TP Tom Vermeil b3 ESGI SI.
Outils de développement et de protection du code
Dans le cadre de ce TP, j'ai d'abord forké le dépôt GitHub de l'application vulnérable afin de travailler sur une copie personnelle. J'ai ensuite utilisé **Snyk**, intégré directement dans GitHub, pour analyser automatiquement le code source et les dépendances du projet. Cet outil m'a permis d'identifier plusieurs failles de sécurité (CVE) de différentes sévérités, allant de **failles critiques** à des vulnérabilités **faibles**, ce qui m'a permis d'avoir une vue d'ensemble sur les risques présents dans l'application. En complément de cette analyse, j'ai exploité certaines de ces failles de manière concrète à l'aide de **Burp Suite**, notamment pour démontrer des attaques comme l'injection SQL ou le Cross-Site Scripting (XSS), ce qui m'a permis de mieux comprendre les implications réelles de ces vulnérabilités dans une application web

---
Désérialisation de Données Non Fiables
Fichier : Program.cs
Risque : Exécution de code à distance (RCE), accès aux fichiers, déni de service.

Remplacer :

JsonConvert.DeserializeObject<object>(Json, new JsonSerializerSettings() { TypeNameHandling = TypeNameHandling.All });

Par :

var settings = new JsonSerializerSettings
{
TypeNameHandling = TypeNameHandling.Objects, // Éviter 'All'
SerializationBinder = new SafeTypeBinder() };

Fichier : Controller/Controller.cs
Risque : Exécution de code à distance (RCE) via évaluation de code arbitraire.

Remplacer :

Result = CSharpScript.EvaluateAsync($"System.Math.Pow(2, {UserStr})")?.Result?.ToString();

Par :

if (int.TryParse(UserStr, out int exponent) && exponent >= 0 && exponent <= 30)
{
double resultValue = Math.Pow(2, exponent);
Result = resultValue.ToString(); }

Server-Side Request Forgery (SSRF)
Fichier : Controller/Controller.cs
Risque : SSRF, lecture de fichiers arbitraires, exécution de code à distance (via inclusions XSLT malveillantes ou requêtes réseau).

Remplacer :

```
var Xsl = XDocument.Parse(Xml);
var MyXslTrans = new XslCompiledTransform(enableDebug: true); var
Settings = new XsltSettings();
MyXslTrans.Load(Xsl.CreateReader(), Settings, null);
```

Par :

```
// Désactiver la fonction document() et les ressources externes via les paramètres XsltSettings et un XmlResolver null
var Xsl = XDocument.Parse(Xml);
var MyXslTrans = new XslCompiledTransform(enableDebug: false);
var Settings = new XsltSettings(enableDocumentFunction: false, enableScript: false);
MyXslTrans.Load(Xsl.CreateReader(), Settings, new XmlUrlResolver { Credentials = null }); // Ou utiliser 'null' comme resolver
```

Injection XXE (XML External Entity)
Fichier : Controller/Controller.cs
Risque : Injection XXE, lecture de fichiers arbitraires, déni de service, SSRF.

Remplacer :

```
XmlReaderSettings ReaderSettings = new XmlReaderSettings();
ReaderSettings.DtdProcessing = DtdProcessing.Parse;
ReaderSettings.XmlResolver = new XmlUrlResolver(); ReaderSettings.MaxCharactersFromEntities
= 6000;

using (MemoryStream stream = new MemoryStream(Encoding.UTF8.GetBytes(Xml))) {
XmlReader Reader = XmlReader.Create(stream, ReaderSettings);
var XmlDocument = new XmlDocument();
XmlDocument.XmlResolver = new XmlUrlResolver();
XmlDocument.Load(Reader);
return XmlDocument.InnerText;
}
```

Par :

```
XmlReaderSettings ReaderSettings = new XmlReaderSettings
{
```

```
DtdProcessing = DtdProcessing.Prohibit,
XmlResolver = null // Empêche le chargement d’entités externes
};

using (MemoryStream stream = new MemoryStream(Encoding.UTF8.GetBytes(Xml)))
{
using (XmlReader Reader = XmlReader.Create(stream, ReaderSettings))
{
var XmlDocument = new XmlDocument
{
XmlResolver = null // S’assurer que XmlDocument ne résout pas non plus les entités externes
};
XmlDocument.Load(Reader);
return XmlDocument.InnerText;
}
} ----------
```

Débordement d'entier dans zlib (CVE-2023-45853)
Fichier : Dockerfile
Risque : Débordement d'entier, déni de service (DoS), corruption potentielle de la mémoire

Package concerné : zlib/zlib1g@1:1.2.13.dfsg-
1 (via debian:latest)

Correction :
Aucune correction directe disponible pour le moment — aucune version corrigée n’a été publiée pour zlib1g dans Debian 12 (bookworm).

Solutions de contournement :

Option 1 : Basculer vers une image de base antérieure ou alternative avec une version sûre (si possible). Exemple (base Ubuntu) :

```
FROM ubuntu:22.04
```

Injection SQL

Fichier concerné : VLAIdentity.cs

Méthode : VulnerableQuery(string User, string Passwd)

Détail : L’authentification utilise une requête SQL dynamique non sécurisée.

PoC :
Injecter du SQL dans les paramètres User et Passwd :

User=admin' OR '1'='1&Passwd=any_password

Désérialisation Insecure (RCE possible)

Fichier concerné : Controller.cs

Méthode : VulnerableDeserialize(string i)

Détail : Utilisation de JsonConvert.DeserializeObject sans validation de l'entrée.

PoC :
Injecter un objet malveillant dans la requête JSON :

```
{
  "$type": "System.IO.FileStream, mscorlib",
  "path": "C:\\Windows\\System32\\calc.exe"
}
```

Exécution de commandes système

Fichier concerné : Controller.cs

Méthode : VulnerableCmd(string cmd)

Détail : Commande insérée directement dans Process.Start sans validation.

PoC :
Injecter une commande shell pour exécuter un script :

nslookup google.com; whoami

IDOR (Insecure Direct Object Reference)

Fichier concerné : Controller.cs

Méthode : VulnerableObjectReference(string i)

Détail : L’utilisateur peut accéder directement aux fichiers en manipulant les noms.

PoC :
Accéder à un fichier sensible en modifiant l’ID :

/Data/../../../../../etc/passwd

Téléversement de fichier non sécurisé

Fichier concerné : Controller.cs

Méthode : VulnerableHandleFileUpload(IFormFile file, string ip)

Détail : Aucun contrôle sur le nom ou le type de fichier téléchargé.

PoC :
Télécharger une Web Shell :

POST /Patch HTTP/1.1
Content-Type: multipart/form-data; boundary=----WebKitFormBoundary
Content-Disposition: form-data; name="file"; filename="shell.php"

Faiblesse dans le JWT (modification des privilèges)

Fichier concerné : VLAIdentity.cs

Méthodes : VulnerableGenerateToken / VulnerableAdminValidateToken

Détail : Le secret fixe dans le JWT permet la falsification de tokens admin.

PoC :
Modifier le JWT pour se faire passer pour un administrateur :

Header: {"alg": "HS256", "typ": "JWT"}
Payload: {"Id": "admin", "IsAdmin": "True"}
Signature: [Sign the payload using the fixed secret]

Manque de validation des entrées (Path Traversal)

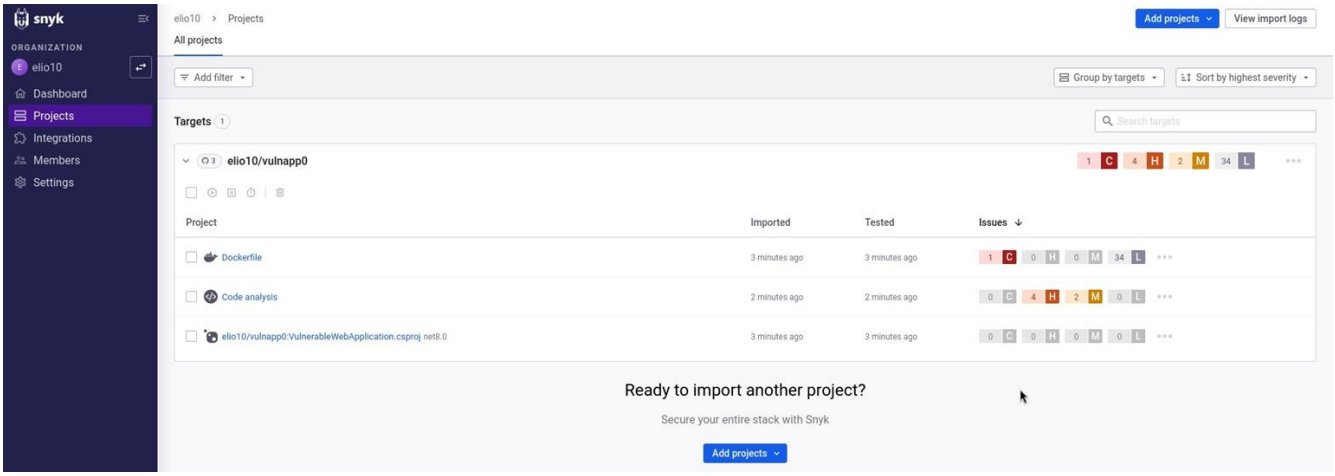
Fichier concerné : Controller.cs

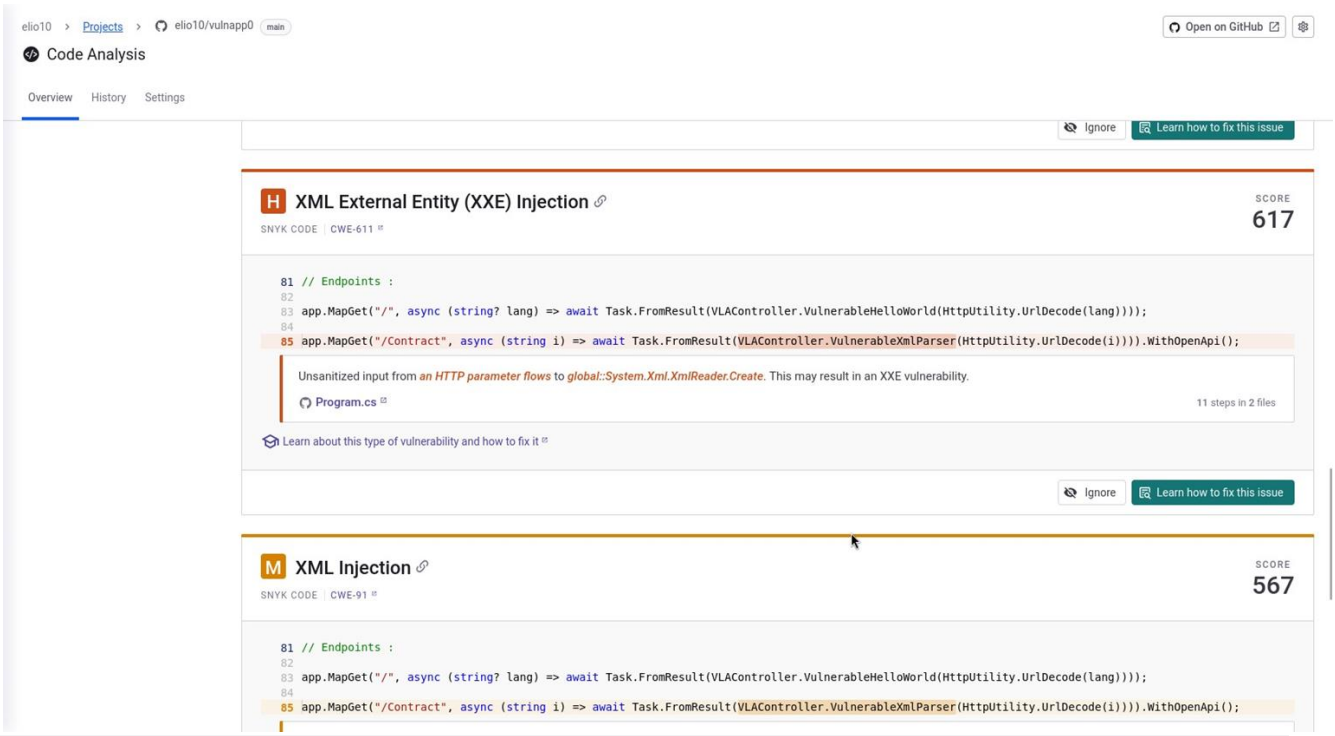
Méthode : VulnerableWebRequest(string i)

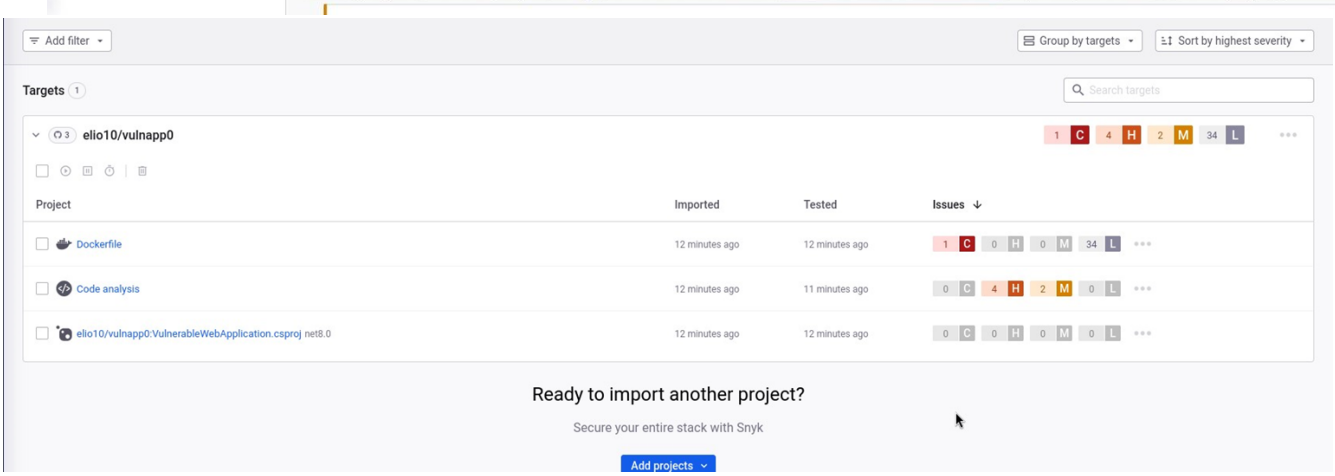
Détail : L'entrée utilisateur est utilisée pour déterminer le chemin sans validation.

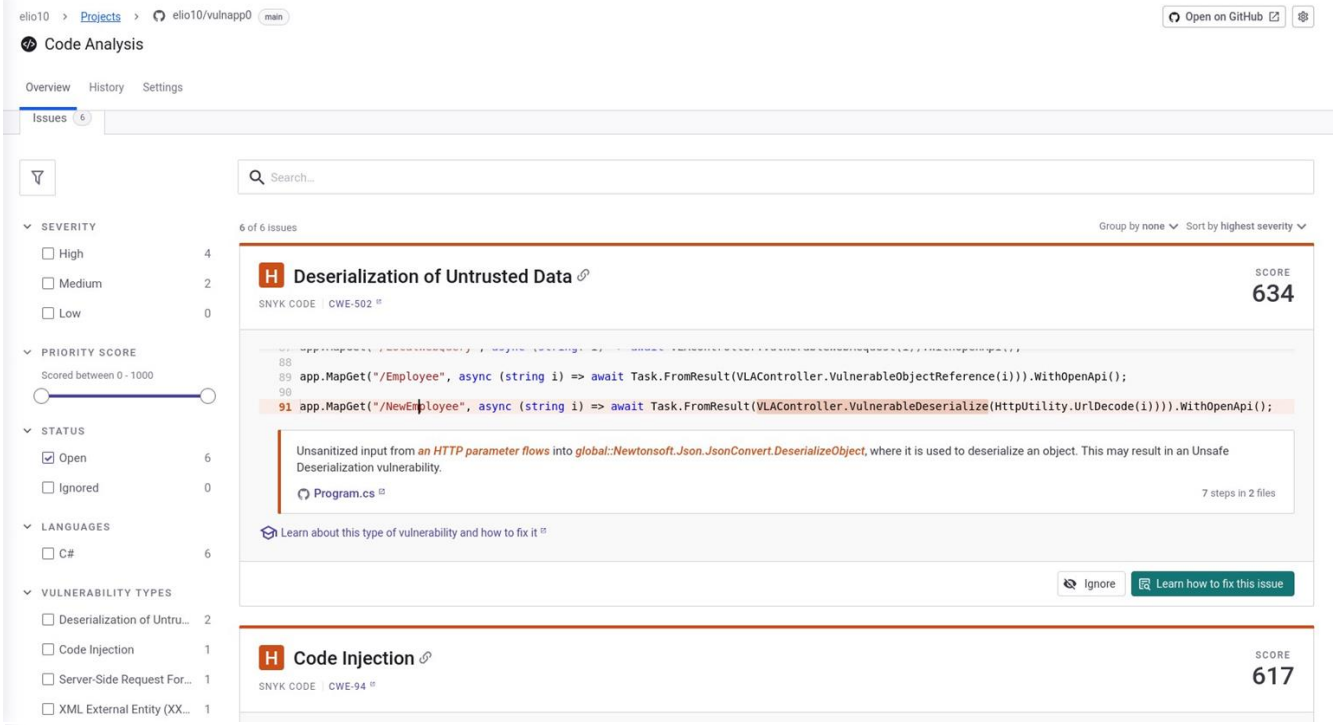
PoC :
Effectuer une attaque de traversée de répertoire :

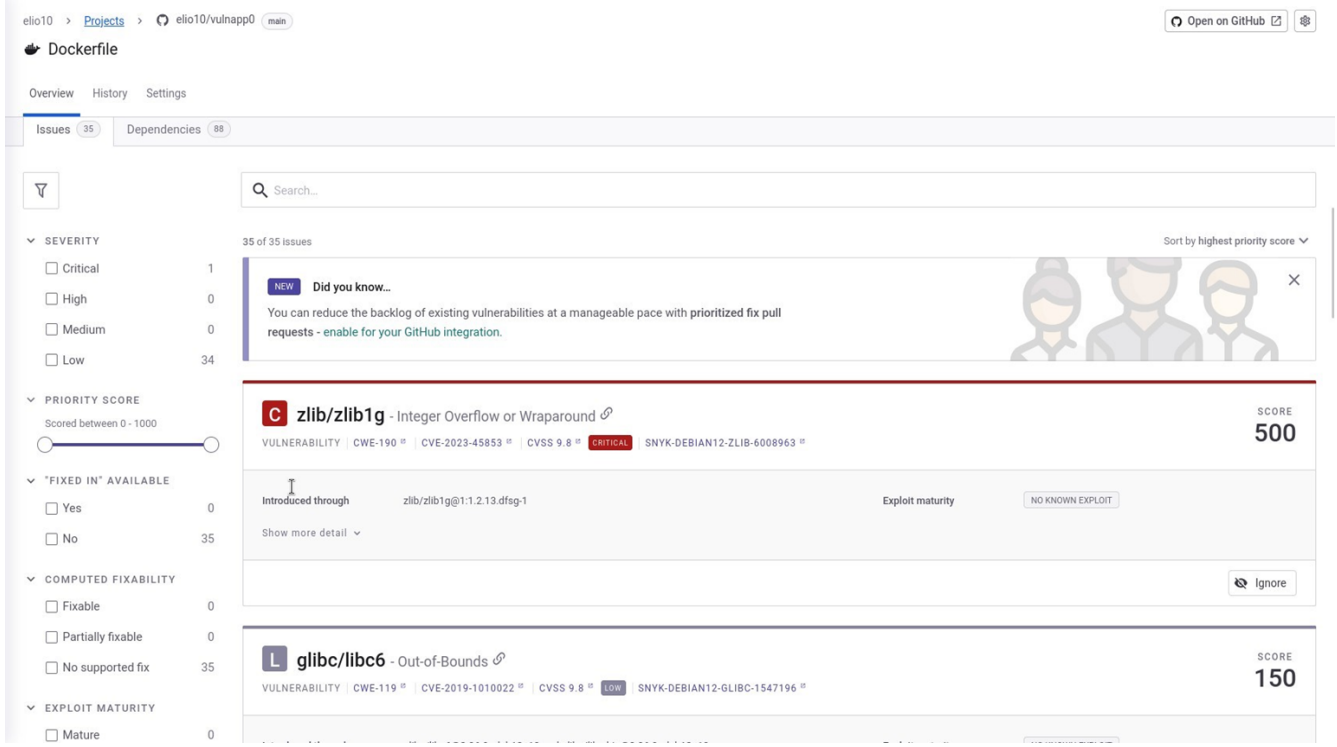
/somebase/../../../../../etc/passwd

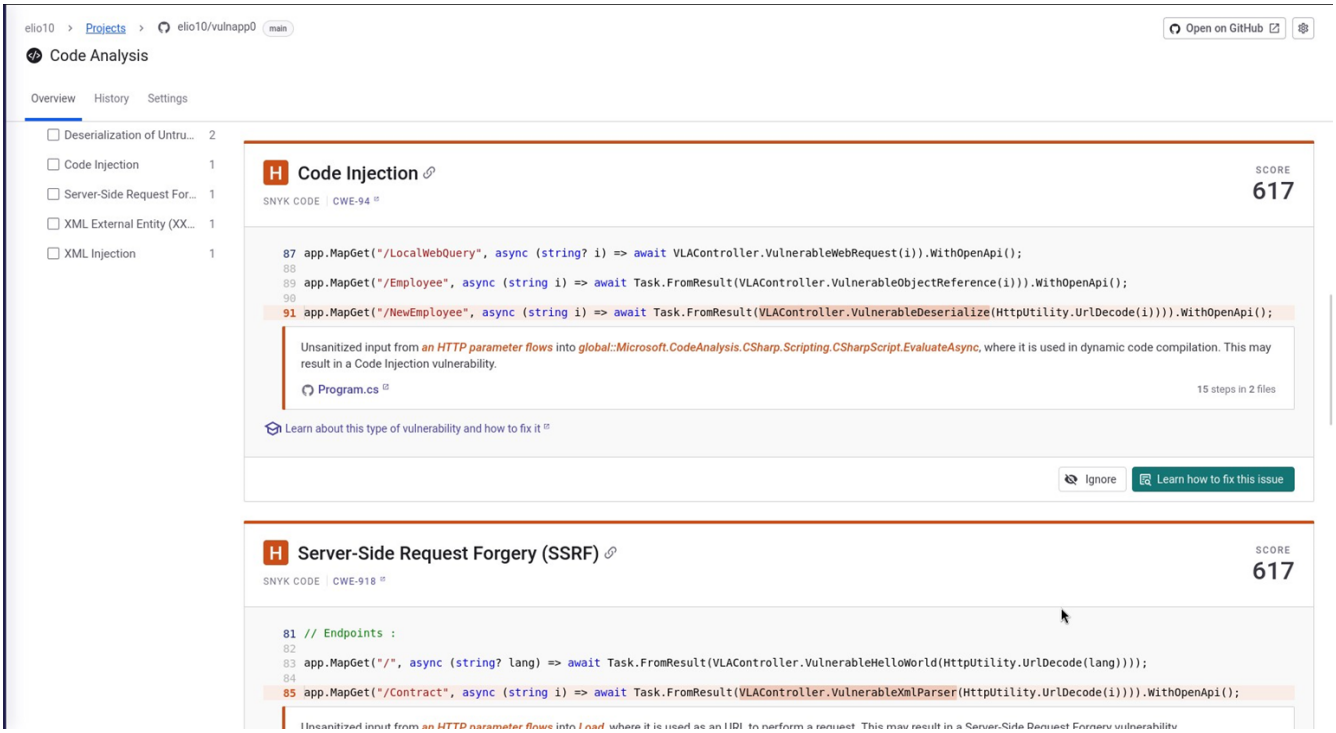

snyk
ORGANIZATION
elio10
Dashboard
Projects
Integrations
Members
Settings
elio10 > Projects
All projects
Add projects
View import logs
Add filter
Group by targets
Sort by highest severity
Targets 1
Search targets
elio10/vulnapp0
Project
Imported
Tested
Issues
Dockerfile
3 minutes ago
3 minutes ago
Code analysis
2 minutes ago
2 minutes ago
elio10/vulnapp0:VulnerableWebApplication.csproj net8.0
3 minutes ago
3 minutes ago
Ready to import another project?
Secure your entire stack with Snyk
Add projects


elio10 > Projects > elio10/vulnapp0 main
Open on GitHub
Code Analysis
Overview
History
Settings
Ignore
Learn how to fix this issue
XML External Entity (XXE) Injection
SNYK CODE | CWE-611
SCORE
617
81 // Endpoints :
82
83 app.MapGet("/", async (string? lang) => await Task.FromResult(VLAController.VulnerableHelloWorld(HttpUtility.UrlDecode(lang))));
84
85 app.MapGet("/Contract", async (string i) => await Task.FromResult(VLAController.VulnerableXmlParser(HttpUtility.UrlDecode(i))).WithOpenApi();
Unsanitized input from an HTTP parameter flows to global::System.Xml.XmlReader.Create. This may result in an XXE vulnerability.
Program.cs
11 steps in 2 files
Learn about this type of vulnerability and how to fix it
Ignore
Learn how to fix this issue
XML Injection
SNYK CODE | CWE-91
SCORE
567
81 // Endpoints :
82
83 app.MapGet("/", async (string? lang) => await Task.FromResult(VLAController.VulnerableHelloWorld(HttpUtility.UrlDecode(lang))));
84
85 app.MapGet("/Contract", async (string i) => await Task.FromResult(VLAController.VulnerableXmlParser(HttpUtility.UrlDecode(i))).WithOpenApi();


Add filter
Group by targets
Sort by highest severity
Targets 1
Search targets
elio10/vulnapp0
Project
Imported
Tested
Issues
Dockerfile
12 minutes ago
12 minutes ago
Code analysis
12 minutes ago
11 minutes ago
elio10/vulnapp0:VulnerableWebApplication.csproj net8.0
12 minutes ago
12 minutes ago
Ready to import another project?
Secure your entire stack with Snyk
Add projects


elio10 › Projects › elio10/vulnapp0 main
Open on GitHub
Code Analysis
Overview History Settings
Issues 6
Search...
SEVERITY
High 4
Medium 2
Low 0
PRIORITY SCORE
Scored between 0 - 1000
STATUS
Open 6
Ignored 0
LANGUAGES
C# 6
VULNERABILITY TYPES
Deserialization of Untru... 2
Code Injection 1
Server-Side Request For... 1
XML External Entity (XX... 1
6 of 6 issues
Group by none
Sort by highest severity
Deserialization of Untrusted Data
SNYK CODE | CWE-502
SCORE
634
app.MapGet("/Employee", async (string i) => await Task.FromResult(VLAController.VulnerableObjectReference(i))).WithOpenApi();
app.MapGet("/NewEmployee", async (string i) => await Task.FromResult(VLAController.VulnerableDeserialize(HttpUtility.UrlDecode(i)))).WithOpenApi();
Unsanitized input from an HTTP parameter flows into global::Newtonsoft.Json.JsonConvert.DeserializeObject, where it is used to deserialize an object. This may result in an Unsafe Deserialization vulnerability.
Program.cs
7 steps in 2 files
Learn about this type of vulnerability and how to fix it
Ignore
Learn how to fix this issue
Code Injection
SNYK CODE | CWE-94
SCORE
617


elio10 › Projects › elio10/vulnapp0 main
Open on GitHub
Dockerfile
Overview History Settings
Issues 35
Dependencies 88
Search...
SEVERITY
Critical 1
High 0
Medium 0
Low 34
PRIORITY SCORE
Scored between 0 - 1000
"FIXED IN" AVAILABLE
Yes 0
No 35
COMPUTED FIXABILITY
Fixable 0
Partially fixable 0
No supported fix 35
EXPLOIT MATURITY
Mature 0
35 of 35 issues
Sort by highest priority score
NEW Did you know...
You can reduce the backlog of existing vulnerabilities at a manageable pace with prioritized fix pull requests - enable for your GitHub integration.
zlib/zlib1g - Integer Overflow or Wraparound
VULNERABILITY | CWE-190 | CVE-2023-45853 | CVSS 9.8 | CRITICAL | SNYK-DEBIAN12-ZLIB-6008963
SCORE
500
Introduced through
zlib/zlib1g@1:1.2.13.dfsg-1
Exploit maturity
NO KNOWN EXPLOIT
Show more detail
Ignore
glibc/libc6 - Out-of-Bounds
VULNERABILITY | CWE-119 | CVE-2019-1010022 | CVSS 9.8 | LOW | SNYK-DEBIAN12-GLIBC-1547196
SCORE
150


elio10 > Projects > elio10/vulnapp0 main
Open on GitHub
Code Analysis
Overview
History
Settings
Deserialization of Untru... 2
Code Injection 1
Server-Side Request For... 1
XML External Entity (XX... 1
XML Injection 1
Code Injection
SNYK CODE | CWE-94
SCORE
617
87 app.MapGet("/LocalWebQuery", async (string? i) => await VLAController.VulnerableWebRequest(i)).WithOpenApi();
88
89 app.MapGet("/Employee", async (string i) => await Task.FromResult(VLAController.VulnerableObjectReference(i))).WithOpenApi();
90
91 app.MapGet("/NewEmployee", async (string i) => await Task.FromResult(VLAController.VulnerableDeserialize(HttpUtility.UrlDecode(i)))).WithOpenApi();
Unsanitized input from an HTTP parameter flows into global::Microsoft.CodeAnalysis.CSharp.Scripting.CSharpScript.EvaluateAsync, where it is used in dynamic code compilation. This may result in a Code Injection vulnerability.
Program.cs
15 steps in 2 files
Learn about this type of vulnerability and how to fix it
Ignore
Learn how to fix this issue
Server-Side Request Forgery (SSRF)
SNYK CODE | CWE-918
SCORE
617
81 // Endpoints :
82
83 app.MapGet("/", async (string? lang) => await Task.FromResult(VLAController.VulnerableHelloWorld(HttpUtility.UrlDecode(lang))));
84
85 app.MapGet("/Contract", async (string i) => await Task.FromResult(VLAController.VulnerableXmlParser(HttpUtility.UrlDecode(i)))).WithOpenApi();