XXE in Openstack nova 23.0.0

HADESS

WWW.HADESS.IO

FOREWARD

OpenStack Nova is a widely used cloud computing platform that allows users to create and manage virtual machines and other resources. As with any complex software system, it is important to ensure that Nova is secure and protected against malicious attacks.

Recently, a vulnerability was discovered in OpenStack Nova that could potentially allow an attacker to gain unauthorized access to sensitive data or take control of the system. This vulnerability underscores the importance of ongoing vigilance and testing to identify and address security issues as they arise.

ADVISORY

Vulnerability	XXE
Vendor	Openstack
Module	nova
CVE	CVE-2021-29410
Version	23.0.0
Severity	High

Openstack Nova

OpenStack Nova is a core component of the OpenStack cloud computing platform that provides virtual machines (VMs) as a service. It is responsible for managing and provisioning compute resources in the cloud, including creating, scheduling, and terminating VMs.

Nova is designed to be highly scalable and flexible, with support for multiple hypervisors, including KVM, Xen, Hyper-V, and VMware. It also includes a robust API that allows users to programmatically provision and manage VMs, as well as a web-based dashboard for easy management and monitoring.

Some of the key features of Nova include:

- Multi-hypervisor support: Nova supports multiple hypervisors, allowing users to choose the best option for their workloads.
- Autoscaling: Nova includes built-in support for automatic scaling of VMs based on usage patterns and resource availability.
- High availability: Nova is designed to be highly available, with support for fault tolerance and automatic recovery in the event of a failure.
- Live migration: Nova supports live migration of VMs, allowing them to be moved between hosts without interrupting service.
- Security: Nova includes features such as network isolation and virtual machine introspection to enhance security in the cloud.

Overall, OpenStack Nova is a key component of the OpenStack cloud computing platform, providing a powerful and flexible way to manage and provision compute resources in the cloud.

What is XXE

XXE (XML External Entity) is a type of vulnerability that can occur in applications that process XML input. It arises when an attacker can control the contents of an XML document that is being parsed by an application, and can insert an external entity reference to a file or resource located on a different system.

When the application parses the XML document, it may access the external entity and include its contents in the output sent back to the user. This can allow the attacker to view sensitive data, execute arbitrary code, or perform other malicious actions on the vulnerable system.

To prevent XXE attacks, developers can implement secure coding practices, such as using a restrictive XML parser configuration or validating XML input against a schema. Additionally, web application firewalls and other security controls can help to detect and block XXE attacks.

Vulnerability

Root Cause

By default, the XMLParser in Python does not disable the use of external entities, which can make it vulnerable to XXE attacks. This means that if an attacker can control the contents of an XML document that is being parsed by Python's XMLParser, they may be able to include an external entity reference that points to a file or resource on a remote system, and the XMLParser will process that external entity, potentially leading to a security breach.

For example:

```
1 XMLParser(self, encoding=None, attribute_defaults=False, dtd_validation=False, load_dtd=False,
  no_network=True, ns_clean=False, recover=False, schema: XMLSchema =None, huge_tree=False,
  remove_blank_text=False, resolve_entities=True, remove_comments=False, remove_pis=False,
  strip_cdata=True, collect_ids=True, target=None, compact=True)
```

Available boolean keyword arguments:

attribute_defaults - inject default attributes from DTD or XMLSchema
dtd_validation - validate against a DTD referenced by the document
load_dtd - use DTD for parsing
no_network - prevent network access for related files (default: True)
ns_clean - clean up redundant namespace declarations
recover - try hard to parse through broken XML
remove_blank_text - discard blank text nodes that appear ignorable
remove_comments - discard comments
remove_pis - discard processing instructions
strip_cdata - replace CDATA sections by normal text content (default: True)
compact - save memory for short text content (default: True)
collect_ids - use a hash table of XML IDs for fast access (default: True, always True with DTD validation)
resolve_entities - replace entities by their text value (default: True)

Vulnerable Code

nova/nova/virt/libvirt/migration.py

Here is an example of vulnerable code that uses the XMLParser in Python and is susceptible to XXE attacks:

```
1 import xml.etree.ElementTree as ET
2
3 xml_data = '''
4 <?xml version="1.0" encoding="UTF-8"?>
5 <!DOCTYPE foo [
6 <!ELEMENT foo ANY>
7 <!ENTITY xxe SYSTEM "file:///etc/passwd">
8 ]>
9 <foo>&xxe;</foo>
10 '''
11
12 root = ET.fromstring(xml_data)
```

In this example, the XML data contains an external entity reference to the /etc/passwd file on the local system. When the XMLParser parses the XML data, it will resolve the external entity and include the contents of the /etc/passwd file in the resulting XML tree. This can potentially expose sensitive information to an attacker.

To prevent XXE attacks in this code, you can disable external entity resolution by passing a custom parser to the ET.fromstring() function, like this:

```
import xml.etree.ElementTree as ET

xml_data = '''
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY>
<!ENTITY xxe SYSTEM "file:///etc/passwd">
]>
<foo>&xxe;</foo>
'''

parser = ET.XMLParser(resolve_entities=False)
root = ET.fromstring(xml_data, parser=parser)
```

In this modified code, the resolve_entities parameter is set to False when creating the custom parser, which prevents external entities from being resolved. This effectively mitigates the XXE vulnerability in the original code.

To prevent XXE attacks in Python, you can use the defusedxml library, which provides a drop-in replacement for Python's built-in XML libraries (including the XMLParser), with XXE protection enabled by default. You can install the defusedxml library using pip:

```
pip install
```

Once installed, you can use the defusedxml library to parse XML documents with XXE protection enabled, like this:

```
from defusedxml.ElementTree import
parse
tree = parse(xml_string)
root = tree.getroot()
```

Alternatively, you can disable external entities in Python's built-in XMLParser by setting the resolve_entities parameter to False, like this:

```
from xml.etree import ElementTree as ET

parser = ET.XMLParser(resolve_entities=False)
tree = ET.fromstring(xml_string,
```

By disabling external entities, you can prevent XXE attacks in your Python code. However, it's important to note that this may not be sufficient on its own, and other security measures may also be necessary to fully protect your application against XXE attacks.

XXE Mitigations

There are several ways to prevent XXE attacks, including:

- Use a restrictive XML parser configuration: Developers can configure the XML parser to disallow the use of external entities. This can prevent attackers from including malicious files or resources in the XML input.

- Validate XML input against a schema: Developers can validate the XML input against a schema that defines the structure and content of the XML document. This can help to ensure that the input is well-formed and does not contain unexpected or malicious elements.

- Use whitelisting: Developers can use a whitelist to explicitly allow only the expected XML elements and attributes to be processed by the application. This can help to prevent unexpected or malicious input from being processed.

- Filter user input: Developers can sanitize user input to remove any unexpected or potentially malicious characters or elements. This can help to ensure that the XML input is well-formed and safe to process.

- Implement web application firewalls: Web application firewalls can help to detect and block XXE attacks by analyzing incoming traffic and blocking any requests that contain suspicious or malicious input.

It's important to note that preventing XXE attacks requires a combination of secure coding practices and security controls. Developers should also stay up-to-date with the latest security advisories and best practices to ensure that their applications remain secure.

About Hadess

Savior of your Business to combat cyber threats
Hadess performs offensive cybersecurity services through infrastructures and software that include vulnerability analysis, scenario attack planning, and implementation of custom integrated preventive projects. We organized our activities around the prevention of corporate, industrial, and laboratory cyber threats.

Contact Us

To request additional information about Hadess's services, please fill out the form below. A Hadess representative will contact you shortly.

Website:

www.hadess.io

Email:

Marketing@hadess.io

Phone No.

+989362181112

Company No.

982128427515

hadess_security

Hadess

Products and Services

→ SAST | Audit Your Products

Identifying and helping to address hidden weaknesses in your Applications.

→ RASP | Protect Applications and APIs Anywhere

Identifying and helping to address hidden weaknesses in your organization's security.

→ Penetration Testing | PROTECTION PRO

Fully assess your organization's threat detection and response capabilities with a simulated cyber-attack.

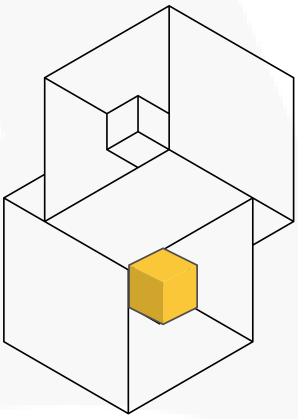
→ Red Teaming Operation | PROTECTION PRO

Fully assess your organization's threat detection and response capabilities with a simulated cyber-attack.

→ ThirdEye | Attack Surface Intelligence

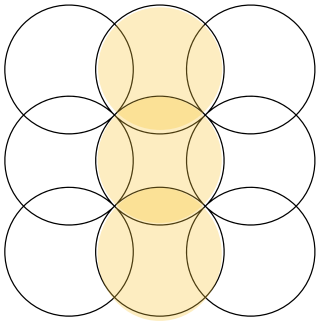
Find your company leakage and monitor attack vector.

Penetration Testing



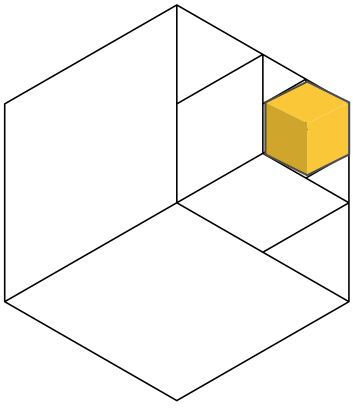
Module-Based

Penetration testing typically involves a combination of several different testing methods and techniques, which can be grouped into different modules. The specific modules used in a penetration test will depend on the goals and scope of the test, as well as the systems and services being evaluated.



Target-Based

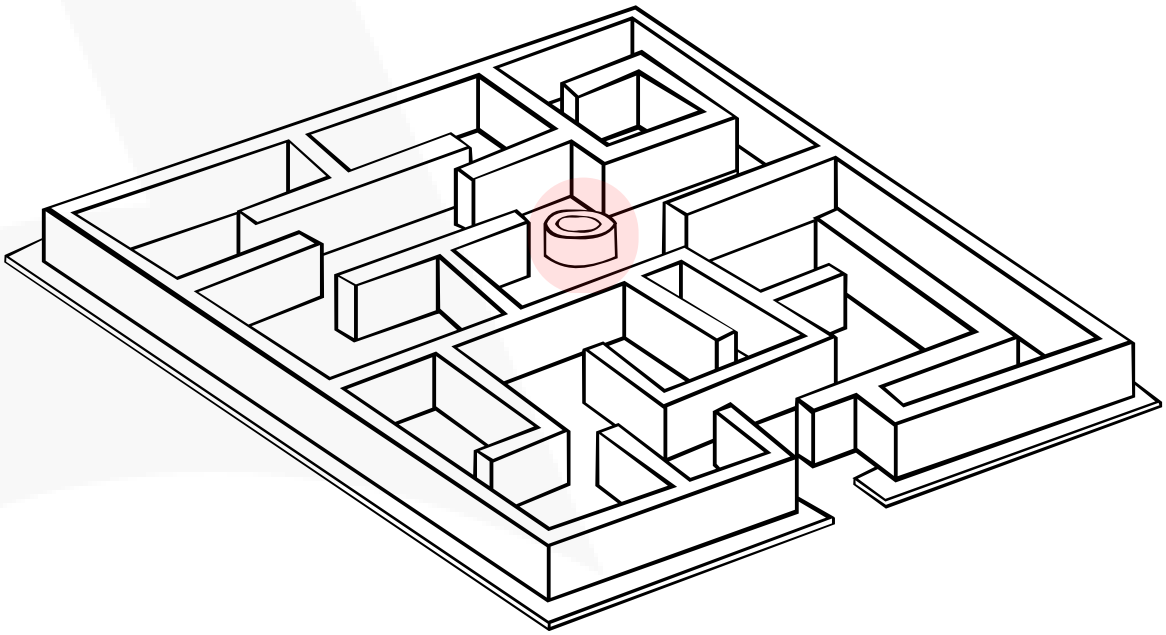
Penetration testing is a simulated cyber attack performed on a computer system, network, or web application to evaluate its security posture. The target of a penetration test can vary based on the specific needs and goals of the organization.



Priority-Based

The priority of a vulnerability during a penetration test is determined by the potential impact of the vulnerability, if exploited, and the likelihood of it being exploited. Vulnerabilities that pose a high risk to the target systems and services are given a higher priority, while those with a lower risk are given a lower priority.

Red Team Operation



OSINT

OSINT (Open-Source Intelligence) is a valuable tool for red teams, as it provides them with the ability to gather information about a target in a non-intrusive manner. Red teams use OSINT to gather information about the target's infrastructure, personnel, systems, and operations. This information can be used to identify potential weaknesses and vulnerabilities that can be exploited during a penetration test or simulated attack.

Hardening

Red team hardening is a technique used by red teams to evaluate an organization's security posture by simulating attacks and attempting to exploit vulnerabilities. The goal of red team hardening is to identify and remediate security weaknesses in an organization's systems, processes, and people, so that they are better prepared to defend against real-world attacks.

Goal-Based

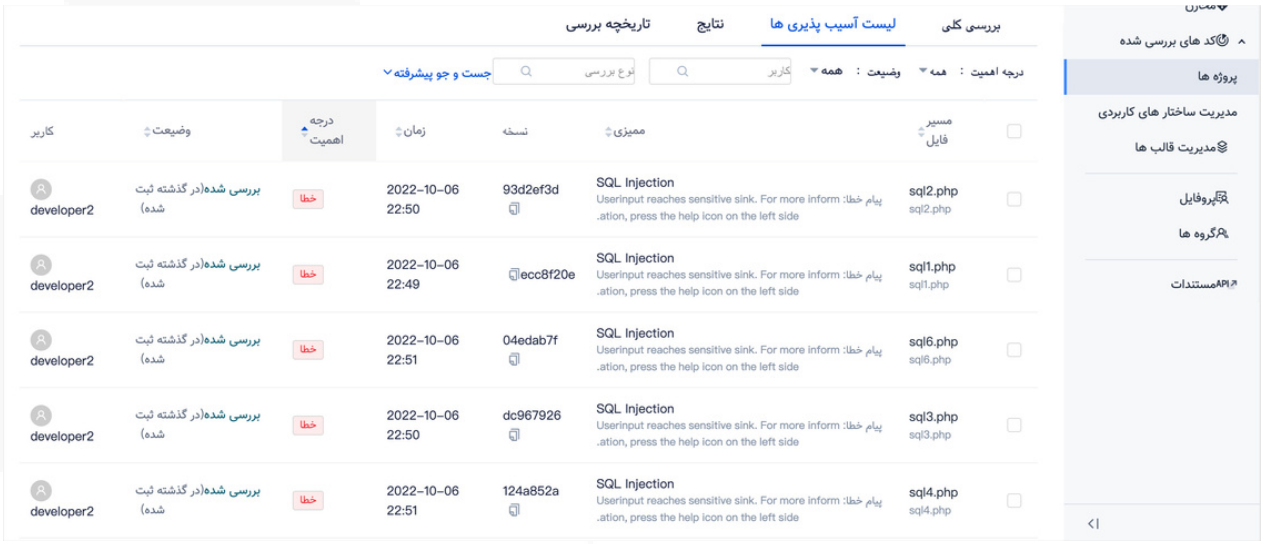
The goal of a red team exercise is to simulate an attack on an organization's systems, processes, and people to identify security weaknesses and vulnerabilities. Red teams use a variety of techniques, including penetration testing, social engineering, and physical security assessments, to test the effectiveness of an organization's defenses and to identify areas where they can be improved.

Asset-Based

Red team asset-based testing involves simulating an attack on a specific asset or group of assets within an organization. The goal of this type of red team exercise is to identify vulnerabilities in the targeted assets and to evaluate the effectiveness of the organization's defenses in protecting those assets.

HADESS

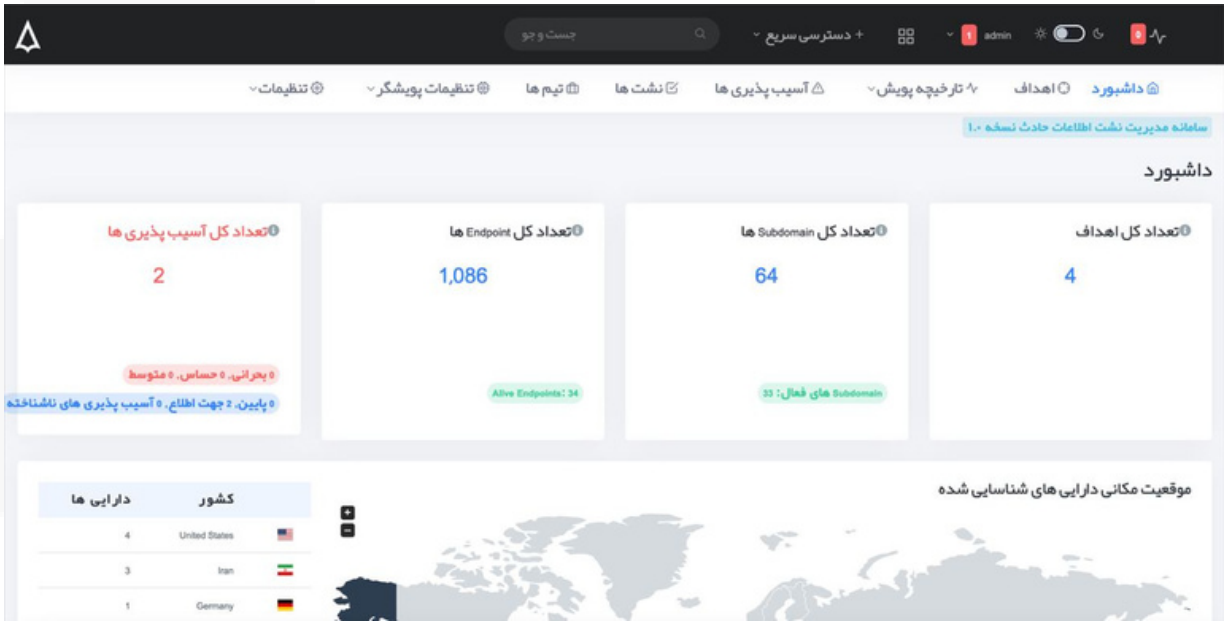
SAST



SAST (Static Application Security Testing) is a type of security testing that involves analyzing the source code of an application, without actually executing the code, to identify potential security vulnerabilities. SAST is performed early in the software development lifecycle, before the application is deployed, and is typically integrated into the development process as part of a DevSecOps approach.

HADESS

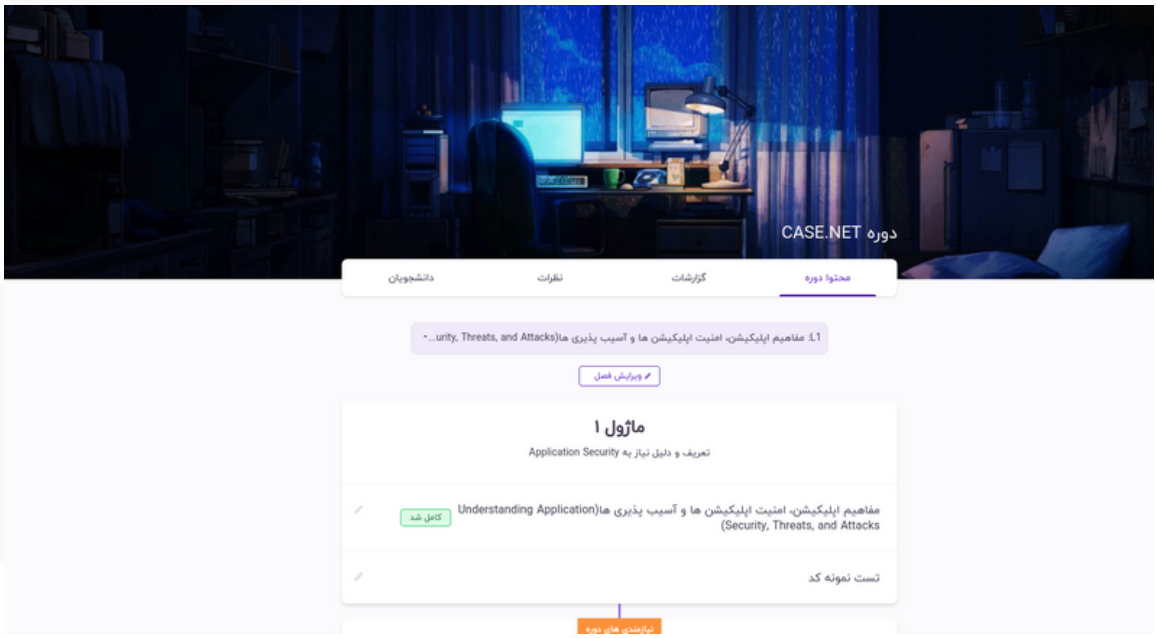
ASM



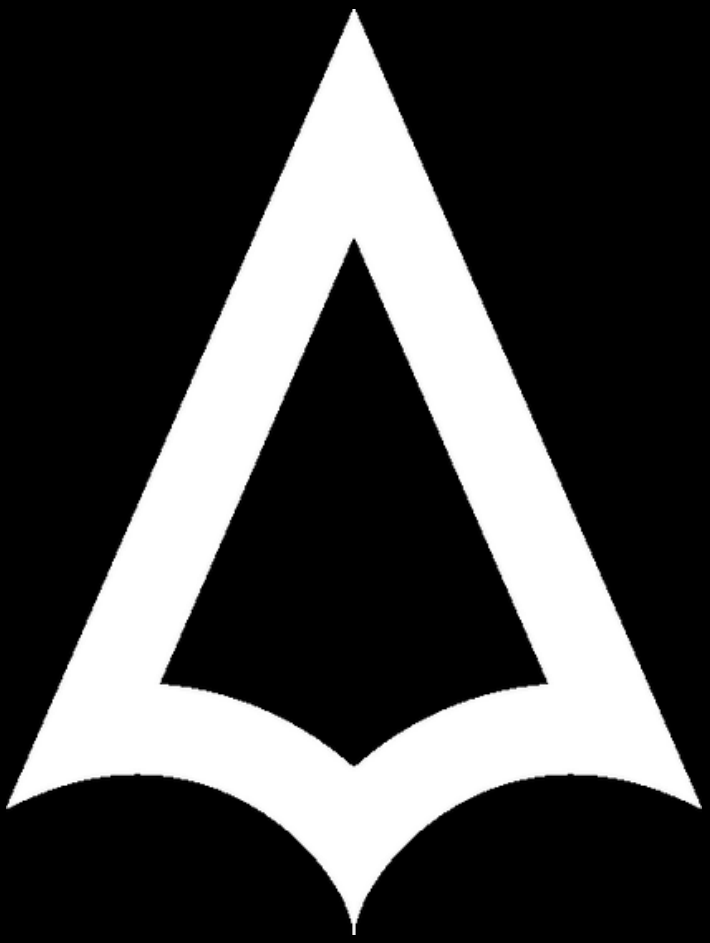
Attack surface management (ASM) is a security practice that involves reducing the attack surface of an organization's systems and services, making them less vulnerable to cyber attacks. The attack surface refers to the total number of potential entry points for an attacker, including network interfaces, applications, services, and other elements of an organization's technology infrastructure.

HADESS

Secure Coding



Secure coding is a software development practice that involves writing code that is free from vulnerabilities and that follows best practices for security. The goal of secure coding is to prevent security issues and vulnerabilities from being introduced into an application during the development process.



HADESS.IO