АНАЛИТИЧЕСКОЕ МОДЕЛИРОВАНИЕ РАБОТЫ ПРОГРАММНОГО КОДА С УЯЗВИМОСТЯМИ

Буйневич М.В.[1], Израилов К.Е.[2]

Аннотация:

Целью работы *является создание аналитической модели программного кода, потенциально содержащего уязвимости, для исследования его свойств в различных представлениях, таких как: Идея, Концептуальная модель, Архитектура, Алгоритмы, Исходный код, Ассемблерный код, Машинный код.*

Метод исследования *заключается в изучении программного кода на всем его жизненном цикле в 4-х следующих базовых аспектах:*

1) динамический – описывает изменения в процессе разработки как самого программного кода, так и находящихся в нем уязвимостей;
2) формализующий – задаёт аналитические правила для описания кода, уязвимостей, их взаимного влияния и динамики изменений, а также для ряда других понятий предметной области;
3) классификационный – вводит классы уязвимостей, зависящие от хронологии используемых представлений программного кода;
4) когнитивный – предлагает в формальном виде механизм восприятия кода человеком и инструментальным средством.

Основным научным результатом является аналитическая модель жизненного цикла программного обеспечения, совместившая в формальном виде фундаментальные понятия программного кода: Представление, Форма, Содержание, Уязвимость, Функционал. Частным научным результатом является схема преобразований кода с уязвимостями в одном представлении, детально описывающая как переход к новому представлению, так и модификацию кода в текущем.

Полученные результаты *являются мощным исследовательским инструментарием в области безопасного программного обеспечения и могут быть использованы для формализации и изучения механизмов возникновения и эволюции уязвимостей в программном коде, что позволит создавать соответствующие правила их обнаружения и нейтрализации современными методами (в том числе методом машинного обучения).*

Ключевые слова: *кибербезопасность, модель разработки, представления программного кода, анализ программы, уязвимость, формализация, классификация, динамика кода, когнитивность*

DOI:10.21681/2311-3456-2020-03-02-12

Введение

Актуальной проблемой информационной безопасности практически с зарождения всей IT-сферы является наличие уязвимостей в программном обеспечении [1]. Последние, возникающие как случайным, так и злонамеренным образом, приводят к функциональным нарушениям в обработке информационных потоков, включая утечку данных третьим лицам (нарушение конфиденциальности), невозможность чтения или записи данных (доступности) и неправомерное изменение данных (целостности). Реализация таких угроз в современном мире может иметь (и имеет) катастрофические последствия: от финансовых потерь до человеческих жертв, а также проигрыш целых кибервойн [2]. Несмотря на очевидные риски использования небезопасного программного обеспечения, применяемые меры противодействия не достигают, по мнению авторов, требуемого уровня по причине недостаточного понимания самой *многоаспектной* сущности уязвимости. Так, например, большинство способов их поиска в программном коде, хотя и считаются практически самой результативной мерой обеспечения информационной безопасности, воспринимают уязвимость, как некоторую обособленную область кода, выполняющую ярко выраженные нетипичные действия, чтобы их можно было считать подозрительными.

Однако, такая постановка задачи (где уязвимость – обособленный статический объект), на взгляд авторов, изначально является неверной по следующим причи-

1 Буйневич Михаил Викторович, доктор технических наук, профессор, профессор кафедры прикладной математики и информационных технологий Санкт-Петербургского университета государственной противопожарной службы МЧС России, профессор кафедры безопасности информационных систем Санкт-Петербургского государственного университета телекоммуникаций им. проф. М.А. Бонч-Бруевича, ORCHID: https://orcid.org/0000-0001-8146-0022. Scopus Author ID: 56122749800. Россия, Санкт-Петербург. E-mail: bmv1958@yandex.ru

2 Израилов Константин Евгеньевич, кандидат технических наук, доцент кафедры защищенных систем связи Санкт-Петербургского государственного университета телекоммуникаций им. проф. М.А. Бонч-Бруевича, старший научный сотрудник лаборатории проблем компьютерной безопасности Санкт-Петербургского института информатики и автоматизации Российской академии наук, ORCHID: https://orcid.org/0000-0002-9412-5693. Scopus Author ID: 56123238800. Россия, Санкт-Петербург. E-mail: konstantin.izrailov@mail.ru

нам. Во-первых, разработка программы представляет собой поэтапный (а зачастую и итеративный) процесс, в результате которого вид уязвимости на начальных и конечных этапах может существенно различаться, что затрудняет ее поиск классическими способами (сигнатурным, эвристическим и др.) [3, 4] – *динамический аспект*. Во-вторых, отсутствие четких правил для идентификации большинства уязвимостей (за редкими исключениями, например [5, 6]), особенно возникших на более концептуальных этапах разработки, приводит к высокому уровню ошибок как I-го, так и II-го рода – *формализующий аспект*. В-третьих, существующие подходы к классификации уязвимостей, определяющие как риски от их эксплуатации, так и применяемые способы поиска и нейтрализации, зачастую имеют пересекающиеся области определения, или же вовсе не позволяют отнести новую уязвимость к некоторому классу [7, 8] – *классификационный аспект*. И, в-четвертых, никак не учитываются терминологические особенности восприятия кода [9] и уязвимостей в нем [10] с грамматической (форма кода), семантической (логика содержания кода) и лингвистической (словарь содержания кода) сторон – *когнитивный аспект*. Исходя из вышесказанного, можно предположить, что более глубокое изучение такого сложного объекта, как уязвимость, будет полезно не только с академической точки зрения, но и позволит создавать более результативные способы их обнаружения и нейтрализации [11]. Решению такой задачи, а именно – *анализу программного кода с уязвимостями в динамическом, формализующем, классификационном и когнитивном аспектах*, результатом которого стала соответствующая *модель предметной области*, и посвящено данное исследование.

Динамический аспект

Опишем более подробно упомянутые аспекты, с позиции которых будет рассмотрен программный код и уязвимости в нем.

Под динамическим аспектом понимается то, что в процессе разработки и модификации программный код подвергается значительным структурным изменениям по мере прохождения его через различные представления разработки – от возникшего образа в голове у создателя до непосредственного воплощения в выполняемую программу. Как следствие, таким же изменениям подвержены и уязвимости, что существенно затрудняет их поиск в представлениях, отличных от тех, в которых они были созданы (в т.ч. после применения патчей [12]). Существует несколько таких представлений, переход любого программного кода между которыми осуществляется согласно выбранному процессу разработки, используемым подходам, языкам программирования, инструментальным средствам и т.п. Представления, через которые проходит программный код, многократно упоминались и описывались авторами [13, 14] и состоят из следующих основных (далее – Представления): 1) Идея – идеальный образ программы, которую задумал создатель; 2) Концептуальная модель – описание программы путем введения базовых элементов ее работы и описания их взаимосвязей; 3) Архитектура – деление программы на структурированные логические подсистемы и модули с указанием подходов для их реализации; 4) Алгоритмы – классическое описание логики работы кода с помощью блок-схем, диаграмм, псевдокода и т.п.; 5) Исходный код – код программы на языке программирования; 6) Ассемблерный код – код программы на языке платформы выполнения (сюда же относится и байт-код для виртуальных машин); 7) Машинный код – программа в бинарном виде, готовом для выполнения процессором (в случае виртуальных машин может создаваться уже в процессе выполнения).

И несмотря на то, что Представления, по сути, описывают один и тот же программный код, внешне они могут существенно отличаться: как на различных этапах разработки – иметь графический, текстовый, бинарный или другой вид, так и в рамках одного этапа – быть различными реализациями одного вида. Например, программный код, ответственный за вычисление факториала, на этапе проектирования может быть записан в математическом виде:

$$Y = X!,$$

хотя его непосредственная реализация на этапе кодирования будет иметь один из следующих программно-языковых (а именно C++) видов:

а) цикл

```
int factorial(int x) {
  int y = 1;
  while (x > 1) {
    y *= x;
    x --;
}
  return y;
}
```

б) рекурсия

```
int factorial(int x) {
  if (x == 0)
    return 1;
  return factorial(x - 1) * x;
}
```

Важным следствием с точки зрения уязвимости будет то, что ее вид на различных этапах также будет существенно отличаться. Так, для примера выше, добавление в конце математической записи одного лишнего знака факториала (т.е. *X!!*) приведет к существенному изменению вида исходного кода (функция *factorial()* должна будет вызываться дважды); это, естественно, потребует бóльших затрат на поиск уязвимости по более поздним Представлениям.

Формализующий аспект

Отсутствие аналитического определения самого понятия уязвимости – т.е. формализующий аспект – очевидно делает невозможным и создание строгих правил для их обнаружения [15]; существующие же подходы работают лишь с их внешними проявлениями –

DOI:10.21681/2311-3456-2020-03-02-12

оперируя внешним видом уязвимостей, а не внутренней логикой.

Попытка описать уязвимость, не приняв во внимание всех ее сторон, приводит к слишком размытым правилам, захватывающим и внешне близкий корректный программный код. Конкретизация же параметров правил хоть и позволяет не срабатывать на безопасном коде, однако и не все модификации уязвимостей удается обнаружить.

Как результат, часть уязвимостей находится некорректно (ошибка II-рода), а часть пропускается (ошибка I-рода). Оправданным может оказаться рассмотрение появления уязвимости не как внесение некой аномалии в код (что абсолютно формализовать практически невозможно), а как отличие функционала кода в двух Представлениях (поскольку функционал программы в момент ее задумки и вплоть до конечной реализации не должен меняться, иначе это будет уже другая программа).

В простейшем аналитическом виде подобная ситуация может быть записана как:

$$V = Funct_{Rep_2} - Funct_{Rep_1}, \quad (1)$$

где V – уязвимость, $Funct_{Rep_1}$ и $Funct_{Rep_2}$ – функционал 1-го и 2-го Представлений (Rep_1 и Rep_2 – сокр. от англ. *Representation*). В интересах детектирования такой ситуации авторами будет предпринята попытка частичной формализации процесса преобразования программного кода, а значит и содержащихся в нем уязвимостей.

Для подтверждения реальности вышеописанного проблемного вопроса формализующего аспекта приведем гипотетический пример исходного кода с уязвимостями и покажем недостатки классических правил детектирования, работающих исключительно с ее видом в коде. Предположим, в модуле идентификации и аутентификации реализуется функция сравнения логина и пароля, введенных пользователем, с разрешенными; также в модуль встроен логин и пароль администратора. Программный код функции в Представлении концептуальной модели в математическом виде может быть следующим:

$$Funct_{Auth}(login, password) = \quad (2)$$
$$= \begin{cases} 1, \begin{bmatrix} \langle login, password\rangle \in \{\langle login_i, password_i\rangle\}_{Users} \\ \langle login, password\rangle \equiv \langle login_{admin}, password_{admin}\rangle \end{bmatrix}, \\ 0, \langle login, password\rangle \notin \{\langle login_i, password_i\rangle\} \end{cases}$$

где $Funct_{Auth}()$ – функция сравнения логина *(login)* и пароля (*password*) пользователя, возвращающая 1 в случае совпадения, иначе – 0; <*login,password*> – кортеж из логина и пароля пользователя; <$login_i$,$password_i$> – кортеж из *i*-го логина и пароля в базе данных разрешенных пользователей $\{...\}_{Users}$; <$login_{admin}$,$password_{admin}$>– кортеж из встроенных логина и пароля администратора.

Предположим также, что в данном Представлении в функцию внесены две уязвимости в виде встроенных программных закладок, которые делают разрешенными изначально отсутствующих пользователей, т.е. $Funct_{Auth}()$ возвращает 1 по следующему принципу – первая (уязвимость № 1) проверяет совпадение введенного логина со встроенным (злоумышленником), а вторая (уязвимость № 2) – сравнивает кортеж из логина и пароля со встроенными:

$$Funct'_{Auth}(login, password) = \quad (3)$$
$$= \begin{cases} 1, \begin{bmatrix} \langle login, password\rangle \in \{\langle login_i, password_i\rangle\}_{Users} \\ \langle login, password\rangle \equiv \langle login_{admin}, password_{admin}\rangle \\ login \equiv login_{Vul_1} \\ \langle login, password\rangle \equiv \langle login_{Vul_2}, password_{Vul_2}\rangle \end{bmatrix}, \\ 0, \langle login, password\rangle \notin \{\langle login_i, password_i\rangle\} \end{cases},$$

где $Funct'_{Auth}()$ – функция сравнения логина и пароля со внедренными закладками; $login_{Vul_1}$ – логин, встроенный злоумышленником при внедрении 1-й закладки Vul_1; <$login_{Vul_2}$,$password_{Vul_2}$> – кортеж из логина и пароля, встроенный злоумышленником при внедрении 2-й закладки Vul_2. Очевидно, что внесенные уязвимости отражаются на функционале $Funct'_{Auth}()$ в виде двух дополнительных сравнений, а уязвимости согласно аналитическому виду (1) могут быть записаны, как:

$$\begin{cases} \text{для } Vul_1: Funct'_{Auth}(login, password) = 1, \\ login \equiv login_{Vul_1} \\ \text{для } Vul_2: Funct'_{Auth}(login, password) = 1, \\ \langle login, password\rangle \equiv \langle login_{Vul_2}, password_{Vul_2}\rangle \end{cases} \quad (4)$$

Функция с закладками в Представлении исходного кода будет иметь следующий вид:

```
int FunctAuth(string login, string password)
{
  // Уязвимость № 1
  if (login == login_vul_1)
  return 1;
  // Уязвимость № 2
   if (login == login_vul_2 && password ==
password_vul_2)
    return 1;
   // Проверка встроенного логина и пароля
администратора
   if (login == login_admin && password ==
password_admin)
    return 1;
   // Стандартная проверка логина и пароля
по базе пользователей
  if (users.Find(login, password) == 1)
    return 1;
   // Пользователь с логином и паролем не
найден
  return 0;
}
```

Можно предположить 2 варианта правил поиска этих уязвимостей по их признакам в коде. Алгоритм первого заключается в анализе критической функции $Funct_{Auth}$ () на предмет наличия в ней каких-либо встроенных проверок логина и пароля, что приведет к срабатыванию не только на уязвимость № 1, но и на проверке встроенного логина и пароля администратора – возникнет ошибка I-рода. Алгоритм второго правила будет менее строгим: поиск встроенных проверок логина приведет к его срабатыванию только на уязвимость № 1, пропустив при этом уязвимость № 2 – возникнет ошибка II-го рода. Хотя наиболее корректное правило должно работать именно в Представлении концептуальной модели (3), поскольку функционал кода поменялся именно в нем, т.к. добавились новые элементы – для Vul_1 и Vul_2.

Классификационный аспект

Проблема классификации уязвимостей, вытекающая из динамического аспекта, заключается в практическом отсутствии классов, связанных с хронологией появления уязвимостей в коде, и требует своего решения, хотя существующие стандарты частично и вводят соответствующие понятия (например, «уязвимость кода» или «уязвимость архитектуры»[3]). Это, во-первых, сделает такое деление достаточно четким (т.к. все Представления связаны с качественно разными задачами, решаемыми на них); во-вторых, позволит задавать правила поиска уязвимостей в терминах текущего Представления, повысив тем самым их результативность; в-третьих, предоставит возможность выбора наиболее подходящих способов поиска, которые имеют наилучшую эффективность лишь для определенных Представлений. Так, например, некорректно спроектированная архитектура вряд ли может быть эффективно идентифицирована анализом исключительно исходного кода, ошибка в логике работы алгоритмов подпрограмм будет «размазана» на несколько разнесенных инструкций в машинном коде [16], а неверный доступ к битовому полю через инструкцию машинного кода и вовсе не будет иметь отражения на уровне архитектуры [17].

Следуя формуле (1), согласно которой уязвимость является отличием функционала программного кода в различных Представлениях, а также формуле (7), далее обобщенно описывающей преобразование Представлений, каждой такой уязвимости можно сопоставить соответствующий класс:

$$\begin{cases} ClassV = \{ClassV_1 \dots ClassV_{N_{Rep}}\} \\ V \in ClassV_i, F_{Rep_{i+1}} \neq F_{Rep_i} \end{cases}, \quad (5)$$

где *ClassV* – полная классификация уязвимостей; $ClassV_i$ – класс уязвимости, соответствующий *i*-му Представлению; N_{Rep} – количество Представлений.

Используя полную классификацию и проходимые программным кодом Представления, предлагается следующая авторская страто-классификация уязвимостей (т.е. полученная последовательным формированием):

- концептуальная («нулевого» уровня) – ошибочный выбор базовых элементов концептуальной модели или их взаимосвязи;
- высокоуровневая – нарушение общих принципов функционирования архитектуры, ее безопасности и т.п. [18];
- среднеуровневая – ошибки в логике алгоритмов подпрограмм (в т.ч. передача неверных входных параметров и возвращаемых значений);
- низкоуровневая – неверная реализация вычислений, доступа к данным и т.п.;
- атомарная – ошибки в действиях (инструкциях) для текущей среды выполнения кода.

Когнитивный аспект

Наиболее интересным и, безусловно, обладающим научной новизной аспектом может считаться когнитивный – связанный непосредственно с познанием кода как человеком, так и автоматом (также обрабатывающим код). При этом, следуя канонам категориальных делений, объекты познаваемого мира являются взаимосвязанной совокупностью двух компонент-противоположностей: Формы vs Содержания [19]. Применительно к программному коду это означает, что в нем есть внешний вид (то, как код «читается» извне – иначе, описание) и внутренняя логика (то, что именно код делает – иначе, функционал):

$$Q = \langle F|C\rangle, \quad (6)$$

где *Q* – программный код, *F* – Форма-компонента кода, *C* – Содержание-компонента кода. Следуя же динамическому аспекту, от Представления к Представлению Содержание кода без уязвимостей инвариантно, а его Форма будет меняться (очевидно, от человекоориентированного вида к машиноориентированному):

$$\begin{cases} \langle F^{Rep_i}|C\rangle \rightarrow \langle F^{Rep_{i+1}}|C\rangle \\ Human(F^{Rep_i}) < Human(F^{Rep_{i+1}}) \end{cases}, \quad (7)$$

где $F^{Rep_{i+1}}$ – Форма кода в следующем *i+1*-ом Представлении Rep_{i+1}, полученном (→) из текущего *i*-го Представления Rep_i с Формой F^{Rep_i}; *Human* () – функция получения уровня человекоориентированности Представления (включающего также комментарии, содержание которых взаимосвязано со смыслом кода [20]).

По аналогии с разговорными языками понимание человеком любого текста может быть обусловлено 3-мя сущностями: формой текста – его внешним

3 ГОСТ Р 56546-2015. Защита информации. Уязвимости информационных систем. Классификация уязвимостей информационных систем.

проявлением через элементы (буквы, слова, знаки препинания и т.п.); логикой текста – конкретной осмысленной последовательностью его форм; словарем текста – значением каждой элементарной формы, которыми оперирует логика. Аналогично дело обстоит и с программным кодом. Следует заметить, что некоторая логика без используемого ей словаря имеет мало смысла – это подобно тому, как человек читает незнакомый ему рассказ или подпрограмму кода – он может уловить общий ход мысли (например, участие в различных частях и взаимосвязь персонажей или переменных программы, и то при условии, что используется известная парадигма), однако сам конкретный смысл всего будет ему не ясен [21]. Таким образом, можно поделить Содержание на две подкомпоненты:

$$C = C^L \cup C^D, \tag{8}$$

где C^L – подкомпонента-логика Содержания кода, C^D – подкомпонента-алфавит Содержания кода.

Проиллюстрируем это утверждение на следующем примере тривиального кода (на языке C++), складывающего два числа (обозначив код, как Q_0):

$$Q_0 \equiv «Z=X+Y;»$$

Очевидно, последовательность токенов (разделенных элементов кода) из множества идентификаторов *X*, *Y* и *Z*, операторов «=» и «+», а также специального символа «;» определяют Форму программного кода, которая может быть записана, как:

$$\begin{cases} F(Q) = \bigcup_{i=1}^{N_{Tok}} F_i \\ F_i \in \{+, -, *, /, =, \dots, 1,2,3 \dots if, else, \dots\} \cup \\ \cup \left\{Ident_1 \dots Ident_{N_{Uniq}(Idents(Q))}\right\} \end{cases}, \tag{9}$$

где $N_{Uniq}(E)$ – функция получения количества уникальных элементов некого множества *E*; *F*(*Q*) – Форма кода *Q* в виде неупорядоченного множества базовых неделимых Форм – ее элементов, количество которых равно количеству токенов исходного кода N_{Tok}; F_i – проиндексированные уникальные элементы Формы, составляющие ее словарь для конкретного языка программирования; *+,-,*,/,=,...,1,2,3...if,else,...* – все возможные элементы словаря Форм; $Ident_1 \dots Ident_{N_{Uniq}(Idents(Q))}$ – все возможные уникальные идентификаторы словаря Форм, количество которых определяется с помощью функции N_{Uniq} () от множества идентификаторов в программном коде – *Idents*(*Q*).

Для вышеупомянутого примера Форма имеет следующее значение:

$$F(Q_0) = \{F_1(Q_0), \dots, F_6(Q_0)\} = \\ = \{+ = X\ Y\ Z\}. \tag{10}$$

Также предположим для простоты, что словарь Форм состоит только из этих элементов, а именно:

$$\begin{cases} F_1 = " + " \\ F_2 = " = " \\ F_3 = "X" \\ F_4 = "Y" \\ F_5 = "Z" \\ F_6 = ";" \end{cases}. \tag{11}$$

Согласно приведенной аналогии с разговорным текстом, логика Содержания кода, соответствующая по сути хранимой им информации, и используя наиболее частую ее интерпретацию (в основном, из статистической физики [22, 23] и теории информации [24, 25]), как обратную меру энтропии – т.е. переход системы из хаотичного состояния в более упорядоченное – может быть определена, как некоторый порядок расположения элементов Формы:

$$C^L = i_1, \dots, i_{N_{Tok}}, \tag{12}$$

где i_j – индекс элемента в словаре Формы для *j*-го токена.

Таким образом, логика Содержания задается путем упорядочения элементов Формы – т.е. понижением неопределенности. Противоположной ситуацией будет отсутствие такой упорядоченности, при которой Формы расположены в случайном порядке, что не несет никакой информации. Логика Содержания примера имеет следующий вид:

$$C^L(Q_0) = 4\ 2\ 5\ 1\ 6\ 2, \tag{13}$$

где последовательность чисел 5 2 3 1 4 6 – индексы элементов (F_i) упрощенного словаря Форм, задавая тем самым сам код:

$$F = F_5\ F_2\ F_3\ F_1\ F_4\ F_6. \tag{14}$$

Резюмируя, понимание сути программного кода происходит опосредованно через его Форму, по составу и расположению элементов которой «собирается» его логика. Однако, простое понимание логики Содержания кода будет бессмысленным без понимания базиса этой логики – т.е. Содержания элементов Формы и их последовательностей. Так, например, применение операции «+» для чисел «*X*» и «*Y*» может означать не их сложение, а объединение в множество двух чисел или даже умножение – все это зависит от смысловой нагрузки как на токен «+», так и на остальные токены. Таким образом, логика Содержания кода должна быть построена в терминах некоторого словаря Содержания, значение элементов которого и составляют конечный смысл кода.

Поскольку смысл программного кода заключается в обработке данных по некоторому закону:

$$Out = Funct(In), \tag{15}$$

где *In/Out* – входные/выходные данные, *Funct*() – закон обработки данных программой или ее функционал, то именно этот функционал и определяется конечным Содержанием, состоящим из логики Содержания и словаря Содержания:

$$\begin{cases} Funct() = \bigcup funct_i \\ funct_i = C^D(C_i^L) \end{cases}, \tag{16}$$

где $funct_i$ – элементарные функции кода (например, сложение или приравнивание), соответствующие значению элементов словаря Содержания $C^D()$ для конкретной элементарной логики Содержания C_i^L.

Применительно к «сквозному» примеру программного кода словарь его Содержания будет следующим:

$$\begin{cases} C^D(1) \equiv f_{Add}(A,B) \\ C^D(2) \equiv f_{Assign}(A,B) \\ C^D(3) \equiv f_{Value}(X) \\ C^D(4) \equiv f_{Value}(Y) \\ C^D(5) \equiv f_{Value}(Z) \\ C^D(6) \equiv f_{End}() \end{cases}, \tag{17}$$

где $f_{Add}(A,B)$ *– сложение двух чисел A и B,* $f_{Assign}(A,B)$ *– приравнивание числа B к числу A,* $f_{Value}(A)$ *– получение значения идентификатора A,* $f_{End}()$ *– окончание выражения, вычисляемого функционалом.*

Схема создания и работы программного кода

Сам по себе программный код не может выполнять каких-либо действий – он лишь задает точный функционал, который воспринимается соответствующим Исполнителем, обрабатывающим данные. При этом в качестве Исполнителя может выступать как человек, так и автомат. Первый читает код и производит мысленные действия над гипотетическими данными – например, понимает, что выражение «*Z=X+Y*» означает прибавление к значению переменной *X* значения переменной *Y* согласно таблице сложения, результат которого необходимо сопоставить с переменной *Z*. Второй (естественно, если он представляет собой интерпретатор математических выражений) принимает на вход некоторое число в переменной *X*, выбирает функцию для операции «+», применяет ее к переменной *Y*, выбирает функцию для операции «=» и применяет ее к переменной *Z*. При этом действия как человека, так и автомата, задаются общим программным кодом (неважно, созданным ли одним из них или же являющимся внешним продуктом), воспринимаемым обеими сторонами через Форму и Содержание, при этом используя собственные словари для преобразования элементов Содержания в конкретные действия – т.е. выполняемый функционал.

Совокупность приведенных формальных записей Формы и Содержания программного кода с уязвимостями составляют его аналитическую модель в каждом из Представлений – получаемую из предыдущей и используемую для последующей (7). А исходя из вышеописанных интерпретаций, процесс преобразований между Представлениями можно описать в виде следующей схемы (рис. 1).

На схеме нижние индексы *S* и *A* обозначают, соответственно, Синтез (от англ. *Synthesis*) и Анализ (от англ. *Analysis*).

Следуя схеме, программный код воспринимается одинаковым образом как после создания его из предыдущего Представления, так и перед получением последующего. Рассмотрим два основных процесса, описываемых схемой: преобразование кода к новому Представлению и его модификация в текущем.

В процессе преобразования программного кода к следующему Представлению из функциональных действий – набору действий, применимых по некоторому правилу к входным данным (*In*) с получением результирующих выходных (*Out*), – создается соответствующий функционал $Funct_S()$, по которому, используя словарь Содержания C_S^D, упорядочиваются элементами логики Содержания C_S^L, уже по которым формируется последовательность элементов Формы кода. Синтезированный подобным образом программный код в нужном Представлении подвергается следующей обработке – происходит его анализ путем восприятия через Форму данного Представления *F*, элементы которой сопоставляются с элементами логики Содержания C_A^L, а затем на основании их упорядоченности и с использованием словаря Содержания C_A^D понимается функционал кода $Funct_A()$, который посредством функциональных действий применим к входным данным. Описанный процесс преобразования может формально быть записан следующим образом:

DOI:10.21681/2311-3456-2020-03-02-12

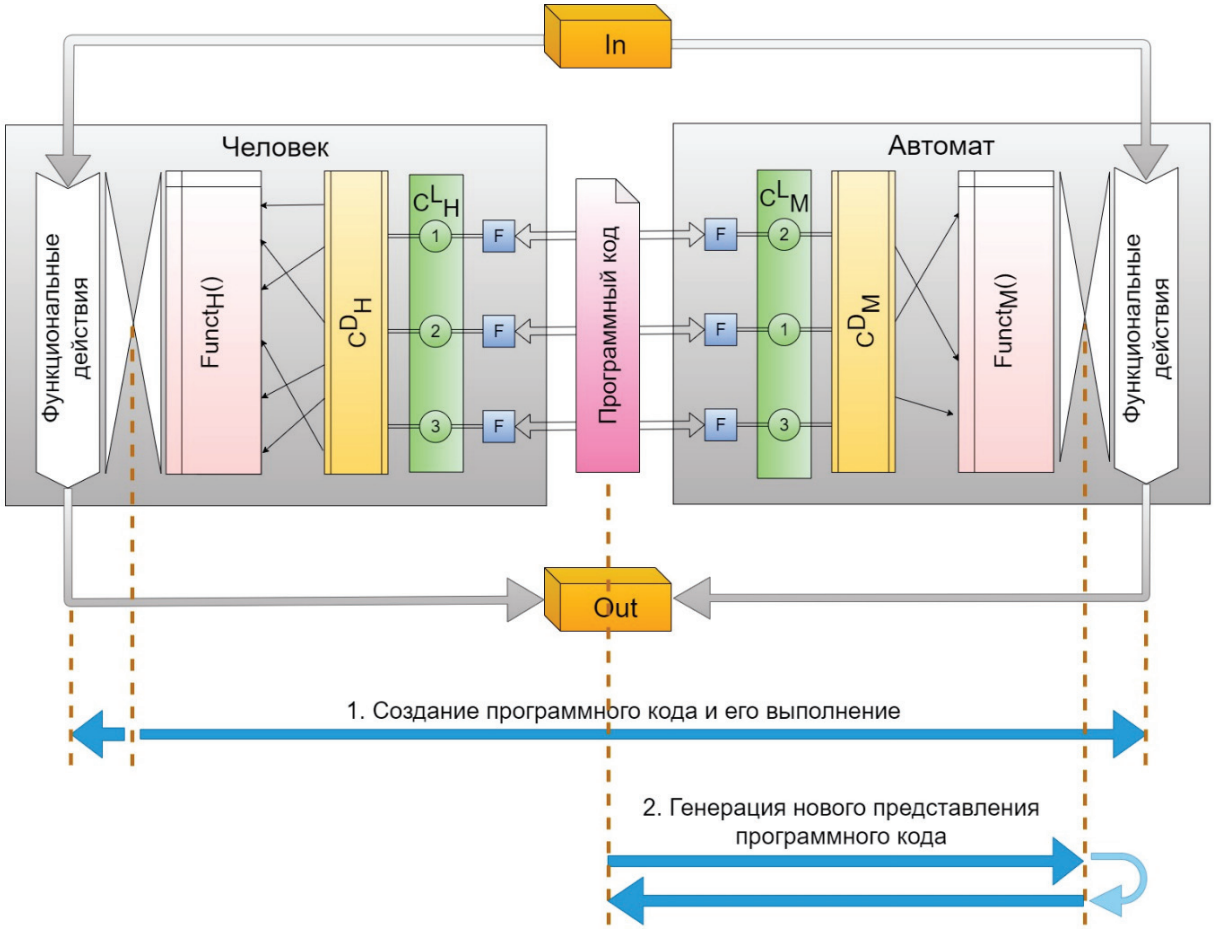

Рис. 1. Схема преобразования программного кода с уязвимостями (точки появления отмечены красными кружками)

$$Funct_S() \to \{C_S^D\} \to \{C_S^L\} \to \{F\} \to Q \to \\ \to \{F\} \to \{C_A^L\} \to \{C_A^D\} \to Funct_A(). \quad (18)$$

В процессе работы с программным кодом в некотором Представлении с целью модификации действия подобны описанным выше с тем отличием, что программный код сначала анализируется на предмет понимания его функционала, а затем (после внесения необходимых изменений и проверки на входных/выходных данных) производится синтез в Форму исходного представления:

$$Q \to \{F_S\} \to \{C_S^L\} \to \{C_S^D\} \to \\ \to Funct_S() \to Funct_S()' \to \\ \to \{C_A^D\}' \to \{C_A^L\}' \to \{F_A\}' \to Q', \quad (19)$$

где верхним индексом «′» отмечены элементы после модификации.

Если оба эти процесса преобразования произведены абсолютно корректно, то их Содержания кода совпадают:

$$C_S^D \cup C_S^L = C_A^D \cup C_A^L, \quad (20)$$

и обработка одинаковых входных данных (In) даст полностью одинаковый результат (Out); в ином случае можно говорить о появлении уязвимости, поскольку функционал нарушился.

Используя предложенную схему на Рис. 1, рассмотрим возможности появления уязвимостей (отмечены на схеме красными точками) при работе человека с исходным кодом, из которого компиляцией будет получен машинный код. Очевидно, что уязвимость может точечно появиться в каждом из шагов преобразования такого кода:

Точка 1. В $Funct_S()$ – человек сделал неверное функциональное описание кода, например, ошибившись в математической формуле.

Точка 2. В C_S^D – человек ошибся в смысле отдельных элементов словаря Содержания, например, в назначении математических операторов.

Точка 3. В C_S^L – человек неверно разложил формулу на элементы ее записи, например, просто перепутал знаки сложения и вычитания.

Точка 4. В F_S – человек допустил ошибку при кодировании, например, напечатал в программном коде одну переменную вместо другой.

Точка 5. В F_A – автомат допустил ошибку при разборе текста программного кода, например, воспринял деление (в языке C/C++ – «/»), как начало однострочного комментария (в языке C++ – «//»).

Точка 6. В C_A^L – автомат ошибся при генерации последовательности кода инструкций (как для их выполнения, так и для записи во внешний файл), например, пропустив инициализацию переменных нулевым значением.

Точка 7. В C_A^D – автомат неверно интерпретировал функционал отдельных инструкций, например, затерев используемые значения регистров результатом выполнения сторонней операции.

Точка 8. В $Funct_A()$ – автомат неверно воссоздал общую формулу вычисления результата работы с данными, например, сгенерировав код для работы с беззнаковыми числами, вместо знаковых.

Таким образом, уязвимости могут появиться в 8-ми точках преобразования программного кода в рамках одного Представления (см. схему на Рис. 1). А поскольку количество введенных авторами небезопасных Представлений равно 7-1=6 (в предположении, что Идея является «идеальной»), то общее количество таких точек появления составляет 6×8=48, что говорит об их когнитивном многообразии. При этом необходимо учитывать то, что в каждой точке могут появляться уязвимости различного типа.

Заключение

Изучение всех указанных аспектов позволило совместить в единой аналитической модели и описать зависимости фундаментальных понятий программного кода, таких как Представление, Форма, Содержание, Уязвимость, Функционал. Преобразования кода в рамках одного Представления являются однотипными и описываются единой схемой.

Работоспособность модели и схемы на приведенных примерах кода подтверждает их корректность.

Полученные результаты могут быть использованы в области безопасного программного обеспечения для формализации и изучения механизмов возникновения и эволюции уязвимостей в программном коде, что позволит создавать соответствующие правила их обнаружения и нейтрализации современными методами (в т.ч. машинного обучения).

Дальнейшее направление исследований авторы видят в получении такого уровня формализации, при котором модель могла бы использоваться не только для установления общего вида зависимостей между понятиями, а непосредственно и численно применяться в интересах обеспечения информационной безопасности кода: для формальной классификации уязвимостей кода и их локализации, умной кибербезопасности [26], построения аналитических моделей целых систем [27, 28] и оценки метрик их безопасности, риск-анализа защищаемых систем [29] и т.п.

Литература

1. Буйневич М.В., Васильева И.Н., Воробьев Т.М., Гниденко И.Г., Егорова И.В., Еникеева Л.А и др. Защита информации в компьютерных системах: монография. – СПб.: СПГЭУ, 2017. 163 с.
2. Бречко А.А., Вершенник Е.В., Закалкин П.В., Стародубцев Ю.И. Взгляды командований зарубежных государств на проведение операции в киберпространстве // Проблемы технического обеспечения войск в современных условиях: сборник трудов IV межвузовской научно-практической конференции (Санкт-Петербург, Российская Федерация, 6 февраля 2019 г.). 2019. с. 132-136.
3. Буйневич М.В., Израилов К.Е. Антропоморфический подход к описанию взаимодействия уязвимостей в программном коде. Часть 1. Типы взаимодействий // Защита информации. Инсайд. 2019. № 5(89). С. 78-85.
4. Буйневич М.В., Израилов К.Е. Антропоморфический подход к описанию взаимодействия уязвимостей в программном коде. Часть 2. Метрика уязвимостей // Защита информации. Инсайд. 2019. № 6(90). С. 61-65.
5. Кубарев А.В. Подход к формализации уязвимостей информационных систем на основе их классификационных признаков // Вопросы кибербезопасности. 2013. № 2(2). С. 29-33.
6. Грабежов И.Е., Леонов Ю.А. Формализация анализа уязвимостей информационной системы при проектировании КСЗИ // Вестник Воронежского государственного университета инженерных технологий. 2017. Т. 79. № 2(72). С. 107-112. DOI: 10.20914/2310-1202-2017-2-107-112
7. Вольфсон Ю.Р., Вольчина А.Е. Проблема классификации теорий информационного общества // Современные исследования социальных проблем (электронный научный журнал). 2017. Т. 8. № 3-1. С. 80-110. DOI: 10.12731/2218-7405-2017-3-80-110
8. Чернышева А.Ф., Трубин И.С., Корепанов А.Г., Репкин Д.А. Классификация угроз информационной безопасности в когнитивных сетях связи // Advanced Science. 2017. № 4. С. 37.
9. Грунина Л.П., Широколобова А.Г. Когнитивный аспект исследования терминов // Филологические науки. Вопросы теории и практики. 2010. № 1-1(5). С. 97-99.

10. Маджаева С.И. Термин инфекция в когнитивном аспекте // Современные проблемы лингвистики и лингводидактики: концепции и перспективы: материалы Пятой заочной Международной научно-методической конференции (Волгоград, Российская Федерация, 30 апреля 2015 г.). – Волгоград: Волгоградский государственный университет, 2015. С. 73-78.
11. Барабанов А.В., Марков А.С., Цирлов В.Л. Актуальные вопросы выявления уязвимостей и недекларированных возможностей в программном обеспечении // Системы высокой доступности. 2018. Т. 14. № 3. С. 12-17. DOI: 10.18127/j20729472-201803-03
12. Вялых А.С., Вялых С.А. Динамика уязвимостей в современных защищенных информационных системах // Вестник Воронежского государственного университета. Серия: Системный анализ и информационные технологии. 2011. № 2. С. 59-63.
13. Buinevich M., Izrailov K., Vladyko A. The life cycle of vulnerabilities in the representations of software for telecommunication devices // Proceedings of the 18th International Conference on Advanced Communications Technology (ICACT-2016, Pyeongchang, South Korea, 31th January - 03rd February). IEEE, 2016. PP. 430-435. DOI:10. 1109/ICACT.2016. 7423420
14. Buinevich M., Izrailov K., Vladyko A. Metric of vulnerability at the base of the life cycle of software representations // Proceedings of the 20th International Conference on Advanced Communication Technology (ICACT-2018, Chuncheon-si Gangwon-do, South Korea, 11-14 February 2018). IEEE, 2018. PP. 1-8. DOI: 10.23919/ICACT.2018.8323940.
15. Головин В.С. Общая теория уязвимостей компьютерных систем // Технические науки: традиции и инновации: сборник трудов III Международной научной конференции (Казань, Российская Федерация, 20-23 марта 2018 г.). – Казань: Молодой ученый, 2018. С. 10-12.
16. Буйневич М.В., Израилов К.Е., Щербаков О.В. Модель машинного кода, специализированная для поиска уязвимостей // Вестник Воронежского института ГПС МЧС России. 2014. № 2(11). С. 46-51.
17. Марков А.С., Фадин А.А., Швец В.В. Сравнение статических анализаторов безопасности программного кода // Защита информации. Инсайд. 2015. № 6(66). С. 38-43.
18. Израилов К.Е. Архитектурные уязвимости программного обеспечения // Шестой научный конгресс студентов и аспирантов СПбГИЭУ (ИНЖЭКОН-2013): сборник тезисов докладов научно-практической конференции факультета информационных систем и экономике и управлении «Инфокоммуникационные технологии и математические методы» (Санкт-Петербург, Российская Федерация, 18-19 апреля 2013 г.). – СПб.: Санкт-Петербургский Государственный Экономический университет, 2013. С. 35.
19. Израилов К.Е., Татарникова И.М. Подход к анализу безопасности программного кода с позиции его формы и содержания // Актуальные проблемы инфотелекоммуникаций в науке и образовании (АПИНО 2019): сборник научных статей VIII Международной научно-технической и научно-методической конференция (Санкт-Петербург, Российская Федерация, 27-28 февраля 2019 г.). – СПб.: СПбГУТ, 2019. С. 462-467.
20. Пустыгин А.Н., Язов Ю.К., Машин О.А., Зубов М.В. К вопросу об автоматическом комментировании на естественном языке исходных текстов программ // Программная инженерия. 2013. № 11. С. 17-21.
21. Марков А.С., Шеремет И.А. Безопасность программного обеспечения в контексте стратегической стабильности // Вестник академии военных наук. 2019. № 2(67). С. 82-90.
22. Бородин А.А. Энтропия и применение энтропии для анализа систем // Вестник Российского нового университета. Серия: Сложные системы: модели, анализ и управление. 2010. № 3. С. 33-36.
23. Чепкасов В.Л., Михайлова Т.Л. Новые смыслы понятия энтропии, или к вопросу о неклассическом варианте понятия энтропии // Международный журнал экспериментального образования. 2014. № 6-1. С. 164-167.
24. Мержвинский А.А. Теория информации - базовое ядро теории информатики // Труды Северо-Кавказского филиала Московского технического университета связи и информатики. 2018. № 1. С. 225-234.
25. Троицкий И.И., Басараб М.А., Глинская Е.В. Определение зависимости количества информации от числа градаций дискретного сигнала в канале связи с равномерным шумом // Радиотехника. 2019. Т. 83. № 10(15). С. 82-86. DOI: 10.18127/ j00338486-201910(15)-13
26. Петренко С.А., Бирюков Д.Н., Петренко А.С. Умная кибербезопасность // The 2019 Symposium on Cybersecurity of the Digital Economy (CDE'19): сборник статей третьей международной научно-технической конференции (Казань, Российская Федерация, 22-24 мая 2019 г.). – СПб.: ООО «Издательский Дом «Афина», 2019. С. 160-172.
27. Калашников А.О., Бугайский К.А., Аникина Е.В. Модели количественного оценивания компьютерных атак // Информация и безопасность. 2019. Т. 22. № 4(4). С. 517-528.
28. Калашников А.О., Бугайский К.А., Аникина Е.В. Модели количественного оценивания компьютерных атак (часть 2) // Информация и безопасность. 2019. Т. 22. № 4(4). С. 529-538.
29. Пахомова А.С., Рахманин Д.Н., Паринова Л.В., Язов Ю.К. Оценка применимости методики CVSS для риск-анализа защищаемых систем // Информация и безопасность. 2017. Т. 20. № 1. С. 129-132.

Рецензент: *Ловцов Дмитрий Анатольевич, доктор технических наук, профессор, заслуженный деятель науки Российской Федерации, заместитель по научной работе директора Института точной механики и вычислительной техники им. С.А. Лебедева Российской академии наук, заведующий кафедрой информационного права, информатики и математики Российского государственного университета правосудия, г. Москва, Россия. E-mail: dal-1206@mail.ru*

ANALYTICAL MODELING OF THE VULNERABLE PROGRAM CODE EXECUTION

Buinevich M.V.[4], *Izrailov K.E.*[5]

Abstract

The goal of the presented investigation is to create an analytical model of vulnerable program code. This will allow one to analyze the characteristics of the code in various representations: Idea, Conceptual model, Architecture, Algorithms, Source code, Assembler code, Machine code.

The research method consists in analysis of the program code during its full lifecycle in the 4 basic aspects:

1) dynamic – describes changes in the process of developing program code and its vulnerabilities;

2) formalizing – describes analytical rules for representing of a code, vulnerabilities, their mutual influence, dynamics of changes and other concepts of the subject area;

3) classification – introduces classes of vulnerabilities that depend on the program code representation chronology;

4) cognitive – describes the mechanism of a code perception by a man or by a tool in a formal way.

The main scientific result of the investigation is an analytical model of software life cycle. The model combines in a formal way the fundamental concepts of program code: Representation, Form, Content, Vulnerability, Functionality. The particular scientific result is a scheme for vulnerable code transformation into the new representation. The scheme describes in detail the transformation of the code to a new representation and the modification of the code in the current one.

The obtained results are a powerful investigation tool in the area of secure software. These results can be used to formalize and investigate the mechanisms of the appearance and evolution of vulnerabilities in a program code. This will allow one to create the detection and neutralization rules for such vulnerabilities using modern methods (including machine learning).

Keywords: *cybersecurity, development model, program code representations, program analysis, formalization, classification, code dynamics, cognition.*

References

1. Buynevich M.V., Vasileva I.N., Vorobev T.M., Gnidenko I.G., Yegorova I.V., Yenikeeva L.A. et al. *Zashchita informatsii v kompyuternykh sistemakh* [Information Protection in Computer Systems]. – St. Petersburg: Saint-Petersburg State University of Economics Publ.; 2017. 163 p. (in Russ.)
2. Brechko A.A., Vershennik Ye.V., Zakalkin P.V., Starodubtsev Yu.I. Vzglyady komandovaniy zarubezhnykh gosudarstv na provedeniye operatsii v kiberprostranstve [The Views of the Commands of Foreign States on Operations in Cyberspace]. *Problemy tekhnicheskogo obespecheniya voysk v sovremennykh usloviyakh: sbornik trudov IV mezhvuzovskoy nauchno-prakticheskoy konferentsii, 6th February 2019, St. Petersburg, Russian Federation* [In Proceedings of the IVth Interuniversity Scientific and Practical Conference on Problems of Technical Support of Troops in Modern Conditions, 6th February 2019, St. Petersburg, Russian Federation]. - St. Petersburg: Military Academy of Communications Publ.; 2019. p.132-136. (in Russ.)
3. Buynevich M.V., Izrailov K.E. Anthropomorphic Approach to Description of the Vulnerabilities Interaction in Program Code. Part 1. Types of Interactions. *Zaŝita informacii. Inside*. 2019;5(89):78-85. (in Russ.)
4. Buynevich M.V., Izrailov K.E. Anthropomorphic Approach to Description of the Vulnerabilities Interaction in Program Code. Pt. 2. Metric of Vulnerabilities. *Zaŝita informacii. Inside*. 2019;6(90):61-65. (in Russ.)
5. Kubarev A. The Classification Characteristics Based Approach to Formalization of Information Systems Vulnerabilities. *Voprosy Kiberbezopasnosti*. 2013;2(2):29-33. (in Russ.)
6. Grabezhov I.E., Leonov Ju.A. Formalization of the analysis of the vulnerabilities of the information system in the design of KSZI. *Proceedings of the Voronezh State University of Engineering Technologies*. 2017;79(2):107-112. (in Russ.) DOI: 10.20914/2310-1202-2017-2-107-112
7. Wolfson Yu.R., Volchina A.E. The Difficulty of Information Society Theories Classification. *Sovremennye issledovaniya sotsialnykh problem (elektronnyy nauchnyy zhurnal)*. 2017;8(3-1):80-110. (in Russ.) DOI: 10.12731/2218-7405-2017-3-80-110
8. Chernysheva A.F., Trubin I.S., Korepanov A.G., Repkin D.A. Klassifikatsiya ugroz informatsionnoy bezopasnosti v kognitivnykh setyakh svyazi [Classification of Information Security Threats in Cognitive Communication Networks]. *Advanced Science*. 2017;4:37. (in Russ.)

4 Mikhail Buinevich, Dr.Sc., Professor, Professor of Dep. Applied Mathematics and Information Technologies of Saint-Petersburg University of State Fire Service of EMERCOM of Russia, Professor of Dep. Information Systems Security of The Bonch-Bruevich Saint-Petersburg State University of Telecommunications, ORCHID: https://orcid.org/0000-0001-8146-0022. Scopus Author ID: 56122749800. Russia, Saint-Petersburg. E-mail: bmv1958@yandex.ru

5 Konstantin Izrailov, Ph.D., Associate Professor of Dep. Secured Communication Systems of The Bonch-Bruevich Saint-Petersburg State University of Telecommunications, Senior Researcher of Laboratory of Computer Security Problems of St. Petersburg Institute for Informatics and Automation of the Russian Academy of Sciences, ORCHID: https://orcid.org/0000-0002-9412-5693. Scopus Author ID: 56123238800. Russia, Saint-Petersburg. E-mail: konstantin.izrailov@mail.ru

9. Grunina L.P., Shirokolobova A.G. Cognitive Aspect of Terms Research. *Philological Sciences. Issues of Theory and Practice*. 2010;1-1(5):97-99. (in Russ.)
10. Madzhaeva S.I. The Term Infection in Cognitive Aspect. *Sovremennye problemy lingvistiki i lingvodidaktiki: kontseptsii i perspektivy: materialy Pyatoy zaochnoy Mezhdunarodnoy nauchno-metodicheskoy konferentsii, 30th April 2015, Volgograd, Russian Federation* [In Proceedings of the Vth Correspondence International Scientific and Methodological Conference on Modern Problems of Linguistics and Linguodidactics: Concepts and Perspectives, 30th April 2015, Volgograd, Russian Federation]. - Volgograd: Volgograd State University Publ.; 2015. p. 73-78. (in Russ.)
11. Barabanov A.V., Markov A.S., Tsirlov V.L. Topical issues of identifying vulnerabilities and undeclared capabilities in software. *Sistyemy vysokoy dostupnosti*. 2018;14(3):12-17. (in Russ.) DOI: 10.18127/j20729472-201803-03
12. Vyalykh A.S., Vyalykh S.A. The Dynamics of Vulnerability in Modern Protected Information Systems. *Proceedings of Voronezh State University. Series: Systems analysis and information technologies*. 2011;2:59-63. (in Russ.)
13. Buinevich M., Izrailov K., Vladyko A. The life cycle of vulnerabilities in the representations of software for telecommunication devices. *In Proceedings of the 18th International Conference on Advanced Communications Technology, ICACT-2016, Pyeongchang, South Korea, 31th January - 03rd February 2016*. IEEE; 2016. p.430-435. DOI: 10.1109/ICACT.2016.7423420
14. Buinevich M., Izrailov K., Vladyko A. Metric of vulnerability at the base of the life cycle of software representations. *In Proceedings of the 20th International Conference on Advanced Communication Technology, ICACT-2018, 11-14 February 2018, Chuncheon-si Gangwon-do, South Korea*. IEEE; 2018. p.1-8. DOI: 10.23919/ICACT.2018.8323940
15. Golovin V.S. Obshchaya teoriya uyazvimostey kompyuternykh system [General Theory of Vulnerabilities in Computer Systems]. *Tekhnicheskie nauki: traditsii i innovatsii: sbornik trudov III Mezhdunarodnoy nauchnoy konferentsii, 20-23 March 2018, Kazan, Russian Federation* [In Proceedings of the IIIrd International Scientific Conference on Technical Sciences: Traditions and Innovations, 20-23 March 2018, Kazan, Russian Federation]. - Kazan: Molodoy uchenyy Publ.; 2018. p. 10-12. (in Russ.)
16. Buinevich M.V., Izrailov K.E., Scherbakov O.V. Model of Machine Code Specialized for Vulnerabilities Search. *Vestnik Voronezhskogo instituta GPS MChS Rossii*. 2014;2(11): 46-51. (in Russ.)
17. Markov A.S., Fadin A.A., Shvets V.V. Comparison of static security code analyzers. *Zaŝita informacii. Inside*. 2015;6(66):38-43. (in Russ.)
18. Izrailov K.E. Arkhitekturnye uyazvimosti programmnogo obespecheniya [Architectural Software Vulnerabilities]. *Shestoy nauchnyy kongress studentov i aspirantov SPbGIEU (INZhEKON-2013): sbornik tezisov dokladov nauchno-prakticheskoy konferentsii fakulteta informatsionnykh sistem i ekonomike i upravlenii «Infokommunikatsionnye tekhnologii i matematicheskie metody», 18-19 April 2013, St. Petersburg, Russian Federation* [In Proceedings of the Sixth Scientific Congress of Undergraduate and Postgraduate Students of Saint Petersburg State University of Economics: Scientific and Practical Conference of the Faculty of Information Systems and Economics and Management “Infocommunication Technologies and Mathematical Methods”, 18-19 April 2013, St. Petersburg, Russian Federation]. - St. Petersburg: Saint Petersburg State University of Economics Publ.; 2013. p. 35. (in Russ.)
19. Izrailov K., Tatarnikova I. An Approach to Analyzing the Security of a Software Code from the Standpoint of its Form and Content. *In Proceedings of the VIIIth International Conference on Infotelecommunications in Science and Education, 27–28 February 2019, St. Petersburg, Russian Federation)*. St. Petersburg: The Bonch-Bruevich Saint-Petersburg State University of Telecommunications Publ.; 2019. p. 462-467. (in Russ.)
20. Pustigin A.N., Yazov Y.K., Mashin O.A., Zubov M.V. To the Question on Automatic Commenting in the Natural Language of Initial Texts Programs. *Software Engineering*. 2013;11:17-21. (in Russ.)
21. Markov A.S., Sheremet I.A. Software Safety in the Context of Strategic Stability. *Vestnik akademii voennykh nauk*. 2019;2(67):82-90. (in Russ.)
22. Borodin A.A. Entropiya i primenenie entropii dlya analiza system [Entropy and the Use of Entropy for Systems Analysis]. *Vestnik of Russian New University. Series: Complex Systems: Models, Analysis and Management*. 2010;3:33-36. (in Russ.)
23. Chepkasov V.L., Mikhaylova T.L. Novye smysly ponyatiya entropii, ili k voprosu o neklassicheskom variante ponyatiya entropii [New meanings of the concept of entropy, or to the question of a non-classical version of the concept of entropy]. *Mezhdunarodnyy zhurnal eksperimentalnogo obrazovaniya*. 2014;6-1:164-167. (in Russ.)
24. Merzhvinskiy A.A. Teoriya informatsii - bazovoe yadro teorii informatiki [Information theory - the basic core of computer science theory]. *Trudy Severo-Kavkazskogo filiala Moskovskogo tekhnicheskogo universiteta svyazi i informatiki*. 2018;1:225-234. (in Russ.)
25. Troitskiy I.I., Basarab M.A., Glinskaya E.V. Determination of the amount of information for a discrete signal with K values in the communication channel with uniform noise. *Radioengineering*. 2019;83(10):82-86. (in Russ.) DOI: 10.18127/j00338486-201910(15)-13
26. Petrenko S.A., Biryukov D.N., Petrenko A.S. Umnaya kiberbezopasnost [Smart cybersecurity]. *In Proceedings of the IIIrd International Scientific and Technical Conference on Cybersecurity of the Digital Economy, CDE’19, 22-24 May 2019, Kazan, Russian Federation*. - St. Petersburg: Afina Publ.; 2019. p.160-172. (in Russ.)
27. Kalashnikov A.O., Bugayskiy K.A., Anikina E.V. Models of Quantitative Estimation of Computer Attacks (Part 1). *Informatsiya i bezopasnost*. 2019;22(4):517-528. (in Russ.)
28. Kalashnikov A.O., Bugayskiy K.A., Anikina E.V. Models of Quantitative Estimation of Computer Attacks (Part 2). *Informatsiya i bezopasnost*. 2019;22(4):529-538. (in Russ.)
29. Pakhomova A.S., Rakhmanin D.N., Parinova L.V., Yazov Yu.K. Usability Assesment of CVSS version 2 vulnerability scoring to risk-analysis of trusted systems. *Informatsiya i bezopasnost.* 2017;20(1):129-132. (in Russ.)