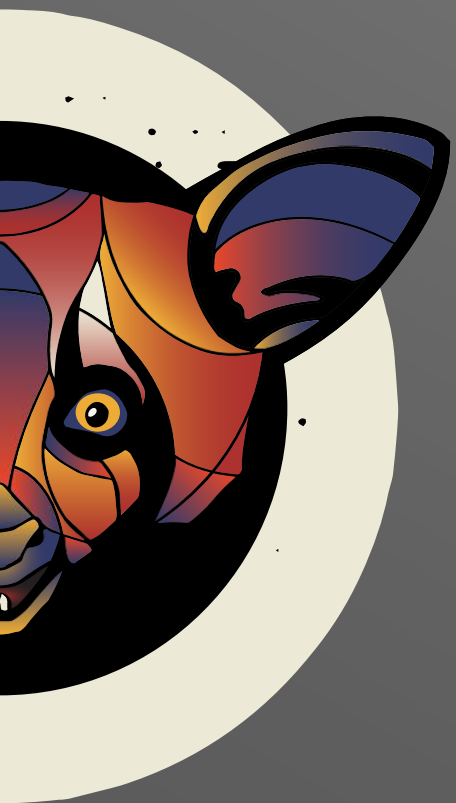




POSITIVE DEVELOPMENT USER GROUP

Побеждая инъекции

Positive Technologies
Владимир Кочетков

:~\$ 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

Открытое сообщество разработчиков и IT-специалистов, которые стремятся создавать безопасные приложения.

https://t.me/ru_appsec



«Параметризация SQL-запросов и ORM решают!»

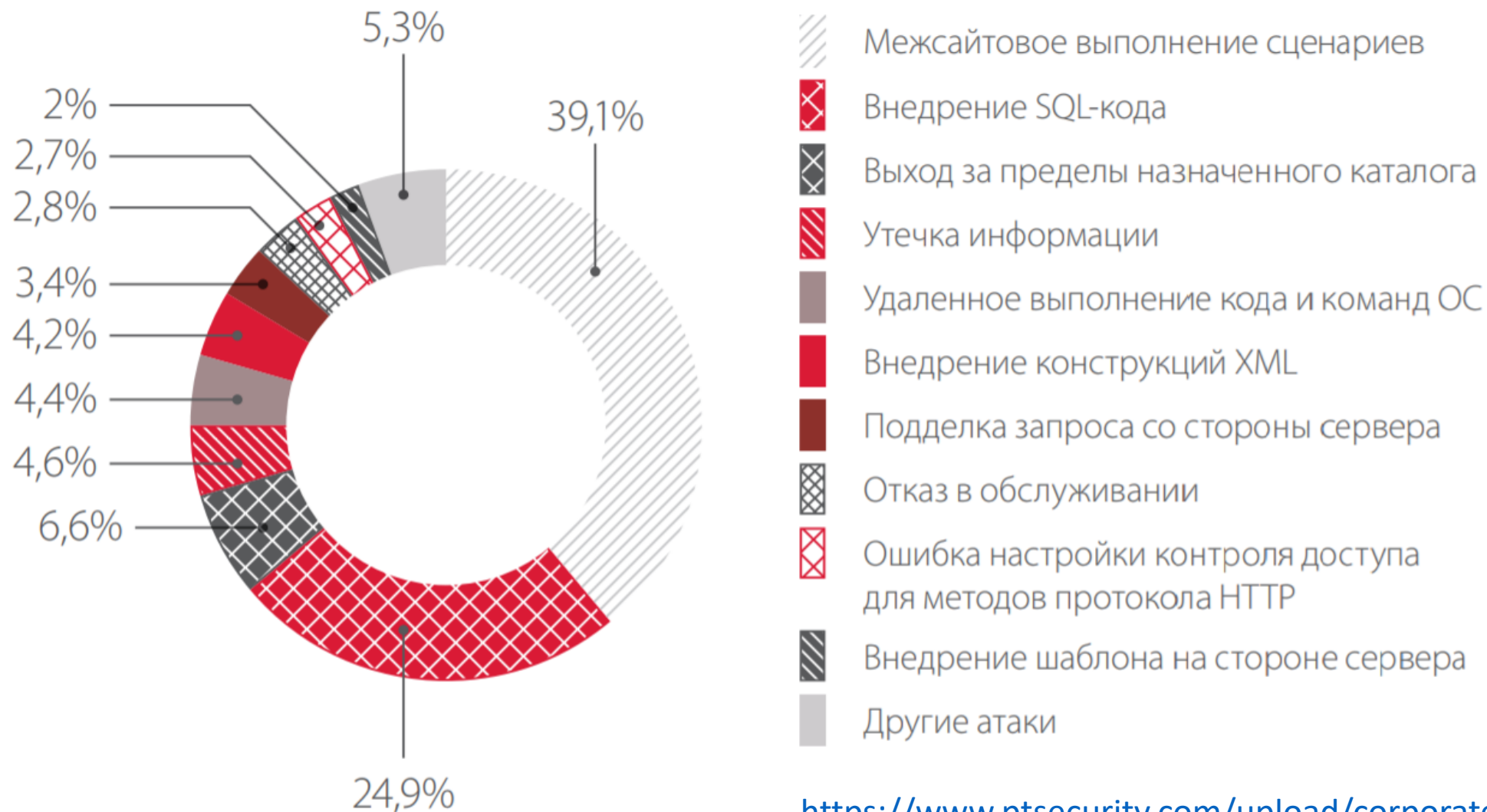
Кто сказал, что речь только об SQL?

- Интерпретируемый код
- Структурированные данные
- Код разметки
- ID внешних ресурсов
- Любые грамматики, с числом токенов > 1



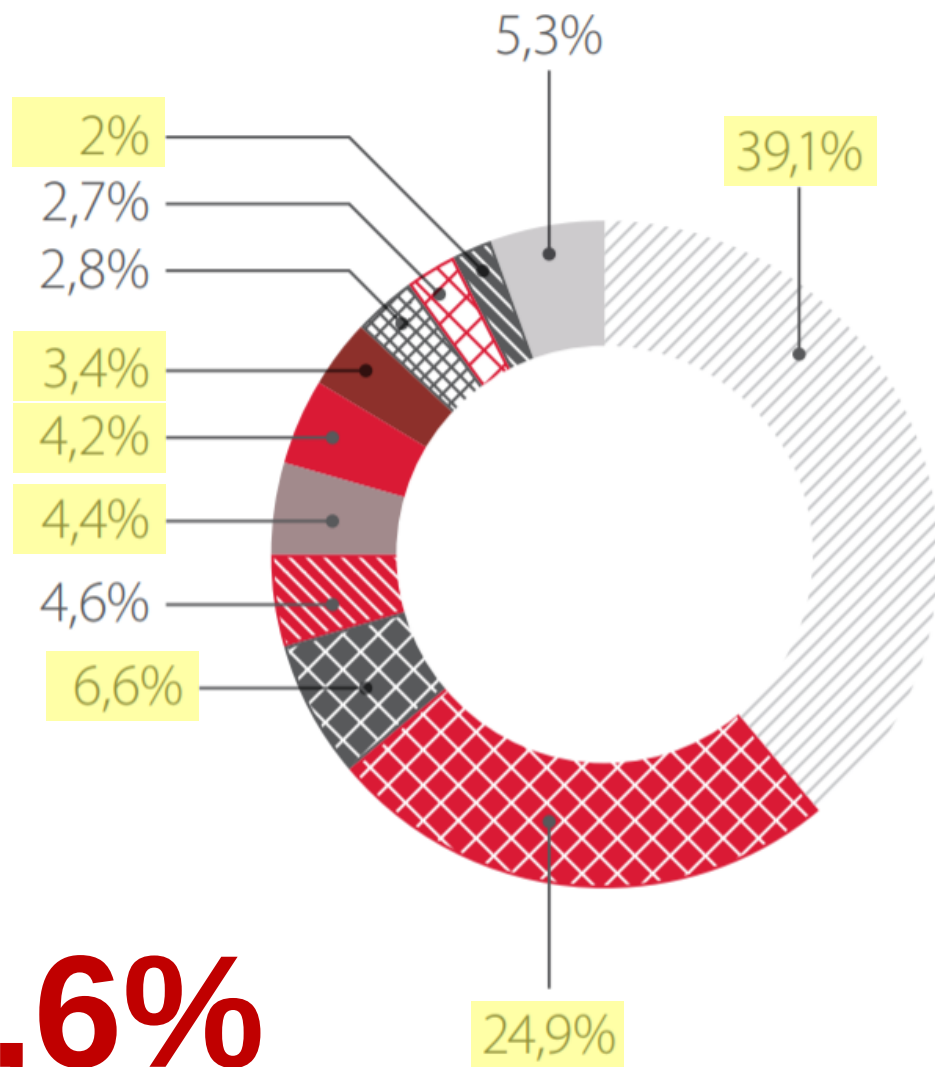
«ТАК – уже давно никто не пишет!»

Что же тогда атаковали хакеры в Q2.2017? (1/2)

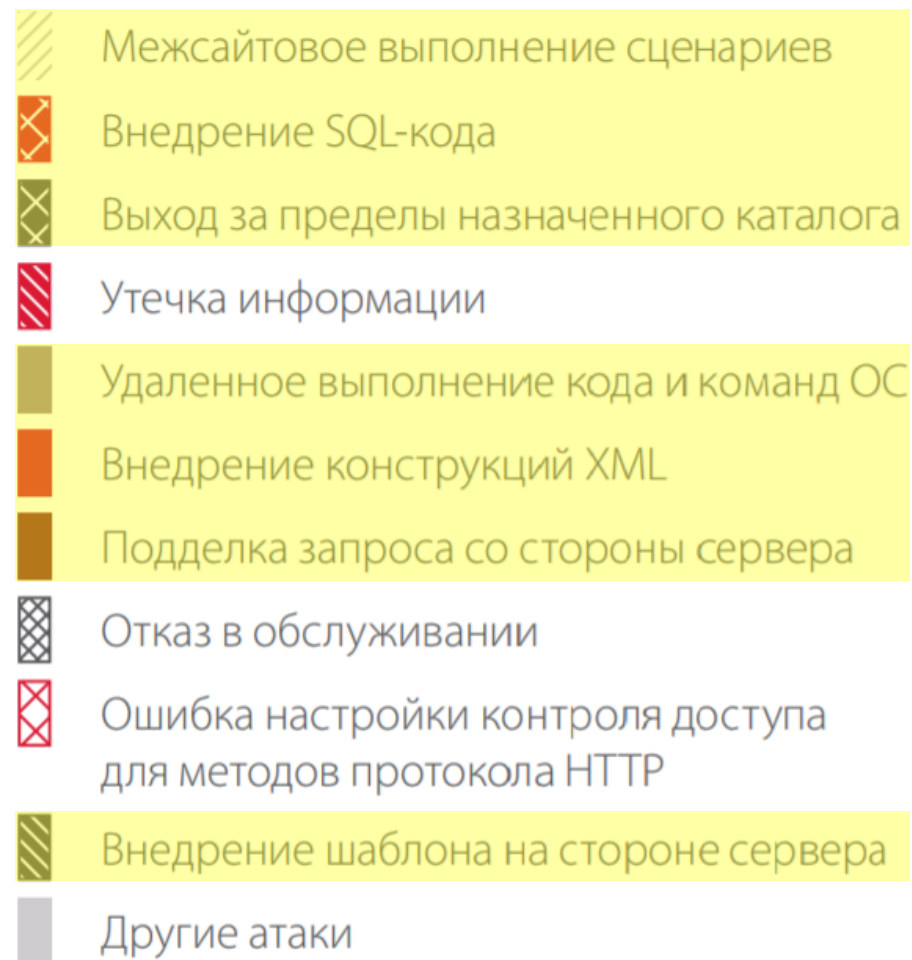


<https://www.ptsecurity.com/upload/corporate/ru-ru/analytics/WebApp-Vulnerabilities-2017-rus.pdf>

Что же тогда атаковали хакеры в Q2.2017? (2/2)



84,6%



<https://www.ptsecurity.com/upload/corporate/ru-ru/analytics/WebApp-Vulnerabilities-2017-rus.pdf>

ТАК – пишут даже в Microsoft (CVE-2017-8759)

```
<service name="Service">
  <port name="Port" binding="tns:Binding">
    <soap:address location="http://localhost?C:\Windows\System32\calc.exe?011"/>
    <soap:address location="";
    System.Diagnostics.Process.Start(_url.Split('?')[1], _url.Split('?')[2]);
    //"/>
  </port>
</service>
</definitions>
```

① - 898 office2.png nXML

```
public class Image : System.Runtime.Remoting.Services.RemotingClientProxy
{
  // Constructor
  public Image()
  {
    base.ConfigureProxy(this.GetType(), @"http://localhost?C:\Windows\System32\calc.exe?011");
    //base.ConfigureProxy(this.GetType(), @"
    System.Diagnostics.Process.Start(_url.Split('?')[1], _url.Split('?')[2]);
    //");
  }
}
```

«Инъекцию ещё надо суметь найти и проэксплуатировать!»

Ищем и эксплуатируем инъекции



<https://youtu.be/HyLk0Nbc8-4>

«Враг знает систему»

- если уязвимость может быть обнаружена, она будет обнаружена;
- если уязвимость может быть проэксплуатирована, она будет проэксплуатирована;
- при эксплуатации уязвимости будет нанесён максимально возможный ущерб.

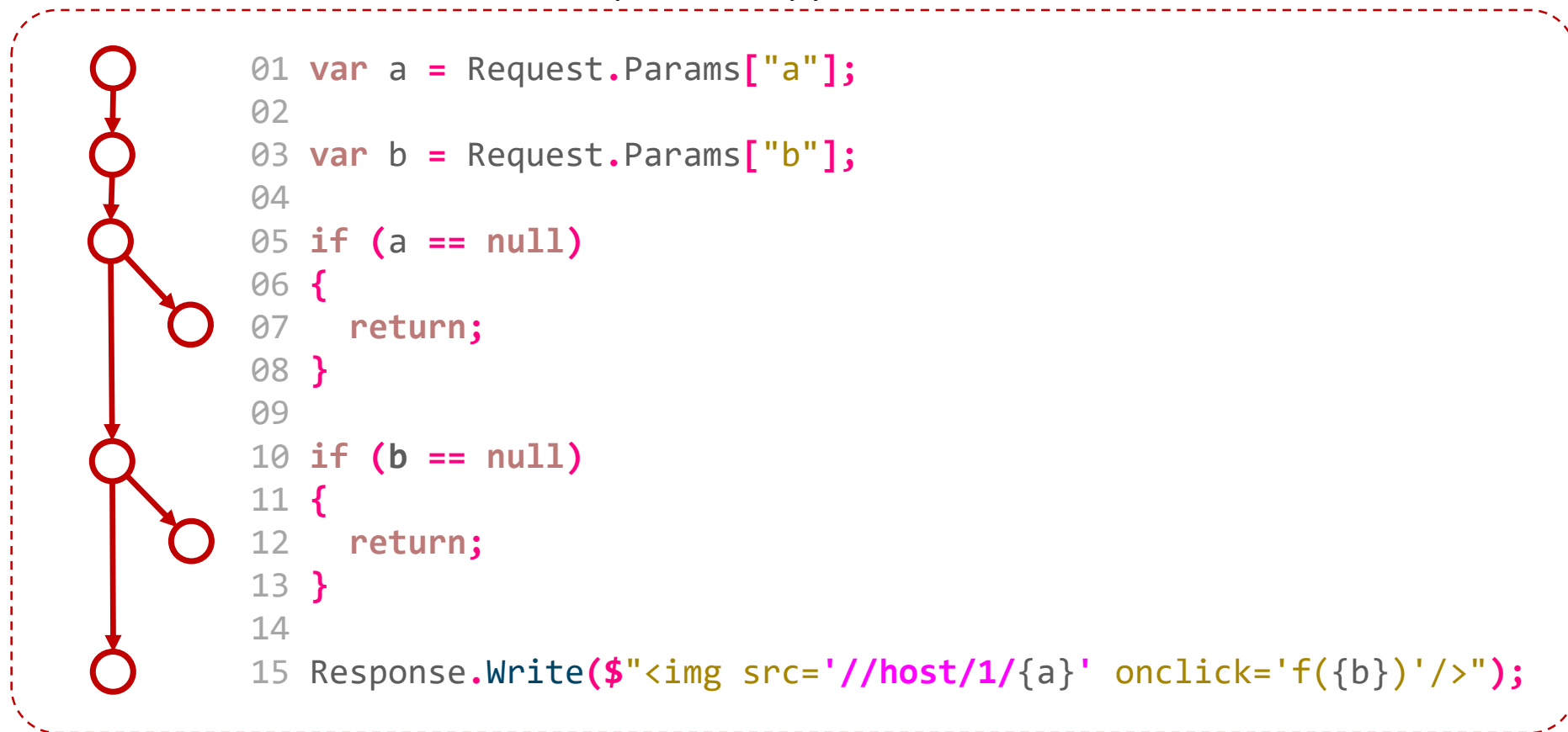
- Инъекции – хорошо формализуемый класс уязвимостей
- Существует формальный подход, как к разработке защищённого от инъекций кода, так и к защите существующего от атак на них



Формально об инъекциях

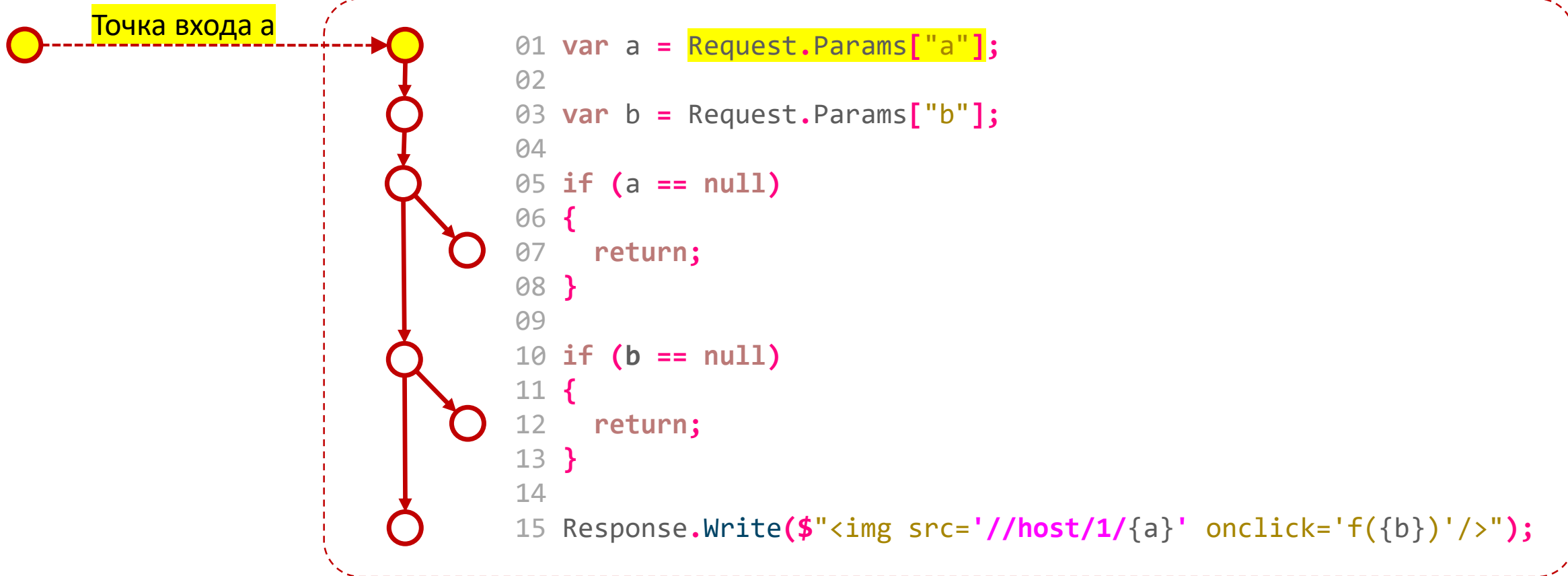
Модель уязвимого приложения (1/7)

Граница окружения



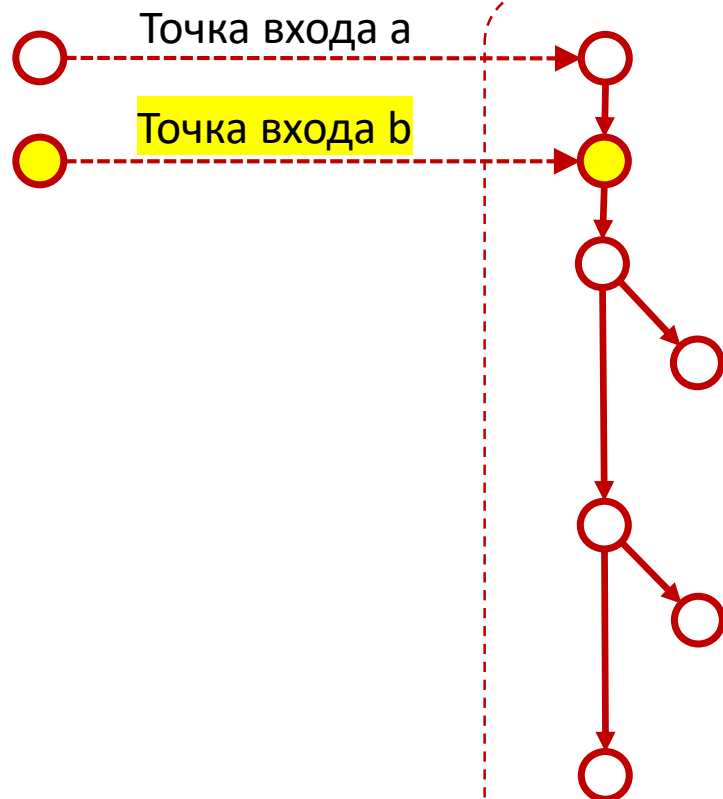
Модель уязвимого приложения (2/7)

Граница окружения



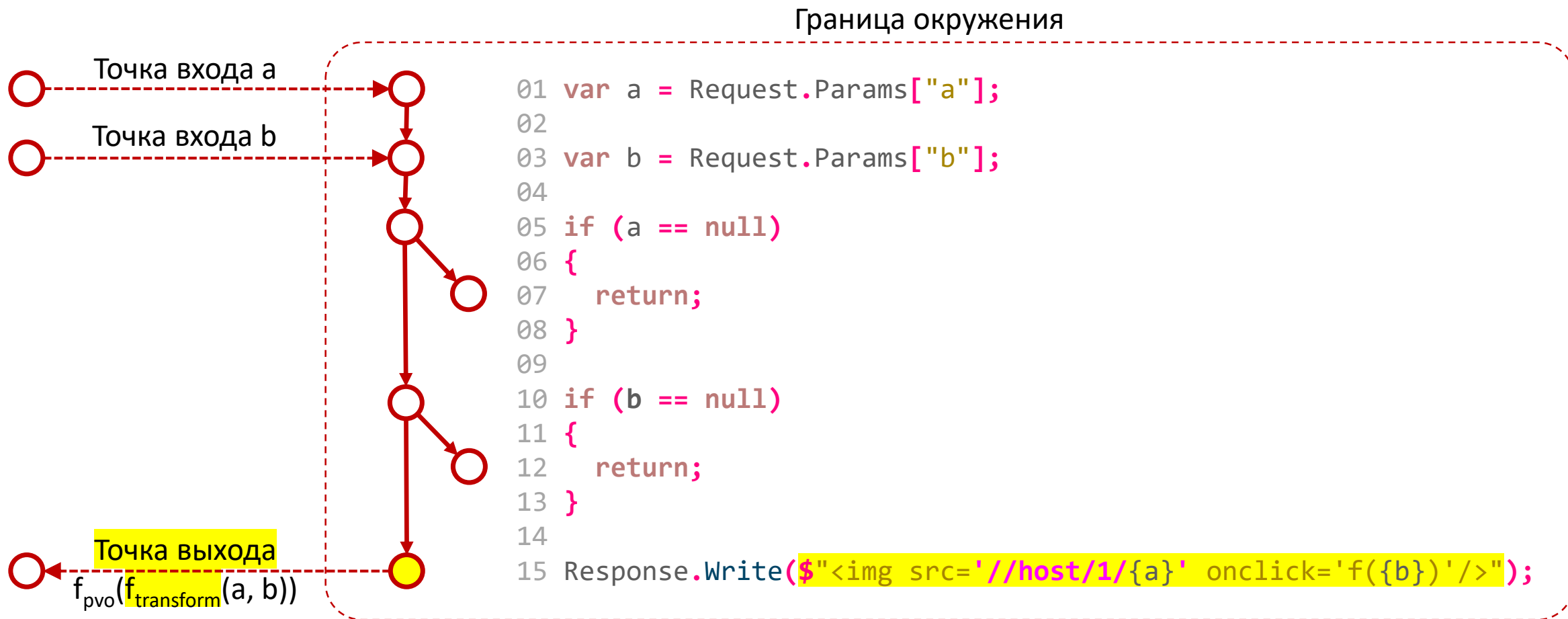
Модель уязвимого приложения (3/7)

Граница окружения

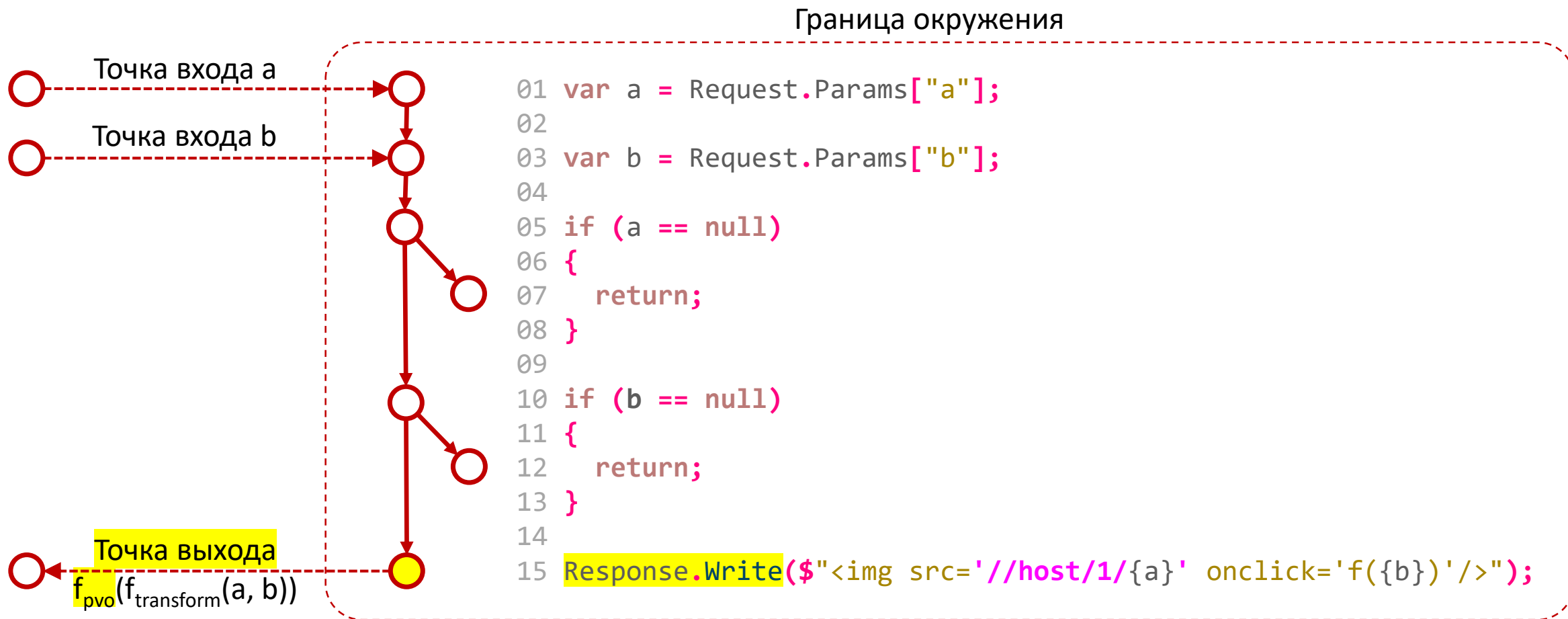


```
01 var a = Request.Params["a"];
02
03 var b = Request.Params["b"];
04
05 if (a == null)
06 {
07     return;
08 }
09
10 if (b == null)
11 {
12     return;
13 }
14
15 Response.Write($"<img src='//host/1/{a}' onclick='f({b})' />");
```

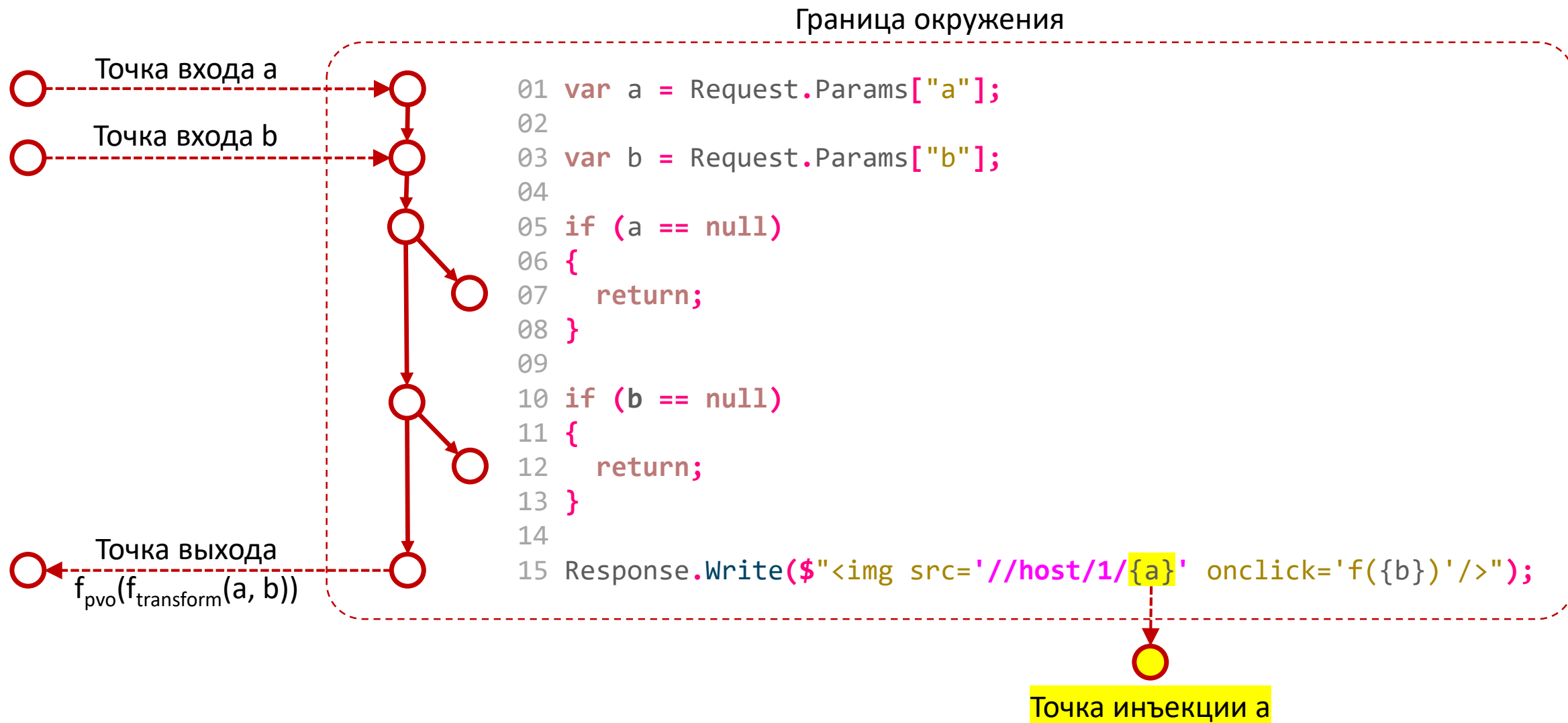
Модель уязвимого приложения (4/7)



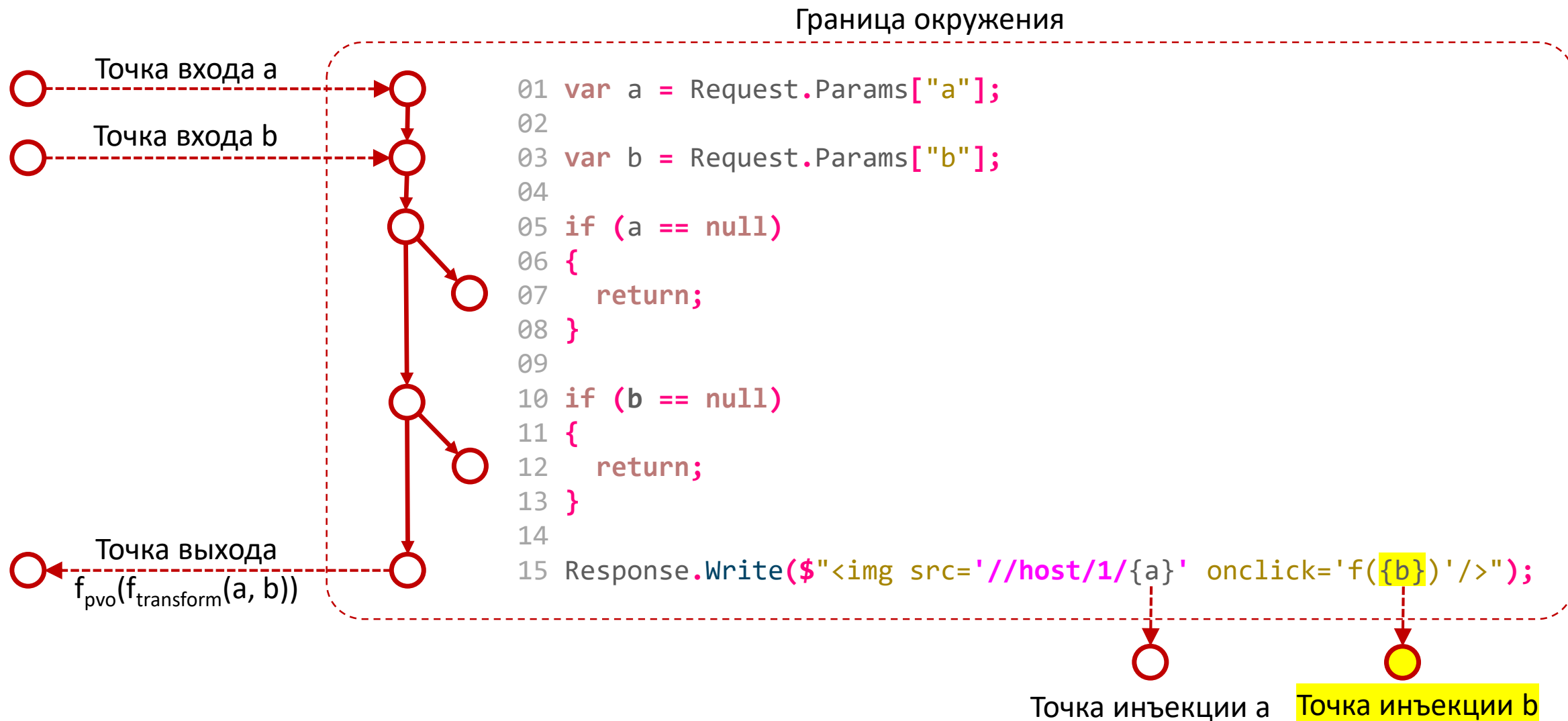
Модель уязвимого приложения (5/7)



Модель уязвимого приложения (6/7)



Модель уязвимого приложения (7/7)

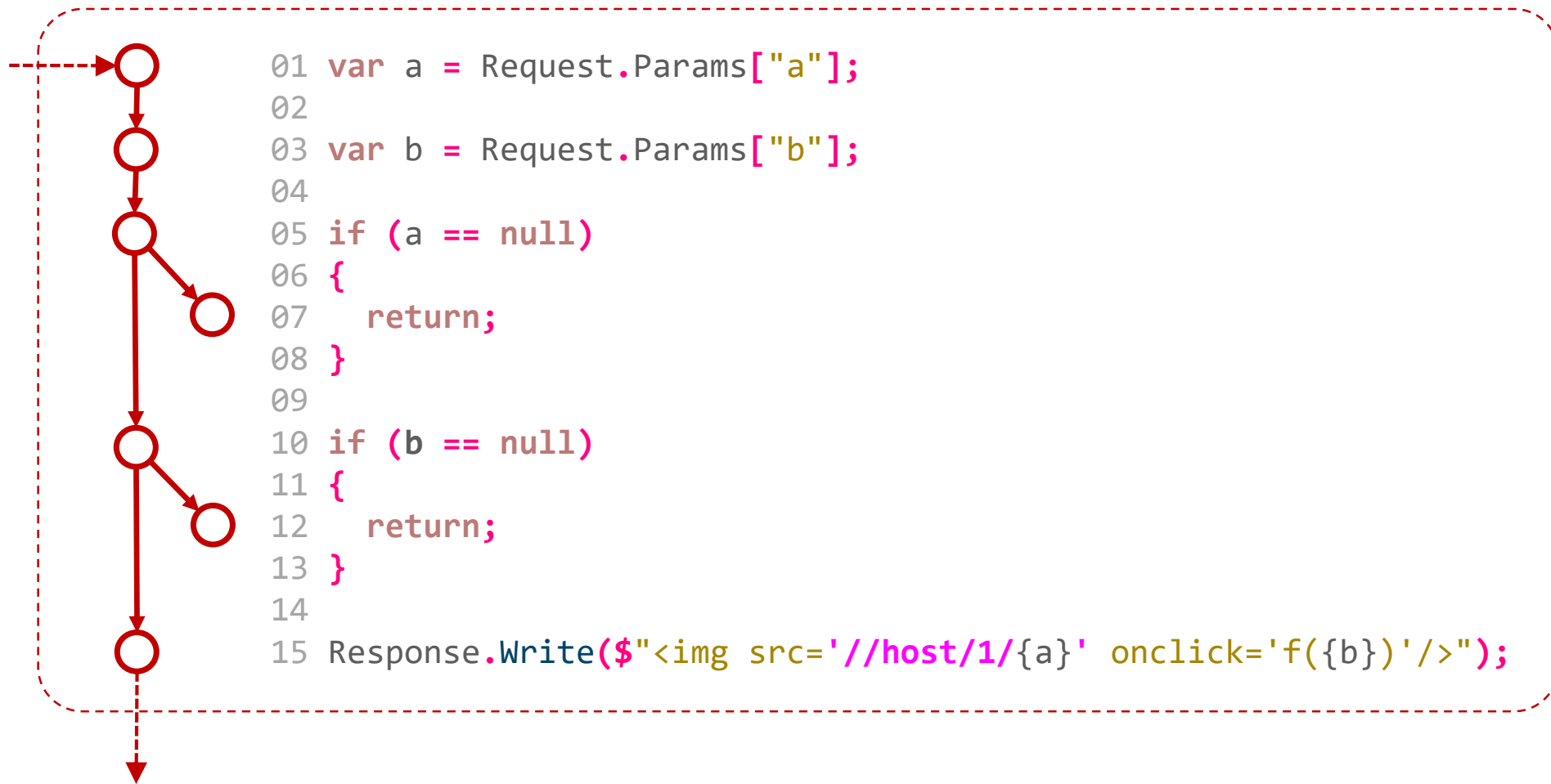


Атаки инъекции направлены на внедрение в точки инъекции грамматических конструкций, не предусмотренных логикой приложения

Модель атаки (HTML/URL, нет инъекции)

Граница окружения

image.jpg

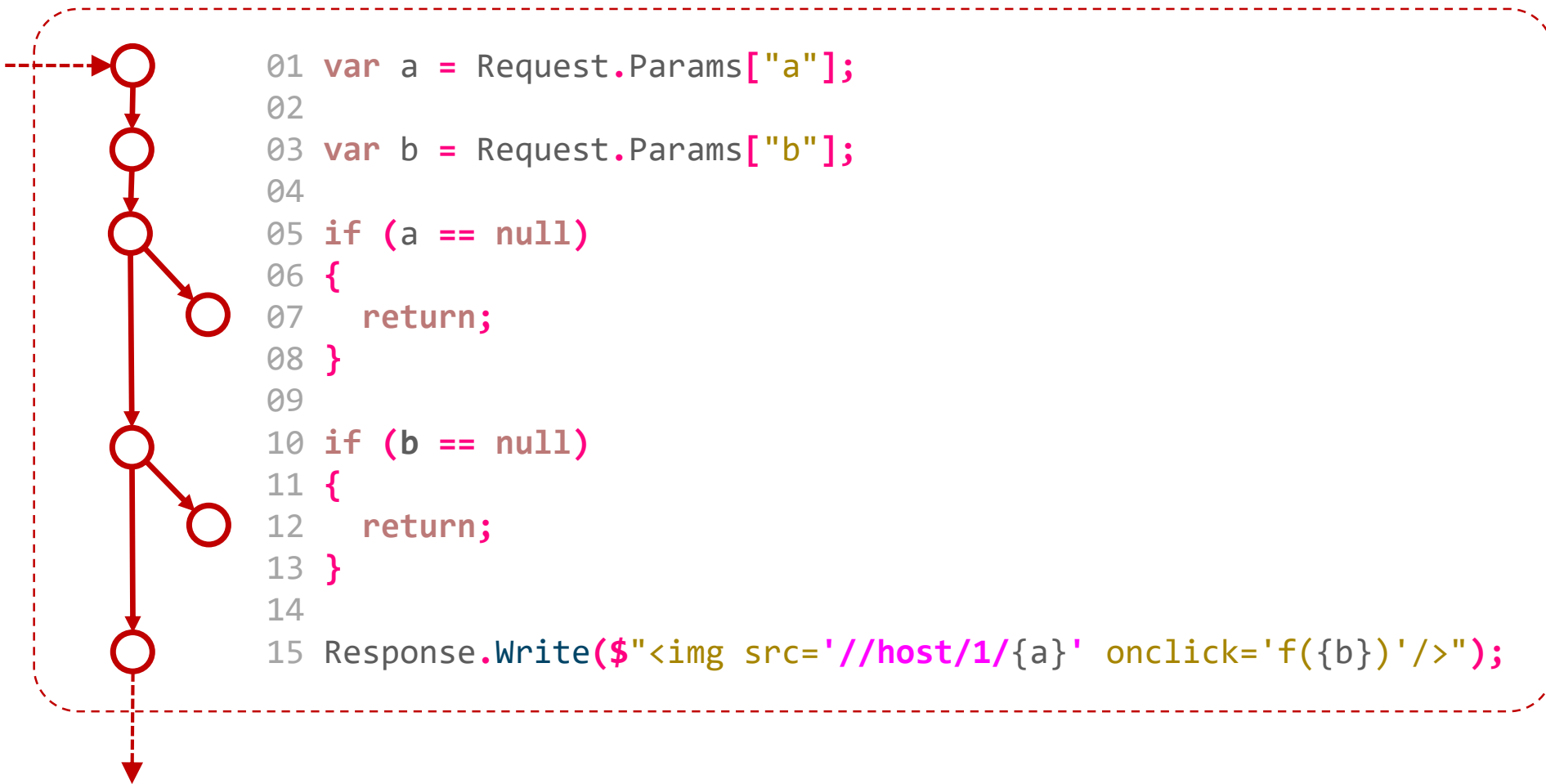


< img src = ' // host / 1 / image.jpg ' ...

Модель атаки (HTML/URL, инъекция URL-пути)

Граница окружения

../image.jpg



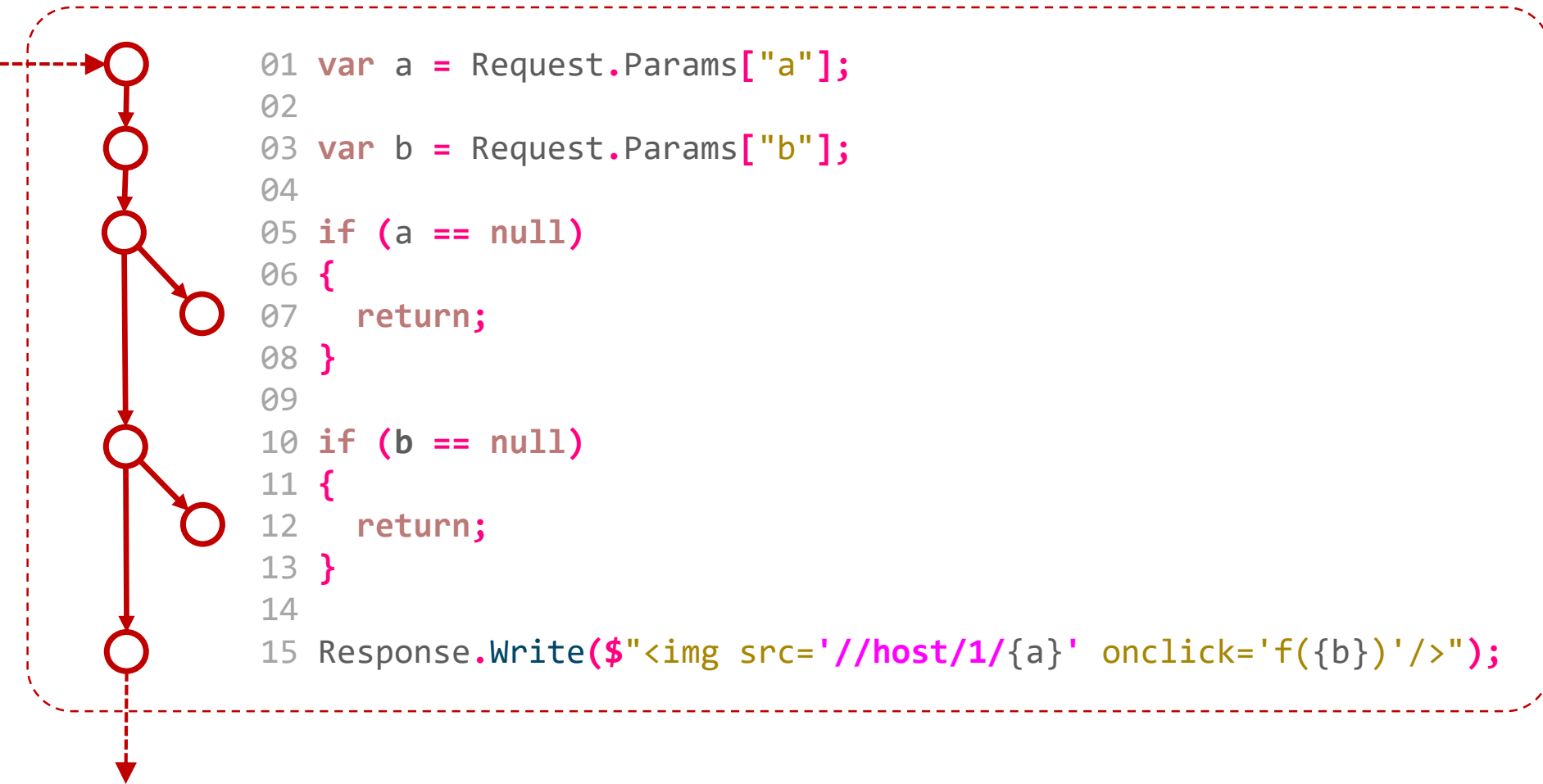
```
01 var a = Request.Params["a"];
02
03 var b = Request.Params["b"];
04
05 if (a == null)
06 {
07     return;
08 }
09
10 if (b == null)
11 {
12     return;
13 }
14
15 Response.Write($"<img src='//host/1/{a}' onclick='f({b})' />");
```

< img src = ' // host / 1 / ../ image.jpg ' ...

Модель атаки (HTML/URL, инъекция HTML-атрибута)

Граница окружения

'onerror='alert(0)



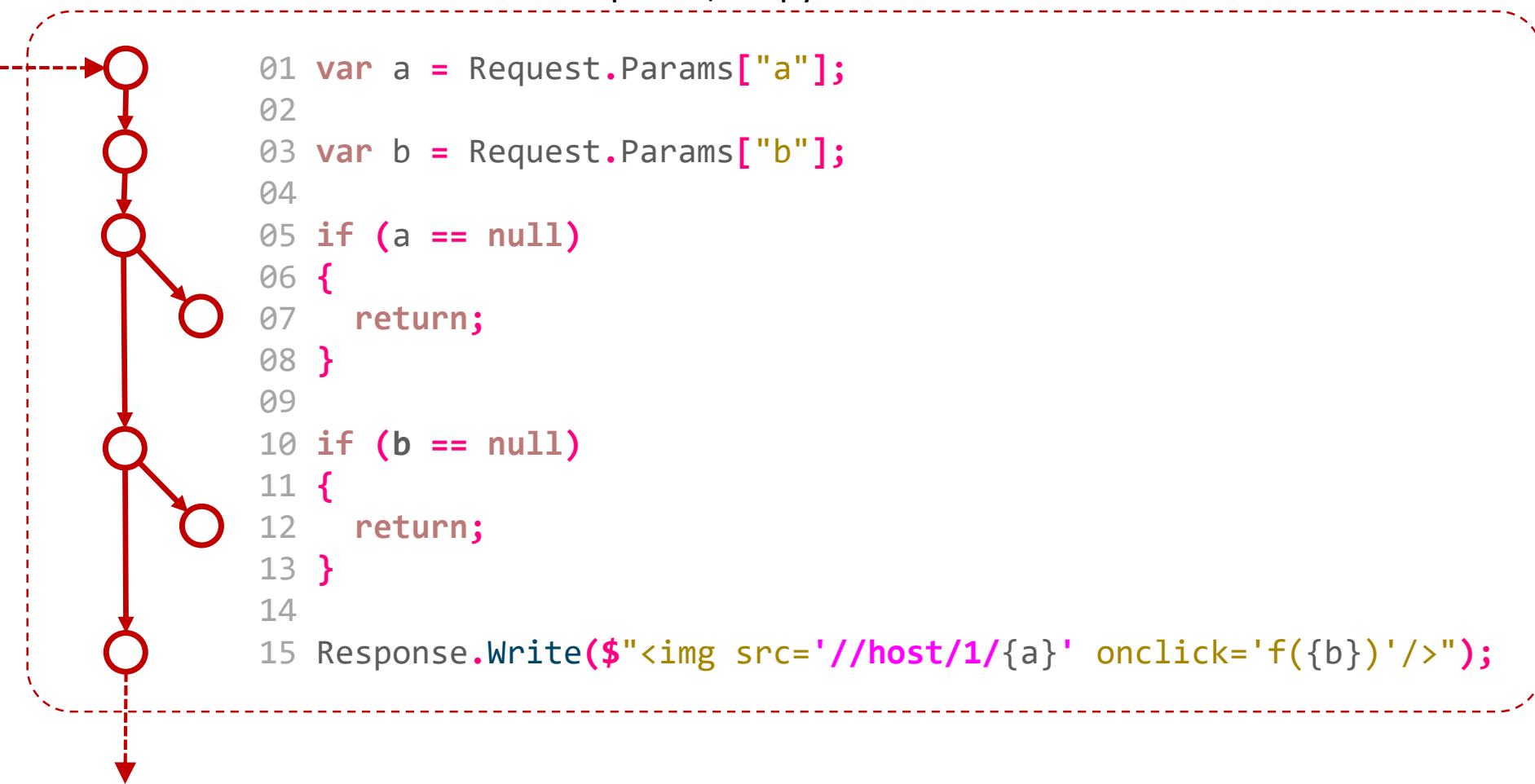
```
01 var a = Request.Params["a"];
02
03 var b = Request.Params["b"];
04
05 if (a == null)
06 {
07     return;
08 }
09
10 if (b == null)
11 {
12     return;
13 }
14
15 Response.Write($"<img src='//host/1/{a}' onclick='f({b})' />");
```

< img src = ' // host / 1 / ' onerror = ' alert (0) ' ...

Модель атаки (HTML/URL, инъекция HTML-тега)

Граница окружения

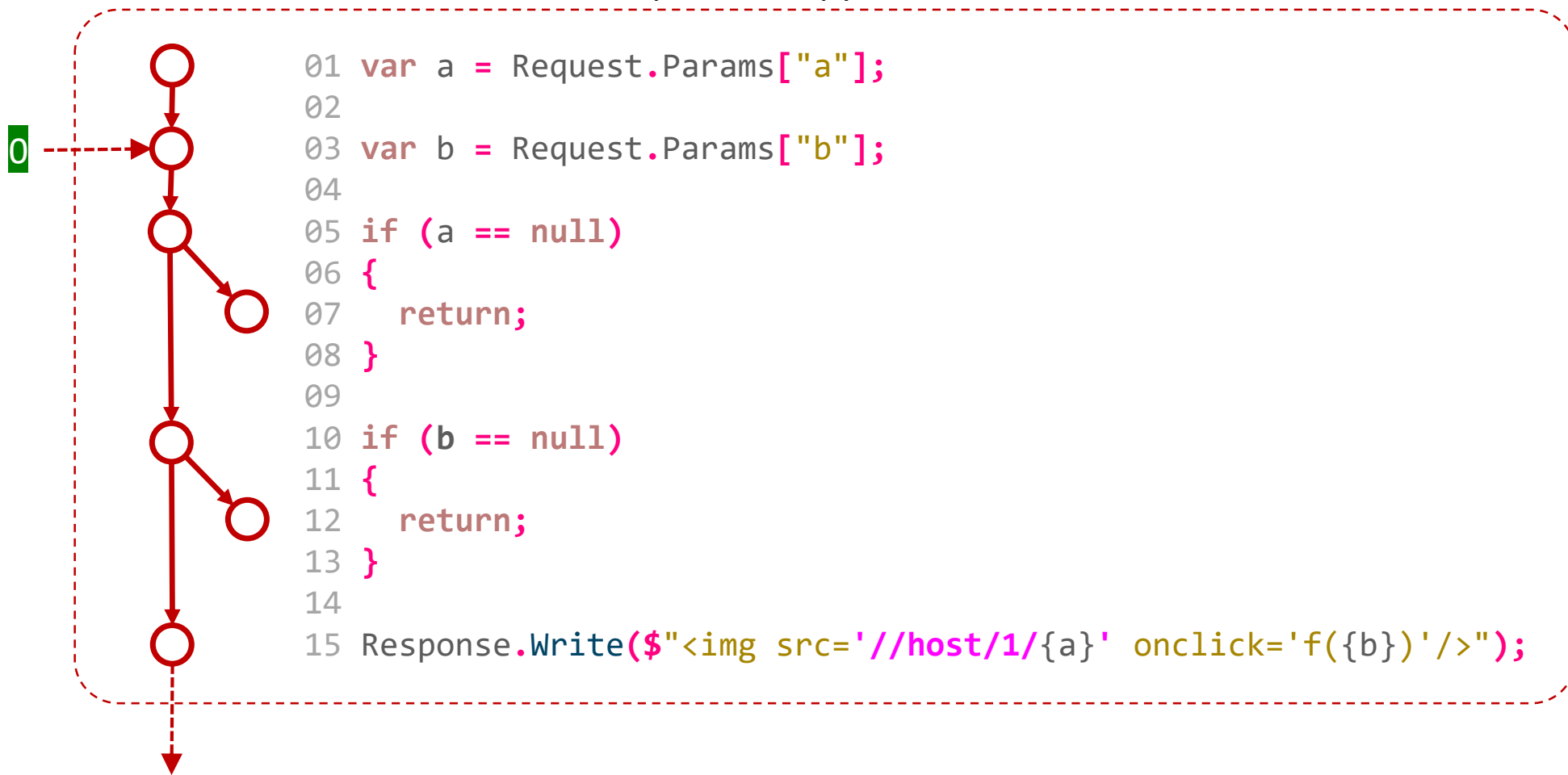
'><script>alert(0)</script>



< img src = ' // host / 1 / ' > < script > alert(0) < / script > ' ...

Модель атаки (HTML/JavaScript, нет инъекции)

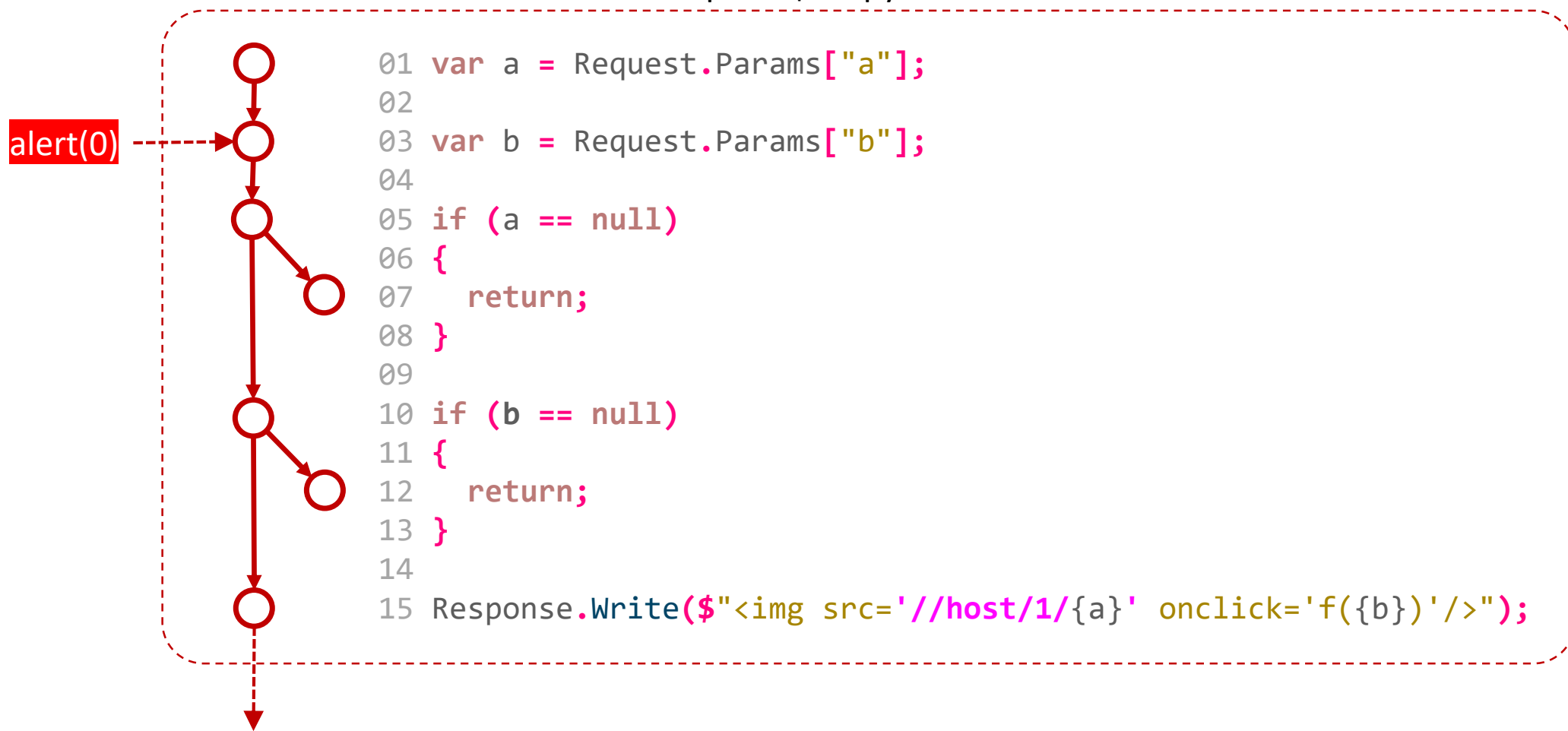
Граница окружения



... onclick = ' f (0) ' />

Модель атаки (HTML/JavaScript, инъекция JavaScript-выражения #1)

Граница окружения

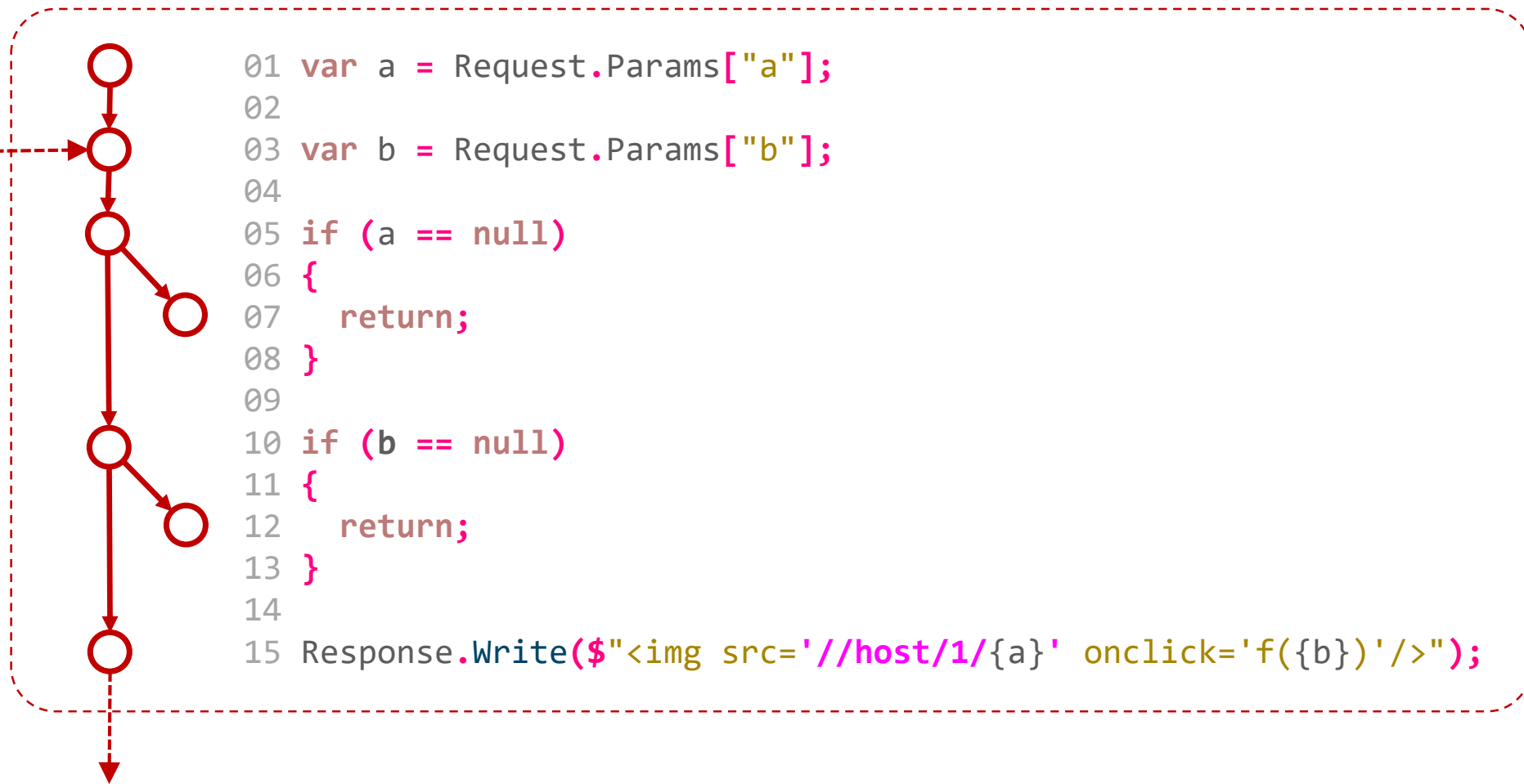


... `onclick = ' f ( alert ( 0 ) ) ' />`

Модель атаки (HTML/JavaScript, инъекция JavaScript-выражения #2)

Граница окружения

0); alert(0)

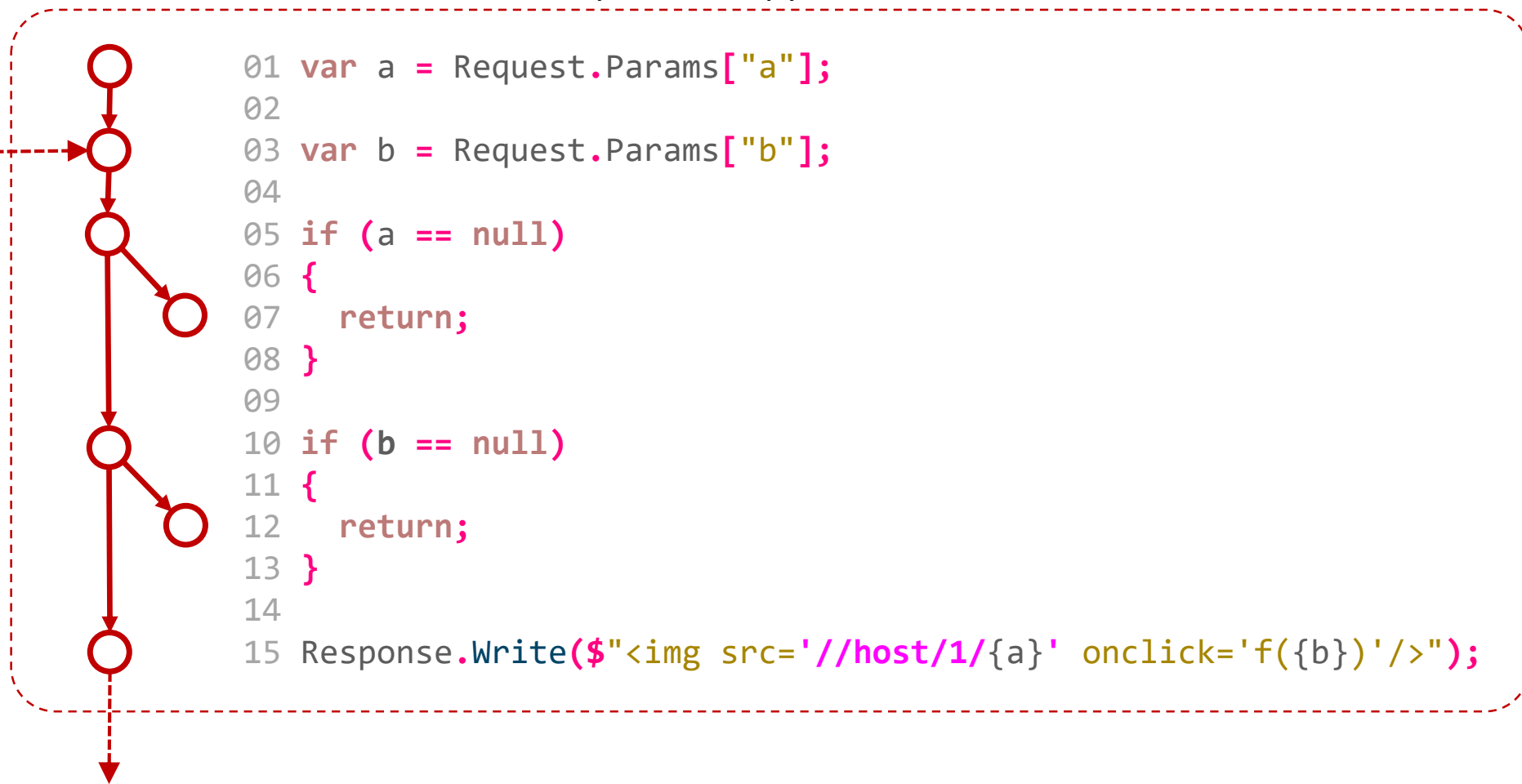


... onclick = ' f (0) ; alert (0) ' />

Модель атаки (HTML/JavaScript, инъекция HTML-атрибута)

Граница окружения

'onclick='alert(0

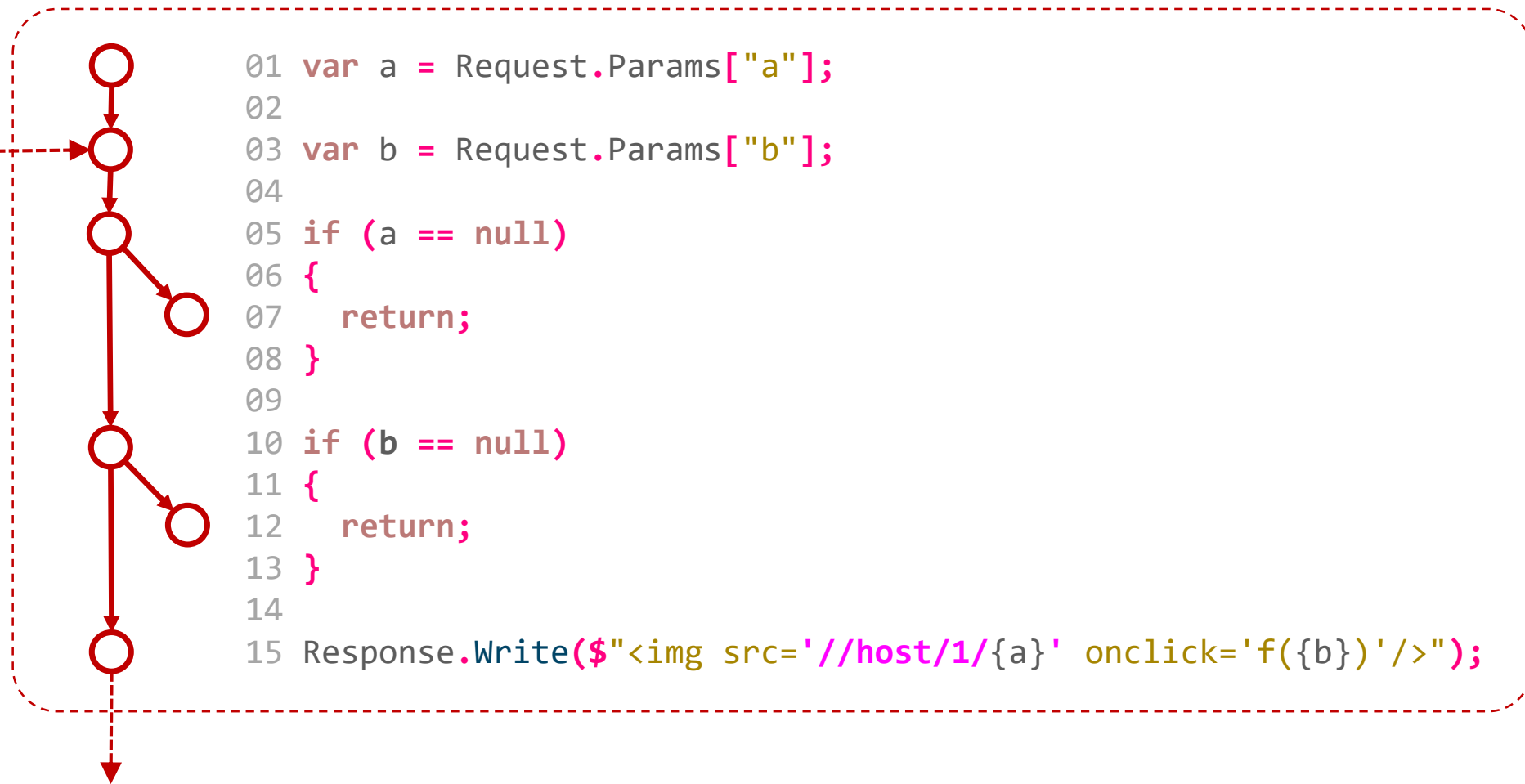


... onclick = ' f (' onclick = ' alert (0) ' />

Модель атаки (HTML/JavaScript, инъекция HTML-тега)

Граница окружения

'><script>alert(0)</script>



... onclick = ' f (' > < script > alert (0) < / script >) ' />

Посчитаем токены?

- Нет атаки – 1 токен в точке инъекции
- Есть атака – более 1 токена в точке инъекции

Приложение защищено от атак инъекций тогда, когда в результате лексического разбора любого возможного выходного потока данных, на каждую точку инъекции приходится не более одного токена.

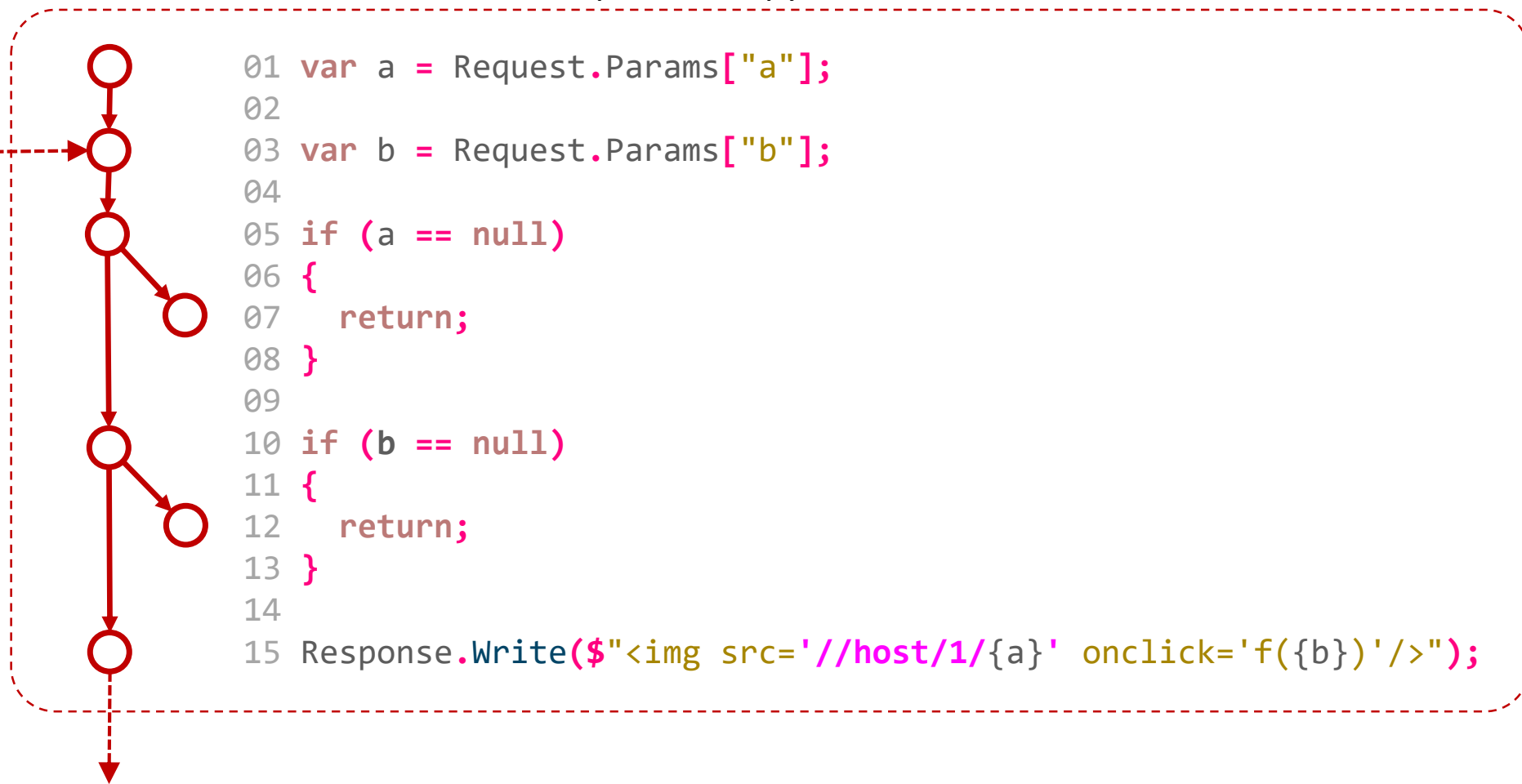
Достаточный критерий не всегда применим

Что если сама логика приложения подразумевает наличие более одного токена в какой-либо точке инъекции?

Модель атаки (HTML/JavaScript, нет инъекции)

Граница окружения

`{ "f1": "val1", "f2": "val2" }`



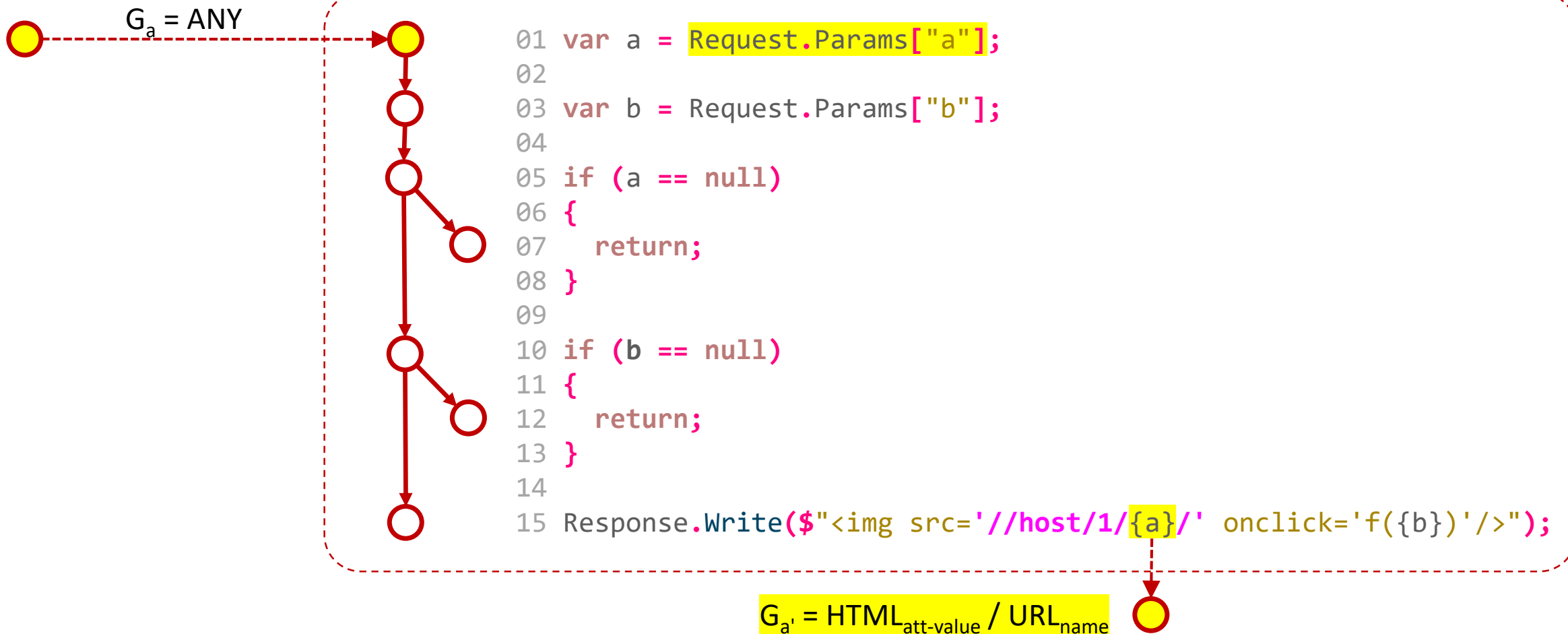
... `onclick = ' f ( { "f1" : "val1" , "f2" : "val2" } ) ' />`

Необходимый критерий защищённости от атак инъекций

Приложение защищено от атак инъекций тогда, когда в результате лексического разбора любого возможного потока выходных данных, множества токенов, приходящиеся на точки инъекции, являются авторизованными.

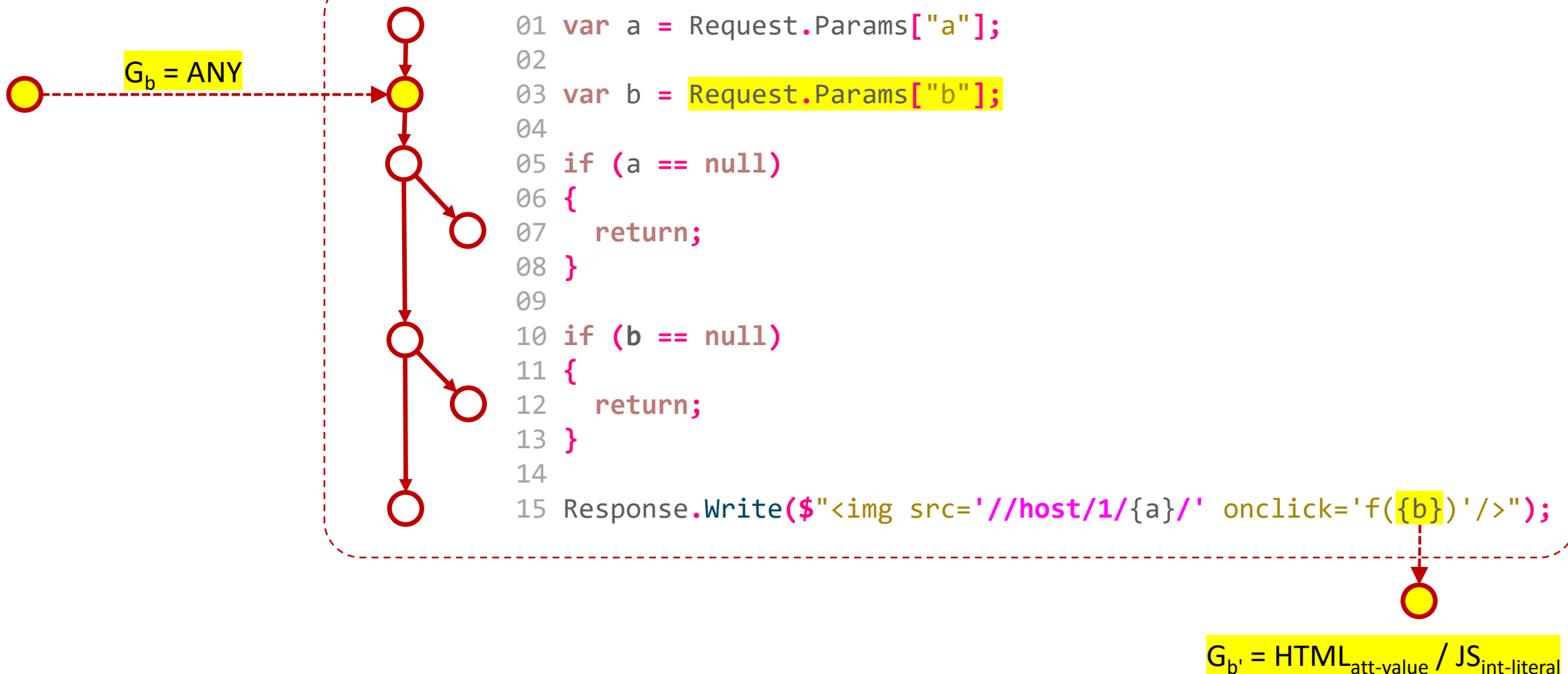
Причина инъекции – несогласованность грамматик (1/3)

Граница окружения



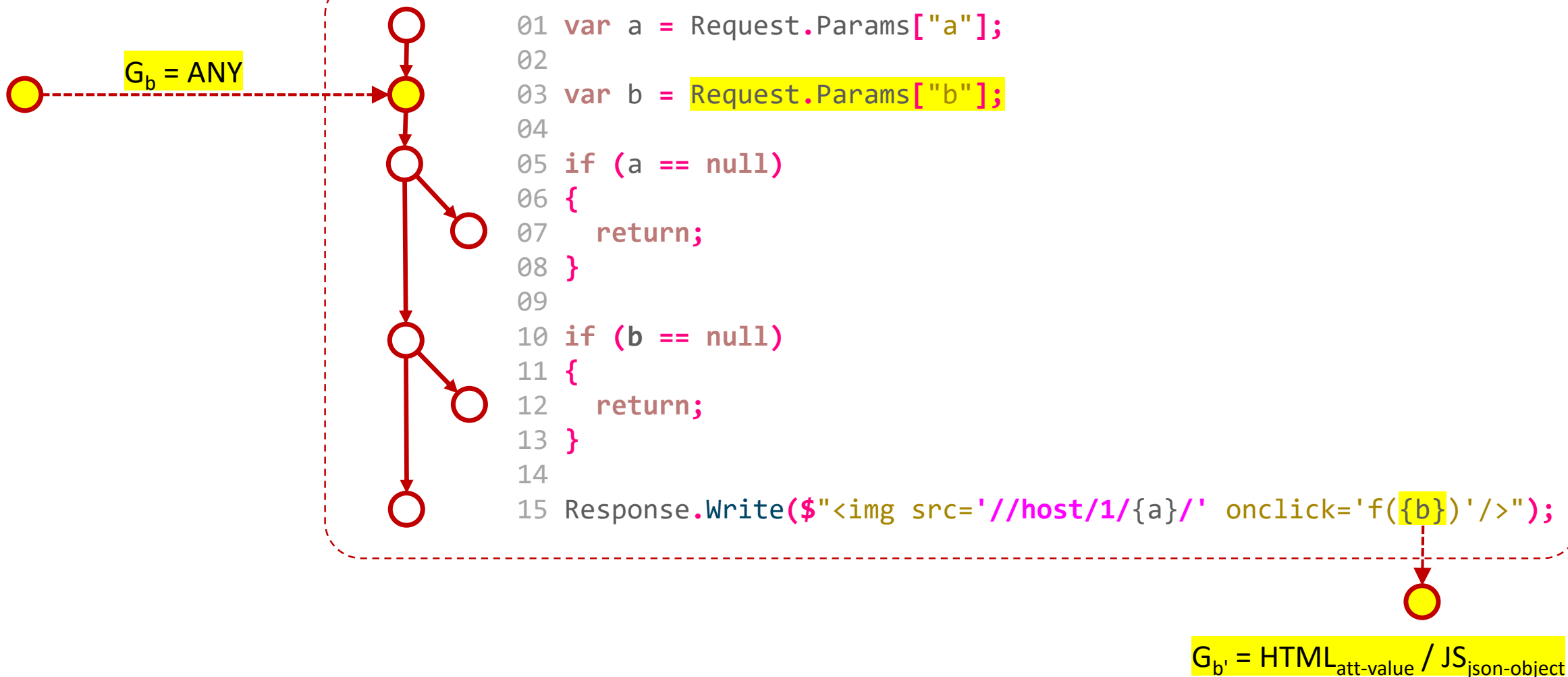
Причина инъекции – несогласованность грамматик (2/3)

Граница окружения



Причина инъекции – несогласованность грамматик (3/3)

Граница окружения



Как победить инъекции?

Победить инъекции возможно, обеспечив выполнение одного из критериев через согласование грамматик входных данных и данных в точках инъекции

Входные данные должны согласовываться с бизнес-логикой приложения за счёт:

- Типизации

- Валидации

- Синтаксической

- Семантической



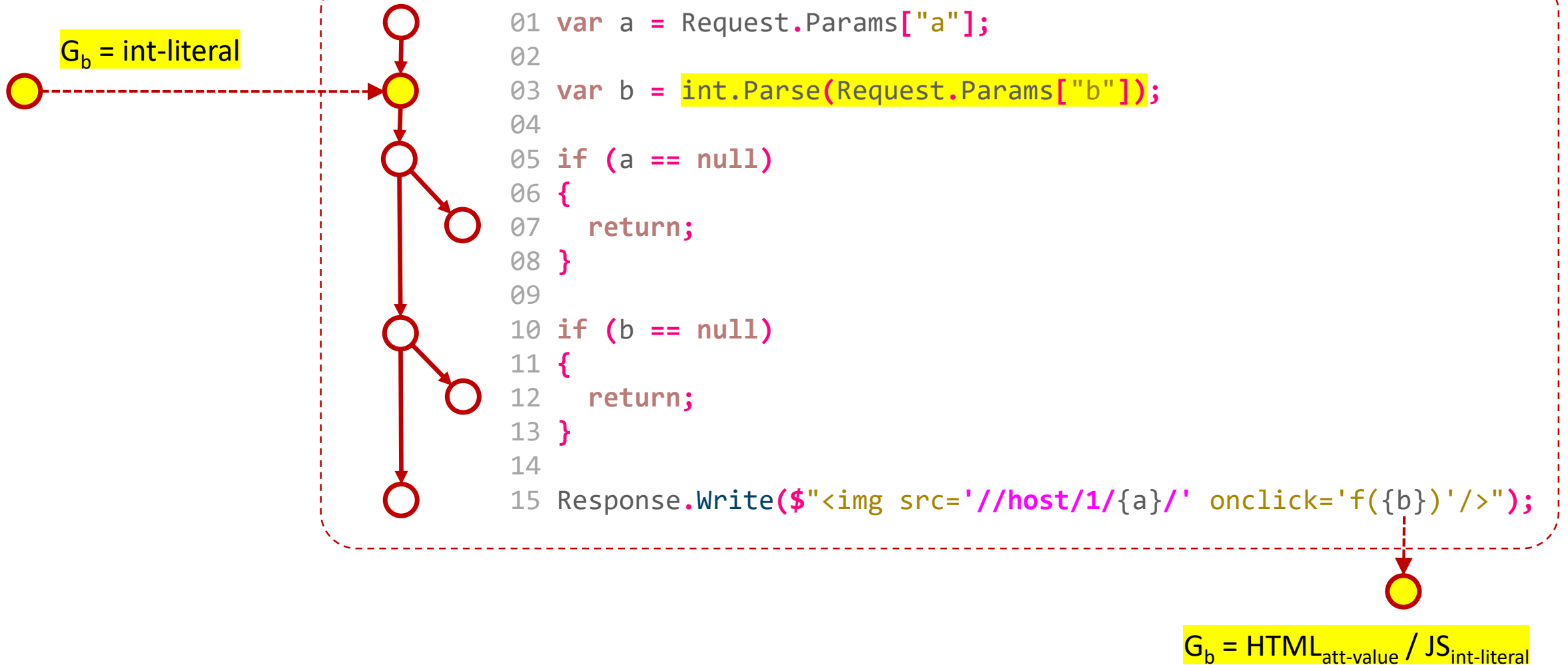
Обеспечение необходимого или достаточного критериев

Типизация – приведение входных данных к конкретному типу:

```
var b = int.Parse(Request.Params["b"])
```

Согласование грамматик (типизация)

Граница окружения



Синтаксическая валидация – проверка входных данных на соответствие какой-либо грамматике:

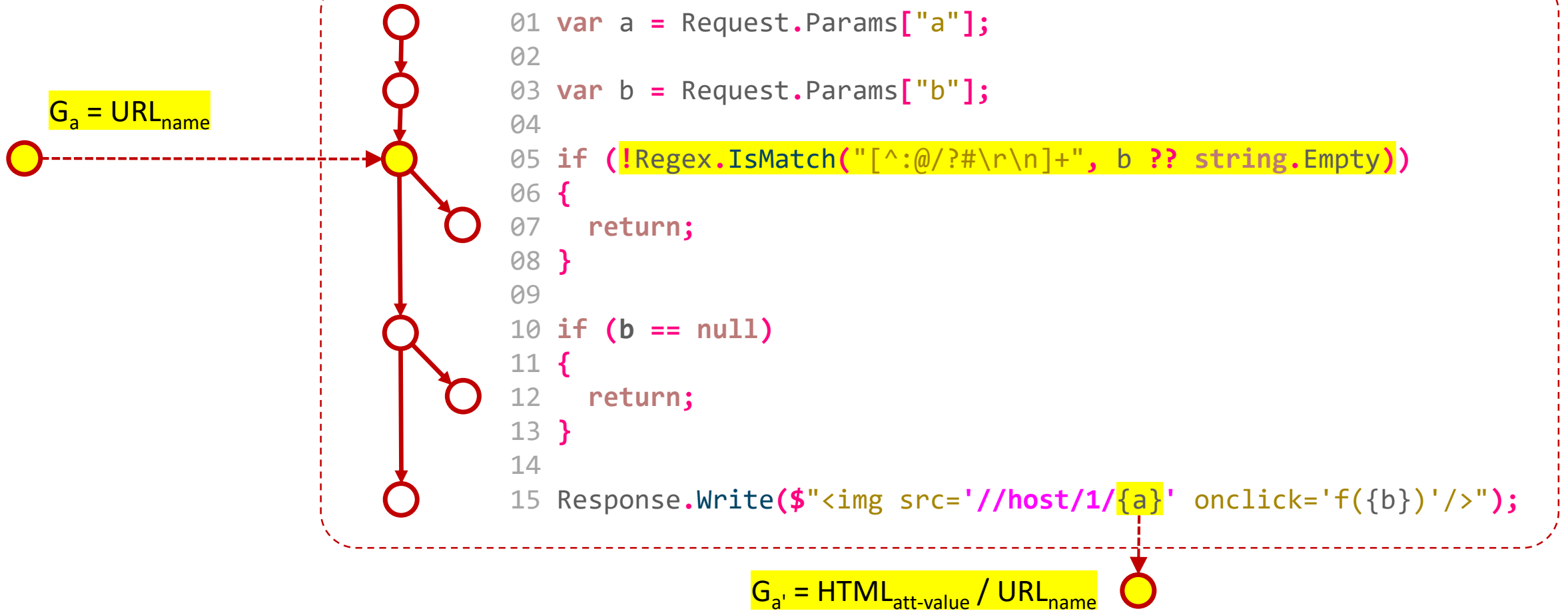
```
01 var url_regex =  
02     "^(([^:/?#]+):)?(//(?:[^/?#]*))?(^[?#]*(\\?([^#]*))?(#(\\.*)?)?" ;  
03  
04 if (!Regex.IsMatch(Request.Params["a"], url_regex))  
05 {  
06     throw new ValidationException();  
07 }
```

Некоторые средства синтаксической валидации в .NET

- Разбор строки вручную (методы `System.String`)
- `System.Text.RegularExpressions`
- `System.Web.UI.WebControls.RegularExpressionValidator`
- `System.ComponentModel.DataAnnotations`
 - `RegularExpressionAttribute`
 - `StringLengthAttribute`

Согласование грамматик (синтаксическая валидация)

Граница окружения



Семантическая валидация – проверка входных данных на корректность с точки зрения логики приложения:

```
01 var schema = JSchema.Parse("some JSON schema");
02 var object = JObject.Parse(Request.Params["b"]);
03
04 if (!object.IsValid(schema))
05 {
06     throw new ValidationException();
07 }
```

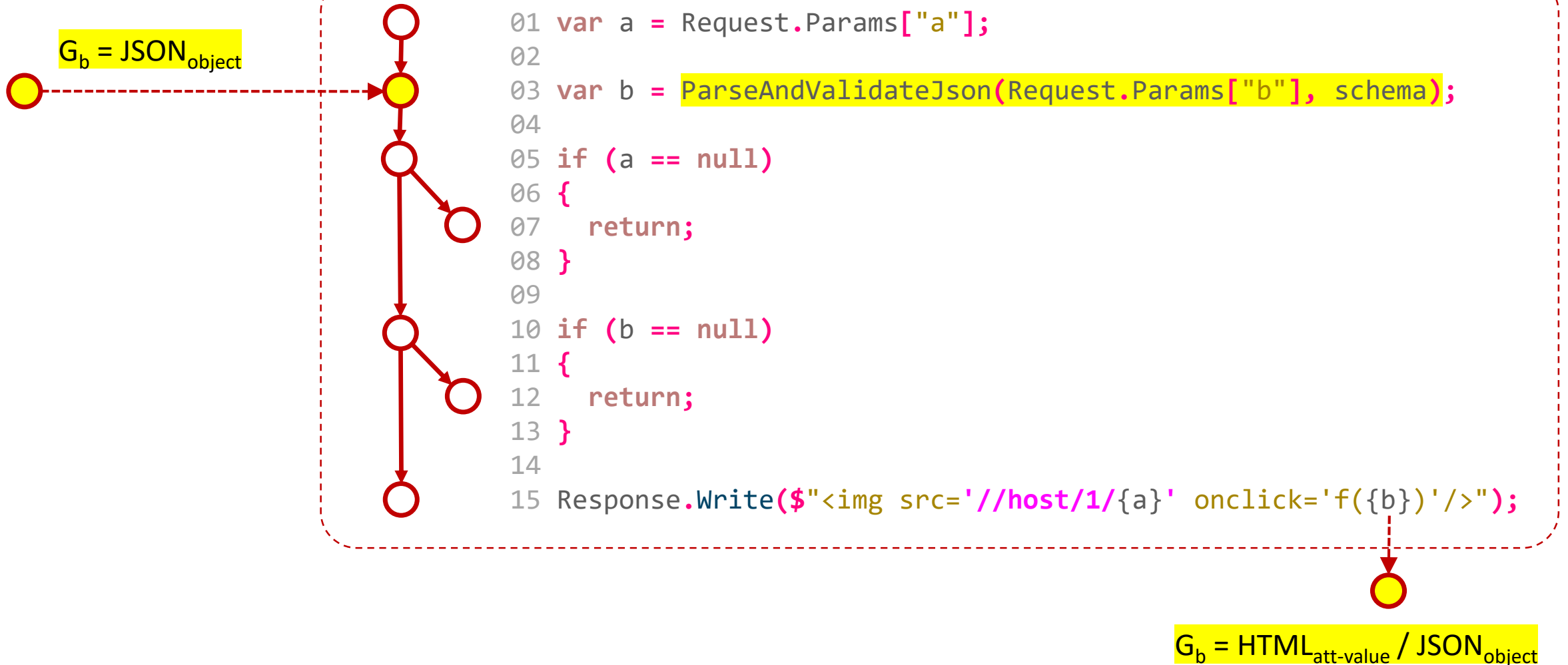
Некоторые средства семантической валидации в .NET

- Проверка свойств типизированных объектов
- `System.Web.UI.WebControls`
 - `CustomValidator`
 - `RangeValidator`
- `System.ComponentModel.DataAnnotations`
 - `CustomValidationAttribute`
 - `DataTypeAttribute`
 - `RangeAttribute`
- `System.Json`

Типизация и семантическая валидация не примитивных типов, может повлечь уязвимости к атакам других классов (SSRF, DoS, RCE и т.п.)

Согласование грамматик (семантическая валидация)

Граница окружения



Выходные данные должны согласовываться с грамматикой принимающей стороны за счёт санитизации



Обеспечение достаточного критерия

Санитизация – преобразование строковых данных к синтаксису какого-либо токена заданной грамматики

```
01 Response.Write(  
02     $"<a href='//host/1/{UrlEncode(Request.Params["a"])}'>...</a>"  
03 );
```

Санитизация вложенных грамматик

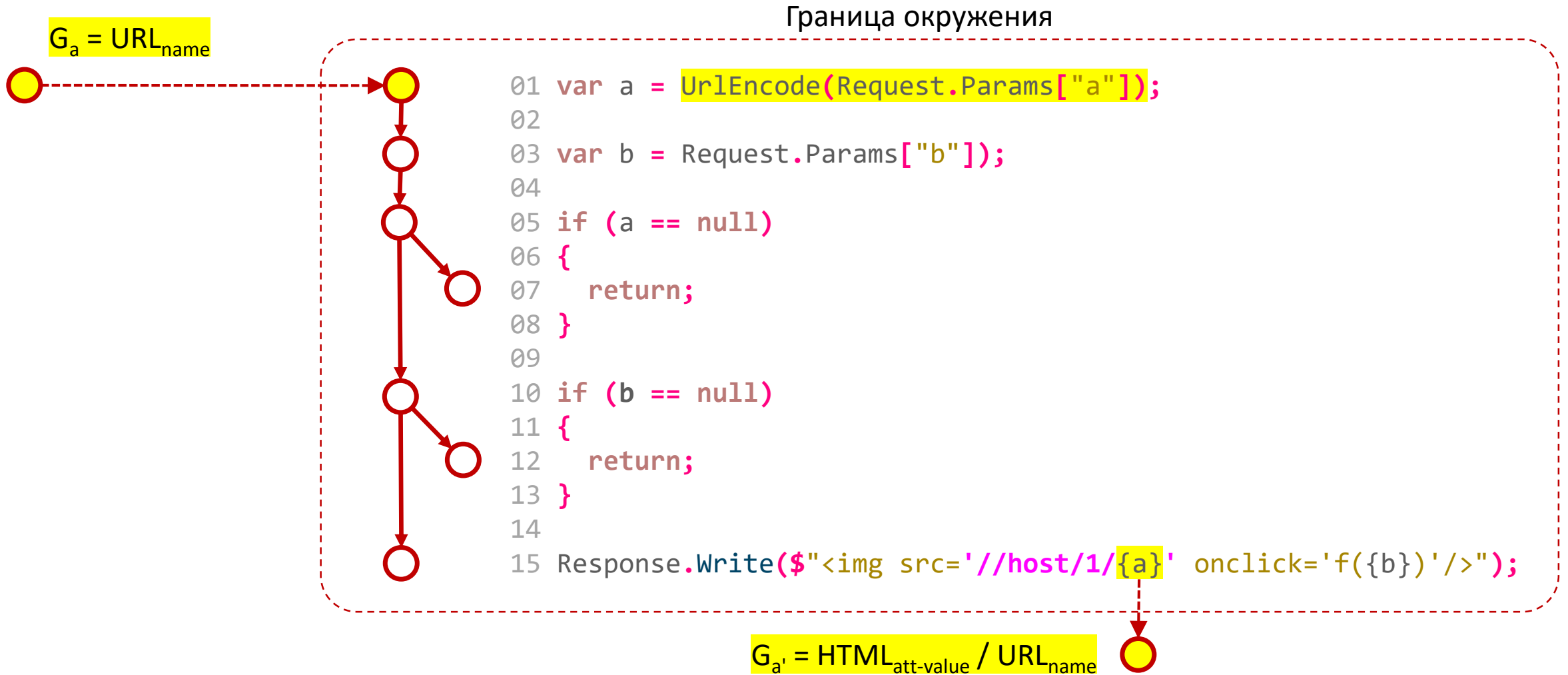
В случае вложенных грамматик, например:

$G = \text{JavaScript}_{\text{context}} / \text{URL}_{\text{context}}$

санитизацию необходимо проводить последовательно, в обратном порядке:

$\text{JavaScriptEncode}_{\text{context}}(\text{UrlEncode}_{\text{context}}(t)).$

Согласование грамматик (санитизация)



Чтобы победить инъекции разработчик должен: (1/2)

- выделить в приложении все точки входа данных;
- выделить все точки инъекции производных от них данных;
- провести типизацию и валидацию в точках входа с учётом требований бизнес-логики;

Чтобы победить инъекции разработчик должен: (2/2)

- провести санитизацию в точках инъекций, с учётом ожидаемых в них грамматик принимающей стороной;
- (опционально) подтверждать авторизованность множества токенов в точках инъекций.

CVE-2017-8759

**(инъекция произвольного C#-кода в
System.Runtime.Remoting. MetadataServices.WsdlParser)**

- Загрузка WSDL сервиса
- Генерация исходного кода прокси (PrintClientProxy)
- Компиляция прокси с помощью CSC
- Создание экземпляра прокси и взаимодействие с ним

PrintClientProxy

```
01 sb.Length = 0;
02 sb.Append(indent2);
03 if (i == 0)
04 {
05     sb.Append("base.ConfigureProxy(this.GetType(), ");
06     sb.Append(WsdlParser.IsValidUrl((string)_connectURLs[i]));
07     sb.Append(");");
08 }
09 else
10 {
11     // Only the first location is used, the rest are commented out in the proxy
12     sb.Append("//base.ConfigureProxy(this.GetType(), ");
13     sb.Append(WsdlParser.IsValidUrl((string)_connectURLs[i]));
14     sb.Append(");");
15 }
16 textWriter.WriteLine(sb);
```

IsValidUrl

```
01 if (value == null) { return "\"\""; }
02 vsb.Length= 0;
03 vsb.Append("@\"");
04
05 for (int i=0; i<value.Length; i++)
06 {
07     if (value[i] == '\\') vsb.Append("\\");
08     else vsb.Append(value[i]);
09 }
10
11 vsb.Append("\\");
12 return vsb.ToString();
```

В результате...

```
<service name="Service">
  <port name="Port" binding="tns:Binding">
    <soap:address location="http://localhost?C:\Windows\System32\calc.exe?011"/>
    <soap:address location="";
    System.Diagnostics.Process.Start(_url.Split('?')[1], _url.Split('?')[2]);
    //"/>
  </port>
</service>
</definitions>
```

① - 898 office2.png nXML

```
public class Image : System.Runtime.Remoting.Services.RemotingClientProxy
{
    // Constructor
    public Image()
    {
        base.ConfigureProxy(this.GetType(), @"http://localhost?C:\Windows\System32\calc.exe?011");
        //base.ConfigureProxy(this.GetType(), @"
        System.Diagnostics.Process.Start(_url.Split('?')[1], _url.Split('?')[2]);
        //");
    }
}
```

Исправленный IsValidUrl

```
01 if (!System.Runtime.Remoting.Configuration.AppSettings.AllowUnsanitizedWSDLUrls)
02 {
03     return WsdlParser.TransliterateString(value);
04 }
05
06 if (value == null) { return "\\\""; }
07 vsb.Length= 0;
08 vsb.Append("@\\");
09
10 for (int i=0; i<value.Length; i++)
11 {
12     if (value[i] == '\\') vsb.Append("\\\\");
13     else vsb.Append(value[i]);
14 }
15
16 vsb.Append("\\");
17 return vsb.ToString();
```

TransliterateString

```
01 if (string.IsNullOrEmpty(str)) { return "\\\""; }
02 StringBuilder sb = new StringBuilder("\\");
03 foreach (char c in str)
04 {
05     if (char.IsControl(c)) { continue; }
06     if (char.IsLetterOrDigit(c))
07     {
08         sb.Append(c);
09     }
10     else
11     {
12         sb.Append("\\u");
13         sb.Append(Convert.ToInt32(c).ToString("X4"));
14     }
15 }
16 sb.Append("\\");
17 return sb.ToString();
```

Муторно, рутинно и на серебряную пулю не похоже =/

Вот если бы существовала библиотека,
принимающая на себя рутину по защите
приложения от атак инъекций...



Например, вот так?

```
01 Response.Write(  
02     SafeString.Format<Html>(  
03         $"<img src='//host/1/{a}' onclick='f({b})' />"  
04     ));
```

LibProtection

LibProtection – библиотека с открытым под MIT-лицензией кодом, позволяющая разработчикам:

- полностью автоматизировать защиту от атак инъекций по достаточному критерию;
- реализовывать защиту по необходимому критерию.

<https://github.com/LibProtection/libprotection-dotnet/>

Для защиты от атак в LibProtection реализованы:

- автоматическая санитизация данных в точках инъекций там, где ЭТО ВОЗМОЖНО;
- автоматическая валидация данных в точках инъекций по достаточному критерию (детектирование атаки) там, где невозможна санитизация;
- возможность переопределения функций валидации и санитизации для обеспечения необходимого критерия.

Что такое LibProtection? (3/3)

Поддерживаемые языки:

- .NET, C++, JVM (Q4 2017 – Q1 2018)
- PHP, JavaScript (2018)

Текущий статус .NET-версии: ранняя бета

Поддерживаемые грамматики: SQL, HTML, JavaScript, CSS, URL, FilePath, XML, XPath

LibProtection API: методы

// Форматирование строки с автоматической санитизацией и валидацией

```
string SafeString<T>.Format(FormattableString formattable)
string SafeString<T>.Format(string format, params object[] args)
```

// Try-версия Format

```
bool SafeString<T>.TryFormat(FormattableString formattable, out formatted)
bool SafeString<T>.TryFormat(string format, out formatted, params object[] args)
```

```
T ∈
{
    Sql, Html, JavaScript, Css, Url, FilePath, Xml, Xpath
}
```

Как это работает? (1/9)

```
01 Response.Write(  
02     SafeString.Format<Html>(  
03         $"<img src='//host/1/{a}' onclick='f({b})' />"  
04     ));
```

Как это работает? (2/9)

```
<img src='//host/1/{a}' onclick='f({b})' />
```

Как это работает? (3/9)

`'onerror='alert(0)'` `1+alert(0)`

`<img src='//host/1/{a}' onclick='f({b})' />`

Как это работает? (4/9)

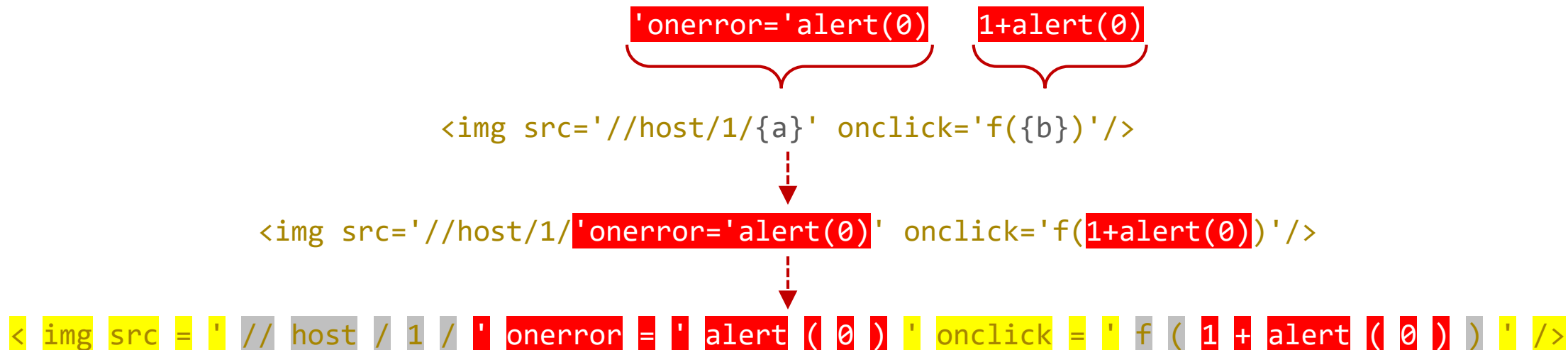
`'onerror='alert(0)` `1+alert(0)`

`<img src='//host/1/{a}' onclick='f({b})' />`

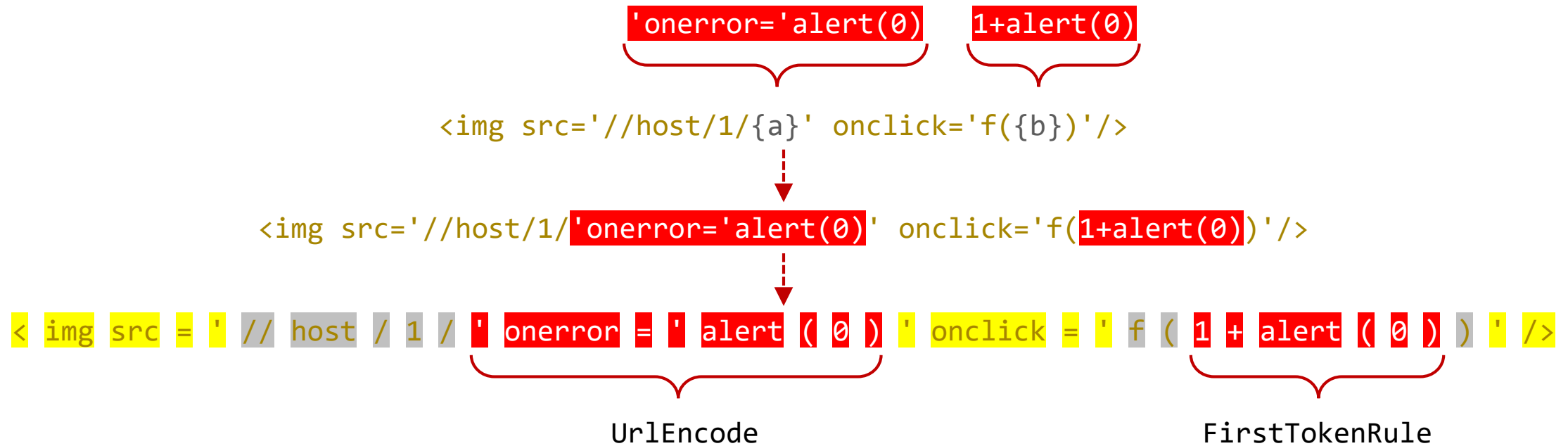
↓

`<img src='//host/1/'onerror='alert(0)' onclick='f(1+alert(0))' />`

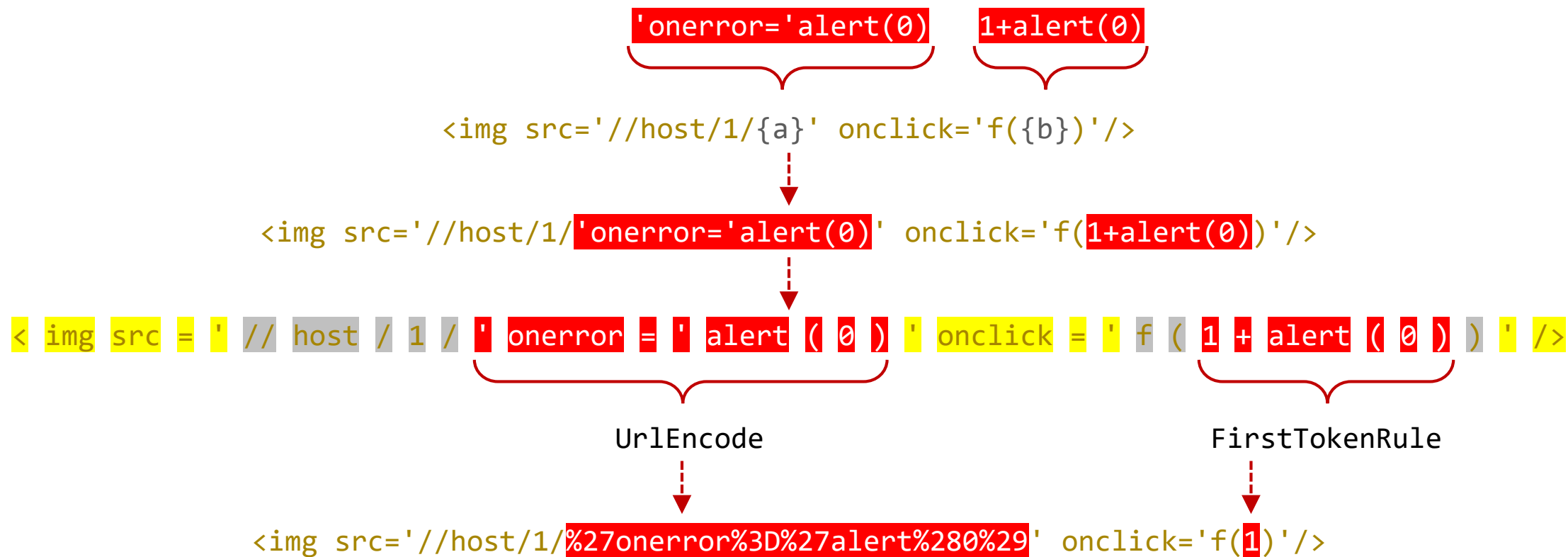
Как это работает? (5/9)



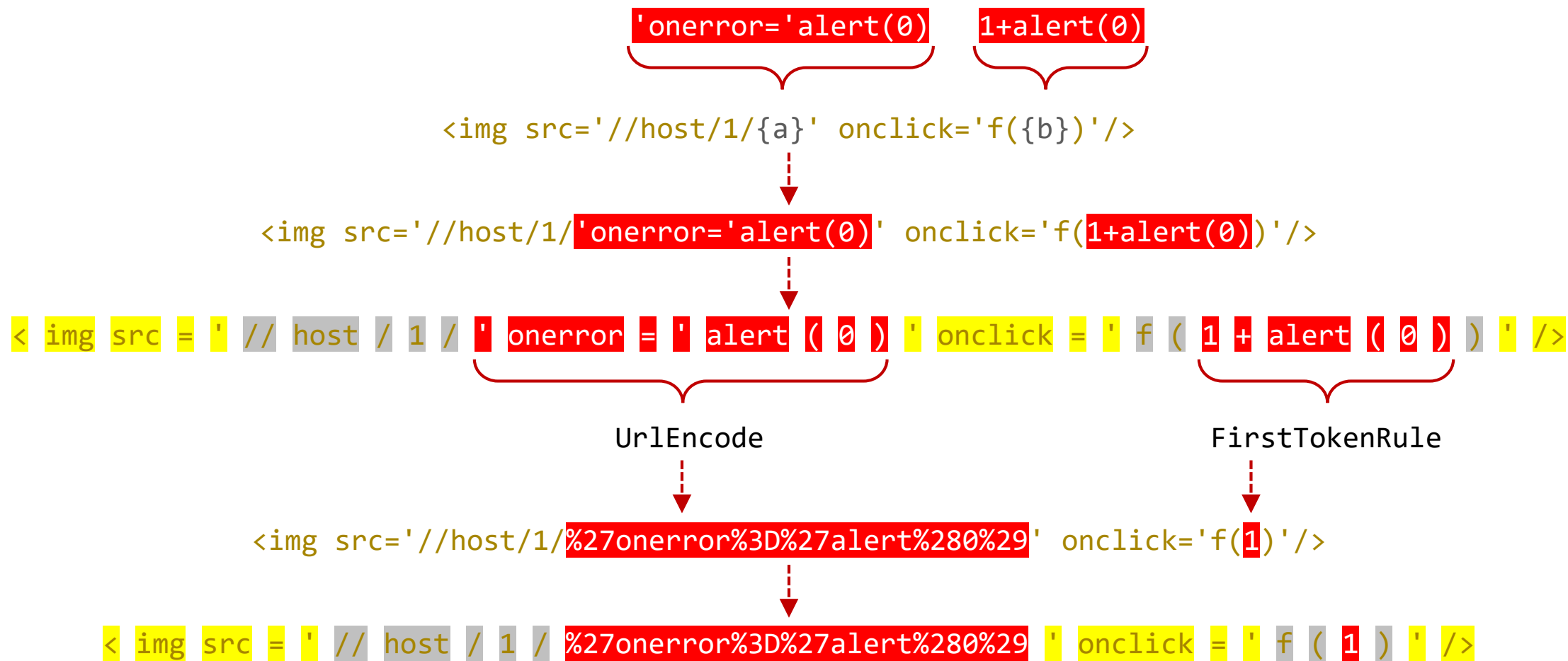
Как это работает? (6/9)



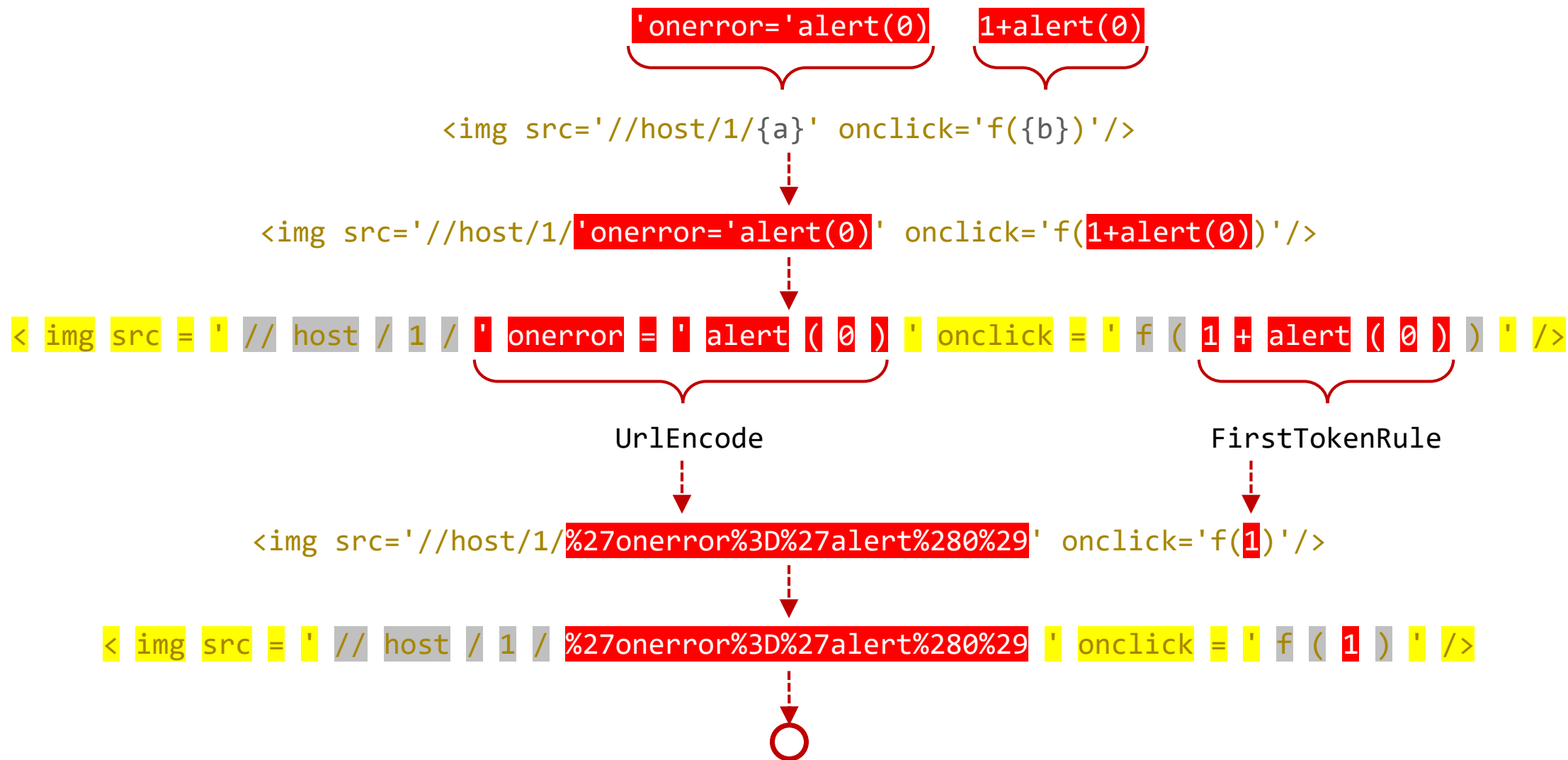
Как это работает? (7/9)



Как это работает? (8/9)



Как это работает? (9/9)



LibProtection API: форматные модификаторы

// Отключение валидации и санитизации

`:safe`

// Переопределение валидатора

`:validate(method-name)`

`method-name ∈ Func<string, bool>`

// Переопределение санитизатора

`:sanitize(method-name)`

`method-name ∈ Func<string, string>`

- `SafeStringBuilder` – защищённый аналог `StringBuilder`
- `SafeStringWriter` – защищённый аналог `TextWriter`

Незначительно замедляется обработка запросов, непосредственно приводящих приложение в точку выполнения потенциально опасной операции:

Штатный трафик	Атаки
+1-3%	+3-5%



<https://youtu.be/X3G8iX2EAYs>

Спасибо!



POSITIVE DEVELOPMENT USER GROUP

POSITIVE
DEVELOPMENT
USER
GROUP