Declarative Security

Declarative security means that the security mechanism for an application is declared and handled externally to the application. Deployment descriptors describe the J2EE application's security structure, including security roles, access control, and authentication requirements.

The Application Server supports the deployment descriptors specified by J2EE v1.4 and has additional security elements included in its own deployment descriptors. Declarative security is the application deployer's responsibility.

There are two levels of declarative security, as follows:

Application Level Security

Component Level Security

Application Level Security

The application XML deployment descriptor (application.xml) contains descriptors for all user roles for accessing the application's servlets and EJB components. On the application level, all roles used by any application container must be listed in a role-name element in this file. The role names are scoped to the EJB XML deployment descriptors (ejb-jar.xml and sun-ejb-jar.xml files) and to the servlet XML deployment descriptors (web.xml and sun-web.xml files). The sun-application.xml file must also contain matching security-role-mapping elements for each role-name used by the application.

Component Level Security

Component level security encompasses web components and EJB components.

programmatic security:

Programmatic security involves an EJB component or servlet using method calls to the security API, as specified by the Java EE security model, to make business logic decisions based on the caller or remote user's security role. Programmatic security should only be used when declarative security alone is insufficient to meet the application's security model.

The API for programmatic security consists of methods of the Java EE Security API Security Context interface, and methods of the EJB Context interface and the Servlet Http Servlet Examples of broken authentication and session management attacks

Once an attacker has gotten hold of a legitimate user's credentials, they can directly access and manipulate transactions associated with the compromised account. Attackers can then orchestrate further attacks within the system without raising suspicion by the user or administrators.

Some common session management techniques that take advantage of broken authentication and session management vulnerabilities include:

Session ID Hijacking

In such an attack mechanism, attackers steal users' valid session IDs and use them to impersonate user identities. In such instances, hackers wait till a user navigates away from their browser/device without logging out, and then continue to exploit the already established user session.

Most often, a hacker also takes advantage of session IDs that appear on the URL of the session. To do so, the attacker typically copies the session ID and uses it to log in under the same user by appending it to the HTTP request/web app's URL.

Credential Stuffing/Brute Force Attacks

In this case, hackers try to use credential data obtained from one server to gain access to an unrelated service. Credential stuffing is often known as Brute Force Attack which has been steadily gaining traction in modern cybersecurity on account of stolen credentials. Typically, attackers combine these stolen lists of known passwords with automated bots to bypass authentication checks and masquerade as legitimate users. Request interface. The Glassfish Server supports these interfaces as specified in the Java EE specification.

session Management:

Establish a session inactivity timeout that is as short as possible. It should be no more than several hours. If a session was established before login, close that session and establish a new session after a successful login. Do not allow concurrent logins with the same user ID. Use well vetted algorithms that ensure sufficiently random session identifiers. Session ID creation must always be done on the server side. Do not pass session identifiers as GET parameters. Server side session data should have appropriate access controls in place. Generate a new session ID and deactivate the old one frequently. Generate a new session token if a user's privileges or role changes. Generate a new session token if the connection security changes from HTTP to HTTPS. Only utilize the system generated session IDs for client side session (state) management. Avoid using parameters or other client data for state management. Utilize per-session random tokens or parameters within web forms or URLs associated with sensitive server-side operations, like account management, to prevent Cross Site Request Forgery attacks. Utilize per-page random tokens or parameters to supplement the main session token for critical operations. Ensure cookies transmitted over an encrypted connection have the "secure" attribute set. Set cookies with the HttpOnlyattribute, unless you specifically require client-side scripts within your application to read or set a cookie's value. The application or system should log attempts to connect with invalid or expired session tokens. Disallow persistent logins and enforce periodic session terminations, even when the session is active. Especially for applications supporting rich network connections or connecting to critical systems. Termination times should support business requirements and the user should receive sufficient notification to mitigate negative impacts.

Cryptography

Cryptographic systems are generally classified along 3 independent dimensions: Type of operations used for transforming plain text to cipher text All the encryption algorithms are based on two general principles: substitution, in which each element in the plaintext is mapped into another element, and transposition, in which elements in the plaintext are rearranged.   The number of keys used If the sender and receiver uses same key then it is said to be symmetric key (or) single key (or) conventional encryption. If the sender and receiver use different keys then it is said to be public key encryption.   The way in which the plain text is processed A block cipher processes the input and block of elements at a time, producing output block for each input block. A stream cipher processes the input elements continuously, producing output element one at a time, as it goes along

.Cryptanalysis The process of attempting to discover X or K or both is known as cryptanalysis. The strategy used by the cryptanalysis depends on the nature of the encryption scheme and the information available to the cryptanalyst. There are various types of cryptanalytic attacks based on the amount of information known to the cryptanalyst. Cipher text only – A copy of cipher text alone is known to the cryptanalyst. Known plaintext – The cryptanalyst has a copy of the cipher text and the corresponding plaintext. Chosen plaintext – The cryptanalysts gains temporary access to the encryption machine.

They cannot open it to find the key, however; they can encrypt a large number of suitably chosen plaintexts and try to use the resulting cipher texts to deduce the key. Chosen cipher text – The cryptanalyst obtains temporary access to the decryption machine, uses it to decrypt several string of symbols, and tries to use the results to deduce the key.

Input and Output Sanitization

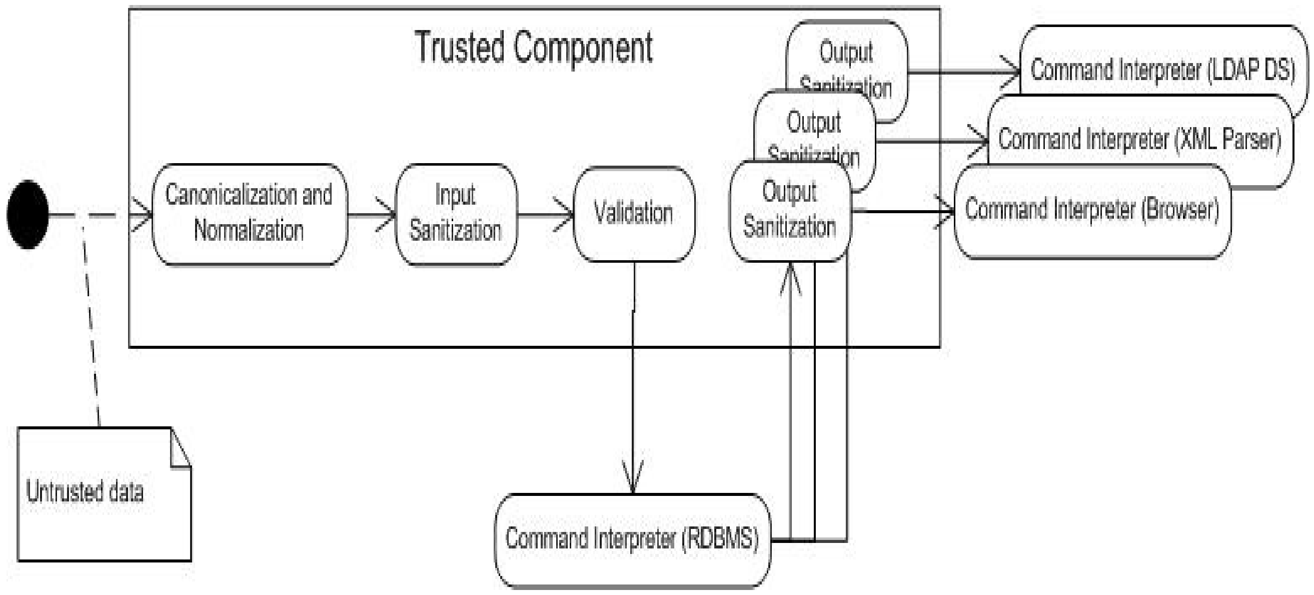
Software programs often contain multiple components that act as subsystems wherein each component operates in one or more trusted domains. For example, one component may have access to the file system but lack access to the network, while another component has access to the network but lacks access to the file system. Distrustful decomposition and privilege separation [Dougherty 2009] are examples of secure design patterns that reduce the amount of code that runs with special privileges by designing the system using mutually untrusting components.

Although software components can obey policies that allow them to transmit data across trust boundaries, they cannot specify the level of trust given to any component. The deployer of the application must define the trust boundaries with the help of a systemwide security policy. A security auditor can use that definition to determine whether the software adequately supports the security objectives of the application.

A Java program can contain both internally developed and third-party code. Java was designed to allow the execution of untrusted code; consequently, third-party code can operate in its own trusted domain. The public API of such third-party code can be considered to be a trust boundary. Data that crosses a trust boundary should be validated unless the code that produces this data provides guarantees of validity. A subscriber or client may omit validation when the data flowing into its trust boundary is appropriate for use as is. In all other cases, inbound data must be validated.

Injection Attacks

Data received by a component from a source outside the component's trust boundary can be malicious and can result in an injection attack, as shown in the scenario in Figure 1



Trusted Component
Output Sanitization
Output Sanitization
Output Sanitization
Command Interpreter (LDAP DS)
Command Interpreter (XML Parser)
Command Interpreter (Browser)
Canonicalization and Normalization
Input Sanitization
Validation
Untrusted data
Command Interpreter (RDBMS)

Programs must take steps to ensure that data received across a trust boundary is appropriate and not malicious. These steps can include the following:

Validation: Validation is the process of ensuring that input data falls within the expected domain of valid program input. This requires that inputs conform to type and numeric range requirements as well as to input invariants for the class or subsystem.

Sanitization:

In many cases, the data is passed directly to a component in a different trusted domain. Data sanitization is the process of ensuring that data conforms to the requirements of the subsystem to which it is passed. Sanitization also involves ensuring that data conforms to security-related requirements regarding leaking or exposure of sensitive data when output across a trust boundary. Sanitization may include the elimination of unwanted characters from the input by means of removing, replacing, encoding, or escaping the characters. Sanitization may occur following input (input sanitization) or before the data is passed across a trust boundary (output sanitization). Data sanitization and input validation may coexist and complement each other. Many command interpreters and parsers provide their own sanitization and validation methods. When available, their use is preferred over custom sanitization techniques because custom-developed sanitization can often neglect special cases or hidden complexities in the parser. Another problem with custom sanitization code is that it may not be adequately maintained when new capabilities are added to the command interpreter or parser software.

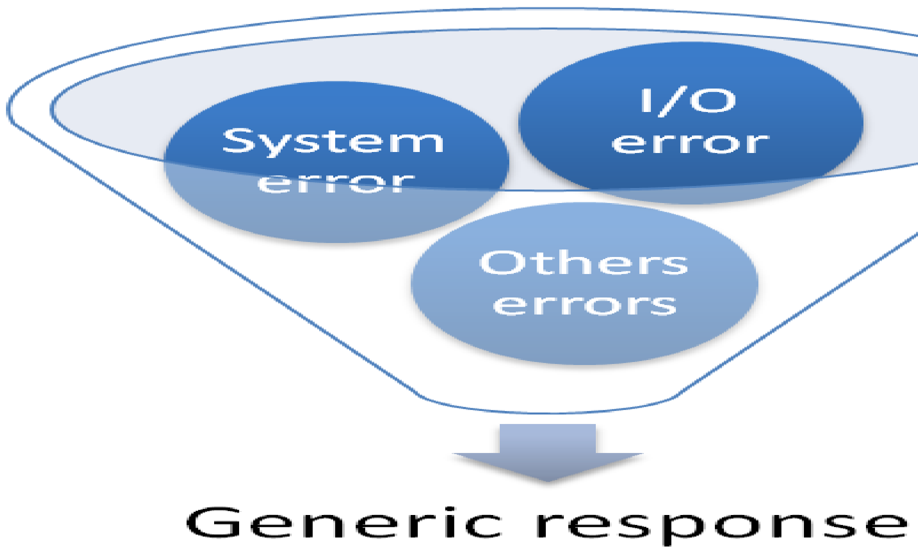
Error handling

Error handling is a part of the overall security of an application. Except in movies, an attack always begins with a Reconnaissance phase in which the attacker will try to gather as much technical information (often name and version properties) as possible about the target, such as the application server, frameworks, libraries, etc.

Unhandled errors can assist an attacker in this initial phase, which is very important for the rest of the attack.

The article shows how to configure a global error handler as part of your application's runtime configuration. In some cases, it may be more efficient to define this error handler as part of your code. The outcome being that when an unexpected error occurs then a generic response is returned by the application but the error details are logged server side for investigation, and not returned to the user.

The following schema shows the target approach:



As most recent application topologies are API based, we assume in this article that the backend exposes only a REST API and does not contain any user interface content. The application should try and exhaustively cover all possible failure modes and use 5xx errors only to indicate responses to requests that it cannot fulfill, but not provide any content as part of the response that would reveal implementation details. For that, RFC 7807 - Problem Details for HTTP APIs defines a document format.

For the error logging operation itself, the logging cheat sheet should be used. This article focuses on the error handling part.

Logging & Auditing

The difference between audit logs and regular system logs (e.g., error logs, operational logs, etc.) is the information they contain, their purpose, and their immutability. Whereas regular system logs are designed to help developers troubleshoot errors, audit logs help organizations document a historical record of activity for compliance purposes and other business policy enforcement. A log from any network device, application, host, or operating system can be classified as an audit log if it contains the information mentioned above and is used for auditing purposes. Compliance frameworks also generally require organizations to meet long-term retention policies, which is why audit logs aim to be immutable so that no user or service can alter audit trails.

Administrative activity

This includes events like creating or deleting a user account, such as deleting a user from your CRM tool (e.g., Salesforce).

Data access and modification

This includes events where a user views, creates, or modifies data, such as downloading a file from payroll software (e.g., Workday).

Most technologies in your tech stack will offer a UI where you can enable audit log collection. Depending on the specific tool, you may also have more granular control over audit log collection. For example, cloud vendors such as Amazon Web Services, Microsoft Azure, and Google Cloud automatically collect a wide range of audit logs. However, you may have to enable audit logging for certain services or certain types of activity to ensure you have enough data to prove compliance or investigate an incident.

Generally accepted security auditing and logging practices should be adhered to ensure that the policies and procedures regarding compliance with the implementation specifications of certain standards and regulations (e.g. HIPAA, PCI, ISO 27001, HITRUST) are being met. Auditing and logging is an essential part of the information security program.

The purpose of auditing and logging is to record and examine activity in information systems that affect information assets. This includes any hardware, software, or procedural controls in place to track such activity as modifying information assets including protected health information within information systems.

Based on the risk assessment and other business needs, security auditing and event logging of information systems should be supported by input from different departments across the entire organization. Among these are failed logins, multiple system authentication, and suspicious after-hours login behaviour, to name a few. Documentation should be kept as to the rationale behind the list of events chosen to adequately support an investigation of a breach or incident.

Auditing records should, at a minimum, detail the unique user ID, unique data subject ID, function performed, type of event, the date and time of an event, the possible identity of the subject of the event, and the information that may have been affected by the event. Logging activity should capture the creation, review, modification, or deletion of identified information. The success/failure of the event, the account involved, and the processes involved should be tracked as part of the audit record for privileged users.

In addition, the execution of privileged functions on information systems should be audited and logged. Information systems should be configured to prevent non-privileged users from executing privileged functions. The activities of privileged users (administrators, operators, etc.) include the success/failure of the event, time the event occurred, the account involved, the processes involved, and additional information about the event. Proper logging should be enabled in order to audit administrator activities; and reviews system administrator and operator logs on a regular basis.

Audit Trail Security Best Practices

Audit trails should be secured so they cannot be altered in any way.

Limit viewing of audit trails to those with a job-related need.

Protect audit trail files from unauthorized modifications.

Promptly back-up audit trail files to a centralized log server or media that is difficult to alter.

Copy logs for wireless networks onto a log server on the internal LAN.

Use file integrity monitoring/change detection software (such as Tripwire) on logs to ensure that existing log data cannot be changed without generating alerts (although new data being added should not cause an alert).

SESSION MANAGEMENT

A session is a succession of events and transactions that are associated with the same user for a certain time frame. Once a user has logged on to a system, they are granted a unique Session ID (Cookies, URL Parameters, Authentication Tokens, etc) that allows for communication between the user and web app for the valid session. Many developers fail to develop the right parameters for sessions, making it easier for a hacker to hijack the session

ID and gain unauthorized system access. Additionally, some developers fail to set time restrictions and rotation plans for sessions, allowing attackers to impersonate users already logged in to the system.

With companies moving more of their sensitive and valuable data to the cloud, hackers are increasingly targeting web applications for their attacks. As a result, broken authentication and session management vulnerabilities are considered as the Top 2 vulnerabilities on the OWASP list since using a valid user's credentials is the easiest way for attackers to access off-limits systems.

Such attacks are also easier and more popular with modern attackers since the vulnerabilities are often neglected by software companies.

These malicious actors rely on a number of techniques to steal credentials, guess them, or deceive users into revealing them, including:

Phishing

Credential stuffing

Password spraying

Examples of broken authentication and session management attacks

Once an attacker has gotten hold of a legitimate user's credentials, they can directly access and manipulate transactions associated with the compromised account. Attackers can then orchestrate further attacks within the system without raising suspicion by the user or administrators.

Some common session management techniques that take advantage of broken authentication and session management vulnerabilities include:

Session ID Hijacking

In such an attack mechanism, attackers steal users' valid session IDs and use them to impersonate user identities. In such instances, hackers wait till a user navigates away from their browser/device without logging out, and then continue to exploit the already established user session.

Most often, a hacker also takes advantage of session IDs that appear on the URL of the session. To do so, the attacker typically copies the session ID and uses it to log in under the same user by appending it to the HTTP request/web app's URL.

Credential Stuffing/Brute Force Attacks

In this case, hackers try to use credential data obtained from one server to gain access to an unrelated service. Credential stuffing is often known as Brute Force Attack which has been steadily gaining traction in modern cybersecurity on account of stolen credentials. Typically, attackers combine these stolen lists of known passwords with automated bots to bypass authentication checks and masquerade as legitimate users.

Exception Management

Exception handling is a programming concept that allows an application to respond to different error states (like network down, or database connection failed, etc) in various ways. Handling exceptions and errors correctly is critical to making your code reliable and secure.

Error and exception handling occurs in all areas of an application including critical business logic as well as security features and framework code.

Error handling is also important from an intrusion detection perspective. Certain attacks against your application may trigger errors which can help detect attacks in progress.

Error Handling Mistakes

Researchers at the University of Toronto have found that even small mistakes in error handling or forgetting to handle errors can lead to catastrophic failures in distributed systems.

Mistakes in error handling can lead to different kinds of security vulnerabilities.

- **Information leakage**: Leaking sensitive information in error messages can unintentionally help attackers. For example, an error that returns a stack trace or other internal error details can provide an attacker with information about your environment. Even small differences in handling different error conditions (e.g., returning “invalid user” or “invalid password” on authentication errors) can provide valuable clues to attackers. As described above, be sure to log error details for forensics and debugging purposes, but don’t expose this information, especially to an external client.
- **TLS bypass**: The Apple goto “fail bug” was a control-flow error in error handling code that lead to a complete compromise of TLS connections on apple systems.
- **DOS**: A lack of basic error handling can lead to system shutdown. This is usually a fairly easy vulnerability for attackers to exploit. Other error handling problems could lead to increased usage of CPU or disk in ways that could degrade the system.

Positive Advice

- Manage exceptions in a centralized manner to avoid duplicated try/catch blocks in the code. Ensure that all unexpected behavior is correctly handled inside the application.
- Ensure that error messages displayed to users do not leak critical data, but are still verbose enough to enable the proper user response.
- Ensure that exceptions are logged in a way that gives enough information for support, QA, forensics or incident response teams to understand the problem.
- Carefully test and verify error handling code.

OWASP also maintains a separate, similar list for application programming interfaces (APIs), which are a crucial building block for most web applications. This list is the OWASP API Security Top 10.

As of 2019*, the OWASP API Security Top 10 includes:

1. **Broken Object Level Authorization:** This refers to manipulation of object identifiers within a request to gain unauthorized access to sensitive data. Attackers access objects (data) they should not have access to, by merely changing the identifiers.
2. **Broken User Authentication:** If authentication is implemented incorrectly, attackers may be able to impersonate API users, enabling them to access confidential data.
3. **Excessive Data Exposure:** Many APIs err on the side of exposing data and count on the API user to filter the data properly. This could allow unauthorized persons to view the data.
4. **Lack of Resources & Rate Limiting:** By default, many APIs do not limit the number or size of requests they can receive at a given time. This leaves them open to denial-of-service (DoS) attacks.
5. **Broken Function Level Authorization:** This risk has to do with authorization. API users may be authorized to do too much, leading to data exposure.
6. **Mass Assignment:** The API automatically applies user inputs to multiple properties. An attacker could use this vulnerability to, for example, change themselves to an admin while updating some other innocuous property of their user profile.
7. **Security Misconfiguration:** This covers a variety of mistakes in setting up the API, including misconfigured HTTP headers, unnecessary HTTP methods, and what OWASP calls "verbose error messages containing sensitive information."
8. **Injection:** In an injection attack, the attacker sends specialized commands to the API that trick it into revealing data or executing some other unexpected action. Learn about SQL injection.
9. **Improper Assets Management:** This occurs when there is no tracking of both current, production APIs and those that have been deprecated, leading to shadow APIs. APIs are vulnerable to this risk because they tend to make so many endpoints available.
10. **Insufficient Logging & Monitoring:** OWASP notes that studies show it typically takes over 200 days to detect a breach. Detailed event logging and close monitoring may enable API developers to detect and stop breaches far earlier.

**As of December 2021, the list had not been updated since 2019.*

To read about these 10 security risks in more depth, see OWASP's official page.

There is some crossover between the OWASP Top 10 list (full list here) and the OWASP API security top 10 list. For instance, injection, broken authentication, and insufficient logging and monitoring appear in both. However, APIs present slightly different risks compared to web applications. Developers should take both lists into account.

Memory Management

Specifically close resources, do not rely on garbage collection. (for example connection objects, file handles, etc.) Properly free allocated memory upon the completion of functions and at all exit points.

Low-level languages such as C or C++ often require, or at the very least allow, developers to perform fine-grained available memory management. Even experienced professionals may get bitten by *memory*-related errors, simply because they are often hard to spot and might trigger unpredictably at run time. Hereafter, we're referring to memory management as the broad set of operations that involve handling available memory at the byte level.

This category includes several bugs resulting from inappropriate actions, among which are:

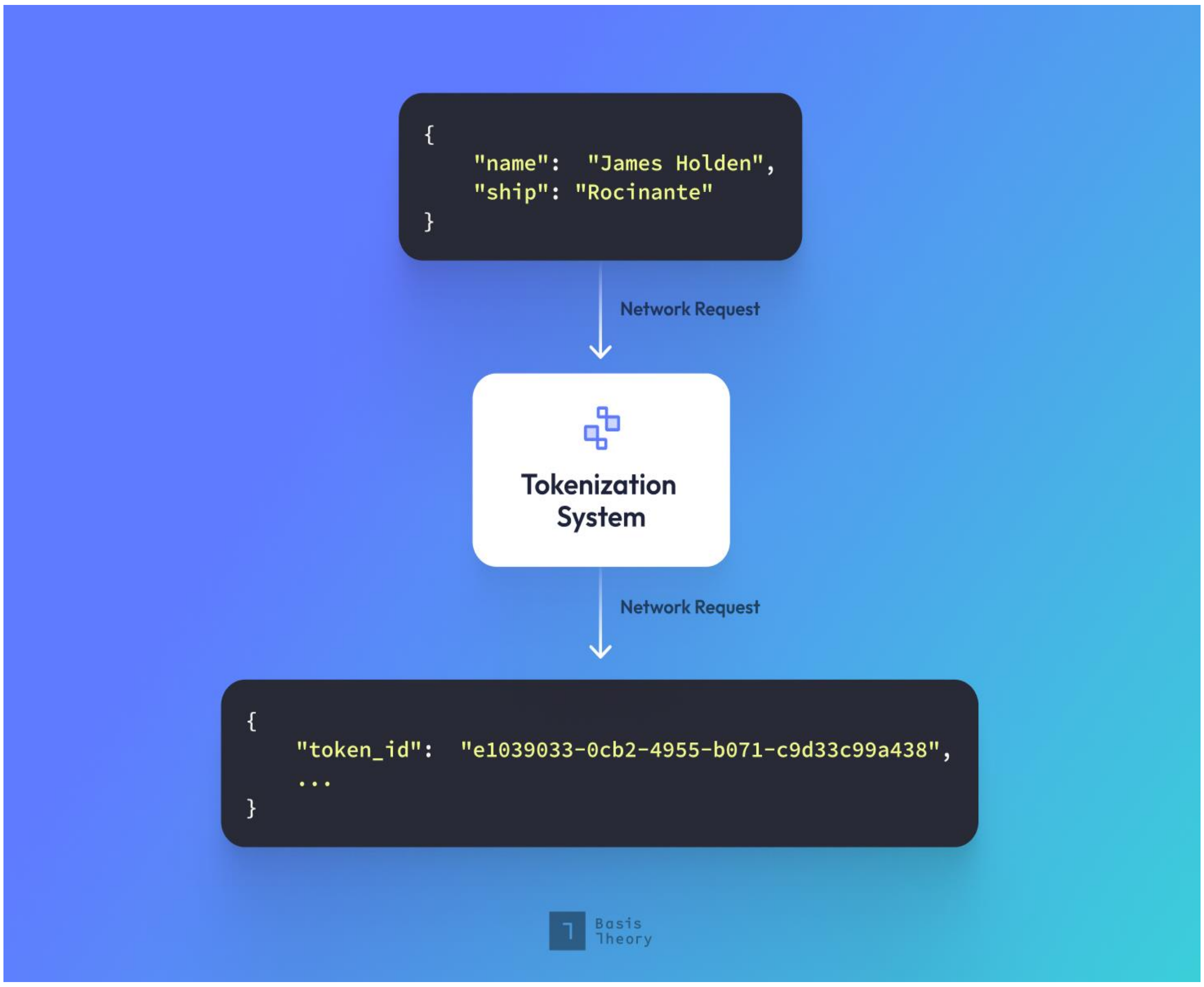
- using a previously de allocated memory region;
- forgetting to de allocate some memory region;
- accessing data outside the bounds of an allocated buffer;
- dereferencing a NULL pointer;
- etc.

The most common and well-known impact is probably the Stack Overflow where data, possibly coming from a un trusted source, is copied in a buffer that resides on the program stack. The lack of bounds determining checks may result in code being written outside the designated area, modifying other elements in the stack, including local variables and the return pointer. In particular, controlling the return pointer means controlling the program flow after the execution of the current function, thus enabling the attacker to reach unexpected code regions that ultimately may lead to the execution of arbitrary code.

- ➢ Check that the buffer is as large as specified
- ➢ When using functions that accept a number of bytes to copy, such as strncpy(), be aware that if the destination buffer size is equal to the source buffer size, it may not NULL-terminate the string
- ➢ Check buffer boundaries if calling the function in a loop and make sure there is no danger of writing past the allocated space
- ➢ Truncate all input strings to a reasonable length before passing them to the copy and concatenation functions
- ➢ Specifically close resources, do not rely on garbage collection. (for example connection objects, file handles, etc.)
- ➢ Properly free allocated memory upon the completion of functions and at all exit points.

A token is a non-exploitable identifier that references sensitive data. Tokens can take any shape, are safe to expose, and are easy to integrate. Tokenization, therefore, refers to the process of storing data and creating a token. This tokenization process is completed by a tokenization platform and in its most simple form looks like:

1. You enter sensitive data into a tokenization platform.
2. The tokenization platform securely stores the sensitive data.
3. The system provides a token for you to use in place of your sensitive data.



```
{
    "name":  "James Holden",
    "ship": "Rocinante"
}
```

```
{
    "token_id":  "e1039033-0cb2-4955-b071-c9d33c99a438",
    ...
}
```

What is a third-party tokenization platform and what does it do?

Third-party tokenization platforms can be split into two functions: a token vault and its services.

The token vault offers a secure and compliant location to store the original data (e.g., Credit card numbers). The services provide organizations the ability to collect, abstract, secure, and use the tokens or variations of the underlying data.

Combined, these two components provide organizations a way to collect, store, and use sensitive data without assuming the ongoing costs, delays, and liabilities of securing it themselves.

How are tokens used?

Instead of using the sensitive data, developers and their applications use previously generated tokens to execute traditional operations that sensitive data would provide, like performing analyses, generating documents, or customer verification.

To see how this works in practice, let’s take a look at two examples:

Example 1: Protecting PII

A company needs Personally Identifiable Information (PII) to generate and send tax documents for its employees. This company doesn’t want to go through the trouble of securing their employees’ data within their own system, so they use a tokenization platform for storing sensitive employee data. During on boarding, employees provide their PII via a form hosted on a company’s website.

Although the company hosts the website, the form uses an i frame which captures and sends PII to the tokenization platform. Tokens are generated to represent the PII and sent back to the company for the team to use instead of the raw PII data. The company then uses the tokenization platform to process and generate the tax document complete with the necessary sensitive information, all without the company worrying about compliance.

Example 2: Avoiding PCI compliance

Another company needs credit card information from its customers to process payment for its e-commerce website. Similar to the company mentioned above, they don’t want to build a compliant system if it doesn’t give them a competitive advantage. Additionally, they don’t want to be locked into a specific payment processor. They opt for a tokenization platform that can process payments with many payment processors.

When customers reach the point on the e-commerce site to enter payment information, an iframe is used to send sensitive credit card information to the tokenization platform. Tokens representing the customer PCI information are sent back to the company. As soon as the customer takes action to purchase, the company makes requests with that customer’s tokens through a proxy that calls a payment processor to charge the customer’s payment method. This was all done seamlessly, without the company needing to comply with stringent PCI compliant policies.

sandboxing

A sandbox is an isolated testing environment that enables users to run programs or open files without affecting the application, system or platform on which they run.

Software developers use sandboxes to test new programming code. Cyber security professionals use sandboxes to test potentially malicious software. Without sandboxing, software or applications could have potentially unlimited access to all the user data and system resources on a network.

Sandboxes are also used to safely execute malicious code to avoid harming the host device, the network or other connected devices. Using a sandbox to detect malware offers an additional layer of protection against security threats, such as stealthy attacks and exploits that use zero-day vulnerabilities.

Importance of sandboxes

As malware becomes more sophisticated, monitoring suspicious behavior to detect malware has become increasingly difficult. Many threats in recent years have employed advanced obfuscation techniques that can evade detection from endpoint and network security products.

Sandboxing protects an organization's critical infrastructure from suspicious code because it runs in a separate system. It also allows IT to test malicious code in an isolated testing environment to understand how it works as well as more rapidly detect similar malware attacks.

Uses of sandboxes

In general, a sandbox is used to test suspicious programs that may contain viruses or other malware, without allowing the software to harm the host devices.

Sandboxing is an important feature of the Java programming language and development environment, where the sandbox is a program area and set of rules that programmers need to use when creating Java code -- called an applet -- that is sent as part of a webpage.

A sandbox can also enable a mirrored production environment that an external developer can use to develop an app that uses a web service from the sandbox. This enables third-party developers to validate their code before migrating it to the production environment.

An API sandbox is targeted at API developers and testers. It mimics the characteristics of the production environment to create simulated responses for APIs that reflect the behavior of a real system.

Static Code Testing vs Dynamic Code Testing

Static Code Analysis (also known as Source Code Analysis) is usually performed as part of a Code Review (also known as white-box testing) and is carried out at the Implementation phase of a Security Development Lifecycle (SDL). Static Code Analysis commonly refers to the running of Static Code Analysis tools that attempt to highlight possible vulnerabilities within ‘static’ (non-running) source code by using techniques such as Taint Analysis and Data Flow Analysis.

Ideally, such tools would automatically find security flaws with a high degree of confidence that what is found is indeed a flaw. However, this is beyond the state of the art for many types of application security flaws. Thus, such tools frequently serve as aids for an analyst to help them zero in on security relevant portions of code so they can find flaws more efficiently, rather than a tool that simply finds flaws automatically.

Some tools are starting to move into the Integrated Development Environment (IDE). For the types of problems that can be detected during the software development phase itself, this is a powerful phase within the development lifecycle to employ such tools, as it provides immediate feedback to the developer on issues they might be introducing into the code during code development itself. This immediate feedback is very useful as compared to finding vulnerabilities much later in the development cycle.

There are various techniques to analyze static source code for potential vulnerabilities that maybe combined into one solution. These techniques are often derived from compiler technologies.

Static Testing Strengths

- Quick in identifying obvious coding flaws
- Can be run in parallel with development to reduce overhead at the end of the development life cycle

As security threats become more commonplace, dev teams that rely on just one type of testing leave their applications vulnerable to attack. To be successful, teams must look beyond the most common testing methods. Two alternative methods, **static application security testing (SAST)** and **dynamic application security testing (DAST)** fill those security gaps. SAST is open box testing that scans a software application from the inside out before it is compiled or executed. In contrast, DAST testing simulates the actions of a malicious actor trying to break into your application from the outside.

Static application security testing analyzes program source code to identify security vulnerabilities. These vulnerabilities include SQL injection, buffer overflows, XML external entity (XXE) attacks, and other OWASP Top 10 security risks.

The SAST methodology guides developers to begin testing their application at early development stages without executing a functional component. This approach discovers application source code security flaws early and avoids leaving security issues to later development phases. This decreases development time and enhances overall program security.

SAST testing tools

Two of the most popular SAST testing tools are:

- Klocwork, a static code analyzer for C, C++, C#, Java, JavaScript, and Python.
- Checkmarx, a tool that supports multiple programming languages.

To mitigate serious security errors and produce more secure applications, many developers now incorporate SAST testing into their continuous integration and continuous deployment (CI/CD) pipelines. There are many use cases for using SAST to create more secure applications.

When can I use SAST?

Here’s an example: SAST can continually monitor source code vulnerabilities for problematic coding patterns that violate software development security best practices. It can also automate testing your application code for a range of vulnerabilities using popular security industry standards, like OWASP Top 10 and SANS Top 25. SAST supports compliance with data protection laws, like the Health Insurance Portability and

Accountability Act (HIPAA), and the Payment Card Industry Data Security Standard (PCI DSS).

What is DAST?

Dynamic application security testing scans software applications in real-time against leading vulnerability sources, like the OWASP Top 10 or SANS/CWE 25, to find security flaws or open vulnerabilities.

The main difference between DAST and SAST lies in how each performs the security testing. SAST scans the application code at rest to discover faulty code posing a security threat, while DAST tests the running application and has no access to its source code.

DAST is a form of closed box testing, which stimulates an outside attacker's perspective. It assumes the tester does not know the application's inner functions. It can detect security vulnerabilities that SAST cannot, such as those that appear only during the program runtime.

Because DAST tests require a complete working application, reserve them for a later phase in your application development process. Testers need to interact with the application: provide inputs, check outputs, and simulate other actions typical of user interactions. These tests help ensure the application is not susceptible to web attacks such as cross-site scripting (XXS) or SQL injection.

the main characteristics and objectives of SAST and DAST testing methodologies, let's discuss which one is best suited to your application testing environment. Organizations should not choose one or the other, but instead, apply both methods to testing applications.

SAST tests the application's internal source code in early development phases to ensure developers follow the best security practices when writing code. In contrast, DAST testing begins in later development phases in a working application. It tests the application while it's running to discover its susceptibility to the most common cyber threats.

SAST testing is technology-dependent. So, your SAST tool should support your programming language and development framework to ensure complete testing coverage. On the other hand, DAST is technology-independent because it tests the application in runtime from an external user perspective.

To achieve maximum security for your software application, consider integrating SAST and DAST tooling as part of the app's CI/CD pipeline. DevSecOps should use both methodologies to integrate security into each development phase. This approach helps development teams integrate security controls into their design process without impacting productivity. **Automating SAST and DAST scans with CI/CD accelerates development time without sacrificing the final product's safety**

Vulnerability Scanning

Vulnerability scanning is the process of discovering, analyzing, and reporting on security flaws and vulnerabilities. Vulnerability scans are conducted via automated vulnerability scanning tools to identify potential risk exposures and attack vectors across an organization's networks, hardware, software, and systems. Vulnerability scanning and assessment is an essential step in the vulnerability management lifecycle.

Once vulnerabilities have been identified through scanning and assessed, an organization can pursue a remediation path, such as patching vulnerabilities, closing risky ports, fixing mis configurations, and even changing default passwords, such as on internet of things (IoT) and other devices.

The Benefits of Vulnerability Scanning

Vulnerability scanning is a vital part of your security team's overall IT risk management approach for several reasons

- Vulnerability scanning lets you take a proactive approach to close any gaps and maintain strong security for your systems, data, employees, and customers. Data breaches are often the result of un patched vulnerabilities, so identifying and eliminating these security gaps, removes that attack vector.
- Cyber security compliance and regulations demand secure systems. For instance, NIST, PCI DSS, and HIPAA all emphasize vulnerability scanning to protect sensitive data.
- Cyber criminals also have access to vulnerability scanning tools, so it is vital to carry out scans and take restorative actions before hackers can exploit any security vulnerabilities.

The Main Types of Vulnerability Scans

Some of vulnerability scanning tools are comprehensive in their coverage, able to perform multiple types of scans across heterogeneous environments that include on-prem, Unix, Linux, Windows, cloud, off-site, and onsite. Other scanning tools serve particular niches, so it's always critical to thoroughly explore your use cases before investing in a scanner.

Let's now explore some different types of vulnerability scans, which each have their place, depending on your use cases.

Credentialed Scans vs. Non-Credentialed Scans

Credentialed and non-Credentialed scans (also respectively referred to as authenticated and non-authenticated scans) are the two main categories of vulnerability scanning.

Non-credentialed scans, as the name suggests, do not require credentials and do not get trusted access to the systems they are scanning. While they provide an outsider's eye view of an environment, they tend to miss most vulnerabilities within a target environment. So, while they can provide some valuable insights to a potential attacker as well as to a security professional trying to gauge risk from the outside, non-credentialed scans give a very incomplete picture of vulnerability exposure.

On the other hand, credentialed scans require logging in with a given set of credentials. These authenticated scans are conducted with a trusted user's eye view of the environment. Credentialed scans uncover many vulnerabilities that traditional (non-credentialed) scans might overlook. Because credentialed scans require privileged credentials to gain access for scanning, organizations should look to integrate an automated privileged password management tool with the vulnerability scanning tool, to ensure this process is streamlined and secure (such as by ensuring scan credentials do not grow stale).

Here are some other ways that scans may be categorized, based on use case.

External Vulnerability Scans

These scans target the areas of your IT ecosystem that are exposed to the internet, or are otherwise not restricted to your internal users or systems. They can include websites, ports, services, networks, systems, and applications that need to be accessed by external users or customers.

Internal Vulnerability Scans

These scan and target your internal corporate network. They can identify vulnerabilities that leave you susceptible to damage once a cyber attacker or piece of malware makes it to the inside. These scans allow you to harden and protect applications and systems that are not typically exposed by external scans.

Environmental Scans

These scans are based on the environment that your technology operates in. Specialized scans are available for multiple different technology deployments, including cloud-based, IoT devices, mobile devices, websites, and more.

Intrusive Versus Non-Intrusive Scans

Non-intrusive scans simply identify a vulnerability and report on it so you can fix it. Intrusive scans attempt to exploit a vulnerability when it is found. This can highlight the likely risk and impact of a vulnerability, but may also disrupt your operational systems and processes, and cause issues for your employees and customers — so use intrusive scanning with caution.

Vulnerability Scanning Challenges

There are several challenges that arise in conducting vulnerability scanning:

A scan only represents a moment in time

Most scans are “snapshots,” not continuous. Because your systems are changing all the time, you should run scans regularly as your IT ecosystem changes

A scan may need human input or further integrations to deliver value

Although the scanning process itself is easily automated, a security expert may still need to review the results, complete remediation, and follow-up to ensure risks are mitigated. Many organizations also integrate vulnerability scanning with automated patch management and other solutions to help reduce the human administrative burden. Regardless, the scan itself is only an early step in the vulnerability management lifecycle.

A credentialed scan may require many privileged access credentials

Depending on how thorough a scan is desired. Therefore automating management and integration of these credentials with scanner should be considered to maximize both the depth of the scan, and privileged access security.

A scan only identifies known vulnerabilities

A vulnerability scanning tool is only as good as its database of known faults and signatures. New vulnerabilities emerge all the time, so your tool will need to be continually updated.

The four following capabilities should top your list of priorities when assessing the suitability of a vulnerability scanning for your enterprise:

Frequency of updates

Your vulnerability scanner database should be continually updated with the latest identified vulnerabilities

Quality and quantity of vulnerabilities

Your scanner should strike the right balance between identifying all vulnerabilities, while minimizing false positives and negatives, and providing high-quality information on flaws, threat priorities, and remediation pathways.

Actionable results

Your scanning tool should provide comprehensive reports that allow you to take practical, corrective actions.

Integrations

Your vulnerability scanner should fit seamlessly into your vulnerability management program, which should include patch management and other solutions.

Implemented correctly, a vulnerability scanning tool is instrumental to identifying and assessing modern security risk, providing your organization with the insight it needs to take corrective actions, comply with regulatory frameworks, and maintain a strong cyber security posture.

Penetration Testing

Penetration testing is the process of evaluating the security of an application and exploiting found vulnerabilities and security risks within an asset like websites, servers, databases, networks, or mobile applications to see the extent of severity they pose to the security.

In a pentest, a security engineer finds security vulnerabilities in the application, network, or system, and helps you fix them before attackers get wind of these issues and exploit them. Pentesting is a non-negotiable fundamental step for any application or business owner.

importance of Penetration Testing

The importance of information security penetration testing assets is the following:

Identification of Vulnerabilities

Penetration testing helps identify vulnerabilities in computer systems, networks, and applications that can be exploited by attackers. This allows organizations to prioritize and fix these vulnerabilities before they can be exploited.

Enhanced Security

Penetration testing helps organizations to enhance their security posture by identifying potential security gaps and improving their security controls.

Meeting Compliance Requirements

Many regulatory and industry standards require regular penetration testing to ensure that organizations meet their security requirements. For example, the Payment Card Industry Data Security Standard (PCI DSS) requires regular penetration testing of networks and applications that process credit card data.

Cost-Effective

Penetration testing helps identify potential security threats in a cost-effective manner, as it allows organizations to identify and fix security issues before they become major security incidents.

These are just a few of the reasons that make penetration testing a valuable process in the continued maintenance of asset security.

Types of Penetration Testing in Cyber Security

Penetration testing can be done for applications and even content management systems as explained below.

Cloud Penetration Testing

Cloud penetration tests analyze the cloud computing environment and platforms for vulnerabilities that could be exploited by hackers. Cloud pentesting forms an essential component of cloud security as it reveals any potential weaknesses in the currently implemented security controls.

Such a pentest can be performed manually or through automated means and is often integrated into the cloud security strategy for optimal maintenance of security. Some of the commonly found vulnerabilities include

- Insecure APIs
- Server Mis configurations
- Weak credentials
- Outdated software
- Insecure codes

Network Penetration Testing

The objective of a network penetration test is to find vulnerabilities in the network infrastructure, either on-premise or cloud environments such as Azure and AWS penetration testing. It is one of the basic tests, and a crucial one too to protect your data and the security of your application. In this test, a wide range of areas such as configurations, encryption, and outdated security patches, are tested and checked.

Below are some of the network penetration tests that are done:

- Testing routers
- Firewall bypasses
- DNS foot printing
- Evasion of IPS/IDS
- Scanning and testing open ports
- SSH attacks
- Tests on proxy servers

Web Application Penetration Testing

Web application penetration testing must be conducted by organizations and individuals with web apps periodically to keep up with the latest attacks methodologies and security flaws. With the rise in web-based applications, huge amounts of data are stored and transmitted through them, making them attractive targets for cyber attackers. Some of the common vulnerabilities include:

- Wireless encryption and network traffic
- Unprotected access points and hotspots
- Spoofing MAC address
- Weak credentials
- DDoS Attacks
- SQL/Code Injections Attacks
- Cross-Site Scripting
- Misconfigured web servers

API Penetration Testing

API penetration testing is the process that aims to find any vulnerabilities within the API for a web application by simulating the actions of a malicious user.

An application Programming Interface is a set of standards that let applications communicate with each other. It enables developers to create customized experiences within a given application.

Some of the major security issues tested for during an API pentest are:

- Broken authentication flaws in identification measures.
- Broken authorization due to exposed endpoints.
- Exposure of data.
- Misconfigurations.
- Injection flaws such as SQL, command injections, and more.

Mobile Penetration Testing

Mobile application penetration testing are done by expert penetration testers to find security vulnerabilities which can then be reported to the developers. Mobile penetration testing isn’t just applicable to Android applications, but also to iOS, Native, and Hybrid applications.

Mobile application penetration is done to gain access to sensitive data or disrupt the application’s functioning. Some of the major security issues in mobile apps include:

- Lack of transport layer protection
- Insecure Communication
- Insecure Authentication
- Weak Encryption
- Lack of Binary Protection

Smart Contract Penetration Testing

Smart contract penetration testing is a process of evaluating a smart contract for security vulnerabilities and compliance with best practices. These are essential to safeguard the money put into them.

Because all transactions on the blockchain are permanent, stolen money cannot be recovered if it is stolen. Major security issues in smart contracts include:

- DoS attacks
- Smart contract with no upgrade options
- Visibility of functions by default
- Unencrypted files on block chain

Social Engineering Testing

Unlike the above tests, where the technical aspect of the application is put under scrutiny, in social engineering, human psychology comes under the scanner. Testers leverage and exploit human nature to break into a system in social engineering pen-testing. Through manipulation, the tester will coax the individual to reveal sensitive information which will be used to penetrate the system and plan further attacks.

Some of the common methods of attack are:

- Phishing attacks
- Masquerading as colleagues, contractors, or vendors
- Tailgating
- Dumpster diving

Even though social engineering pentest is not widely done, it is necessary to get a complete picture of your application’s security standards.