

Common Sources of Untrusted Data

The diagram illustrates various sources of untrusted data. It is organized into two main columns. The left column includes icons for a terminal window, a folder, a web browser, and a keyboard/mouse. The right column includes icons for a database, network connections, and an HTTP header. Labels next to each icon provide specific names for the data source.

- USER INTERFACES AND COMMAND LINE PARAMETERS
- FILES
- WEB FORM DATA
- DATA FROM EXTERNAL COMPONENTS OR SYSTEMS
- DATABASE QUERIES
- NETWORK DATA
- <HTTP> HTTP HEADERS AND VALUES

Narration

Untrusted data can enter your application through any interface, such as those listed on-screen.

When in doubt, consider all input to be untrusted and properly validate that input before using it in your application.

On Screen Text

Common Sources of Untrusted Data

Notes About Input Validation

The screenshot shows a presentation slide with a blue header bar containing the title 'Fundamentals of Application Security'. Below the header, there is a navigation bar with icons for back, forward, search, and print. The main content area has a dark grey background and contains the following text:

Move screen reader to main content

Notes About Input Validation

24/60

Input validation is not a silver bullet for application security. It helps protect against many known vulnerabilities but not all.

- It does not protect against vulnerabilities such as those caused by weak cryptographic algorithms or race conditions.
- For unstructured input, you need to use additional security measures such as output encoding and data parameterization.
- Not all malicious input comes directly from users. When in doubt always validate input data.

Narration

While input validation is a very effective control for mitigating application security risk, it should not be considered a silver bullet. It helps protect against many known vulnerabilities, but not all.

For example, some application vulnerabilities are not triggered due to malicious input, such as use of weak encryption algorithms, or timing-based vulnerabilities that allow an attacker to act upon a resource ahead of your application. In addition, using input validation is only effective in cases where the input has a defined structure, such as phone numbers, dates and credit card numbers. For unstructured input, such as a user comment, you need to use additional security measures to protect your application, such as output encoding and data parameterization, which will be discussed later in this course.

Finally, note that not all malicious input comes from users.

Malicious data can come from compromised libraries or outside services that your application uses, or even dependencies, such as a third-party data repository.

When in doubt as to whether data is trusted or not, always err on the side of caution and validate input data.

On Screen Text

Notes About Input Validation

Knowledge Check

The screenshot shows a knowledge check interface. At the top left is the 'SECURITY INNOVATION' logo. To its right is a link 'Move screen reader to main content'. On the far right are three icons: a yellow book, a green question mark, and a red document. The main title 'Fundamentals of Application Security' is centered at the top in a blue bar. Below it, the page number '25/60' is visible. The main content area contains the question 'Input validation:' followed by a list of four options, each preceded by a blue radio button. A 'Submit' button is located in the bottom right corner of the content area.

Input validation:

- Only verifies that the user is not an attacker
- Only validates that the user has entered input in the expected sequence
- Only validates that the user has entered required number of characters
- Helps validate that input is in a correct and safe format before it used by your application

Submit

Narration

On Screen Text

Input validation:

Only verifies that the user is not an attacker

Only validates that the user has entered input in the expected sequence

Only validates that the user has entered required number of the characters

Helps validate that input is in a correct and safe format before it used by your application

Knowledge Check

The screenshot shows a knowledge check interface. At the top, there's a header bar with the 'SECURITY INNOVATION' logo, a 'Move screen reader to main content' link, and three icons (book, question mark, and print). The main title 'Fundamentals of Application Security' is centered at the top of the page. Below the title, a sub-header reads 'The whitelist validation approach checks for:'. A list of four options is provided, each preceded by a blue circular checkbox:

- Known bad exploitation signatures.
- Expected formats, types, ranges and lengths.
- Known hacker or threat agents.
- Known formats and attack signatures.

In the bottom right corner of the main content area, there is a dark grey 'Submit' button.

Narration

On Screen Text

The whitelist validation approach checks for:

- Known formats and attack signatures.
- Known bad exploitation signatures.
- Known hacker or threat agents.
- Expected formats, types, ranges and lengths

Common Application Attacks

The screenshot shows a slide from a presentation. At the top left is the 'SECURITY INNOVATION' logo. To its right is a link 'Move screen reader to main content'. On the far right are three icons: a yellow square, a green circle with a question mark, and a red square with a minus sign. The title 'Fundamentals of Application Security' is at the top center. Below it, the section title 'Common Application Attacks' is in blue. A text block says: 'Let's review a sampling of common application attacks:' followed by a bulleted list: '• Buffer overflows', '• SQL Injection attacks', and '• Race conditions'. Below this, another text block says: 'Then, we will look at resources for learning more about these and other application attacks'.

Narration

Now that you're familiar with the basic anatomy of an application attack and how input validation plays a role in significantly reducing the risk to your application from attack, let's turn our attention to looking at the common real-world attacks listed on-screen. This list is a very small subset of the total application attacks in the wild. The goal here is to simply become familiar with common application attacks and how attackers use them. We'll then discuss resources that are available for learning more about these and other application attacks.

On Screen Text

Common Application Attacks

Let's review a sampling of common application attacks:

- Buffer overflows
- SQL Injection attacks
- Race conditions

Then, we will look at resources for learning more about these and other application attacks

Buffer Overflows

The screenshot shows a presentation slide with a blue header bar containing the title 'Fundamentals of Application Security'. Below the header, the slide has a white background with a large, dark grey rectangular area. In the center of this area, there is a red curved arrow pointing from a small red box containing the number '394' towards a teal-colored cup-like container. The slide is part of a larger interface with a navigation bar at the top featuring icons for back, forward, search, and help, and a status bar at the bottom right indicating '28/60'.

Narration

Buffer overflows occur in applications where more data is written into a buffer than that buffer has been allocated to handle.

The result is that memory adjacent to the buffer gets overwritten, which typically causes the application to behave incorrectly or crash.

Buffer overflows can be exploited whenever an application attempts to write untrusted data into a fixed-length buffer without first validating the input or accurately calculating the buffer size.

On Screen Text

Buffer Overflows

Stack-Based Overflows

The diagram illustrates a stack-based overflow attack. On the left, a vertical stack frame is shown with memory addresses increasing downwards. It contains four sections: 'Buffer' (yellow), 'Frame Pointer' (green), 'Return Address' (light green), and 'Parameters' (cyan). A red arrow points from a speech bubble containing the code 'strcpy(buffer, verylongstring)' towards the 'Return Address'. To the right of the stack frame is a teal square icon with a white letter 'A' inside, representing the source of the attack. A green arrow labeled 'Increasing memory address...' points downwards along the left side of the stack frame.

Narration

Stack-based buffer overflows are caused by the fact that data is written into a local buffer without proper bounds checking.

When a function is called, a stack frame is pushed onto the program stack.

The program stack consists of the function's parameters, the return address, the frame pointer, and the function's local variables.

When a local buffer is overflowed, following memory locations on the stack including the return address of the function end up being overwritten by user data.

This poses a security issue because the return address determines the location of the next instruction to be executed after the function returns.

This means that if an attacker can control the value of a function's return address he can potentially control execution of the vulnerable program.

On Screen Text

Stack-Based Overflows

Mitigating Risk from Buffer Overflow Attacks

The screenshot shows a slide from a presentation. At the top left is the 'SECURITY INNOVATION' logo. To its right is a link 'Move screen reader to main content'. On the far right are three icons: a yellow square with a gear, a green square with a question mark, and a red square with a minus sign. The title 'Fundamentals of Application Security' is at the top center. Below it, the slide title 'Mitigating Risk from Buffer Overflow Attacks' is displayed. In the bottom right corner of the slide area, the text '30/60' indicates the slide's position in the sequence.

Buffer overflow vulnerabilities are not just stack-based, they can also be exploited on memory heaps.

To mitigate the risk from buffer overflow attacks, developers need to verify that the length of the data being copied does not exceed the maximum size of the receiving buffer.

Narration

Keep in mind that buffer overflow vulnerabilities are not just stack-based, they can also be exploited on memory heaps.

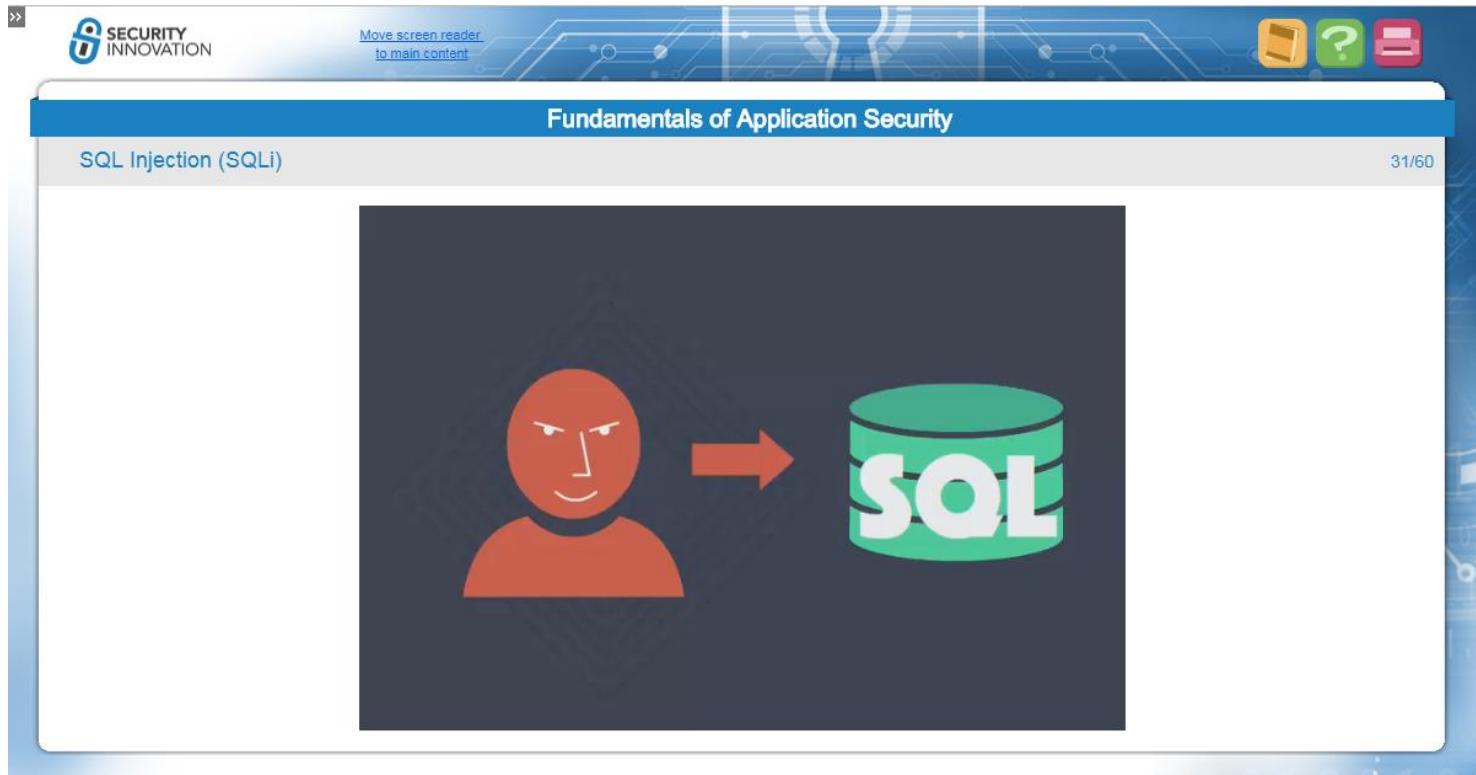
To mitigate the risk from buffer overflow attacks, developers need to implement proper input validation in their applications. Specifically, they need to verify that the length of data being copied into a static buffer does not exceed the maximum size of that static buffer.

If it does, then there may be an exploitable buffer overflow vulnerability in their application code.

On Screen Text

Mitigating Risk from Buffer Overflow Attacks

SQL Injection (SQLi)



Narration

The next attack we will discuss is SQL injection, or SQLi, which is a technique that allows attackers to inject SQL commands into dynamic SQL statements. These dynamic SQL statements are then executed by backend databases, enabling attackers to potentially execute commands in the context of the database.

SQL injection vulnerabilities are yet another example of attacks that are possible due to the failure of applications to validate un-trusted data.

SQL Injections are one of the most common attacks today. One reason is the ubiquity of SQL Data Stores, which increases the SQL injection attack surface.

Another reason may be that many applications are moving toward the Web,

which makes them more accessible to attack than traditional applications that reside within restricted internal networks.

Finally, nearly all modern day database management systems support SQL.

This means that malicious users need only learn the SQL language once and then they are able to apply those attack skills to virtually any database management system, regardless of the vendor platform (Microsoft, Oracle, IBM, MySQL, and so on).

Let's take a closer look at this attack pattern to review our understanding of this type of attack.

On Screen Text

SQL Injection (SQLi)

Fundamentals of Application Security

When Does SQL Injection Occur?

The screenshot shows a presentation slide with a blue header bar containing the title 'Fundamentals of Application Security' and a navigation bar with icons for search, help, and print. Below the header is a sub-header 'When Does SQL Injection Occur?'. The main content area contains a text box stating: 'SQL injection occurs whenever untrusted malicious input data is used to construct and execute dynamic SQL statements'. Below this text is a diagram illustrating the process. It features a red cartoon character on the left with a speech bubble containing the malicious input: 'My name is: 0; DROP DATABASE SI--'. An arrow points from the character to a green square icon with a white letter 'A' in the center. Another arrow points from the 'A' icon to a green and yellow skull-and-crossbones icon on the right. Above the 'A' icon is another speech bubble containing the resulting dynamic SQL statement: 'SELECT * FROM Users WHERE Name =0; DROP DATABASE SI--'.

Narration

SQL injection vulnerabilities in code are created whenever untrusted input data is used to construct and execute dynamic SQL statements.

Let's examine a common scenario where an application reads data from a user,

constructs a dynamic SQL statement based on that data, and then forwards that dynamic SQL statement to a backend database for execution.

First, a malicious user locates an input into the application that is used to create a dynamic SQL statement.

For example, an input field where the user is supposed to enter their user name.

The malicious user instead enters "0; DROP DATABASE SI",

which contains malicious SQL commands to drop the database called SI.

The application reads that input, fails to validate it and constructs a dynamic SQL statement.

In actuality, two dynamic SQL statements were created.

The first is to select everything from the Users table where the Name column equals 0,

and the second is a malicious command which instructs the executing database engine to drop or delete the database called SI.

The application forwards the SQL statements to the database,

which executes both commands. As a result, the database called SI is dropped.

On Screen Text

When Does SQL Injection Occur?

Preventing SQLi: Use Parameterized APIs

The screenshot shows a slide from a presentation titled "Fundamentals of Application Security". The title bar includes the course name and a progress indicator "33/60". The main content area has a dark background with the title "PARAMETERIZED APIs" in large green letters. Below the title, a list of steps for using parameterized APIs is provided:

- To properly use parameterized APIs for database access, you should:
- Identify available database access APIs. Review the database API used by your application and determine if parameterized statements are supported.
- Choose whether to use parameterized statements or parameterized stored procedures. Parameterized statements are typically easier to use, but if those are unavailable then use parameterized stored procedures instead.
- Identify all the SQL queries performed by your application.
- Once you have located all SQL queries, identify the parameters in each query for their format and type.
- Use your chosen parameterized API to execute each SQL query made by your application.

Narration

While input validation can be used to mitigate the risks from many attacks, there may be times where input validation alone may not be sufficient.

In the case of SQL injection, it may be hard to validate input where the structure of the input is unknown or undefined, such as in a product review.

As a good security practice, type-safe SQL command parameters should be used in all situations where any variables are included in a SQL statement.

Type-safe SQL command parameters indicate to the database engine that the data defined in certain parameters are values only and should never be executed.

Marking input data as non-executable neutralizes any SQL commands that a malicious user may try to inject and is extremely effective for mitigating risk from SQL injection attacks.

To properly use parameterized APIs for database access, you should:

Identify available database access APIs.

Review the database API used by your application and determine if parameterized statements are supported.

Choose whether to use parameterized statements or parameterized stored procedures.

Parameterized statements are typically easier to use,

but if those are unavailable then use parameterized stored procedures instead.

Identify all the SQL queries performed by your application.

Once you have located all SQL queries, identify the parameters in each query for their format and type.

Use your chosen parameterized API to execute each SQL query made by your application.

On Screen Text

Preventing SQLi: Use Parameterized APIs

Race Conditions

The screenshot shows a slide titled "Race Conditions" from a course titled "Fundamentals of Application Security". The slide contains the following text and icons:

Race conditions occur whenever the timing of certain events has security implications for an application.

- Open a network socket
- Make a request for a certain file.
- Read the file data.
- Save the file data locally.
- Execute or open the file.

Icons representing these actions include a network socket, a question mark, a circular arrow, a floppy disk, and a greater than sign with a minus sign.

Narration

The next vulnerability we'll discuss is race conditions.

All applications can be decomposed into a series of steps that need to be executed in some order to perform a given function.

For instance, an application may be programmed to download and execute a file from the Internet,

in which case the steps to fulfill that function could be decomposed into the following steps:

Open a network socket.

Make a request for a certain file.

Read the file data.

Save the file data locally.

Execute or open the file.

When the timing of certain application events has security implications,

the possibility for an application security vulnerability known as "race conditions" is created.

Note that race conditions are an example of a security vulnerability that is not caused by failure to validate untrusted input data.

On Screen Text

Race Condition

Race Condition Example

The screenshot shows a slide from a presentation titled "Fundamentals of Application Security". The slide has a blue header bar with the title and a navigation bar with icons for back, forward, search, and help. Below the header is a main content area with a dark background. In the center, there is a diagram illustrating a race condition. On the left, a teal icon labeled 'A' represents an application. A green arrow points from this icon to a central white document icon, which is enclosed in a green dashed box labeled 'ACL'. To the right of the document is a red silhouette of a person representing a user. A yellow arrow points from the user towards the document. The overall theme is security and access control.

Narration

Here is an example of a race condition. In this example, the application is storing potentially sensitive data in a temporary file.

To do this, the application performs the following simple steps:

It creates and saves sensitive information to a temporary file.

It applies a security access control list (ACL) so that only the application can access the temporary file.

Let's see how a malicious user can exploit this application behavior.

First the application creates and saves sensitive information to a temporary file.

Before the application has a chance to apply an ACL that would allow access to the file only by the application, a malicious user copies the file.

The application applies the ACL to the file,

not knowing that a malicious user has already attained a copy of the temporary file containing sensitive information.

On Screen Text

Race Condition Example

Time of Check Time of Use (TOCTOU) Race Conditions

The screenshot shows a presentation slide with a blue header bar containing the title 'Fundamentals of Application Security'. Below the header, the slide has a dark grey background with white text. The text defines TOCTOU race conditions as follows:

In general, race conditions are specific to individual applications and the events and steps they implement. However, there is a type of race condition called Time of Check Time of Use that is broadly applicable and fairly common.

Time of check time of use (TOCTOU) is a type of race condition that occurs when the application checks for a certain condition, and then performs a security action based on that condition.

At the top of the slide, there is a navigation bar with icons for back, forward, search, and other presentation controls. A small text link 'Move screen reader to main content' is also visible in the top left corner.

Narration

In general, race conditions are specific to individual applications and the events and steps they implement.

However, there is a type of race condition called Time of Check Time of Use that is broadly applicable and fairly common.

TOCTOU race conditions are a category of race conditions that occurs when an application checks for a certain condition, and, based on the result of that condition check, performs a corresponding security action.

Let's take a look at an example to gain a better understanding of TOCTOU issues.

On Screen Text

Time of Check Time of Use (TOCTOU) Race Conditions

TOCTOU Example

The screenshot shows a presentation slide titled "TOCTOU EXAMPLE: INSTALLING PATCHES". The slide features a dark background with several icons: a file labeled "Patch.exe", a green rounded square containing a white letter "A", a red user icon, and two files with a double-headed arrow between them. A large green arrow points from the "A" icon towards the "Patch.exe" file. The slide is part of a larger presentation with a blue header bar containing the title "Fundamentals of Application Security", the page number "37/60", and other navigation icons.

Narration

In this example, the application has been designed to automatically check for a valid digital signature when installing patches.

So the series of steps or events that the application implements to fulfill this scenario include:

Download and save the patch file.

Verify the digital signature of the patch file.

If the digital signature is invalid or missing, delete the patch file.

Let's see how a malicious user can use a race condition to exploit this scenario:

The application downloads and saves the patch file.

The application reads and verifies the digital signature of the file.

The digital signature is valid. Before the application can execute the patch file,

a malicious user swaps the patch file with an infected version.

The application executes the file that it believes to be un-tampered with and is subsequently compromised.

On Screen Text

TOCTOU Example

Best Practices for Reducing Risk From Race Conditions

The screenshot shows a presentation slide with a blue header bar containing the title 'Fundamentals of Application Security'. Below the header, there is a sub-section title 'Best Practices for Reducing Risk From Race Conditions'. A large text box contains the following content:

Race conditions are difficult to prevent entirely, but follow these best practices to reduce risk:

- Ensure that the application has exclusive rights to a resource before operating on the resource.
- Ensure that the application has exclusive rights on the container in which a resource resides.
- Ensure that the container or resource being used was created by the application itself or by a trusted source.
- First set ACLs on the container, and then create the resource within the protected container.
- Use random container and resource names whenever possible.

Narration

Since race conditions are highly specific to individual applications, there is no single technology or library that developers can use to reduce the risk from race conditions, as with the case of other vulnerabilities, such as SQL injection.

There are however best practices that developers can follow that can help reduce the risk from race condition attacks. They are:

If an application relies on a resource, make sure that the application has attained exclusive rights to that resource prior to use, and that the application has attained exclusive rights to the container that holds the resource.

For example, in the case of a file-based resource, ensure that the application has exclusive rights to the directory that contains the file.

If an application relies on a container or resource, ensure that the container or resource was created by the application itself or by a trusted source.

Going back to the example of a directory and file, ensure that it was the application or some other trusted source that created the directory and/or file.

When creating a container to hold a resource, such as a file,

set the ACLs on the container first and then create the file within the protected container.

Use random names when creating a container or resource.

There is no one fix-all solution for race conditions;

however if you follow the tips provided above when implementing your applications

the overall risk from race condition attacks should be significantly reduced.

On Screen Text

Best Practices for Reducing Risk From Race Conditions

Fundamentals of Application Security

The STRIDE Model

The screenshot shows a presentation slide titled "Fundamentals of Application Security". The slide has a dark blue header with the title and a light blue footer. The main content area is a dark grey box containing the title "STRIDE MODEL" in green. Below it is a bulleted list of six threat types:

- Spoofing
- Tampering
- Repudiation
- Information Disclosure
- Denial of Service
- Elevation of Privilege

Narration

We've looked at buffer overflows, SQL injection, and race conditions,

which are just a few of the many security threats that you will confront as a secure application developer.

A valuable tool to help you understand, classify, and mitigate the broad spectrum of application security threats is known as the STRIDE model.

The STRIDE acronym stands for six categories of threats: spoofing, tampering, repudiation, information disclosure, denial of service, and elevation of privilege.

Spoofing threats involve an adversary impersonating someone or something to fool and exploit people or systems.

Tampering threats involve an adversary modifying data, usually as it flows across a network, in memory, or on disk.

Repudiation threats involve an adversary who performs an action and then plausibly denies having performed the action.

Information disclosure threats involve data that can be accessed by someone who should not have access.

Denial-of-service threats involve denying legitimate users access to a system or component.

Elevation of privilege threats involve a user or component being able to access data or programs for which they are not authorized.

You can use the STRIDE model to anticipate the types of threats that can affect your application.

To apply STRIDE, it is useful to gather program managers, developers, and testers together to discuss and identify potential threats and vulnerabilities.

Testers, with their focus on how things go wrong, are often best suited to drive this meeting.

After the group has finished enumerating the threats, a program manager should create a list of all the elements from the diagram and list the threats that apply to each element.

On Screen Text

The STRIDE Model

Secure Application Development Principles and Best Practices

The screenshot shows a slide from a presentation titled "Fundamentals of Application Security". The slide has a blue header bar with the title and a sub-header "Secure Application Development Principles and Best Practices". Below the header is a dark gray content area containing text and a bulleted list. The text reads: "Secure application development principles are coding practices that can greatly increase the security of your application. Key principles include:" followed by a bulleted list of six items. The slide is part of a larger course interface, with a navigation bar at the top and a progress bar at the bottom.

Secure application development principles are coding practices that can greatly increase the security of your application.

Key principles include:

- Use input validation
- Use standard security libraries
- Use language, compiler, and platform protection
- Perform code analysis and code review
- Leverage vendor technology security guidance
- Leverage application security models, standards, and guidelines

Narration

Secure coding principles are coding practices which, when followed, can greatly increase the security of an application.

There is no one principle that will render your application free from security vulnerabilities,

together these key principles can help you greatly improve your overall security posture:

Use input validation

Use standard security libraries

Use language, compiler, and platform protection

Perform code analysis and code review

Leverage vendor technology security guidance

Leverage application security models, standards, and guidelines

On Screen Text

Secure Application Development Principles and Best Practices

Input Validation Best Practices

The screenshot shows a presentation slide with a blue header bar containing the title 'Fundamentals of Application Security'. Below the header, there is a navigation bar with icons for back, forward, search, and other controls. The main content area has a dark background and contains a bulleted list of best practices for input validation:

- Input validation
 - Verifies that input is in a correct and safe format before it is used by your application
 - Critical to any application security strategy
 - Considered the most effective method for mitigating risk from nearly every known application vulnerability
- Validate all untrusted input
- Implement input validation that is well-tailored to the specific needs of your application
- Always perform validation on the server side—the client side is too vulnerable to be relied upon for security
 - Input passes both client and server validation = likely that it is safe
 - Input passes client validation but fails on the server side = likely sign of an attack

Narration

Recall that input validation verifies that input is in a correct and safe format before it is used by your application.

It is a critical component of any application security strategy and when properly implemented,

it is considered the most effective method for mitigating risk from nearly every known application vulnerability.

As a general rule, any untrusted input should be validated. When in doubt, perform input validation anyways.

Be sure to implement input validation that is well tailored to the needs of your application.

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

So, use routines that cover the types of input that your application will need to handle.

Always perform validation on the server side. This is because the client side is too vulnerable to be relied upon for security—it may 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, it's a likely sign of an attack.

Note that validating on the client-side is acceptable, as long as it is used as a supplement to robust server side validation.

On Screen Text

Input Validation Best Practices

Use Standard Security Libraries

The screenshot shows a presentation slide titled "Fundamentals of Application Security" with the sub-section "Use Standard Security Libraries". The slide features a dark blue background with three yellow icons representing different types of security libraries:

- SECURE COMMUNICATION LIBRARIES**: Represented by a yellow icon with a Wi-Fi signal.
- CRYPTOGRAPHIC LIBRARIES**: Represented by a yellow icon with binary code (0101010101...).
- ENCODING LIBRARIES**: Represented by a yellow folder icon.

The top of the slide includes the "Fundamentals of Application Security" header, the page title "Use Standard Security Libraries", and a page number "42/60". The top right corner has icons for a book, a question mark, and a document.

Narration

When implementing security controls in your application, there is often a standard library already written for you.

For example, Web applications are often susceptible to an attack called a cross-site scripting attack.

To mitigate the risk from these attacks, developers should use encoding libraries.

Cryptographic algorithms are the primary way with which applications protect the sensitive data they manage.

Always be sure to use a standard cryptographic library and avoid developing your own at all costs; and use secure communications libraries.

Not only does using a standardized library help ensure correctness of security mitigations and controls,

it also has the benefit of reducing your overall workload because you benefit from the work already done for you by security experts.

On Screen Text

Use Standard Security Libraries

Use Language, Compiler and Platform Protection

The screenshot shows a slide from a presentation titled "Fundamentals of Application Security". The slide has a blue header bar with the title. Below the header, there's a navigation bar with icons for back, forward, search, and print. The main content area has a dark grey background and contains text about security features of ASP.NET and Visual C++. It also includes a note about leveraging language/platform capabilities.

ASP.NET

- Includes Request Validation feature that inspects all HTTP requests
- Helps automatically detect cross-site scripting attacks

Visual C++

- Includes compiler /GS option
- Provides limited run-time protection against stack-based overflow

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

Narration

Programming languages and technology platforms often have built-in security features that you can use to reduce your overall workload in writing secure applications.

For example, Microsoft's ASP.NET has a built-in mechanism called Request Validation that detects certain cross-site scripting attacks, a very common Web attack.

Or if you are writing an application in C/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 to understand the full security capabilities of the language and platform you are developing your secure application on and leverage those capabilities whenever possible.

On Screen Text

Use Language, Compiler and Platform Protection

Knowledge Check

The screenshot shows a knowledge check interface. At the top left is the 'SECURITY INNOVATION' logo. To its right is a link 'Move screen reader to main content'. On the far right are three icons: a yellow book, a green question mark, and a red document. The main title 'Fundamentals of Application Security' is centered at the top in a blue bar. Below it, the page number '44/60' is visible. The main content area contains the following text: 'When performing input validation, you should only perform client-side validation.' Below this text is a question with two options: 'True' and 'False', each preceded by a radio button. A 'Submit' button is located in the bottom right corner of the main content area.

When performing input validation, you should only perform client-side validation.

True
 False

Submit

Narration

On Screen Text

When performing input validation, you should only perform client-side validation.

True

False

Knowledge Check

The screenshot shows a knowledge check interface. At the top left is the 'SECURITY INNOVATION' logo. To its right is a link 'Move screen reader to main content'. On the far right are three icons: a yellow book, a green question mark, and a red document. The main title 'Fundamentals of Application Security' is centered at the top. Below it, a sub-section title reads 'Leveraging language, compiler, and platform protection enables application developers to:'. A list of four options follows, each preceded by a blue circular checkbox:

- Automatically quarantine attack data
- Replace the need for input validation and other security controls
- Auto-correct vulnerabilities in application code
- Supplement existing security best practices and controls with automated protection

In the bottom right corner of the main content area is a dark grey 'Submit' button.

Narration

On Screen Text

Leveraging language, compiler, and platform protection enables application developers to:

Automatically quarantine attack data

Replace the need for input validation and other security controls

Auto-correct vulnerabilities in application code

Supplement existing security best practices and controls with automated protection

Knowledge Check

The screenshot shows a knowledge check interface. At the top left is the 'SECURITY INNOVATION' logo. To its right is a link 'Move screen reader to main content'. On the far right are icons for a mobile device, a question mark, and a document. The main title 'Fundamentals of Application Security' is centered at the top. In the top right corner, it says '46/60'. Below the title, a question asks: 'Programming languages and technology platforms often include built-in security features that:'. Four options are listed in a box with blue circular checkboxes:

- Provide protection against a limited set of application security attacks and vulnerabilities
- Provide protection against all known application security attacks and vulnerabilities
- Provide protection only against elevation of privilege attacks and vulnerabilities, such as buffer overflows
- Provide protection only against attacks on applications built with Microsoft Visual C++ or ASP.NET

In the bottom right corner of the main area is a 'Submit' button.

Narration

On Screen Text

Programming languages and technology platforms often include built-in security features that:

Provide protection against limited set of application security attacks and vulnerabilities

Provide protection against all known application security attacks and vulnerabilities

Provide protection only against elevation of privilege attacks and vulnerabilities, such as bugger overflows

Provide protection only against attacks on applications built with Microsoft Visual C++ or ASP.NET

Use Code Analysis and Code Review

The screenshot shows a slide from a presentation. At the top left is the 'SECURITY INNOVATION' logo. To its right is a link 'Move screen reader to main content'. On the far right are three icons: a yellow square, a green square with a question mark, and a red square with a minus sign. The title 'Fundamentals of Application Security' is at the top center. Below it, the slide is titled 'Use Code Analysis and Code Review'. In the center is a graphic featuring a green user icon, a magnifying glass, and a clipboard with a checklist.

Narration

The next secure development principle is to use code analysis tools and code review.

Code analysis tools are software-based tools that inspect application source code in either compiled or non-compiled forms for known defects.

These tools can be categorized into static analysis and binary analysis.

The details of these two tool categories will be discussed momentarily.

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

Both code analysis and code review are an important component of any secure application development effort.

They allow you to detect and correct vulnerabilities in legacy applications,

as well as detect and correct vulnerabilities in current applications fairly early in the development lifecycle,

where the cost of fixing vulnerabilities is low.

On Screen Text

Use Code Analysis and Code Review

Understanding Code Analysis

The screenshot shows a presentation slide titled "Fundamentals of Application Security" with the sub-section "Understanding Code Analysis". The main diagram illustrates the compilation process. On the left, a green folder icon with code symbols (</>) represents "Source code". An arrow points from it to a teal factory-like icon labeled "Compiler and Linker". From the factory, another arrow points to a yellow arrow pointing right, labeled "Binary analysis tools". Above the factory, a white box labeled "Static code analysis tools" has a speech bubble pointing towards the source code icon. To the right of the factory, a white box labeled "Binary analysis tools" has a speech bubble pointing towards the output binary code. The output binary code is shown as a vertical column of binary digits: 010101010101, 001010101001, 010100101010, 010101010010, 101010100101, 010101100111.

Narration

Let's first focus our attention on code analysis.

As mentioned earlier, there are two categories of code analysis tools: static analysis and binary analysis.

In order to better understand the difference between the two, consider the application compilation lifecycle.

All applications are first expressed as source code.

These are the human readable instructions created by developers that implement software, such as Java code.

A compiler and linker takes that source code and turns it into machine code, often called an application binary.

The key difference between static analysis tools and binary analysis tools is the input into these tools.

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

Each have specific benefits and limitations which we will discuss next.

On Screen Text

Understanding Code Analysis

Static Code Analysis Pros and Cons

The screenshot shows a presentation slide titled "STATIC ANALYSIS". The slide is divided into two main sections: "PROS" (left) and "CONS" (right). The "PROS" section lists six benefits with green bullet points: "Scales easily", "Objective", "Mature technology", "Find more vulnerabilities", "Provide more accuracy", and "Easier for developers to debug issues". The "CONS" section lists three limitations with red bullet points: "Language specific; may require multiple tools", "Risk of false positives and false negatives", and "Detects implementation flaws only". The slide has a blue header bar with the title and a navigation bar at the top.

PROS	CONS
<ul style="list-style-type: none">▶ Scales easily▶ Objective▶ Mature technology▶ Find more vulnerabilities▶ Provide more accuracy▶ Easier for developers to debug issues	<ul style="list-style-type: none">▶ Language specific; may require multiple tools▶ Risk of false positives and false negatives▶ Detects implementation flaws only

Narration

We'll first take a look at static analysis tools to understand some of their benefits and limitations.

Let's begin on the "pros" side: Compared to manual code reviewers, static analysis software tools can scale fairly easily to analyze large amounts of code, and they are objective and unbiased.

When compared to binary analysis, static analysis technology is generally more mature.

Static analysis tools tend to find more vulnerabilities and provide more accuracy than binary analysis tools.

Finally the issues identified by static analysis tools are easier to debug, since they work with human readable source code rather than compiled code.

Static analysis tools also have certain limitations. First, they read source code and are language specific, so organizations that use many programming languages may require many static analysis tools.

In addition, they may produce false positives by flagging items that are not vulnerabilities, and produce false negatives by missing actual vulnerabilities.

This often creates more work for development teams, and can eventually degrade confidence in these tools.

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

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

For example, if an application incorrectly assigns excessive rights to a user contrary to the intended design,

or if an application contains a bug where certain operations are performed out of order, a static analysis tool might not be able to detect it.

On Screen Text

Static Code Analysis Pros and Cons

Binary Analysis Pros and Cons

The screenshot shows a presentation slide with a blue header bar containing the title 'Fundamentals of Application Security'. Below the header is a navigation bar with icons for back, forward, search, and print. The main content area has a dark background with the title 'BINARY ANALYSIS' in large green letters at the top. Below the title is a grid divided into two columns: 'PROS' on the left and 'CONS' on the right. The 'PROS' column contains three green bullet points: '▶ Scales easily', '▶ Objective', and '▶ More accurate'. The 'CONS' column contains five red bullet points: '▶ Less mature technology', '▶ Difficult to debug identified issues', '▶ Language specific; multiple tools may be required', '▶ Risk of false positives and false negatives', and '▶ Detects implementation flaws only'. At the bottom of the slide, there is a footer bar with the text 'Binary Analysis Pros and Cons' on the left and '50/60' on the right.

PROS	CONS
<ul style="list-style-type: none">▶ Scales easily▶ Objective▶ More accurate	<ul style="list-style-type: none">▶ Less mature technology▶ Difficult to debug identified issues▶ Language specific; multiple tools may be required▶ Risk of false positives and false negatives▶ Detects implementation flaws only

Narration

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.

And, binary analysis tools are more accurate than static analysis tools.

This is because binary analysis tools operate on the actual code that gets executed,

whereas static analysis tools operate on source code, which may get optimized and re-arranged by compilers.

However, binary analysis tools do have limitations.

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

Also, the bugs identified by binary analysis tools are difficult to debug,

since developers have to be able to interpret compiled code.

Finally, binary analysis tools suffer from the same limitations as static analysis tools:

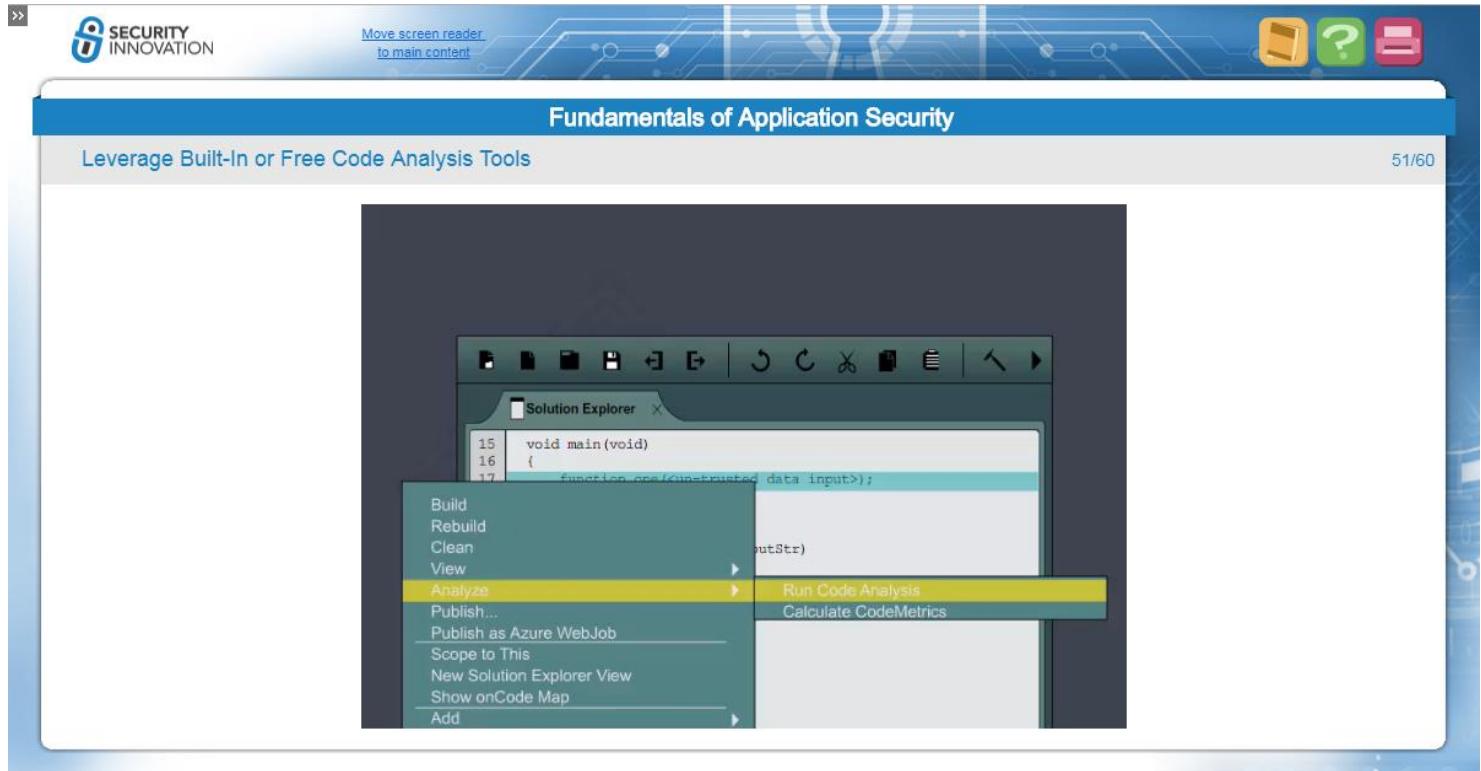
They are language-specific, so organizations may require many tools;

they produce false positives and false negatives; and they are unable to detect issues other than implementation flaws.

On Screen Text

Binary Analysis Pros and Cons

Leverage Built-In or Free Code Analysis Tools



Narration

It is important to note that many Integrated Development Environments (IDEs) now include built-in code analysis tools.

While these built-in tools may not find quite the comprehensive set of vulnerabilities as their commercial counterparts, they can often help you identify common vulnerabilities in code at no extra cost to you.

On Screen Text

Leverage Built-In or Free Code Analysis Tools

Understanding Code Review

The screenshot shows a slide titled "CODE REVIEW". On the left, there's a green icon of a document with '</>' symbols. A speech bubble above it contains the text "Code review". To the right of this is a blue factory-like icon with a yellow arrow pointing to a column of binary code. The binary code is:

```
010101010101  
001010101001  
010100101010  
010101010010  
101010100101  
010101100111
```

The slide is titled "Fundamentals of Application Security" and is part of a section titled "Understanding Code Review". The top navigation bar includes icons for a screen reader, search, and help, along with the "SECURITY INNOVATION" logo.

Narration

Let's now turn our attention back to the application compilation lifecycle, in order to understand code review.

Recall that applications are first expressed as source code.

The source code is processed by a compiler and linker to produce binary code.

Code review, like static analysis, uses 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

The screenshot shows a presentation slide titled "CODE REVIEW". The slide is divided into two main sections: "PROS" on the left and "CONS" on the right, each containing a bulleted list of points. The "PROS" section includes: "Fewer false negatives and false positives" and "Can more easily identify implementation and design flaws". The "CONS" section includes: "Time consuming, difficulty scaling", "Highly susceptible to fatigue and human error", and "Highly subjective". The slide has a dark background with white text and a light blue border. At the top of the slide, there is a header bar with the title "Fundamentals of Application Security" and a sub-header "Code Review Pros and Cons". On the far left, there is a small icon for "SECURITY INNOVATION". On the right side of the slide, there are three small icons: a yellow square with a question mark, a green square with a checkmark, and a red square with a minus sign. The overall background of the slide is a stylized circuit board pattern.

Narration

Manual code review has benefits and limitations of its own.

To begin with the benefits: Manual code reviews have the distinct advantage over code analysis of generally producing fewer false negatives and false positives.

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

While manual code review provides some great benefits, it does come with certain limitations.

First, manual code review is very time consuming and does not scale very well.

Another limitation is that reviewers can become fatigued from reviewing large amounts of code, which diminishes their ability to accurately identify vulnerabilities.

In addition, different reviewers have different areas of expertise, so different vulnerabilities may get identified depending on who reviews the code.

On Screen Text

Code Review Pros and Cons

Fundamentals of Application Security

Knowledge Check

The screenshot shows a knowledge check interface. At the top left is the 'SECURITY INNOVATION' logo. To its right is a link 'Move screen reader to main content'. On the far right are three icons: a yellow book, a green question mark, and a red print symbol. The main title 'Fundamentals of Application Security' is centered at the top. In the top right corner, it says '54/60'. Below the title is a question: 'An organization does not have the source code for a legacy application they are using. They want to understand the security posture of this application. Which code analysis technique would you recommend that they use?'. Four options are listed in a box with blue radio buttons: 'Static analysis', 'Binary analysis', 'Manual code review', and 'Threat analysis'. In the bottom right corner of the main area is a dark grey 'Submit' button.

Narration

On Screen Text

An organization does not have the source code for a legacy application they are using. They want to understand the security posture of this application. Which code analysis technique would you recommend that they use?

Static analysis

Binary analysis

Manual code review

Threat analysis

Knowledge Check

The screenshot shows a knowledge check interface. At the top left is the 'SECURITY INNOVATION' logo. To its right is a link 'Move screen reader to main content'. On the far right are three icons: a yellow book-like icon, a green question mark icon, and a red square icon. The main title 'Fundamentals of Application Security' is centered at the top in a blue bar. Below it, a question is displayed: 'Which of the following is not a benefit of static analysis tools?'. A list of four options follows, each preceded by a blue circular radio button:

- Easier integration into a secure SDLC
- Detects language specific issues
- No false positives or false negatives
- Scale easily

In the bottom right corner of the main area is a dark grey 'Submit' button.

Narration

On Screen Text

Which of the following is not a benefit of static analysis tools?

Easier integration into a secure SDLC

Detects language specific issues

No false positives or false negatives

Scale easily

Leverage Vendor Technology Security Guidance

The screenshot shows a slide from a presentation. At the top left is the 'SECURITY INNOVATION' logo. To its right is a link 'Move screen reader to main content'. The top bar is blue with the title 'Fundamentals of Application Security'. Below the title, the slide content is titled 'Leverage Vendor Technology Security Guidance'. On the right side of the slide, there is a small text '56/60'. The main content area contains the following text:

Vendors often publish guidance on the most secure ways to use their technologies. For example, Microsoft publishes freely available guidance on how to securely use the Azure cloud platform, ASP.NET, SQL server, and more.

Narration

Applications are often composed of several technologies working together to create value for organizations.

It's in the vendor's best interest to help customers successfully implement their technologies,

so vendors often publish freely available guidance on the most secure ways to use their technologies.

For example, Microsoft publishes guidance on how to securely use the Azure cloud platform,

ASP.NET, SQL Server, and more

You can save yourself significant effort in researching best practices for a given technology

by simply leveraging this work that vendors have already done for you.

On Screen Text

Leverage Vendor Technology Security Guidance

Leverage Secure Development Models, Standards and Guidelines

Move screen reader to main content

Fundamentals of Application Security

Leverage Secure Development Models, Standards and Guidelines 57/60

- Hacker techniques are continuously evolving
- Compliance does not make your application “hacker-proof”

Narration

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

However, you still have a responsibility to your organization and your users to develop secure code.

Following secure development models, standards, and guidelines can help you understand key mitigations to reduce the risk from application security vulnerabilities.

This can help alleviate the need for you to become a security expert,

while still providing an efficient way for you to ensure a good level of security for your application and its users.

When you follow secure development models and guidelines,

keep these important points in mind: The first is that attacks and hacker techniques are continuously evolving.

Naturally, models and guidelines are regularly updated to reflect new threat landscapes.

It is therefore important that you periodically review the latest secure development models and guidelines

to help ensure that your application is resilient to current risks and attacks.

Second, following a particular secure development model or guideline will not render your application “hacker-proof” or free from exploitable vulnerabilities.

Models and guidelines highlight key risks and threats to your application at the time of writing,

but they do not cover all possible threats. There will undoubtedly be other risks that your application is exposed to that are not addressed by the model or guideline.

On Screen Text

Leverage Secure Development Models, Standards and Guidelines

Knowledge Check

The screenshot shows a knowledge check interface. At the top left is the 'SECURITY INNOVATION' logo. To its right is a link 'Move screen reader to main content'. On the far right are three icons: a yellow square with a book, a green square with a question mark, and a red square with a document. The main title 'Fundamentals of Application Security' is centered at the top in a blue bar. Below it, the text 'Following secure development models, standards, and guidelines allows you to eliminate security risks from your application.' is displayed. A question box contains two options: 'True' and 'False', each preceded by a blue radio button. In the bottom right corner of the main area is a dark grey 'Submit' button.

Narration

On Screen Text

Following secure development models, standards, and guidelines allows you to eliminate security risks from your application.

True

False

Module Summary

The screenshot shows a slide from a presentation titled "Fundamentals of Application Security". At the top left is the "SECURITY INNOVATION" logo. A blue banner across the top has the title "Fundamentals of Application Security". Below the banner, the section title "Module Summary" is displayed. In the top right corner, there are icons for a document, a question mark, and a printer, with the text "59/60" indicating the slide number. A note at the top says "Move screen reader to main content". The main content area contains the text "Click each objective to learn more." followed by a list of objectives. The first objective, "Understanding Application Attacks", is highlighted with a yellow box and a small icon of a book with caution tape. A sidebar on the left lists the three objectives: 1. Understanding Application Attacks (highlighted), 2. Common Application Attacks, and 3. Key Secure Application Development Principles. The content for "Understanding Application Attacks" includes a brief description of the topic, a link to review it again, and a note at the bottom stating "Note - This slide does not contain audio. Please continue to the next section once you have finished reviewing this material."

Narration

On Screen Text

Module Summary

Understanding Application Attacks

In this topic you learned about the basic anatomy of an application attack and how input validation is used as a primary countermeasure. You learned about important input validation techniques, regular expressions, and general input validation pitfalls to avoid.

Click [here](#) to review this section again.

Common Application Attacks

In this topic you were introduced to some common application attacks, such as buffer overflows, SQL injection, and race conditions. You also learned about additional resources to get more information about these and many other attacks.

Click [here](#) to review this section again.

Key Secure Application Development Principles

In this topic you learned about some secure development principles and best practices, such as those for input validation, using standard libraries, and leveraging language and platform protection. You also learned about code analysis and code review, and the differences between the two. In addition, you learned about leveraging vendor guidance and secure development models, standards and guidelines as a means to help you in your secure development efforts.

Click [here](#) to review this section again.

Fundamentals of Application Security

Thank You

The screenshot shows a web-based course interface. At the top left is the "SECURITY INNOVATION" logo. To its right is a link "Move screen reader to main content". The top navigation bar has the title "Fundamentals of Application Security". Below the title, the text "Thank You" is displayed. To the right of this, "60/60" is shown. A message states, "This concludes the **Fundamentals of Application Security** course. Thank you." Below this message is the instruction, "Click the "Take the Exam" button to proceed to the exam." A large, prominent "Take The Exam" button is centered at the bottom of the page.

Narration

On Screen Text

Thank You

This concludes the **Fundamentals of Application Security** course. Thank you.

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