ДОДАТОК А

Код програмного продукту

```
namespace NVDHunter.Controllers
{
    [ApiController]
    public class VulnerabilitiesController : ControllerBase
    {
        private readonly IVulnerabilitiesManager vulnerabilitiesManager;

        /// <summary>
        /// Creates instance of a controller
        /// </summary>
        /// <param name="vulnerabilitiesManager"></param>
        public VulnerabilitiesController(IVulnerabilitiesManager vulnerabilitiesManager)
        {
            this.vulnerabilitiesManager = vulnerabilitiesManager;
        }

        /// <summary>
        /// Get Vulnerabitilies by filter
        /// </summary>
        ///
        /// <returns></returns>
        [HttpGet("Vulnerabitilies")]
        [ProducesResponseType(200)]
        [ProducesResponseType(404)]
        public ActionResult<Vulnerabilities> Get(string product, [FromQuery] Filter filter)
        {
            try
            {
                var response = this.vulnerabilitiesManager.Get(product, filter);

                return this.Ok(response);
            }
            catch (Exception ex)
            {
                return this.BadRequest(ex.Message);
            }
        }

        /// <summary>
        /// Get vulnerability by cve
        /// </summary>
        /// <param name="cve"></param>
        /// <returns></returns>
        [HttpGet("Vulnerabilities/{cve}")]
        [ProducesResponseType(200)]
        [ProducesResponseType(400)]
        [ProducesResponseType(404)]
        public ActionResult<VulnerabilityDetails> GetById(string cve)
        {
            try
            {
                var response = this.vulnerabilitiesManager.GetById(cve);

                if (response == null)
                {
```

```
                    return this.NotFound();
                }

                return this.Ok(response);
            }
            catch (Exception ex)
            {
                return this.BadRequest(ex.Message);
            }
        }

        /// <summary>
        /// Get vulnerability assessment
        /// </summary>
        /// <param name="cve"></param>
        /// <param name="privileges"></param>
        /// <returns></returns>
        [HttpGet("VulnerabilityAssessment/{cve}")]
        [ProducesResponseType(200)]
        [ProducesResponseType(400)]
        [ProducesResponseType(404)]
        public ActionResult<VulnerabilityAssessment> GetAssessment(string cve, [FromQuery]
PrivilegesEnum privileges)
        {
            try
            {
                string privilegesValue = null;
                if ((int)privileges == 0)
                {
                    privilegesValue = "L";
                }
                else if ((int)privileges == 1)
                {
                    privilegesValue = "H";
                }
                else
                {
                    privilegesValue = "N";
                }

                var response = this.vulnerabilitiesManager.GetAssessment(cve,
privilegesValue);

                if (response == null)
                {
                    return this.NotFound();
                }

                return this.Ok(response);
            }
            catch (Exception ex)
            {
                return this.BadRequest(ex.Message);
            }
        }
    }
       }
namespace NVDHunter.API.Extensions.Swagger
{
   public static class SwaggerConfiguration
    {
        public static void SwaggerConfigureServices(this IServiceCollection services,
IConfiguration configuration, Info swaggerInfo = null)
        {
```

```
            var appName = "NVDHunter";

            if (swaggerInfo == null)
            {
                swaggerInfo = SwaggerConstants.GetDefaultInfo(appName);
            }

            services.AddSwaggerGen(conf =>
            {
                conf.SwaggerDoc("swagger", swaggerInfo);

                conf.IncludeXmlComments(GetXmlFilePath(appName));


//conf.IncludeXmlComments(Path.Combine(Path.GetDirectoryName(Assembly.GetEntryAssembly().Loc
ation), "NVDHunter.xml"));

                conf.ResolveConflictingActions(apiDescriptions => apiDescriptions.First());
                conf.CustomOperationIds(x => $"{x.HttpMethod}_{x.RelativePath}");

                var security = new Dictionary<string, IEnumerable<string>>
                {
                    {"cookieAuth", Enumerable.Empty<string>()},
                };
                conf.AddSecurityRequirement(security);

                conf.DescribeAllEnumsAsStrings();
            });
        }

        public static void UseCustomSwagger(this IApplicationBuilder app)
        {
            app.UseSwagger();
            app.UseSwaggerUI(conf =>
            {
                conf.SwaggerEndpoint($"swagger/swagger.json", string.Empty);
                conf.DocExpansion(DocExpansion.None);
            });
        }

        private static string GetXmlFilePath(string fileName)
        {
            var xmlFile = $"{fileName}.xml";

            return Path.Combine(AppContext.BaseDirectory, xmlFile);
        }
    }
}


namespace NVDHunter.API.Extensions.Swagger
{
   public static class SwaggerConstants
    {
        public static Info GetDefaultInfo(string appName)
        {
            return new Info
            {
                Title = appName,
                Description = "Vulnerabilities assessment API"
            };
        }
    }
}
namespace NVDHunter.BussinessLogic.CountersFunctionality
```

```
{
    public static class AssessmentGenerator
    {
        public static string GetAssessment(double baseScore)
        {
            if (baseScore <= AssessmentsConstants.MAX_LOW_VALUE)
            {
                return AssessmentsConstants.LOW;
            }

            if (baseScore >= AssessmentsConstants.MIN_MEDIUM_VALUE && baseScore <=
AssessmentsConstants.MAX_MEDIUM_VALUE)
            {
                return AssessmentsConstants.MEDIUM;
            }

            if (baseScore >= AssessmentsConstants.MIN_HIGH_VALUE && baseScore <=
AssessmentsConstants.MAX_HIGH_VALUE)
            {
                return AssessmentsConstants.HIGH;
            }

            return AssessmentsConstants.CRITICAL;
        }
    }
}
namespace NVDHunter.BussinessLogic.CountersFunctionality
{
    public class BaseScoreCounter
    {
        private readonly Dictionary<string, string> marksDictionary;

        public BaseScoreCounter(string vectorString)
        {
            this.marksDictionary =
MarksDictionaryGenerator.CreateMarksDictionary(vectorString);
        }

        public double GetBaseScore(string privilegesString)
        {
            var baseScore = this.CalculateAuxiliaryValues(privilegesString);

            return baseScore;
        }

        private double CalculateAuxiliaryValues(string privilegesString)
        {
            var baseScore = 0.0;

            var av = this.GetAttactVectorScore();
            var ac = this.GetAttactComplexityScore();
            var ui = this.GetUserInteractionScore();
            var ci = this.GetConfidentialityImpactScore();
            var ii = this.GetIntegrityImpactScore();
            var ai = this.GetAvailabilityImpactScore();
            var pr = this.GetPrivilegesRequiredScore(privilegesString);

            var exploitability = 20 * av * ac * pr * ui;
            var preISC = 10.41*(1 - (1 - ci) * (1 - ii) * (1 - ai));
            baseScore = this.GenerateBaseScore(preISC, exploitability);

            return baseScore;
        }
```

```
        private double GetAttactVectorScore()
        {
            string paramMark = null;

            if (this.marksDictionary.TryGetValue(VectorConstants.ATTACK_VECTOR, out
paramMark))
            {
                if (paramMark == MarksConctants.ADJACENT_NETWORK)
                {
                    return ValuesOfMarks.AV_ADJACENT_NETWORK_VALUES;
                }

                if (paramMark == MarksConctants.NETWORK)
                {
                    return ValuesOfMarks.AV_NETWORK_VALUES;
                }

                if (paramMark == MarksConctants.LOCAL)
                {
                    return ValuesOfMarks.AV_LOCAL_VALUES;
                }

                if (paramMark == MarksConctants.PHYSICAL)
                {
                    return ValuesOfMarks.AV_PHYSICAL_VALUES;
                }
            }

            return ValuesOfMarks.AV_NONE_VALUES;
        }

        private double GetAttactComplexityScore()
        {
            string paramMark = null;

            if (this.marksDictionary.TryGetValue(VectorConstants.ATTACK_COMPLEXITY, out
paramMark))
            {
                if (paramMark == MarksConctants.LOW)
                {
                    return ValuesOfMarks.AC_LOW_VALUES;
                }

                if (paramMark == MarksConctants.HIGH)
                {
                    return ValuesOfMarks.AC_HIGH_VALUES;
                }
            }

            return ValuesOfMarks.AC_NONE_VALUES;
        }

        private double GetPrivilegesRequiredScore(string privilegesString)
        {
            if (privilegesString == MarksConctants.LOW)
            {
                return ValuesOfMarks.PR_LOW_VALUES;
            }

            if (privilegesString == MarksConctants.HIGH)
            {
                return ValuesOfMarks.PR_HIGH_VALUES;
            }
```

```
            return ValuesOfMarks.PR_NONE_VALUES;
        }

        private double GetUserInteractionScore()
        {
            string paramMark = null;

            if (this.marksDictionary.TryGetValue(VectorConstants.USER_INTERACTION, out
paramMark))
            {
                if (paramMark == MarksConctants.REQUIRED)
                {
                    return ValuesOfMarks.UI_REQUIRED_VALUES;
                }

                if (paramMark == MarksConctants.NONE)
                {
                    return ValuesOfMarks.UI_NONE_VALUES;
                }
            }

            return ValuesOfMarks.UI_NOT_DEFINED_VALUES;
        }

        private double GenerateBaseScore(double preISC, double exploitability)
        {
            string paramMark = null;
            double authScore = 0.0;
            double isc = 0.0;

            if (this.marksDictionary.TryGetValue(VectorConstants.SCOPE, out paramMark))
            {
                if (paramMark == MarksConctants.CHANGED)
                {
                    authScore =  ValuesOfMarks.S_CHANGED_VALUES;

                    isc = authScore * (preISC - 0.029) - 3.25 * Math.Pow((preISC - 0.02),
15.0);

                    return Math.Abs(1.08 * (6*isc + 4 * exploitability - 1.5) * authScore);
                }

                if (paramMark == MarksConctants.UNCHANGED)
                {
                    authScore = ValuesOfMarks.S_UNCHANGED_VALUES;

                    isc = 1.176 + authScore;

                    return Math.Abs(1.08 * (0.6 * preISC + 0.4 * exploitability - 1.5) *
isc);
                }
            }

            authScore = ValuesOfMarks.S_NONE_VALUES;
            isc = 1.176 + authScore;

            var test = 0.6 * preISC;
            var t2 = 0.4 * exploitability + 1.5;
            var t3 = 1.08 * isc;
            var t4 = t3 * (test + t2);
            var t5 = Math.Abs(t4);

            return t5;
```

```
        }

        private double GetConfidentialityImpactScore()
        {
            string paramMark = null;

            if (this.marksDictionary.TryGetValue(VectorConstants.CONFIDENTIALITY_IMPACT, out
paramMark))
            {
                if (paramMark == MarksConctants.LOW)
                {
                    return ValuesOfMarks.CI_LOW_VALUES;
                }

                if (paramMark == MarksConctants.HIGH)
                {
                    return ValuesOfMarks.CI_HIGH_VALUES;
                }

                if (paramMark == MarksConctants.NONE)
                {
                    return ValuesOfMarks.CI_NONE_VALUES;
                }
            }

            return ValuesOfMarks.CI_NOT_DEFINED_VALUES;
        }

        private double GetIntegrityImpactScore()
        {
            string paramMark = null;

            if (marksDictionary.TryGetValue(VectorConstants.INTEGRITY_IMPACT, out
paramMark))
            {
                if (paramMark == MarksConctants.LOW)
                {
                    return ValuesOfMarks.I_LOW_VALUES;
                }

                if (paramMark == MarksConctants.HIGH)
                {
                    return ValuesOfMarks.I_HIGH_VALUES;
                }

                if (paramMark == MarksConctants.NONE)
                {
                    return ValuesOfMarks.I_NONE_VALUES;
                }
            }

            return ValuesOfMarks.I_NOT_DEFINED_VALUES;
        }

        private double GetAvailabilityImpactScore()
        {
            string paramMark = null;

            if (this.marksDictionary.TryGetValue(VectorConstants.AVAILABILITY_IMPACT, out
paramMark))
            {
                if (paramMark == MarksConctants.LOW)
                {
                    return ValuesOfMarks.A_LOW_VALUES;
```

```
                }

                if (paramMark == MarksConctants.HIGH)
                {
                    return ValuesOfMarks.A_HIGH_VALUES;
                }

                if (paramMark == MarksConctants.NONE)
                {
                    return ValuesOfMarks.A_NONE_VALUES;
                }
            }

            return ValuesOfMarks.A_NOT_DEFINED_VALUES;
        }
    }
       }
namespace NVDHunter.BussinessLogic.CountersFunctionality
{
   public static class MarksDictionaryGenerator
    {
        public static Dictionary<string, string> CreateMarksDictionary(string vectorString)
        {
            vectorString = vectorString.ToUpper();

            var marksKeyValuePairs = new Dictionary<string, string>();

            var arrayMarks = vectorString.Split(new string[] { "/" },
StringSplitOptions.RemoveEmptyEntries);

            foreach (var item in arrayMarks)
            {
                var keyValueArray = item.Split(new string[] { ":" },
StringSplitOptions.RemoveEmptyEntries);

                marksKeyValuePairs.Add(keyValueArray[0], keyValueArray[1]);
            }

            return marksKeyValuePairs;
        }

    }
}
namespace NVDHunter.BussinessLogic.CountersFunctionality
{
   public class TemporalyScoreCounter
    {
        private readonly Dictionary<string, string> marksDictionary;

        public TemporalyScoreCounter(string vectorString)
        {
            this.marksDictionary =
MarksDictionaryGenerator.CreateMarksDictionary(vectorString);
        }

        public double GetTemporalyScore(double baseScore)
        {
            double temporalyScore = 0.0;

            var exploitability = this.GenereteExploitabilityScore();
            var remendiationLevel = this.GenereteRemediationLevelScore();
            var report = this.GenereteReportScore();

            temporalyScore = baseScore * exploitability * remendiationLevel * report;
```

```
            return temporalyScore;
        }

        private double GenereteExploitabilityScore()
        {
            string paramMark = null;

            if (this.marksDictionary.TryGetValue(VectorConstants.EXPLOITABILITY, out
paramMark))
            {
                if (paramMark == MarksConctants.EXPLOIT_EXIST)
                {
                    return ValuesOfMarks.EXPLOITABILITY_UNPROVEN_VALUES;
                }

                if (paramMark == MarksConctants.FUNCTIONAL_EXPLOIT_EXIST)
                {
                    return ValuesOfMarks.EXPLOITABILITY_FUNCTIONAL_VALUES;
                }

                if (paramMark == MarksConctants.PROOF_OF_CONCEPT_CODE)
                {
                    return ValuesOfMarks.EXPLOITABILITY_PROOF_OF_CONCERT_VALUES;
                }

                if (paramMark == MarksConctants.HIGH)
                {
                    return ValuesOfMarks.EXPLOITABILITY_HIGH_VALUES;
                }

                if (paramMark == MarksConctants.NOT_DEFINED)
                {
                    return ValuesOfMarks.EXPLOITABILITY_NOT_DEFINED_VALUES;
                }
            }

            return ValuesOfMarks.EXPLOITABILITY_NONE_VALUES;
        }

        private double GenereteRemediationLevelScore()
        {
            string paramMark = null;

            if (this.marksDictionary.TryGetValue(VectorConstants.REMEDIATION_LEVEL, out
paramMark))
            {
                if (paramMark == MarksConctants.OFFICIAL_FIX)
                {
                    return ValuesOfMarks.REMEDIATION_OFFICIAL_FIX_VALUES;
                }

                if (paramMark == MarksConctants.TEMPORARY_FIX)
                {
                    return ValuesOfMarks.REMEDIATION_TEMPORARY_FIX_VALUES;
                }

                if (paramMark == MarksConctants.WORKAROUND)
                {
                    return ValuesOfMarks.REMEDIATION_WORKAROUND_VALUES;
                }

                if (paramMark == MarksConctants.UNAVAILABLE)
                {
```

```
                    return ValuesOfMarks.REMEDIATION_UNAVAILABLE_VALUES;
                }

                if (paramMark == MarksConctants.NOT_DEFINED)
                {
                    return ValuesOfMarks.REMEDIATION_NOT_DEFINED_VALUES;
                }
            }

            return ValuesOfMarks.REMEDIATION_NONE_VALUES;
        }

        private double GenereteReportScore()
        {
            string paramMark = null;

            if (this.marksDictionary.TryGetValue(VectorConstants.REPORT_CONFIDENCE, out
paramMark))
            {
                if (paramMark == MarksConctants.UNKNOWN)
                {
                    return ValuesOfMarks.REPORT_UNCONFIRMED_VALUES;
                }

                if (paramMark == MarksConctants.CONFIRMED)
                {
                    return ValuesOfMarks.REPORT_CONFIRMED_VALUES;
                }

                if (paramMark == MarksConctants.REASONABLE)
                {
                    return ValuesOfMarks.REPORT_UNCORROBORATED_VALUES;
                }

                if (paramMark == MarksConctants.NOT_DEFINED)
                {
                    return ValuesOfMarks.REPORT_NOT_DEFINED_VALUES;
                }
            }

            return ValuesOfMarks.REPORT_NONE_VALUES;
        }
    }
}
namespace NVDHunter.BussinessLogic
{
    public interface IVulnerabilitiesManager
    {
        Vulnerabilities Get(string product, Filter filter);

        VulnerabilityDetails GetById(string cve);

        VulnerabilityAssessment GetAssessment(string cve, string privilegesString);
    }
       }
namespace NVDHunter.BussinessLogic
{
   public class VulnerabilitiesManager : IVulnerabilitiesManager
    {
        private RestClient client;

        public VulnerabilitiesManager()
        {
            this.client = new RestClient();
```

```
        }
        public Vulnerabilities Get(string product, Filter filter)
        {
            this.client = new RestClient();
            this.client.BaseUrl = new Uri(UrlConstants.GET_ALL);

            var content = filter.GetType().GetProperties(BindingFlags.Instance |
BindingFlags.Public)
          .ToDictionary(prop => prop.Name, prop => prop.GetValue(filter, null));

            var response = this.Send(Method.GET, OtherConstants.LIST, content);

            var result = JsonConvert.DeserializeObject<Vulnerabilities>(response.Content);

            if (product != null)
            {
                var resultsByProduct = result.Results.Where(x =>
x.Description.Contains(product)).ToList();

                result.Results = resultsByProduct;
                result.TotalNumberOfResults = resultsByProduct.Count;
            }

            return result;
        }

        public VulnerabilityDetails GetById(string cve)
        {
            this.client.BaseUrl = new Uri(UrlConstants.GET_BY_ID_URL);

            var response = this.Send(Method.GET, cve + OtherConstants.JSON);

            if (response.StatusCode == HttpStatusCode.NotFound)
            {
                return null;
            }

            var result = this.MapToDetailsModel(response);

            return result;
        }

        public VulnerabilityAssessment GetAssessment(string cve, string privilegesString)
        {
            var vulnerabilityDetails = this.GetById(cve);

            var baseScore = new BaseScoreCounter(vulnerabilityDetails.VectorString);
            var temporalyScore = new
TemporalyScoreCounter(vulnerabilityDetails.VectorString);

            var baseScoreValue = baseScore.GetBaseScore(privilegesString);
            var temporaryScoreValue = temporalyScore.GetTemporalyScore(baseScoreValue);
            var assessment = AssessmentGenerator.GetAssessment(baseScoreValue);

            return new VulnerabilityAssessment()
            {
                CVE = vulnerabilityDetails.Guid,
                NVDBaseScore = vulnerabilityDetails.NVDBaseScore,
                BaseScore = baseScoreValue,
                TemporaryScore = temporaryScoreValue,
                Assessment = assessment
            };
        }
```

```
        private VulnerabilityDetails MapToDetailsModel(IRestResponse response)
        {
            var details = new VulnerabilityDetails();
            var jObject = JObject.Parse(response.Content);

            details.Description =
jObject.SelectToken("cve.description.description_data.[0].value").ToString();
            details.Guid = jObject.SelectToken("cve.CVE_data_meta.ID").ToString();

            details.Vendors =
jObject.SelectToken("cve.affects.vendor.vendor_data")?.Select(x => new Vendor()
            {
                VendorName = (string)x.SelectToken("vendor_name"),
                Products = x.SelectToken("product.product_data")?.Select(p => new Product()
                {
                    ProductName = (string)p.SelectToken("product_name"),
                    Versions = p.SelectToken("version.version_data")?.Select(v =>
(string)v.SelectToken("version_value")).ToList()
                }).ToList()
            }).ToList();

            details.VectorString =
(string)jObject.SelectToken("impact.baseMetricV3.cvssV3.vectorString") ??
(string)jObject.SelectToken("impact.baseMetricV2.cvssV2.vectorString");

            details.NVDBaseScore =
(double?)jObject.SelectToken("impact.baseMetricV3.cvssV3.baseScore") ??
(double?)jObject.SelectToken("impact.baseMetricV2.cvssV2.baseScore");
            return details;
        }

        private IRestResponse Send(Method method, string query, Dictionary<string, object>
content = null)
        {
            var httpRequest = new RestRequest(query, method);

            if (content != null)
            {
                httpRequest = this.AddRarametersToRequest(httpRequest, content);
            }

            var response = this.client.Execute(httpRequest);

            if (response.StatusCode == HttpStatusCode.Unauthorized)
            {
                this.client.AddDefaultHeader(Constants.AUTH, Constants.TOKEN);
                 response = this.Send(method, query, content);
            }

            return response;
        }

        private RestRequest AddRarametersToRequest(RestRequest request, Dictionary<string,
object> content)
        {
            foreach (var keyValue in content)
            {
                request.AddParameter(keyValue.Key, keyValue.Value);
            }

            return request;
        }

    namespace NVDHunter.Core.Constants.Metrics
```

```
{
   public static class MarksConctants
    {
        //ATTACK VECTOR MARKS
        public const string NETWORK = "N";
        public const string ADJACENT_NETWORK = "A";
        public const string LOCAL = "L";
        public const string PHYSICAL = "P";

        //COMMON MARKS
        public const string NONE = "N";
        public const string LOW = "L";
        public const string HIGH = "H";
        public const string MEDIUM = "M";
        public const string NOT_DEFINED = "X";

        //USER INTERACTION MARKS
        public const string REQUIRED = "R";

        //SCOPE MARKS
        public const string UNCHANGED = "U";
        public const string CHANGED = "C";

        //EXPLOITABILITY MARKS
        public const string EXPLOIT_EXIST = "U";
        public const string PROOF_OF_CONCEPT_CODE= "P";
        public const string FUNCTIONAL_EXPLOIT_EXIST = "F";

        //REMEDIATION LEVEL MARKS
        public const string OFFICIAL_FIX = "O";
        public const string TEMPORARY_FIX = "T";
        public const string WORKAROUND = "W";
        public const string UNAVAILABLE = "U";

        //REPORT CONFIDENCE MARKS
        public const string UNKNOWN = "U";
        public const string REASONABLE = "R";
        public const string CONFIRMED = "C";
    }
}
public static class ValuesOfMarks
    {
        //ATTACK VECTOR MARKS VALUES
        public const double AV_NONE_VALUES = 0.2;
        public const double AV_ADJACENT_NETWORK_VALUES = 0.9;
        public const double AV_LOCAL_VALUES = 1.0;
        public const double AV_NETWORK_VALUES = 0.8;
        public const double AV_PHYSICAL_VALUES = 1.5;

        //ATTACK COMPLEXITY MARKS VALUES
        public const double AC_LOW_VALUES = 1;
        public const double AC_HIGH_VALUES = 1.5;
        public const double AC_NONE_VALUES = 0.2;

        //PRIVILEGES REQUIRED MARKS VALUES
        public const double PR_LOW_VALUES = 1;
        public const double PR_HIGH_VALUES = 2;
        public const double PR_NONE_VALUES = 0.2;

        //USER INTERACTION MARKS VALUES
        public const double UI_REQUIRED_VALUES = 0.3;
        public const double UI_NONE_VALUES = 0.5;
        public const double UI_NOT_DEFINED_VALUES = 0.1;
```

```
        //SCOPE MARKS VALUES
        public const double S_UNCHANGED_VALUES = 0.7;
        public const double S_CHANGED_VALUES = 1;
        public const double S_NONE_VALUES = 0.2;

        //CONFIDENTIALITY IMPACT MARKS VALUES
        public const double CI_LOW_VALUES = 1.0;
        public const double CI_HIGH_VALUES = 1.5;
        public const double CI_NONE_VALUES = 0.4;
        public const double CI_NOT_DEFINED_VALUES = 0.1;

        //INTEGRITY IMPACT MARKS VALUES
        public const double I_LOW_VALUES = 1.0;
        public const double I_HIGH_VALUES = 1.5;
        public const double I_NONE_VALUES = 0.4;
        public const double I_NOT_DEFINED_VALUES = 0.1;

        //AVAILABILITY IMPACT MARKS VALUES
        public const double A_LOW_VALUES = 1.0;
        public const double A_HIGH_VALUES = 2.0;
        public const double A_NONE_VALUES = 0.4;
        public const double A_NOT_DEFINED_VALUES = 0.1;

        //EXPLOITABILITY MARKS VALUES
        public const double EXPLOITABILITY_UNPROVEN_VALUES = 0.85;
        public const double EXPLOITABILITY_PROOF_OF_CONCERT_VALUES = 0.9;
        public const double EXPLOITABILITY_FUNCTIONAL_VALUES = 0.95;
        public const double EXPLOITABILITY_HIGH_VALUES = 1.0;
        public const double EXPLOITABILITY_NOT_DEFINED_VALUES = 1.0;
        public const double EXPLOITABILITY_NONE_VALUES = 0.5;

        //REMEDIATION LEVEL MARKS VALUES
        public const double REMEDIATION_OFFICIAL_FIX_VALUES = 0.87;
        public const double REMEDIATION_TEMPORARY_FIX_VALUES = 0.9;
        public const double REMEDIATION_WORKAROUND_VALUES = 0.95;
        public const double REMEDIATION_UNAVAILABLE_VALUES = 1.0;
        public const double REMEDIATION_NOT_DEFINED_VALUES = 1.0;
        public const double REMEDIATION_NONE_VALUES = 0.5;

        //REPORT CONFIDENCE MARKS VALUES
        public const double REPORT_NOT_DEFINED_VALUES = 1.0;
        public const double REPORT_NONE_VALUES = 0.5;
        public const double REPORT_UNCONFIRMED_VALUES = 0.9;
        public const double REPORT_UNCORROBORATED_VALUES = 0.95;
        public const double REPORT_CONFIRMED_VALUES = 1.0;
    }
}
namespace NVDHunter.Core.Constants.Metrics
{
   public static class VectorConstants
    {
        //BASE METRICS
        public const string ATTACK_VECTOR = "AV";
        public const string ATTACK_COMPLEXITY = "AC";
        public const string PRIVILEGES_REQUIRED = "PR";
        public const string USER_INTERACTION = "UI";
        public const string SCOPE = "S";
        public const string CONFIDENTIALITY_IMPACT = "C";
        public const string INTEGRITY_IMPACT = "I";
        public const string AVAILABILITY_IMPACT = "A";

        //TEMPORAL METRICS
        public const string EXPLOITABILITY = "E";
        public const string REMEDIATION_LEVEL = "RL";
```

```
        public const string REPORT_CONFIDENCE = "RC";

        //ENVIRONMENTAL METRICS
        public const string CONFIDENTIALITY_REQUIREMENT = "CR";
        public const string INTEGRITY_REQUIREMENT = "IR";
        public const string AVAILABILITY_REQUIREMENT = "AR";
    }
}
namespace NVDHunter.Core.Constants
{
   public static class AssessmentsConstants
    {
        //Assessment names
        public const string LOW = "Low";

        public const string MEDIUM = "Medium";

        public const string HIGH = "High";

        public const string CRITICAL = "Critical";

        //Assessment values
        public const double MIN_LOW_VALUE = 0.1;

        public const double MAX_LOW_VALUE = 3.9;

        public const double MIN_MEDIUM_VALUE = 4.0;

        public const double MAX_MEDIUM_VALUE = 6.9;

        public const double MIN_HIGH_VALUE = 7.0;

        public const double MAX_HIGH_VALUE = 9.4;

        public const double MIN_CRITICAL_VALUE = 9.5;

        public const double MAX_CRITICAL_VALUE = 10.0;
    }
}
namespace NVDHunter.Core.Enums
{
   public enum PrivilegesEnum
    {
        Low,
        High
    }
}
namespace NVDHunter.Core.Models.ResponseOfDetails
{
   public class Product
    {
        public string ProductName { get; set; }

        public List<string> Versions { get; set; } = new List<string>();
    }
}namespace NVDHunter.Core.Models.ResponseOfDetails
{
   public class Vendor
    {
        public string VendorName { get; set; }

        public List<Product> Products { get; set; } = new List<Product>();
    }
}namespace NVDHunter.Core.Models
```

```
{
    public class BaseVulnerabilityData
    {
        public string Guid { get; set; }

        public string Description { get; set; }
    }
}
namespace NVDHunter.Core.Models
{
    public class Filter
    {
        public string startDate { get; set; }

        public string endDate { get; set; }

        public int pageNumber { get; set; } = 1;

        public int resultsPerPage { get; set; } = 20;
    }
}
namespace NVDHunter.Core.Models
{
    public class VulnerabilityAssessment
    {
        public string CVE { get; set; }

        public double? NVDBaseScore { get; set; }

        public double? BaseScore { get; set; }

        public double? TemporaryScore { get; set; }

        public string Assessment { get; set; }
    }
}
```

№ док.	Позначення	Найменування	Дод. відом.
1		Текстові документи	
	ГЮІК.ХХХХХ1649Ст.15ПЗ	Пояснювальна записка	88 с.
2		Графічні документи	
		Презентаційний матеріал	20 А4.
3		Інші документи	
		Електронна версія ПЗ	1
		Відзив керівника	1 с.
		Рецензія	1 с.

Изм.	Лист	№ докум.	Підпис	Дата	ГЮІК. ХХХХХ1649Ст.15ВД		
Розроб.		Ревізорова К.І..			Критичні вразливості захисту інформації в ОС	Лит.	Лист 1 / Листів 1
Перевір.		Гріненко Т.О.				Т	
Н.контр.		Конєва Н.Ф.				ХНУРЕ Кафедра БІТ	
Затверд.		Халімов Г.З.					