C#

C# static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your C# code

LDAP queries should not be vulnerable to injection attacks

Analyze your code

Vulnerability   Blocker   injection cwe owasp

User-provided data such as URL parameters should always be considered as untrusted and tainted. Constructing LDAP names or search filters directly from tainted data enables attackers to inject specially crafted values that changes the initial meaning of the name or filter itself. Successful LDAP injections attacks can read, modify or delete sensitive information from the directory service.

Within LDAP names, the special characters `' '`, `'#'`, `'"'`, `'+'`, `','`, `';'`, `'<'`, `'>'`, `'\'` and `null` must be escaped according to RFC 4514, for example by replacing them with the backslash character `'\'` followed by the two hex digits corresponding to the ASCII code of the character to be escaped. Similarly, LDAP search filters must escape a different set of special characters (including but not limited to `'*'`, `'('`, `')'`, `'\'` and `null`) according to RFC 4515.

Noncompliant Code Example

```
using System.DirectoryServices;
using Microsoft.AspNetCore.Mvc;

namespace WebApplicationDotNetCore.Controllers
{
    public class RSPEC2078LDAPInjectionNoncompliantControlle
    {
        public IActionResult Index()
        {
            return View();
        }

        public DirectorySearcher ds { get; set; }

        public IActionResult Authenticate(string user, strin
        {
            ds.Filter = "(&(uid=" + user + ")(userPassword="

            // If the special value "*)(uid=*))(|(uid=*" is
            // Indeed, if it is passed as a user, the filter
            // (&(uid=*)(uid=*))(|(uid=*)(userPassword=...))
            // as uid=* match all users, it is equivalent to
            // (|(uid=*)(userPassword=...))
            // again, as uid=* match all users, the filter b

            return Content(ds.FindOne() != null ? "success"
        }
    }
}
```

Compliant Solution

```
using System.DirectoryServices;
using System.Text.RegularExpressions;
```

Public methods should not have multidimensional array parameters

Code Smell

"async" and "await" should not be used as identifiers

Code Smell

TestCases should contain tests

Code Smell

Short-circuit logic should be used in boolean contexts

Code Smell

JWT should be signed and verified

```
using Microsoft.AspNetCore.Mvc;

namespace WebApplicationDotNetCore.Controllers
{
    public class RSPEC2078LDAPInjectionCompliantController :
    {
        public IActionResult Index()
        {
            return View();
        }

        public DirectorySearcher ds { get; set; }

        public IActionResult Authenticate(string user, strin
        {
            // restrict the username and password to letters
            if (!Regex.IsMatch(user, "^[a-zA-Z]+$") || !Rege
            {
                return BadRequest();
            }

            ds.Filter = "(&(uid=" + user + ")(userPassword="
            return Content(ds.FindOne() != null ? "success"
        }
    }
}
```

See

- OWASP Top 10 2021 Category A3 - Injection
- OWASP Top 10 2017 Category A1 - Injection
- RFC 4514 - LDAP: String Representation of Distinguished Names
- RFC 4515 - LDAP: String Representation of Search Filters
- MITRE, CWE-20 - Improper Input Validation
- MITRE, CWE-90 - Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')

Available In:

sonarcloud | sonarqube Developer Edition