*Моделирование и анализ информационных систем*. Т. 26, № 1 (2019), с. 75–89
*Modeling and Analysis of Information Systems*. Vol. 26, No 1 (2019), pp. 75–89



**DOI:** 10.18255/1818-1015-2019-1-75-89

УДК 004.7

# Эффективный алгоритм разрешения коллизий в правилах политики безопасности

**Моржов С. В., Соколов В. А.**

*Поступила в редакцию 15 декабря 2018*
*После доработки 14 января 2019*
*Принята к публикации 18 февраля 2019*

**Аннотация.** Межсетевой экран является основным классическим инструментом для контроля и управления сетевым трафиком в локальной сети. Его задача — сравнивать проходящий через него сетевой трафик с установленными правилами безопасности. Эти правила, которые часто также называют политикой безопасности, могут быть определены как до, так и во время работы межсетевого экрана. Управление политикой безопасности крупных корпоративных сетей является сложной задачей. Для того чтобы правильно ее реализовать, правила фильтрации межсетевого экрана должны быть написаны и организованы аккуратно и без ошибок. Кроме того, процесс изменения или вставки новых правил должен выполняться только после тщательного анализа отношений между изменяемыми или вставляемыми правилами, а также правилами, которые уже существуют в политике безопасности. В данной статье авторы рассматривают классификацию отношений, в которых могут находиться правила политики безопасности между собой, и дают определение возможных коллизий между ними. Авторы представляют также новый эффективный алгоритм обнаружения и устранения коллизий в правилах межсетевого экрана на примере контроллера ПКС Floodlight.

**Ключевые слова:** список контроля доступа, межсетевой экран, программно-конфигурируемая сеть, ПКС, дерево политики безопасности

**Для цитирования:** Моржов С. В., Соколов В. А., "Эффективный алгоритм разрешения коллизий в правилах политики безопасности", *Моделирование и анализ информационных систем*, **26**:1 (2019), 75–89.

**Об авторах:**
Моржов Сергей Владимирович, аспирант, orcid.org/0000-0001-6652-3574
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, Ярославль, 150003, Россия, e-mail: smorzhov@gmail.com

Соколов Валерий Анатольевич, д-р физ.-мат. наук, профессор, orcid.org/0000-0003-1427-4937
НОМЦ Центр интегрируемых систем, Ярославский государственный университет им. П.Г. Демидова
ул. Советская, 14, Ярославль, 150003, Россия, e-mail: valery-sokolov@yandex.ru

**Благодарности:**
Работа выполнена при финансовой поддержке РФФИ, научный проект № 17-07-00823 А, и в рамках государственного задания Министерства образования и науки РФ, проект № 1.10160.2017/5.1

## Введение

Межсетевой экран является основным классическим инструментом для контроля и управления сетевым трафиком в локальной сети. Его задача — сравнивать прохо-

дящий через него сетевой трафик с установленными правилами безопасности. Эти правила, которые часто также называют политикой безопасности, могут быть определены как до, так и во время работы межсетевого экрана. Управление политикой безопасности крупных корпоративных сетей является сложной задачей. Для того чтобы правильно ее реализовать, правила фильтрации межсетевого экрана должны быть написаны и организованы аккуратно и без ошибок. Кроме того, процесс изменения или вставки новых правил должен выполняться только после тщательного анализа отношений между изменяемыми или вставляемыми правилами, а также правилами, которые уже существуют в политике безопасности.

В данной статье авторы рассматривают классификацию отношений, в которых могут находиться правила политики безопасности между собой, и дают определение возможных коллизий между ними. Авторы представляют также новый эффективный алгоритм обнаружения и устранения коллизий в правилах межсетевого экрана на примере контроллера ПКС Floodlight.

# 1. Формализация возможных коллизий в правилах политики безопасности

Основными источниками установки правил на сетевое оборудование в традиционной компьютерной сети являются межсетевой экран, список контроля доступа, система предотвращения вторжений, сервер IP-телефонии, системный администратор и т.п. Добавление нового правила без координации с другими источниками может привести к коллизиям с существующими правилами. Коллизия в правилах политики безопасности — это ситуация, когда в политике безопасности существуют два или более правила, значения фильтрующих атрибутов которых пересекаются. Наличие коллизий в правилах политики безопасности серьезно затрудняет анализ установленных правил политики безопасности, тем самым осложняя работу сетевого администратора. Кроме того, коллизии потенциально могут приводить к появлению брешей в политике безопасности, а следовательно, напрямую влияют на функционирование сети в целом.

Таким образом, при создании правил коммутации пакетов в традиционной сети могут возникать коллизии, которые необходимо разрешать.

## 1.1. Определение отношений между двумя правилами безопасности контроллера ПКС Floodlight

Правило политики безопасности[1] представляет собой кортеж, элементами которого являются пары атрибут/значение. Так как значением в общем случае является множество, правило может быть представлено, как совокупность или набор этих множеств. Таким образом, для сравнения правил между собой можно использовать отношения над множествами, представленные на рисунке 1.

Возможные отношения между двумя правилами могут быть определены следующим образом [1]:

[1] Здесь и далее, если не будет указано другого, под правилом политики безопасности будет пониматься правило политики безопасности контроллера ПКС Floodlight.

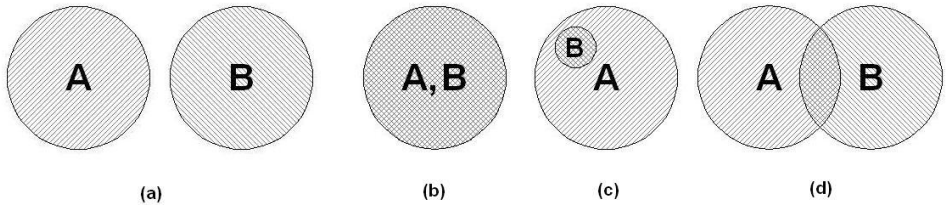


Рис. 1: (a) $A \cap B = \emptyset$; (b) $A = B$; (c) $B \subset A$; (d) $A \cap B \neq \emptyset$, $A \not\subset B \wedge B \not\subset A$

Fig. 1: (a) $A \cap B = \emptyset$; (b) $A = B$; (c) $B \subset A$; (d) $A \cap B \neq \emptyset$, $A \not\subset B \wedge B \not\subset A$

**Определение 1.** *Два правила $r$ и $s$ не пересекаются ($\mathfrak{R}_D$), если они имеют хотя бы один атрибут, для которого заданы непересекающиеся значения. Формально это можно записать так:*

$$r\mathfrak{R}_D s, \exists a \in attr, a_r \cap a_s = \emptyset \ . \tag{1}$$

Например, правила 1 и 2 ниже не пересекаются, так как имеют разные значения атрибута *«src-port»* (21 и 9050):

1. tcp, 193.168.*, 192.168.0.1, 21, allow
2. tcp, 193.168.*, 192.168.0.1, 9050, deny

**Определение 2.** *Два правила $r$ и $s$ полностью совпадают ($\mathfrak{R}_{EM}$), если значения всех их атрибутов равны. Формально это можно записать так:*

$$r\mathfrak{R}_{EM} s, \forall a \in attr, a_r = a_s \ . \tag{2}$$

Например, правила 1 и 2 полностью совпадают, так как значения всех атрибутов равны:

1. tcp, 193.168.*, 192.168.0.1, 21, allow
2. tcp, 193.168.*, 192.168.0.1, 21, allow

**Определение 3.** *Правило $r$ является подмножеством правила $s$ ($\mathfrak{R}_{IM}$), если существует хотя бы один атрибут правила $r$, значение которого является подмножеством к соответствующему атрибуту правила $s$, а остальные атрибуты правил равны. Формально это можно записать так:*

$$r\mathfrak{R}_{IM} s, \exists a \in attr, a_r \subset a_s \wedge \forall b \subset attr \setminus \{a\}, b_r = b_s \vee b_r \subset b_s \ . \tag{3}$$

Например, правило 1 является подмножеством правила 2, так как все атрибуты правила 1 равны соответствующим им атрибутам правила 2 за исключением атрибута «src-ip». У правила 1 значение атрибута *«src-ip»* является подмножеством значения атрибута *«src-ip»* правила 2:

1. tcp, 193.168.0.1, 192.168.0.1, 21, deny
2. tcp, 193.168.*, 192.168.0.1, 21, deny

**Определение 4.** *Два правила $r$ и $s$ коррелируют ($\mathfrak{R}_C$), если не выполнено условие определения 1, а также правила не являются подмножеством или надмножеством друг друга. Формально, это можно записать так:*

$$r\mathfrak{R}_C s, (r \neg \mathfrak{R}_D s) \wedge (r \neg \mathfrak{R}_{EM} s) \wedge (r \neg \mathfrak{R}_{IM} s) \ . \tag{4}$$

Например, правила 1 и 2 коррелируют, так как значения атрибутов *«nw-proto»* и *«src-port»* равны, у правила 1 значение атрибута *«src-ip»* является подмножеством значения соответствующего атрибута правила 2, а значение атрибута *«dst-ip»* правила 1 является надмножеством значения соответствующего атрибута правила 2:

1. tcp, 193.168.*.*, 21, deny
2. tcp, *, 192.168.0.1, 21, deny

**Лемма 1.** *Любые два правила, имеющие два атрибута, могут находиться в одном из четырех отношений:* $\mathfrak{R}_D, \mathfrak{R}_{EM}, \mathfrak{R}_{IM}$ *или* $\mathfrak{R}_C$.

*Доказательство.* Рассмотрим правила $R_x = \langle x_1, x_2\rangle$ и $R_y = \langle y_1, y_2\rangle$. Отношение между ними определяется отношением между соответствующими значениями их атрибутов, то есть $x_i \mathfrak{R} y_i$, где $\mathfrak{R} \in \{=, \subset, \supset, \bowtie\}, i = 1, 2$. Оператор $\bowtie$ определим следующим образом: $x \bowtie y \iff x \neq y \wedge x \not\subset y \wedge x \not\supset y \wedge x \cap y \neq \emptyset$. Рассмотрим все возможные отношения между атрибутами $R_x$ и $R_y$:

Если $x_1 = y_1$ и $x_2 = y_2$, то $R_x \mathfrak{R}_{EM} R_y$
Если $x_1 = y_1$ и $x_2 \subset y_2$, то $R_x \mathfrak{R}_{IM} R_y$
Если $x_1 = y_1$ и $x_2 \supset y_2$, то $R_x \mathfrak{R}_{IM} R_y$
Если $x_1 = y_1$ и $x_2 \bowtie y_2$, то $R_x \mathfrak{R}_C R_y$
Если $x_1 = y_1$ и $x_2 \neg\mathfrak{R} y_2$, то $R_x \mathfrak{R}_D R_y$
Если $x_1 \subset y_1$ и $x_2 = y_2$, то $R_x \mathfrak{R}_{IM} R_y$
Если $x_1 \subset y_1$ и $x_2 \subset y_2$, то $R_x \mathfrak{R}_{IM} R_y$
Если $x_1 \subset y_1$ и $x_2 \supset y_2$, то $R_x \mathfrak{R}_C R_y$
Если $x_1 \subset y_1$ и $x_2 \bowtie y_2$, то $R_x \mathfrak{R}_C R_y$
Если $x_1 \subset y_1$ и $x_2 \neg\mathfrak{R} y_2$, то $R_x \mathfrak{R}_D R_y$
Если $x_1 \supset y_1$ и $x_2 = y_2$, то $R_x \mathfrak{R}_{IM} R_y$
Если $x_1 \supset y_1$ и $x_2 \subset y_2$, то $R_x \mathfrak{R}_C R_y$
Если $x_1 \supset y_1$ и $x_2 \supset y_2$, то $R_x \mathfrak{R}_{IM} R_y$
Если $x_1 \supset y_1$ и $x_2 \bowtie y_2$, то $R_x \mathfrak{R}_C R_y$
Если $x_1 \supset y_1$ и $x_2 \neg\mathfrak{R} y_2$, то $R_x \mathfrak{R}_D R_y$
Если $x_1 \bowtie y_1$ и $x_2 = y_2$, то $R_x \mathfrak{R}_C R_y$
Если $x_1 \bowtie y_1$ и $x_2 \subset y_2$, то $R_x \mathfrak{R}_C R_y$
Если $x_1 \bowtie y_1$ и $x_2 \supset y_2$, то $R_x \mathfrak{R}_C R_y$
Если $x_1 \bowtie y_1$ и $x_2 \bowtie y_2$, то $R_x \mathfrak{R}_C R_y$
Если $x_1 \bowtie y_1$ и $x_2 \neg\mathfrak{R} y_2$, то $R_x \mathfrak{R}_D R_y$
Если $x_1 \neg\mathfrak{R} y_1$ и $x_2 = y_2$, то $R_x \mathfrak{R}_D R_y$
Если $x_1 \neg\mathfrak{R} y_1$ и $x_2 \subset y_2$, то $R_x \mathfrak{R}_D R_y$
Если $x_1 \neg\mathfrak{R} y_1$ и $x_2 \supset y_2$, то $R_x \mathfrak{R}_D R_y$
Если $x_1 \neg\mathfrak{R} y_1$ и $x_2 \bowtie y_2$, то $R_x \mathfrak{R}_D R_y$
Если $x_1 \neg\mathfrak{R} y_1$ и $x_2 \neg\mathfrak{R} y_2$, то $R_x \mathfrak{R}_D R_y$

Таким образом, было показано, что $R_x$ и $R_y$ всегда находятся в одном из четырех отношений: $\mathfrak{R}_D, \mathfrak{R}_{EM}, \mathfrak{R}_{IM}$ или $\mathfrak{R}_C$. □

**Лемма 2.** *Добавление одного атрибута к любым двум правилам* $R_x$ *и* $R_y$*, находящимся в отношении* $\mathfrak{R} \in \{\mathfrak{R}_D, \mathfrak{R}_{EM}, \mathfrak{R}_{IM}, \mathfrak{R}_C\}$*, оставляет правила* $R_x$ *и* $R_y$ *в прежнем отношении* $\mathfrak{R}$ *или переводит их в новое отношение*

$$\mathfrak{R}' \in \{\mathfrak{R}_D, \mathfrak{R}_{EM}, \mathfrak{R}_{IM}, \mathfrak{R}_C\}.$$

*Доказательство.* Рассмотрим правила $R_x = \langle x_1, ..., x_k \rangle$ и $R_y = \langle y_1, ..., y_k \rangle$. Добавим к $R_x$ атрибут $x_{k+1}$, а к $R_y$ атрибут $y_{k+1}$. Обозначим новые правила как $\mathfrak{R}'_x$ и $R'_y$ соответственно. Если $R_x$ и $R_y$ были в отношении $\mathfrak{R} \in \{\mathfrak{R}_D, \mathfrak{R}_{EM}, \mathfrak{R}_{IM}, \mathfrak{R}_C\}$, то $\mathfrak{R}'_x$ и $R'_y$ могут быть в одном из следующих отношений:

Если $R_x \mathfrak{R}_D R_y$ и $x_{k+1} = y_{k+1}$, то $R'_x \mathfrak{R}_D R'_y$
Если $R_x \mathfrak{R}_D R_y$ и $x_{k+1} \subset y_{k+1}$, то $R'_x \mathfrak{R}_D R'_y$
Если $R_x \mathfrak{R}_D R_y$ и $x_{k+1} \supset y_{k+1}$, то $R'_x \mathfrak{R}_D R'_y$
Если $R_x \mathfrak{R}_D R_y$ и $x_{k+1} \bowtie y_{k+1}$, то $R'_x \mathfrak{R}_D R'_y$
Если $R_x \mathfrak{R}_D R_y$ и $x_{k+1} \neg\mathfrak{R} y_{k+1}$, то $R'_x \mathfrak{R}_D R'_y$
Если $R_x \mathfrak{R}_{EM} R_y$ и $x_{k+1} = y_{k+1}$, то $R'_x \mathfrak{R}_{EM} R'_y$
Если $R_x \mathfrak{R}_{EM} R_y$ и $x_{k+1} \subset y_{k+1}$, то $R'_x \mathfrak{R}_{IM} R'_y$
Если $R_x \mathfrak{R}_{EM} R_y$ и $x_{k+1} \supset y_{k+1}$, то $R'_x \mathfrak{R}_{IM} R'_y$
Если $R_x \mathfrak{R}_{EM} R_y$ и $x_{k+1} \bowtie y_{k+1}$, то $R'_x \mathfrak{R}_C R'_y$
Если $R_x \mathfrak{R}_{EM} R_y$ и $x_{k+1} \neg\mathfrak{R} y_{k+1}$, то $R'_x \mathfrak{R}_D R'_y$
Если $R_x \mathfrak{R}_{IM} R_y$ и $x_{k+1} = y_{k+1}$, то $R'_x \mathfrak{R}_{IM} R'_y$
Если $R_x \mathfrak{R}_{IM} R_y$ и $x_{k+1} \subset y_{k+1}$, то $R'_x \mathfrak{R}_{IM} R'_y$
Если $R_x \mathfrak{R}_{IM} R_y$ и $x_{k+1} \supset y_{k+1}$, то $R'_x \mathfrak{R}_C R'_y$
Если $R_x \mathfrak{R}_{IM} R_y$ и $x_{k+1} \bowtie y_{k+1}$, то $R'_x \mathfrak{R}_C R'_y$
Если $R_x \mathfrak{R}_{IM} R_y$ и $x_{k+1} \neg\mathfrak{R} y_{k+1}$, то $R'_x \mathfrak{R}_D R'_y$
Если $R_x \mathfrak{R}_C R_y$ и $x_{k+1} = y_{k+1}$, то $R'_x \mathfrak{R}_C R'_y$
Если $R_x \mathfrak{R}_C R_y$ и $x_{k+1} \subset y_{k+1}$, то $R'_x \mathfrak{R}_C R'_y$
Если $R_x \mathfrak{R}_C R_y$ и $x_{k+1} \supset y_{k+1}$, то $R'_x \mathfrak{R}_C R'_y$
Если $R_x \mathfrak{R}_C R_y$ и $x_{k+1} \bowtie y_{k+1}$, то $R'_x \mathfrak{R}_C R'_y$
Если $R_x \mathfrak{R}_C R_y$ и $x_{k+1} \neg\mathfrak{R} y_{k+1}$, то $R'_x \mathfrak{R}_D R'_y$

Таким образом, было показано, что $R'_x$ и $R'_y$ всегда находятся в одном из четырех отношений: $\mathfrak{R}_D, \mathfrak{R}_{EM}, \mathfrak{R}_{IM}$ или $\mathfrak{R}_C$. □

**Теорема 1.** *Отношения $\mathfrak{R}_D, \mathfrak{R}_{EM}, \mathfrak{R}_{IM}, \mathfrak{R}_C$ образуют универсальное множество отношений между двумя правилами $R_x$ и $R_y$.*

*Доказательство.* Докажем теорему методом математической индукции.

Для $k = 2, R_x = \langle x_1, x_2 \rangle, R_y = \langle y_1, y_2 \rangle$ — верно (см. Лемму 1).

Пусть утверждение теоремы верно для $R_x = \langle x_1, ..., x_k \rangle, R_y = \langle y_1, ..., y_k \rangle$ и $R_x \mathfrak{R} R_y$, где $\mathfrak{R} \in \{\mathfrak{R}_D, \mathfrak{R}_{EM}, \mathfrak{R}_{IM}, \mathfrak{R}_C\}$. Добавим к $R_x$ атрибут $x_{k+1}$, а к $R_y$ атрибут $y_{k+1}$ и обозначим новые правила как $R'_x$ и $R'_y$ соответственно. Так как правила $R'_x$ и $R'_y$ были получены путем добавления одного атрибута к правилам $R_x$ и $R_y$ таким, что $R_x \mathfrak{R} R_y$, то по Лемме 2 $R'_x \mathfrak{R}' R'_y$, где $\mathfrak{R}' \in \{\mathfrak{R}_D, \mathfrak{R}_{EM}, \mathfrak{R}_{IM}, \mathfrak{R}_C\}$. □

## 1.2. Возможные коллизии между двумя правилами

Исходя из отношений между правилами, обозначенных в разделе 1.1., возможные коллизии можно определить следующим образом [2]:

**Определение 5.** *Перекрытие. Правило r перекрывается правилом s, если s имеет более высокий приоритет и фильтрует все пакеты, которые может отфильтро-*

*вать $r$. Как следствие, правило $r$ никогда не будет использовано. Формально правило $r$ перекрывается правилом $s$, если $s$ имеет более высокий приоритет, $r\mathfrak{R}_{EM}s$ и $action_r \neq action_s$, или $s$ имеет более высокий приоритет, $r\mathfrak{R}_{IM}s$ и $action_r \neq action_s$.*

**Определение 6.** *Корреляция. Два правила $r$ и $s$ коррелируют, если они фильтруют пересекающиеся множества пакетов, то есть найдутся такие пакеты, которые могут быть отфильтрованы правилом $r$ и $s$ одновременно. Формально правила $r$ и $s$ коррелируют, если $r\mathfrak{R}_C s$ и $action_r \neq action_s$.*

**Определение 7.** *Избыточность. Избыточное правило $r$ фильтрует те же пакеты, что и правило $s$, и политика безопасности не пострадает при удалении правила $r$. Формально правило $r$ избыточно по отношению к правилу $s$, если $s$ имеет более высокий приоритет, $r\mathfrak{R}_{EM}s$ и $action_r = action_s$, или $s$ имеет более высокий приоритет, $r\mathfrak{R}_{IM}s$ и $action_r = action_s$; в то же время, правило $s$ избыточно к правилу $r$, если $s$ имеет более высокий приоритет, $s\mathfrak{R}_{IM}r$ и $action_r = action_s$ и не существует $t$, такого что $s$ имеет более высокий приоритет по отношению к $t$, а $t$ имеет более высокий приоритет по отношению к $r$, $s\{\mathfrak{R}_{IM}, \mathfrak{R}_C\}t, action_r \neq action_t$.*

## 2. Разрешение коллизий

При разработке алгоритмов разрешения и недопущения возникновения коллизий в правилах межсетевого экрана или списке контроля доступа (Access Control List — ACL) разумно потребовать соблюдение следующих условий:

1. Алгоритмы должны корректно разрешать все возможные коллизии, определенные в разделе 1.2. как в уже существующей политике безопасности, так и в режиме реального времени. Последнее необходимо для реализации алгоритмов, например, в виде сетевого приложения, способного осуществлять постоянный мониторинг устанавливаемых правил, не допуская возникновения всех возможных коллизий.

2. Полученная в результате работы алгоритмов новая политика безопасности, свободная от всех возможных коллизий, должна фильтровать те же пакеты, что и исходная политика безопасности. Таким образом, новая политика безопасности не должна содержать уязвимостей, которых была лишена исходная политика безопасности.

3. Алгоритмы должны обеспечивать по возможности максимальную быстроту следующих операций над правилами политики безопасности: операции добавления, удаления, итерирования правил.

Принимая во внимание вышеизложенные условия, разработку алгоритма следует начать с выбора оптимальной структуры данных для хранения и представления правил политики безопасности.

## 2.1. Дерево правил

Будем называть правило «хорошим», если при добавлении его в политику безопасности она остается свободной от всех возможных коллизий. Если же последнее условие не выполняется, то правило будем называть «плохим». Разумно допустить, что «хорошие» правила добавляются в политику безопасности чаще, чем «плохие». Кроме этого, стоит отметить, что понять, «хорошее» перед нами правило или «плохое», нельзя, не сравнив его с правилами, ранее добавленными в политику безопасности. Очевиден факт: понять, что перед нами «хорошее» правило вычислительно не проще, а скорее всего сложнее, чем понять, что перед нами «плохое» правило, так как в общем случае вывод, что правило «хорошее», может быть сделан только после того, как мы убедимся, что новое правило не создает коллизий с каждым правилом, добавленным в политику безопасности ранее. Отсюда следует, что операция добавления «хорошего» правила должна быть как можно более вычислительно простой.

Рассмотрим базовые структуры данных, пригодных для хранения правил политики безопасности. Первым и наиболее очевидным выбором может быть хранение правил в виде массива. Данный подход обеспечит быстроту и удобство итерирования правил. Так как операция добавления наиболее частая, то трудоемкость алгоритма в случае хранения правил в массиве не может быть лучше $O(n)$ [2], потому что «хорошие» правила будут сравниваться со всеми, ранее добавленными. Кроме того, операции добавления и удаления правил в этом случае — вычислительно сложные, так как сопряжены с изменением размера массива. Отсюда общая трудоемкость будет хуже, чем $O(n)$. Использование словарей, хеш-таблиц или даже баз данных кардинально не улучшит оценочную трудоемкость, так как все эти структуры данных принципиально не отличаются от массива и не решают главной проблемы — каждое новое правило нельзя отнести к «хорошим» или «плохим», не сравнив его в общем случае со всеми ранее добавленными в политику безопасности правилами.

Рассмотрим древовидную структуру данных и покажем, что она способна удовлетворить всем поставленным условиям и что она является оптимальной для поставленной задачи. Так как алгоритм разрабатывается для контроллера ПКС Floodlight, при описании структуры данных будем иметь в виду правило модуля межсетевого экрана контроллера ПКС Floodlight. Оно состоит из 12 атрибутов (см. таблицу 1).

Каждый атрибут может принимать одно значение из определенного множества. Например, атрибут «dl-type» может принимать только значения из множества $\{ipv4, arp, *^3\}$, мощность которого — 3, а атрибут *«src-ip»* может быть равен любому из $(2^8 + 1)^4$ IP-адресов (учитываются возможные символы-джокеры). Упорядочим атрибуты (все, кроме *«priority»* и *«action»*) в порядке возрастания мощностей множеств их возможных значений. Атрибуты *«priority»* и *«action»* поместим в конец полученной последовательности, так как при определении возможной коллизии они играют второстепенную роль, и их следует учитывать в последнюю очередь. Более подробно об этом будет сказано дальше при описании алгоритмов разрешения коллизий.

[2] Здесь и далее для упрощения выкладок будет приводиться трудоемкость операции добавления одного правила в политику безопасности фиксированного размера $n$.

[3] Символ-джокер (wildcard) для замены любой строки символов

Таблица 1: Атрибуты правила межсетевого экрана контроллера ПКС Floodlight [5]

Table 1: Rule fields of Floodlight SDN controller

| **Атрибут** | **Значение** | **Описание** |
|---|---|---|
| dl-type | ARP или IPv4 | Протокол канального уровня |
| nw-proto | TCP или UDP или ICMP | Протокол сетевого уровня |
| switchid | xx:xx:xx:xx:xx:xx:xx:xx | Идентификационный номер коммутатора |
| src-inport | short | Номер входящего порта коммутатора |
| tp-src | short | Номер порта источника |
| tp-dst | short | Номер порта получателя |
| src-ip | A.B.C.D/M | IP-адрес источника |
| dst-ip | A.B.C.D/M | IP-адрес получателя |
| src-mac | xx:xx:xx:xx:xx:xx | MAC-адрес источника |
| dst-mac | xx:xx:xx:xx:xx:xx | MAC-адрес получателя |
| priority | integer | Приоритет правила |
| action | allow or deny | Разрешить или запретить пакеты, удовлетворяющие правилу |

Итак, в результате упорядочивания получен 12-мерный кортеж. Так как правила будут храниться в дереве, то 12-мерному кортежу должен соответствовать один путь от вершины до листа в этом дереве, а каждому пути из вершины до листа в дереве должен соответствовать один 12-мерный кортеж, представляющий правило политики безопасности. Установим взаимно однозначное соответствие между значениями 12-мерного кортежа и уровнями дерева: первое значение кортежа будет соответствовать вершине дерева, второе — вершинам на первом уровне, третье — вершинам на втором уровне и т.д., двенадцатый элемент кортежа будет соответствовать листьям на одиннадцатом уровне дерева (см. рисунок 2). Так как атрибут *«dl-type»* может иметь всего три значения, то он будет соответствовать корню дерева. Атрибут *«nw-proto»* имеет четыре различных значения. Ему будут соответствовать вершины первого уровня дерева правил. Листья на одиннадцатом уровне дерева будут содержать одно из двух возможных значений: *allow* – для разрешающего правила и *deny* – для запрещающего. Нумерация правил на рисунке 2 вставлена исключительно для удобства восприятия.

Пусть каждая вершина дерева правил [3] [4] содержит хеш-таблицу (пару ключ-значение). В качестве ключа в хеш-таблице будут храниться значения соответствующих уровню атрибутов правил политики безопасности, а в качестве значений для каждого из ключей в таблице будут храниться адреса памяти смежной вершины на уровень ниже. Таким образом в хеш-таблице корня дерева, представленного на рисунке 2, будут содержаться два ключа: *«ipv4»* и *«arp»*. Значения этих ключей — адреса на левое и правое поддеревья соответственно.

Отметим, что порядок элементов в кортеже, представляющем собой правило политики безопасности, выбран не случайно. Все атрибуты, за исключением *«priority»* и *«action»*, были упорядочены по возрастанию мощностей множеств их возможных значений с той целью, чтобы минимизировать длины нисходящих путей, возникающих при добавлении правил в дерево. Покажем, что такой подход обеспечивает наиболее эффективное расходование памяти, необходимой для хранения дерева правил.

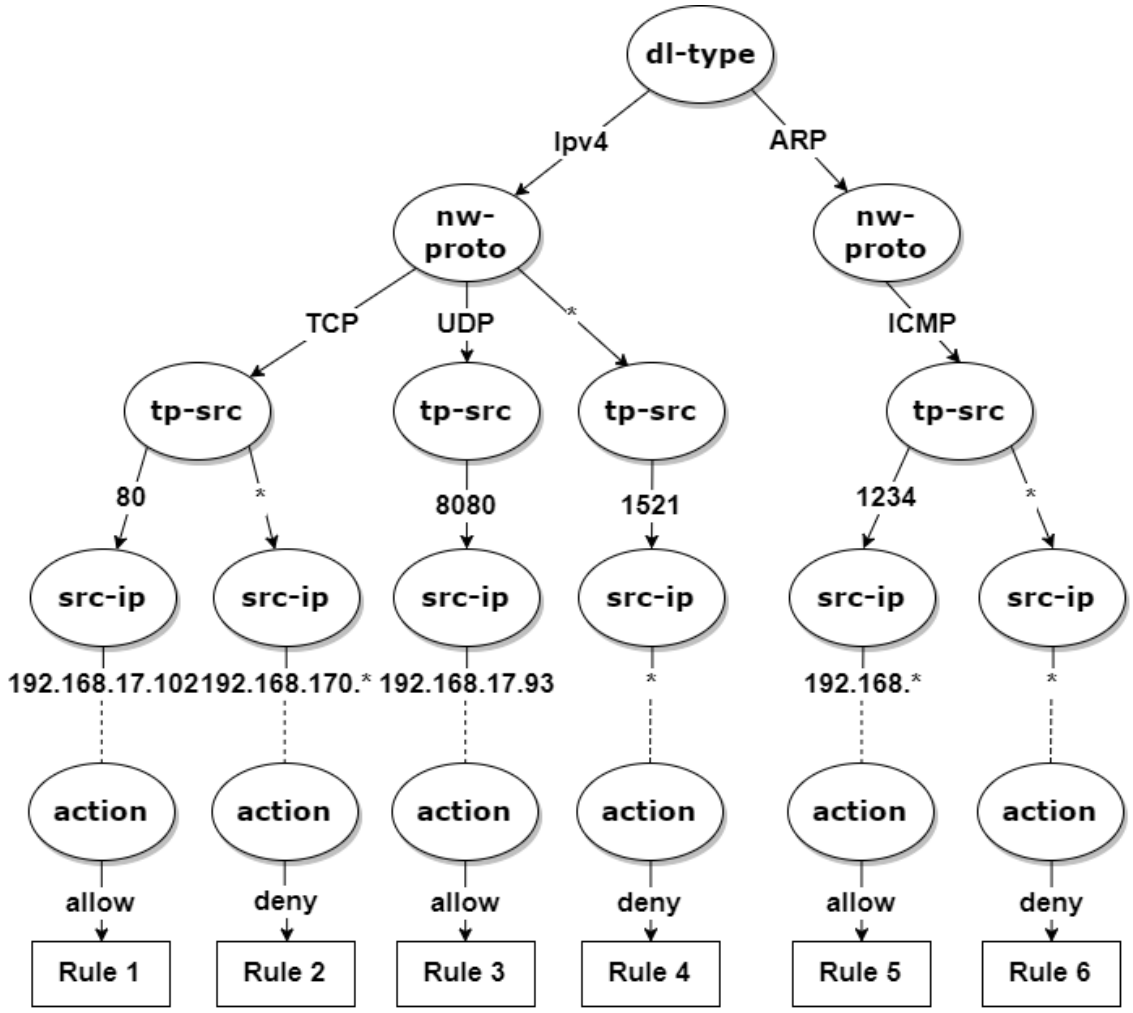


Рис. 2: Пример дерева правил

Fig. 2: Policy tree example

Пусть имеется два дерева правил. Пусть первое построено так, как было описано выше, а второе — так же, но с той лишь разницей, что атрибуты в кортеже упорядочены в порядке убывания мощностей множеств их возможных значений. Таким образом, первое дерево расширяется от корня к листьям, а второе дерево в общем случае становится широким сразу. Оценим максимально возможное количество хеш-таблиц для первого и второго дерева. При оценке, для простоты выкладок, без ограничения общности, предположим, что оба дерева хранят правила с пятью атрибутами — *«dl-type»*, *«nw-proto»*, *«tp-src»*, *«src-ip»* и *«action»*.

Рассмотрим первое дерево. В хеш-таблице первого уровня максимально может быть три записи — *«ipv4»*, *«arp»* и *«*»*. Отсюда следует, что на втором уровне будет содержаться 3 хеш-таблицы, каждая из которых может содержать по четыре записи — *«tcp»*, *«udp»*, *«icmp»* и *«*»*. Следовательно, на третьем уровне максимально может быть $3 * 4 = 12$ хеш-таблиц, содержащих $2^{16}$ записей, так как для хранения номера порта (третий уровень дерева соответствует атрибуту *«tp-src»* —

номер порта источника) выделяется 16 бит памяти. На четвертом уровне хранятся IP-адреса, а значит, хеш-таблица максимально может содержать $(2^8 + 1)^4$ записей, учитывая возможные символы-джокеры. Таким образом, первое дерево будет содержать $1 + 3 + 3 * 4 + 3 * 4 * 2^{16} = A$ хеш-таблиц.

Аналогично подсчитав количество хеш-таблиц для второго дерева, получим $1 + (2^8 + 1)^4 + (2^8 + 1)^4 * 2^{16} + (2^8 + 1)^4 * 2^{16} * 4 = B$, $B \ggg A$.

Очевидно, что, как бы ни упорядочивались атрибуты, количество хеш-таблиц $A$ в первом дереве правил будет наименьшим.

Для построения дерева правил может быть использован следующий алгоритм [4].

Algorithm 2.1: Алгоритм BuildPolicyTree

```
1   input: rule, field, node
2   begin
3       value_found ← FALSE
4       if field ≠ ACTION then
5       foreach branch in node->branch_list do
6           if branch->value = rule->field->value then
7               value_found ← TRUE
8               BuildPolicyTree(rule, field->next, branch->node)
9           elif branch->value ⊂ rule->field->value
10              or branch->value ⊃ rule->field->value then
11              BuildPolicyTree(rule, field->next, branch->node)
12      if value_found = FALSE then
13          new_branch ← new TreeBranch(rule, rule->field, rule->filed->value)
14          node->branch_list->add(new_branch)
15          if field ≠ ACTION then
16              BuildPolicyTree(rule, field->next, new_branch->node)
17          end if
18      end if
19  end
```

## 2.2. Алгоритм разрешения коллизий

На основе определенных в разделе 1.1. отношений между правилами, а также возможных коллизий, определенных в разделе 1.2. был разработан алгоритм борьбы с коллизиями. Основная идея алгоритм состоит в том, что новое правило нужно добавить в дерево правил, свободных от всяческих коллизий. Если путь в дереве, соответствующий новому правилу, совпадает с каким-нибудь путем в дереве, представляющим собой ранее добавленное правило, то была найдена коллизия, которую необходимо разрешить. В противном случае добавление нового правила оставит политику безопасности свободной от всяческих коллизий. Данная логика может быть имплементирована в алгоритм построения дерева правил. Ниже приведен алгоритм *DiscoverAnomaly*, представляющий собой усовершенствованный алгоритм *BuildPolicyTree* [4].

Algorithm 2.2: Алгоритм DiscoverAnomaly

```
1  input: rule, field, node, anomaly_state
2  begin
3      if field ≠ ACTION then
4          value_found ← FALSE
5          foreach branch in node->branch_list do
6              if branch->value = rule->field->value then
7                  value_found = TRUE
8                  if anomaly_state = NOANOMALY then
9                      anomaly_state ← REDUNDANT
10                     DiscoverAnomaly(rule, field->next, branch->node, anomaly_state)
11                 end if
12             else
13                 if rule->field->value ⊂ branch->value then
14                     if anomaly_state = GENERALIZATION then
15                         DiscoverAnomaly(rule, field->next, branch->node, CORRELATION)
16                     else
17                         DiscoverAnomaly(rule, field->next, branch->node, SHADOWING)
18                     end if
19                 elif rule->field->value ⊃ branch->value
20                     if anomaly_state = SHADOWING then
21                         DiscoverAnomaly(rule, field->next, branch->node, CORRELATION)
22                     else
23                         DiscoverAnomaly(rule ,field->next ,branch->node ,GENERALIZATION)
24                     end if
25                 end if
26             end if
27         if value_found = False then
28             new_branch ← new TreeBranch(rule, rule->field, rule->filed->value)
29             node->branch_list->add(new_branch)
30             DiscoverAnomaly(rule, field->next, new_branch->node, NOANOMALY)
31         end if
32     else
33         DecideAnomaly(rule, field, node, anomaly_state)
34     end if
35 end
```

Алгоритм *DecideAnomaly* [4], который выносит конечный вердикт относительно наличия или отсутствия коллизий, вызывается, когда путь добавляемого правила в дереве правил был полностью определен и достигнут атрибут *«action»*.

Algorithm 2.3: Алгоритм DecideAnomaly

```
1  input: rule, field, node, anomaly
2  begin
3      if node in branch_list then
4          branch ← node->branch_list->first()
5          if anomaly = CORRELATION then
6              if rule->action ≠ branch->value then
7                  report rule rule->id is in correlation with rule branch->rule->id
8              end if
9          elif anomaly = GENERALIZATION and rule->action ≠ branch->value then
10             report rule rule->id is a generalization of rule branch->rule->id
11         elif anomaly = GENERALIZATION and rule->action = branch->value then
12             branch->rule->setAnomaly(REDUNDANCY)
13             report rule branch->rule->id is redundant to rule rule->id
14         elif rule->action = branch->value then
15             anomaly ← REDUNDANCY
16             report rule rule->id is redundant to rule branch->rule->id
17         elif rule->action ≠ branch->value then
18             anomaly ← SHADOWING
19             report rule rule->id is shadowed by rule branch->rule->id
20         end if
21     end if
22     rule->setAnomaly(anomaly)
23 end
```

После того как тип коллизии был определен, новое правило может быть добавлено в политику безопасности (коллизии отсутствуют), не добавлено (перекрытие или избыточность) или добавлено частично (корреляция). Последний случай является наиболее интересным и сложным, поэтому его стоит рассмотреть отдельно.

Если было определено, что два правила коррелируют, то их можно «разбить» на непересекающиеся части и добавить полученные новые правила в политику безопасности. С этой целью сначала находится множество атрибутов, значения которых у данных правил различны. После этого для каждого из найденных атрибутов вызывается алгоритм *Split*, приведенный ниже, который изменяет правила $r$ и $s$ таким образом, чтобы они не пересекались. Алгоритм *Split* получает на вход два пересекающихся правила $r$ и $s$ и атрибут $a$, значение которого у данных правил не равно.

Как видно из рисунка 3, общая часть значения атрибута всегда начинается с $max(r.a.start, s.a.start)$ и заканчивается $min(r.a.end, s.a.end)$. Непересекающаяся часть до общей части всегда начинается с $min(r.a.start, s.a.start)$ и заканчивается $max(r.a.start, s.a.start) - 1$. Непересекающаяся часть после общей части всегда начинается с $min(r.a.end, s.a.end) + 1$ и заканчивается $max(r.a.end, s.a.end)$. Непересекающиеся части правил $r$ и $s$ могут быть добавлены в дерево правил посредством алгоритма *DiscoverAnomaly*, так как не гарантировано, что они не конфликтуют с уже существующими правилами.

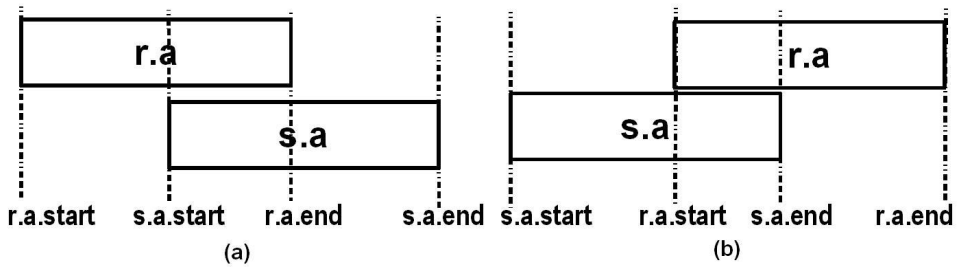


Рис. 3: (a) $r.a.start < s.a.start \wedge r.a.end < s.a.end$, таким образом, интервал может быть разбит на $[r.a.start, s.a.start - 1]$, $[s.a.start, r.a.end]$ и $[r.a.end + 1, s.a.end]$; (b) $r.a.start > s.a.start \wedge r.a.end > s.a.end$, таким образом, интервал может быть разбит на $[s.a.startr.a.start - 1]$, $[r.a.start, s.a.end]$ и $[s.a.end + 1, r.a.end]$

Fig. 3: (a) $r.a.start < s.a.start \wedge r.a.end < s.a.end$, thus, this interval can be split into $[r.a.start, s.a.start - 1]$, $[s.a.start, r.a.end]$, and $[r.a.end + 1, s.a.end]$; (b) $r.a.start > s.a.start \wedge r.a.end > s.a.end$, thus, this interval can be split into $[s.a.startr.a.start - 1]$, $[r.a.start, s.a.end]$, and $[s.a.end + 1, r.a.end]$

В алгоритме *Split* [4] сначала вычисляется общая часть двух пересекающихся правил, затем вычисляется непересекающаяся часть до общей части и добавляется в дерево правил как новое правило. После этого вычисляется непересекающаяся часть после общей части и также добавляется в дерево правил.

Algorithm 2.4: Алгоритм Split

```
1   input: r, s, a
2   begin
3       left ← min(r->a->start, s->a->start)
4       right ← max(r->a->end, s->a->end)
5       common_start ← max(r->a->start, s->a->start)
6       common_end ← min(r->a->end, s->a->end)
7       if r->a->start > s->a->start then
8           DiscoverAnomaly(⟨(left, common_start-1), s' rest attributes ⟩,
9               first_filed, first_node, NOANOMALY)
10      elif r->a->start < s->a->start then
11          DiscoverAnomaly(⟨(left, common_start-1), r's rest attributes ⟩,
12              first_filed, first_node, NOANOMALY))
13      elif r->a->end > s->a->end then
14          DiscoverAnomaly(⟨(common_end+1, right), r's rest attributes ⟩,
15              first_filed, first_node, NOANOMALY))
16      elif r->a->end < s->a->end then
17          DiscoverAnomaly(⟨(common_end+1, right), s' rest attributes ⟩,
18              first_filed, first_node, NOANOMALY))
19      r ← ⟨(common_start, common_end), r's rest attributes ⟩
20      s ← ⟨(common_start, common_end), s' rest attributes ⟩
21  end
```

После завершения работы алгоритмов дерево правил будет свободно от всех коллизий, обозначенных в разделе 1.2.

## 2.3. Анализ сложности алгоритмов разрешения коллизий

Сложность алгоритма определения и разрешения коллизий *DiscoverAnomaly* во многом зависит от правил, которые подаются на вход. Если алгоритм получает на вход правило, которое не пересекается или является надмножеством для всех правил в политике безопасности, то данное правило добавляется в политику безопасности сразу. Сложность данной операции в среднем — $O(1)$ (в худшем $O(n)$), так как в общем случае понадобится на всех 12 уровнях дерева правил по одному разу добавить в хеш-таблицу новое значение либо удостовериться, что значение уже присутствует и спуститься на уровень ниже в дереве (приведенный анализ трудоемкости не учитывает возможное перехеширование). Если добавляемое правило является подмножеством какого-либо ранее добавленного правила, такое правило будет добавлено в политику безопасности. Исходя из вышесказанного, сложность данной операции в среднем $O(1)$ (в худшем $O(n)$).

Поступающее правило будет отброшено, если оно равно какому-либо ранее установленному правилу. Чтобы это понять, в худшем случае придется пройти по дереву от корня и до листа, что, учитывая фиксированную высоту дерева, тоже имеет трудоемкость в среднем $O(1)$ (в худшем $O(n)$).

Если поступающее правило коррелирует с каким-либо ранее установленным, то данные правила будут разбиты на непересекающиеся части. В худшем случае число сгенерированных таким образом правил будет в два раза превышать число атрибутов правил, а установлены данные правила будут с помощью вызовов метода DiscoverAnomaly. Так как количество атрибутов у правил фиксировано, трудоемкость данной операции в среднем — $O(1)$ (в худшем $O(n)$).

Таким образом, мы получили, что сложность предложенного алгоритма в среднем — $O(1)$ (в худшем $O(n)$).

# Заключение

Основным преимуществом алгоритма разрешения коллизий, представленного в разделе 2.2., является, в общем случае, его константная трудоемкость. Однако данный алгоритм имеет один существенный недостаток. Без кардинальных изменений его трудно адаптировать для работы с правилами, содержащими символы-джокеры. Слово «трудно» здесь следует понимать в том смысле, что задача адаптации представленного алгоритма таким образом, чтобы он корректно обрабатывал символы-джокеры, сохраняя при этом константную трудоемкость, является нетривиальной. В будущих исследованиях планируется решить данную проблему.

Помимо всего прочего, вопрос о необходимости устранения всех типов коллизий остается открытым. Часто системные администраторы намеренно включают в политику безопасности правила, которые коррелируют с другими. Разрешение таких коллизий является ошибкой, а следовательно, в дальнейших работах по усовершенствованию представленного алгоритма следует это учитывать.

# Список литературы / References

[1] Al-Shaer E., et al., “Automated Firewall Analytics. Design, Configuration and Optimization”, Proceedings of 2016 IEEE Trustcom/BigDataSE/ISPA, 2014, 15–18.

[2] Abedin M., Nessa S., Khan L., Thuraisingham B., “Detection and Resolution of Anomalies in Firewall Policy Rules”, *Data and Applications Security XX*, Springer, 2006, 15–29.

[3] Morzhov S., Nikitinskiy M., “Development and research of the PreFirewall network application for Floodlight SDN controller”, 2018 Moscow Workshop on Electronic and Networking Technologies (MWENT) (Moscow, March 14–16), 2018, 1–4.

[4] Morzhov S., Sokolov V., Nikitinskiy M., Chaly D., “Building a Security Policy Tree for SDN Controllers”, 2018 International Scientific and Technical Conference Modern Computer Network Technologies (MoNeTec) (Moscow, October 25), 2018, 1–6.

[5] Morzhov S., Alekseev I., Nikitinskiy M., “Firewall application for Floodlight SDN controller”, XII International Siberian Conference on Control and Communications (SIBCON-2016) (Moscow, May 12–14), 2016, 1–5.

---

**Morzhov S. V., Sokolov V. A.**, "An Effective Algorithm for Collision Resolution in Security Policy Rules", *Modeling and Analysis of Information Systems*, **26**:1 (2019), 75–89.

**DOI:** 10.18255/1818-1015-2019-1-75-89

**Abstract.** A firewall is the main classic tool for monitoring and managing the network traffic on a local network. Its task is to compare the network traffic passing through it with the established security rules. These rules, which are often also called security policy, can be defined both before and during the operation of the firewall. Managing the security policy of large corporate networks is a complex task. In order to properly implement it, firewall filtering rules must be written and organized neatly and without errors. In addition, the process of changing or inserting new rules should be performed only after a careful analysis of the relationship between the rules being modified or inserted, as well as the rules that already exist in the security policy. In this article, the authors consider the classification of relations between security policy rules and also give the definition of all sorts of conflicts between them. In addition, the authors present a new efficient algorithm for detecting and resolving collisions in firewall rules by the example of the Floodlight SDN controller.

**Keywords:** access control list, firewall, software defined network, ACL, SDN, security policy tree

**On the authors:**
Sergey V. Morzhov, graduate student, orcid.org/0000-0001-6652-3574
P.G. Demidov Yaroslavl State University,
14 Sovetskay st., Yaroslavl, 150003, Russia, e-mail: smorzhov@gmail.com

Valeriy A. Sokolov, Doctor, Professor, orcid.org/0000-0003-1427-4937
Centre of Integrable Systems, P.G. Demidov Yaroslavl State University,
14 Sovetskay st., Yaroslavl, 150003, Russia, e-mail: valery-sokolov@yandex.ru

**Acknowledgments:**
The work was funded by Russian Foundation for Basic Research, according to the research projects No. 17-07-00823 A, and was carried out within the framework of the state program of the Ministry of Education and Science of the Russian Federation, project № 1.10160.2017/5.1