DOI: 10.18698/0236-3933-2016-1-98-111

УДК 004.056

ЭВРИСТИЧЕСКИЙ АНАЛИЗ БЕЗОПАСНОСТИ ПРОГРАММНОГО КОДА

А.С. Марков[1], **В.А. Матвеев**[1], **А.А. Фадин**[2], **В.Л. Цирлов**[1]

[1]МГТУ им. Н.Э. Баумана, Москва, Российская Федерация
e-mail: a.markov@bmstu.ru; v.a.matveev@bmstu.ru; v.tsirlov@bmstu.ru

[2]НПО "Эшелон", Москва, Российская Федерация
e-mail: af@cnpo.ru

Рассмотрены структурный статический анализ безопасности программного кода и решение задачи обеспечения полноты проводимых проверок. Для реализации полноты проверок выявления уязвимостей программного кода обосновано использование эвристического (сигнатурного) анализа безопасности программ, учитывающего полный спектр классов дефектов программ. Разработана семантическая метамодель описания эвристических алгоритмов выявления уязвимостей и дефектов безопасности программ на различных уровнях представления программного кода. Отмечено, что наиболее актуальными моделями представления программ с позиции безопасности являются абстрактное синтаксическое дерево и абстрактный синтаксический граф. Необходимый уровень быстродействия, формальная простота и наглядность реализации эвристического анализа могут быть достигнуты применением системы продукционных правил. Приведены примеры частных семантических моделей эвристик выявления актуальных классов дефектов безопасности программ, а также достоинства и ограничения предложенных решений. Представлены сведения о практической реализации и апробации предлагаемых решений. Отмечено, что в практике сертификационных испытаний средств защиты информации 88 % критических уязвимостей было идентифицировано путем применения эвристического анализа. Сделан вывод, что эвристический анализ может служить базой при использовании различных техник аудита безопасности программного кода.

***Ключевые слова*:** информационная безопасность, безопасность программ, тестирование, статический анализ, продукционные модели, эвристический анализ, уязвимости, дефекты, недекларированные возможности.

HEURISTIC ANALYSIS OF SOURCE CODE SECURITY

A.S. Markov[1], **V.A. Matveev**[1], **A.A. Fadin**[2], **V.L. Tsirlov**[1]

[1]Bauman Moscow State Technical University, Moscow, Russian Federation
e-mail: a.markov@bmstu.ru; v.a.matveev@bmstu.ru; v.tsirlov@bmstu.ru

[2]Scientific Production Association Echelon, Moscow, Russian Federation
e-mail: af@cnpo.ru

The paper is devoted to the structural static analysis of source code security and the problem of ensuring the conducted inspection completeness. To detect comprehensively the source code vulnerabilities the heuristic (signature) analysis of program security is used taking into account the whole range of program defects classes. The semantic metamodel of the heuristic algorithms description to detect

defects and vulnerabilities in the program security at different levels of source code representation is developed. It has been noted that the most essential program representation models for computer security are an abstract syntax tree and abstract syntax graph. The required processing speed, formal simplicity and implementation visibility of the heuristic analysis can be achieved by using a production rule system. The examples of particular semantic models for heuristics of detecting the essential program security defects as well as their advantages and limitations are given. The information about the practical implementation and approbation of the proposed solutions is provided. It was noted that 88 % of critical vulnerabilities were indentified by using the heuristic analysis during certification tests of information security facilities. The conclusion was made that the heuristic analysis can be used as a base in various techniques of source code security audit.

***Keywords*:** information security, software security, testing, static analysis, production models, heuristic analysis, vulnerabilities, defects, undeclared capabilities.

Введение. При проведении испытаний программных систем по требованиям безопасности информации испытательные лаборатории сталкиваются с задачей принципиального характера — подтверждение полноты выполненных проверок. Решение этой задачи не является тривиальным вследствие использования различных техник тестирования, ориентированных на определенные классы уязвимостей. Так, применение традиционных методик fuzz-тестирования малоэффективно при выявлении уязвимостей, связанных с редкими сочетаниями входных данных, например, программных закладок [1, 2]. Прикладные методы верификации программ, основанные на алгебраических моделях, ограничены узким классом уязвимостей, как нефункциональными ошибками (некорректностями кодирования) [3–8]. Отечественная нормативная база также предусматривает декомпозиционный разбор программ в целях контроля отсутствия недекларированных возможностей, сводящийся, фактически, к идентификации лишь “мертвого кода” [9].

В настоящей работе предложен прикладной метод эвристического анализа безопасности программ, обеспечивающий полноту и непротиворечивость проверок за счет поддержки известных реестров дефектов безопасности кода.

Следует отметить востребованность тематики, так как требования к обязательности структурного анализа безопасности программного кода определены популярной методологией Common Criteria для высших оценочных уровней доверия [10, 11] и стандартами PCI DSS, PA-DSS, NISTIR 4909. Подтверждением этого также можно считать инициирование нового международного проекта в области статического анализа SATEC [12].

Введение в проблему безопасности кода. В настоящее время критическая сложность программных продуктов не позволяет выполнять проверки безопасности кода без участия эксперта. Основная задача автоматизации поиска уязвимостей — предоставление эксперту фрагментов кода, с максимальной степенью достоверности содержащего

Рис. 1. Схема, иллюстрирующая факторы безопасности информации в рамках жизненного цикла программ

уязвимость конкретного класса. В таком случае статический анализатор манипулирует собственно не с уязвимостями (vulnerability), а с дефектами безопасности кода, под которым понимают недостаток создания программы, потенциально влияющий на степень безопасности информации. Эксплуатируемый дефект безопасности представляет собой уязвимость, реализация которой на объекте информатизации составляет угрозу информационной безопасности (ИБ) для конкретного ресурса (рис. 1).

Под эвристическим анализом будем понимать поиск программных дефектов в программном коде путем сопоставления фрагментов кода с образцами из базы данных шаблонов (эвристических правил) дефектов безопасности.

Объектами исследования при анализе, кроме исходных текстов программного обеспечения (ПО), могут быть объектный и исполняемый программные коды, проектная, компиляционная, отладочная и компоновочная информация. Анализ может быть (в зависимости от уровня унификации) выполнен на различных моделях представления кода, например, на лексическом коде, синтаксическом дереве, абстрактном синтаксическом дереве (дереве Канторовича), графе потока управления, графе потока данных, модели однократного статического присваивания, абстрактном семантическом графе.

Полнота проверок может быть достигнута путем привязки базы данных шаблонов к актуальным реестрам дефектов программ (CWE, HP Fortify, DoD SFP, WASC-TC и др.) [13].

Сопоставление методов статического анализа — методов прикладной верификации (проверки свойств) и метода эвристического анализа — на наиболее популярных уровнях представления кода приведено в табл. 1.

Повышение качества работы статического анализа кода связано со снижением числа ложных срабатываний при сохранении максимальной полноты обнаруживаемых классов потенциально опасных конструкций [14–16]. Аппарат, описывающий шаблоны (сигнатуры, эвристики) дефектов кода, должен обеспечивать максимально гибкое

Таблица 1

Сравнение методов поиска уязвимостей

Представление кода	Проверка свойств	Поиск дефектов по шаблону
Исходные тексты	Свойства недоступны	Поиск сигнатур по регулярным выражениям
Абстрактное синтаксическое дерево	Лексический и синтаксический анализ	Лексический и синтаксический анализ, поиск сигнатур по дереву
Абстрактный семантический граф потока данных	Абстрактная интерпретация (интервальный анализ, анализ указателей, анализ зависимостей по данным)	Анализ потока данных, поиск внутрипроцедурных сигнатур последовательностей инструкций с учетом передаваемых значений

определение дефекта с учетом разнообразия синтаксиса исследуемого языка программирования, что и представляет основную задачу исследования.

Разработка метамодели выявления безопасности дефектов кода. Для обеспечения простоты представления метамодели используем продукционную модель знаний. Это связано с тем, что для отдельного языка программирования в настоящее время известно не более 100 классов критических дефектов. В качестве системы продукций предложим систему вида

$$\begin{cases} R_1 =< s_j \in S;\ CP_1(s_j);\ DA_1(s_j) \to DB_1(s_j);\ P_1(CL_1, PR_1) >; \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ R_n =< s_j \in S;\ CP_n(s_j); DA_n(s_j) \to DB_n(s_j); P_n(CL_n, PR_n) > . \end{cases}$$

Здесь n — число продукционных правил в модели, связанных с анализом кода; R_i — обозначение i-й продукции модели, $i = \overline{1,n}$; $s_j \in S$ — описание класса ситуаций (в данной предметной области речь идет о проверке способности анализатора обрабатывать пришедшие входные данные); S — множества всех лексем исходных текстов программного проекта, которые способен обрабатывать данный анализатор (ограничение определяется типом языка программирования, версией его грамматики, видом поддерживаемых сигнатур для анализа); $j = \overline{1,k}$ — индекс лексемы в векторе лексем $\overline{s}$, которая находится в данный момент в обработке статическим анализатором, $k = |\,\overline{s}\,|$; $\overline{s}$ — вектор лексем узлов абстрактного синтаксического дерева кода, полученного в результате работы препроцессора, лексического и синтаксического анализа; $CP(s_j)$ — условие активации продукции, функция, возвращающая “1” при наличии заданных в аргументе типовых конструкций кода программного проекта, где наиболее вероятно возникновение программного дефекта (заданного типа) или присутствует важное для анализа свойство программы (принадлежность

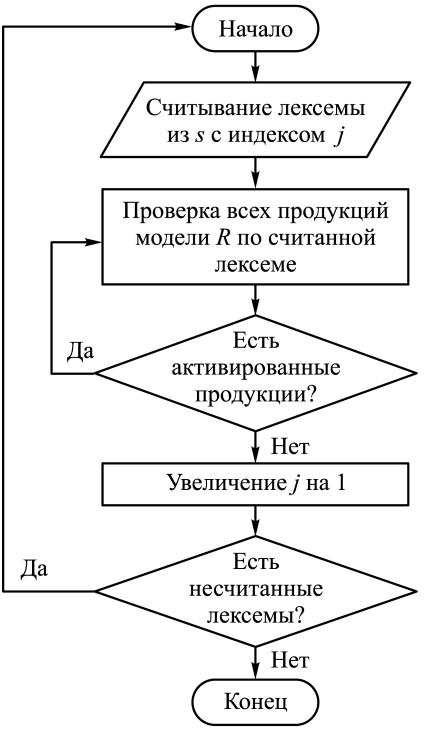

Рис. 2. Блок-схема функционирования системы продукционных моделей

функционального объекта к тому или иному виду, константное или статистически предсказуемое значение и т.п.); $DA(s_j) \rightarrow DB(s_j)$ — выражение, так называемое ядро продукции с продукционным выводом о дефекте кода или дополнительных свойствах кода на основе существования типовых синтаксических конструкций в коде, определяющих либо возможность реализации дефекта, либо наличие свойств кода, косвенно влияющих на возникновение дефекта (вывод может быть промежуточным или заключительным); $P(CL, PR)$ — функция, вызываемая по достижению логического вывода о существовании вероятного дефекта кода, имеющая следующие параметры: CL — категория (тип) дефекта (задается экспертом в области аудита кода при формировании продукционной модели, представляет собой числовой номер типа дефекта в международных классификациях дефектов); PR — степень критичности данного дефекта, задаваемая экспертом.

Параметрами перечисленных выше функций являются различные виды функциональных и информационных объектов, полученных из структурного представления кода.

Методика функционирования предложенной системы продукционных моделей включает в себя следующие шаги (рис. 2).

1. Считывание лексемы s_j (изначально j имеет индекс первого элемента кортежа лексем).

2. Последовательная $(\overline{1,n})$ проверка всех продукций модели.

3. Если ни одна из продукций модели не была активирована на шаге 2, то j увеличиваем на 1 и переходим к шагу 2.

4. Если во входном потоке больше нет лексем, то завершаем работу, в противном случае переходим к шагу 1.

Рассмотренная система продукций является многоуровневой метамоделью описания эвристик поиска конкретных уязвимостей, что продемонстрировано далее.

Методика формализации эвристик для различных классов дефектов кода. При описании конкретных эвристик приходится отталкиваться от конкретных классов дефектов, конкретной программной

языковой среды и уровня представления кода программ. Поскольку детальное описание всевозможных эвристик связано с весьма большим объемом и со сложностью восприятия, ограничимся демонстрацией нескольких примеров. Введем следующие определения и обозначения.

Функциональный объект — элемент программной системы, осуществляющий выполнение действий по реализации законченного фрагмента алгоритма работы программной системы. В качестве функциональных объектов могут выступать функции, процедуры, методы классов и объектов.

Множество функциональных объектов API — множество функциональных объектов (ФО) как стандартных библиотек среды программирования, операционной системы, так и внешних компонентов (библиотеки, фреймворки), обеспечивающих реализацию заданного типа функционала.

Кортеж допустимых типов объектов в ПО (процедурного и объектного типа) определим как $M =< M_{const}, M_{func}, M_{info}, M_{obj}, I_{det} >$, где M_{const} — множество допустимых постоянных значений в ПО (литералы, константы, макросы и т.д.); M_{func} — множество ФО, определенных в программной системе; M_{info} — множество информационных объектов (ИО), определенных в программной системе; M_{obj} — множество объектов (классов), определенных в программе; I_{det} — множество ИО, имеющих статистически предсказуемое значение (системное время, идентификаторы процессов, а также любые константы).

Кортеж допустимых типов используемых внешних ФО (функций API) в ПО зададим следующим образом: $F =< F_{prng}, F_{sql} >$, где F_{prng} — множество функций API, реализующих инициализацию генераторов псевдослучайных чисел (ГПСЧ); F_{sql} — множество функций API, реализующих выполнение SQL-запросов.

Кортеж допустимых типов операндов в ПО имеет вид $o = =< o_{coint}, o_{call}, o_{ret}, o_{asg}(V, R) >$, где o_{coint} — множество операторов и функций, реализующих целочисленные преобразования объектов; o_{call} — множество операторов и функций, реализующих передачу управления ФО; o_{ret} — множество операторов и функций, реализующих возврат управления из ФО; $o_{asg}(V, R)$ — множество операторов и функций, реализующих присваивание значений множества ИО V объекту R.

Кортеж допустимых функций над структурным представлением кода: $D =< f_{inside}(b), f_{call}(b), t_{obj}(b) >$, где $f_{inside}(b)$ — функция, возвращающая множество функциональных и информационных объектов, находящихся внутри объекта b; $f_{call}(b)$ — функция, возвращающая кортеж аргументов, которые используются для запуска либо ФО, либо инициализации ИО, в зависимости от типа объекта b; $t_{obj}(b)$ — функция, возвращающая тип объекта b, если он был задан ранее в рамках

исходных текстов программы, если на данном этапе анализа тип не определен (например, используется язык программирования с динамической типизацией), то она возвращает пустое множество.

Определим множество L лексем, в которых были найдены потенциально опасные конструкции (это множество заполняется в процессе работы продукционной модели) и функцию $AP(b, e)$, добавляющую элемент b в множество e.

Далее составим на основе перечисленных множеств правила продукционного вывода для дефектов различного типа.

Примеры реализации эвристического анализа на основе продукционной модели. Для выявления потенциально опасного фрагмента воспользуемся внутрипроцедурным уровнем представления кода (абстрактным синтаксическим деревом). Примеры дефектов безопасности кодов приведены в табл. 2.

Таблица 2

Примеры дефектов безопасности кода

Элемент CWE уровня		Признаки наличия	Подверженные языки и технологии программирования
верхнего	нижнего		
Использование недостаточно случайных величин (Use of Insufficiently Random Values), CWE-330	Предсказуемый вектор инициализации ГПСЧ (Predictable Seed in PRNG), CWE-337	Использование в качестве вектора инициализации констант или предсказуемых значений	Не зависит от языка программирования
Неверное следование спецификациям вызывающей стороной (Improper Following of Specification by Caller), CWE-573	Нарушение объектной модели: определен лишь один Equals или один Hashcode (Object Model Violation: Just One of Equals and Hashcode Defined), CWE-581	Переопределение одного из методов Equals или Hashcode без определения парного ему метода	Java
Ошибки условий, возвращаемых значений, статус кодов (Error Conditions, Return Values, Status Codes), CWE-389	Оператор возврата в блоке Finally (Return Inside Finally Block), CWE-584	Наличие оператора возврата в блоке Finally	Java

Пример дефекта кода, заключающегося в предсказании случайного числа. Рассмотрим дефект кода CWE-337, определяемый по классификации CWE как предсказуемый вектор инициализации ГПСЧ

(Predictable Seed in PRNG). Если вектор, используемый в программе для инициализации ГПСЧ, предсказуем (он формируется на основе системного времени, идентификатора процесса и других легко прогнозируемых параметров), то злоумышленник имеет возможность с высокой степенью вероятности "угадать" значения, выдаваемые ГПСЧ. В зависимости от их дальнейшего использования это может позволить реализовать атаки на сессионные ключи, хеши паролей, гамма-последовательности шифрования, подмену временных файлов и др.

Для выявления дефекта можно составить следующий эвристический алгоритм:

1) поиск функций установки вектора инициализации ГПСЧ;

2) анализ вектора инициализации на предмет того, используется ли для его инициализации константа и (или) статистически предсказуемое значение (например, день недели) с учетом того, что они могли быть определены ранее;

3) если вектор инициализации формируется только из этих видов данных, то, как правило, в программе присутствует дефект CWE-337.

Продукционная модель дефекта (CWE-337) будет иметь вид

$$\begin{cases} R_1^{337} =< s_j \in S;\ s_j \in o_{asg}\left(s_{j-1}, Y\right); \\ \qquad\qquad (Y \cap I_{det} \neq \emptyset) \rightarrow AP(s_{j-1}, I_{\text{det}}) > \\ R_2^{337} =< s_j \in S;\ s_j \in M_{func} \wedge s_{j+1} \in \\ \qquad\qquad \in o_{call}\left(z\right) \wedge z \in F_{prng};\ f_{arg} \cap I_{det} \neq \emptyset) \rightarrow \\ AP\left(s_{j+1}, L\right);\ P_2\left(337, 6\right). \end{cases}$$

Рассмотрим первое правило R_1^{337} построенной продукционной модели. Областью кода программного проекта, где наиболее вероятно возникновение потенциально опасной конструкции (ПОК), является любой оператор (функция), реализующий присваивание операнду x одного из значений множества ИО Y: $CP_1\left(S, N, M, F, o, D, L\right) = = CP_1\left(o\right) = N_{ij} \in o_{asg}\left(x, Y\right)$.

Ядро первой продукции рассматриваемой эвристики имеет следующий содержательный вид: "**если** ИО присваивается значение одного из элементов множества статически предсказуемых значений, **то** этот объект имеет статически предсказуемое значение".

Рассмотрим второе правило R_2^{337} построенной продукционной модели. Областью кода программного проекта, где наиболее вероятно возникновение ПОК, является любой оператор (функция), реализующий передачу управления ФО, в рассматриваемом случае — любой оператор (функция), вызывающий функцию API, которая реализует инициализацию ГПСЧ.

Ядро второй продукции рассматриваемой эвристики имеет вид: "**если** функция API, реализующая инициализацию ГПСЧ, имеет предсказуемые аргументы, **то** это потенциально опасная конструкция".

Постусловие продукционного правила предполагает формулирование вывода о ПОК (дефекте) с номером 337.

Пример дефекта кода, заключающегося в нарушении объектной модели. Рассмотрим дефект кода CWE-581, который состоит в нарушении объектной модели. Дефект заключается в том, что в программе определен лишь один метод Equals или один метод Hashcode (Object Model Violation: Just One of Equals and Hashcode Defined). Это обусловлено тем, что в языке Java от объектов ожидается несколько постоянных свойств, связанных с проверкой их идентичности. Одно из таких свойств — идентичные объекты должны иметь одинаковые хеши. Другими словами, должна выполняться следующая проверка: $if a.equals(b) == truethena.hashCode() == b.hashCode()$.

Определение только одного из этих методов приводит к тому, что идентичные объекты перестают быть равными, и, наоборот, в некоторых случаях разные объекты становятся идентичными. Это может привести к ошибкам работы с коллекциями (Collections), картами (Maps) и наборами (Sets).

Описание эвристического алгоритма выявления дефекта следующее:

1) поиск в классах переопределения метода Equals;

2) поиск переопределения метода Hashcode;

3) выдача предупреждения о возможном дефекте кода CWE-581 в случае наличия только одного из них.

Продукционная модель эвристического правила имеет вид

$$\begin{cases} R_1 =< s_j \in S;\ s_j \in M_{obj} \wedge t_{obj}\,(s_j) = \text{finally}; \\ (f_{inside}\,(s_j) \cap M_{info} \cap \{\text{equals}, \text{hashCode}\}) \neq \emptyset \\ \rightarrow AP\ (s_j, L);; P_1(581, 3) > . \end{cases}$$

Пример дефекта кода, заключающегося в некорректном возврате значения. Рассмотрим дефект кода CWE-584, когда оператор возврата в блоке Finally (Return Inside Finally Block). Суть дефекта — оператор возврата в блоке Finally отбрасывает все исключения, сгенерированные в блоке Try, тем самым не будет проведена обработка внештатных ситуаций, в том числе связанных и с механизмами безопасности.

Описание эвристики выявления дефекта приведено ниже:

1) поиск блоков Finally;

2) анализ тела блока на наличие операторов возврата.

Продукционная модель CWE-584 может быть следующей:

$$\Big\{ R_1 =< s_j \in M_{func};\ t_{obj}\,(s_j) = \\ = 'finally';\ I_{ret} \in f_{inside}\ (s_j) \rightarrow AP\ (s_j, L);\ P_1(584, 2) > .$$

Об эффективности и результативности эвристического анализа. Предложенный способ выявления широкого спектра уязвимостей методом эвристического анализа имеет непосредственно прикладное значение, так как доведен до практической реализации и прошел апробацию в реальных проектах.

Разработанный теоретический аппарат стал основой создания анализатора AppChecker, реализованного на парсерах языков Java и C/C++. Правила продукционной системы анализатора выполнены с помощью XPath-запросов к XML-представлению абстрактного синтаксического дерева кода [17].

Полнота проверок обеспечена их привязкой к международному реестру дефектов безопасности CWE. В настоящее время анализатор (версия 1.0) включает в себя 74 эвристики для языка Java и 55 для языка C/C++.

Сравнение статических анализаторов (основанных на проверках свойств и поиске по шаблонам) на тестовых примерах с открытым кодом показал сопоставимую их результативность и качество работы [18].

Следует понимать, что эвристические подходы в большей степени зависят от квалификации экспертов: как при написании эвристик, так и при анализе потенциально опасного фрагмента кода [19]. В то же время, выявление преднамеренных программных закладок обеспечено только методом эвристического анализа [1].

Предложенный метод не претендует на исключительность: в реальности важен синтез возможных методик выявления конкретного класса дефектов по критериям быстродействия и уровней ошибок первого и второго рода. Например, международный стандарт ISO/IEC TR 20004:2012 предлагает методики выявления известных уязвимостей, опубликованных в открытых бюллетенях по безопасности [20]. Такой подход повышает эффективность структурного тестирования, например, при первичном анализе заимствованных компонентов.

Заключение. По результатам исследования можно сделать следующие выводы.

1. Предлагаемый эвристический метод обеспечивает полноту и непротиворечивость проверок, так как не налагает ограничений на классификации дефектов программного кода.

2. Метод может служить базовым методом при практическом комплексировании различных техник статического анализа безопасности программ (с учетом мнений квалифицированных экспертов).

3. Особенность метода эвристического анализа — возможность идентифицировать преднамеренные закладки, опираясь на экспертные признаки деструктивного кода.

4. Разработанная метамодель позволяет формализовать описание эвристик поиска дефектов на различных уровнях представления кода (например, на уровне абстрактного синтаксического дерева и абстрактного семантического графа), что обеспечивает повышение унификации выявления сложных уязвимостей. К достоинству метамодели можно добавить:

— наглядность и относительную простоту реализации эвристических алгоритмов выявления уязвимостей и собственно базы шаблонов (сигнатур);

— высокую скорость работы на практике;

— легкость модификации сигнатур и их портирования на различные платформы и языки программирования.

Предложенные метод и инструментарий полезны не только аккредитованным испытательным лабораториям, но и разработчикам безопасных программных средств.

ЛИТЕРАТУРА

1. *Марков А.С.*, *Цирлов В.Л.* Опыт выявления уязвимостей в зарубежных программных продуктах // Вопросы кибербезопасности. 2013. № 1 (1). С. 42–48.
2. *Markov A.*, *Luchin D.*, *Rautkin Y.*, *Tsirlov V.* Evolution of a Radio Telecommunication Hardware-Software Certification Paradigm in Accordance with Information Security Requirements // In Proceedings of the 11th International Siberian Conference on Control and Communications (Omsk, Russia, May 21–23, 2015). SIBCON–2015. IEEE, 2015. P. 1–4. URL: http://dx.doi.org/10.1109/SIBCON.2015.7147139 DOI: 10.1109/SIBCON.2015.7147139
3. *Using* Static Analysis to Find Bugs / N. Ayewah, D. Hovemeyer, J.D. Morgenthaler, J. Penix, W. Pugh. Using Static Analysis to Find Bugs // IEEE Software. 25, 5 (Sep./Oct. 2008). P. 22–29. URL: http://dx.doi.org/10.1109/MS.2008.130 DOI: 10.1109/MS.2008.130
4. *Chen H.*, *Wagner D.* MOPS: an infrastructure for examining security properties of software // In Proceedings of the 9th ACM conference on Computer and communications security. CCS'02. New York, 2002. P. 235–244.
5. *Hovemeyer D.*, *Spacco J.*, *Pugh W.* Evaluating and tuning a static analysis to find null pointer bugs // CM SIGSOFT Software Engineering Notes. 2006. Vol. 31. No. 1. P. 13–19. URL: http://dx.doi.org/10.1145/1108768.1108798 DOI: 10.1145/1108792.1108798
6. *Logozzo F.*, *Fähndrich M.* On the Relative Completeness of Bytecode Analysis Versus // Source Code Analysis. LNCS. 2008. No. 4959. P. 197–212.
7. *Detecting* mobile malware threats to homeland security through static analysis / S.-H. Seoa, A. Guptaa, A.M. Sallama, E. Bertinoa, K. Yimb // Journal of Network and Computer Applications. 2014. Vol. 38. P. 43–53. URL: http://dx.doi.org/10.1016/j.jnca.2013.05.008 DOI: 10.1016/j.jnca.2013.05.008
8. *Zhu F.*, *Wei J.* Static analysis based invariant detection for commodity operating systems // Computers and Security. 2014. No. 43. P. 49–63. URL: http://dx.doi.org/10.1016/j.cose.2014.02.00 DOI: 10.1016/j.cose.2014.02.00
9. *Осовецкий Л.Г.* Технология выявления недекларированных возможностей при сертификации промышленного программного обеспечения по требованиям безопасности информации // Вопросы кибербезопасности. 2015. № 1 (9). С. 60–64.

10. *Барабанов А.В.*, *Марков А.С.*, *Цирлов В.Л.* Оценка соответствия средств защиты информации “Общим критериям” // Информационные технологии. 2015. Т. 21. № 4. С. 264–270.
11. *Barabanov A.*, *Markov A*. Modern Trends in The Regulatory Framework of the Information Security Compliance Assessment in Russia Based on Common Criteria // In Proceedings of the 8th International Conference on Security of Information and Networks (Sochi, Russian Federation, September 8–10, 2015). SIN’15. ACM New York, NY, USA. 2015. P. 30–33. URL: http://dx.doi.org/10.1145/2799979.2799980 DOI: 10.1145/2799979.2799980
12. *Static* Analysis Technologies Evaluation Criteria v1.0.; ed. by Sherif Koussa; Russian translation by A. Shcherbakov, A. Markov. Web Application Security Consortium. 2013. URL: http://projects.webappsec.org/w/page/71979863/Static%20Analysis%20Technologies%20Evaluation%20 Criteria%20-%20Russian
13. *Марков А.С.*, *Фадин А.А*. Систематика уязвимостей и дефектов безопасности программных ресурсов // Защита информации. Инсайд. 2013. № 3 (51). С. 56–61.
14. *Аветисян А.И.*, *Белеванцев А.А.*, *Чукляев И.И.* Технологии статического и динамического анализа уязвимостей программного обеспечения // Вопросы кибербезопасности. 2014. № 3 (4). С. 20–28.
15. *Медведев Н.В.*, *Марков А.С.*, *Фадин А.А*. Применение метода статического сигнатурного анализа для выявления дефектов безопасности веб-приложений // Наука и образование. МГТУ им. Н.Э. Баумана. Электрон. журн. 2012. № 9. С. 297–316. URL: http://technomag.bmstu.ru/doc/461281.html DOI: 10.7463/0912.0461281
16. *Boulanger J.L.* (Ed.). Static Analysis of Software: The Abstract Interpretation. Wiley-ISTE, 2011. 331 p.
17. *Система* для определения программных закладок / В.В. Вылегжанин, А.Л. Маркин, А.С. Марков, Р.А. Уточка, А.А. Фадин, А.К. Фамбулов, В.Л. Цирлов. Патент на полезную модель RUS 114799. Заявка № 2011153967, 29.12.2011. 2012. Бюл. 10.
18. *Марков А.С.*, *Фадин А.А.*, *Швец В.В*. Сравнение статических анализаторов безопасности программного кода // Защита информации. Инсайд. 2015. № 6 (66). С. 38–47.
19. *Жидков И.В.*, *Кадушкин И.В.* О признаках потенциально опасных событий в информационных системах // Вопросы кибербезопасности. 2014. № 1 (2). С. 40–48.
20. *Барабанов А.В.*, *Евсеев А.Н.* Вопросы повышения эффективности анализа уязвимостей при проведении сертификационных испытаний программного обеспечения по требованиям безопасности информации // Труды международного симпозиума “Надежность и качество”. 2015. Т. 1. С. 330–333.

REFERENCES

[1] Markov A.S., Tsirlov V.L. Experience in Identifying Vulnerabilities in Software. *Voprosy kiberbezopasnosti* [Cybersecurity issues], 2013, no. 1 (1), pp. 42–48 (in Russ.).
[2] Markov A., Luchin D., Rautkin Y., Tsirlov V. Evolution of a Radio Telecommunication Hardware-Software Certification Paradigm in Accordance with Information Security Requirements, In Proceedings of the 11th International Siberian Conference on Control and Communications (Omsk, Russia, May 21–23, 2015). SIBCON-2015. *IEEE*, 2015, pp. 1–4. DOI = http://dx.doi.org/10.1109/SIBCON.2015.7147139
[3] Ayewah N., Hovemeyer D., Morgenthaler J.D., Penix J., Pugh W. Using Static Analysis to Find Bugs. *IEEE Software*, 2008, 25, 5 (Sep./Oct. 2008), pp. 22–29. DOI= http://dx.doi.org/10.1109/MS.2008.130

[4] Chen H., Wagner D. MOPS: an infrastructure for examining security properties of software. *Proc. of the 9th ACM conference on Computer and communications security. CCS'02*. N.Y., 2002, pp. 235–244.

[5] Hovemeyer D., Spacco J., Pugh W. Evaluating and tuning a static analysis to find null pointer bugs. CM SIGSOFT Software Engineering Notes, 2006, 31, 1 (Jan.), pp. 13–19. DOI= http://dx.doi.org/10.1145/1108768.1108798

[6] Logozzo F., Fähndrich M. On the Relative Completeness of Bytecode Analysis Versus Source Code Analysis. LNCS. 4959, 2008, pp. 197–212.

[7] Seoa S.-H., Guptaa A., Sallama A.M., Bertinoa E., Yimb K. 2014. Detecting mobile malware threats to homeland security through static analysis. *Journal of Network and Computer Applications*, 2014, 38 (Feb.), pp. 43–53. URL: http://dx.doi.org/10.1016/j.jnca.2013.05.008 DOI: 10.1016/j.jnca.2013.05.008

[8] Zhu F., Wei J. Static analysis based invariant detection for commodity operating systems. *Computers and Security*, 2014, no. 43, pp. 49–63. DOI=http://dx.doi.org/10.1016/j.cose.2014.02.00

[9] Osovetskiy L.G. Detection Technology Undeclared Capabilities (Ndv) for the Certification of the Software Industry at the Request of Information Security. *Voprosy kiberbezopasnosti* [Cybersecurity issues], 2015, no. 1 (9), pp. 60–64 (in Russ.).

[10] Barabanov A.V., Markov A.S., Tsirlov V.L. The Conformity Assessment of Information Security Solutions According to the Common Criteria. *Informatsionnye tekhnologii* [Information technologies], 2015, vol. 21, no. 4, pp. 264–270 (in Russ.).

[11] Barabanov A., Markov A. Modern Trends in The Regulatory Framework of the Information Security Compliance Assessment in Russia Based on Common Criteria. *Proc. of the 8th International Conference on Security of Information and Networks* (Sochi, Russian Federation, September 8–10, 2015). SIN '15. ACM New York, N.Y., USA, 2015, pp. 30–33. URL: http://dx.doi.org/10.1145/2799979.2799980 DOI: 10.1145/2799979.2799980

[12] Static Analysis Technologies Evaluation Criteria v1.0. Ed. by Sherif Koussa; Russian translation by Alec Shcherbakov, Alexey Markov, *Web Application Security Consortium*, 2013. Available at: http://projects.webappsec.org/w/page/71979863/Static%20Analysis%20Technologies%20Evaluation%20Criteria%20-%20Russian

[13] Markov A.S., Fadin A.A. Systematics of vulnerabilities and security defects of program resources. *Zasita informacii. Inside*, 2013, no. 3 (51), pp. 56–61 (in Russ.).

[14] Avetisyan A.I., Belevantsev A.A., Chuklyaev I.I. The Technologies of Static and Dynamic Analyses Detecting Vulnerabilities of Software. *Voprosy kiberbezopasnosti* [Cybersecurity issues], 2014, no. 3 (4), pp. 20–28 (in Russ.).

[15] Medvedev N.V., Markov A.S., Fadin A.A. Application of static signature analysis to detect defects in web applications security. *Nauka i obrazovanie*. *MGTU im. N.E. Baumana* [Science & Education of the Bauman MSTU. Electronic Journal], 2012, no. 9, p. 21. Available at: http://technomag.edu.ru/en/doc/461281.html DOI: 10.7463/0912.0461281

[16] Boulanger, J.L., ed. Static Analysis of Software. The Abstract Interpretation. Wiley-ISTE. 2011.

[17] Vylegzhanin V.V., Markin A.L., Markov A.S., Utochka R.A., Fadin A.A., Fambulov A.K., Tsirlov V.L. Sistema dlya opredeleniya programmnykh zakladok [The System for Determining the Software Bugs]. Patent granting for useful model no. RUS 114799. 29.12.2011.

[18] Markov A.S., Fadin A.A., Shvets V.V. Comparison of Software Code Security Static Analyzers. *Zasita informacii. Inside*, 2015, no. 6 (66), pp. 2–7 (in Russ.).

[19] Zhidkov I.V., Kadushkin I.V. About the Signs of Potentially Dangerous Events in Information Systems. *Voprosy kiberbezopasnosti* [Cybersecurity issues], 2014, no. 1 (2), pp. 40–48 (in Russ.).

[20] Barabanov A.V., Evseev A.N. Improving the Efficiency of Vulnerability Analysis during Software Certification Testing for Information Security Requirements. *Tr. Mezhdunar. simpoziuma Nadezhnost' i kachestvo* [Proceedings of the International Symposium "Reliability and Quality"], 2015, vol. 1, pp. 330–333 (in Russ.).

Статья поступила в редакцию 10.11.2015

Марков Алексей Сергеевич — д-р техн. наук, старший научный сотрудник, профессор кафедры "Информационная безопасность" МГТУ им. Н.Э. Баумана.
МГТУ им. Н.Э. Баумана, Российская Федерация, 105005, Москва, 2-я Бауманская ул., д. 5.

Markov A.S. — Dr. Sci. (Eng.), Senior Research Fellow, Professor of Information Security department, Bauman Moscow State Technical University.
Bauman Moscow State Technical University, 2-ya Baumanskaya ul. 5, Moscow, 105005 Russian Federation.

Матвеев Валерий Александрович — д-р техн. наук, профессор, заведующий кафедрой "Информационная безопасность" МГТУ им. Н.Э. Баумана, руководитель НУК "Информатика и системы управления" МГТУ им. Н.Э. Баумана.
МГТУ им. Н.Э. Баумана, Российская Федерация, 105005, Москва, 2-я Бауманская ул., д. 5.

Matveev V.A. — Dr. Sci. (Eng.), Professor, Head of Information Security department, Bauman Moscow State Technical University, Head of Scientific and Educational Complex for Information Technologies and Control Systems, Bauman Moscow State Technical University.
Bauman Moscow State Technical University, 2-ya Baumanskaya ul. 5, Moscow, 105005 Russian Federation.

Фадин Андрей Анатольевич — главный конструктор НПО "Эшелон".
НПО "Эшелон", Российская Федерация, 107023, Москва, ул. Электрозаводская, д. 24.

Fadin A.A. — Design Manager, Scientific Production Association Echelon.
Scientific Production Association Echelon, Electrozavodskaya ul. 24, Moscow, 107023 Russian Federation.

Цирлов Валентин Леонидович — канд. техн. наук, доцент кафедры "Информационная безопасность" МГТУ им. Н.Э. Баумана.
МГТУ им. Н.Э. Баумана, Российская Федерация, 105005, Москва, 2-я Бауманская ул., д. 5.

Tsirlov V.L. — Cand. Sci. (Eng.), Assoc. Professor of Information Security department, Bauman Moscow State Technical University.
Bauman Moscow State Technical University, 2-ya Baumanskaya ul. 5, Moscow, 105005 Russian Federation.

Просьба ссылаться на эту статью следующим образом:

Марков А.С., Матвеев В.А., Фадин А.А., Цирлов В.Л. Эвристический анализ безопасности программного кода // Вестник МГТУ им. Н.Э. Баумана. Сер. Приборостроение. 2016. № 1. С. 98–111. DOI: 10.18698/0236-3933-2016-1-98-111

Please cite this article in English as:

Markov A.S., Matveev V.A., Fadin A.A., Tsirlov V.L. Heuristic analysis of source code security. *Vestn. Mosk. Gos. Tekh. Univ. im. N.E. Baumana, Priborostr.* [Herald of the Bauman Moscow State Tech. Univ., Instrum. Eng.], 2016, no. 1, pp. 98–111.
DOI: 10.18698/0236-3933-2016-1-98-111