HTTP Header Injections: A Splitting Headache

Sofia Lindqvist

Binary Security

Oslo, October 3rd 2025

BINARY SECURITY

Whoami

- PhD in pure maths
- Worked three years as a developer at Cisco
- Worked as a pentester for three years (at Binary Security since October 2023)
- Contact: sofia@binarysecurity.no

Outline

- Introduction to basic concepts
  - HTTP requests
  - CRLF injection
  - Header injection
  - Request splitting/smuggling
- DEMO
- Impact
- Hunting in open source code
- DEMO with real library
- Hunting in azure-sdk-for-net
- Conclusions

BINARY SECURITY

A typical HTTP/1.1 request

```
GET /about/ HTTP/1.1\r\n
Host: binarysecurity.no\r\n
Accept-Language: en-US\r\n
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64;
x64) AppleWebKit/537.36 (KHTML, like Gecko) Chro
me/126.0.6478.127 Safari/537.36\r\n
Accept: text/html,application/xhtml+xml,applic
ation/xml;q=0.9,image/avif,image/webp,image/apng,
*/*;q=0.8,application/signed-exchange;v=b3;
q=0.7\r\n
Accept-Encoding: gzip, deflate, br\r\n
\r\n
```

A (shorter) typical HTTP/1.1 request

```
GET /about/ HTTP/1.1 \r\n
Host: binarysecurity.no \r\n
Accept-Language: en-US \r\n
Accept: text/html \r\n
Accept-Encoding: gzip, deflate, br \r\n
 \r\n
```

BINARY SECURITY

A (shorter) typical HTTP/1.1 request

```
GET /about/ HTTP/1.1\r\n
Host: binarysecurity.no\r\n
Accept-Language: en-US\r\n
Accept: text/html\r\n
Accept-Encoding: gzip, deflate, br\r\n
\r\n
```

CRLF

- Carriage return and Line Feed
- CRLF
- \r\n
- ASCII: 13 and 10 (decimal), 0x0D and 0x0A (hexadecimal)

A (shorter) typical HTTP/1.1 request

```
GET /about/ HTTP/1.1\r\n
Host: binarysecurity.no\r\n
Accept-Language: en-US\r\n
Accept: text/html\r\n
Accept-Encoding: gzip, deflate, br\r\n
\r\n
```

A (shorter) typical HTTP/1.1 request

```
GET /about/ HTTP/1.1\r\n
Host: binarysecurity.no\r\n
Accept-Language: en-US\r\n
Accept: text/html\r\n
Accept-Encoding: gzip, deflate, br\r\n
\r\n
```

A (shorter) typical HTTP/1.1 request

```
GET /about/ HTTP/1.1\r\n
Host: binarysecurity.no\r\n
Accept-Language: en-US\r\n
Accept: text/html\r\n
Accept-Encoding: gzip, deflate, br\r\n
\r\n
```

BINARY SECURITY

A (shorter) typical HTTP/1.1 request

```
GET /about/ HTTP/1.1\r\n
Host: binarysecurity.no\r\n
Accept-Language: en-US\r\n
Accept: text/html\r\n
Accept-Encoding: gzip, deflate, br\r\n
\r\n
```

More about HTTP Headers

- A header *name* may consist of any printable ASCII characters except for whitespace and ()[]⟨⟩{}\@“/?=,:;

More about HTTP Headers

- A header *name* may consist of any printable ASCII characters except for whitespace and ()[]⟨⟩{}\@“/?=,:;
- A header *value* may consist of any non-control ASCII characters

More about HTTP Headers

- A header *name* may consist of any printable ASCII characters except for whitespace and ()[]⟨⟩{}\@“/?=,:;
- A header *value* may consist of any non-control ASCII characters
- Notably: CR and LF characters are not allowed in either*

When things go wrong

```
GET /about/ HTTP/1.1\r\n
Host: binarysecurity.no\r\n
Some-Header: <user controllable value> \r\n
\r\n
```

BINARY SECURITY

When things go wrong

```
GET /about/ HTTP/1.1\r\n
Host: binarysecurity.no\r\n
Some-Header: <user controllable value> \r\n
\r\n
```

```
bla\r\nInjected-Header: it works!
```

When things go wrong

```
GET /about/ HTTP/1.1\r\n
Host: binarysecurity.no\r\n
Some-Header: bla\r\n
Injected-Header: injected\r\n
\r\n
```

Request splitting/smuggling

- Instead of injecting a header, inject a whole request

BINARY SECURITY

When things go wrong cont.

```
GET /about/ HTTP/1.1\r\n
Host: binarysecurity.no\r\n
Some-Header: <user controllable value> \r\n
\r\n
```

When things go wrong cont.

```
GET /about/ HTTP/1.1\r\n
Host: binarysecurity.no\r\n
Some-Header: <user controllable value> \r\n
\r\n
```

```
bla\r\n\r\nGET /smuggled HTTP/1.1\r\nHost:
binarysecurity.no
```

BINARY SECURITY

When things go wrong cont.

```
GET /about/ HTTP/1.1\r\n
Host: binarysecurity.no\r\n
Some-Header: bla\r\n
\r\n
GET /smuggled HTTP/1.1\r\n
Host: binarysecurity.no\r\n
\r\n
```

BINARY SECURITY

When things go wrong cont.

```
GET /about/ HTTP/1.1\r\n
Host: binarysecurity.no\r\n
Some-Header: bla\r\n
\r\n


GET /smuggled HTTP/1.1\r\n
Host: binarysecurity.no\r\n
\r\n
```

BINARY SECURITY

Demo: basic request splitting

Setup:

- Apache server serving various hosts
- `internal.site` can only be accessed from localhost
- `basicdemo.site` makes a request to `http://internal.site/status.html` and returns the resulting status code
- the page `http://basicdemo.site` takes an optional query parameter name
    - If the parameter is present, it is included in a custom header in the request to `http://internal.site/status.html`

Sidenote: why this works

- HTTP/1.1 has a feature called *HTTP Pipelining*
- Multiple HTTP requests sent on a single TCP connection without waiting for corresponding responses
- The only way the server can know where one request ends and the next begins is by looking for CRLFCRLF, or computing a content length (for requests with bodies)

Impact of request splitting

- Server intends to make a single request, but the attacker causes it to make multiple

BINARY SECURITY

Impact of request splitting

- Server intends to make a single request, but the attacker causes it to make multiple
- Bypass internal access controls

Impact of request splitting

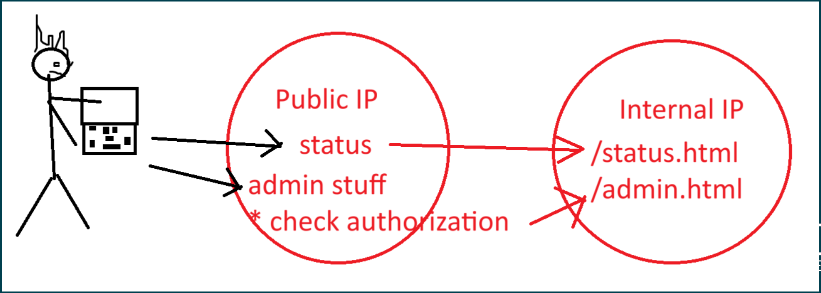
- Server intends to make a single request, but the attacker causes it to make multiple
- Bypass internal access controls
- Bypass client authentication

Impact of request splitting

- Server intends to make a single request, but the attacker causes it to make multiple
- Bypass internal access controls
- Bypass client authentication
- Combine with redirects

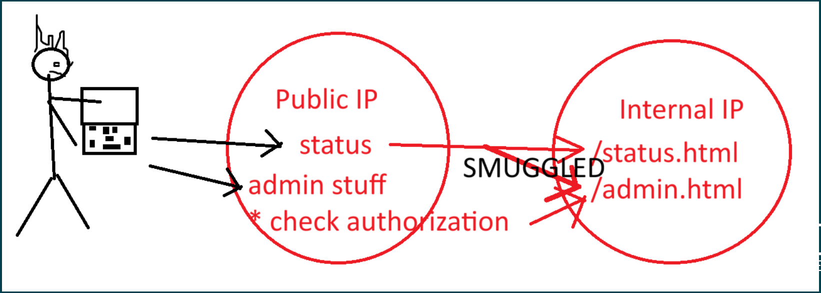
Impact: example

- Microservice architecture
- Lots of microservices running internally on the same host
- Requests from the internet go through an API gateway that checks authentication and authorization, and then forwards the request to the appropriate microservice
- In the demo, we could access an internal admin endpoint
- Even if we can't see the response, let's say there is an endpoint to add a new admin user



Impact: example

- Microservice architecture
- Lots of microservices running internally on the same host
- Requests from the internet go through an API gateway that checks authentication and authorization, and then forwards the request to the appropriate microservice
- In the demo, we could access an internal admin endpoint
- Even if we can't see the response, let's say there is an endpoint to add a new admin user



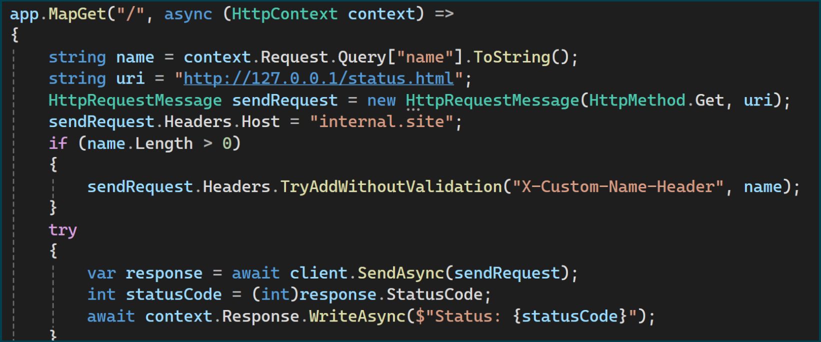
The Plan

- Pick something *really* specific in code that allows CRLF injections
    - ... and thus header injections and request splitting

The Plan

- Pick something *really* specific in code that allows CRLF injections
  - ... and thus header injections and request splitting
- Search through open source repos and see what we find

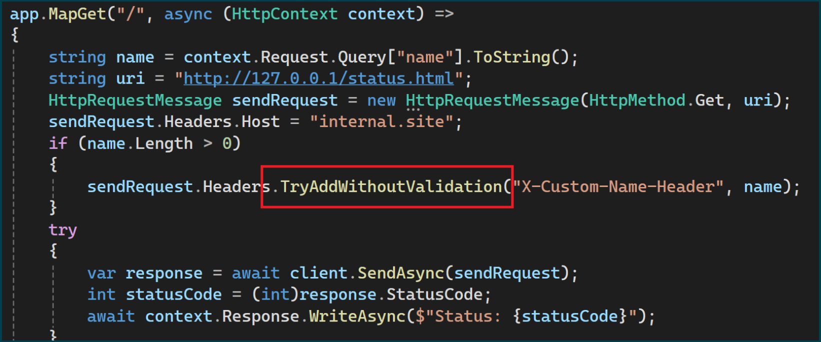
Vulnerable code (C#)



```
app.MapGet("/", async (HttpContext context) =>
{
    string name = context.Request.Query["name"].ToString();
    string uri = "http://127.0.0.1/status.html";
    HttpRequestMessage sendRequest = new HttpRequestMessage(HttpMethod.Get, uri);
    sendRequest.Headers.Host = "internal.site";
    if (name.Length > 0)
    {
        sendRequest.Headers.TryAddWithoutValidation("X-Custom-Name-Header", name);
    }
    try
    {
        var response = await client.SendAsync(sendRequest);
        int statusCode = (int)response.StatusCode;
        await context.Response.WriteAsync($"Status: {statusCode}");
    }
```

BINARY SECURITY

Vulnerable code (C#)



```
app.MapGet("/", async (HttpContext context) =>
{
    string name = context.Request.Query["name"].ToString();
    string uri = "http://127.0.0.1/status.html";
    HttpRequestMessage sendRequest = new HttpRequestMessage(HttpMethod.Get, uri);
    sendRequest.Headers.Host = "internal.site";
    if (name.Length > 0)
    {
        sendRequest.Headers.TryAddWithoutValidation("X-Custom-Name-Header", name);
    }
    try
    {
        var response = await client.SendAsync(sendRequest);
        int statusCode = (int)response.StatusCode;
        await context.Response.WriteAsync($"Status: {statusCode}");
    }
```

BINARY SECURITY

Searching in GitHub

- GitHub
  - There are 19720* hits for `TryAddWithoutValidation` in C# code on GitHub
  - Hits are spread across 4518 unique repositories

Searching in GitHub

- GitHub
    - There are 19720* hits for `TryAddWithoutValidation` in C# code on GitHub
    - Hits are spread across 4518 unique repositories
- NuGet public packages
    - There are 677407* publicly available NuGet packages
    - Can get download counts for 380000 of them
    - Downloaded the 5000 most popular public NuGet packages
    - There were *121* packages with hits for `TryAddWithoutValidation` among these

BINARY SECURITY

Searching in GitHub

- GitHub
    - There are 19720* hits for `TryAddWithoutValidation` in C# code on GitHub
    - Hits are spread across 4518 unique repositories
- NuGet public packages
    - There are 677407* publicly available NuGet packages
    - Can get download counts for 380000 of them
    - Downloaded the 5000 most popular public NuGet packages
    - There were *121* packages with hits for `TryAddWithoutValidation` among these
- Checked 50+30* GitHub results and all 121 NuGet hits

BINARY SECURITY

Results

- Command line applications (not that interesting)

Results

- Command line applications (not that interesting)
- Ancient/unused repositories/packages

Results

- Command line applications (not that interesting)
- Ancient/unused repositories/packages
- APIs/libraries that expose vulnerable methods

BINARY SECURITY

Results

- Command line applications (not that interesting)
- Ancient/unused repositories/packages
- APIs/libraries that expose vulnerable methods
    - RestSharp

BINARY SECURITY

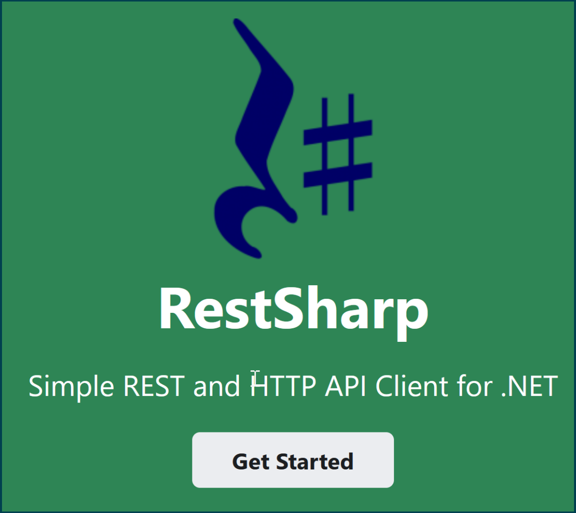
Results

- Command line applications (not that interesting)
- Ancient/unused repositories/packages
- APIs/libraries that expose vulnerable methods
  - RestSharp
  - Refit

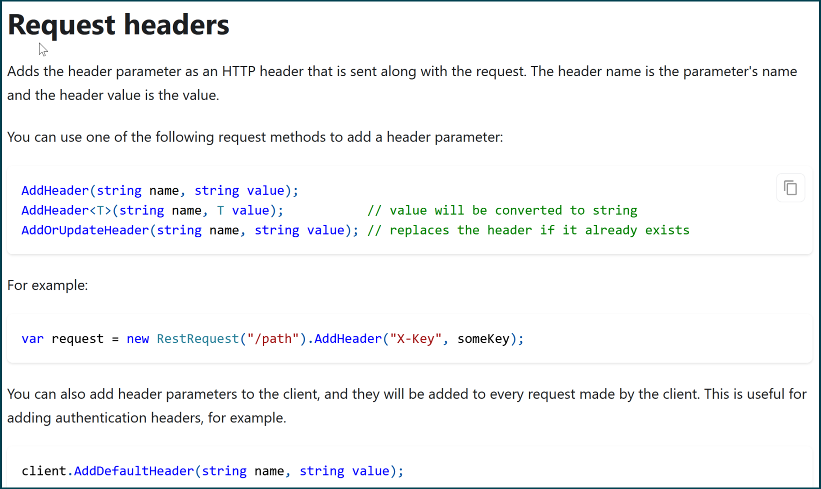
Results

- Command line applications (not that interesting)
- Ancient/unused repositories/packages
- APIs/libraries that expose vulnerable methods
  - RestSharp
  - Refit
- Microsoft/Azure SDKs

RestSharp (https://restsharp.dev/)



RestSharp Documentation



Request headers

Adds the header parameter as an HTTP header that is sent along with the request. The header name is the parameter's name and the header value is the value.

You can use one of the following request methods to add a header parameter:

```
AddHeader(string name, string value);
AddHeader<T>(string name, T value);           // value will be converted to string
AddOrUpdateHeader(string name, string value); // replaces the header if it already exists
```

For example:

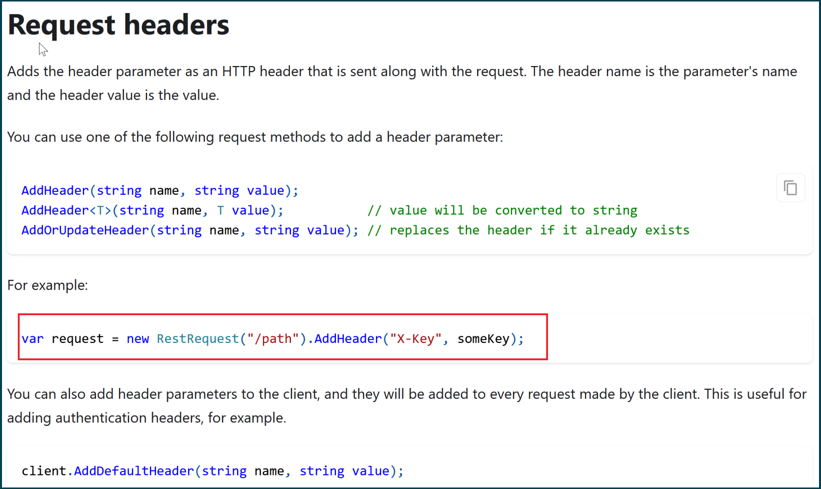
```
var request = new RestRequest("/path").AddHeader("X-Key", someKey);
```

You can also add header parameters to the client, and they will be added to every request made by the client. This is useful for adding authentication headers, for example.

```
client.AddDefaultHeader(string name, string value);
```

BINARY SECURITY

RestSharp Documentation



Request headers

Adds the header parameter as an HTTP header that is sent along with the request. The header name is the parameter's name and the header value is the value.

You can use one of the following request methods to add a header parameter:

```
AddHeader(string name, string value);
AddHeader<T>(string name, T value);           // value will be converted to string
AddOrUpdateHeader(string name, string value); // replaces the header if it already exists
```

For example:

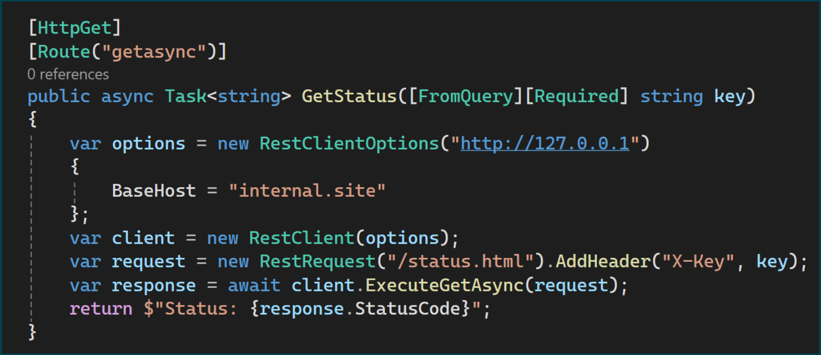
```
var request = new RestRequest("/path").AddHeader("X-Key", someKey);
```

You can also add header parameters to the client, and they will be added to every request made by the client. This is useful for adding authentication headers, for example.

```
client.AddDefaultHeader(string name, string value);
```

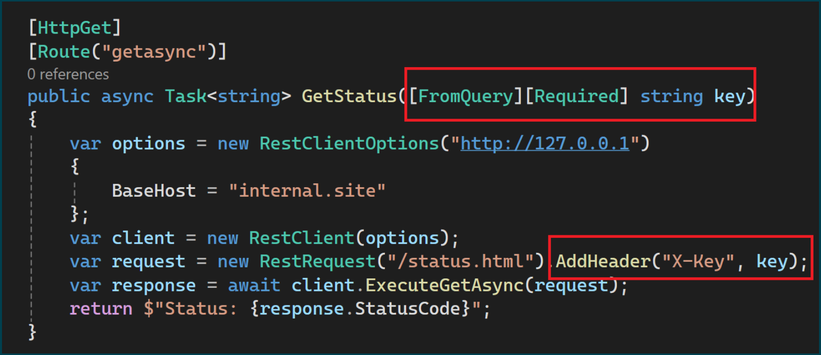
BINARY SECURITY

RestSharp Demo


```
[HttpGet]
[Route("getasync")]
0 references
public async Task<string> GetStatus([FromQuery][Required] string key)
{
    var options = new RestClientOptions("http://127.0.0.1")
    {
        BaseHost = "internal.site"
    };
    var client = new RestClient(options);
    var request = new RestRequest("/status.html").AddHeader("X-Key", key);
    var response = await client.ExecuteGetAsync(request);
    return $"Status: {response.StatusCode}";
}
```

BINARY SECURITY

RestSharp Demo



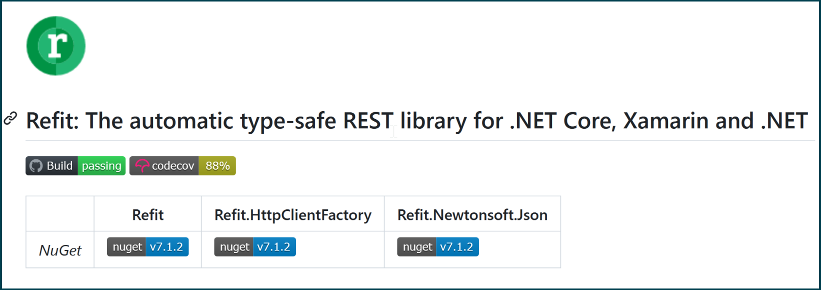
```
[HttpGet]
[Route("getasync")]
0 references
public async Task<string> GetStatus([FromQuery][Required] string key)
{
    var options = new RestClientOptions("http://127.0.0.1")
    {
        BaseHost = "internal.site"
    };
    var client = new RestClient(options);
    var request = new RestRequest("/status.html").AddHeader("X-Key", key);
    var response = await client.ExecuteGetAsync(request);
    return $"Status: {response.StatusCode}";
}
```

BINARY SECURITY

Fallout

- Assigned CVE-2024-45302 (CVSSv3.1: 7.8 High)
- Fixed in RestSharp v112.0.0

Refit

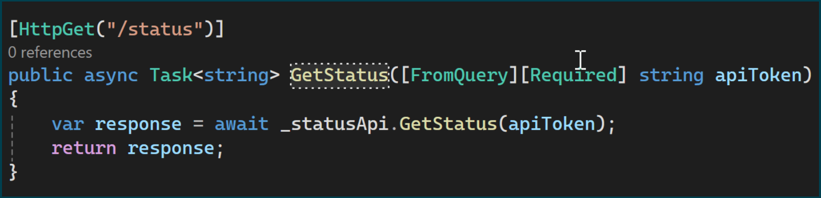

Refit: The automatic type-safe REST library for .NET Core, Xamarin and .NET

Build passing | codecov 88%

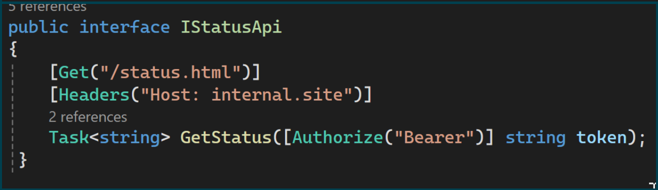
	Refit	Refit.HttpClientFactory	Refit.Newtonsoft.Json
NuGet	nuget v7.1.2	nuget v7.1.2	nuget v7.1.2

BINARY SECURITY

Refit



```
[HttpGet("/status")]
0 references
public async Task<string> GetStatus([FromQuery][Required] string apiToken)
{
    var response = await _statusApi.GetStatus(apiToken);
    return response;
}
```



```
5 references
public interface IStatusApi
{
    [Get("/status.html")]
    [Headers("Host: internal.site")]
    2 references
    Task<string> GetStatus([Authorize("Bearer")] string token);
}
```

BINARY SECURITY

Fallout

- Assigned CVE-2024-51501 (not fully assessed yet)

Fallout

- Assigned CVE-2024-51501 (not fully assessed yet)
- Fixed in Refit v7.2.22 and v8.0.0

Conclusions?

- Library that uses library that uses … that uses vulnerable method

BINARY SECURITY

Conclusions?

- Library that uses library that uses ... that uses vulnerable method
- Be careful with user-controllable input!

BINARY SECURITY

Conclusions?

- Library that uses library that uses ... that uses vulnerable method
- Be careful with user-controllable input!
- CVE in RestSharp and fixed in v112

Conclusions?

- Library that uses library that uses ... that uses vulnerable method
- Be careful with user-controllable input!
- CVE in RestSharp and fixed in v112
- CVE in Refit and fixed in v7.2.22 and v8.0.0

BINARY SECURITY

Conclusions?

- Library that uses library that uses … that uses vulnerable method
- Be careful with user-controllable input!
- CVE in RestSharp and fixed in v112
- CVE in Refit and fixed in v7.2.22 and v8.0.0

Questions?

BINARY SECURITY