DRAFT REPORT
Security Assessment of Open Tech Fund's oLink

DISCLAIMER: The purpose of this Draft Report is to provide insight into the results and potential findings identified during the course of the engagement. The potential findings and information contained herein may differ from the final report that has completed IncludeSec internal Quality Assurance (QA) review. As such, QA has not yet completed and the information contained herein may reflect false positives, non-adjusted risk categorizations, non-ideal grammar/spelling/formatting/syntax, and does not contain an executive-level summary or supplemental appendices.

IncludeSec recommends waiting until the Final Report is delivered if there is a need to share results with compliance, regulatory, customers, or other related internal or external stakeholders. Furthermore, it is recommended this Draft Report be deleted from internal bug tracking and file stores upon receipt of the Final Report.

TABLE OF CONTENTS

RISK CATEGORIZATIONS

At the conclusion of the assessment, Include Security categorized findings into five levels of perceived security risk: Critical, High, Medium, Low, or Informational. **The risk categorizations below are guidelines that IncludeSec understands reflect best practices in the security industry and may differ from a client's internal perceived risk. Additionally, all risk is viewed as "location agnostic" as if the system in question was deployed on the Internet. It is common and encouraged that all clients recategorize findings based on their internal business risk tolerances. Any discrepancies between assigned risk and internal perceived risk are addressed during the course of remediation testing.**

Critical-Risk findings are those that pose an immediate and serious threat to the company's infrastructure and customers. This includes loss of system, access, or application control, compromise of administrative accounts or restriction of system functions, or the exposure of confidential information. These threats should take priority during remediation efforts.

High-Risk findings are those that could pose serious threats including loss of system, access, or application control, compromise of administrative accounts or restriction of system functions, or the exposure of confidential information.

Medium-Risk findings are those that could potentially be used with other techniques to compromise accounts, data, or performance.

Low-Risk findings pose limited exposure to compromise or loss of data, and are typically attributed to configuration, and outdated patches or policies.

Informational findings pose little to no security exposure to compromise or loss of data which cover defense-in-depth and best-practice changes which we recommend are made to the application. Any informational findings for which the assessment team perceived a direct security risk, were also reported in the spirit of full disclosure but were considered to be out of scope of the engagement.

The findings represented in this draft report are listed by a risk rated short name (e.g., C1, H2, M3, L4, I5) and finding title. Each draft finding may include: Description, Impact, Reproduction (evidence necessary to reproduce each finding), Recommended Remediation, and References.

DRAFT SUMMARY OF POTENTIAL FINDINGS

C1: Operating System Command Injection

Description:

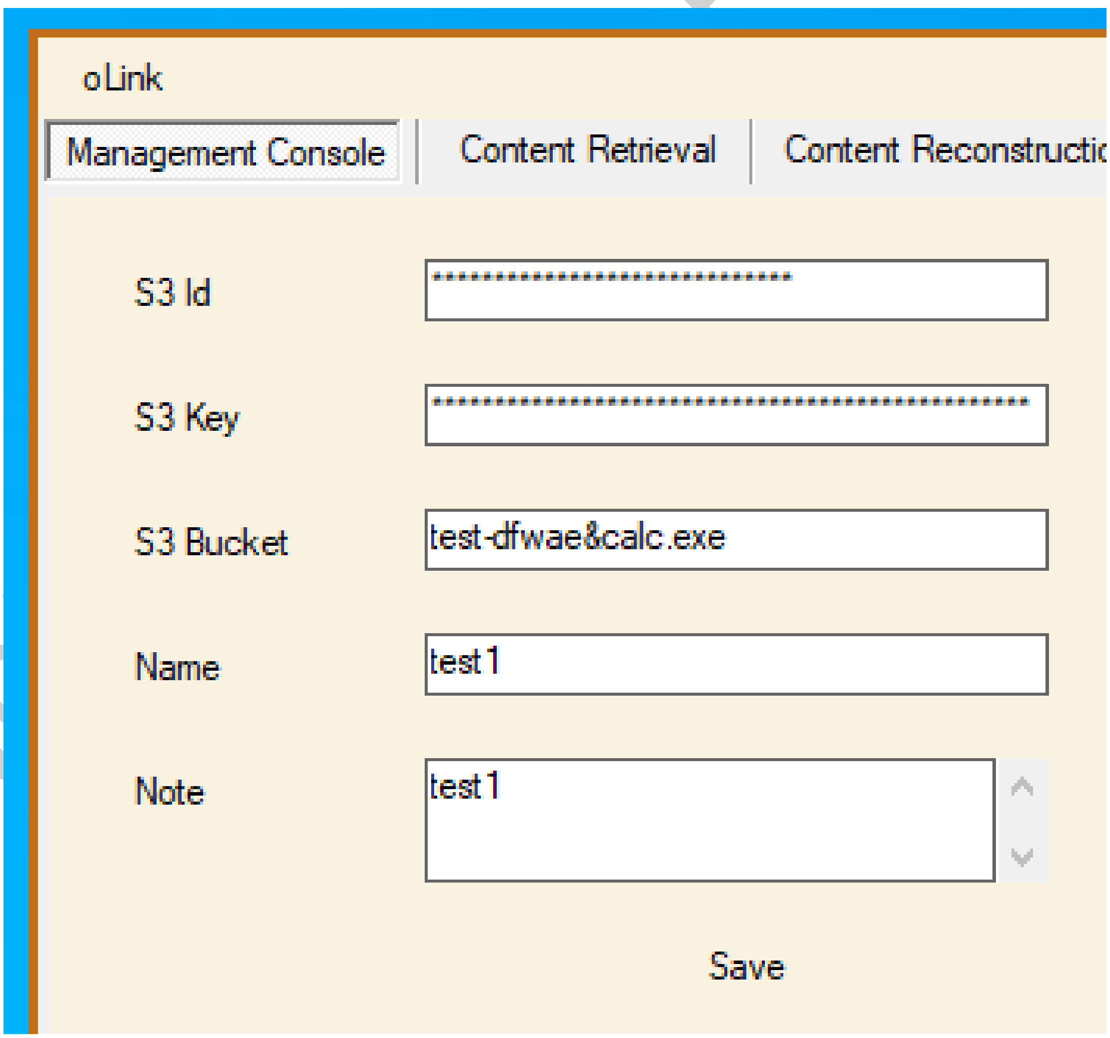
An operating system command injection vulnerability was found in the **oLink** application, which could be exploited to run arbitrary OS commands (terminal commands) on the host running **oLink**.

Impact:

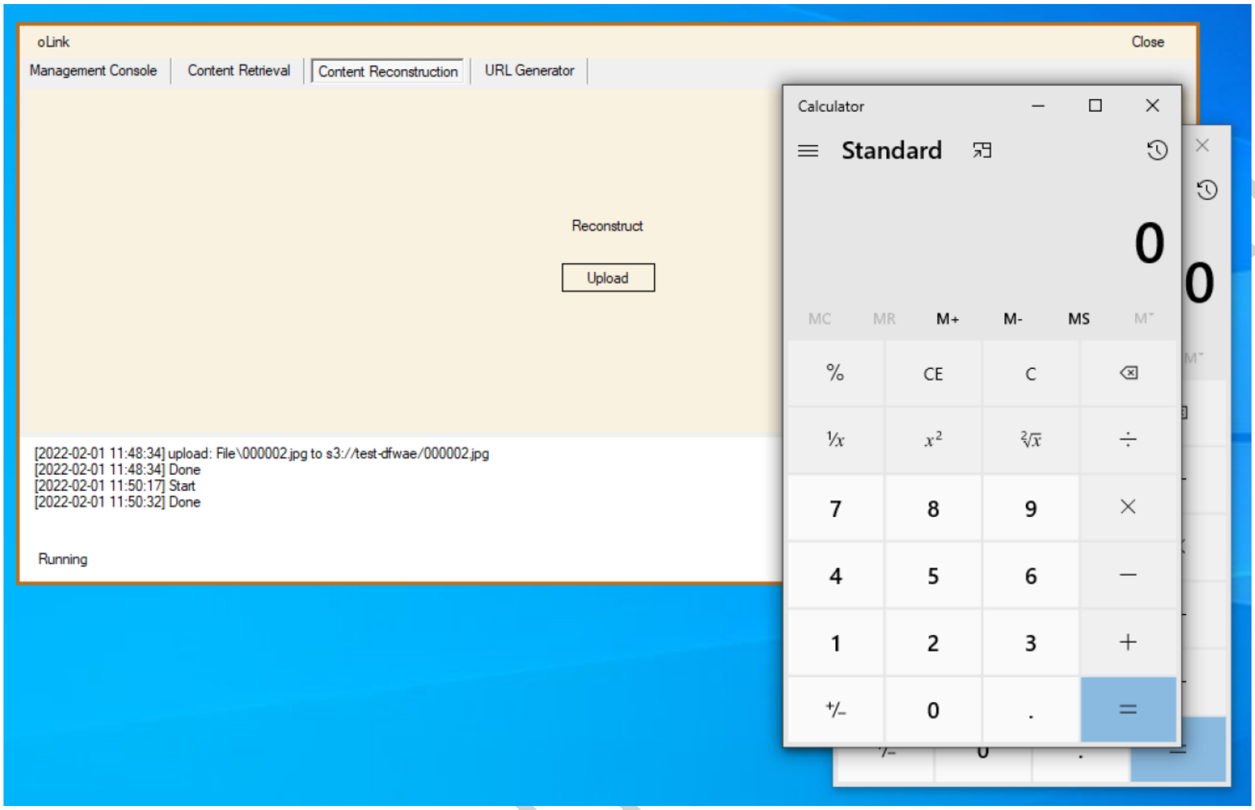
An attacker who can exploit this vulnerability would have complete and total control over the **uLink** user's machine and all data and information passing through it. On its own, this vulnerability colud be exploited by the **oLink** user entering malicous commands into the **oLink** user interface themselves (for example through social engineering). However, since the data in the affected inputs is stored in the database, an attacker who has access to the database could store malicious commands in the database, to be executed when **oLink** is next run.

Reproduction:

To reproduce this vulnerability, the string **&calc.exe** was added to the end of the **S3 Id**, **S3 Key**, and **S3 Bucket** values in **oLink**. Note that **oLink** obscured the contents of the **S3 Id** and **S3 Key** values, making them more likely to be targeted for exploitation.



Next, when the **Upload** button was pressed, the **calc.exe** several intstances of the Windows Calulator application were launched.



The root cause was identified in the file **olink/oLink/FormLink.cs**, lines 339-369:

```
339         private void Upload()
340         {
341             ShowMsgD("Start");
342             try
343             {
344                 string s0 = Path.GetDirectoryName(Application.ExecutablePath);
345                 string s = @"set AWS_ACCESS_KEY_ID=^S3Id^
346 set AWS_SECRET_ACCESS_KEY=^S3Key^
347 aws s3 sync C:\oLink\Site s3://^S3Bucket^/Site --acl public-read --region eu-west-1
348 aws s3 sync C:\oLink\File s3://^S3Bucket^/File --acl public-read --region eu-west-1"
349 .Replace("^S3Id^", textBoxS3Id.Text.Trim())
350 .Replace("^S3Key^", textBoxS3Key.Text.Trim())
351 .Replace("^S3Bucket^", textBoxS3Bucket.Text.Trim());
352                 Process p = new Process();
353                 p.StartInfo.FileName = "cmd.exe";
354                 p.StartInfo.UseShellExecute = false;
355                 p.StartInfo.RedirectStandardInput = true;
356                 p.StartInfo.RedirectStandardOutput = true;
357                 p.StartInfo.RedirectStandardError = true;
358                 p.StartInfo.CreateNoWindow = true;
359                 p.OutputDataReceived += new DataReceivedEventHandler(OnOutputDataReceived);
360                 p.Start();
361                 p.BeginOutputReadLine();
362                 ##$$
363                 p.StandardInput.WriteLine("exit");
364                 p.WaitForExit();
365                 if (p.ExitCode != 0) ShowMsgD(p.StandardError.ReadToEnd());
366             }
367             catch (Exception ex) { Log(MethodBase.GetCurrentMethod().Name + ": " + ex.Message); }
368             ShowMsgD("Done");
369         }
```

The code launches the **aws** command-line tool in order to upload data to **S3**. It does so by starting a **cmd.exe** process and writing the commands to it. The commands themselves are constructed by replacing placeholder text within strings with the parameters from the user interface inputs, which previously were loaded from the database.

Recommended Remediation:

The assessment team recommends avoiding direct OS commands from application-layer code whenever possible, as many application frameworks provide APIs to achieve the same functionality. Amazon publishes an AWS SDK for .NET applications.

If untrusted input must be passed to OS commands, the assessment team recommends validating it against a whitelist of allowed values. For example, if the system needs an alphanumeric file name, the user input can be checked against the regular expression **/[A-Za-z0-9]/**.

H1: Cryptographic Secrets Stored in Source Code Repository

Description:

A database password was found within the **oLink** application source code repository. Access to the source code repository and its history could be exposed by another exploit, or if the application is open sourced, which would provide access to these database credentials to an attacker.

Impact:

The database credentials were likely used by developers, and potentially by users of the application. An attacker with access to the database credentials and access to the database server (for example the network where the database server is installed, depending on the database configuration and deployment) could cause the **oLink** application to execute arbitrary code (see the **Operating System Command Injection** issue for more details on how modifying the AWS credentials stored in the database could lead to code execution), gain access to the AWS account used by **oLink**, and learn what web pages **oLink** had been used to copy.

Reproduction:

The database credentials were found in the file **olink/oLink/App.config**, on line 4 (the password has been redacted):

```
1 <?xml version="1.0" encoding="utf-8" ?>
2 <configuration>
3     <appSettings>
4               <add key="DB" value="server=.\SQLEXPRESS;Initial Catalog=oLink;User ID=sas;Password=[REDACTED]"/>
5       </appSettings>
6     <startup>
7         <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
8     </startup>
9 </configuration>
```

Recommended Remediation:

The assessment team recommends removing the **App.config** configuration file from the source code repository, removing it from the **git** history, and changing the password on any database that uses the password stored in the file.

H2: Html Sanitization Bypass

Description:

The **oLink** application contained code that attempted to sanitize HTML by removing all tags that are not explicitly allowed by the code. However, it was possible to bypass this sanitization, allowing JavaScript code and other content to be injected into the site mirrored using **oLink**.

Impact:

An attacker who controls a site that is mirrored using **oLink** could inject JavaScript code and other HTML content into the resulting mirrored page. The attacker may host their own malicius site or attempt to put malicious code in a compromised site. The payload could allow the attacker to identify anyone viewing the mirrored page, for example by injecting JavaScript that makes requests to an attacker-controlled host from the context of the compromised mirror on **S3**.

Reproduction:

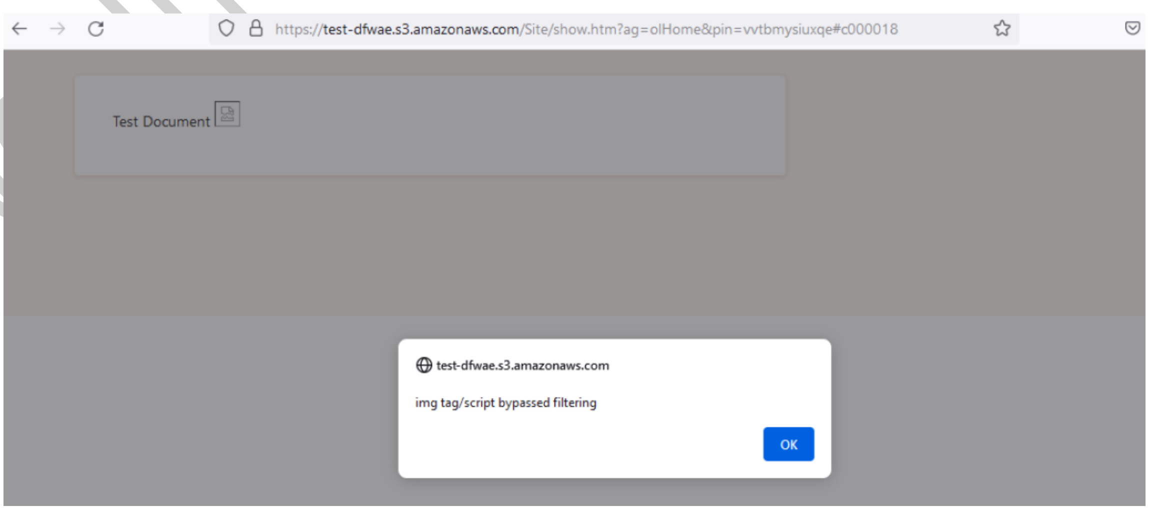
This HTML document demonstrates the vulnerability. It was retrieved, reconstructed, and uploaded using **oLink**.

```
<html>
<head>
<title>Test document</title>
</head>
<body>
Test Document
<><<>img src=x onerror="alert('img tag/script bypassed filtering');">
</body>
```

When reconstructed, this file was created at **C:\oLink\Site\c000018.htm**, showing that an unwanted **img** tag containing JavaScript code existed in the document:

```
<div class='main'>
  <div class='maia'>
    <div class='lisc' style='padding:0 0 15px 0;'>
      <div class='artl'>
<div class="artl">
Test Document
<img src=x onerror="alert('img tag/script bypassed filtering');">
</div>
      </div>
    </div>
  </div>
  <div style='height:36px; clear:both;'></div>
</div>
```

When uploaded to **S3** and viewed in a browser, the JavaScript code executed, showing the **alert** dialog:



The root cause existed in the **HtmlDel2** method which existed in **olink/oLink/FormLink.cs**, lines 1051-1141. This method was used to sanitize HTML throughout **oLink**, especially HTML that has been downloaded from a site that will be mirrored.

```
1051        public string HtmlDel2(string content, string url)
1052        {
1053            try
1054            {
1055                content = HtmlDel(content, "(<head)([\\S\\s]*?)(</head>)");
...
1075
1076                MatchCollection mc0 = new Regex("(<)([\\S\\s]*?)(>)").Matches(content);
1077                foreach (Match m in mc0)
1078                {
1079                    if (m.Value == "<p>" || m.Value == "</p>" || m.Value == "<p class=\"artl\">" || m.Value == "<p
class=\"artc\">"
1080                        || m.Value == "<p class=\"artl\" style=\"text-align:center;\">" || m.Value == "<p
style=\"text-align:center;\">"
1081                        || m.Value == "<b>" || m.Value == "</b>" || m.Value == "<br/>" || m.Value == "<br>" ||
m.Value == "<br />"
1082                        || m.Value == "<strong>" || m.Value == "</strong>"
1083                        || m.Value == "</h1>" || m.Value == "</h2>" || m.Value == "</h3>" || m.Value == "</h4>" ||
m.Value == "</h5>"
1084                        || m.Value == "<em>" || m.Value == "</em>" || m.Value == "<sup>" || m.Value == "</sup>") ;
1085                    else if (m.Value.StartsWith("<h1")) { content = content.Replace(m.Value, "<h1 style=\"text-
align:center;\">"); }
1086                    else if (m.Value.StartsWith("<h2")) { content = content.Replace(m.Value, "<h2 style=\"text-
align:center;\">"); }
...
1132                    else content = content.Replace(m.Value, "");
1133                }
1134
1135                content = content.Replace("\t", "").Replace("\n\n", "\n").Replace("\n\n", "\n").Replace("\n\n", "\n")
1136                    .Replace("\n", "\r\n").Replace("\r\r\n", "\r\n");
1137                return content;
1138            }
1139            catch (Exception ex) { Log(MethodBase.GetCurrentMethod().Name + ": " + ex.Message); }
1140            return "";
1141        }
```

Line 1076 searches the content for HTML tags, then lines 1079-1131 handle any allowed tags or other special cases, and line 1132 attempts to delete any other tag.

In the test case above, this is the relevant line:

```
<><<>img src=x onerror="alert('img tag/script bypassed filtering');">
```

The code first identified the regular expression match **<>** in line 1076, and delted all instances of that string within **content**, resulting in the line becoming:

```
<img src=x onerror="alert('img tag/script bypassed filtering');">
```

The next regular expression match was **<<>**, however since **<>** was a substring of this match and was already deleted from the **content**, the **<<>** match no longer existed in **content**, and the **img** tag remained.

Recommended Remediation:

In general, the **oLink** source code uses regular expressions extensively to process HTML, which is considered an unsafe practice because it can lead to security issues such as this one. Instead, the assessment team recommends using a HTML parsing and sanitization library designed to be robust against malicious or untrusted HTML input. These libraries are often used to sanitize HTML to prevent cross-site scripting attacks against web applications. The **HtmlSanitizer** library is one example of such a library.

M1: HTTPS Not Enforced and Certificate Legitimacy Not Confirmed by Client

Description:

The **oLink** application downloads articles from arbitrary domains to host on **S3**. These downloads were not required to be encrypted using **HTTPS**, and when they did use **HTTPS**, the application disables certificate validation.

Impact:

As a result, a server-spoofing or Man-in-the-Middle attack could be performed against the application for downloads over **HTTP** or **HTTPS**.

Reproduction:

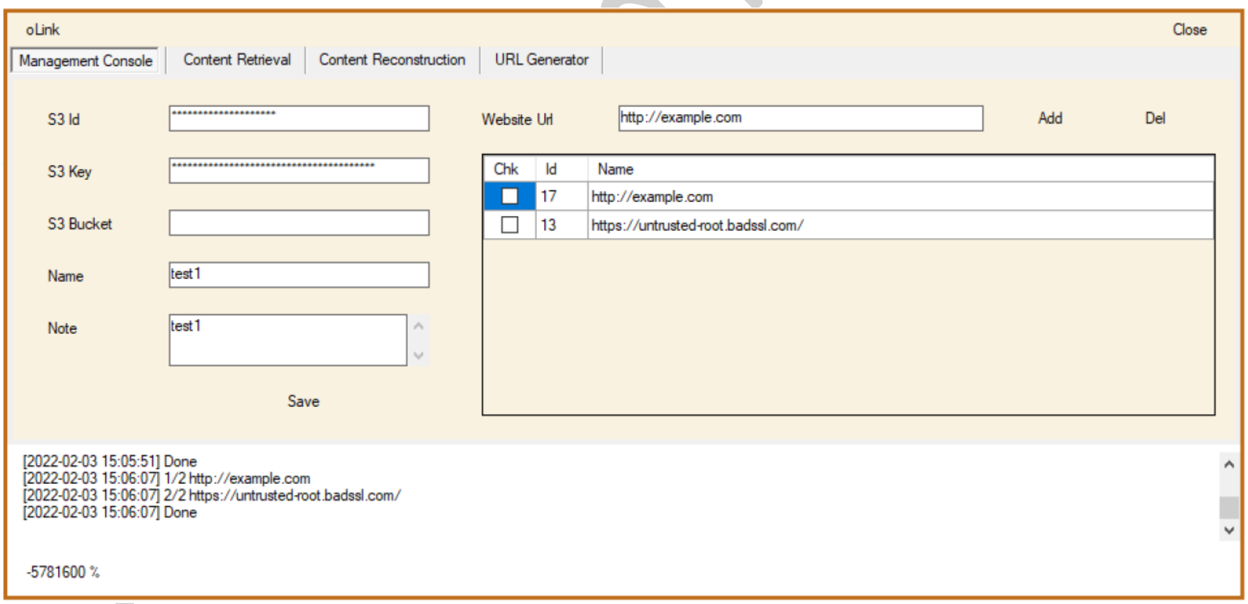
The code disables certificate validation by setting the **ServerCertificateValidationCallback** to a function that always returns true in **olink/oLink/FormLink.cs**, lines 37-42:

```
37          private void FormMain_Load(object sender, EventArgs e)
38          {
39              try
40              {
41                  ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;
42                  ServicePointManager.ServerCertificateValidationCallback = delegate { return true; };
```

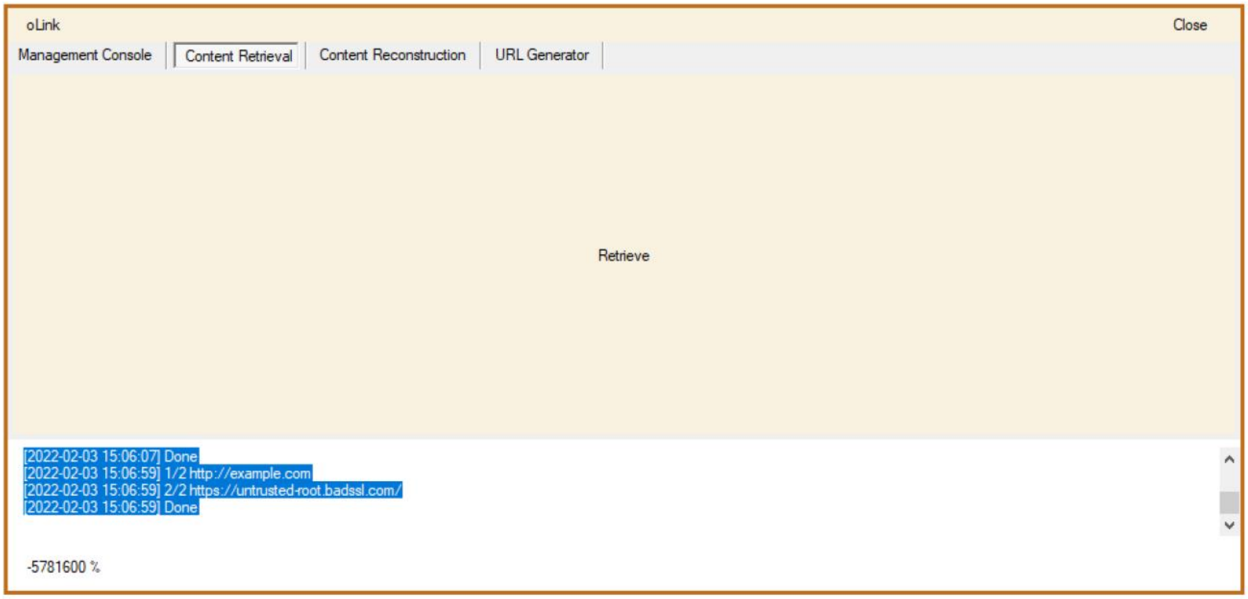
Downloading pages over **HTTP** and over **HTTPS** with a bad certificate were tested by using the tool to download from the following URLs:

- http://example.com
- https://untrusted-root.badssl.com/

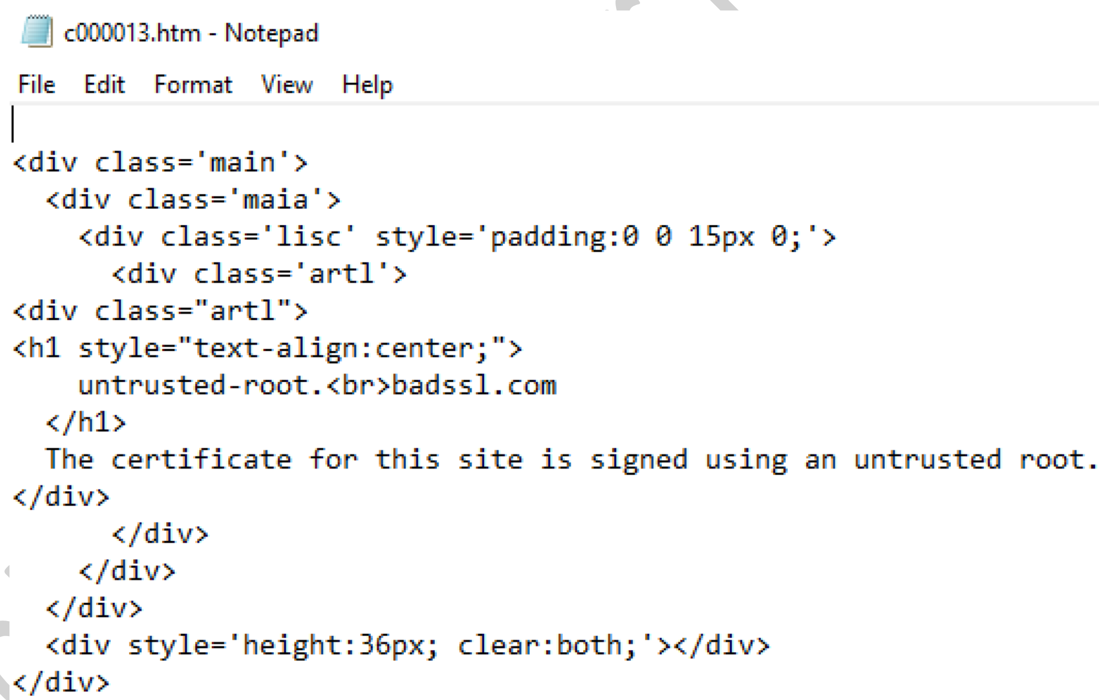
This screenshot shows the test URLs loaded into **oLink**:



Next, the pages were retreived using the **Retrieve** button:



The contents were saved locally; in this case the **https** page with untrusted certificate was saved in **C:\oLink\Site\c000013.htm**:




```
<div class='main'>
  <div class='maia'>
    <div class='lisc' style='padding:0 0 15px 0;'>
      <div class='artl'>
<div class="artl">
<h1 style="text-align:center;">
    untrusted-root.<br>badssl.com
  </h1>
  The certificate for this site is signed using an untrusted root.
</div>
      </div>
    </div>
  </div>
  <div style='height:36px; clear:both;'></div>
</div>
```

Recommended Remediation:

The assessment team recommends only allowing downloads over **HTTPS** from hosts that present a valid and trusted certificate. If downloading over a plaintext **HTTP** connection, or from hosts using invalid certificates is required, the **oLink** application could present the user with a warning of the risks of attack assoicated with the affected user-supplied URLs.

L1: Application Targets Deprecated .NET Version

Description:

The **oLink** application was found to target a deprecated version of the .NET Framework. The application targeted .NET version 4.5, for which support ended in January 2016. .NET Framework versions 4.5.2, 4.6, and 4.6.1 are scheduled to end support in April 2022.

Impact:

Versions of the .NET Framework that are no longer supported no longer receive security updates or support. Therefore, software targeting deprecated versions could be exposed to security vulnerabilities in the supporting libraries that will not be addressed.

Reproduction:

The **oLink** project file referenced the target .NET Framework version as **v4.5** in **olink/oLink/oLink.csproj**, line 11:

```
11      <TargetFrameworkVersion>v4.5</TargetFrameworkVersion>
```

The **App.config** file also referenced version **v4.5** in olink/oLink/App.config, line 7:

```
7          <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
```

Recommended Remediation:

The assessment team recommends migrating the **oLink** application to .NET Framework version 4.6.2 or higher.

L2: User Manual Recommends Creating Programmatic AWS User with Administrative Permissions

Description:

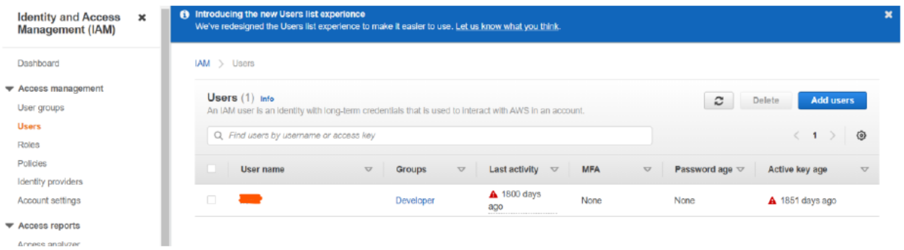
The **oLink** user manual recommended that users create an **AWS IAM** programmatic user with full administrator access permissions.

Impact:

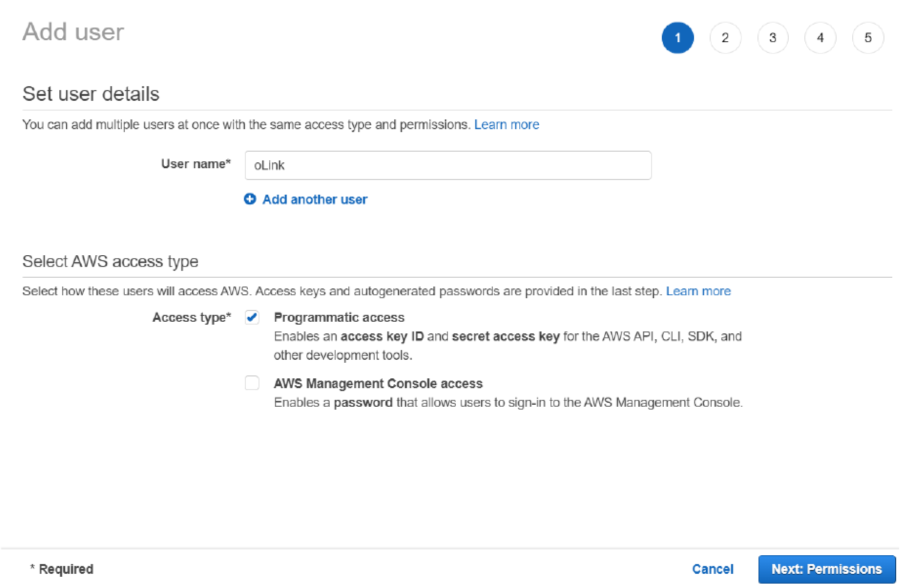
If an attacker is able to discover the administrative programmatic user's credentials, for example by extracting them from the database of a machine where **oLink** has been used, they could gain full access to the **AWS** account, poentially exposing other sensitive information, or costing money by allocating **AWS** resources.

Reproduction:

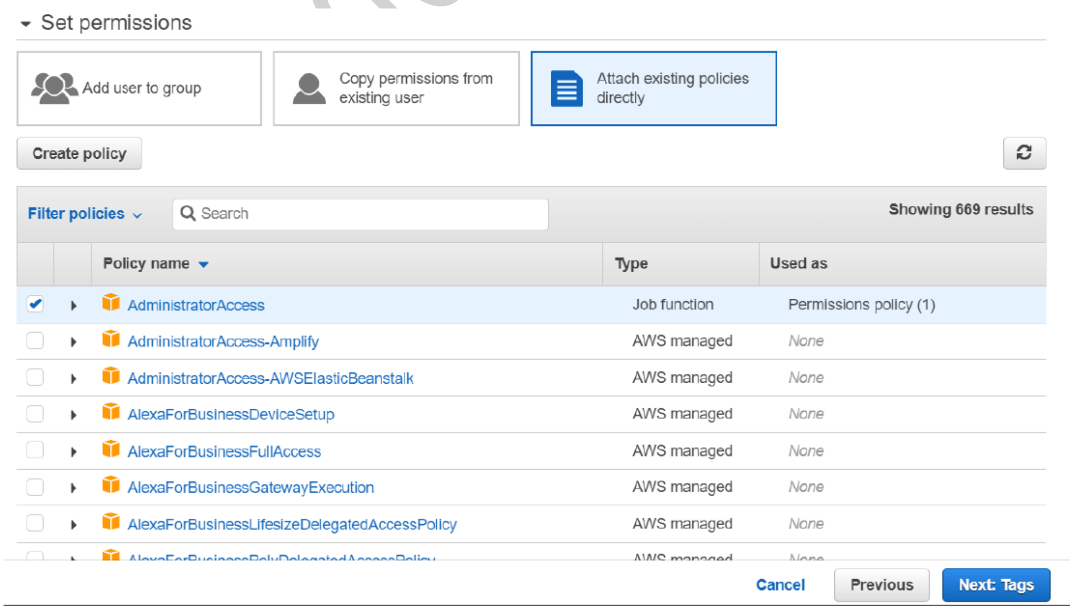
The instructions to grant administrative access to the **AWS IAM** user are depicted in pages 20-21 of the manual:



4. Key in “User Name”, Tick Access type: “Programmatic access”, then click “Next Permissions”



5. Select “Attach existing policies directly”, tick “AdministratorAccess”, then tick “Next Tags” .



6. Click “Next: Review”

Recommended Remediation:

The assessment team recommends that **oLink** users create programmatic **AWS** users with the least privilege necessary. The manual could instruct the user to create an **IAM** policy that grants write access only to the bucket in the region that will be used by **oLink**.

L3: SQL Server Database Complexity

Description:

The **oLink** application used a Microsoft SQL Server database to store **AWS** credentials as well as lists of URLs of sites that had been mirrored using the application, and URLs of assets that had been downloaded associated with those sites. The dependency on SQL Server adds unneccesary complexity to installing and using **oLink**, as well to securing data. In additon, though SQL injection vulnerabilites were not identified, the application uses unsafe practices to access the database making those vulnerabilites potentially more likely in the future.

Impact:

Using SQL Server for storing data creates the risk of attackers gaining access to sensitive data, or injecting malicious data via other applications running on the same host, or depending on how the database server is configured, remote connections. The use of SQL introduces the risk of SQL injection issues. Access to the database could also lead to code execution (see the finding **Operating System Command Injection**).

Reproduction:

One example of *oLink*'s database usage is the storage of **AWS** credentials. The SQL strings related to **AWS** credentials are in **olink/oLink/ooData.cs**, lines 11-12:

```
11          static public string sG10配置Set = @"update [oLink].[dbo].[G10配置] set
S3Id=N'^S3Id^',S3Key=N'^S3Key^',S3Bucket=N'^S3Bucket^',Name=N'^Name^',Note=N'^Note^'";
12          static public string sG10配置Get = @"SELECT S3Id,S3Key,S3Bucket,Name,Note FROM [oLink].[dbo].[G10配置]";
```

The code to store credentials uses the **Replace** method to replace placeholders in the **SQL** string with parameters:

olink/oLink/FormLink.cs, lines 143-156:

```
143         private void button2_Click(object sender, EventArgs e)
144         {
145             try
146             {
147                 string s1 = ooData.sG10配置Set
148                     .Replace("^S3Id^", GetSqlParam(textBoxS3Id.Text.Trim()))
149                     .Replace("^S3Key^", GetSqlParam(textBoxS3Key.Text.Trim()))
150                     .Replace("^S3Bucket^", GetSqlParam(textBoxS3Bucket.Text.Trim()))
151                     .Replace("^Name^", GetSqlParam(textBoxName.Text.Trim()))
152                     .Replace("^Note^", GetSqlParam(textBoxNote.Text.Trim()));
153                 ExecuteSQL(s1);
154             }
155             catch (Exception ex) { Log(MethodBase.GetCurrentMethod().Name + ": " + ex.Message); }
156         }
```

The **GetSqlParam** method escapes singlequotes in input strings, in order to prevent injection; it is defined in **olink/oLink/FormLink.cs**, lines 1812-1815:

```
1812         static public string GetSqlParam(string s)
1813         {
1814             return s.Replace("'", "''");
1815         }
```

Recommended Remediation:

The assessment team recommends using a simpler library or format for storing sensitive and other persistent user data on disk. In addition, the assessment team recommends using Microsoft's Data Protection API to encrypt sensitive data stored on disk.

In additon, wherever SQL is used, the assessment team recommends using parameterized SQL queries rather than string concatenation to build SQL statements throughout applications. This technique enforces separation between the structure of the SQL statement and the data it uses. Each SQL statement can still be defined with placeholders for data to be supplied at runtime, with the database library providing the escaping and placeholder replacing in a robust manner.

L4: Arbitrary File Download

Description:

When the **oLink** applicaiton retrieves a web page, it downloads HTML as well as assets associated with that page, such as image files and videos. The application does not validate these assets are safe or have any known format.

Impact:

An attacker who controls or has compromised a site being processed by **oLink** could cause **oLink** to download malicious files. This colud be used as a vector for introducing malicious code to a host running **oLink** or to **S3** as part of a larger attack.

Reproduction:

To test this vulnerability, a test document was hosted alongside a Windows executable file named **example.exe**. This was the test document:

```
<title>Title</title>
<img src="example.exe">
```

The document was retrieved using **oLink**, which resulted in **example.exe** being downloaded and stored at **C:\oLink\File\000011.**.

The code that downloads HTML documents as well as other assets is named **DownloadHtml**. **DownloadHtml** is in **olink/oLink/FormLink.cs**, lines 1687-1740:

```
1687        public bool DownloadHtml(string name, string host, string referer, string sFileName)
1688        {
1689            HttpWebRequest request2 = null;
1690            HttpWebResponse response2 = null;
1691            try
1692            {
1693                request2 = (HttpWebRequest)WebRequest.Create(name);
1694                if (host != "") request2.Host = host;
1695                if (referer != "") request2.Referer = referer;
1696                response2 = (HttpWebResponse)request2.GetResponse();
1697                if (response2.StatusCode != HttpStatusCode.OK)
1698                {
1699                    if (response2 != null) { response2.Close(); response2 = null; }
1700                    if (request2 != null) request2 = null;
1701                    return false;
1702                }
1703                if (File.Exists(sFileName))
1704                {
1705                    FileInfo fi = new FileInfo(sFileName);
1706                    if (response2.ContentLength == -1 || fi.Length == response2.ContentLength)
1707                    {
```

```
                ///BeginInvoke(new ShowMsgDelegate(ShowMsg), new object[] { "Skip" });
                if (response2 != null) { response2.Close(); response2 = null; }
                if (request2 != null) request2 = null;
                return true;
            }
        }
        ShowMsgD("Downloading: " + sFileName);

        byte[] buffer = new byte[8 * 1024];
        Stream outStream = File.Create(sFileName);
        Stream inStream = response2.GetResponseStream();
        long length = response2.ContentLength;
        long total = 0;
        int l = 0;
        while ((l = inStream.Read(buffer, 0, buffer.Length)) > 0)
        {
            total += l;
            int progress = (int)(((float)total / length) * 100);
            BeginInvoke(new ShowMsgDelegate(ShowLabel), new object[] { progress + " %" });
            outStream.Write(buffer, 0, l);
        }
        outStream.Close();
        if (response2.ContentLength != -1 && length != total) { }
        //BeginInvoke(new ShowMsgDelegate(ShowLabel), new object[] { "Done" });
        if (response2 != null) { response2.Close(); response2 = null; }
        if (request2 != null) request2 = null;
        return true;
    }
    catch (Exception ex) { }
    if (response2 != null) { response2.Close(); response2 = null; }
    if (request2 != null) request2 = null;
    return false;
}
```

The code makes an HTTP request, downloads the file to a buffer, and saves the buffered data directly to disk.

Recommended Remediation:

The assessment team recommends verifying the downloaded files are of the expected filetype, for example HTML, image, video, or audio files, before saving them to disk.

I1: Unused Elements in Production Codebase

Description:

A number of methods in the **oLink** application's codebase were defined but never invoked or used. Unused code elements can provide attackers with insight into future functionality, or indicate that legacy code is being released into the production environment.

Impact:

Unused code can increase application attack surface, as an attacker could try to insert a malicious feature with the name of an unused element to make the code function improperly. In addition, much of the unused code, if used unsafely in future could expose the application to additional vulnerabilities, for example more ways for malicious or compromised web sites to inject malicious data into **oLink** output.

Numerous methods were discovered that were defined but never referenced are listed here:

- GetSoundPlayer
- GetFlashPlayer
- GetImagePlayer
- GetDownloadPlayer
- GetTwitter

- GetTxt部分
- GetString千万
- GetString最长
- Get一行
- Get时距Param
- GetPict中
- GetPictParam
- Get时间
- HtmlEn
- RSAEncrypt
- RSADecrypt
- ShowLabelD
- WriteMsg
- PostHtml
- GetHtmlMethod (This method has several overloaded definitions; three out of five are unused).
- GetHtmlMethodOrigin
- CheckHtml

In addition, it was noted that much of the **GetVideoPlayer** method in particular was unreachable because it was in an **if**-statement block with a condition that could never be evaluated as **true**.

Reproduction:

The following screenshots of the **oLink** source code in **Visual Studio** show many methods that have zero references:

```
0 references
static private string GetSoundPlayer(string url, string cover = "")...

1 reference
static private string GetM3u8Player(string url)...

0 references
static private string GetFlashPlayer(string url)...

1 reference
static private string GetImagePlayerTop(string url)...

0 references
static private string GetImagePlayer(string url)...

0 references
static private string GetDownloadPlayer(string url)...

0 references
static private string GetTwitter(string co, string cover, string title)...
```

```
0 references
static private string GetTxt部分(string s摘要, string s链接, int i长度)[...]
0 references
static private string GetString千万(string page)[...]
0 references
static private string GetString最长(string sIn, int iLen)[...]
0 references
static private string Get一行(string sIn)[...]
1 reference
static private string Get换行(string sIn)[...]
0 references
static private string Get时距Param(DateTime dt)[...]
0 references
static private string GetPict中(string sPict)[...]
0 references
static private string GetPictParam(string sPict, string sNumb)[...]
0 references
static private string Get时间(string s)[...]

0 references
static public string HtmlEn(string input, string password)[...]

0 references
static public string RSAEncrypt(string publickey, string content)[...]

0 references
static public string RSADecrypt(string privatekey, string content)[...]

// Show
private delegate void ShowMsgDelegate(string msg);
2 references
private void ShowLabel(string msg)[...]
0 references
private void ShowLabelD(string msg)[...]
```

```
0 references
static private void WriteMsg(string message)[...]

1 reference
static private void WriteErr(string message)[...]
2 references
private void SaveFile(string message, string sFile)[...]

// Html Utility
8 references
static public string GetHtml(string sName, bool bFail = false)[...]

0 references
static public string PostHtml(string sName, string sData)[...]

4 references
static public string GetHtmlMethod(string sMethod, string sName, string sData, string sHeader, string sReferer,
    string sCode)[...]

1 reference
static public string GetHtmlMethod(string sMethod, string sName, string sData, string sHeader, string sReferer,
    string sCode, string sContentType, out string sCookie)[...]

0 references
static public string GetHtmlMethod(string sMethod, string sName, string sData, string sHeader, string sReferer,
    string sContentType, string sHost, string sCode)[...]

0 references
static public string GetHtmlMethod(string sMethod, string sName, string sData, string sHeader, string sReferer,
    string sContentType, string sCode)[...]

0 references
static public string GetHtmlMethod(string sMethod, string sName, string sData, string sHeader, string sReferer)[...]

0 references
static public string GetHtmlMethodOrigin(string sName)[...]

1 reference
public bool DownloadHtml(string name, string host, string referer, string sFileName)[...]

0 references
public bool CheckHtml(string sName)[...]
```

The unreachable code in the **GetVideoPlayer** method can be identified by first noting that the method is only called twice, in **olink/oLink/FormLink.cs**, lines 1032-1033:

olink/oLink/FormLink.cs, lines 1028-1033:

```
1028                         coMedia = GetFile(coMedia);
1029                         if (coMedia.EndsWith(".jpg") || coMedia.EndsWith(".png") || coMedia.EndsWith(".jpeg")
1030                              || coMedia.EndsWith(".gif") || coMedia.EndsWith(".webp")) co = GetImagePlayerTop(coMedia) +
"\r\n" + co;
1031                         else if (coMedia.EndsWith(".mp3")) co = GetAudioPlayer(coMedia, cover) + "\r\n" + co;
1032                         else if (coMedia.EndsWith(".mp4")) co = GetVideoPlayer(coMedia, "", cover) + "\r\n" + co;
1033                         else co = GetVideoPlayer(coMedia, "", cover) + "\r\n" + co;
```

In both cases, the first parameter comes from **coMedia**, which is return value from **GetFile**, which will only return a numerical file name, and the second parameter (**track**) is set to an empty string.

The **GetVideoPlayer** method itself is defined in **olink/oLink/FormLink.cs**, lines 405-639:

```
405         static public string GetVideoPlayer(string url, string track = "", string cover = "", string myip = "", bool
bDownload = true)
406         {
407             if (url == "") return "";
408             if (url.StartsWith("https://player.vimeo.com/video/")) url = url.Replace("https://player.vimeo.com/",
"https://vimeo.com/");
409             if (url.StartsWith("https://www.youtube.com/") || url.StartsWith("https://www.youtube.com/embed/"))
410                 url = url.Replace("https://www.youtube.com/watch?v=",
"https://youtu.be/").Replace("https://www.youtube.com/embed/", "https://youtu.be/");
411
412             string sVideoL = ""; string sVideoM = ""; string sVideoH = ""; string sVideoV = ""; string sVideoM2 = "";
string sVideoV2 = "";
413             string sAudio140 = ""; string sAudio171 = ""; string sAudio249 = ""; string sAudio250 = ""; string
sAudio251 = "";
414             string sTrack = "";
415             string sTrackZh = "", sTrackEn = "";
416             string sTrackZhSrc = "", sTrackEnSrc = "";
417             if (track != "" && track.EndsWith(".vtt"))
418             {
...                 [unreachable code]
422             }
423
424             if (url.StartsWith("https://www.youtube.com/") || url.StartsWith("https://youtu.be/") ||
url.StartsWith("https://www.youtube.com/embed/"))
425             {
...                 [unreachable code]
535             }
536
...
639         }
```

Recommended Remediation:

The assessment team suggests reviewing the codebase to eliminate unused and legacy code from the production codebase.

Additionally, the assessment team suggests keeping two branches of the **oLink** source: one for releases and another for development. Then unused code and test features can be removed from the release branch to minimize attack surface.

I2: Owner of S3 Bucket May Be Discoverable

Description:

An attacker could be able to discover the **AWS** account owner of an **S3** bucket used to mirror websites using the **oLink** tool.

Impact:

The **oLink** application depends on **AWS S3** to host the contents of mirrored web pages. If an attacker is able to identify the accounts used by **oLink** users, it may help the attacker identify **oLink** users.

Reproduction:

An **AWS** account was not provided as part of the assessment scope, so this attack was not attempted. However, public tools exist to identify accounts that buckets **S3** buckets belong to, for example https://github.com/WeAreCloudar/s3-account-search.

Recommended Remediation:

Since hosting articles on **S3** is a significantly fundamental feature of **oLink**, the assessment team recommends analyzing and acknowledging any potential risk of the owners of **oLink** associated **S3** buckets being discoverable.

I3: Application Third Party Service Dependencies

Description:

The **oLink** application depends on several third party services, compromise of which could compromise the security of **oLink** users as well as those who visit mirrored sites generated by **oLink**.

Impact:

Since **oLink** uploads mirrored sites to Amazon **S3**, **AWS** is of course a third party service that must be trusted. In addition, however, **oLink** relies on these services:

- **jsdelivr.net** is a JavaScript hosting service. Compromise of this service could allow an attacker to inject JavaScript code into mirrored sites generated by **oLink**, modifying the rendered contents of those sites, and identifying visitors of those sites to the attacker.
- **is.gd** is a URL Shortening service. Compromise of this service could allow an attacker to identify users of **oLink** and identify visitors of short links, as well as redirect users to malicious sites intstead of sites mirrored by **oLink**.

Finally, this service is referenced by currently unused code (see the finding **Unused Elements in Production Codebase**):

- **or9a.odisk.org**

Reproduction:

References to **jsdelivr.net** are in **Site/show.htm**, lines 8 and 274-275:

```
8    <script src="https://cdn.jsdelivr.net/jquery/1.12.4/jquery.min.js"></script>
...
274    <link  href="https://cdn.jsdelivr.net/npm/video.js@7.5.4/dist/video-js.min.css" rel="stylesheet">
275    <script src="https://cdn.jsdelivr.net/npm/video.js@7.5.4/dist/video.min.js"></script>
```

The **is.gd** URL shortener is used in **olink/oLink/FormLink.cs**, lines 391-402:

```
391         private void Generate()
392         {
393             try
394             {
395                 string url = "https://s3.amazonaws.com/^S3Bucket^/Site/show.htm?ag=olHome&pin=^random^#olHome"
396                     .Replace("^S3Bucket^", textBoxS3Bucket.Text.Trim())
397                     .Replace("^random^", GetRandom());
398                 string s = GetHtmlMethod("GET", "https://is.gd/create.php?format=simple&url=" + EnUrlSymbol(url),
```

```
"", "", "", "");
                    ShowMsgD(s);
                }
                catch (Exception ex) { Log(MethodBase.GetCurrentMethod().Name + ": " + ex.Message); }
            }
```

The **or9a.odisk.org** dependency is referenced in this code from **olink/oLink/FormLink.cs**, lines 442-454:

```
442                 Match mM3u8 = new Regex("(?<=%22hlsManifestUrl%22%3A%22)([\\S\\s]*?)(?=%22)").Match(s1);
443                 if (mM3u8.Success)
444                 {
445                     string sM3u8 = GetHtml(mM3u8.Value.Replace("%3A", ":").Replace("%2F", "/").Replace("%252C",
",").Replace("%253D", "="));
446                     Match mM3u8_360 = new Regex("(?<=RESOLUTION=640x[\\S\\s]*?)(https[\\S]*?)(?=\\s)").Match(sM3u8);
447                     if (!mM3u8_360.Success) return sNFound;
448                     string m3u8 = GetHtml(mM3u8_360.Value);
449                     string[] m3u8s = m3u8.Split('\n'); //Log(url + " " + m3u8s.Length+" "+ mM3u8_360.Value);
450                     if (m3u8s.Length > 100 || m3u8 == "") return sConvert;
451                     sM3u8 = "http://or9a.odisk.org/oo.aspx?name=get_m3u8&ag=" +
HttpUtility.UrlEncode(mM3u8_360.Value)
452                         + "&myip=" + myip + "&type=play.m3u8";
453                     return GetM3u8Player(sM3u8);
454                 }
```

Recommended Remediation:

The assessment team recommends removing dependencies if possible. The **jsdelivr.net** dependency could be easily removed by hosting JavaScript dependencies alongside mirrored sites in **S3**. The use of a URL shortener could be made optional, and the use of **is.gd** in particular could be evaluated. The **or9a.odisk.org** reference can be removed by deleting unused code.