



POSITIVE TECHNOLOGIES

Исключительно простая теория AppSec .NET

Владимир Кочетков

Application Inspector/Compiling Applications Analysis/Team Lead

Positive Technologies

Июньская встреча SPB.NET Community, 2015

:~\$ whoami && ~~whonotme~~

- .NET-разработчик, руководитель группы анализа компилируемых приложений в Positive Technologies
- AppSec-исследователь
- RSDN тимер
- ~~Оторванный от реальности теоретик~~
- ~~Упоротый параноик~~

Всего один вопрос...

...что такое уязвимость?



Какой из фрагментов кода уязвим?

1)

```
var cmd = new SqlCommand("SELECT Value FROM Discounts WHERE CouponCode = '" +  
Request["CouponCode"] + "'");  
var connection = new SqlConnection(connectionString);  
connection.Open();  
cmd.Connection = connection;  
var couponValue = cmd.ExecuteScalar();  
...
```

2)

```
var cmd = new SqlCommand("SELECT Value FROM Discounts WHERE CouponCode =  
@CouponCode");  
cmd.Parameters.AddWithValue("@CouponCode ", Request["CouponCode"]);  
var connection = new SqlConnection(connectionString);  
connection.Open();  
cmd.Connection = connection;  
var couponValue = cmd.ExecuteScalar();  
...
```

Какой из фрагментов кода уязвим?

1)

```
[Authorize(Roles = "All")]  
public ActionResult SomeAction()  
{  
    ...  
    return View();  
}
```

2)

```
[Authorize(Roles = "Baz, Qux")]  
public ActionResult SomeAction()  
{  
    ...  
    return View();  
}
```

Уязвимость к RCE?

```
var code = wrapCode("Foo", "Bar", "Qux", Request["code"]);

var provider = new CSharpCodeProvider();

var compilerParams = new CompilerParameters
{
    GenerateInMemory = true,
    GenerateExecutable = false
};

var results = provider.CompileAssemblyFromSource(
    compilerParams, code);

if (!results.Errors.Any())
{
    var o = results.CompiledAssembly.CreateInstance("Foo.Bar");
    var mi = o.GetType().GetMethod("Qux");
    mi.Invoke(o, null);
}
```


Критерии **уязвимости**
определяются
множествами правил
предметных областей
приложения

Примеры предметных областей

Нефункциональные:

- управление потоками данных;
- управление потоками выполнения;
- контроль доступа.

Функциональные:

- интернет-торговля;
- онлайн-банкинг;
- бухучет;
- ...

Театр начинается с
вешалки, а **уязвимость** –
с **недостатка**

Недостаток

- неэффективный контроль выполнения правил предметных областей приложения

Примеры недостатков:

- неэффективная предварительная обработка данных;
- неэффективный контроль аутентичности источника запросов;
- неэффективный контроль доступа;
- неэффективный контроль жизненного цикла транзакции;
- неэффективный контроль распределения ролей.

**Приложение - это поток
управления,
обрабатывающий
множество потоков
данных**

Потоки управления

```
var name = Request.Params["name"];
var key1 = Request.Params["key1"];
var parm = Request.Params["parm"];

var data = string.IsNullOrEmpty(parm) ? new char[0]: Convert.FromBase64String(parm);

string str1;
if (name + "in" == "admin")
{
    if (key1 == "validkey")
    {
        str1 = Encoding.UTF8.GetString(data);
    }
    else
    {
        str1 = "Wrong key!";
    }

    Response.Write(str1);
}
```

Потоки управления

```
var name = Request.Params["name"];
```

```
var key1 = Request.Params["key1"];
```

```
var parm = Request.Params["parm"];
```

```
var data = string.IsNullOrEmpty(parm) ? new char[0]: Convert.FromBase64String(parm);
```

```
string str1;
```

```
if (name + "in" == "admin")
```

```
{
```

```
    if (key1 == "validkey")
```

```
    {
```

```
        str1 = Encoding.UTF8.GetString(data);
```

```
    }
```

```
    else
```

```
    {
```

```
        str1 = "Wrong key!";
```

```
    }
```

```
    Response.Write(str1);
```

```
}
```

Потоки управления

```
var name = Request.Params["name"];  
var key1 = Request.Params["key1"];  
var parm = Request.Params["parm"];
```

```
var data = string.IsNullOrEmpty(parm) ? new char[0]: Convert.FromBase64String(parm);
```

```
string str1;  
if (name + "in" == "admin")  
{  
    if (key1 == "validkey")  
    {  
        str1 = Encoding.UTF8.GetString(data);  
    }  
    else  
    {  
        str1 = "Wrong key!";  
    }  
  
    Response.Write(str1);  
}
```


Потоки управления

```
var name = Request.Params["name"];  
var key1 = Request.Params["key1"];  
var parm = Request.Params["parm"];
```

```
var data = string.IsNullOrEmpty(parm) ? new char[0]: Convert.FromBase64String(parm);
```

```
string str1;  
if (name + "in" == "admin")  
{  
    if (key1 == "validkey")  
    {  
        str1 = Encoding.UTF8.GetString(data);  
    }  
    else  
    {  
        str1 = "Wrong key!";  
    }  
  
    Response.Write(str1);  
}
```

Потоки управления

```
var name = Request.Params["name"];
var key1 = Request.Params["key1"];
var parm = Request.Params["parm"];

var data = string.IsNullOrEmpty(parm) ? new char[0]: Convert.FromBase64String(parm);

string str1;
if (name + "in" == "admin")
{
    if (key1 == "validkey")
    {
        str1 = Encoding.UTF8.GetString(data);
    }
    else
    {
        str1 = "Wrong key!";
    }

    Response.Write(str1);
}
```

Потоки управления

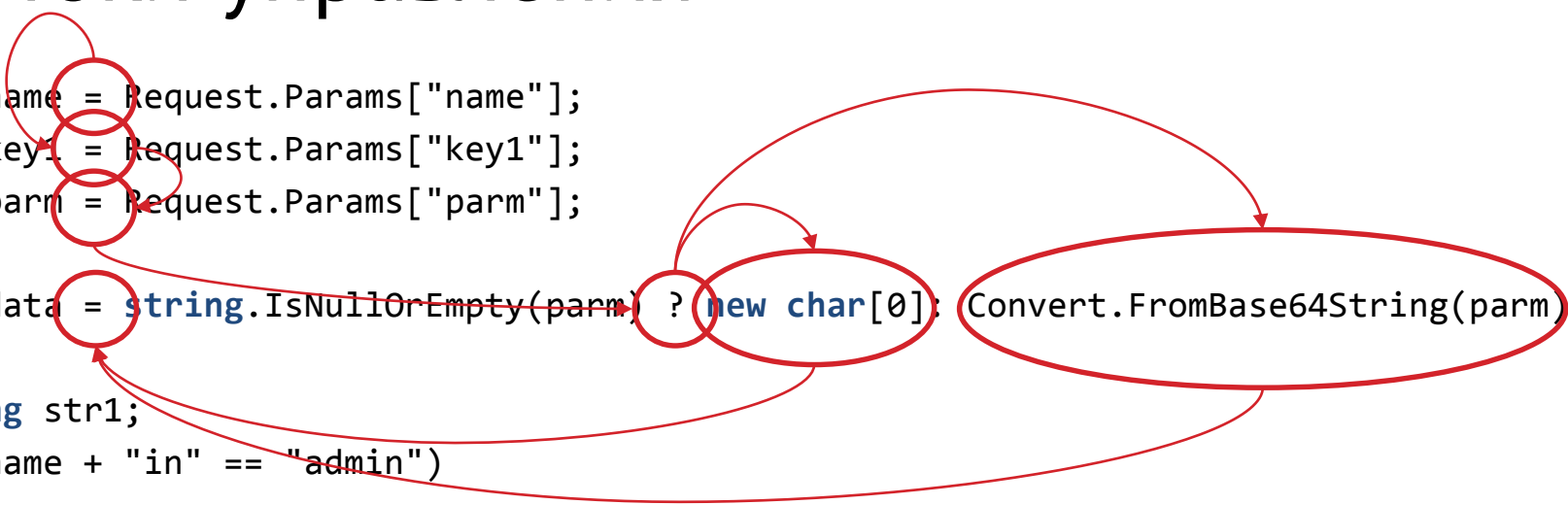
```
var name = Request.Params["name"];
var key1 = Request.Params["key1"];
var parm = Request.Params["parm"];

var data = string.IsNullOrEmpty(parm) ? new char[0] : Convert.FromBase64String(parm);

string str1;
if (name + "in" == "admin")
{
    if (key1 == "validkey")
    {
        str1 = Encoding.UTF8.GetString(data);
    }
    else
    {
        str1 = "Wrong key!";
    }

    Response.Write(str1);
}
```

Потоки управления



```
graph TD
    L1[Request.Params["name"]] --> L2[Request.Params["key1"]]
    L2 --> L3[Request.Params["parm"]]
    L3 --> L4["string.IsNullOrEmpty(parm) ? new char[0] : Convert.FromBase64String(parm)"]
    L4 --> L5["if (name + \"in\" == \"admin\")"]
    L5 --> L6["if (key1 == \"validkey\")"]
    L6 --> L7["str1 = Encoding.UTF8.GetString(data);"]
    L7 --> L8["else"]
    L8 --> L9["str1 = \"Wrong key!\";"]
    L9 --> L10["Response.Write(str1);"]
    L10 --> L11[End]
```

```
var name = Request.Params["name"];
var key1 = Request.Params["key1"];
var parm = Request.Params["parm"];

var data = string.IsNullOrEmpty(parm) ? new char[0] : Convert.FromBase64String(parm);

string str1;
if (name + "in" == "admin")
{
    if (key1 == "validkey")
    {
        str1 = Encoding.UTF8.GetString(data);
    }
    else
    {
        str1 = "Wrong key!";
    }

    Response.Write(str1);
}
```

Потоки управления

```
var name = Request.Params["name"];
var key1 = Request.Params["key1"];
var parm = Request.Params["parm"];

var data = string.IsNullOrEmpty(parm) ? new char[0] : Convert.FromBase64String(parm);

string str1;
if (name + "in" == "admin")
{
    if (key1 == "validkey")
    {
        str1 = Encoding.UTF8.GetString(data);
    }
    else
    {
        str1 = "Wrong key!";
    }

    Response.Write(str1);
}
```

Потоки управления

```
var name = Request.Params["name"];
var key1 = Request.Params["key1"];
var parm = Request.Params["parm"];

var data = string.IsNullOrEmpty(parm) ? new char[0] : Convert.FromBase64String(parm);

string str1;
if (name + "in" == "admin")
{
    if (key1 == "validkey")
    {
        str1 = Encoding.UTF8.GetString(data);
    }
    else
    {
        str1 = "Wrong key!";
    }

    Response.Write(str1);
}
```

The diagram illustrates the control flow of the provided C# code. Red circles highlight the following elements:

- The variable `name` in the first three `var` declarations.
- The variable `key1` in the second `var` declaration.
- The variable `parm` in the third `var` declaration.
- The conditional expression `string.IsNullOrEmpty(parm)` in the `var data` declaration.
- The variable `data` in the `var data` declaration.
- The variable `key1` in the inner `if` statement.
- The variable `str1` in the `string str1;` declaration.
- The condition `name + "in" == "admin"` in the outer `if` statement.

Red arrows indicate the flow of execution:

- From the `var` declarations to the `var data` declaration.
- From the `var data` declaration to the `string str1;` declaration.
- From the `string str1;` declaration to the `if (name + "in" == "admin")` condition.
- From the `if (name + "in" == "admin")` condition to the `if (key1 == "validkey")` condition.
- From the `if (key1 == "validkey")` condition to the `str1 = Encoding.UTF8.GetString(data);` statement.
- From the `str1 = Encoding.UTF8.GetString(data);` statement to the `else` block.
- From the `else` block to the `str1 = "Wrong key!";` statement.
- From the `str1 = "Wrong key!";` statement to the `Response.Write(str1);` statement.
- From the `Response.Write(str1);` statement to the closing brace of the `if` statement.

Потоки управления

```
var name = Request.Params["name"];
var key1 = Request.Params["key1"];
var parm = Request.Params["parm"];

var data = string.IsNullOrEmpty(parm) ? new char[0] : Convert.FromBase64String(parm);

string str1;
if (name + "in" == "admin")
{
    if (key1 == "validkey")
    {
        str1 = Encoding.UTF8.GetString(data);
    }
    else
    {
        str1 = "Wrong key!";
    }

    Response.Write(str1);
}
```

Потоки управления

```
var name = Request.Params["name"];
var key1 = Request.Params["key1"];
var parm = Request.Params["parm"];

var data = string.IsNullOrEmpty(parm) ? new char[0] : Convert.FromBase64String(parm);

string str1;
if (name + "in" == "admin")
{
    if (key1 == "validkey")
    {
        str1 = Encoding.UTF8.GetString(data);
    }
    else
    {
        str1 = "Wrong key!";
    }

    Response.Write(str1);
}
```


Потоки управления

```
graph TD
    L1[Request.Params["name"]] --> L2[Request.Params["key1"]]
    L2 --> L3[Request.Params["parm"]]
    L3 --> L4[IsNullOrEmpty(parm)]
    L4 --> L5[new char[0]]
    L5 --> L6[Convert.FromBase64String(parm)]
    L6 --> L7[Encoding.UTF8.GetString(data)]
    L7 --> L8[Response.Write(str1)]
    L8 --> L9[End]
    L4 --> L10[if (name + "in" == "admin")]
    L10 --> L11[if (key1 == "validkey")]
    L11 --> L7
    L10 --> L12[else]
    L12 --> L13[Response.Write(str1)]
    L13 --> L9
```

```
var name = Request.Params["name"];
var key1 = Request.Params["key1"];
var parm = Request.Params["parm"];

var data = string.IsNullOrEmpty(parm) ? new char[0] : Convert.FromBase64String(parm);

string str1;
if (name + "in" == "admin")
{
    if (key1 == "validkey")
    {
        str1 = Encoding.UTF8.GetString(data);
    }
    else
    {
        str1 = "Wrong key!";
    }
    Response.Write(str1);
}
```

The image shows a C# code snippet with red annotations highlighting control flow. Red circles are drawn around the variable declarations, the ternary operator, the conditional statements, and the data variable. Red arrows trace the execution path: from the variable declarations to the ternary operator, then to the conditional statements, and finally to the data variable and the response write statement. The annotations illustrate the flow of control through the code, showing how the program branches based on the values of the variables.

Потоки управления

```
var name = Request.Params["name"];
var key1 = Request.Params["key1"];
var parm = Request.Params["parm"];

var data = string.IsNullOrEmpty(parm) ? new char[0] : Convert.FromBase64String(parm);

string str1;
if (name + "in" == "admin")
{
    if (key1 == "validkey")
    {
        str1 = Encoding.UTF8.GetString(data);
    }
    else
    {
        str1 = "Wrong key!";
    }
}

Response.Write(str1);
}
```

```
graph TD
    A["var name = Request.Params[\"name\"];  
var key1 = Request.Params[\"key1\"];  
var parm = Request.Params[\"parm\"];"] --> B["var data = string.IsNullOrEmpty(parm) ? new char[0] : Convert.FromBase64String(parm);"]
    B --> C["if (name + \"in\" == \"admin\")"]
    C --> D["if (key1 == \"validkey\")"]
    D --> E["str1 = Encoding.UTF8.GetString(data);"]
    E --> F["else"]
    F --> G["str1 = \"Wrong key!\";"]
    G --> H["Response.Write(str1);"]
    H --> I["}"]
```

Потоки управления

```
var name = Request.Params["name"];
var key1 = Request.Params["key1"];
var parm = Request.Params["parm"];

var data = string.IsNullOrEmpty(parm) ? new char[0] : Convert.FromBase64String(parm);

string str1;
if (name + "in" == "admin")
{
    if (key1 == "validkey")
    {
        str1 = Encoding.UTF8.GetString(data);
    }
    else
    {
        str1 = "Wrong key!";
    }
}

Response.Write(str1);
}
```

```
graph TD
    name[Request.Params["name"]] --> key1[Request.Params["key1"]]
    key1 --> parm[Request.Params["parm"]]
    parm --> data["string.IsNullOrEmpty(parm) ? new char[0] : Convert.FromBase64String(parm)"]
    data --> if_name["if (name + 'in' == 'admin')"]
    if_name --> if_key["if (key1 == 'validkey')"]
    if_key --> str1_decode["str1 = Encoding.UTF8.GetString(data)"]
    if_key --> str1_wrong["str1 = 'Wrong key!'"]
    str1_decode --> write["Response.Write(str1);"]
    str1_wrong --> write
    if_name --> write
    if_key --> write
```

Потоки управления

```
var name = Request.Params["name"];
var key1 = Request.Params["key1"];
var parm = Request.Params["parm"];

var data = string.IsNullOrEmpty(parm) ? new char[0] : Convert.FromBase64String(parm);

string str1;
if (name + "in" == "admin")
{
    if (key1 == "validkey")
    {
        str1 = Encoding.UTF8.GetString(data);
    }
    else
    {
        str1 = "Wrong key!";
    }
    Response.Write(str1);
}
```

Потоки данных

```
var name = Request.Params["name"];
var key1 = Request.Params["key1"];
var parm = Request.Params["parm"];

var data = string.IsNullOrEmpty(parm) ? new char[0]: Convert.FromBase64String(parm);

string str1;
if (name + "in" == "admin")
{
    if (key1 == "validkey")
    {
        str1 = Encoding.UTF8.GetString(data);
    }
    else
    {
        str1 = "Wrong key!";
    }

    Response.Write(str1);
}
```

Потоки данных

```
var name = Request.Params["name"];
var key1 = Request.Params["key1"];
var parm = Request.Params["parm"];

var data = string.IsNullOrEmpty(parm) ? new char[0]: Convert.FromBase64String(parm);

string str1;
if (name + "in" == "admin")
{
    if (key1 == "validkey")
    {
        str1 = Encoding.UTF8.GetString(data);
    }
    else
    {
        str1 = "Wrong key!";
    }

    Response.Write(str1);
}
```

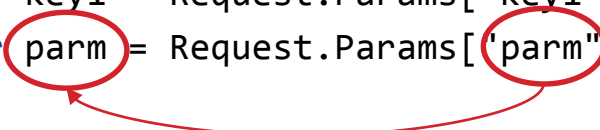
Потоки данных

```
var name = Request.Params["name"];
var key1 = Request.Params["key1"];
var parm = Request.Params["parm"];

var data = string.IsNullOrEmpty(parm) ? new char[0]: Convert.FromBase64String(parm);

string str1;
if (name + "in" == "admin")
{
    if (key1 == "validkey")
    {
        str1 = Encoding.UTF8.GetString(data);
    }
    else
    {
        str1 = "Wrong key!";
    }

    Response.Write(str1);
}
```

A red circle highlights the variable 'parm' in the third line of code. Another red circle highlights the argument 'parm' in the 'Convert.FromBase64String' method call in the fifth line. A red arrow points from the 'parm' in the second circle to the 'parm' in the first circle, indicating that the value of 'parm' is used in the conversion.

Потоки данных

```
var name = Request.Params["name"];
var key1 = Request.Params["key1"];
var parm = Request.Params["parm"];

var data = string.IsNullOrEmpty(parm) ? new char[0]: Convert.FromBase64String(parm);

string str1;
if (name + "in" == "admin")
{
    if (key1 == "validkey")
    {
        str1 = Encoding.UTF8.GetString(data);
    }
    else
    {
        str1 = "Wrong key!";
    }

    Response.Write(str1);
}
```

The diagram illustrates the flow of data from the `Request.Params` object to the variables `name`, `key1`, and `parm`. Red circles highlight the variable names `parm` and the string `"parm"` in the third line, and the `new char[0]` and `parm` in the fourth line. Red arrows show the flow of data from `Request.Params["parm"]` to `parm` in the fourth line, and from `parm` in the fourth line to the `Convert.FromBase64String(parm)` method call.

Потоки данных

```
var name = Request.Params["name"];
var key1 = Request.Params["key1"];
var parm = Request.Params["parm"];
var data = string.IsNullOrEmpty(parm) ? new char[0] : Convert.FromBase64String(parm);
string str1;
if (name + "in" == "admin")
{
    if (key1 == "validkey")
    {
        str1 = Encoding.UTF8.GetString(data);
    }
    else
    {
        str1 = "Wrong key!";
    }

    Response.Write(str1);
}
```

Потоки данных

```
var name = Request.Params["name"];
var key1 = Request.Params["key1"];
var parm = Request.Params["parm"];
var data = string.IsNullOrEmpty(parm) ? new char[0]: Convert.FromBase64String(parm);
string str1;
if (name + "in" == "admin")
{
    if (key1 == "validkey")
    {
        str1 = Encoding.UTF8.GetString(data);
    }
    else
    {
        str1 = "Wrong key!";
    }

    Response.Write(str1);
}
```

Потоки данных

```
var name = Request.Params["name"];
var key1 = Request.Params["key1"];
var parm = Request.Params["parm"];
var data = string.IsNullOrEmpty(parm) ? new char[0] : Convert.FromBase64String(parm);

string str1;
if (name + "in" == "admin")
{
    if (key1 == "validkey")
    {
        str1 = Encoding.UTF8.GetString(data);
    }
    else
    {
        str1 = "Wrong key!";
    }

    Response.Write(str1);
}
```

Потоки данных

```
var name = Request.Params["name"];
var key1 = Request.Params["key1"];
var parm = Request.Params["parm"];

var data = string.IsNullOrEmpty(parm) ? new char[0] : Convert.FromBase64String(parm);

string str1;
if (name + "in" == "admin")
{
    if (key1 == "validkey")
    {
        str1 = Encoding.UTF8.GetString(data);
    }
    else
    {
        str1 = "Wrong key!";
    }

    Response.Write(str1);
}
```

The diagram illustrates the flow of data in the provided C# code. Red circles highlight specific variables and expressions, and red arrows show the flow of data between them. The flow is as follows:

- `Request.Params["name"]` flows to `name`.
- `Request.Params["key1"]` flows to `key1`.
- `Request.Params["parm"]` flows to `parm`.
- `parm` flows to `data` in the conditional expression.
- `Convert.FromBase64String(parm)` flows to `data`.
- `data` flows to `Encoding.UTF8.GetString(data)` inside the nested if block.
- `Encoding.UTF8.GetString(data)` flows to `str1`.
- `str1` flows to `Response.Write(str1)`.

Потоки данных

```
var name = Request.Params["name"];
var key1 = Request.Params["key1"];
var parm = Request.Params["parm"];

var data = string.IsNullOrEmpty(parm) ? new char[0] : Convert.FromBase64String(parm);

string str1;
if (name + "in" == "admin")
{
    if (key1 == "validkey")
    {
        str1 = Encoding.UTF8.GetString(data);
    }
    else
    {
        str1 = "Wrong key!";
    }

    Response.Write(str1);
}
```

Потоки данных

```
var name = Request.Params["name"];
var key1 = Request.Params["key1"];
var parm = Request.Params["parm"];

var data = string.IsNullOrEmpty(parm) ? new char[0] : Convert.FromBase64String(parm);

string str1;
if (name + "in" == "admin")
{
    if (key1 == "validkey")
    {
        str1 = Encoding.UTF8.GetString(data);
    }
    else
    {
        str1 = "Wrong key!";
    }
    Response.Write(str1);
}
```

Потоки данных

```
var name = Request.Params["name"];
var key1 = Request.Params["key1"];
var parm = Request.Params["parm"];

var data = string.IsNullOrEmpty(parm) ? new char[0] : Convert.FromBase64String(parm);

string str1;
if (name + "in" == "admin")
{
    if (key1 == "validkey")
    {
        str1 = Encoding.UTF8.GetString(data);
    }
    else
    {
        str1 = "Wrong key!";
    }
    Response.Write(str1);
}
```

**Потоки управления
являются производными
от потоков данных**

Множества значений
всех **ПОТОКОВ ДАННЫХ** в
конкретной точке **ПОТОКА**
выполнения определяют
состояние приложения

Угроза

- возможность обхода правил **предметных областей** приложения, приводящего к нарушению свойств защищенности хотя бы одного **потока данных**:

- конфиденциальности;
- целостности;
- доступности;
- аутентичности;
- авторизованности.



Уязвимость – состояние приложения, в котором возможна реализация угрозы

Иными словами...

То, **что может** сделать с потоками данных атакующий, нарушив правила предметных областей, называется **угрозой (threat)**

То, **где и благодаря чему** он может это сделать, называется **уязвимостью (vulnerability)**, обусловленной **недостатком (weakness)**

То, **как** он может это сделать, называется **атакой (attack)**

То, **с какой вероятностью** у него это удастся и **какие последствия** может повлечь, называется **риском (risk)**

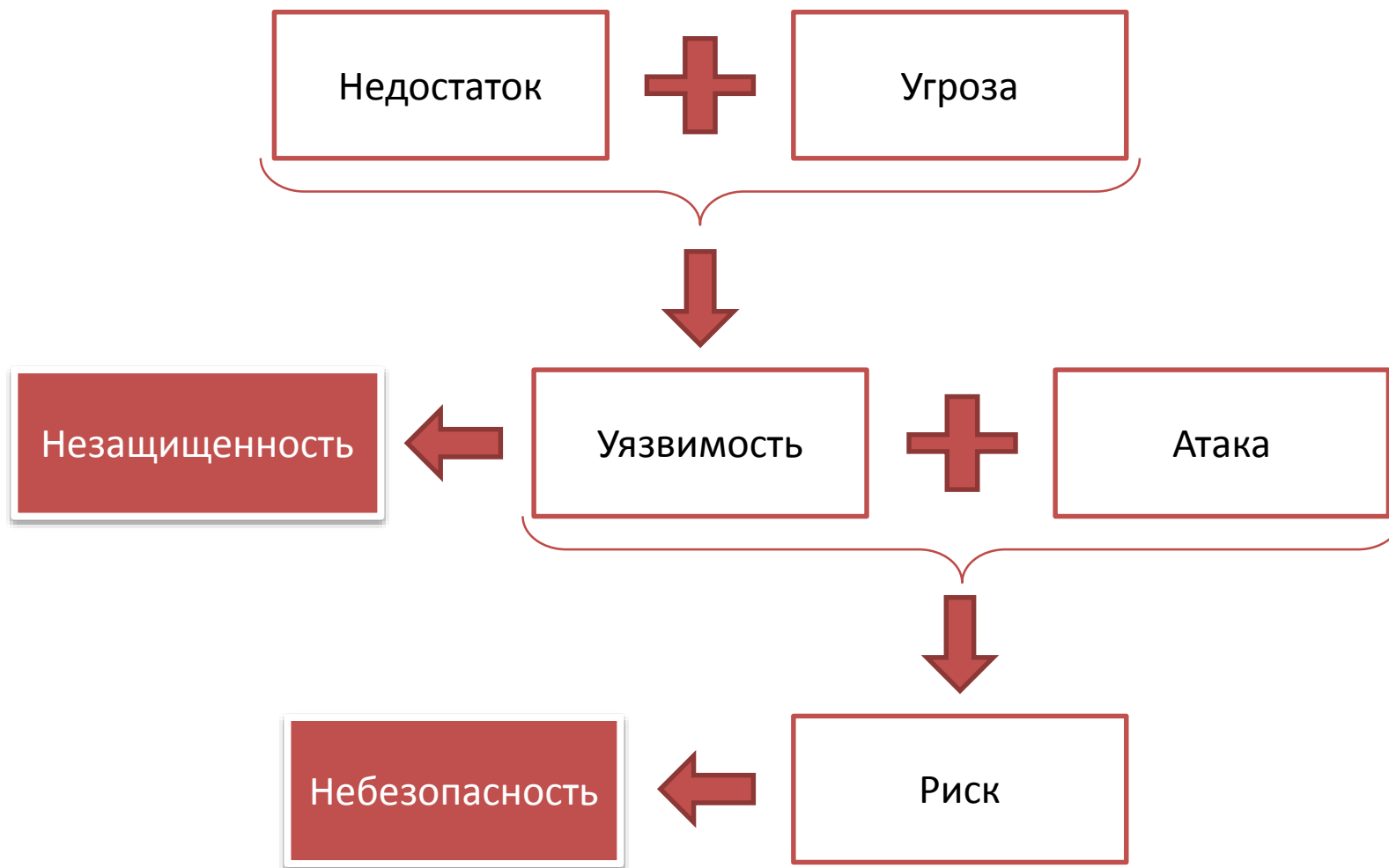
Иными словами...

То, что **не позволяет** атакующему провести атаку, обеспечивает **защищенность (security)**

То, что **минимизирует риск**, обеспечивает **безопасность (safety)**

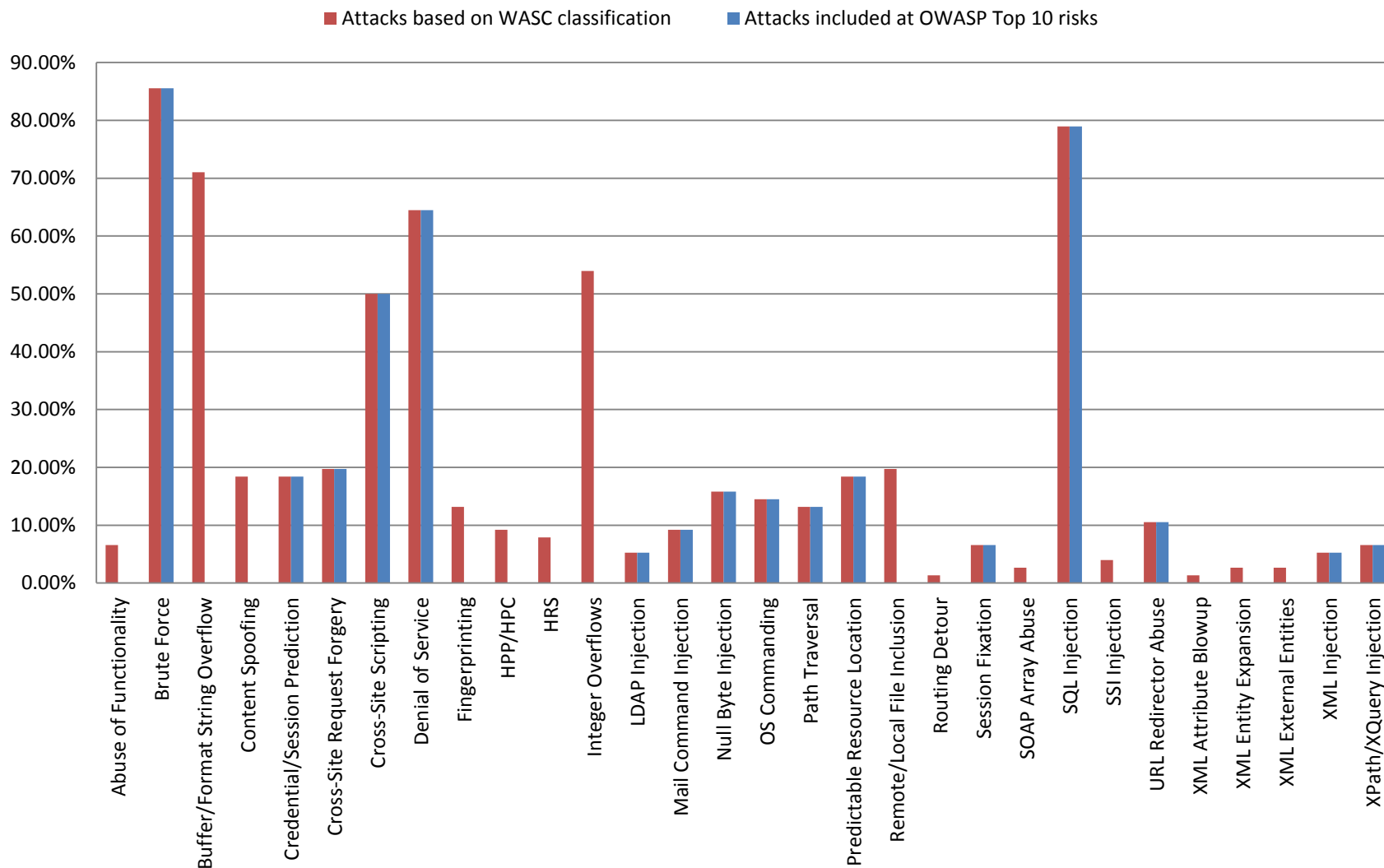
Разработчикам следует обеспечивать **защищенность**, не допуская появления **недостатков** в коде

Причинно-следственные связи



Бороться необходимо с причинами, а не со следствиями!

Осведомленность разработчиков*



* на основе результатов опроса <http://www.rsdn.ru/?poll/3488>

Классификация

Классификация уязвимостей возможна по:

- **предметной области;** *// управление потоками данных*
- **недостатку;** *// неэффективная предварительная обработка данных*
- **потоку данных;** *// первообразная потока управления*
- **угрозе;** *// нарушение целостности*

== уязвимость к атакам инъекций (в зависимости от интерпретатора потока данных: XSS, SQLi, XMLi, XPATHi, Path Traversal, LINQj, XXE и т.п.)

Предварительная обработка данных



Подходы к предварительной обработке

- **Типизация** — создание объектного представления входных данных из строкового типа (парсинг и десериализация).
- **Санитизация** — приведение данных в соответствие с грамматикой, допускаемой политикой защищенности.
- **Валидация** — проверка данных на соответствие установленным критериям:
 - грамматическая;
 - семантическая.

Смотри, не перепутай...

Типизация и валидация **на входе**, санитизация — **на выходе!**



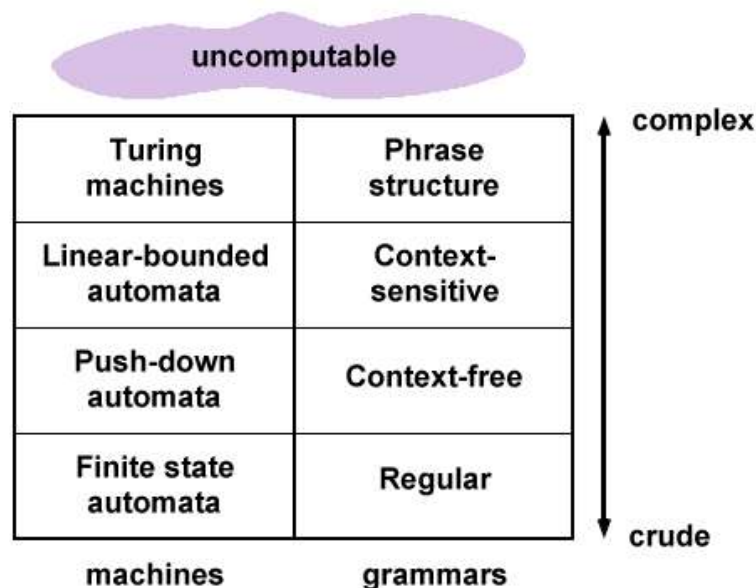
Обобщенный подход

Протоколы потоков данных— **формальные языки**.

Некоторые языки **распознаются сложнее**, чем остальные.

Для некоторых языков **распознавание неразрешимо**.

Чем сложнее язык, тем тяжелее
сформировать критерии к входным
данным, описывающим множества
конфигураций системы.



Обобщенный подход

Тестирование эквивалентности конечных автоматов или детерминированных стековых автоматов* **разрешимо**.

Для недетерминированных стековых автоматов и более мощных моделей вычислений такое тестирование является **неразрешимым**.

В первом случае возможно полное покрытие тестами элементов парсера языка обрабатываемых данных или их статический анализ.

Во втором случае — **НЕТ!**

Обобщенный подход

Упрощение или декомпозиция языка входных данных до множества регулярных или детерминированных контекстно-свободных грамматик.

Внедрение в код проверок (типизации/валидации) **входных данных** в соответствии с грамматикой их языка **как можно ближе к началу потока управления.**

Внедрение в код санитайзеров выходных данных, построенных в соответствии с грамматикой принимающей стороны, **как можно ближе к потенциально уязвимому состоянию.**

Островные языки

Островными – называются языки, грамматики которых описывают правила распознавания отдельных выражений («островов»), окруженных выражениями, принадлежащими другому языку («морем»).

```
string.Format("SELECT * FROM Table WHERE Id= {0}", Request["Id"]);
```

more (C#) остров (SQL) more (C#)

Работа с любыми **выходными данными** сводится к формированию текста на островном языке, а также приводит к образованию новых островных языков, т.к.:

острова с параметрами являются верхнеуровневыми островными языками

Формальные признаки инъекции

- Потенциально уязвимая операция $PVO(text)$: операция прямой или косвенной интерпретации текста $text$ на формальном островном языке
- $text = transform(argument)$, где $argument$ – элемент множества аргументов точки входа EP , а $transform$ – функция промежуточных преобразований
- Не существует или недостижимо ни одно множество таких значений элементов EP , при которых происходит изменение структуры синтаксического дерева значения $text$, достигающего PVO , не предусмотренное правилами прочих предметных областей

Анатомия атак инъекций

```
1' AND 1=(SELECT COUNT(*) FROM tablenames); --
```

```
'><script>alert/XSS/</script><!--
```

```
../../../../../../../../etc/passwd%00
```

```
admin*)((|userPassword=*)
```

Что общего между ними?

Анатомия инъекций

```
1' AND 1=(SELECT COUNT(*) FROM tablenames); --
```

```
'><script>alert/XSS/</script><!--
```

```
../../../../../../../../etc/passwd%00
```

```
admin*)((|userPassword=*)
```

Атакующие используют синтаксические элементы основной грамматики, чтобы выбраться за пределы острова и нарушить целостность потока выполнения принимающей стороны

Островные языки

Какой грамматике соответствует запрос:

```
SELECT * FROM Table WHERE Id={:int}
```



Контекстно-свободная SQL?

```
select stmt      ::= SELECT select list from clause
                  | SELECT select list from clause where clause
select list      ::= id list
                  | *
id list          ::= id
                  | id , id list
from clause      ::= FROM tbl list
tbl list         ::= id list
where clause     ::= WHERE bool cond
bcond            ::= bcond OR bterm
                  | bterm
bterm            ::= bterm AND bfactor
                  | bfactor
bfactor          ::= NOT cond
                  | cond
cond             ::= value comp value
value            ::= id
                  | str lit
                  | num
str lit          ::= ' lit '
comp             ::= = | < | > | <= | >= | !=
```

... (+ еще ~1000 страниц отборной EBNF)

Регулярная, островная!

```
SELECT \* FROM Table WHERE Id=[0-9]+
```

Море

Остров

Явное выделение всех островных грамматик облегчает реализацию контроля выходных данных

Чтобы реализовать **достаточный и эффективный** контроль выходных данных, в общем случае, необходимо взять из основной островной грамматики правила, описывающие разбор токена, в который попадают входные данные и **обеспечить невозможность их выхода** за пределы этого токена.

Пример: LINQ Injection

```
public AjaxStoreResult GetCustomers(int limit, int start, string dir, string sort)
{
    var query = (from c in this.DBContext.Customers
                  select new
                  {
                      c.CustomerID,
                      c.CompanyName,
                      c.ContactName,
                      c.Phone,
                      c.Fax,
                      c.Region
                  }).OrderBy(string.Concat(sort, " ", dir));

    int total = query.ToList().Count;

    query = query.Skip(start).Take(limit);
    return new AjaxStoreResult(query, total);
}
```

Пример: LINQ Injection

```
public AjaxStoreResult GetCustomers(int limit, int start, string dir, string sort)
{
    var query = (from c in this.DBContext.Customers
                  select new
                  {
                      c.CustomerID,
                      c.CompanyName,
                      c.ContactName,
                      c.Phone,
                      c.Fax,
                      c.Region
                  }).OrderBy(string.Concat(sort, " ", dir));

    int total = query.ToList().Count;

    query = query.Skip(start).Take(limit);
    return new AjaxStoreResult(query, total);
}
```

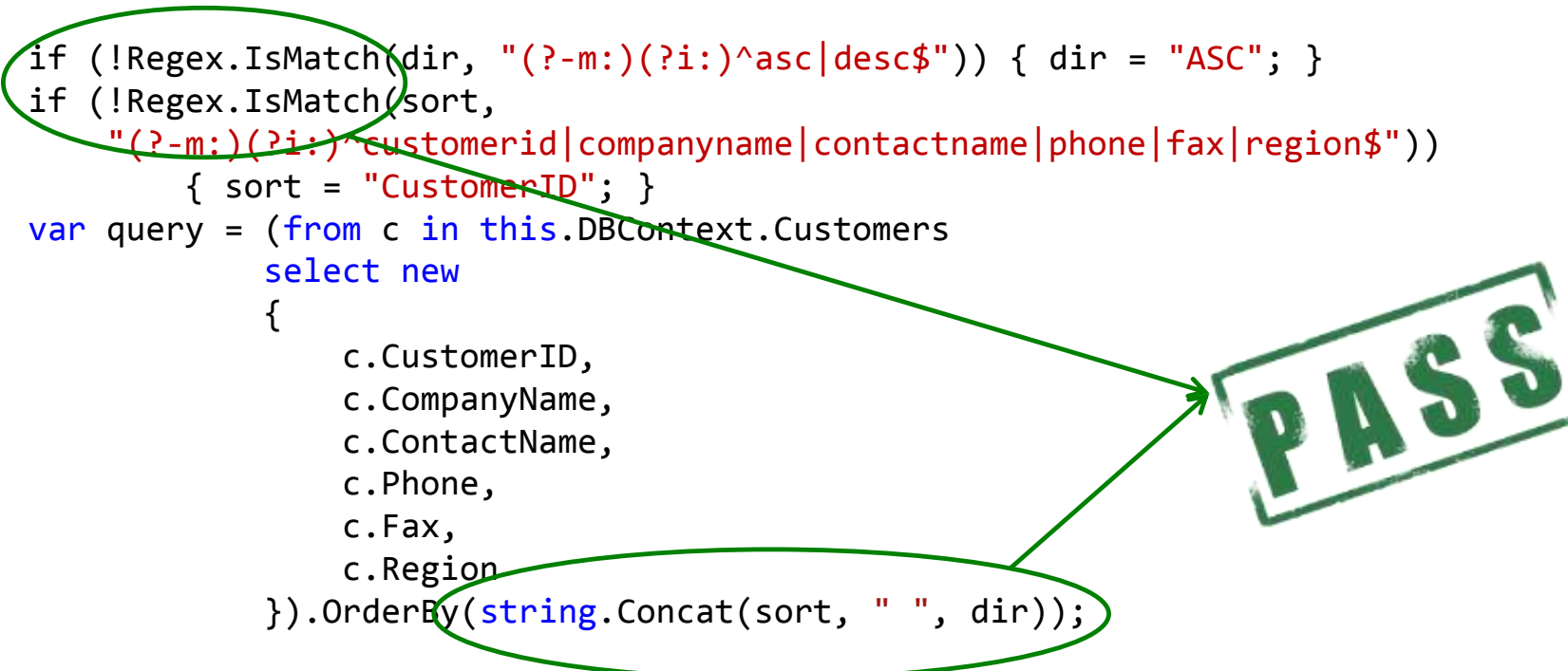


FAIL

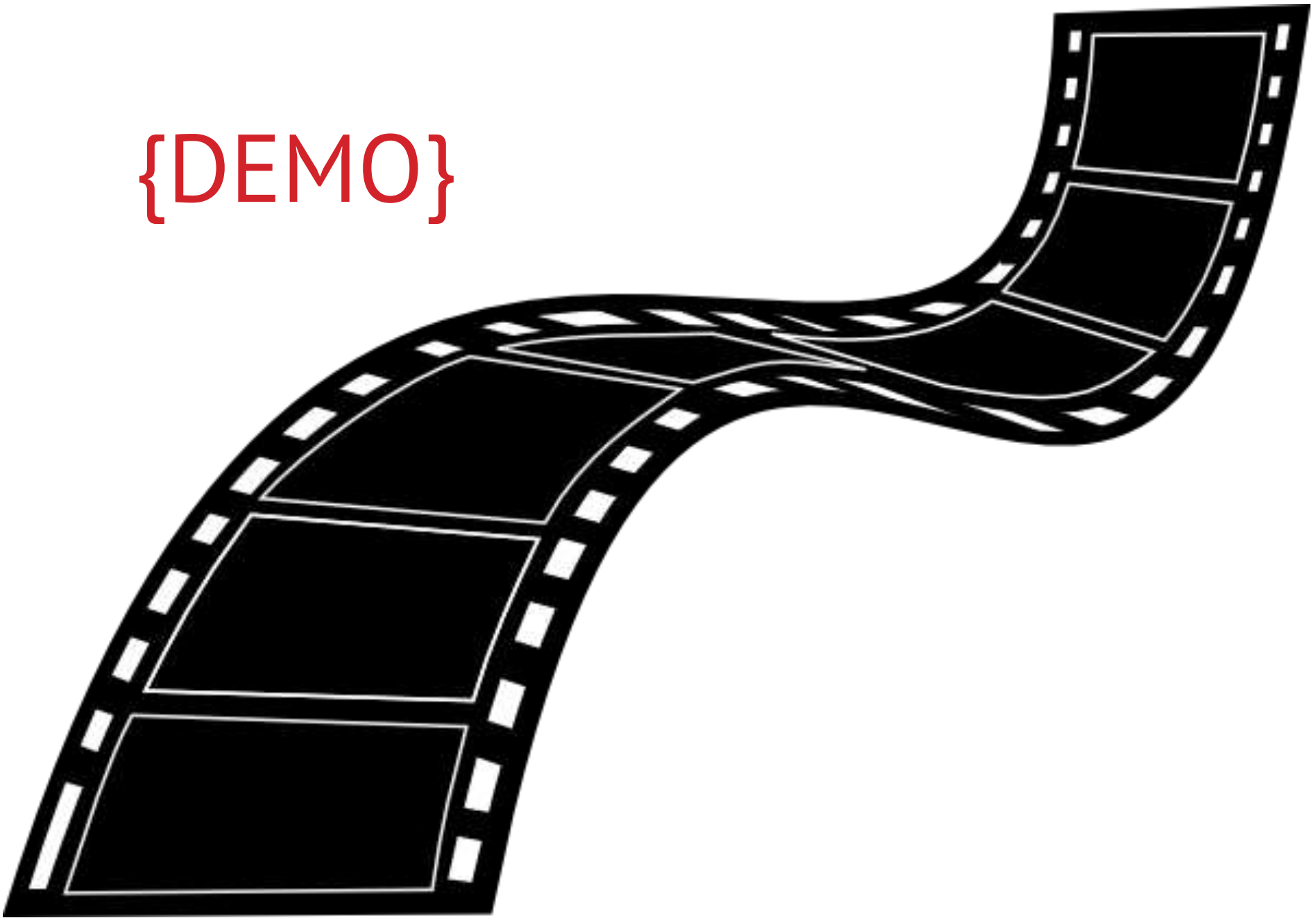
Пример: LINQ Injection

```
public AjaxStoreResult GetCustomers(int limit, int start, string dir, string sort)
{
    if (!Regex.IsMatch(dir, "(?-m:)(?i:)^asc|desc$")) { dir = "ASC"; }
    if (!Regex.IsMatch(sort,
        "(?-m:)(?i:)^customerid|companyname|contactname|phone|fax|region$"))
        { sort = "CustomerID"; }
    var query = (from c in this.DBContext.Customers
        select new
        {
            c.CustomerID,
            c.CompanyName,
            c.ContactName,
            c.Phone,
            c.Fax,
            c.Region
        }).OrderBy(string.Concat(sort, " ", dir));

    var total = query.ToList().Count;
    query = query.Skip(start).Take(limit);
    return new AjaxStoreResult(query, total);
}
```



{DEMO}



Вопросы?

Владимир Кочетков

Application Inspector/Compiling Applications Analysis/Team Lead
Positive Technologies

vkochetkov@ptsecurity.com

[@kochetkov_v](#)



POSITIVE TECHNOLOGIES