International Journal of Computer Trends and Technology
ISSN: 2231–2803 / https://doi.org/10.14445/22312803/IJCTT-V73I7P106

Volume 73 Issue 7, 44-54, July 2025

Original Article

Isolated Build Environments for Supply Chain Security: Defending Against Insider Threats

Karthikeyan Thirumalaisamy

Independent Researcher, Washington, USA.

Corresponding Author : kathiru11@gmail.com

Received: 30 May 2025 Revised: 23 June 2025 Accepted: 15 July 2025 Published: 28 July 2025

Abstract - *The rise in frequency and sophistication of software supply chain attacks has highlighted insider threats as a significant vulnerability in the build process for software. Insiders likely have access and trust in the build system, whether as a developer, administrator, or compromised CI/CD build infrastructure, to place malicious software, alter dependencies, or modify outputs - typically without the owner's knowledge. This paper proposes isolated build environments using secure enclaves, such as Intel SGX and AMD SEV, as a way to improve the integrity and confidentiality of the build process by executing it in isolated hardware-protected environments. It outlines a secure build pipeline using enclaves for attesting build environments, preventing data exfiltration, and isolating unauthorized code changes. This paper proposes a design that integrates with existing DevOps tools, employs reproducible builds, supports artefact signing, and enables secure key management using enclaves. This paper conducts threat modeling, provides implementation techniques, and also provide performance evaluation to claim that enclave-based isolated build systems can substantially reduce the attack surface while blocking insider threats with low performance overhead. Enclaves offer a scalable and effective way to improve trust in the software supply chain and are well-suited for high-assurance or regulated environments.*

Keywords - *Isolated Build, Secure Enclaves, Secure CI, Confidential Compute, Supply chain security, Insider Threats.*

1. Introduction

The trustworthiness of the software supply chain has become a top concern for organisations as software becomes further integrated into critical infrastructure, financial systems, healthcare systems, and defence. The build process is the step in the workflow in which human-written code is turned into development artifacts for production, and it is particularly vulnerable to insider threats. Malicious insiders or external adversaries who designate privileged systems or accounts have the ability to modify source code: insert backdoors and manipulate build tools, which typically go unnoticed by their organization. The publicity surrounding high-visibility attacks on well-known public organizations (i.e., SolarWinds) can provide evidence of the damage that a single attacker can do to targeted organizations, and their downstream users. Access control, and equally as important, effective audit logging are necessary security features, but they may not be sufficient to protect against adversaries who are legitimate users with real credentials, and who have a good understanding of the organization's internal systems. This issue raises concerns about the potential compromise of the build process through insider and external threats. This paper evaluates the effectiveness of the isolated environment using secure enclaves - a hardware-based Trusted Execution Environment (TEE) - as a foundational security level to better secure the build process. Secure enclaves - such as Intel Software Guard Extensions (SGX) or AMD Secured Encrypted Virtualization (SEV) provide isolated execution environments, where the enclaves protect both code and data from access or tampering from privileged embedded systems software - e.g., hypervisors and operating systems.

This paper presents a new system architecture that securely integrates enclaves into a modern pipeline to provide users with a means to confidentially verify and potentially tamper-resistant software builds. Restricting critical operations (source code decryption, dependencies resolution, compiling, signing artifacts) to enclaves creates strong cryptographic guarantees that the final build outputs could not only be true to authenticity but also, due to the reliance on other verification mechanisms, were not modified. The system presents the foundation for how enclaves can plug into CI/CD without violating the current developers workflow. In our demonstration, the isolated enclave-based builds effectively and strongly protected against insider threats with reasonable performance impacts for any expected usage. Enclave-based builds will serve high-security environments and organizations desiring improved resilience in the supply chain.

2. Understanding Insider Threats

Before insiders become a threat, they are a risk, which is defined as the potential for a person to use authorized access to the organization's assets - either maliciously or unintentionally in a way that negatively affects the organization. Access includes both physical and virtual access, and assets include information, processes, systems, and facilities.

2.1. What is an Insider?

An insider is a trusted individual who has been granted access to, or possesses privileged knowledge of, an organization's internal systems, resources, or data that are not publicly available. This access may be part of their job responsibilities or derived from their role within the company. Insiders can include employees, contractors, third-party vendors, or partners who are authorized to interact with sensitive infrastructure. The types of information and systems insiders may have access to include:

- People who have a badge or other device that allows them to continuously access the company's physical property, such as a data center or corporate headquarters.
- People who have a company computer with network access.
- People who have access to a company's corporate network, cloud resources, applications, or data.
- People who have knowledge about a company's strategy and knowledge of its financials.
- People who build the company's products or services

2.2. Categories of Insider Threats

Insider risks present an even more difficult challenge in detection than external threats due to the fact that insiders have authorized access to an organization's internal systems, data, and processes.

Due to their insider status, they are trusted by default and are also familiar with an organization's operations, such as security tools, workflows, and infrastructure. This gives them the ability to blend into their organization's systems without triggering the same types of alerts that would be generated while performing unsanctioned outside activity.

Moreover, they can misuse their access in a less visible way than an external attacker, who would have to physically break into any system. The actions taken by insiders can appear legitimate on the surface, they may access sensitive source code repositories or modify build configuration - even raise the alert threshold for automatic alerts in some instances, this would be necessary given the actions taken by the insider, or simply because they were in the correct role with the proper credentials at the time. When considering these examples, it can be incredibly difficult to distinguish between behavior that is either normal or a potential risk.

Insider threats also tend to occur over an extended period of time, so organizations will discover them much more slowly through security monitoring or normal anomaly detection monitoring. Furthermore, because of normal operational complacency, insiders could put the organizations' assets at risk for many weeks, months, or even years undetected before realizing they had taken an action that is harmful (data leaks, code tampering, and intellectual property theft are all examples of actions occurring after the fact).

Most importantly, it's critical that companies understand the types of insider risks, such as intentionally doing harm or damage, simple carelessness or mistakes, and/or compromised credentials. Each insider risk should be identified and characterized so companies can have a clear understanding of how threats can emerge from the inside. By identifying and characterizing these risks, organizations may understand the vulnerabilities posed by their insider environments and prepare effectively for trusted insiders and the challenge of monitoring them.

The following section examines the various categories of insider threats:

2.2.1. Accidental Insider Threats

Accidental insider threats, also called inadvertent or unintentional insider threats, occur when people compromise security by accident, usually by a person misunderstanding something or simply being negligent, or lacking in training. These incidents have no intent to harm but can come with serious consequences related to data compromise, system compromise, or compliance obligations.

Accidental insider threats arise when individuals misunderstand or ignore their security limitations, often due to misplaced trust or access within an organization. These threats can be hard to detect because such behaviors may appear harmless or go unnoticed until damage occurs. Common examples are:

Misdirected Communication

A business partner or employee mistakenly sends confidential documents (customer data, source code, confidential financials) to the wrong person, who assumes they were authorized.

Phishing/Social Engineering

An employee gets an email that appears normal and clicks a link, and somehow downloads malware and provides credentials to an attacker.

Mishandling Data

An individual stores sensitive files on unencrypted USB drives, uploads them to unauthorized cloud services, or shares them over insecure channels like personal email.

2.2.2. Negligent Insider Threats

Negligent insiders represent a specific type of insider threat. Like accidental insiders, they do not act out of malice. However, negligent insider actions are distinct from truly accidental because negligence by standard definition means to act with knowledge, they are violating a policy or best practice. Negligent insiders usually know their actions violate company policy or security rules, but act against them anyway. The most common drivers of negligent insider actions are certainty due to experience, complacency, time pressure, habit, convenience, or a false sense of security.

Negligent actions may become evident in environments that view security policies as a hindrance to operational performance or in places where individuals are unaware of the negative impact their activities can create. Negligent actions may not appear to be dangerous by themselves, but when you combine negligent acts, they may create vulnerabilities or allow data leaks or unauthorized access.

Common types of negligent insider actions include:

Lack of Physical Security

Allowing a visitor or coworker to "tailgate" into a secure building or area without verifying credentials, and just assuming they belong there.

Bad Policy Choices for Convenience

Deactivating endpoint protection, agreeing to log in and access resources without a VPN, or removing multi-factor authentication (MFA) to troubleshoot an issue or speed access.

Accessing your Sensitive Corporate Information/Network on an Insecure Network

Authorizing access to sensitive corporate resources (dashboard, dev environment, and/or email) over an unsecured public WiFi without a secure channel such as a VPN.

2.2.3. Malicious Insider Threats

Malicious insider threats are one of the more dangerous and difficult to detect forms of internal risks that organizations can face. While accidental or negligent insiders may cause issues through oversight or carelessness, malicious insiders act with intent; they purposely abuse their authorized access to an organization, either to harm it, steal sensitive information,/or disrupt its ability to function. These individuals are often trusted employees, contractors, or third-party partners who understand internal systems and security controls, making their actions more calculated and harder to discover.

Malicious insiders may operate individually or collaborate with external threat actors. The reasons for insider threats are diverse, including personal circumstances as well as ideological motivations. Additionally, there are typically some financial, political, or competitive advantage or gain factors that also have a role in the desires of malicious insiders.

Some common motivations of malicious insiders are:

Financial Gains

Selling proprietary data, customer records, source code, or intellectual property to rivals or on black markets.

Revenge or Personal Grievance

An unhappy employee leaks sensitive documents or destroys systems, processes, and/or data after being demoted or let go, or after a longstanding dispute or conflict.

Surveillance

Stealing confidential information for another organization on behalf of a foreign government, competitor, or activist group.

Career Advancement or Recognition

Stealing trade secrets to obtain an advantage at a future job, and/or to discredit former colleagues or organizational leadership.

2.2.4. Collusion-Based Insider Threats

Collusion-based insider threats occur when a trusted person inside an organization engages external threat actors such as cybercriminal syndicates, nation-state operatives, or competitors to perpetrate harm. Collusion-based insider threats are more sophisticated and dangerous insider risks that combine internal trust and external capability to execute sophisticated attacks. Rather than malicious insiders who operate autonomously, colluding insiders are generally recruited, forced or motivated by external actors to misuse access for some operational goal such as corporate espionage, intellectual property theft, disruption of operations, or establishing long-term persistence in the organization.

Some common motivations of insider collusion are:

Monetary Gain

Receiving payment to provide credentials, system access or confidential information

Coercion or Blackmail

Being compelled to comply under threat to a person or exposure of compromising information

Ideological

Partnering among like-minded individuals with shared goals in pursuing damage to the organization (e.g., hacktivists, nation-states)

Career or Personal Advancement

Working with competitors to gain favor or employment by sharing trade secrets.

3. How Insider Threats Occur at the Build Stage

The build stage in a software supply chain is where the source code, dependencies, and configurations get combined and built into deployable artifacts such as containers,

executables, or libraries. The build occurs within an automated CI/CD pipeline, usually leveraging tools like Jenkins, GitHub Actions, GitLab CI, or Azure DevOps. This stage of software supply chains is uniquely susceptible to insider threats, as it has an extremely high level of automation and trust implicit within the environment. Insider actors can be accidental, negligent, malicious, colluding, or compromised individuals who can exploit their access or errors made during the build phase to introduce vulnerabilities, manipulate build outputs, or weaken security controls.

The diagram below shows the details of how insider threats occur at the build stage.

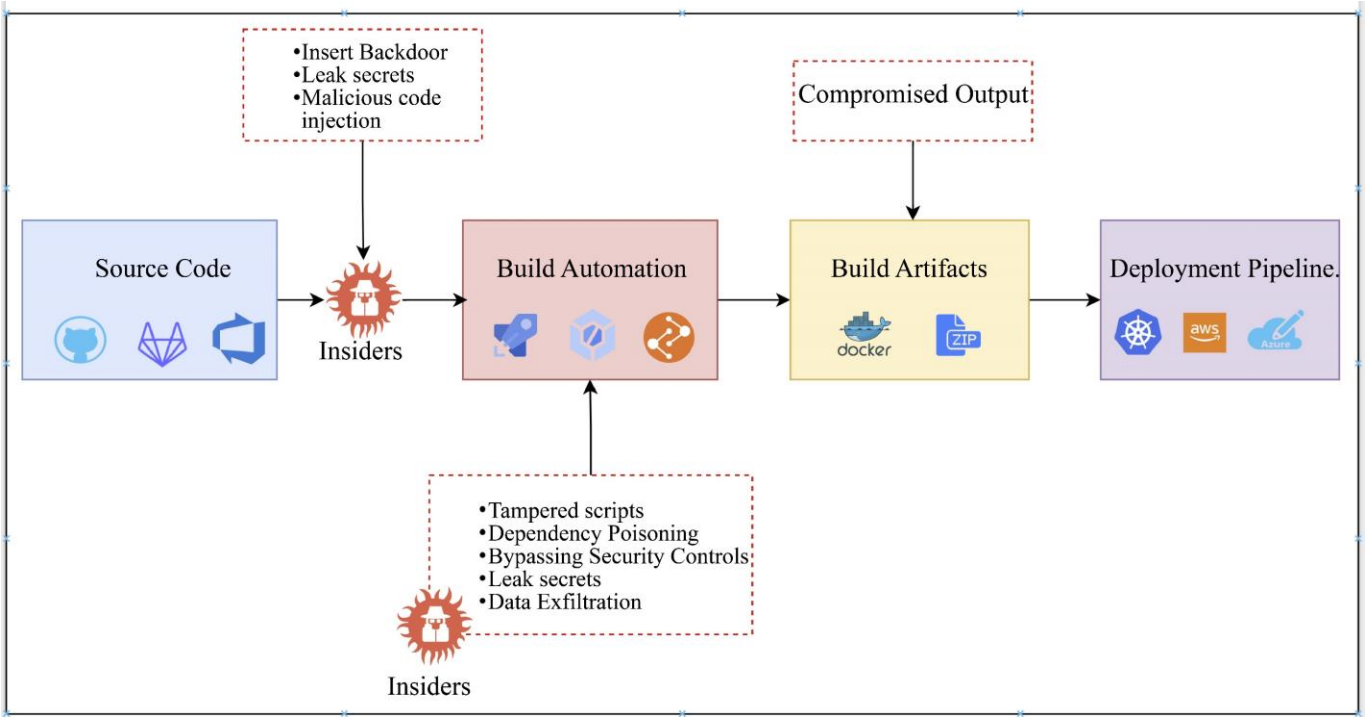

Fig. 1 Illustration of Insider threats at build stage

3.1. Insider Threats at the Source Code Stage

The source code stage is the building block for the software development life cycle, where a programmer, DevOps engineer, and software architect can all work together while writing, reviewing, and maintaining code. The source code phase is primarily via version control products such as GitHub, GitLab, and Azure Repos, which allow for code commits, branching options, pull requests, merge reviews, and audit trails. However, this trust-based ecosystem can be exploited by insiders whose access is legitimate and who, to some degree, have an understanding of the organization and the process they support. Insider threats can frequently inject malicious logic or vulnerabilities into an application's build from this phase onward in a covert and nefarious manner.

One of the most significant transitions in this pipeline is the code-completed to CI trigger pivot. Lots of the time, the transition from code to job is automatic, meaning that once code is merged into a tracked branch, it automatically triggers a CI/CD job via git hooks and runs your job builds, tests, packages, and possibly deploys your software without additional human verification. Job automation provides opportunities for blind spots if unvetted, malicious, vulnerable, or sub-standard code is merged into a trusted branch where more trusted, unassorted artifacts can be deployed without a human having to trust another instance/human to validate the artifacts.

3.1.1. Complex Code Obfuscation

Insiders can introduce malicious logic in a hidden way that is physically challenging to detect in code reviews - i.e. payloads hidden in programming language string comparisons, nested functions, or dynamic dependencies.

3.1.2. Credentials & Secrets

Leaks or exposure of sensitive information or secrets can be metered in a commit, or configuration files, such as: .env, settings.py, or .yaml files for which any of these when they are committed even temporarily, it transparently scrape sensitive information when enacted from either internally or externally (e.g., if these activities are migrated to a public repo at some later point in time).

3.1.3. Dependency Disorder

Insiders can warp dependency files like: package.json, requirements.txt, or pom.xml, to include any number of

malicious packages either via dependency confusion, typographical-decoy tampering, or utilizing already compromised and public dependency registries.

Once hostile code is merged into the CI/CD system, there is no real friction keeping hostile code from ever becoming a production artifact. Once a persistent backdoor, endpoint exploit, or leak is present in the code of a hosted application, there is a significant risk that the vulnerability may propagate throughout the build and deployment pipeline, potentially resulting in widespread compromise.

To address these vulnerabilities, governance, trust, and transparency are promoted through tools that enable the following:

- Code reviews with two or three people's approval
- Protected branches and merge gates that restrict deployments to builds from designated branches (e.g., the main branch) only
- Static code analysis and linting
- Scanning, Audit logging, and commit attribution

3.2. Insider Threats at the Build Automation Stage

The build automation stage is when the source code is converted into executable artifacts using automated platforms such as Azure Pipelines, GitHub Actions, GitLab CI, and other CI/CD orchestration platforms. These automated platforms compile and validate code by running tests, while packaging and publishing outputs often without human intervention. This stage is a high-value target for insider threats, because CI/CD systems generally run with elevated credentials and have access to:

- Secrets and credentials (e.g., API keys, signing certificates)
- Production-like build environments
- Sensitive scripts and deployment hooks
- Infrastructure provisioning logic

A malicious, negligent, or compromised insider can interact with CI/CD systems to introduce silent and trusted compromises that persist for the entire lifecycle of the software.

3.2.1. Tampered Scripts

Tampered scripts are a major insider threat during the automation build process. A trusted insider may intentionally or otherwise modify CI/CD configuration files or build scripts, changing the behavior of the pipeline. These scripts are contained in various files (kernel to the build process) representing common CI/CD examples: azure-pipelines.yml, .gitlab-ci.yml, GitHub Actions workflows, Jenkins files, and/or shell scripts. The build scripts themselves determine how code is built, tested, scanned, and deployed as a product artifact. Having an insider change the build scripts means that an insider can overwrite critical security checks, skip automated tests, and/or insert malicious command scripts that execute as part of the build process. For example, an insider may comment out or remove unit tests and static analysis steps so that code with vulnerabilities leaks past these checks. Another example is to insert a command that downloads an obfuscated payload from an external server and executes it.

Malicious insiders may also manipulate environment variables and output variables to leak secrets (e.g., API keys or credentials) or redirect the final build artifacts to their own repository or location on the file system that they control. The build system is usually set up under a level of high trust with little or no manual review; therefore, the changes typically will not be noticed until it is too late. The result? The organization finds itself creating compromised artifacts and moves through the software development lifecycle. Tampering at this point is especially dangerous because it is not just manipulating the contents of the product; it is manipulating the process of how the software gets built. This enables a deep compromise, often with little visibility, into the end product delivered to the customer.

3.2.2. Dependency Poisoning

Dependency poisoning is an insider threat that takes advantage of trusted individuals who can manipulate a build process by introducing or replacing previously trusted software dependencies with malicious, compromised or unauthorized packages. Today, software projects typically incorporate heavy reliance on third-party libraries from either public or internal repositories (examples include: npm, PyPI, Maven, NuGet); during a build process, any listed dependencies are automatically pulled in based on the requirements defined in a dependency file (e.g. package.json, requirements.txt, pom.xml, etc.). The single insider (malicious, negligent, or forced) can modify the dependency files or alter the registry configuration for the software development project to introduce a poison dependency that contains unforeseen and inadvertent back doors, surveillance code or logic bombs. The means from which an insider may use to add or replace malicious dependencies can include typosquatting (adding a dependency with almost an identical name, or similar spelling to a legitimate dependency and then subsequently replacing with malware), dependency confusion (exploiting an unexpected name collision of internal packages and public packages), or modifying a hash value in a lock file for that dependency that substitutes for the trojanize version of a dependency. After a malicious package has been added to the dependency install, it is part of the build and will exist as part of the compiled product artifact. Attackers can leverage this approach for remote access or the ability to exfiltrate data from a production environment.

3.2.3. Bypassing Security Controls

The insider threat of bypassing security controls involves a trusted user disabling or manipulating security mechanisms that protect the CI/CD pipeline through deliberate actions. The

modern build systems implement various automated tools to enforce code quality and security through Static analysis, container scanning and policy-as-code checks. The insider who controls pipeline configurations or build scripts possesses the ability to modify these controls for vulnerability detection evasion. The insider can achieve this by disabling test or scan stages in YAML pipeline files, by setting failure conditions to always succeed or by filtering output logs to conceal alerts. Insiders sometimes disable dependency check results and set build tools to bypass signing and verification operations. Such actions present a significant threat because they produce misleading security indicators which make builds appear clean and compliant even though they contain vulnerabilities. Insiders who bypass security controls enable vulnerable non-compliant or malicious code to enter production deployment. The post-factum detection of this threat becomes challenging because the artifacts appear valid without alerts, yet essential security gates remain disabled. The tampering of security controls in regulated industries or zero-trust environments leads to compliance violations and exposes customers, and creates supply chain vulnerabilities.

3.2.4. Leakage of secrets

The exposure of sensitive credentials represents a major insider threat because it happens when API keys, passwords and tokens, signing certificates and cloud access credentials become visible through accidental or intentional disclosure within the build environment. The build automation process requires these secrets to enter pipelines through environment variables and secure files, and secret management systems, including Azure Key Vault, HashiCorp Vault and GitHub Secrets. Insiders who are negligent or malicious can reveal these secrets through modifications made to build scripts, which result in secret exposure through console output logging, compiled artifact inclusion or insecure intermediate file storage. Developers who mean well sometimes make mistakes by adding .env files or configuration files containing secrets to the source repository because scanning and validation controls are not present. Secrets can be revealed during testing and debugging operations without awareness that CI/CD logs will be stored for extended periods and accessible to other team members. The discovery of accidental secret leaks proves challenging in real-time operations until the exposed secrets lead to more serious security breaches, such as unauthorized access to production systems or cloud resources. Build pipelines face increased risk because their automated nature and trusted environment allow exposed secrets to persist across multiple builds and environments. The risk of insider exposure requires strict access controls together with automated secret scanning and log masking, and least privilege access in CI/CD environments to minimize accidental secret exposure.

3.2.5. Data Exfiltration

The unauthorized transfer of sensitive information, such as source code and configuration files, credentials, build logs, and compiled artifacts, occurs during the build automation stage through data exfiltration by insiders to external systems or unauthorized internal destinations. The threat poses a significant danger to CI/CD environments because these systems manage large volumes of proprietary code and production-grade secrets while maintaining elevated access to internal repositories, build artifacts, and cloud services. A malicious insider can embed custom pipeline steps that secretly extract data, which gets transmitted to servers controlled by attackers through cloud storage buckets or messaging platforms. The insider adds scripts to transfer compiled binaries to external FTP servers, sends environment variables to webhooks, and uploads log files to third-party storage services. Insiders sometimes use legitimate integrations such as Slack, email, and artifact mirrors to hide their data exfiltration activities within normal pipeline operations. Such activities remain undetected for extended periods because CI/CD systems are trusted, and log audits are not performed in detail, especially when malicious code exists within operational builds. Successful data exfiltration leads to severe consequences that include intellectual property theft and credential exposure, and enables additional attacks throughout the software supply chain. The prevention of this threat demands implementing least-privilege access alongside build script immutability, outbound traffic restrictions and continuous pipeline behavior monitoring for anomaly detection.

4. Investigate Confidential Computing and Secure Enclave Technologies

The traditional CI/CD systems operate with extensive privileges on common infrastructure, which enables both trusted users and compromised accounts to modify source code and introduce malicious dependencies and leak sensitive data during the build process. The Secure Enclave concept presents an Isolated Build Environment solution that uses hardware-enforced execution environments to deliver confidentiality and integrity, and verifiability for build pipelines. The build process executes inside a cryptographically isolated zone through confidential computing technologies, including Intel SGX and AMD SEV and cloud-native solutions like Azure Confidential VMs and AWS Nitro Enclaves, which remain invisible to the host OS, hypervisor and other users.

The isolation model implements zero trust at runtime to defend against insider threats even when operating within trusted infrastructure. The enclave protects code and secrets and builds logic from insider access to view, modify, or intercept them. The build process can be verified through remote attestation, which allows external systems to confirm that the hardware was not tampered with and the code was approved before accepting the resulting artifacts. The combination of hardware-based isolation with minimal trusted compute base and verifiable integrity transforms the build

system into a hardened supply chain component. The system minimizes insider threats while enforcing strict role separation and providing trusted build output regardless of who controls the infrastructure.

4.1. Intel SGX Enclaves

Intel SGX operates as a hardware-based security technology from Intel that produces processor-based isolated execution environments called enclaves. Enclaves protect sensitive code and data through full encryption of RAM memory, which remains inaccessible to operating systems, hypervisors and firmware and elevated administrative users.

The architectural isolation of SGX functions as a potent security mechanism that protects against insider threats during software supply chain operations by defending source code and preventing backdoor injections and secret exposures during build operations.

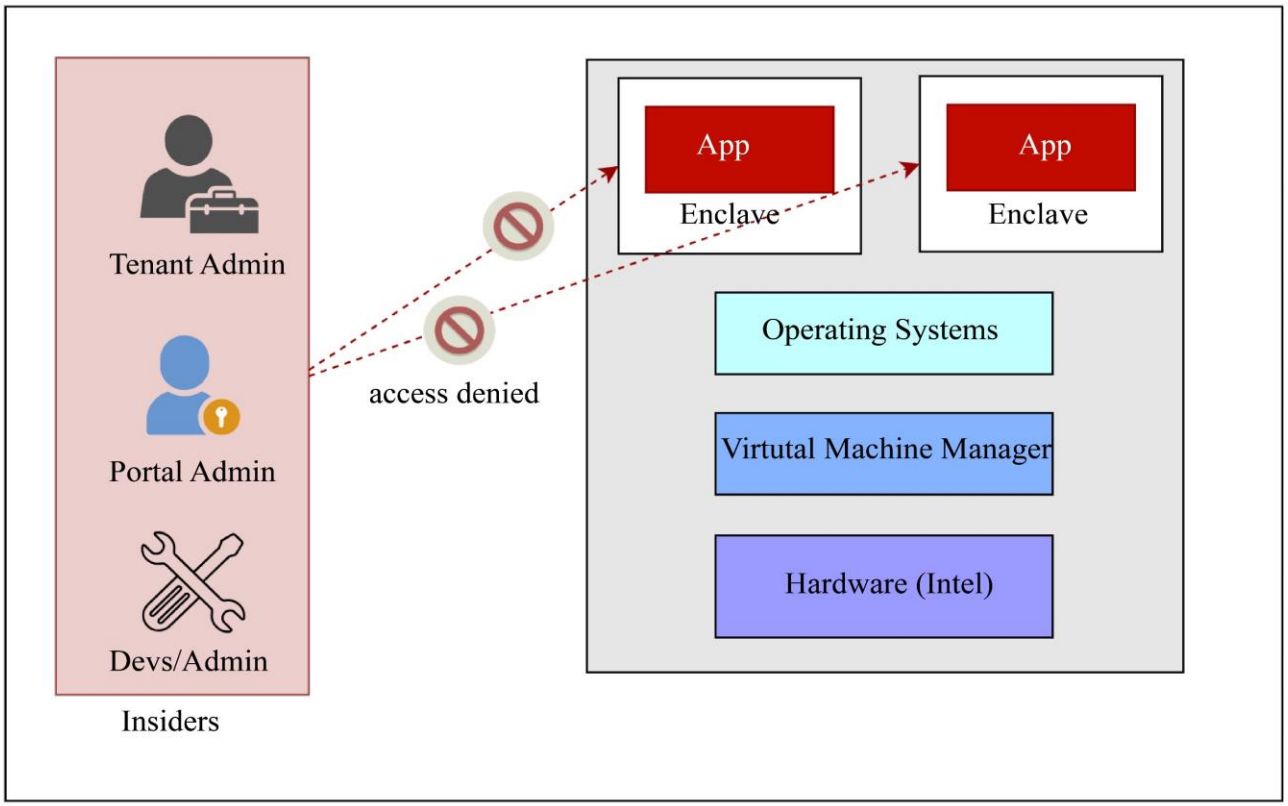

Fig. 2 Illustration of Insider threats in Intel SGX Enclaves Enclaves

Secure build environments leverage SGX to create trusted hardware-isolated enclaves that protect all vital build pipeline operations, including source code compilation, dependency verification, credential injection, and artifact signing.

Once the enclave initializes, it becomes impossible for any user, including those with complete system access, to modify or observe its contents.

The enclave contains all secrets and signing keys, which stay inside without appearing in logs, memory, or disk storage. SGX technology provides remote attestation capabilities, which enable downstream systems to authenticate that the build occurred in a genuine SGX enclave through trusted code verification. The build stage becomes tamperproof through SGX technology because it creates a trustworthy link from source code to artifact, which prevents insider interference.

SGX delivers robust security features, yet its deployment faces significant resource-related challenges. SGX enclaves operate with restricted memory capabilities because they have only 128 MB of EPC (Encrypted Page Cache) available for use by default. Using paging as a memory expansion technique leads to substantial performance losses that prevent the practical execution of extensive processing operations. SGX enclaves need to perform disk operations through untrusted components located outside their secure environment because they lack direct disk access, which adds complexity to development while restricting its usability. Large source code bases and artifacts measuring between 2 and 5 GB in size become difficult to handle inside the enclave because the available memory space remains limited, which prevents direct repository retrieval into the enclave unless developers implement specific chunking strategies, buffering mechanisms and validation protocols.

4.2. AWS Nitro Enclaves

AWS Nitro Enclaves established a secure hardware-based execution environment inside EC2 instances, which protects sensitive data through tamperproof operation even when privileged users attempt to access it. The AWS Nitro Hypervisor enables Nitro Enclaves to allocate specific CPU and memory resources from the parent instance for creating isolated enclaves without storage or network connectivity and interactive login capabilities. The isolation mechanism prevents all users with root or admin privileges, including tenant admins, portal admins and CI/CD pipeline operators, from accessing or modifying the enclave contents. After enclave launch, the software becomes immutable, so attackers cannot add spyware, modify code or disrupt enclave operations.

AWS Nitro Enclaves serve to protect the most sensitive parts of CI/CD pipeline operations through artifact signing and cryptographic hashing and build attestation. The lack of network and disk access prevents full-scale compilers and package managers from running inside these environments, but they excel at processing small security-critical build components. The enclave accepts compiled binaries and source hashes through a virtual socket (vsock) before processing or signing them with enclave secrets that remain inside the enclave, and then returns the results to the parent instance for distribution. The architectural design protects all secrets from interception and internal logic inspection and builds outcome tampering for all users, including system administrators with complete privileges.

4.3. AMD Secure Encrypted Virtualization (SEV-SNP)

The AMD SEV (Secure Encrypted Virtualization) feature operates as a virtual machine-based security mechanism that protects VM memory data by encrypting it against unauthorized access, including hypervisor and host operating system and cloud provider administrator access. The AMD Secure Processor securely manages encryption keys, which it distributes to each virtual machine to establish their unique cryptographic access. The hardware performs encryption and decryption of VM memory data automatically through its CPU during memory access without interfering with applications or the guest operating system. The memory contents of a virtual machine remain protected from unauthorized access through VM memory encryption because the underlying infrastructure cannot access them, even with root access, hypervisor bugs or insider activities.

The security features of SEV align perfectly with software supply chain practices since they protect important operations like source code building and artifact signing, which require trusted execution environments. The cloud-based CI/CD pipelines benefit from AMD SEV because this technology allows developers to perform safe builds inside VMs that protect against infrastructure threats. Source code compilation, along with dependency validation and secret handling, can be performed in build VMs on cloud provider infrastructure without fear of unauthorized data access because the data remains protected inside the VM. The security of SEV reaches new heights through its variants SEV-ES and SEV-SNP, which provide encrypted CPU register state protection along with memory mapping attack defense.

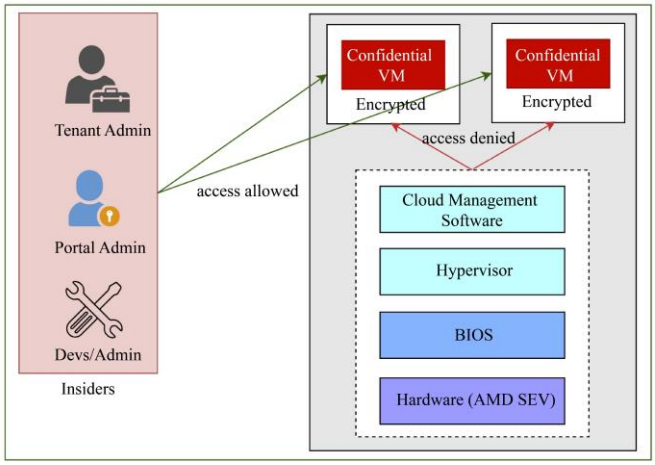

Fig. 3 Illustration of Insider threats in AMD SEV-SNP

The VM-level protection provided by SEV does not extend to application-level isolation that Intel SGX and AWS Nitro Enclaves offer. Other security measures, such as secure coding practices and restricted access policies, together with runtime controls, are needed to defend against insider threats that arise from compromised users or malicious code possessing OS-level privileges within the VM. The encryption of disk data through SEV requires additional configuration and network communications must be secured independently. The power of AMD SEV serves to protect confidential computing in shared cloud infrastructure by providing workload-level protection against infrastructure threats without requiring application code modifications.

4.4. Confidential Containers on Azure Container Instances (ACI)

ACI uses AMD SEV-SNP to create Confidential Containers which provide totally isolated build environments to stop insider threats during all phases of the software supply chain. The main principle of this solution depends on Confidential Virtual Machines (CVMs), which use AMD SEV-SNP for hardware-based memory protection and verification. The confidential workload contents that include source code credentials and runtime data remain completely protected against unauthorized access and modifications by Azure cloud administrators and compromised host systems. The Child Utility Virtual Machine (Child UVM) functions as a lightweight virtual machine that creates a secure boundary for container execution. Azure launches confidential containers within a Child UVM, which runs with AMD SEV-SNP to create complete isolation from the host OS and other tenants. The Child UVM is:

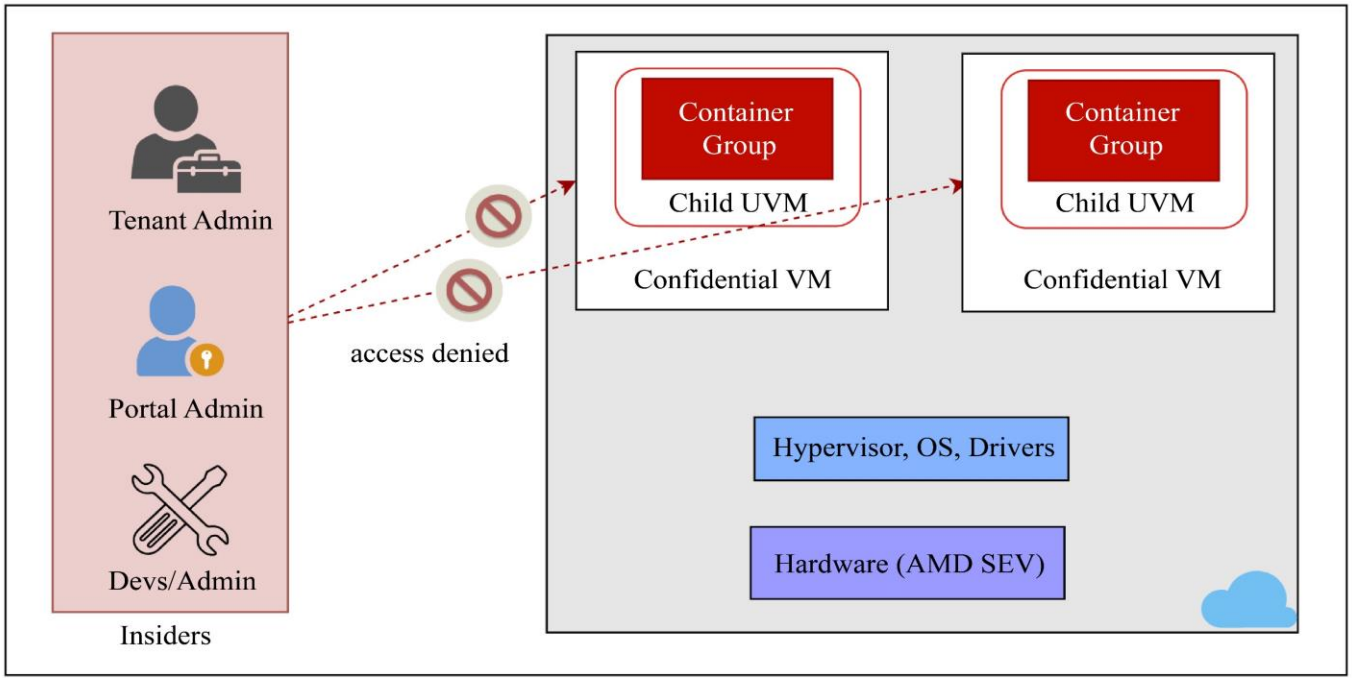

Fig. 4 Illustration of Insider threats in Confidential Containers in ACI

4.1.1. Dedicated and Ephemeral System

It starts only for container workloads and is destroyed when the container stops.

4.1.2. Measured and Attested

Child UVM receives verification from container owners to prove its genuine AMD SEV-SNP firmware alongside its correct container image.

4.1.3. Immutable at Runtime

After deployment, the runtime environment remains completely immutable because Azure staff and tenant administrators cannot access the system hardware-based protection for code execution, even when attackers control the hypervisor and management plane. The Child UVM interacts exclusively with the container runtime system while being completely inaccessible to host-level logging tools and debuggers.

The container inside the Child UVM obtains large source repositories by using pre-provisioned time-limited credentials (such as short-lived GitHub tokens or signed URLs). The container can then:

- Build software using standard tool chains.
- Run security and quality tests.
- Package and sign artifacts.
- Push outputs to external registries or storage.

Child UVM provides complete confidentiality for all source code, intermediate artifacts, secrets and signing keys during both high-throughput and long-running build operations. After completing its task, the system automatically deletes the container together with its Child UVM to eliminate any remaining data traces as part of ephemeral security. Confidential Containers deployed on ACI use AMD SEV-SNP together with Child UVMs to create a protected temporary build environment with complete isolation. The system allows secure source code downloads and protected builds while enabling safe artifact publishing to create a complete insider threat defense for secure software supply chains.

5. Proposed Approach: Ephemeral, Isolated Secure Build Environments

Organizations can establish completely secure build processes that prevent insider threats by using Azure Container Instances (ACI) with Confidential Containers supported by AMD SEV-SNP to create ephemeral, isolated build environments. The entire software build lifecycle, from source code retrieval through compilation, testing, signing and artifact distribution, runs inside an encrypted runtime which protects the environment from both cloud administrators and privileged tenant users. A confidential container starts running when the CI/CD pipeline initiates the build process with 50 GB of ephemeral disk space for pulling source code and running compilations and tests. The container starts by conducting remote attestation to check both its hardware integrity and its container image state. The system releases sensitive assets, including Git access tokens and cryptographic signing keys, after attestation verification, which uses Azure Key Vault or Managed HSM to enforce access control with attestation awareness. Refer to the diagram below for the proposed build pipeline approach.

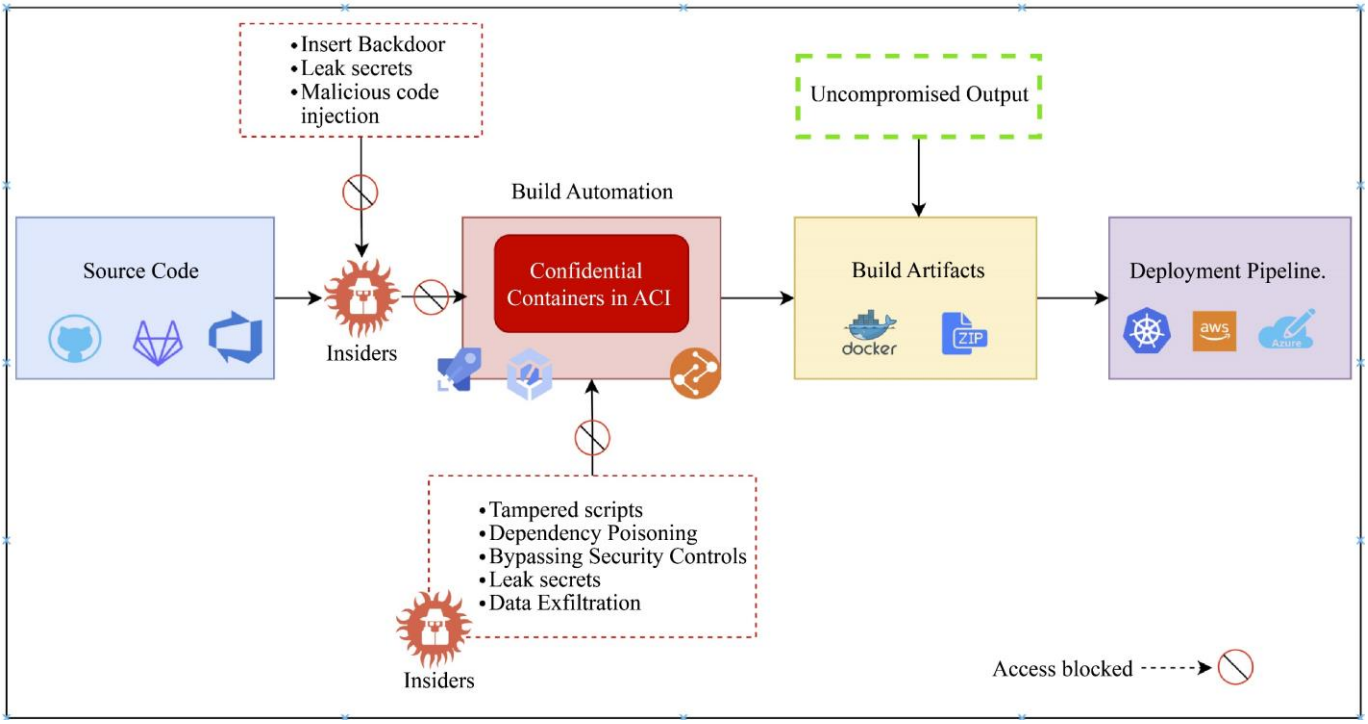

Fig. 5 Illustration of Proposed Build Pipeline using Confidential Containers in ACI

The enclave (Confidential Containers) performs secure source code retrieval using attested credentials before executing an immutable build and test process inside a network-restricted environment. The signing keys perform cryptographic signatures on build artifacts while all secrets stay encrypted in memory, where external processes cannot access them. After the build process finishes, the final artifacts get transferred to trusted storage locations such as Storage or Container Registry before the container shuts down. The system removes all source code, secrets and logs before the hardware automatically clears the memory.

The architecture implements AMD SEV-SNP confidential computing technology, which encrypts memory and CPU state at the hardware level, thus preventing any physical, root or administrative access to inspect or tamper with or exfiltrate runtime data from the container. No user, including tenant admins, CI/CD operators, cloud administrators and even infrastructure root accounts, can log in or attach a debugger or modify the container behavior when execution takes place in the sealed non-interactive container runtime.

All forms of insider threats are mitigated by design. The container image stays immutable after deployment, which prevents malicious developers from embedding backdoors or changing build scripts. The container prevents any form of external access or custom script execution by its design, while ensuring that negligent insiders cannot leak secrets through logs, because the system requires remote attestation for all actions. The encrypted and isolated memory and CPU registers prevent cloud platform engineers, along with DevOps staff who have elevated access to VMs, from gaining any insight into the contents or behavior of the container. After a successful attestation process, secrets along with source code and artifacts enter the container but are destroyed when the container exits. The confidential container provides protection against both host and CI/CD pipeline compromises by preventing any build process manipulation or observation by insiders.

All build-related secrets, such as signing keys and source tokens, remain in memory during processing while Azure Key Vault services perform secure attestation-based gating to prevent insiders from accessing secrets even when the parent pipeline is compromised. The container's ephemeral nature deletes all memory after completing the build process so that no build-related information, including source code intermediates or secrets, remains. This design provides complete protection against insider threats through interactive access restrictions and locked-down networking, together with sealed execution environments and attestation-backed trust mechanisms.

6. Conclusion

The growing complexity of software supply chains creates a major security problem because insiders represent a difficult challenge to protect against, whether their actions stem from malicious intent, negligence or external compromise. Security models that establish trust based on internal roles and infrastructure introduce a significant

vulnerability when these trusted roles are compromised. This paper introduces a practical solution that uses Isolated Build Environments within Secure Enclaves to protect public cloud build processes through hardware-enforced runtime isolation and cryptographic attestation.

The paper proposes a solution that uses Confidential Containers on Azure Container Instances (ACI) to create sealed ephemeral environments that protect runtime operations from human interaction regardless of privilege level. The containers function without shell privileges while preventing unauthorized network access and accepting sensitive data like source code, secrets and signing keys only after completing remote attestation. The build process ends with the complete destruction of the container and all runtime memory, which removes any potential attack surfaces that could persist.

The proposed build pipeline architecture eliminates trust-based assumptions through its design, which targets all fundamental insider threat categories. All personnel with developer or administrator privileges or cloud infrastructure operator roles have no access to the build process or any means of observing it. The build process becomes entirely automated through an immutable, auditable runtime that removes all negligent behavior, including skipped scans, secret leaks and unsafe dependency usage. The implementation of zero-trust build systems achieves both insider threat defense and regulatory compliance while maintaining best practices for software integrity and high-assurance security.

Ensuring confidentiality and integrity in the build process has become critical, as software supply chains are increasingly targeted by external attackers. The isolated build environments presented in this paper provide a cloud-native, scalable solution that helps organizations safeguard their software pipelines from internal as well as external threats.

References

[1] Microsoft, What is an Insider Threat?, 2022. [Online]. Available: https://www.microsoft.com/en-us/security/business/security-101/what-is-insider-threat

[2] Microsoft, Learn about Insider Risk Management, 2025. [Online]. Available: https://learn.microsoft.com/en-us/purview/insider-risk-management

[3] IBM, What are Insider Threats?, 2021. [Online]. Available: https://www.ibm.com/think/topics/insider-threats

[4] Narendan Vaideeswaran, Insider Threats Explained, 2025. [Online]. Available: https://www.crowdstrike.com/en-us/cybersecurity-101/identity-protection/insider-threat/

[5] Matt Heusser, CI/CD Pipeline Security: Know the Risks and Best Practices, 2024. [Online]. Available: https://www.techtarget.com/searchitoperations/tip/9-ways-to-infuse-security-in-your-CI-CD-pipeline

[6] Martin Hermannsen, Intel SGX Enclave Instructions — Explained, 2020. [Online]. Available: https://medium.com/magicofc/establish-an-intel-sgx-enclave-c6208f820ff9

[7] Microsoft, Confidential Containers on Azure Container Instances, 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/container-instances/container-instances-confidential-overview

[8] AWS, What is Nitro Enclaves?. [Online]. Available: https://docs.aws.amazon.com/enclaves/latest/user/nitro-enclave.html

[9] CNCF, Securing Build Pipelines. [Online]. Available: https://tag-security.cncf.io/community/publications/supply-chain-security-tools/securing-build-pipelines/

[10] Ron Powell, How to Secure Your CI Pipeline, 2024. [Online]. Available: https://circleci.com/blog/secure-ci-pipeline/

[11] Fortinet, What Is An Insider Threat?. [Online]. Available: https://www.fortinet.com/resources/cyberglossary/insider-threats

[12] Sentinelone, What are Insider Threats? Types, Prevention & Risks, 2025. [Online]. Available: https://www.sentinelone.com/cybersecurity-101/threat-intelligence/insider-threats/

[13] Robert C. Swanson et al., "Method to Increase Cloud Availability and Silicon Isolation Using Secure Enclaves," US9798641B2, 2017. [Google Scholar] [Publisher Link]

[14] Pradipta Banerjee, and Samuel Ortiz, Understanding the Confidential Containers Attestation Flow, 2022. [Online]. Available: https://www.redhat.com/en/blog/understanding-confidential-containers-attestation-flow

[15] Microsoft, Confidential Containers on Azure, 2023. [Online]. Available: https://learn.microsoft.com/en-us/azure/confidential-computing/confidential-containers

[16] Matthew A. Johnson et al., "Confidential Container Groups: Implementing Confidential Computing on Azure Container Instances," *Queue*, vol. 22, no. 2, pp. 57-86, 2024. [CrossRef] [Google Scholar] [Publisher Link]