

Table of Contents

Table of Contents	1
UserGate Application and Security Language (UASL)	2
UserGate Application and Security Language (UASL)	3
Метаинформация	4
Идентификатор	5
Фильтрация по IP-адресам	6
Фильтрация по портам	7
Сканирование пакетов без полезной нагрузки	8
Поиск шаблонов	9
Модификаторы области поиска	10
Частота срабатывания	13
Направление анализа	14
Поиск бинарных данных	15
Работа с метками	17
Протокольные анализаторы	18
Примеры	21

UserGate Application and Security Language (UASL)

Product: NGFW

Version: 7.x

Documentation: 

Общие положения

UserGate Application and Security Language (UASL) - язык, разработанный для возможности написания пользовательских сигнатур и приложений.

После создания пользовательские сигнатуры и приложения можно добавлять в профили COV и приложений для дальнейшего использования в правилах межсетевого экранирования.

Структура сигнатуры выглядит следующим образом:

UASL (.parameter1=<value1>; .parameter2=<value2>; ...)

Параметры сигнатуры записываются в круглых скобках; для разделения параметров используется точка с запятой (;).

Также допускается использование многострочного ввода:

```
UASL (.parameter1=<value1>;  
      .parameter2=<value2>;  
      ...  
      )
```

Примечание Максимальная длина пользовательской сигнатуры составляет 1024 байта.

Все условия одной сигнатуры без исключений будут объединены логическим оператором **И**.

Создание, редактирование или удаление сигнатур можно отслеживать по записям журнала событий.

Подробнее о семантике, конструкциях и правилах языка UASL читайте в следующих статьях данного раздела.

Метаинформация

Version: 7.x

Documentation: ☒

Product: NGFW

Поле **.rev** предназначено для указания дополнительной информации: пользователь, который создал сигнатуру, дата создания, тип и версия сигнатуры. Поле является опциональным и не влияет на работу COB; может быть использовано для отслеживания изменений.

Формат поля следующий:

.rev = <date>,<version>,<status>,<author>;

где параметры имеют следующий тип данных:

- <date>: целочисленный;
- <version>: целочисленный;
- <status>: строковый;
- <author>: строковый.

Идентификатор

Documentation: ☒

Product: NGFW

Version: 7.x

Для задания идентификатора сигнатуры или приложения используется поле .id:

.id=<id_value>

Указание данного поля является опциональным. Идентификатор также может быть задан в свойствах сигнатуры/приложения.

ПримечаниеЗначение, заданное в свойствах сигнатуры на вкладке *Общие*, имеет приоритет над значением, заданным с использованием UASL.

Идентификатор сигнатуры может принимать значения от 1000000 до 1049999 ; идентификатор приложения — от 1050000 до 1099999.

При задании идентификатора вручную его значение не является уникальным и может повторяться.

Если идентификатор не был задан администратором, то UserGate назначит его автоматически; при назначении в автоматическом режиме значения идентификатора не повторяются. При исчерпании пула идентификаторов будет отображена ошибка.

Фильтрация по IP-адресам

Documentation: ☒

Product: NGFW

Version: 7.x

Данные параметры позволяют настроить проверку заданных IP-адресов:

`.src_addr[!]=<IP_address/subnet>;`

`.dst_addr[!]=<IP_address/subnet>;`

Допустимы следующие форматы указания IP-адресов:

- A.B.C.D;
- A.B.C.D/E;
- A.B.C.D:E.

При задании IP-адреса указание маски подсети необязательно; чтобы задать несколько IP-адресов используйте квадратные скобки, например:

`.src_addr=[<IP_address1>, <IP_address2>;`

Примеры:

`.src_addr!=10.10.10.1;`

`.src_addr=10.10.10.0/24;`

`.src_addr=[10.10.10.0/24, 10.10.20.0/24, 10.10.30.0/24];`

Фильтрация по портам

Documentation: ☒
Product: NGFW
Version: 7.x

Для задания проверки TCP/UDP-портов используйте параметры:

- **.src_port** — для проверки портов источника:
- **.dst_port** — для проверки портов назначения:

Доступно использование следующих выражений:

Наименование	Описание
.src_port=<port_number>; .dst_port=<port_number>;	Указание конкретного порта источника и/или назначения.
.src_port!=<port_number>; .dst_port!=<port_number>;	Проверка всех портов кроме указанного.
.src_port=<port_number>; .dst_port=<port_number>;	Проверка всех портов, номер которых меньше или равен номеру указанного порта.
.src_port!=<port_number>; .dst_port!=<port_number>;	Проверка всех портов, номер которых больше указанного.
.src_port=<port_number>;; .dst_port=<port_number>;;	Проверка всех портов, номер которых больше или равен номеру указанного порта.
.src_port!=<port_number>;; .dst_port!=<port_number>;;	Проверка всех портов, номер которых меньше указанного.
.src_port=<port_number1>; <port_number2>; .dst_port=<port_number1>; <port_number2>;	Проверка портов, находящихся в указанном диапазоне (port_number1 и port_number2 включительно).
.src_port!=<port_number1>; <port_number2>; .dst_port!=<port_number1>; <port_number2>;	Проверка всех портов, не входящих в указанный диапазон (т.е. проверяются порты с номерами менее port_number1 и более port_number2).

Примеры:

.src_port=1000;; # '>= 1000'
.dst_port=5060;
.dst_port=1000;;
.dst_port=2000:8000; # '>= 2000 and <= 8000'
.dst_port!=2000:8000; # '< 2000 or > 8000'

Сканирование пакетов без полезной нагрузки

Documentation: ☒

Product: NGFW

Version: 7.x

Поле **.nopayload** предназначено для возможности сканирования пакетов, в которых отсутствует полезная нагрузка.

Примечание Поле неприменимо для сигнатур с поиском шаблонов и проверкой метки.

Например, данное поле может быть использовано для обнаружения сканирования портов пакетами без полезной нагрузки. Далее представлена сигнатура для обнаружения SYN-сканирования:

```
UASL(.protocol=tcp; .tcp.flags=S; .rate=1000,2; .track=src_ip; .nopayload;)
```

Примечание Указание поля *.nopayload* не исключает сканирования пакетов с нагрузкой.

Version: 7.x

Documentation: ☒

Product: NGFW

Следующий параметр позволяет задать шаблон, на наличие которого система обнаружения и предотвращения вторжений будет проверять содержимое пакетов (packet payload):

`.pattern[!]="string";`

ПримечаниеСледует помнить, что при поиске учитывается регистр символов.

Данные, представленные в шестнадцатеричной системе счисления (HEX), должны быть указаны с использованием символа вертикальной черты (|), например, |05 00 27|.

Для указания специальных символов используйте соответствия, представленные в таблице:

Символ	Запись в шестнадцатеричном формате
"	22 .
;	3B или 3b .
\	5C или 5c .
	7C или 7c .
:	3A или 3a .

Помимо оператора =, может быть использован и оператор !=. Тогда будет производится поиск пакетов, не содержащих заданный шаблон.

Общий формат задания параметра:

`.pattern[!]="string"; [.where=<MODE>;] [.no_case;;] [.distance=<RANGE>[,<MODE>;]] [.within=<RANGE>[,<MODE>;]] [.service=<MODE>;]`

Модификаторы области поиска (**.where**, **.no_case**, **.distance**, **.within**, **.service**) будут рассмотрены далее.

При написании сигнатуры доступно использование нескольких параметров **.pattern** с целью снижения уровня ложных срабатываний.

ПримечаниеИспользование параметра `.host` в кастомизированной пользовательской сигнатуре делает эту сигнатуру диссекторной. В этом случае параметр поиска шаблонов `.pattern` обязательно нужно использовать вместе с параметром области поиска [.where](#).

Модификаторы области поиска

Documentation: ☒
Product: NGFW
Version: 7.x

.no_case

Модификатор **no_case** позволяет осуществлять поиск, заданный параметром **.pattern**, без учета регистра символов.

Пример:

UASL(.id=XXX; .name="XXX"; .pattern="Host: "; .pattern="google.com");

.where

Модификатор **.where** используется для указания области поиска сигнатуры:

.where=<MODE>;

где <MODE> может принимать следующие значения:

Наименование	Описание
packet_origin	Областью поиска является весь пакет без протокольного декодера.
uri	Область поиска — поле URI заголовка HTTP.
host	Областью поиска для HTTP-сессии является поле Host (до переноса строки).
header	Областью поиска для HTTP-сессии являются http заголовки, smtp и pop3 команды, заголовки протоколов tls и ssh.
body	Областью поиска является содержимое пакетов HTTP.
file	Областью поиска являются: декодированное http содержимое, вложения в eml, ftp-data сессии.

ПримечаниеПри создании сигнатуры с модификатором **.where** поле диссектора автоматически считается, как поле с модификатором **.service=any** , то есть применяются все диссекторы, подходящие под значение модификатора **.where**.

.service

Данный модификатор области поиска предназначен для выбора диссектора:

.service=<MODE>;

где <MODE> может принимать следующие значения:

Наименование	Описание
http	Диссектор HTTP
dns	Диссектор DNS

Пример:

UASL(.name="block.google"; .protocol=tcp; .service=http; .host="google.com");

.distance, .within, .at, .startin

Данные модификаторы позволяют:

Наименование	Описание

Наименование	Описание
.distance	Пропустить заданное количество байтов (RANGE) от начала или последнего найденного блока. Задаётся в формате: <pre>.distance=<RANGE> [,<MODE>];</pre> где <RANGE> — это целое число, начиная с 0. Опциональные параметры (<MODE>) будут рассмотрены далее. Например, следующая запись задаёт пропуск 10 байтов с начала для первого шаблона или от последнего найденного для второго и последующего шаблонов: <pre>.distance=10;</pre> Пример использования опциональных параметров: - пропустить 10 байтов от начала: <pre>.distance=10, start;</pre> - пропустить 10 байтов от последнего найденного шаблона: <pre>.distance=10, match;</pre>
.within	Сканировать в заданном интервале (RANGE) от начала или последнего найденного блока (шаблон полностью попадает в заданный интервал). Задаётся в формате: <pre>.within=<RANGE> [,<MODE>];</pre> где <RANGE> — это целое число, начиная с 1. Опциональные параметры (<MODE>) будут рассмотрены далее. Например, следующая запись задаёт поиск с 1-го по 10-й байт с начала для первого шаблона или от последнего найденного для второго и последующего шаблона: <pre>.within=10;</pre> Пример использования опциональных параметров: - поиск с 1-го (с начала) по 10-й байт: <pre>.within=10, start;</pre> - поиск в диапазоне 10 байтов после последнего найденного шаблона: <pre>.within=10, match;</pre>
.startin	Сканировать в заданном интервале (RANGE) от начала или последнего найденного блока (для совпадения в заданный интервал может попадать только начало шаблона). Задаётся в формате: <pre>.startin=<RANGE> [,<MODE>];</pre> где <RANGE> — это целое число, начиная с 1. Опциональные параметры (<MODE>) будут рассмотрены далее.
.at	Проверять наличие шаблона в заданной позиции. Важно! Данный модификатор не совместим с модификаторами .distance и .within. Задаётся в формате: <pre>.at=<RANGE> [,<MODE>];</pre> где <RANGE> — это целое число, начиная с 0. Опциональные параметры (<MODE>) будут рассмотрены далее.

В следующей таблице представлены опциональные параметры:

--

Наименование	Описание
Наименование start	Поиск с начала потока данных. Описание
	Важно! Является значением по умолчанию для первого шаблона.
packet	Сканирование с начала пакета.
reverse	Поиск с конца пакета (удобно для проверки Next Protocol в ESP).
match	Поиск от последнего найденного шаблона. Важно! Является значением по умолчанию для второго и последующего шаблона.
lastmark	Сканирование от последней метки, выставленной с помощью .mark pset .

Примечание Если опциональный параметр для модификаторов `.distance` и `.within` совпадают, то отсчёт значения модификатора `.within` производится от значения `.distance`.

Например, запись:

```
.distance=10,match; .within=5,match
```

задаёт поиск в диапазоне с 10-го по 15-й байт от последнего найденного шаблона.

Примеры:

```
UASL(.name="sock5.proxy.detect"; .protocol=tcp; .flow=from_client; .seq=1,relative; .pattern="|05 01 00|"; .within=1;)
```

```
UASL(.name="block.google"; .protocol=tcp; .service=http;.pattern="google.com"; .within=512;)
```

.protocol

Данный модификатор позволяет указать протокол транспортного уровня, к которому будет применена сигнатура:

```
.protocol=<MODE>;
```

где `<MODE>` может принимать следующие значения:

- **icmp**: анализ трафика протокола ICMP.
- **udp**: анализ трафика протокола UDP.
- **tcp**: анализ трафика протокола TCP.

Примечание Можно задать только один протокол. Если протокол не указан, то сигнатура применяется только к трафику TCP и UDP.

Пример:

```
UASL(.name="block.google"; .protocol=tcp; .pattern="google.com";)
```

Частота срабатывания

Documentation: ☒

Product: NGFW

Version: 7.x

При задании частоты срабатывание сигнатуры COB будет происходить не при каждом совпадении, а только после обнаружения заданного количества совпадений за указанный промежуток времени. Данный параметр может быть полезен при написании сигнатур для детектирования, например, атак типа брутфорс.

Для указания частоты срабатывания:

`.rate=<count>, <period>;`

где `<count>` — количество срабатываний;

`<period>` — интервал времени (в секундах), за который должно произойти заданное количество срабатываний.

Следующий параметр является необязательным и задаёт параметр, по которому совпадения будут сгруппированы:

`.track=<MODE>;`

где `<MODE>` — свойство, по которому происходит отслеживание пакетов.

`<MODE>` может принимать следующие значения:

- **src_ip** — отслеживание по IP-адресу источника;
- **dst_ip** — отслеживание по IP-адресу назначения.

Если **.track** не указан, то счётчик учитывает все совпадения, и при достижении заданного лимита происходит срабатывание сигнатуры.

Например:

```
UASL(.name="pop3.brute.force"; .protocol=tcp; .pattern="USER"; .flow=from_server; .rate=3,60; .track=src_ip;)
```

Срабатывание сигнатуры произойдёт после обнаружения шаблона USER (`.pattern="USER";`) в пакетах, отправленных с одного IP-адреса (`.track=src_ip;`), более 3-х раз за 60 секунд (`.rate=3, 60;`).

Направление анализа

Documentation: ☒
Product: NGFW
Version: 7.x

Данный параметр является опциональным и позволяет применять сигнатуру к определённым потокам трафика. В результате можно создавать сигнатуры, которые будут анализировать трафик от клиента, от сервера или в обоих направлениях.

.flow=<MODE>;

где <MODE> может принимать следующие значения:

Наименование	Описание
from_client	Анализ трафика от клиента.
from_server	Анализ трафика от сервера.
bi_directional	Анализ трафика в обоих направлениях.

Поиск бинарных данных

Product: NGFW
Version: 7.x
Documentation: ☒

Параметр **.byte_test** позволяет сравнить байт с заданным значением и применим к данным, представленным в двоичном или символьном форматах.

Общий формат записи:

.byte_test = <bytes>,<operator>,<value>,<offset>[,<multiplier>][,<modifiers>];

Доступные параметры отображены в таблице:

Наименование	Описание
<bytes>	Количество байтов в текущей позиции с заданным смещением, считываемых из пакета. Может принимать значения 1, 2 или 4.
<size>	Длина строки; указывается для строковых данных.
*	Использование всех символов до первого нечислового символа.
<operator>	Оператор, использующийся для сравнения байта с заданным значением: • < — меньше; • > — больше; • = — равно; • != — не равно; • & — результат выполнения операции логического И для <bytes> и <MASK> (число, задающее интересующие биты) не равен 0; • ~ — результат выполнения операции логического И для <bytes> и <MASK> равен 0; • ^ — результат выполнения операции XOR для <bytes> и <MASK> не равен 0. Например: .byte_test=1,&,0x80,0; проверяет, что старший разряд первого байта поля данных пакета установлен в 1.
<value>	Значение, с которым выполняется сравнение, или размер пакета. Значение может быть задано с использованием префикса 0x; доступно использование арифметических операторов (+, -, *, /).
<offset>	Смещение в поле данных пакета: • relative: от последней точки совпадения. Если параметр смещения не указан, то анализ, по умолчанию, производится с начала пакета.
<post_offset>	Число байтов, на которое необходимо сместить точку начала сканирования. Важно! Применим для .byte_jump.
<multiplier>	Числовое значение, на которое необходимо умножить извлечённое число перед сравнением или переносом точки начала сканирования; параметр является опциональным.

Наименование	Описание
<modifiers>	Модификаторы (опционально): • big — обработка данных со старшего разряда; • little — обработка данных с младшего разряда; • string — данные в пакете являются строковыми; • hex — преобразование строки данных в шестнадцатеричное число; • dec — преобразование строки данных в десятичное число; • oct — преобразование строки данных в восьмеричное число; • align — округление числа конвертируемых байтов до следующей 32-битной границы; используется только для .byte_jump (например, 0 → 0; 1,2,3,4 → 4; 5,6 → 8 и т.д.).

Пример сравнения первых 4 байтов каждого из пакетов со значением 1234; данные в пакете представлены в символьном формате в десятичной системе счисления:

.byte_test=4,=,1234,0,string,dec;

Параметр **.byte_jump** переносит точку начала сканирования на указанное число байтов. Общий формат записи при обработке данных со старшего или младшего разряда (т.е. для модификаторов **big** и **little**):

.byte_jump = <bytes>,<offset>,<post_offset>[,<multiplier>][,<modifiers>];

Для строковых данных (модификатор **string**):

.byte_jump = (<size> | *),<offset>,<post_offset>[,<multiplier>][,<modifiers>];

Работа с метками

Documentation: ☒
Product: NGFW
Version: 7.x

Для каждого потока данных может быть указана именованная метка. Для ее указания используется:

.mark <parameter>=<value>;

где <value> - название метки (указывается с использованием кавычек "");

<parameter> может принимать значения, представленные в таблице ниже.

Сопоставление шаблонов основано, в большинстве случаев, на работе с пакетами данных. Метки используются в случаях, если шаблон атак присутствует в нескольких пакетах. Сигнатура, сработавшая для предыдущего пакета, может поставить метку; наличие меток проверяется при отправке пакетов в рамках одной сессии.

Наименование	Описание
set	Установить именованную метку для текущего потока данных.
pset	Установить и запомнить последнюю выставленную метку для возможности её использования с модификаторами области поиска .distance и .within .
clear	Удалить именованную метку.
toggle	Изменить статус метки.
test	Проверить, существует ли метка.
reset	Сбросить все метки.

Documentation: ☒
Product: NGFW
Version: 7.x

В данном разделе будут рассмотрены поля протоколов ICMP, TCP, UDP, HTTP.

Т.к. для одной сигнатуры можно задать только один протокол, то использование ниже представленных параметров автоматически определяет его, т.е равносильно использованию параметра **.protocol**.

ПримечаниеУказание параметров разных протоколов приведёт к ошибке.

ICMP

Для проверки свойств заголовка ICMP доступно использование следующих параметров:

Наименование	Описание
.icmp.type	Проверка типа ICMP. Доступно использование следующих операторов: =, !=.
.icmp.code	Проверка значения кода ICMP. Доступно использование следующих операторов: =, !=.
.icmp.id	Проверка значений идентификаторов ICMP. Доступно использование следующих операторов: =, !=.
.icmp.checksum	Проверка контрольной суммы, которая используется при обнаружении ошибок. Доступно использование следующих операторов: <, >, <=, >=, =, !=.
.icmp.data_size	Проверка размера поля данных пакета. Параметр используется для обнаружения пакетов аномальных размеров, зачастую использующихся для переполнения буфера. Доступно использование следующих операторов: <, >, <=, >=, =, !=. Если заданы несколько условий, то они объединяются логическим оператором И .

Примеры:

UASL(.name="Ping Death"; .protocol=icmp; .data_size>32000)
UASL(.name="Ping Death"; .icmp.data_size>32000)

TCP

Для проверки свойств заголовка TCP доступно использование следующих параметров:

Наименование	Описание
.tcp.sport	Проверка номера порта или диапазона портов источника. Доступно использование следующих операторов: =, !=.
.tcp.dport	Проверка номера порта или диапазона портов назначения. Доступно использование следующих операторов: =, !=.
.tcp.window_size	Проверка размера окна TCP. Поддерживаются следующие операторы: <, >, <=, >=, =, !=.
.tcp.checksum	Проверка контрольной суммы, использующейся для проверки на наличие ошибок при передаче и/или приеме отправленного пакета. Доступно использование следующих операторов: <, >, <=, >=, =, !=.

Наименование	Описание
.tcp.seq	Проверка значения порядковых номеров TCP. Поддерживаются следующие операторы: <, >, <=, >=, =, !=. Доступно использование модификатора relative — проверка относительно начального номера последовательности. Применение: .tcp.seq=<value>,relative; где <value> - порядковый номер TCP.
.tcp.flags	Проверка TCP-флагов: .tcp.flags=[<mod>]<tcp_flags>; где <mod> — модификатор. <tcp_flags> — флаг TCP, который может быть указан как в буквенном, так и в числовом (десятиричной или шестнадцатеричной системах) форматах. Флаги: • 0 — флаги не установлены; • F, 1, 0X001 — FIN; • S, 2, 0X002 — SYN; • R, 4, 0X004 — RST; • P, 8, 0X008 — PSH; • A, 16, 0X010 — ACK; • U, 32, 0X020 — URG; • E, 64, 0X040 — ECE; • C, 128, 0X080 — CWR; • N, 256, 0X100 — NS. Модификаторы: • * — должен быть установлен хотя бы один из заданных флагов, остальные не проверяются; • + — должны быть установлены все заданные флаги, остальные не проверяются; • ! — все заданные флаги должны быть сброшены, остальные не проверяются; • !0 — должен быть установлен хотя бы один флаг (любой). Важно! Если ни один из модификаторов не указан, то должны быть установлены все заданные флаги (строгое соответствие), остальные - сброшены.
.tcp.data_size	Размер полезной нагрузки пакета TCP (без учета заголовков). Поддерживаются следующие операторы: <, >, <=, >=, =, !=. Возможно указание как .data_size, тогда параметр будет относиться к протоколам TCP и UDP.

UDP

Для проверки свойств заголовка UDP доступно использование следующих параметров:

Наименование	Описание
.udp.sport	Проверка номера порта или диапазона портов источника. Доступно использование следующих операторов: =, !=.
.udp.dport	Проверка номера порта или диапазона портов назначения. Доступно использование следующих операторов: =, !=.

Наименование	Описание
.udp.checksum	Проверка контрольной суммы. Доступно использование следующих операторов: <, >, <=, >=, =, !=.
.udp.data_size	Размер полезной нагрузки пакета UDP (без учета заголовков). Поддерживаются следующие операторы: <, >, <=, >=, =, !=. Возможно указание как .data_size, тогда параметр будет относиться к протоколам TCP и UDP.

HTTP

Для проверки свойств заголовка HTTP доступно использование следующих параметров:

Наименование	Описание
.uri	Проверка поля идентификатора ресурса (URI).
.header	Проверка произвольных заголовков HTTP.
.body	Проверка тела содержимого запроса или ответа HTTP.
.host	Проверка доменного имени узла.

Примеры

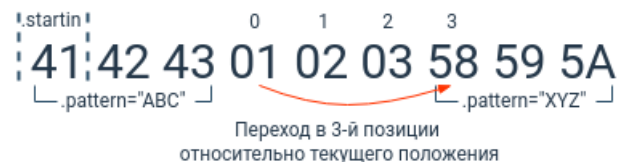
Documentation: ☒

Product: NGFW

Version: 7.x

В данном разделе приведены примеры, написанные с использования UASL.

Пример 1

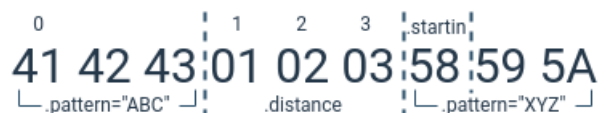


```
UASL(
  .id = 1;
  .pattern = "ABC"; .startin = 1;
  .pattern = "XYZ"; .at = 3, match;
)
```

В данном примере выполняется последовательный поиск 2-х шаблонов:

- поиск начала шаблона ABC в заданном диапазоне, т.е. первый байт заданного шаблона обязательно должен находится в диапазоне. Значение модификатора `.startin` равно 1 (диапазон = 1, поиск шаблона с начала сессии, т.к. первый модификатор по умолчанию — `start`) — байт, входящий в этот диапазон должен быть равен началу шаблона ABC;
- поиск шаблона XYZ начиная с позиции 4 (`.at=3;`) от последнего найденного шаблона ABC.

Пример 2



```
UASL(
  .id = 2;
  .pattern = "ABC"; .at = 0;
  .pattern = "XYZ"; .distance = 3, match; .startin = 1, match;
)
```

Выполняется последовательный поиск 2-х шаблонов:

- поиск начала шаблона ABC с начала пакета (`.at=0;`);
- пропустив 3 байта от последнего найденного шаблона ABC (`.distance=3,match;`) производится поиск начала шаблона XYZ; для срабатывания начало шаблона XYZ должно быть равно первому байту, с которого начинается поиск, т.к. `.startin=1,match;`

Пример 3

```
UASL(
  .id = 3;
  .pattern = "ABC";
  .pattern = "XYZ"; .distance = 3, match; .within = 3, match;
)
```

Производится последовательный поиск 2-х шаблонов:

- поиск шаблона ABC;
- через 3 байта от последнего найденного шаблона ABC (`.distance=3,match;`) производится поиск шаблона XYZ: шаблон XYZ должен полностью попадать в последующие 3 байта, т.к. `.within=3.`

