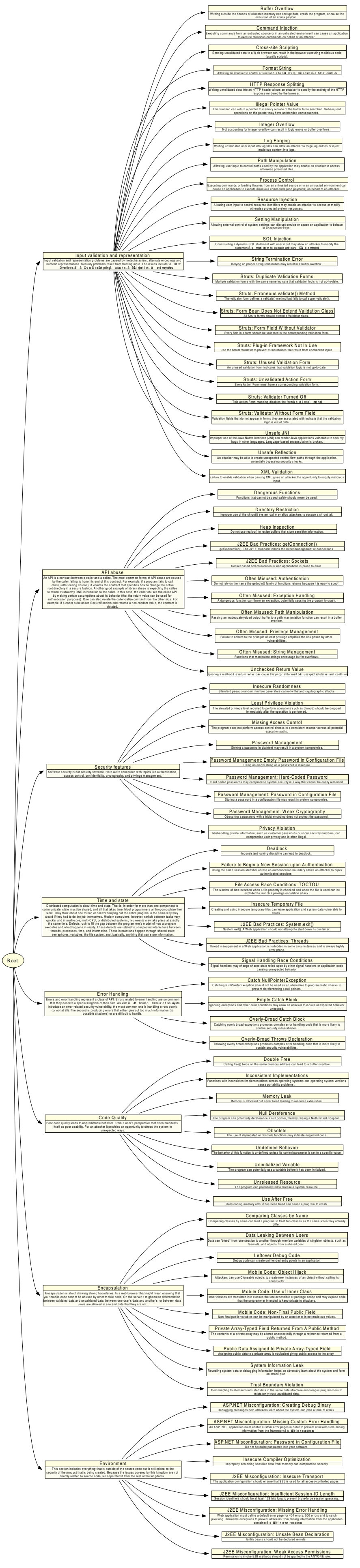
Root
Buffer Overflow
Writing outside the bounds of allocated memory can corrupt data, crash the program, or cause the execution of an attack payload.
Command Injection
Executing commands from an untrusted source or in an untrusted environment can cause an application to execute malicious commands on behalf of an attacker.
Cross-site Scripting
Sending unvalidated data to a Web browser can result in the browser executing malicious code (usually scripts).
Format String
Allowing an attacker to control a functionâ s fo rmt stri ng may result in a buffer overflow
HTTP Response Splitting
Writing unvalidated data into an HTTP header allows an attacker to specify the entirety of the HTTP response rendered by the browser.
Illegal Pointer Value
This function can return a pointer to memory outside of the buffer to be searched. Subsequent operations on the pointer may have unintended consequences.
Integer Overflow
Not accounting for integer overflow can result in logic errors or buffer overflows.
Log Forging
Writing unvalidated user input into log files can allow an attacker to forge log entries or inject malicious content into logs.
Path Manipulation
Allowing user input to control paths used by the application may enable an attacker to access otherwise protected files.
Process Control
Executing commands or loading libraries from an untrusted source or in an untrusted environment can cause an application to execute malicious commands (and payloads) on behalf of an attacker.
Resource Injection
Allowing user input to control resource identifiers may enable an attacker to access or modify otherwise protected system resources.
Setting Manipulation
Allowing external control of system settings can disrupt service or cause an application to behave in unexpected ways.
SQL Injection
Constructing a dynamic SQL statement with user input may allow an attacker to modify the statementâ s meaning or to execute arbitrary SQL commands.
Input validation and representation
Input validation and representation problems are caused by metacharacters, alternate encodings and numeric representations. Security problems result from trusting input. The issues include: â Buffer Overflows,â â Cross-Site Scriptingâ attacks, â SQL Injection, â and many others.
String Termination Error
Relying on proper string termination may result in a buffer overflow.
Struts: Duplicate Validation Forms
Multiple validation forms with the same name indicate that validation logic is not up-to-date.
Struts: Erroneous validate() Method
The validator form defines a validate() method but fails to call super.validate().
Struts: Form Bean Does Not Extend Validation Class
All Struts forms should extend a Validator class.
Struts: Form Field Without Validator
Every field in a form should be validated in the corresponding validation form.
Struts: Plug-in Framework Not In Use
Use the Struts Validator to prevent vulnerabilities that result from unchecked input.
Struts: Unused Validation Form
An unused validation form indicates that validation logic is not up-to-date.
Struts: Unvalidated Action Form
Every Action Form must have a corresponding validation form.
Struts: Validator Turned Off
This Action Form mapping disables the formâ s validate() method
Struts: Validator Without Form Field
Validation fields that do not appear in forms they are associated with indicate that the validation logic is out of date.
Unsafe JNI
Improper use of the Java Native Interface (JNI) can render Java applications vulnerable to security bugs in other languages. Language-based encapsulation is broken.
Unsafe Reflection
An attacker may be able to create unexpected control flow paths through the application, potentially bypassing security checks.
XML Validation
Failure to enable validation when parsing XML gives an attacker the opportunity to supply malicious input.
Dangerous Functions
Functions that cannot be used safely should never be used.
Directory Restriction
Improper use of the chroot() system call may allow attackers to escape a chroot jail.
Heap Inspection
Do not use realloc() to resize buffers that store sensitive information.
J2EE Bad Practices: getConnection()
getConnection(). The J2EE standard forbids the direct management of connections.
J2EE Bad Practices: Sockets
Socket-based communication in web applications is prone to error.
API abuse
An API is a contract between a caller and a callee. The most common forms of API abuse are caused by the caller failing to honor its end of this contract. For example, if a program fails to call chdir() after calling chroot(), it violates the contract that specifies how to change the active root directory in a secure fashion. Another good example of library abuse is expecting the callee to return trustworthy DNS information to the caller. In this case, the caller abuses the callee API by making certain assumptions about its behavior (that the return value can be used for authentication purposes). One can also violate the caller-callee contract from the other side. For example, if a coder subclasses SecureRandom and returns a non-random value, the contract is violated.
Often Misused: Authentication
Do not rely on the name the getlogin() family of functions returns because it is easy to spoof.
Often Misused: Exception Handling
A dangerous function can throw an exception, potentially causing the program to crash.
Often Misused: Path Manipulation
Passing an inadequatelysized output buffer to a path manipulation function can result in a buffer overflow.
Often Misused: Privilege Management
Failure to adhere to the principle of least privilege amplifies the risk posed by other vulnerabilities.
Often Misused: String Management
Functions that manipulate strings encourage buffer overflows.
Unchecked Return Value
Ignoring a methodâ s return value can cause the program to overlook unexpected states and conditions
Insecure Randomness
Standard pseudo-random number generators cannot withstand cryptographic attacks.
Least Privilege Violation
The elevated privilege level required to perform operations such as chroot() should be dropped immediately after the operation is performed.
Missing Access Control
The program does not perform access control checks in a consistent manner across all potential execution paths.
Password Management
Storing a password in plaintext may result in a system compromise.
Security features
Software security is not security software. Here we're concerned with topics like authentication, access control, confidentiality, cryptography, and privilege management.
Password Management: Empty Password in Configuration File
Using an empty string as a password is insecure.
Password Management: Hard-Coded Password
Hard coded passwords may compromise system security in a way that cannot be easily remedied.
Password Management: Password in Configuration File
Storing a password in a configuration file may result in system compromise.
Password Management: Weak Cryptography
Obscuring a password with a trivial encoding does not protect the password.
Privacy Violation
Mishandling private information, such as customer passwords or social security numbers, can compromise user privacy and is often illegal.
Deadlock
Inconsistent locking discipline can lead to deadlock.
Failure to Begin a New Session upon Authentication
Using the same session identifier across an authentication boundary allows an attacker to hijack authenticated sessions.
File Access Race Conditions: TOCTOU
The window of time between when a file property is checked and when the file is used can be exploited to launch a privilege escalation attack.
Time and state
Distributed computation is about time and state. That is, in order for more than one component to communicate, state must be shared, and all that takes time. Most programmers anthropomorphize their work. They think about one thread of control carrying out the entire program in the same way they would if they had to do the job themselves. Modern computers, however, switch between tasks very quickly, and in multi-core, multi-CPU, or distributed systems, two events may take place at exactly the same time. Defects rush to fill the gap between the programmer's model of how a program executes and what happens in reality. These defects are related to unexpected interactions between threads, processes, time, and information. These interactions happen through shared state: semaphores, variables, the file system, and, basically, anything that can store information.
Insecure Temporary File
Creating and using insecure temporary files can leave application and system data vulnerable to attack.
J2EE Bad Practices: System.exit()
System.exit(). A Web application should not attempt to shut down its container.
J2EE Bad Practices: Threads
Thread management in a Web application is forbidden in some circumstances and is always highly error prone.
Signal Handling Race Conditions
Signal handlers may change shared state relied upon by other signal handlers or application code causing unexpected behavior.
Catch NullPointerException
Catching NullPointerException should not be used as an alternative to programmatic checks to prevent dereferencing a null pointer.
Error Handling
Errors and error handling represent a class of API. Errors related to error handling are so common that they deserve a special kingdom of their own. As with â API Abuse,â there are two ways to introduce an error-related security vulnerability: the most common one is handling errors poorly (or not at all). The second is producing errors that either give out too much information (to possible attackers) or are difficult to handle.
Empty Catch Block
Ignoring exceptions and other error conditions may allow an attacker to induce unexpected behavior unnoticed.
Overly-Broad Catch Block
Catching overly broad exceptions promotes complex error handling code that is more likely to contain security vulnerabilities.
Overly-Broad Throws Declaration
Throwing overly broad exceptions promotes complex error handling code that is more likely to contain security vulnerabilities.
Double Free
Calling free() twice on the same memory address can lead to a buffer overflow.
Inconsistent Implementations
Functions with inconsistent implementations across operating systems and operating system versions cause portability problems.
Memory Leak
Memory is allocated but never freed leading to resource exhaustion.
Code Quality
Poor code quality leads to unpredictable behavior. From a user's perspective that often manifests itself as poor usability. For an attacker it provides an opportunity to stress the system in unexpected ways.
Null Dereference
The program can potentially dereference a null pointer, thereby raising a NullPointerException.
Obsolete
The use of deprecated or obsolete functions may indicate neglected code.
Undefined Behavior
The behavior of this function is undefined unless its control parameter is set to a specific value.
Uninitialized Variable
The program can potentially use a variable before it has been initialized.
Unreleased Resource
The program can potentially fail to release a system resource.
Use After Free
Referencing memory after it has been freed can cause a program to crash.
Comparing Classes by Name
Comparing classes by name can lead a program to treat two classes as the same when they actually differ.
Data Leaking Between Users
Data can "bleed" from one session to another through member variables of singleton objects, such as Servlets, and objects from a shared pool.
Leftover Debug Code
Debug code can create unintended entry points in an application.
Mobile Code: Object Hijack
Attackers can use Cloneable objects to create new instances of an object without calling its constructor.
Encapsulation
Encapsulation is about drawing strong boundaries. In a web browser that might mean ensuring that your mobile code cannot be abused by other mobile code. On the server it might mean differentiation between validated data and unvalidated data, between one user's data and another's, or between data users are allowed to see and data that they are not.
Mobile Code: Use of Inner Class
Inner classes are translated into classes that are accessible at package scope and may expose code that the programmer intended to keep private to attackers.
Mobile Code: Non-Final Public Field
Non-final public variables can be manipulated by an attacker to inject malicious values.
Private Array-Typed Field Returned From A Public Method
The contents of a private array may be altered unexpectedly through a reference returned from a public method.
Public Data Assigned to Private Array-Typed Field
Assigning public data to a private array is equivalent giving public access to the array.
System Information Leak
Revealing system data or debugging information helps an adversary learn about the system and form an attack plan.
Trust Boundary Violation
Commingling trusted and untrusted data in the same data structure encourages programmers to mistakenly trust unvalidated data.
ASP.NET Misconfiguration: Creating Debug Binary
Debugging messages help attackers learn about the system and plan a form of attack.
ASP.NET Misconfiguration: Missing Custom Error Handling
An ASP .NET application must enable custom error pages in order to prevent attackers from mining information from the frameworkâ s built-in responses.
ASP.NET Misconfiguration: Password in Configuration File
Do not hardwire passwords into your software.
Environment
This section includes everything that is outside of the source code but is still critical to the security of the product that is being created. Because the issues covered by this kingdom are not directly related to source code, we separated it from the rest of the kingdoms.
Insecure Compiler Optimization
Improperly scrubbing sensitive data from memory can compromise security
J2EE Misconfiguration: Insecure Transport
The application configuration should ensure that SSL is used for all access-controlled pages.
J2EE Misconfiguration: Insufficient Session-ID Length
Session identifiers should be at least 128 bits long to prevent brute-force session guessing.
J2EE Misconfiguration: Missing Error Handling
Web application must define a default error page for 404 errors, 500 errors and to catch java.lang.Throwable exceptions to prevent attackers from mining information from the application containerâ s built-in error response
J2EE Misconfiguration: Unsafe Bean Declaration
Entity beans should not be declared remote.
J2EE Misconfiguration: Weak Access Permissions
Permission to invoke EJB methods should not be granted to the ANYONE role.