

Forum:[Hack Forums](#)

Thread: [\[Weekend code\] Protocol Detector](#)

itcorp - September 19, 2025, 5:54 pm

Here is my weekend code for community from Quantum and Lengeds...Its a simple protocol detector which is able to detects standard and industry protocols...I hope you find a purpose for it and benefit from this contribution but I also look forward to your responses for potential future improvements.

```
Code    public class PortProbeService : IPortProbeService, IDisposable
    {
        private readonly ILogger<PortProbeService> _logger;
        private readonly IDeviceAnalysisConfigProvider _configProvider;

        private readonly ConcurrentDictionary<int, List<PortProbeConfig>> _portProbes;

        private readonly ConcurrentDictionary<string, List<PortProbeConfig>> _protocolProbes;

        private readonly Timer _configRefreshTimer;
        private readonly object _refreshLock = new();
        private bool _disposed = false;
        private bool _isRefreshing = false;

        public IReadOnlyDictionary<int, PortProbeConfig> PortProbes
        {
            get
            {
                var dict = new Dictionary<int, PortProbeConfig>();
                foreach (var kvp in _portProbes)
                {
                    if (kvp.Value.Count > 0)
                    {
                        dict[kvp.Key] = kvp.Value[0];
                    }
                }
                return dict;
            }
        }

        public PortProbeService(
            ILogger<PortProbeService> logger,
            IDeviceAnalysisConfigProvider configProvider)
        {
            _logger = logger ?? throw new ArgumentNullException(nameof(logger));
            _configProvider = configProvider ?? throw new
ArgumentNullException(nameof(configProvider));

            _portProbes = new ConcurrentDictionary<int, List<PortProbeConfig>>();
            _protocolProbes = new ConcurrentDictionary<string,
```

```
List<PortProbeConfig>>(StringComparer.OrdinalIgnoreCase);
```

```
RefreshProbesSync();
```

```
_configRefreshTimer = new Timer(  
    _ => RefreshProbesAsync().ConfigureAwait(false).GetAwaiter().GetResult(),  
    null,  
    TimeSpan.FromMinutes(5),  
    TimeSpan.FromMinutes(5));
```

```
}
```

```
public PortProbeConfig? GetProbeForPort(int port)
```

```
{  
    if (_portProbes.TryGetValue(port, out var probes) && probes.Count > 0)  
    {  
        return probes[0];  
    }  
    return null;  
}
```

```
public PortProbeConfig? GetProbeForProtocol(string protocol)
```

```
{  
    if (string.IsNullOrEmpty(protocol))  
    {  
        throw new ArgumentException("Protocol cannot be null or empty", nameof(protocol));  
    }  
  
    if (_protocolProbes.TryGetValue(protocol, out var probes) && probes.Count > 0)  
    {  
        return probes[0];  
    }  
    return null;  
}
```

```
public IReadOnlyList<PortProbeConfig> GetProbesForProtocol(string protocol)
```

```
{  
    if (string.IsNullOrEmpty(protocol))  
    {  
        throw new ArgumentException("Protocol cannot be null or empty", nameof(protocol));  
    }  
  
    if (_protocolProbes.TryGetValue(protocol, out var probes))  
    {  
        return probes.AsReadOnly();  
    }  
    return Array.Empty<PortProbeConfig>();  
}
```

```
public IEnumerable<PortProbeConfig> GetAllProbes()
```

```
{  
    return _portProbes.Values  
        .SelectMany(probes => probes)  
        .Distinct()
```

```

        .ToList();
    }

    public IEnumerable<string> GetAllProtocols()
    {
        return _protocolProbes.Keys;
    }

    public IEnumerable<PortProbeConfig> GetPotentialProbes(int port)
    {
        var portSpecific = _portProbes.TryGetValue(port, out var probes)
            ? probes
            : Enumerable.Empty<PortProbeConfig>();

        var genericProbes = _portProbes
            .Where(kvp => kvp.Key != port)
            .SelectMany(kvp => kvp.Value.Where(p => p.IsGeneric));

        return portSpecific
            .Concat(genericProbes)
            .OrderBy(p => p.Priority)
            .ThenBy(p => p.Protocol.Contains("HTTP/1.0") ? 1 : 0)
            .ThenBy(p => p.Protocol);
    }

    private async Task RefreshProbesAsync()
    {
        if (_isRefreshing) return;

        lock (_refreshLock)
        {
            if (_isRefreshing) return;
            _isRefreshing = true;
        }

        try
        {
            await Task.Run(() => RefreshProbesSync());
        }
        finally
        {
            _isRefreshing = false;
        }
    }

    private void RefreshProbesSync()
    {
        try
        {
            var newProbes = LoadPortProbes();
            var changesDetected = false;

            foreach (var probe in newProbes)

```

```

{
    if (!_portProbes.ContainsKey(probe.Key) ||
        !_portProbes[probe.Key].Contains(probe.Value))
    {
        _portProbes.AddOrUpdate(probe.Key,
            new List<PortProbeConfig> { probe.Value },
            (_, existingList) =>
            {
                if (!existingList.Contains(probe.Value))
                {
                    existingList.Add(probe.Value);
                    changesDetected = true;
                }
                return existingList;
            });

        UpdateProtocolProbe(probe.Value);
        changesDetected = true;
    }
}

foreach (var port in _portProbes.Keys.ToList())
{
    if (!newProbes.ContainsKey(port))
    {
        if (_portProbes.TryRemove(port, out var removedProbes))
        {
            foreach (var probe in removedProbes)
            {
                RemoveProtocolProbe(probe);
            }
            changesDetected = true;
        }
    }
}

if (changesDetected)
{
    _logger.LogInformation("Port probes configuration refreshed. Total probes: {Count}",
        _portProbes.Count);
}
}
catch (Exception ex)
{
    _logger.LogError(ex, "Failed to refresh port probes configuration");
}
}

private void UpdateProtocolProbe(PortProbeConfig probe)
{
    if (string.IsNullOrWhiteSpace(probe.Protocol))
    {
        return;
    }
}

```

```

    }

    _protocolProbes.AddOrUpdate(
        probe.Protocol,
        _ => new List<PortProbeConfig> { probe },
        (_, existingList) =>
        {
            if (!existingList.Any(p => p.Port == probe.Port))
            {
                existingList.Add(probe);
            }
            return existingList;
        });
}

private void RemoveProtocolProbe(PortProbeConfig probe)
{
    if (string.IsNullOrEmpty(probe.Protocol))
    {
        return;
    }

    if (_protocolProbes.TryGetValue(probe.Protocol, out var probes))
    {
        probes.RemoveAll(p => p.Port == probe.Port);
        if (probes.Count == 0)
        {
            _protocolProbes.TryRemove(probe.Protocol, out _);
        }
    }
}

private Dictionary<int, PortProbeConfig> LoadPortProbes()
{
    var probes = new Dictionary<int, PortProbeConfig>();

    if (_configProvider?.AnalysisConfig?.PortProbes == null)
    {
        _logger.LogError("Port probes configuration is missing");
        return probes;
    }

    foreach (var probeConfig in _configProvider.AnalysisConfig.PortProbes)
    {
        try
        {
            if (probeConfig == null || probeConfig.Port <= 0 || probeConfig.Port > 65535)
            {
                _logger.LogWarning("Invalid port probe configuration: {Port}", probeConfig?.Port);
                continue;
            }

            if (string.IsNullOrEmpty(probeConfig.Protocol))

```

```

        {
            _logger.LogWarning("Missing protocol for port {Port}", probeConfig.Port);
            continue;
        }

        probes[probeConfig.Port] = probeConfig;
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Failed to parse probe data for port {Port}", probeConfig?.Port);
    }
}

return probes;
}

public void Dispose()
{
    if (_disposed) return;
    _configRefreshTimer?.Dispose();
    _disposed = true;
    GC.SuppressFinalize(this);
}
}

Code public class ProtocolDetector : IProtocolDetector
{
    private readonly ILogger<ProtocolDetector> _logger;
    private readonly IPortProbeService _portProbeService;
    private readonly IIndustrialProtocolDetector _industrialDetector;
    private readonly IStandardProtocolDetector _standardDetector;

    public ProtocolDetector(
        ILogger<ProtocolDetector> logger,
        IPortProbeService portProbeService,
        IIndustrialProtocolDetector industrialDetector,
        IStandardProtocolDetector standardDetector)
    {
        _logger = logger;
        _portProbeService = portProbeService;
        _industrialDetector = industrialDetector;
        _standardDetector = standardDetector;
    }

    public async Task<string?> DetectProtocolAsync(string ip, int port, string? banner,
        List<HttpHeader?> headers, CancellationToken ct)
    {
        try
        {
            _logger.LogInformation("Starting protocol detection for {IP}:{Port}", ip, port);

            var potentialProbes = _portProbeService.GetPotentialProbes(port).ToList();

```

```

var industrialProbes = potentialProbes
    .Where(p => _industrialDetector.IsProtocol(p.Protocol))
    .ToList();

var standardProbes = potentialProbes
    .Where(p => _standardDetector.IsProtocol(p.Protocol))
    .ToList();

if (industrialProbes.Any())
{
    var industrialProtocol = await _industrialDetector.DetectFromProbesAsync(
        ip, port, industrialProbes, banner, ct);
    if (industrialProtocol != null) return industrialProtocol;
}

if (standardProbes.Any())
{
    var standardProtocol = await _standardDetector.DetectFromProbesAsync(
        ip, port, standardProbes, banner, ct);
    if (standardProtocol != null) return standardProtocol;
}

var portProbe = _portProbeService.GetProbeForPort(port);
if (portProbe != null)
{
    IProtocolDetectorBase detector = _industrialDetector.IsProtocol(portProbe.Protocol)
        ? _industrialDetector
        : _standardDetector;

    if (await detector.VerifyProtocolAsync(ip, port, portProbe.Protocol, ct))
    {
        return portProbe.Protocol;
    }
}

return "Unknown";
}
catch (Exception ex)
{
    _logger.LogError(ex, "Protocol detection failed for {IP}:{Port}", ip, port);
    return null;
}
}

public async Task<bool> VerifyProtocolAsync(string ip, int port, string protocol,
CancellationTokens ct)
{
    try
    {
        IProtocolDetectorBase detector = _industrialDetector.IsProtocol(protocol)
            ? _industrialDetector
            : _standardDetector;
    }
}

```

```

        return await detector.VerifyProtocolAsync(ip, port, protocol, ct);
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Protocol verification failed for {Protocol} on {IP}:{Port}", protocol, ip,
port);
        return false;
    }
}
}

```

Code public class StandardProtocolDetector : IStandardProtocolDetector

```

{
    private readonly ILogger<StandardProtocolDetector> _logger;
    private readonly IPortProbeService _portProbeService;
    private readonly IIndustrialProtocolDetector _industrialDetector;
    public StandardProtocolDetector(
        ILogger<StandardProtocolDetector> logger,
        IPortProbeService portProbeService,
        IIndustrialProtocolDetector industrialDetector)
    {
        _logger = logger;
        _portProbeService = portProbeService;
        _industrialDetector = industrialDetector;
    }

    public bool IsProtocol(string protocol)
    {
        return !_industrialDetector.IsProtocol(protocol);
    }
    private bool IsHttp2Preface(byte[] response, int length)
    {
        byte[] http2Preface = Encoding.ASCII.GetBytes("PRI * HTTP/2.0\r\n\r\nSM\r\n\r\n");

        if (length < http2Preface.Length)
            return false;

        for (int i = 0; i < http2Preface.Length; i++)
        {
            if (response[i] != http2Preface[i])
                return false;
        }
        return true;
    }
    private bool IsHttpResponse(byte[] response, int length, out string? version)
    {
        version = null;
        if (length < 8) return false;

        try
        {
            var responseText = Encoding.ASCII.GetString(response, 0, Math.Min(length, 128));
            if (responseText.StartsWith("HTTP/"))
            {

```

```

        var firstSpace = responseText.IndexOf(' ');
        if (firstSpace > 0)
        {
            version = responseText.Substring(5, firstSpace - 5).Trim();
        }
        return true;
    }

    var statusCodes = new[] { " 200 ", " 301 ", " 302 ", " 401 ", " 403 ", " 404 ", " 500 " };
    return statusCodes.Any(code => responseText.Contains(code));
}
catch
{
    return false;
}
}

public async Task<string?> DetectFromProbesAsync(string ip, int port,
IEnumerable<PortProbeConfig> probes, string? banner, CancellationToken ct)
{
    try
    {
        _logger.LogDebug("Detecting standard protocols from probes for {IP}:{Port}", ip, port);

        foreach (var probe in probes.Where(p => IsProtocol(p.Protocol)))
        {
            if (ct.IsCancellationRequested)
                break;

            if (await VerifyProtocolWithProbeAsync(ip, port, probe, ct))
            {
                return probe.Protocol;
            }
        }

        if (!string.IsNullOrEmpty(banner))
        {
            return await DetectByBannerAsync(ip, port, banner, ct);
        }

        return null;
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Standard protocol detection from probes failed for {IP}:{Port}", ip,
port);
        return null;
    }
}

public async Task<string?> DetectStandardProtocolAsync(string ip, int port, string? banner,
CancellationToken ct)
{
    try
    {

```

```

_logger.LogDebug("Starting standard protocol detection for {IP}:{Port}", ip, port);

var probes = _portProbeService.GetPotentialProbes(port)
    .Where(p => IsProtocol(p.Protocol))
    .ToList();

var result = await DetectFromProbesAsync(ip, port, probes, banner, ct);
if (result != null) return result;

var portProbe = _portProbeService.GetProbeForPort(port);
if (portProbe != null &&
    IsProtocol(portProbe.Protocol) &&
    await VerifyProtocolWithProbeAsync(ip, port, portProbe, ct))
{
    return portProbe.Protocol;
}

return null;
}
catch (Exception ex)
{
    _logger.LogError(ex, "Standard protocol detection failed for {IP}:{Port}", ip, port);
    return null;
}
}

public async Task<bool> VerifyProtocolAsync(string ip, int port, string protocol,
CancellationTokens ct)
{
    if (!IsProtocol(protocol)) return false;

    var probe = _portProbeService.GetProbeForProtocol(protocol);
    if (probe == null) return false;

    return await VerifyProtocolWithProbeAsync(ip, port, probe, ct);
}

private async Task<string?> DetectByBannerAsync(string ip, int port, string banner,
CancellationTokens ct)
{
    foreach (var probe in _portProbeService.GetAllProbes())
    {
        if (banner.Contains(probe.Protocol, StringComparison.OrdinalIgnoreCase) &&
            await VerifyProtocolWithProbeAsync(ip, port, probe, ct))
        {
            return probe.Protocol;
        }
    }
    return null;
}

private async Task<bool> VerifyProtocolWithProbeAsync(
    string ip,
    int port,
    PortProbeConfig probe,
    CancellationTokens ct)

```

```

{
    try
    {
        if (string.IsNullOrEmpty(probe.ProbeData)) return false;

        if (IsUdpProtocol(probe.Protocol))
        {
            return await VerifyUdpProtocolAsync(ip, port, probe, ct);
        }
        return await VerifyTcpProtocolAsync(ip, port, probe, ct);
    }
    catch (Exception ex)
    {
        _logger.LogDebug(ex, "Probe verification failed for {Protocol} on {IP}:{Port}",
            probe.Protocol, ip, port);
        return false;
    }
}

```

```

private async Task<bool> VerifyTcpProtocolAsync(
    string ip,
    int port,
    PortProbeConfig probe,
    CancellationToken ct)
{
    try
    {
        using var client = new TcpClient();
        await client.ConnectAsync(ip, port, ct);

        client.SendTimeout = 2000;
        client.ReceiveTimeout = 2000;

        Stream stream = client.GetStream();

        bool isSsl = port == 443 || probe.Protocol.ToUpper().Contains("HTTPS") ||
            probe.Protocol.ToUpper().Contains("SSL");

        if (isSsl)
        {
            try
            {
                var sslStream = new SslStream(stream, false,
                    (sender, certificate, chain, errors) => true);

                await sslStream.AuthenticateAsClientAsync(ip, null,
                    SslProtocols.Tls12 | SslProtocols.Tls11 | SslProtocols.Tls,
                    false);

                stream = sslStream;
            }
            catch
            {
            }
        }
    }
}

```

```

        isSsl = false;
    }
}

byte[] probeData = GetProbeData(probe);
await stream.WriteAsync(probeData, ct);

var buffer = new byte[4096];
int bytesRead;

using (var cts = CancellationTokenSource.CreateLinkedTokenSource(ct))
{
    cts.CancelAfter(TimeSpan.FromSeconds(2));
    bytesRead = await stream.ReadAsync(buffer, cts.Token);
}

if (bytesRead == 0) return false;

if (probe.Protocol.StartsWith("HTTP", StringComparison.OrdinalIgnoreCase))
{
    var responseText = Encoding.ASCII.GetString(buffer, 0, bytesRead);
    if (responseText.Contains(" 401 ") || responseText.Contains("WWW-Authenticate:"))
    {
        return true;
    }
}

return VerifyResponse(probe.Protocol, buffer, bytesRead, isSsl);
}
catch (Exception ex)
{
    _logger.LogDebug(ex, "TCP verification failed for {Protocol} on {IP}:{Port}",
        probe.Protocol, ip, port);
    return false;
}
}

private async Task<bool> VerifyUdpProtocolAsync(
    string ip,
    int port,
    PortProbeConfig probe,
    CancellationToken ct)
{
    using var client = new UdpClient();
    byte[] probeData = GetProbeData(probe);

    await client.SendAsync(probeData, probeData.Length, ip, port);
    var response = await client.ReceiveAsync(ct);

    return response.Buffer.Length > 0 &&
        VerifyResponse(probe.Protocol, response.Buffer, response.Buffer.Length);
}

private byte[] GetProbeData(PortProbeConfig probe)

```

```

{
    return probe.Encoding switch
    {
        "ASCII" => Encoding.ASCII.GetBytes(probe.ProbeData),
        "Hex" => HexStringToByteArray(probe.ProbeData),
        "Base64" => Convert.FromBase64String(probe.ProbeData),
        _ => throw new NotSupportedException($"Encoding {probe.Encoding} not supported")
    };
}
private bool VerifyResponse(string protocol, byte[] response, int length, bool isSsl = false)
{
    var protocolUpper = protocol.ToUpper();
    var responseText = Encoding.ASCII.GetString(response, 0, Math.Min(length, 1024));
    var responseUpper = responseText.ToUpper();
    var hexResponse = BitConverter.ToString(response, 0, Math.Min(length, 64)).Replace("-",
    "");

    bool isHttpResponse = IsHttpResponse(response, length, out var httpVersion);

    bool isHttp2 = IsHttp2Preface(response, length) ||

    hexResponse.Contains("505249202A20485454502F322E300D0A0D0A534D0D0A0D0A");
    bool isHttp3 = hexResponse.Contains("00001D00000000100");

    bool isHttpStatusCode = responseUpper.Contains("HTTP/") ||
        Regex.IsMatch(responseUpper, @"\s\d{3}\s");

    switch (protocolUpper)
    {
        case "FTP":
            return responseUpper.Contains("220") ||
                responseUpper.Contains("FTP") ||
                responseUpper.StartsWith("220-") ||
                responseUpper.Contains("READY") ||
                (length >= 3 && response[0] == '2' && response[1] == '2' && response[2] == '0');
        case "SSH":
            return responseUpper.Contains("SSH-") ||
                (length >= 3 && response[0] == 'S' && response[1] == 'S' && response[2] == 'H') ||
                hexResponse.StartsWith("5353482D");
        case "TELNET":
            return responseUpper.Contains("LOGIN:") ||
                responseText.Contains("Username:") ||
                responseText.Contains("Password:") ||
                responseUpper.Contains("TELNET") ||
                responseUpper.Contains("LOGIN PROMPT") ||
                (length > 0 && (response[0] == 0xFF || response[0] == 0xFB));
        case "SMTP":
            return (responseUpper.Contains("220") && responseUpper.Contains("SMTP")) ||
                responseUpper.Contains("ESMTP") ||
                responseUpper.Contains("MAIL SERVICE") ||
                responseUpper.Contains("MAIL SERVER") ||
                (length >= 3 && response[0] == '2' && response[1] == '2' && response[2] ==

```

'0');

```
case "DNS":
    return hexResponse.Contains("8180") ||
        hexResponse.Contains("8400") ||
        hexResponse.Contains("8000") ||
        (length >= 4 && response[0] == 0x00 && response[1] == 0x00) ||
        hexResponse.Contains("000F0001");
case "DHCP":
    return hexResponse.Contains("0201") ||
        hexResponse.Contains("63825363") ||
        (length >= 240 && response[0] == 0x02);
case "TFTP":
    return (response[0] == 0x00 && response[1] == 0x03) ||
        (response[0] == 0x00 && response[1] == 0x05) ||
        (response[0] == 0x00 && response[1] == 0x04);
case "HTTP":
    return isHttpResponse || isHttp2 || isHttp3 || isHttpStatusCodes;
case "HTTP/1.0":
    return (httpVersion == "1.0") ||
        (isHttpResponse && !isHttp2 && !isHttp3 && responseUpper.Contains("HTTP/1.0"));
case "HTTP/1.1":
    return (httpVersion == "1.1") ||
        (isHttpResponse && !isHttp2 && !isHttp3 &&
            (responseUpper.Contains("HTTP/1.1") || httpVersion == null));

case "HTTP/2":
    return isHttp2 || (httpVersion == "2.0") ||
        responseUpper.Contains("HTTP/2.0");
case "HTTP/3":
    return isHttp3 || (httpVersion == "3.0") ||
        responseUpper.Contains("HTTP/3.0");
case "HTTPS":
case "HTTPS-ALT":
case "WOWZA HTTP":
case "WOWZA HTTPS":
case "TR-069":
case "WD-NAS":
case "SYNOLOGY-MANAGER":
    return isHttpResponse || isHttp2 || isHttp3 || isHttpStatusCodes || isSsl;
case "POP3":
    return responseUpper.Contains("+OK") ||
        responseUpper.Contains("POP3") ||
        responseUpper.Contains("MAIL SERVER") ||
        (length >= 3 && response[0] == '+' && response[1] == 'O' && response[2] == 'K');
case "IMAP":
    return responseUpper.Contains("* OK") ||
        responseUpper.Contains("IMAP") ||
        (length >= 4 && response[0] == '*' && response[1] == ' ' &&
            response[2] == 'O' && response[3] == 'K');
case "SNMP":
    return length > 20 &&
        response[16] == 0x02 &&
        response[17] == 0x04 &&
```

```

        response[18] == 0x00 &&
        response[19] == 0x00 &&
        response[20] == 0x00 &&
        response[21] == 0x01;
case "BGP":
    return length >= 19 && response[18] == 0x02;
case "RADIUS":
    return length > 0 && (response[0] == 0x02 || response[0] == 0x03);
case "REDIS":
    return length >= 7 && response[0] == 0x2B && response[1] == 0x50 && response[2]
== 0x4F;
case "MYSQL":
    return length >= 4 && response[0] == 0x0A;
case "POSTGRESQL":
    return length >= 1 && response[0] == 0x52;
case "AMQP":
    return length >= 8 && response[0] == 0x41 && response[1] == 0x4D && response[2]
== 0x51;
case "MQTT":
case "MQTT-SSL":
    return length >= 4 && (response[0] == 0x20 || response[0] == 0x10);
case "SIP":
case "SIPS":
    return responseText.Contains("SIP/2.0") &&
        (responseUpper.Contains("200 OK") ||
        responseUpper.Contains("401 UNAUTHORIZED") ||
        responseUpper.Contains("SIP SERVICE"));
case "RTSP":
    return responseUpper.Contains("RTSP/1.0");
case "RDP":
    return length >= 19 && response[0] == 0x03;
case "VNC":
    return responseUpper.Contains("RFB ");
case "IPSEC":
    return length > 0 && response[0] == 0x00;
case "LPD":
    return length > 0 && response[0] == 0x00;
case "IPP":
    return length >= 8 && response[0] == 0x01;
case "PPTP":
    return length >= 156 && response[1] == 0x02;
case "RTMP":
    return length > 0 && response[0] == 0x03;
case "DICOM":
    return length >= 6 && response[0] == 0x02;
case "WINBOX":
    return length >= 8 && response[0] == 0x00 && response[1] == 0x00;
case "MIKROTIK-API":
case "MIKROTIK-API-SSL":
    return responseUpper.Contains("!DONE") || responseUpper.Contains("!TRAP");
case "PJL":
    return responseUpper.Contains("@PJL") || responseUpper.Contains("INFO ID");
case "NVC STREAMING":

```

```

        return responseUpper.Contains("NVC STREAMING") ||
responseUpper.Contains("VIDEO/MP4");
    case "NVC PRIVATE API":
        return responseUpper.Contains("NVC API") ||
responseUpper.Contains("APPLICATION/JSON");
    case "UBIQUITI-INFORM":
        return length > 0 && response[0] == 0x01;
    case "STEAM":
        return length > 0 && response[0] == 0xFF && response[1] == 0xFF;
    case "TACACS+":
        return length >= 4 && response[0] == 0x01 && response[1] == 0x01;
    case "WIREGUARD":
        return length >= 4 && response[0] == 0x01 && response[1] == 0x00;
    case "NTP":
        return length > 0 && response[0] == 0x23;
    case "RPC":
        return length > 0 && response[0] == 0x05;
    case "NETBIOS":
        return length > 0 && response[0] == 0x80;
    case "ONVIF":
        return responseUpper.Contains("ONVIF") ||
            responseUpper.Contains("DISCOVER") ||
            responseUpper.Contains("WS-DISCOVERY") ||
            responseUpper.Contains("SCHEMAS-ONVIF-ORG");
    case "HIKVISION":
    case "HIKVISION-API":
        return responseUpper.Contains("HIKVISION") ||
            responseUpper.Contains("ISAPI") ||
            responseUpper.Contains("HIK/1.0");
    case "DAHUA":
    case "DAHUA-PRIVATE":
        return responseUpper.Contains("DAHUA") || hexResponse.Contains("377777");
    case "AXIS-API":
        return responseUpper.Contains("AXIS") || responseUpper.Contains("CGI");
    case "AXIS-VAPIX":
    case "AXIS-VAPIX-SSL":
        return responseUpper.Contains("AXIS") ||
            responseUpper.Contains("VAPIX") ||
            responseUpper.Contains("CGI") ||
            responseUpper.Contains("PARAM.CGI");
    case "GV-PROTOCOL":
        return length > 0 && (response[0] == 0x00 || response[0] == 0x01);
    case "PELCO-D":
        return length >= 7 && response[0] == 0xFF && response[1] == 0x00;
    case "PELCO-P":
        return length >= 7 && response[0] == 0xA0 && response[1] == 0x00;
    case "PSIA":
        return responseUpper.Contains("PSIA") ||
            responseUpper.Contains("XML") ||
            responseUpper.Contains("MEDIA/WSDL");
    case "BOSCH-RCP+":
        return length > 0 && response[0] == 0x00 && response[1] == 0x00;
    case "SAMSUNG-TECHWIN":

```

```

        return responseUpper.Contains("SAMSUNG") ||
            responseUpper.Contains("TECHWIN") ||
            responseUpper.Contains("SNC") ||
            responseUpper.Contains("WISENET");
    case "COAXITRON":
        return length > 0 && response[0] == 0x01;
    case "CAMERAWEBINTERFACES":
        return responseUpper.Contains("CAMERA") ||
            responseUpper.Contains("WEBCAM") ||
            responseUpper.Contains("VIDEO") ||
            responseUpper.Contains("STREAM") ||
            responseUpper.Contains("VIEWER") ||
            responseUpper.Contains("WEBVIEW") ||
            responseUpper.Contains("WEBINTERFACE") ||
            responseUpper.Contains("WEBACCESS") ||
            (responseUpper.Contains("HTTP") &&
            (responseUpper.Contains("LOGIN") ||
            responseUpper.Contains("LIVE") ||
            responseUpper.Contains("VIDEO") ||
            responseUpper.Contains("STREAM")));
    case "REOLINK-API":
        return responseUpper.Contains("REOLINK") || responseUpper.Contains("CMD");
    case "AMCREST-API":
        return responseUpper.Contains("AMCREST") ||
responseUpper.Contains("MAGICBOX");
    default:
        return responseUpper.Contains(protocolUpper) ||
            hexResponse.Contains(protocolUpper);
    }
}
private bool IsUdpProtocol(string protocol)
{
    var udpProtocols = new HashSet<string>(StringComparer.OrdinalIgnoreCase)
    {
        "DNS", "DHCP", "TFTP", "SNMP", "RADIUS", "NTP"
    };
    return udpProtocols.Contains(protocol);
}
private static byte[] HexStringToByteArray(string hex)
{
    hex = hex.Replace("\\x", "").Replace(" ", "");
    return Enumerable.Range(0, hex.Length)
        .Where(x => x % 2 == 0)
        .Select(x => Convert.ToByte(hex.Substring(x, 2), 16))
        .ToArray();
}
}
}

```

```

Code public class IndustrialProtocolDetector : IIndustrialProtocolDetector
{
    private readonly ILogger<IndustrialProtocolDetector> _logger;
    private readonly IPortProbeService _portProbeService;

    public IndustrialProtocolDetector(

```

```

ILogger<IndustrialProtocolDetector> logger,
IProbeService portProbeService)
{
    _logger = logger;
    _portProbeService = portProbeService;
}

public bool IsProtocol(string protocol)
{
    var industrialProtocols = new HashSet<string>(StringComparer.OrdinalIgnoreCase)
    {
        "MODBUS", "S7COMM", "PROFINET", "ETHERCAT", "BACNET", "DNP3",
        "IEC 61850", "IEC 60870-5-104", "OPC UA", "CANOPEN", "DEVICENET",
        "CONTROLNET", "ETHERNET/IP", "CC-LINK", "PROFIBUS", "FOUNDATION FIELDBUS",
        "HART PROTOCOL", "WIRELESSHART", "LONWORKS", "KNX", "M-BUS", "DALI",
        "IEC 60870-5-101", "IEC 60870-5-103", "MTCONNECT", "ODVA CIP",
        "ISO 9506 (MMS)", "POWERLINK", "SERCOS III", "AS-I", "IO-LINK",
        "SAFETYBUS P", "PROFISAFE", "CIP SAFETY", "ETHERCAT SAFETY",
        "MODBUS SECURE", "ENOCAN", "ZIGBEE", "Z-WAVE", "DMX512",
        "TCN (TRAIN COMMUNICATION NETWORK)"
    };
    return industrialProtocols.Contains(protocol);
}

public async Task<string?> DetectFromProbesAsync(string ip, int port,
IEnumerable<PortProbeConfig> probes, string? banner, CancellationToken ct)
{
    try
    {
        _logger.LogDebug("Detecting industrial protocols from probes for {IP}:{Port}", ip, port);

        foreach (var probe in probes)
        {
            if (ct.IsCancellationRequested)
                break;

            if (await VerifyProtocolWithProbeAsync(ip, port, probe, ct))
            {
                return probe.Protocol;
            }
        }

        if (!string.IsNullOrEmpty(banner))
        {
            return await DetectByBannerAsync(ip, port, banner, ct);
        }

        return null;
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Industrial protocol detection from probes failed for {IP}:{Port}", ip,
port);
    }
}

```

```

        return null;
    }
}

public async Task<string?> DetectIndustrialProtocolAsync(string ip, int port, string? banner,
Cancellation token ct)
{
    try
    {
        _logger.LogDebug("Starting industrial protocol detection for {IP}:{Port}", ip, port);

        var probes = _portProbeService.GetPotentialProbes(port)
            .Where(p => IsProtocol(p.Protocol))
            .ToList();

        var result = await DetectFromProbesAsync(ip, port, probes, banner, ct);
        if (result != null)
        {
            return result;
        }

        var portProbe = _portProbeService.GetProbeForPort(port);
        if (portProbe != null && !probes.Contains(portProbe) &&
            IsProtocol(portProbe.Protocol) &&
            await VerifyProtocolWithProbeAsync(ip, port, portProbe, ct))
        {
            return portProbe.Protocol;
        }

        return null;
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Industrial protocol detection failed for {IP}:{Port}", ip, port);
        return null;
    }
}

public async Task<bool> VerifyProtocolAsync(string ip, int port, string protocol,
Cancellation token ct)
{
    var probe = _portProbeService.GetProbeForProtocol(protocol);
    if (probe == null) return false;

    return await VerifyProtocolWithProbeAsync(ip, port, probe, ct);
}

private async Task<string?> DetectByBannerAsync(string ip, int port, string banner,
Cancellation token ct)
{
    foreach (var probe in _portProbeService.GetAllProbes())
    {
        if (banner.Contains(probe.Protocol, StringComparison.OrdinalIgnoreCase) &&

```

```

        await VerifyProtocolWithProbeAsync(ip, port, probe, ct))
    {
        return probe.Protocol;
    }
}
return null;
}

```

```

private async Task<bool> VerifyProtocolWithProbeAsync(
    string ip,
    int port,
    PortProbeConfig probe,
    CancellationToken ct)
{
    try
    {
        if (string.IsNullOrEmpty(probe.ProbeData)) return false;

        if (IsUdpProtocol(probe.Protocol))
        {
            return await VerifyUdpProtocolAsync(ip, port, probe, ct);
        }
        return await VerifyTcpProtocolAsync(ip, port, probe, ct);
    }
    catch (Exception ex)
    {
        _logger.LogDebug(ex, "Probe verification failed for {Protocol} on {IP}:{Port}",
            probe.Protocol, ip, port);
        return false;
    }
}

```

```

private async Task<bool> VerifyTcpProtocolAsync(
    string ip,
    int port,
    PortProbeConfig probe,
    CancellationToken ct)
{
    using var client = new TcpClient();
    await client.ConnectAsync(ip, port, ct);

    using var stream = client.GetStream();

    byte[] probeData = GetProbeData(probe);
    await stream.WriteAsync(probeData, ct);

    var buffer = new byte[4096];
    var bytesRead = await stream.ReadAsync(buffer, ct);

    if (bytesRead == 0) return false;

    return VerifyIndustrialResponse(probe.Protocol, buffer, bytesRead);
}

```

```

private async Task<bool> VerifyUdpProtocolAsync(
    string ip,
    int port,
    PortProbeConfig probe,
    CancellationToken ct)
{
    using var client = new UdpClient();
    byte[] probeData = GetProbeData(probe);

    await client.SendAsync(probeData, probeData.Length, ip, port);
    var response = await client.ReceiveAsync(ct);

    return response.Buffer.Length > 0 &&
        VerifyIndustrialResponse(probe.Protocol, response.Buffer, response.Buffer.Length);
}

private byte[] GetProbeData(PortProbeConfig probe)
{
    return probe.Encoding switch
    {
        "ASCII" => Encoding.ASCII.GetBytes(probe.ProbeData),
        "Hex" => HexStringToByteArray(probe.ProbeData),
        "Base64" => Convert.FromBase64String(probe.ProbeData),
        _ => throw new NotSupportedException($"Encoding {probe.Encoding} not supported")
    };
}

private bool VerifyIndustrialResponse(string protocol, byte[] response, int length)
{
    var protocolUpper = protocol.ToUpper();
    var responseText = Encoding.ASCII.GetString(response, 0, length);
    var responseUpper = responseText.ToUpper();
    var hexResponse = BitConverter.ToString(response, 0, length).Replace("-", "");

    switch (protocolUpper)
    {
        case "MODBUS":
            return length >= 8 && response[7] == 0x03;
        case "S7COMM":
            return length > 0 && response[0] == 0x03 && response[5] == 0xD0;
        case "PROFINET":
            return length > 0 && response[0] == 0x03 && response[12] == 0xC1;
        case "ETHERCAT":
            return length >= 2 && response[0] == 0x00 && response[1] == 0x00;
        case "BACNET":
            return length > 0 && response[0] == 0x81;
        case "DNP3":
            return length > 0 && response[0] == 0x05 && response[1] == 0x64;
        case "IEC 61850":
            return length > 0 && response[0] == 0x61;
        case "IEC 60870-5-104":
            return length > 0 && response[0] == 0x68;
    }
}

```

```

case "OPC UA":
    return length > 0 && response[0] == 0x41 && response[1] == 0x43 &&
        response[2] == 0x4B && response[3] == 0x46;
case "CANOPEN":
    return length >= 8 && response[0] == 0x40 && response[4] == 0x43;
case "DEVICENET":
    return length >= 4 && response[0] == 0x85;
case "CONTROLNET":
    return length > 0 && response[0] == 0x00;
case "ETHERNET/IP":
    return length > 0 && response[0] == 0x65 && response[2] == 0x00 && response[3] ==
0x00;
case "CC-LINK":
case "CC-LINK SAFETY":
    return length >= 8 && response[0] == 0x54;
case "PROFIBUS":
    return length >= 6 && response[0] == 0x68;
case "FOUNDATION FIELDBUS":
    return length >= 10 && response[4] == 0x01;
case "HART PROTOCOL":
case "WIRELESSHART":
    return length >= 11 && response[6] == 0x02;
case "LONWORKS":
    return length >= 7 && response[0] == 0x01;
case "KNX":
    return length >= 10 && response[0] == 0x06 && response[1] == 0x10;
case "M-BUS":
    return length >= 4 && response[0] == 0xE5;
case "DALI":
    return length >= 4 && response[0] == 0x01;
case "IEC 60870-5-101":
case "IEC 60870-5-103":
    return length >= 4 && response[0] == 0x10 && response[1] == 0x4B;
case "MTCONNECT":
    return responseUpper.Contains("MTConnect");
case "ODVA CIP":
    return length > 0 && response[0] == 0x65;
case "ISO 9506 (MMS)":
    return length > 0 && response[0] == 0x61;
case "POWERLINK":
case "POWERLINK SAFETY":
    return length >= 14;
case "SERCOS III":
case "SERCOS SAFETY":
    return length >= 14;
case "AS-I":
    return length > 0 && response[0] == 0x00;
case "IO-LINK":
    return length > 0 && response[0] == 0x00;
case "SAFETYBUS P":
    return length >= 14;
case "PROFISAFE":
    return length >= 14 && response[0] == 0x00;

```

```

        case "CIP SAFETY":
            return length >= 14 && response[0] == 0x65;
        case "ETHERCAT SAFETY":
            return length >= 14 && response[0] == 0x00;
        case "MODBUS SECURE":
            return length >= 8 && response[7] == 0x03 && response[0] == 0x00;
        case "ENOCLEAN":
            return length > 0 && response[0] == 0x55;
        case "ZIGBEE":
            return length > 0 && response[0] == 0x00;
        case "Z-WAVE":
            return length > 0 && response[0] == 0x01;
        case "DMX512":
            return length > 0 && response[0] == 0x00;
        case "TCN (TRAIN COMMUNICATION NETWORK)":
            return length > 0 && response[0] == 0x00;
        default:
            return responseUpper.Contains(protocolUpper) ||
                hexResponse.Contains(protocolUpper);
    }
}

private bool IsUdpProtocol(string protocol)
{
    var udpProtocols = new HashSet<string>(StringComparer.OrdinalIgnoreCase)
    {
        "BACNET", "DNP3", "PROFINET", "ETHERCAT"
    };
    return udpProtocols.Contains(protocol);
}

private static byte[] HexStringToByteArray(string hex)
{
    hex = hex.Replace("\\x", "").Replace(" ", "");
    return Enumerable.Range(0, hex.Length)
        .Where(x => x % 2 == 0)
        .Select(x => Convert.ToByte(hex.Substring(x, 2), 16))
        .ToArray();
}
}

```

sample of port probe

```

Code "PortProbes": [
    {
        "Port": 5671,
        "ProbeData": "414D515100010000",
        "Encoding": "Hex",
        "Protocol": "AMQP",
        "IsGeneric": false,
        "Priority": 2
    },
    {
        "Port": 5672,

```

```
"ProbeData": "414D515100010000",
"Encoding": "Hex",
"Protocol": "AMQP",
"IsGeneric": false,
"Priority": 2
},
{
  "Port": 80,
  "ProbeData": "GET / HTTP/1.1\r\nHost: camera\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "CameraWebInterface",
  "IsGeneric": false,
  "Priority": 2
},
{
  "Port": 80,
  "ProbeData": "GET /axis-cgi/param.cgi?action=list HTTP/1.1\r\nHost: camera\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "Axis-VAPIX",
  "IsGeneric": false,
  "Priority": 1
},
{
  "Port": 8080,
  "ProbeData": "GET /axis-cgi/param.cgi?action=list HTTP/1.1\r\nHost: camera\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "Axis-VAPIX",
  "IsGeneric": false,
  "Priority": 2
},
{
  "Port": 443,
  "ProbeData": "GET /axis-cgi/param.cgi?action=list HTTP/1.1\r\nHost: camera\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "Axis-VAPIX-SSL",
  "IsGeneric": false,
  "Priority": 1
},
{
  "Port": 8443,
  "ProbeData": "GET /axis-cgi/param.cgi?action=list HTTP/1.1\r\nHost: camera\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "Axis-VAPIX-SSL",
  "IsGeneric": false,
  "Priority": 2
},
{
  "Port": 407,
  "ProbeData": "\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00",
  "Encoding": "Hex",
  "Protocol": "AS-i",
  "IsGeneric": false,
  "Priority": 3
}
```

[illegible]

```
"Protocol": "CC-Link Safety",
"IsGeneric": false,
"Priority": 3
},
{
  "Port": 50003,
  "ProbeData": "\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00",
  "Encoding": "Hex",
  "Protocol": "CIP Safety",
  "IsGeneric": false,
  "Priority": 3
},
{
  "Port": 5683,
  "ProbeData":
"\\x40\\x01\\x39\\x77\\x65\\x6c\\x6c\\x2d\\x6b\\x6e\\x6f\\x77\\x6e\\x04\\x63\\x6f\\x72\\x65",
  "Encoding": "Hex",
  "Protocol": "CoAP",
  "IsGeneric": false,
  "Priority": 3
},
{
  "Port": 3600,
  "ProbeData": "\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00",
  "Encoding": "Hex",
  "Protocol": "Coaxitron",
  "IsGeneric": false,
  "Priority": 1
},
{
  "Port": 3601,
  "ProbeData": "\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00",
  "Encoding": "Hex",
  "Protocol": "Coaxitron",
  "IsGeneric": false,
  "Priority": 2
},
{
  "Port": 80,
  "ProbeData": "GET /cgi-bin/api.cgi?cmd=GetDevInfo HTTP/1.1\\r\\nHost: camera\\r\\n\\r\\n",
  "Encoding": "ASCII",
  "Protocol": "Dahua-API",
  "IsGeneric": false,
  "Priority": 2
},
{
  "Port": 37777,
  "ProbeData": "\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00",
  "Encoding": "Hex",
  "Protocol": "Dahua-Private",
  "IsGeneric": false,
  "Priority": 3
},
}
```

[illegible]

```
"IsGeneric": false,
"Priority": 3
},
{
  "Port": 2575,
  "ProbeData": "GET / HTTP/1.0\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "DICOM",
  "IsGeneric": false,
  "Priority": 3
},
{
  "Port": 512,
  "ProbeData": "\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00",
  "Encoding": "Hex",
  "Protocol": "DMX512",
  "IsGeneric": false,
  "Priority": 3
},
{
  "Port": 20000,
  "ProbeData": "\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00",
  "Encoding": "Hex",
  "Protocol": "DNP3",
  "IsGeneric": false,
  "Priority": 2
},
{
  "Port": 53,
  "ProbeData": "\\x00\\x0a\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00",
  "Encoding": "Hex",
  "Protocol": "DNS",
  "IsGeneric": false,
  "Priority": 1
},
{
  "Port": 44818,
  "ProbeData": "6f00000000000000000000000000000000000000000000000000000000000000",
  "Encoding": "Hex",
  "Protocol": "EtherNet/IP",
  "IsGeneric": false,
  "Priority": 2
},
{
  "Port": 34963,
  "ProbeData": "\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00",
  "Encoding": "Hex",
  "Protocol": "EtherCAT",
  "IsGeneric": false,
  "Priority": 2
},
{
  "Port": 50005,
```

```

"ProbeData": "\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00",
"Encoding": "Hex",
"Protocol": "EtherCAT Safety",
"IsGeneric": false,
"Priority": 3
},
{
  "Port": 1089,
  "ProbeData": "\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00",
  "Encoding": "Hex",
  "Protocol": "FOUNDATION Fieldbus",
  "IsGeneric": false,
  "Priority": 3
},
{
  "Port": 21,
  "ProbeData": "USER anonymous\r\n",
  "Encoding": "ASCII",
  "Protocol": "FTP",
  "IsGeneric": false,
  "Priority": 2
},
{
  "Port": 2121,
  "ProbeData": "USER anonymous\r\n",
  "Encoding": "ASCII",
  "Protocol": "FTP",
  "IsGeneric": false,
  "Priority": 3
},
{
  "Port": 50051,
  "ProbeData": "\\x00\\x00\\x00\\x04\\x00\\x00\\x00\\x00\\x00",
  "Encoding": "Hex",
  "Protocol": "gRPC",
  "IsGeneric": false,
  "Priority": 2
},
{
  "Port": 6000,
  "ProbeData": "\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00",
  "Encoding": "Hex",
  "Protocol": "GV-Protocol",
  "IsGeneric": false,
  "Priority": 1
},
{
  "Port": 6001,
  "ProbeData": "\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00",
  "Encoding": "Hex",
  "Protocol": "GV-Protocol",
  "IsGeneric": false,
  "Priority": 2
}

```

```
,
{
  "Port": 6002,
  "ProbeData": "\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00",
  "Encoding": "Hex",
  "Protocol": "GV-Protocol",
  "IsGeneric": false,
  "Priority": 2
},
{
  "Port": 5094,
  "ProbeData": "\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00",
  "Encoding": "Hex",
  "Protocol": "HART Protocol",
  "IsGeneric": false,
  "Priority": 3
},
{
  "Port": 8000,
  "ProbeData": "GET /ISAPI/System/deviceInfo HTTP/1.1\r\nHost: camera\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "Hikvision",
  "IsGeneric": false,
  "Priority": 2
},
{
  "Port": 8000,
  "ProbeData": "GET /ISAPI/System/deviceInfo HTTP/1.1\r\nHost: camera\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "Hikvision-API",
  "IsGeneric": false,
  "Priority": 2
},
{
  "Port": 8001,
  "ProbeData": "GET /ISAPI/System/deviceInfo HTTP/1.1\r\nHost: camera\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "Hikvision-API",
  "IsGeneric": false,
  "Priority": 2
},
{
  "Port": 8002,
  "ProbeData": "GET /ISAPI/System/deviceInfo HTTP/1.1\r\nHost: camera\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "Hikvision-API",
  "IsGeneric": false,
  "Priority": 2
},
{
  "Port": 554,
```

```
"ProbeData": "DESCRIBE rtsp://camera/Streaming/Channels/101 RTSP/1.0\r\nCSeq:
1\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "Hikvision-RTSP",
  "IsGeneric": false,
  "Priority": 2
},
{
  "Port": 0,
  "ProbeData": "GET / HTTP/1.0\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "HTTP",
  "IsGeneric": true,
  "Priority": 1
},
{
  "Port": 0,
  "ProbeData": "GET / HTTP/1.1\r\nHost: example.com\r\nConnection: close\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "HTTP",
  "IsGeneric": true,
  "Priority": 1
},
{
  "Port": 0,
  "ProbeData": "PRI * HTTP/2.0\r\n\r\nSM\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "HTTP/2",
  "IsGeneric": true,
  "Priority": 2
},
{
  "Port": 0,
  "ProbeData": "GET / HTTP/1.1\r\nHost: example.com\r\nConnection: Upgrade,
HTTP2-Settings\r\nUpgrade: h2c\r\nHTTP2-Settings: AAMAAABkAAQAAP__\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "HTTP/2",
  "IsGeneric": true,
  "Priority": 2
},
{
  "Port": 0,
  "ProbeData": "GET / HTTP/3\r\nHost: example.com\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "HTTP/3",
  "IsGeneric": true,
  "Priority": 3
},
{
  "Port": 80,
  "ProbeData": "GET / HTTP/1.0\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "HTTP",
```

```
"IsGeneric": false,
"Priority": 1
},
{
  "Port": 80,
  "ProbeData": "GET / HTTP/1.1\r\nHost: example.com\r\nConnection: close\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "HTTP",
  "IsGeneric": false,
  "Priority": 1
},
{
  "Port": 80,
  "ProbeData": "PRI * HTTP/2.0\r\n\r\nSM\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "HTTP/2",
  "IsGeneric": false,
  "Priority": 2
},
{
  "Port": 80,
  "ProbeData": "GET / HTTP/1.1\r\nHost: example.com\r\nConnection: Upgrade,
HTTP2-Settings\r\nUpgrade: h2c\r\nHTTP2-Settings: AAMAAABkAAQAAP__\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "HTTP/2",
  "IsGeneric": false,
  "Priority": 2
},
{
  "Port": 81,
  "ProbeData": "GET / HTTP/1.0\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "HTTP",
  "IsGeneric": false,
  "Priority": 2
},
{
  "Port": 81,
  "ProbeData": "GET / HTTP/1.1\r\nHost: example.com\r\nConnection: close\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "HTTP",
  "IsGeneric": false,
  "Priority": 2
},
{
  "Port": 81,
  "ProbeData": "PRI * HTTP/2.0\r\n\r\nSM\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "HTTP/2",
  "IsGeneric": false,
  "Priority": 3
},
{
  "Port": 81,
  "ProbeData": "PRI * HTTP/2.0\r\n\r\nSM\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "HTTP/2",
  "IsGeneric": false,
  "Priority": 3
},
{
  "Port": 81,
  "ProbeData": "PRI * HTTP/2.0\r\n\r\nSM\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "HTTP/2",
  "IsGeneric": false,
  "Priority": 3
}
```

```
"Port": 2000,
"ProbeData": "GET / HTTP/1.0\r\n\r\n",
"Encoding": "ASCII",
"Protocol": "HTTP",
"IsGeneric": false,
"Priority": 3
},
{
  "Port": 2000,
  "ProbeData": "GET / HTTP/1.1\r\nHost: example.com\r\nConnection: close\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "HTTP",
  "IsGeneric": false,
  "Priority": 3
},
{
  "Port": 2000,
  "ProbeData": "PRI * HTTP/2.0\r\n\r\nSM\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "HTTP/2",
  "IsGeneric": false,
  "Priority": 3
},
{
  "Port": 3000,
  "ProbeData": "GET / HTTP/1.0\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "HTTP",
  "IsGeneric": false,
  "Priority": 3
},
{
  "Port": 3000,
  "ProbeData": "GET / HTTP/1.1\r\nHost: example.com\r\nConnection: close\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "HTTP",
  "IsGeneric": false,
  "Priority": 3
},
{
  "Port": 3000,
  "ProbeData": "PRI * HTTP/2.0\r\n\r\nSM\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "HTTP/2",
  "IsGeneric": false,
  "Priority": 3
},
{
  "Port": 3456,
  "ProbeData": "GET / HTTP/1.0\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "HTTP",
  "IsGeneric": false,
```

```
"Priority": 3
},
{
  "Port": 3456,
  "ProbeData": "GET / HTTP/1.1\r\nHost: example.com\r\nConnection: close\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "HTTP",
  "IsGeneric": false,
  "Priority": 3
},
{
  "Port": 3456,
  "ProbeData": "PRI * HTTP/2.0\r\n\r\nSM\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "HTTP/2",
  "IsGeneric": false,
  "Priority": 3
},
{
  "Port": 4000,
  "ProbeData": "GET / HTTP/1.0\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "HTTP",
  "IsGeneric": false,
  "Priority": 3
},
{
  "Port": 4000,
  "ProbeData": "GET / HTTP/1.1\r\nHost: example.com\r\nConnection: close\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "HTTP",
  "IsGeneric": false,
  "Priority": 3
},
{
  "Port": 4000,
  "ProbeData": "PRI * HTTP/2.0\r\n\r\nSM\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "HTTP/2",
  "IsGeneric": false,
  "Priority": 3
},
{
  "Port": 5000,
  "ProbeData": "GET / HTTP/1.0\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "HTTP",
  "IsGeneric": false,
  "Priority": 3
},
{
  "Port": 5000,
  "ProbeData": "GET / HTTP/1.1\r\nHost: example.com\r\nConnection: close\r\n\r\n",
```

```
"Encoding": "ASCII",
"Protocol": "HTTP",
"IsGeneric": false,
"Priority": 3
},
{
  "Port": 5000,
  "ProbeData": "PRI * HTTP/2.0\r\n\r\nSM\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "HTTP/2",
  "IsGeneric": false,
  "Priority": 3
},
{
  "Port": 7000,
  "ProbeData": "GET / HTTP/1.0\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "HTTP",
  "IsGeneric": false,
  "Priority": 3
},
{
  "Port": 7000,
  "ProbeData": "GET / HTTP/1.1\r\nHost: example.com\r\nConnection: close\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "HTTP",
  "IsGeneric": false,
  "Priority": 3
},
{
  "Port": 7000,
  "ProbeData": "PRI * HTTP/2.0\r\n\r\nSM\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "HTTP/2",
  "IsGeneric": false,
  "Priority": 3
},
{
  "Port": 8000,
  "ProbeData": "GET / HTTP/1.0\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "HTTP",
  "IsGeneric": false,
  "Priority": 3
},
{
  "Port": 8000,
  "ProbeData": "GET / HTTP/1.1\r\nHost: example.com\r\nConnection: close\r\n\r\n",
  "Encoding": "ASCII",
  "Protocol": "HTTP",
  "IsGeneric": false,
  "Priority": 3
}
```

]

Share your thoughts and observations so I can learn throughout the process.

MadScientist - September 19, 2025, 6:17 pm

Awesome contribution, going to have to mess about with it this weekend. I imagine this could be a useful part of a variety of tools

itcorp - September 19, 2025, 6:17 pm

As our great member [Z](#) said, share not just consume.

(Sep 19, 2025 06:17 PM)MadScientist Wrote: [\[CITE\]](#) Awesome contribution, going to have to mess about with it this weekend. I imagine this could be a useful part of a variety of tools
ty bud <3 maybe I'll do a few more things this weekend if I'm motivated enough :P

ttbr0 - September 19, 2025, 6:24 pm

this is very cool. there is so much people can learn from by reading this code. one thing i would improve on is the detection code. you have a lot of if statements to return a boolean and it doesnt scale very well if you want to add more. my naive suggestion would be using the strategy pattern <https://refactoring.guru/design-patterns/strategy> but if anyone who is a better engineer than i am, i would like to know what fits better here

itcorp - September 19, 2025, 6:40 pm

(Sep 19, 2025 06:24 PM)ttbr0 Wrote: [\[CITE\]](#) this is very cool. there is so much people can learn from by reading this code. one thing i would improve on is the detection code. you have a lot of if statements to return a boolean and it doesnt scale very well if you want to add more. my naive suggestion would be using the strategy pattern <https://refactoring.guru/design-patterns/strategy> but if anyone who is a better engineer than i am, i would like to know what fits better hereFirst of all, thank you for the kind words.

Second...I saw you online and was actually waiting for your response.And I have to tell you that you are actually right there are many ways to improve this using maps through dictionary etc etc and further improve the code.Most of the code is not for production ready so that members can join in the discussion and contribute in order to reach a final solution.

I'm glad you gave your constructive opinion. <3

flyingduck - September 21, 2025, 8:39 am

great stuff, like always :)

itcorp - September 21, 2025, 8:40 am

(Sep 21, 2025 08:39 AM)flyingduck Wrote: [\[CITE\]](#) great stuff, like always :)
ty bud <3

Oh_Well - September 21, 2025, 12:33 pm

Thanks for sharing this. Seriously impressive piece of kit for a weekend project. You've covered a ton of protocols already.

I've been looking through the code, and I thought of some next steps (I'd love to get ur feedback)

You can break out the Industrial and Standard detectors into a plugin interface (e.g., "IProtocolDetectorPlugin"). This would let you (and others) drop in new DLLs for things like IoT protocols, game server protocols, or proprietary vendor stuff without ever touching the main core. Makes it super easy for the community to extend.

Right now it's mostly banner grabbing and single-packet checks, right? Adding simple stateful analysis to track a conversation over a few packets would potentially improve the accuracy for tricky protocols that need a handshake.

There are a few sneaky "GetAwaiter()" "GetResult()" calls (like in the timer). Swapping all of that to pure async/await would make it handle large-scale scans without breaking a sweat.

So I guess the plugin architecture would be my first pick as the way to get others involved and build a whole ecosystem around this engine.

What's your take? What do you think are the next steps?
Great work!

itcorp - September 21, 2025, 2:04 pm

(Sep 21, 2025 12:33 PM)Oh_Well Wrote: [\[CITE\]](#) Thanks for sharing this. Seriously impressive piece of kit for a weekend project. You've covered a ton of protocols already.

I've been looking through the code, and I thought of some next steps (I'd love to get ur feedback)

You can break out the Industrial and Standard detectors into a plugin interface (e.g., "IProtocolDetectorPlugin"). This would let you (and others) drop in new DLLs for things like IoT protocols, game server protocols, or proprietary vendor stuff without ever touching the main core. Makes it super easy for the community to extend.

Right now it's mostly banner grabbing and single-packet checks, right? Adding simple stateful analysis to track a conversation over a few packets would potentially improve the accuracy for tricky protocols that need a handshake.

There are a few sneaky "GetAwaiter()" "GetResult()" calls (like in the timer). Swapping all of that to pure async/await would make it handle large-scale scans without breaking a sweat.

So I guess the plugin architecture would be my first pick as the way to get others involved and build a whole ecosystem around this engine.

What's your take? What do you think are the next steps?
Great work!

Thank you for the actual code analysis, you spotted a anti-pattern. But there are still some minor problems that the reader should pay attention to like thread safe, proper handling of cancellation token, expensive consumption through linq etc etc.

Thank you for the constructive response.