

Table of Contents

Table of Contents	1
UserGate Policy Language (UPL)	2
UserGate Policy Language (Описание)	3
Общие положения	4
Условия	6
Встроенные библиотеки	11
Определения	12
Свойства	13
block_invalid_cert Блокирование сайтов с некорректными сертификатами. Действительно только для правил инспектирования SSL. Синтаксис: InvalidCertificate ::= block_invalid_cert '(' boolean ')' boolean ::= yes no true false	
check_revoc_cert Проверка по списку отозванных сертификатов. Действительно только для правил инспектирования SSL. Синтаксис: ChekRevocation ::= check_revoc_cert '(' boolean ')' boolean ::= yes no true false	
block_expired_cert Блокировка сертификатов с истекшим сроком действия. Действительно только для правил инспектирования SSL. Синтаксис: ExpiredCertificate ::= block_expired_cert '(' boolean ')' boolean ::= yes no true false	
block_self_signed_cert Блокировка самоподписанных сертификатов. Действительно только для правил инспектирования SSL. Синтаксис: SelfSignedCertificate ::= block_self_signed_cert '(' boolean ')' boolean ::= yes no true false	
ssl_profile Профиль SSL. Действительно только для правил инспектирования SSL, reverse-прокси, веб-портала. Синтаксис: SslProfile ::= ssl_profile '(' string word ')' Свойства правил публикации	
is_https Включение поддержки HTTPS. Действительно только для правил публикации. Синтаксис: IsHttps ::= is_https '(' boolean ')' boolean ::= yes no true false	
rewrite_path Подмена путей. Действительно только для правил публикации. Синтаксис: RewritePath ::= rewrite_path '(' path_from, path_to ')' path_from — изменить с (домен и/или путь URL, которые требуется изменить); path_to — изменить на (домен и/или путь URL, на которые требуется заменить старые).	
waf_profile WAF-профиль. Действительно только для правил публикации. Синтаксис: WafProfile ::= waf_profile '(' string word list ')' 14	
Действия 16	
Типы правил 21	
Список поддерживаемых HTTP-заголовков 23	

Version: 7.7.x

Product: WAF

UPL (UserGate Policy Language) — язык описания политик UserGate. Термин «политика» употребляется здесь в контексте конфигурации правил, применяемых для принятия решений по требованиям аутентификации, правам доступа или преобразования контента.

Правила настраиваются с использованием действий, условий и свойств.

Для каждого правила настраивается одно из действий. **Действия** — настройки, которые управляют обработкой транзакции (OK, WARNING, PASS, DENY, FORCE_PASS, FORCE_DENY). При настройке правил, в которых не предусмотрено указание действия (например, правила DNS, NAT и маршрутизации, пропускной способности и т.п.), необходимо указать действия PASS или OK.

Условия задаются знаками равно (=) или не равно (!=), например, зоны, адреса, GeoIP источников и назначения, сервисы, приложения и т.д.; все условия в правиле проверяются по логическому И, т.е. правило сработает, если будут выполнены все условия.

Важно! Условие не должно в качестве триггера содержать другое условие. В противном случае, если правило с условием-триггером будет добавлено в профиль WAF, другие правила в этом профиле срабатывать не будут.

Свойства правил задаются в круглых скобках и используются для указания дополнительной информации, например, название правил, их описание, функция журналирования и т.д.

Примечание При настройке правил сначала указывается действие, потом условия и затем свойства.

UPL используется для создания WAF-правил в персональных слоях. Также с помощью UPL в интерфейсе CLI создаются правила политик сети для следующих разделов:

- Межсетевой экрана (уровень: **network-policy firewall**).
- Правила публикации (уровень: **global-portal reverse-proxy-rules**).

Структура команды для создания правила:

```
Admin@nodename# create <level> <position> upl-rule <str-upl-syntax>
```

где <level> — уровень, на котором необходимо создать правило.

<position> — позиция, на которую будет помещено правило.

<str-upl-syntax> строка, в которой описано правило в UPL синтаксисе.

Структура команды для обновления существующего правила:

```
Admin@nodename# set <level> <position> upl-rule <str-upl-syntax>
```

где <level> — уровень, на котором необходимо обновить правило.

<position> — номер правила, которое необходимо обновить.

<str-upl-syntax> строка, в которой описано правило в UPL синтаксисе.

Структура команды для удаления правила:

```
Admin@nodename# delete <level> <position | all>
```

где <level> — уровень, на котором необходимо удалить правило.

<position> — номер правила, которое необходимо удалить.

<all> — удалить все правила.

Структура команды для отображения правила:

```
Admin@nodename# show <level> <position | all>
```

где <level> — раздел, правила которого нужно отобразить.

<position> — номер правила, которое необходимо отобразить.

<all> — отобразить все правила.

Пример создания правила межсетевого экрана с использованием UPL (использован многострочный ввод):

```
Admin@nodename# create network-policy firewall 1 upl-rule \  
...DENY \  
...src.zone = Trusted \  
...dst.zone = Untrusted \  
...user = known \  
...time = lib.time("Working hours") \  
...rule_log(session) \  
...name("Example of firewall rule created in CLI") \  
...enabled(true)
```

После создания правило отобразится в начале списка правил межсетевого экрана (на позиции 1). Данное правило запрещает HTTPS-трафик из зоны Trusted в зону Untrusted пользователям, идентифицированным системой. Правило работает в соответствии с расписанием «Working hours». При срабатывании правила в журнал будет записана информация о начале сессии.

В последующих статьях данного раздела можно найти более подробную информацию об [общих положениях](#) языка UPL, [определениях](#), [встроенных библиотеках](#), [условиях](#), [действиях](#), [свойствах](#) и [типах правил](#).

Version: 7.7.x
Product: WAF

Комментарии

Любая строка, начинающаяся с символа " %", является комментарием. Символ процента "%" после пробела или табуляции вводит комментарий, который продолжается до конца строки (кроме случаев, когда символ процента отображается внутри кавычек (""), как часть выражения).

Пример:

% Это комментарий
DENY("Too many Host headers") request.header.Host.count = 2.. % и это тоже

Комментарии могут быть в любом месте файла с описанием политик.

Правила

Правило политики (rule) состоит из условий и некоторого количества действий, записанных в любом порядке. Есть также свойства (properties), которые синтаксически выглядят как действие, но при этом активных действий не производят. Например, свойство *late* просто добавляет атрибут "имя" в правило.

Правила обычно пишутся в одной строке, но могут быть разбиты на строки с помощью специального символа — обратного слеша " \".

Когда правило выполняется, условие проверяется для текущей конкретной транзакции. Если условие оценивается как *True* (истина), выполняются все перечисленные действия и текущий слой заканчивается при наличии префиксов *PASS / FORCE_PASS / DENY / FORCE_DENY / WARNING / OK* . Если сработавшее правило не имеет префиксов *PASS / FORCE_PASS / DENY / FORCE_DENY / WARNING / OK* , то выполняются действия и дальше обрабатывается уже следующее правило. Если условие оценивается как *False* для этой транзакции, то дальше обрабатывается уже следующее правило.

Все условия в правиле проверяются по логическому " И". Другими словами, правило сработает, когда будут выполнены все условия.

В свою очередь, условие является логической комбинацией триггеров. Триггеры — это отдельные тесты, которые можно выполнить с компонентами запроса, ответа, связанными пользователями или состоянием системы.

Действия — это настройки, которые управляют обработкой транзакции. Например, запретить (*deny*) или обработать объект (изменить заголовок — *rewrite*).

Синтаксис:

Rule ::= (PASS | FORCE_PASS | DENY| (DENY '(' string ')') | FORCE_DENY | FORCE_DENY '(' string ') | WARNING | OK)? Conditions '?' Actions

Conditions ::= condition '?' Conditions

Actions ::= action '?' Actions

Пример:

Запрос будет запрещен, когда сработают оба триггера:

- домен будет example.com
- время будет между 9 и 17 часами

DENY url.domain = "example.com" time=09:00..17:00

Слои

Слой (layer)— это конструкция UPL, используемая для группировки правил и принятия одного решения. Раздельные принятия решения помогают контролировать сложность политики. Это делается путем написания каждого решения в отдельном слое.

У любого правила в слое может быть префикс *PASS / FORCE_PASS / DENY / FORCE_DENY / OK / WARNING*, когда срабатывает правило с таким префиксом, все остальные правила в слое пропускаются.

В случае если сработало правило с префиксом *FORCE_PASS* или *FORCE_DENY*, то это является окончательным результатом обработки, в противном случае обработка переходит на следующий слой. После обработки всех слоев запрос будет заблокирован или пропущен в зависимости от того, что было последним — *PASS / FORCE_PASS* или *DENY / FORCE_DENY*. Если процессинг остановится на *WARNING*, будет добавлено предупреждение в тело ответа.

Префикс *OK* подразумевает остановку обработки правил в текущем слое при выполнении условий и действий (если таковые указаны). Если префикс отсутствует при выполнении условий и действий, то остановка не подразумевается.

Действия *FORCE_PASS* и *FORCE_DENY* похожи на *PASS* и *DENY*, за исключением того, что они могут быть переопределены на последующих слоях. *FORCE_DENY* и *FORCE_PASS* немедленно прекращают проверку правил как на текущем, так и на последующих слоях, и этот результат является окончательным.

Синтаксис:

Layer ::= '[' layer_type layer_name ']'

layer_type ::= firewall | reverseproxy | reverseproxy_balancing

layer_name ::= string

atom ::= [a-z][0-9a-zA-Z-]+

string ::= "" произвольная строка ""

Пример 1:

[content "L1"]
DENY enabled(true) % по умолчанию все запрещено

```
[content "Devs"]
DENY group != Developers enabled(true)
%... дальше идут правила, которые будут применяться только для группы Developers
```

Пример 2:

```
[content "Admin"]
FORCE_PASS group = Admins enabled(true)
```

```
[content "L2"]
DENY enabled(true) % по умолчанию все запрещено
```

Динамические значения

Значения "адрес запроса" (*url, url.host, url.path*), "IP-адрес источника/назначения" (*src.ip, dst.ip*), "значения заголовков" (*request* и *response*) и "параметры запроса" (*qparam*) могут сравниваться между собой, а также использоваться в качестве аргумента в действиях (*actions*), где это предусмотрено.

Условия

Version: 7.7.x

Product: WAF

Условие (condition) в языке UPL является логической комбинацией триггеров. Триггеры — это отдельные тесты, которые можно выполнить с компонентами запроса, ответа, связанными пользователями или состоянием системы. Все триггеры условия сравниваются со значениями с помощью операторов "=" и "!=". В роли значения могут выступать константные значения, такие как строки, целочисленные значения, диапазоны значений, динамические значения.

Синтаксис:

condition ::= condition_name ('=' | '!=') condition_value

condition_value ::= pattern | list

list ::= '(' ((pattern ',') pattern)? ')'*

pattern ::= word | string | integer | float | boolean | range | condition_name

string ::= "" произвольная строка ""

*word ::= [a-zA-Z][0-9a-zA-Z_\-]**

boolean ::= yes|no|true|false

range ::= integer .. [integer] | [integer] .. integer | float .. [float] | [float] .. float

numeric ::= integer | range

http.method

Проверка используемого HTTP-метода. Метод можно указывать как в кавычках, так и без.

Синтаксис:

http.method = GET | CONNECT | DELETE | HEAD | POST | PUT | TRACE | OPTIONS | TUNNEL | LINK | UNLINK | PATCH | PROPFIND | PROPPATCH | MKCOL | COPY | MOVE | LOCK | UNLOCK | MKDIR | INDEX | RMDIR | COPY | MOVE

http.request.version

Проверка версии HTTP-запроса.

Синтаксис:

http.request.version = 0.9 | 1.0 | 1.1

http.response.version

Проверка версии HTTP-ответа.

Синтаксис:

http.response.version = 0.9 | 1.0 | 1.1

http.response.code

Проверка HTTP-кода ответа. Валидные значения: 100 - 999.

Синтаксис:

http.response.code = NNN *%(где NNN число от 100 до 999)*

http.request.body, http.request.body.nocase, http.response.body и http.response.body.nocase

Проверка тела запроса/ответа HTTP на содержание определенной сигнатуры.

Пример:

DENY http.response.body.nocase = "<title>index of" http.response.body = ">"

http.request.body.re2, http.response.body.re2

Проверка тела HTTP-запроса или HTTP-ответа с использованием регулярных выражений.

Синтаксис:

request.body.re2 = string

response.body.re2 = string

Пример:

http.request.body.re2 = "example|test"

request.header.<h_name> и response.header.<h_name>

Проверка HTTP-заголовка запроса/ответа. *h_name* может принимать одно из поддерживаемых значений. Подробнее о поддерживаемых HTTP-заголовках — в

разделе «[Список поддерживаемых HTTP-заголовков](#)».

Синтаксис:

```
request.header.<h_name>[.base64][.nocase] = string
```

Пример:

```
DENY url="http://usergate.com" request.header.Pragma="no-cache"
```

```
PASS request.header.User-Agent = lib.useragent("Browsers")
```

```
PASS request.header.Content-Type = lib.mime("Applications")
```

```
DENY request.header.Connection.substring = "Upgrade"
```

request.header.<h_name>.substring и response.header.<h_name>.substring

Проверка HTTP-заголовка запроса/ответа на вхождение подстроки. *h_name* может принимать одно из поддерживаемых значений. Подробнее о поддерживаемых HTTP-заголовках — в разделе «[Список поддерживаемых HTTP-заголовков](#)».

Синтаксис:

```
request.header.<h_name>[.base64]substring[.nocase] = string
```

Пример:

```
DENY request.header.User-Agent.substring = "curl/"
```

request.header.<h_name>.regex и response.header.<h_name>.regex

Проверка HTTP-заголовка запроса/ответа на регулярное выражение PCRE. *h_name* может принимать одно из поддерживаемых значений. Подробнее о поддерживаемых HTTP-заголовках — в разделе «[Список поддерживаемых HTTP-заголовков](#)».

Синтаксис:

```
request.header.<h_name>[.base64].regex = string
```

Пример:

```
DENY("Accept only digits in content length") request.header.Content-Length.regex != "[0-9]*"
```

request.header.<h_name>.re2 и response.header.<h_name>.re2

Проверка HTTP-заголовка запроса/ответа на регулярное выражение RE2. *h_name* может принимать одно из поддерживаемых значений. Подробнее о поддерживаемых HTTP-заголовках — в разделе «[Список поддерживаемых HTTP-заголовков](#)».

Синтаксис:

```
request.header.<h_name>[.base64].re2 = string
```

Пример:

```
DENY("Accept only digits in content length") request.header.Content-Length.re3 != "[0-9]*"
```

request.header.<h_name>.count и response.header.<h_name>.count

Проверка количества заголовков *<h_name>* в HTTP-запросе/ответе. *h_name* может принимать одно из поддерживаемых значений. Подробнее о поддерживаемых HTTP-заголовках — в разделе «[Список поддерживаемых HTTP-заголовков](#)».

Синтаксис:

```
request.header.<h_name>.count = integer | range
```

Пример:

```
DENY("Too many Host headers") request.header.Host.count = 2..
```

request.header.<h_name>.length и response.header.<h_name>.length

Проверка длины значений всех заголовков *<h_name>* в HTTP-запросе/ответе. *h_name* может принимать одно из поддерживаемых значений. Подробнее о поддерживаемых HTTP-заголовках — в разделе «[Список поддерживаемых HTTP-заголовков](#)».

Синтаксис:

```
request.header.<h_name>.length = integer | range
```

Пример:

```
DENY("Too much Cookie data") request.header.Cookie.length = 2048..
```

request.header_names, request.header_values, response.header_names и response.header_values

Проверка имени/значения всех HTTP-заголовков запроса/ответа на значение.

Синтаксис:

```
request.header_values[.base64].regex = string
```

```
request.header_values[.base64].re2 = string
```

```
request.header_values[.base64].substring[.nocase] = string
```

request.header_values.count = integer | range

request.header_values.length = integer | range

request.x_header.<xh_name> и response.x_header.<xh_name>

Проверка HTTP-заголовка запроса/ответа на значение. *xh_name* — произвольный HTTP-заголовок.

Синтаксис:

request.x_header.<xh_name>[.base64][.nocase] = string

request.x_header.<xh_name>[.base64].regex = string

request.x_header.<xh_name>[.base64].re2 = string

request.x_header.<xh_name>[.base64].substring[.nocase] = string

request.x_header.<xh_name>.count = integer | range

request.x_header.<xh_name>.length = integer | range

Пример:

DENY url="http://usergate.com" request.x_header.Test="test1"

Возможны также суффиксы **length**, **count**, **regex**, **re2** как и в случае с *h_name*.

DENY("Too much X-Test data") request.x_header.X-Test.length = 2048..

DENY("Too much X-Test2 headers data") request.x_header.X-Test2.count = 2..

PASS request.x_header.Test.regex = "[0-9]"

request.header.Cookie.<cookie_name>

Проверка заголовка запроса Cookie на значение.

Синтаксис:

request.header.Cookie.<cookie_name>[.base64][.(nocase | substring | substring.nocase | regex | re2)] = string

Пример:

DENY http.method = POST request.header.Cookie.csrf_token != qparam.CSRF_TOKEN enabled(true) name("Check CSRF")

time, day, hour, minute

Проверка соответствия текущего времени заданному условию. Если не указан суффикс *utc*, время берется локальное, иначе — по Гринвичу.

Синтаксис:

day[.utc] = monday | tuesday | wednesday | thursday | friday | saturday | sunday | DD | list

time[.utc] = HH:MM | range | lib.time(<name>)

hour[.utc] = HH | range

minute[.utc] = MM | range

HH ::= 00 - 23

MM ::= 00 - 59

DD ::= 1 - 31

Пример:

PASS time = 12:00..13:00 % разрешить каждый день с 12 до 13 часов

PASS time = lib.time("Праздники") % использовать библиотеку "Праздники"

DENY day = (sunday, saturday) % запретить на выходных

DENY day = (monday, 15) hour = 9..18 % запретить каждый понедельник и каждое 15 число месяца с 9 до 18 часов

Открытые интервалы учитываются по границе суток/часа.

PASS hour = 18.. % означает, что разрешено с 18 часов до полуночи

minute = ..10 % первые 10 минут каждого часа

url, url.host и url.address

Проверка url или его части на значение. Проверка использует нормализованный URI с декодированными * %*.

Синтаксис:

url[.(prefix | substring | substring.nocase | suffix | regex | re2)] = string

url.host[.(prefix | substring | suffix | regex | re2)] = string

url.domain[.(prefix | substring | suffix | regex | re2)] = string

url.address = ip_address | subnet | subnet_label

url.port = [low_port]..[high_port] | port

url.path.base64[(prefix | substring | substring.nocase | suffix | regex | re2)] = string

url.is_absolute = yes | no % полный или нет URL

prefix ::= string % начало строки

substring ::= string % подстрока

suffix ::= string % окончание строки

regex ::= string % регулярное выражение PCRE

re2 ::= string % регулярное выражение RE2

url.address — это, по сути, синоним *dst.ip*.

Пример:

DENY url.path.base64.re2 = "(?i)bondisconnecting\W*" enabled(true) name("ondisconnecting (URI)")

qparam.<name>, qparam.values и qparam.names

Проверка значения параметров запроса. Проверка использует имена и значения параметров с декодированными * %*.

Синтаксис:

qparam.length = numeric % проверить общую длину query-параметров

qparam.count = numeric % проверить количество query-параметров

qparam.<name>[(length | count)] = numeric % Проверить длину/количество query-параметров <name>

qparam.<name>[.base64][.(nocase | substring | substring.nocase | regex | re2)] = string % проверить имя <name> на вхождение подстроки/регулярные выражения

qparam.values[.base64].substring[.nocase] = string % проверить все значения на вхождение подстроки

qparam.names[.base64].substring[.nocase] = string % проверить все имена на вхождение подстроки

qparam.values[.base64].regex = string % проверить все значения на регулярное выражение

qparam.names[.base64].regex = string % проверить все имена на регулярные выражения

qparam.values[.base64].re2 = string % проверить все значения на регулярное выражение

qparam.names[.base64].re2 = string % проверить все имена на регулярные выражения

numeric ::= integer | range % число либо диапазон

qparam.values.length = numeric % проверить длину всех значений

qparam.names.length = numeric % проверить длину всех имен

regex ::= string % регулярное выражение PCRE

re2 ::= string % регулярное выражение RE2

Пример:

DENY("limit arguments total length") qparam.length = 1024.. % total
DENY("Limit argument value length") qparam.values.length = 1024.. % for each
DENY("Limit argument name length") qparam.names.length = 1024.. % for each
DENY("Maximum number of arguments in request limited") qparam.count = 12.. % total
DENY("PHP injection attempt") qparam.values.base64.substring.nocase = "\${@print"

src и dst

Проверка условия на IP-адрес, зону или GeoIP источника/назначения.

Синтаксис:

src.ip = ip_address | subnet | subnet_label | list | lib

dst.ip = ip_address | subnet | subnet_label | list | lib

src.zone = integer | zone_name

dst.zone = integer | zone_name

src.geoip = iso3166 | list

dst.geoip = iso3166 | list

src.mac = mac_address | list

dst.mac = mac_address | list

lib ::= lib.(network | url) '(' list_libs ')'

list_libs ::= lib_name ',' list_libs

lib_name ::= word | string

iso3166 ::= [A-Z][A-Z]

url.address — это, по сути, синоним *dst.ip*.

response_time

Проверка времени ответа в миллисекундах.

Синтаксис:

response_time = integer

Встроенные библиотеки

Version: 7.7.x

Product: WAF

Библиотеки (*lib*) — это элементы языка UPL, которые служат для доступа к встроенным и пользовательским библиотекам. Как правило, это достаточно большие списки, которые неудобно описывать через определения `def`. Обращение к библиотекам происходит по их именам.

Синтаксис:

```
library ::= lib.<url | useragent | network | time>(list_names)
```

```
list_names ::= name list_names
```

```
name ::= word | string
```

url — список URL;

useragent — список юзерагентов;

network — список сетей/IP-адресов;

time — библиотека с промежутками времени.

Пример:

```
DENY src.ip = lib.network("Bad ips", "Test ips")
```

```
DENY dst.ip = lib.network("Bad ips")
```

```
DENY dst.ip = lib.url("Bad urls") % в данном случае домены будут резолвиться в ip-адреса
```

```
PASS request.header.User-Agent = lib.useragent("Browsers")
```

```
DENY time = lib.time(Weekends)
```

Определения

Version: 7.7.x

Product: WAF

В файлах политик определения (def) служат для объединения наборов условий или действий. Каждое определение должно иметь уникальное пользовательское имя, по которому к нему можно обратиться из правил.

def condition

Наборы условий. Все условия в одной строке проверяются по логическому *И*. Перевод строки означает логическое *ИЛИ*. Символ экранирования — обратный слэш ("**") в конце строки позволяет перенести условие по *И* на следующую строку.

Синтаксис:

```
def condition label_name

    conditions

end

conditions ::= condition '?' [conditions]

condition ::= name '=' value

label_name ::= atom

atom ::= [a-z][0-9a-zA-Z_]+
```

def var

Определение переменных. Служит для подсчета некоторых событий за определенный интервал времени. Для изменения значения предназначены действия *inc* и *dec*.

Синтаксис:

```
def var label_name

    init ::= integer

    window ::= time

    key ::= condition_name | condition_list

end

label_name ::= atom

atom ::= [a-z][0-9a-zA-Z_]+

condition_list ::= '(' condition_name , condition_list ')'
```

init — это начальное значение переменной, к которому она вернется по истечении времени *window*;

key — поле или список полей, по которым группируются значения переменной (необязательный параметр).

Свойства

Version: 7.7.x

Product: WAF

Свойства (properties) — это некие атрибуты правила, например, *name* или *enabled*. Они используются для предоставления дополнительной информации в процессе обработки правил. Синтаксис свойств точно такой же, как у действий.

Синтаксис:

```
property = prop_name | prop_name '(' list_params ')'
```

```
prop_name ::= name | desc | id | rule_log | enabled
```

```
list_params ::= value ',' list_params
```

name и desc

Атрибуты *имя* и *описание* для правила.

Синтаксис:

```
Name ::= name '(' string|word ')'
```

```
Description ::= desc '(' string ')'
```

Пример:

```
DENY hour = 9..18 category = News name("Запретить News") desc("Запретить категорию News в рабочее время")
```

enabled

Атрибут, который включает или выключает работу правила.

Синтаксис:

```
Enable ::= enabled '(' boolean ')'
```

```
boolean ::= yes | no | true | false      % (по умолчанию false)
```

rule_log

Устанавливает атрибут журналирования правила. Значение *session* действительно только для правил межсетевого экрана, защиты от dos-атак и пропускной способности.

Синтаксис:

```
Logging ::= rule_log '(' boolean | session ')'
```

```
LoggingFwRule ::= rule_log '(' boolean , interval, burst)'
```

```
boolean ::= yes | no | true | false      % (по умолчанию no)
```

```
interval ::= "integer/[s,m,h,d]"
```

```
burst ::= integer
```

interval — среднее число пакетов, попадающих под условие *limit* в единицу времени (1/s, 1/m, 1/h, 1/d) , default = 3/h;

burst — максимальное число пакетов, попадающих в под условие *limit* за один раз (default = 5).

profile

Устанавливает профиль правила.

Синтаксис:

```
Profile ::= profile '(' string | word | list ')'
```

certificate

Сертификат, используемый для поддержки HTTPS-соединения. Действительно только для правил reverse-прокси.

Синтаксис:

```
CertAuthEnabled ::= cert_auth_enabled '(' boolean ')'
```

```
Certificate ::= certificate '(' certificate_name ')'
```

```
certificate_name ::= string | word
```

gateway

Шлюз. Имя одного из существующих шлюзов. Действительно только для правил NAT и маршрутизации, и для условий сценария "Проверка состояния".

Синтаксис:

```
Gateway ::= gateway '(' string | word ')'
```

Свойства правил межсетевого экрана

reject_with

Устанавливает способ, с помощью которого будет блокироваться трафик. Действительно только для правил межсетевого экрана.

Синтаксис:

Reject ::= reject_with '(' "tcp-reset-both" | "tcp-rst" | "host-unreach")'

fragmented

Проверка на фрагментированность пакетов. Действительно только для правил межсетевого экрана.

Синтаксис:

Fragmented ::= fragmented '(' boolean ')'

boolean ::= yes | no | true | false

yes — проверяются только фрагментированные пакеты;

no — проверяются только нефрагментированные пакеты;

—, если свойство *fragmented* не указано, то будут проверяться все пакеты.

block_invalid_cert

Блокирование сайтов с некорректными сертификатами. Действительно только для правил инспектирования SSL.

Синтаксис:

InvalidCertificate ::= block_invalid_cert '(' boolean ')'

boolean ::= yes | no | true | false

check_revoc_cert

Проверка по списку отозванных сертификатов. Действительно только для правил инспектирования SSL.

Синтаксис:

ChekRevocation ::= check_revoc_cert '(' boolean ')'

boolean ::= yes | no | true | false

block_expired_cert

Блокировка сертификатов с истекшим сроком действия. Действительно только для правил инспектирования SSL.

Синтаксис:

ExpiredCertificate ::= block_expired_cert '(' boolean ')'

boolean ::= yes | no | true | false

block_self_signed_cert

Блокировка самоподписанных сертификатов. Действительно только для правил инспектирования SSL.

Синтаксис:

SelfSignedCertificate ::= block_self_signed_cert '(' boolean ')'

boolean ::= yes | no | true | false

ssl_profile

Профиль SSL. Действительно только для правил инспектирования SSL, reverse-прокси, веб-портала.

Синтаксис:

SslProfile ::= ssl_profile '(' string | word ')'

Свойства правил публикации

is_https

Включение поддержки HTTPS. Действительно только для правил публикации.

Синтаксис:

IsHttps ::= is_https '(' boolean ')'

boolean ::= yes | no | true | false

rewrite_path

Подмена путей. Действительно только для правил публикации.

Синтаксис:

RewritePath ::= rewrite_path '(' path_from, path_to ')'

path_from — изменить с (домен и/или путь URL, которые требуется изменить);

path_to — изменить на (домен и/или путь URL, на которые требуется заменить старые).

waf_profile

WAF-профиль. Действительно только для правил публикации.

Синтаксис:

WafProfile ::= waf_profile '(' string | word | list ')'

Действия

Version: 7.7.x

Product: WAF

Действие (action) — это то, что будет выполнено, если условия в правиле истинны. В качестве параметров могут использоваться константные значения, или динамические значения там, где это предусмотрено.

Синтаксис:

```
action = action_name | action_name '(' list_params ')'
```

```
action_name ::= log_message | append | delete | set | replace | encrypt | inc/dec | reset | redirect | encrypt_body_url | decrypt_path | body_inject | set_cookie_token |  
body_replace | lookup_and_auth | encode_cookie | decode_cookie | sma | action_label
```

```
action_label ::= 'action'.<action_label_name>
```

```
action_label_name ::= atom
```

```
list_params ::= value ',' list_params
```

log_message

Записать сообщения в журнал.

Пример:

```
DENY category = lib.category(Productivity) log_message("Deny porno")
```

append

Добавить заголовок к HTTP-запросу/ответу. Подробнее о поддерживаемых заголовках — в разделе « [Список поддерживаемых HTTP-заголовков](#) ».

Первый параметр может быть опущен, если заголовок относится к одной группе *request* или *response*.

Синтаксис:

```
append([request | response,] <headername>, value)
```

headername — см. в разделе « [Список поддерживаемых HTTP-заголовков](#) ».

value ::= *string* | *numeric* | *condition_name*.

set

Переписать значение конкретному HTTP-заголовку. Подробнее о поддерживаемых заголовках — в разделе « [Список поддерживаемых HTTP-заголовков](#) ».

Первый параметр может быть опущен, если заголовок относится к одной группе *request* или *response*.

Синтаксис:

```
set([request | response,] <headername>, value)
```

headername — см. в разделе « [Список поддерживаемых HTTP-заголовков](#) ».

value ::= *string* | *numeric* | *condition_name*.

delete

Удалить HTTP-заголовок. Подробнее о поддерживаемых заголовках — в разделе « [Список поддерживаемых HTTP-заголовков](#) ».

Первый параметр может быть опущен, если заголовок относится к одной группе *request* или *response*.

Синтаксис:

```
delete([request | response,] <headername>)
```

headername — см. в разделе « [Список поддерживаемых HTTP-заголовков](#) ».

replace

Модифицировать значение HTTP-заголовка. Подробнее о поддерживаемых заголовках — в разделе « [Список поддерживаемых HTTP-заголовков](#) ».

Первый параметр может быть опущен, если заголовок относится к одной группе *request* или *response*.

Синтаксис:

```
replace([request | response,] <headername>, regex, value)
```

```
regex ::= string % регулярное выражение
```

```
value ::= string | condition_name
```

headername — см. в разделе « [Список поддерживаемых HTTP-заголовков](#) ».

Пример 1:

Добавить заголовок Referer:

```
PASS append(Referer, "http://example.com") enabled(true)
```

Удалить заголовок:

PASS delete(Referer)

Переписать заголовок:

PASS set(request, Cache-Control, no-cache)

Модифицировать заголовок Location:

PASS response.header.Location.count = 1.. replace(response, Location, "http://example.com", url.host) enabled(true)

Пример 2:

```
define action delete_referer
    log_message("Referer header deleted")
    delete(request, Referer)
end
```

encrypt

Шифровать часть пути в HTTP-заголовке. Подробнее о поддерживаемых заголовках — в разделе «[Список поддерживаемых HTTP-заголовков](#)».

Первый параметр может быть опущен, если заголовок относится к одной группе *request* или *response*.

Ключ шифрования и флаг "Использовать IP как часть ключа шифрования" — необязательные параметры.

Синтаксис:

encrypt([request | response,] <headername>, <url>[, <user_key>[, <add_ip>]])

url ::= string % часть url для фильтрации

user_key ::= string % пользовательский ключ шифрования (необязательный параметр)

add_ip ::= boolean % добавлять ли IP к ключу шифрования (логическое значение, необязательный параметр)

boolean ::= yes | no | true | false

headername — см. в разделе «[Список поддерживаемых HTTP-заголовков](#)».

encrypt_body_url

Шифровать часть пути в ссылках тела ответа.

Ключ шифрования и флаг "Использовать IP как часть ключа шифрования" — необязательные параметры.

Синтаксис:

encrypt_body_url(<url>[, <user_key>[, <add_ip>]])

url ::= string % часть url для фильтрации

user_key ::= string % пользовательский ключ шифрования (необязательный параметр)

add_ip ::= boolean % добавлять ли IP к ключу шифрования (логическое значение, необязательный параметр)

boolean ::= yes | no | true | false

decrypt_path

Дешифровать часть пути запроса.

Первый параметр может быть опущен, если заголовок относится к одной группе *request* или *response*.

Ключ шифрования и флаг "Использовать IP как часть ключа шифрования" — необязательные параметры.

Синтаксис:

decrypt_path(<path>[, <user_key>[, <add_ip>]])

path ::= string % часть пути для фильтрации

user_key ::= string % пользовательский ключ шифрования (необязательный параметр)

add_ip ::= boolean % добавлять ли IP к ключу шифрования (логическое значение, необязательный параметр)

boolean ::= yes | no | true | false

Пример:

Шифровать все относительные пути в заголовке *Location* и теле ответа, и дешифровать путь запроса:

```
decrypt_path("/", "User_Key", true) enabled(true) name("Path decode")
http.response.code = 302 encrypt(Location, "/", "User_Key", true) enabled(true) name("Encrypt Location header")
encrypt_body_url("/", "User_Key", true) enabled(true) name("Encrypt all relative URL")
```

body_inject

Вставить скрипт в тело ответа.

Синтаксис:

body_inject(inject_text)

inject_text ::= string

set_cookie_token

Добавить в ответ заголовок 'Set-Cookie' со сгенерированным токеном.

Синтаксис:

set_cookie_token(cookie_name, parameter, expires_date)

cookie_name ::= string

parameter ::= string

expires_date ::= [DD_]HH:MM % время которое будет прибавлено к текущему времени

Пример:

Реализация CSRF защиты:

```
DENY http.method = POST request.header.Referer.substring = "/login.php" qparam.UCSRF_TOKEN != request.header.Cookie.ucsrftoken enabled(true) name("Check CSRF")
url.path.prefix = "/login.php" set_cookie_token(ucsrftoken, "path=/", 01_00:00) body_inject("<script language=JavaScript>
var tokenName = 'UCSRF_TOKEN';
```

```
document.addEventListener('DOMContentLoaded', function()
{
var t_res = document.cookie.match(/ucsrftoken=(.+?)(;|$)/);
var tokenValue = t_res ? t_res[1] : "";

var forms = document.getElementsByTagName('form');
for(i=0; i<forms.length; i++)
{
var html = forms[i].innerHTML;
html += '<input type=hidden name=' + tokenName + ' value=' + tokenValue + ' />';
forms[i].innerHTML = html;
}
});
</script>") enabled(true) name("Inject")
```

encode_cookie

Шифровать значения Cookie в заголовке *Set-Cookie* с заданным именем.

Синтаксис:

encode_cookie(cookie_name[, condition_name][, user_kry_string][, f_encrypt])

cookie_name ::= string

condition_name % условие используемое для кодирования (по умолчанию src.ip)

user_kry_string ::= string % пользовательский ключ шифрования (по умолчанию "")

f_encrypt := true % необходимо шифрование (по умолчанию false)

decode_cookie

Дешифровать токен в заголовке Cookie с заданным именем.

Синтаксис:

decode_cookie(cookie_name[, condition_key][, user_kry_string][, f_decrypt])

cookie_name ::= string

condition_name % условие используемое для декодирования (по умолчанию src.ip)

user_kry_string ::= string % пользовательский ключ шифрования (по умолчанию "")

f_encrypt := true % необходимо шифрование (по умолчанию false)

Пример:

Шифрование и дешифрование Cookie с именем *security*:

```
response.header.Set-Cookie.count != 0 encode_cookie("security", src.ip, true) enabled(true) name("encode_cookie")
request.header.Cookie.count != 0 decode_cookie("security", src.ip, true) enabled(true) name("decode_cookie")
```

body_replace

Модифицировать тело ответа. Выполняется не более двух (первых) модификаций для каждого ответа.

Синтаксис:

body_replace(<regex>, <value>)

regex ::= string % регулярное выражение

value ::= string

Пример:

```
PASS \
body_replace("(\\+7|8)[\\s]?([\\d]{3})[\\s])?([\\d]{3})[\\s-]?([\\d]{2})[\\s-]?([\\d]{2})", "+\\1 (\\2) \\3-XX-XX" \\
body_replace("(\\w{1})[\\w\\.]*([\\w{1}])@([\\w]+)\\.([\\w]+)", "\\1***\\2@\\3.\\4") \\
```

```
enabled(true) \
name("Replace mail and phone")
```

lookup_and_auth

Аутентифицировать пользователя. В случае если IP не указан, запрос маркируется именем пользователя.

Синтаксис:

```
lookup_and_auth(<user_login>[, <ip_address>[, <session_timeout>]])
```

```
user_login ::= string | condition_name    % Логин аутентификации
```

```
ip_address ::= string | condition_name    % IP адрес
```

```
session_timeout ::= integer                % тайм-аут сессии, по умолчанию 0.
```

Пример:

```
lookup_and_auth(request.x_header.X-Authenticated-User, request.x_header.X-Forwarded-For, 300) enabled(true) name("User authentication")
lookup_and_auth(request.x_header.X-Authenticated-User) enabled(true) name("Mark request")
```

redirect

При блокировке перенаправить пользователя на адрес, который указан в редиректе.

Синтаксис:

```
Redirect ::= redirect(RespCode[, RedirectText], Url)
```

```
RespCode ::= 301 | 302 | 305 | 307
```

```
RedirectText ::= string
```

```
Url ::= string
```

Пример:

```
DENY src.zone = Trusted redirect(302, "Custom test (Moved)", "https://block.captive/block")
DENY src.zone = Untrusted redirect(302, "https://block.captive/block")
```

inc и dec

Используются для изменения значения переменных, объявленных как *def var*.

Синтаксис:

```
inc(var.<var_name>, integer)
```

```
dec(var.<var_name>, integer)
```

Пример:

На каждый *http.response.code = 500* увеличивается значение *rps* на 1. Если превысили 10 таких запросов за 5 минут, блокируем дальнейшие ответы. Через 5 минут переменная *rps* будет сброшена в 0:

```
def var rps
  init = 0
  window = 00:05
  key = src.ip
end
```

```
http.response.code = 500 var.rps=..10 inc(var.rps, 1) enabled(true)
```

```
PASS var.rps = 5 log_message("Warning!") enabled(true)
```

```
DENY var.rps=11.. log_message("Too many 500 errors!") enabled(true)
```

reset

Сбросить значения переменных, объявленных как *init* в *def var*, в начальное значение.

Синтаксис:

```
reset(var.<var_name>)
```

sma

Используются для подсчета среднего значения в окне времени, которое определяется в переменной как *window* в *def var*.

Синтаксис:

```
sma(var.<var_name>, integer)
```

Пример:

Блокируются запросы, когда среднее время запроса за 30-секундный интервал превысит 2 секунды:

```
def var avg_time
  init = 0
  window = 00:00:30
```

```
    key = src.ip  
end
```

```
src.zone = Untrusted sma(var.avg_time, response_time) enabled(true) name("sma")  
DENY src.zone = Untrusted var.avg_time = 2000.. enabled(true) name("sma res")
```

Product: WAF
Version: 7.7.x

Экспертные правила временной блокировки IP-адреса

Префиксы

Имя	Описание
DENY	Блокировка IP-адреса

Условия

[http.response.code](#)

Свойства

[name](#), [enabled](#).

Пример

В примере для конкретного источника (условие `src.ip`) настроен подсчет количества кодов ответа 404. Если количество таких ответов за последние 30 секунд превысит 10, IP-адрес этого источника будет заблокирован на одну минуту.

```
def var counter_404
  init = 0
  window = 00:00:30
  key = src.ip
end
def var block
  init = 0
  window = 00:01:00
  key = src.ip
end

DENY var.block = 1.. log_message("Black list") enabled(true) name("Black list")
http.response.code = 404 inc(var.counter_404, 1) log_message("Incriment counter") enabled(true) name("Incriment counter")
DENY var.counter_404 = 10.. inc(var.block, 1) log_message("Enable block") enabled(true) name("Enable block")
```

Сетевые правила, настраиваемые с помощью CLI

Правила межсетевого экрана

Префиксы

Имя	Описание
PASS	Разрешение трафика.
DENY	Блокировка трафика.

Условия

[src.zone](#), [src.geoip](#), [src.ip](#).
[dst.zone](#), [dst.geoip](#), [dst.ip](#).
[time](#), [url](#).

Свойства

[name](#), [desc](#), [enabled](#), [rule_log](#), [reject_with](#).

Пример

```
[firewall "Firewall rules"]
% ----- 1 -----
DENY \
  scenario = "Example torrent detection scenario" \
  dst.zone = Untrusted \
  dst.ip = lib.network("Botnets IP list") \
  rule_log(session) \
  reject_with("host-unreach") \
  enabled(true) \
  name("Example block RU RKN by IP list")
% ----- 2 -----
PASS \
```

```
scenario = "Example torrent detection scenario" \
src.zone = Trusted \
dst.zone = Untrusted \
rule_log(yes, "3/h", 5) \
enabled(true) \
name("Allow trusted to untrusted")
```

Правила публикации

Префиксы

Имя	Описание
OK	Всегда ОК.

Условия

[src.zone](#), [src.geoip](#), [src.ip](#), [src.mac](#).

[dst.zone](#), [dst.geoip](#), [dst.ip](#).

[request.header.User-Agent](#), [url.port](#).

Свойства

[name](#), [desc](#), [enabled](#), [rule_log](#), [profile](#), [certificate](#), [is_https](#), [ssl_profile](#),

[waf_profile](#), [rewrite_path](#).

Пример

```
[reverseproxy "Reverse proxy Rules"]
% ----- 1 -----
OK \
  url.port = 80 \
  src.zone = Untrusted \
  desc("Example reverse proxy rule. This is an example rule which can be changed or deleted if necessary. ") \
  profile("Example reverse proxy server") \
  rewrite_path("example.com/path1", "example.local/path1") \
  waf_profile("Example WAF profile") \
  enabled(true) \
  name("Example reverse proxy rule")
```

Правила балансировки

Префиксы

Имя	Описание
OK	Всегда ОК.

Свойства

[name](#), [desc](#), [enabled](#), [profile](#).

Пример

```
[reverseproxy_balancing "Reverse proxy load balancing Rules"]
% ----- 1 -----
OK \
  profile("Example reverse proxy server") \
  enabled(true) \
  name("Reverse-proxy load balancing")
```

Список поддерживаемых HTTP-заголовков

Version: 7.7.x
Product: WAF

HTTP Header	Request/Response	SET/REPLACE/ENCRYPT	APPEND	DELETE
Accept	Request	+	+	+
Accept-Charset	Request	+	+	+
Accept-Encoding	Request	+	+	+
Accept-Language	Request	+	+	+
Accept-Ranges	Response	+	+	+
Age	Response			
Allow	Request/Response	+	+	+
Authorization	Request			
Cache-Control	Request/Response	+	+	+
Client-IP	Request	+	+	
Connection	Request/Response		+	
Content-Encoding	Request/Response		+	
Content-Language	Request/Response		+	
Content-Length	Request/Response			
Content-Location	Request/Response		+	+
Content-Range	Request/Response			
Content-Type	Request/Response			
Cookie	Request	+	+	+
Date	Request/Response			
ETag	Response	+	+	
Expect	Request	+		
Expires	Request/Response		+	+
From	Request	+	+	
Host	Request			
If-Match	Request	+		
If-Modified-Since	Request			
If-None-Match	Request	+		
If-Range	Request			

HTTP Header	Request/Response	SET/REPLACE/ENCRYPT	APPEND	DELETE
If-Unmodified-Since	Request			
Last-Modified	Request/Response			
Location	Response	+	+	
Max-Forwards	Request			
Meter	Request/Response		+	+
Pragma	Request/Response		+	+
Proxy-Authenticate	Response	+		
Proxy-Authorization	Request	+		
Proxy-Connection	Request	+		
Range	Request	+	+	
Referer	Request	+	+	
Retry-After	Response	+	+	
Server	Response	+	+	
Set-Cookie	Response	+	+	+
TE	Request	+		
Trailer	Request/Response		+	
Transfer-Encoding	Request/Response		+	
Upgrade	Request/Response		+	
User-Agent	Request	+	+	
Vary	Response	+	+	+
Via	Request/Response	+	+	+
Warning	Request/Response	+	+	+
WWW-Authenticate	Response			

