www.mashizdat.ru

УДК 681.3.06

СТАТИЧЕСКИЙ СИГНАТУРНЫЙ АНАЛИЗ БЕЗОПАСНОСТИ ПРОГРАММ

[1]*Марков А.С.*, [2]*Фадин А.А.*

[1]*МГТУ им.Н.Э.Баумана*
[2]*НПО «Эшелон»*

Рассматриваются концептуальные вопросы статического анализа безопасности программ. Проводится сравнительный анализ методов статического анализа безопасности программ относительно моделей представления программного кода. Обосновывается применимость статического сигнатурного анализа для выявления уязвимостей программ. Представлены примеры описания сигнатур и выявления дефектов безопасности методом статического сигнатурного анализа.

Ключевые слова: Информационная безопасность, безопасность программ, структурное тестирование, статический анализ, сигнатурный анализ, уязвимости, дефекты, недекларированные возможности.

STATIC SIGNATURE ANALYSIS OF SECURITY PROGRAMS

[1]*Markov A.S.*, [2]*Fadin A.A.*

[1]*Bauman MSTU*
[2]*NPO «Echelon»*

The conceptual issues of the static analysis of the security programs are discussed. A comparative analysis of the static analysis on code presentation models is carried out. The applicability of static signature analysis to identify vulnerabilities is substantiated. The examples of identify signatures and security weakness by static signature analysis are described

Keywords*:* information security, software security, structural testing, static analysis, signature analysis, vulnerability, security weakness, undeclared capabilities.

Введение

Проблема безопасности программного обеспечения (ПО) является одной из основных в области информационной безопасности (ИБ), требующей решения комплекса задач по непрерывному совершенствованию методов тестирования, испытаний и контроля. По причине астрономической размерности задач динамического тестирования безопасности программных систем, актуальным считается внедрение современных методов статического анализа программного кода. Более того, применение некоторых техник статического анализа для контроля отсутствия недекларированных возможностей (НДВ) регламентировано нормативными документами регуляторов [1]. В современной литературе по статическому анализу основанное внимание уделено техникам верификации программ по моделям представления их кода, соответственно направленным на выявление некорректностей кодирования (нефункциональных ошибок) [2-10]. В то же время, комплексное решение реальных задач обеспечения безопасности ПО обуславливает исследование методов сигнатурного анализа, позволяющих выявлять любые потенциально опасные, по мнению экспертов, фрагменты кода.

Таксономии уязвимостей и дефектов безопасности

В настоящее время имеется нечеткость в определении уязвимостей, ошибок, дефектов и НДВ ПО, влияющих на безопасность системы и подлежащих выявлению в процессе анализа безопасности ПО.

В международной практике статического анализа условно разделяют понятия уязвимости и дефекта безопасности. Под дефектом безопасности (weakness, bug) понимают недостаток создания ПО, потенциально влияющий на степень безопасности информации. Эксплуатируемый дефект безопасности представляет собой уязвимость (vulnerability), реализация которой составляет угрозу ИБ. Таким образом, статический анализ имеет дело в первую очередь с выявлением дефектов, которые в процессе локализации могут быть идентифицированы как уязвимости.

В российской нормативной базе имеется ряд определений отдельных классов уязвимостей, а именно: программной закладки (см. ГОСТ Р 50.1.053-2005), скрытого канала (ГОСТ Р 53113.2-2009), бреши/уязвимости (ГОСТ Р 50922-2006) и НДВ (РД Гостехкомиссии России). В частности, в РД Гостехкомиссии России [1] НДВ и программная закладка соответственно трактуются как некая уязвимость безопасности, не описанная в документации, и преднамеренная уязвимость инициируемого типа.

В настоящее время популярен ряд международных классификаций и таксономий, ориентированных на категории угроз ИБ, наиболее известными из которых являются: HP Fortunity, OWASP и Mitre CWE.

В классификации Fortify выделяется 7 типов (так называемых разрушительных «царств» по аналогии с биологией) дефектов безопасности исходного кода, касающихся валидации и представления ввода, реализации API, механизмов безопасности, времени и состояния, обработки ошибок, качества кода, инкапсуляции, а также окружения. Текущая классификация OWASP определяет 10 классов критических уязвимостей веб-проектов: инъекции, межсайтовый скриптинг, нарушения аутентификации и управления сессиями, незащищенная прямая ссылка на объект, подделка межсайтовых запросов, некорректная конфигурация настроек безопасности, незащищенное хранилище криптографических объектов, ошибки ограничения доступа, недостаточная защита транспортного уровня, некорректные редиректы и перессылки. Среди стандартов Mitre следует назвать всестороннюю классификацию дефектов CWE (Common Weaknesses Enumeration) и реестр уязвимостей CVE (Common Vulnerability Enumeration). Например, текущая версия CWE 2.1 включает в себя 617 дефектов ПО, отнесённых к 104 категориям.

Кроме актуальных категорий угроз ИБ, в академическом плане выделяют следующие классификационные признаки дефектов и уязвимостей безопасности:

- место в жизненном цикле (архитектурные, кодирования, эксплуатации);
- степень преднамеренности (например: функциональные и нефункциональные ошибки, отладочные средства, программные закладки, скрытые каналы);
- компрометируемая подсистема безопасности (аутентификации, авторизации, целостности, аудита, криптографической защиты, протоколы безопасности, интерфейсы безопасности и др.), а также ее реализация и версия.

Пример классификации, имеющей практическое приложение при анализе безопасности систем, представлен на рисунке 1.

С точки зрения реализации статического *сигнатурного* анализа прикладное значение, на наш взгляд, имеет следующая модель уязвимости:

$$SD = <CP; DD; CL; PR>,$$

где: *CP* – признаки кодирования и проектирования дефекта, *DD* - область данных, определяющая эксплуатацию дефекта; *CL* – категория (тип) уязвимости, *PR*- степень критичности.

Рассмотрим методы статического анализа, используемые в области ИБ.

Методы статического анализа безопасности программ

Статический анализ программ - это тип структурного тестирования (по принципу «белого ящика») ПО без реального выполнения программного кода. Объектами исследования при статическом анализе, кроме исходных текстов ПО, могут быть объектный и исполняемый программный код, проектная, компиляционная, отладочная и компоновочная информация. Современные методы статического анализа в целях унификации алгоритмов используют различные модели представления кода, например:

- лексический код;
- синтаксическое дерево (дерево разбора);

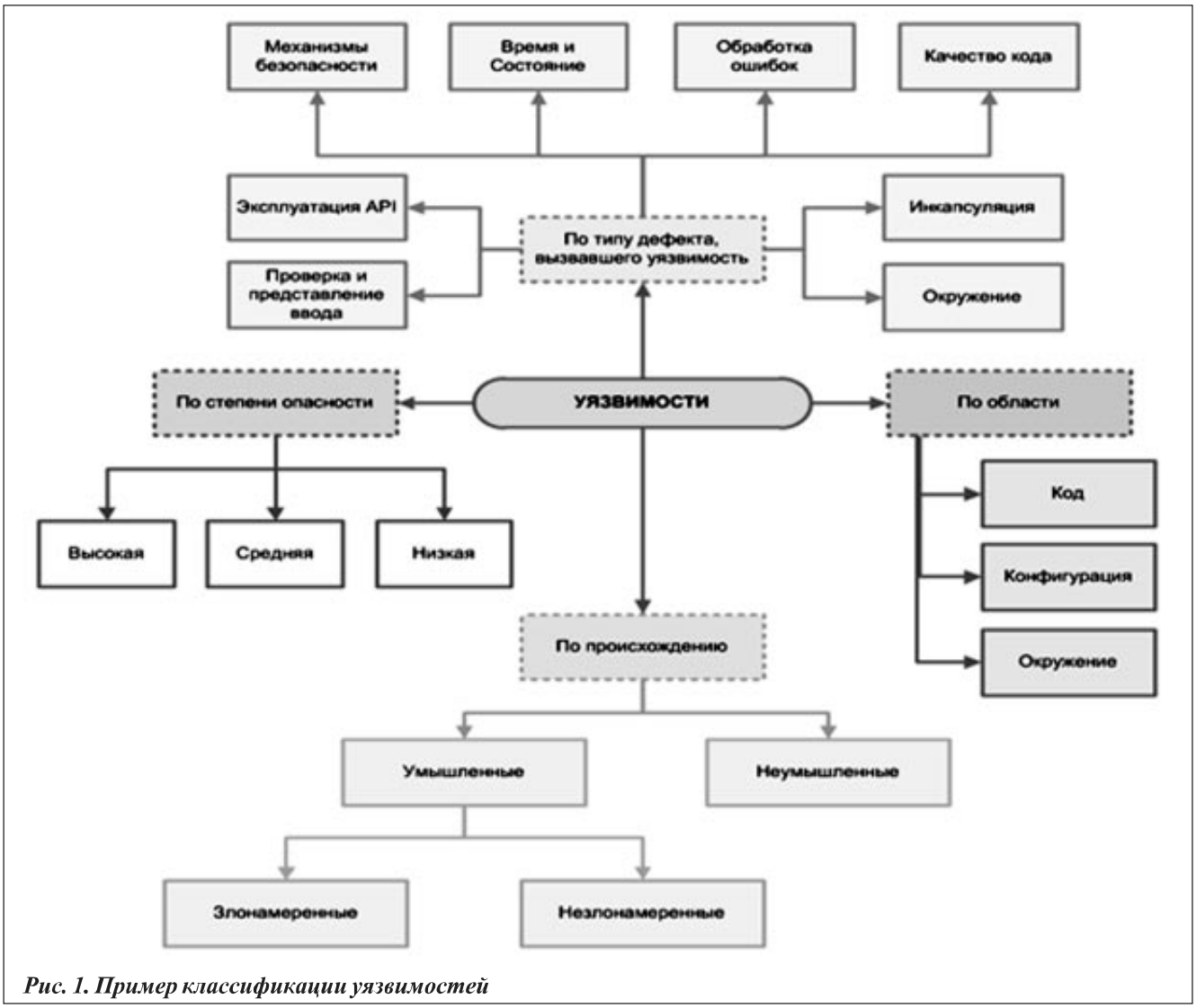

Рис. 1. Пример классификации уязвимостей

- дерево Канторовича или абстрактное дерево разбора (abstract syntax tree, AST);
- граф потока управления (control flow graph, CFG);
- граф зависимости по данным (data flow graph, DFG);
- модель однократного статического присваивания (static single assignment, SSA);
- абстрактный семантический граф (abstract semantic graph, ASG).

Надо понимать достоинства и недостатки статического анализа. Статический анализ не требует перебора всевозможных комбинаций входных данных, как следствие, одинаково эффективен для выявления как ошибок, связанных с редко используемыми входными данными, так и очевидных ошибок. В то же время статический анализ не позволяет контролировать динамические моменты кода, например, всевозможные условия выполнения циклов или рекурсий.

В области ИБ можно выделить четыре категории методов статического анализа:

- анализ структуры и декомпозиция программы, ориентированный на выявление нарушений полноты и избыточности;
- измерение метрик сложности, направленное на выявление сложных и аномальных фрагментов;
- проверка свойств (ограничений) моделей представления программного кода, позволяющая выявлять некорректности кодирования;
- сигнатурный поиск, ориентированный на выявление потенциально опасных фрагментов кода.

С точки зрения локализации дефектов наиболее интересны два последних подхода. Первый из них связан с проверкой корректности свойств моделей представления кода, получаемых в процессе компиляции или семантического анализа ПО. Это позволяет выявить разного рода некорректности кодирования, например: ошибки типов данных, ошибки указателей, неиспользуемые или неинициализированные переменные, утечки памяти, переполнение буфера и т.п.

Наиболее известными алгоритмами выявления некорректностей кодирования являются: интервальный анализ, анализ указателей, анализ зависимостей по данным, контроль типов данных и др.

К достоинству указанного подхода относят формальную строгость, привязку к модели представления ПО, способность «объяснить» некорректность кодирования (перечислением нарушенных атрибутов модели представления) и, как правило, меньшее число ложных срабатываний. Несмотря на некоторый прогресс в области практической реализации [6], на наш взгляд, остаются достаточно серьезные проблемы с автоматизацией анализа полного графа ПО, как правило, из-за сложности построения полной модели кода, большого числа связей между элементами высокого уровня (функций, классов и т.п.), ведущих к значительному росту объема вычислений. На практике разработчики средств анализа данного типа делают определенные упрощения, аппроксимации, создавая например какие-либо обобщенные свойства (атрибуты) для функций. При этом, как отмечалось, данный подход не позволяет определить разного рода нарушения спецификаций (НДВ), особенно преднамеренного типа, когда с точки зрения кодирования все корректно.

Второй подход, получивший название сигнатурного анализа [11], подразумевает поиск программных дефектов в программном коде путем сопоставления фрагментов кода с образцами из базы данных шаблонов (сигнатур) дефектов безопасности. В табл. 1 приведены по два примера правил формирования сигнатур (в данном случае, продукционных правил) для функциональных и нефункциональных ошибок, соответствующих стандарту CWE [1,12].

Как видно из табл. 1, сигнатурные методы не ограничены видами дефектов и наиболее предпочтительны, когда идет речь о программных закладках. Например, опыт аудита безопасности кода показывает, что самыми распространенными дефектами являются парольные константы и генераторы, а также наличие возможности снижения стойкости криптоалгоритмов. Так как эти дефекты с точки зрения системы программирования корректны, то их можно обнаружить только применяя сигнатурные подходы с привлечением экспертов. К недостаткам сигнатурного подхода относят большое число ложных срабатываний, поэтому фактически он направлен на снижение трудоемкости работы эксперта, предоставляя для его внимания наиболее критические фрагменты кода.

Результаты сравнения применимости категорий методов статического анализа (относительно различных уровней представления программного кода) приведены в табл. 2.

Из табл. 2 видно, что методы сигнатурного анализа применимы для выявления дефектов на различных уровнях представления кода. Разумеется, чем информативнее модель кода, тем больше возможностей для автоматизации выявления широкого класса дефектов. В тоже время, можно использовать различные модели представления кода в зависимости от сложности выявления дефекта безопасности или уязвимости.

Можно отметить согласованность метода статического сигнатурного анализа с действующей нормативной базой, а именно РД Гостехкомиссии России [1], применяемого во всех системах сертификации средств защиты информации. Согласно документу Гостехкомиссии России, для 2-го уровня контроля необходимо выполнять синтаксический контроль наличия заданных конструкций в исходных текстах ПО из списка (базы) потенциально опасных конструкций, а для 1-го уровня контроля необходимо выполнять семантический контроль наличия заданных конструкций в исходных текстах ПО из списка (базы) потенциально опасных конструкций.

Пример работы сигнатурного анализа

В качестве примера использования сигнатурного анализа рассмотрим фрагмент программы на языке PHP с потенциально реализуемой SQL-инъекцией содержимого cookies клиентского браузера в СУБД PostgreSQL, имеющий следующий вид:

```
<?php
```

Таблица 1

Примеры эвристик выявления дефектов безопасности

Элемент CWE верхнего уровня	Элемент CWE нижнего уровня	Признаки наличия	Порядок формирования продукционного правила
Недостаточная инкапсуляция (Insufficient Encapsulation) (485)	Остаточный отладочный код (Leftover Debug Code) (489)	*<наличие отладочного кода>*	Поиск присутствия вызовов функций из списка отладочных, наличия подозрительных правил в названиях отладочных функций, ключевых слов по debug, в т.ч. в комментариях
Механизмы безопасности (Security Features) (254)	Использование жестко прописанных паролей (Use of hard-coded password) (259)	*<наличие паролей и других данных авторизации непосредственно в коде приложения>*	Проверка вызовов функций с параметрами авторизации, использующих аргументы со строковыми константами; поиск функций, требующих авторизации вместе с переменными, которым были присвоены константы
Индикатор плохого качества кода (Indicator of Poor Code Quality) (398)	Присваивание вместо (Assigning instead of Comparing) (481)	*<наличие "=" вместо "==" в блоке if, где сравниваются лишь переменные>*	Проверка заданного списка регулярных выражений для содержимого заданных типов блоков
Ошибки очистки и инициализации (Initialization and Cleanup Errors) (452)	Неверная инициализация (Improper Initialization) (665)	*<отсутствие инициализации значений объектов перед их использованием>*	Поиск объявлений объектов, которые требуют инициализации перед своим использованием; поиск первого использования объекта в вызовах и операциях; при отсутствии их инициализации (левая часть в присваивании, функции, использующие их в виде выходных значений)

```
...
$id = $_COOKIE["mid»]; //получение данных
...
$query = "SELECT MessageID FROM messages WHERE
MessageID = '$id'"; //формирование запроса к БД
...
$result = pg_query($conn, $query); //выполнение запроса

?>
```

Для выявления потенциально опасного фрагмента можно легко воспользоваться, например, внутрипроцедурным уровнем представления кода (см. табл.2, строка 4). Для выделения потенциально опасного фрагмента кода можно использовать две проверки:

- для выявления потенциально опасных операций (в данном случае, pg_query);
- для выявления непроверенных источников данных (_COOKIE).

Выполнение внутрипроцедурного анализа может составить 4 шага:

```
1:(0,("_COOKIE"),"$id")
2:(0, "$id", "$query")
3:("pg_query", "$query" ,"$result")
4:("pg_query", "_COOKIE","$result")
```

Таблица 2

Модели представления кода, используемые при статическом анализе

Представление кода	Категории методов статического анализа			
	Декомпозиция и анализ структуры программы	**Проверка стиля, подсчет метрик**	**Проверка свойств**	**Поиск багов/дефектов по шаблону**
Исходные тексты ПО	Контроль зависимостей на уровне компонентов и отдельных файлов проекта	Подсчет базовых метрик длины кода	Свойства недоступны	Поиск сигнатур по регулярным выражениям
AST	Лексический и синтаксический анализ	Лексический и синтаксический анализ	Лексический и синтаксический анализ	Лексический и синтаксический анализ, Поиск сигнатур по AST
ASG: поток управления	Определение связей объектов в иерархии	Подсчет метрик связности по управлению	Анализ потока выполнения	Поиск сигнатур последовательностей инструкций
ASG: поток данных	Определение зависимостей по данным	Подсчет метрик связности по данным	Абстрактная интерпретация: интервальный анализ, анализ указателей, анализ зависимостей по данным и т.п.	Поиск внутрипроцедурных сигнатур последовательностей инструкций с учетом передаваемых значений
ASG: межпроцедурный анализ	Не требуется	Подсчет метрик связности	Анализ на основе систем уравнений	Поиск сигнатур с учетом связей между процедурами
ASG: семантические описания конструкций	Не требуется	Подсчет метрик связности с учетом семантики	Абстрактная интерпретация, ресурсный анализ, темпоральная логика	Поиск сигнатур на основе семантической базы конструкций, темпоральная логика
ASG: формальные методы верификации	Не требуется	Не требуется	Дедуктивная верификация	Не требуется

После шага 4 должны сработать одновременно две сигнатуры критичных операций (получение данных из COOKIE и выполнение запроса к СУБД), что сигнализирует о наличии потенциально опасной конструкции: использование данных, введенных пользователем в SQL-запросах без предварительной фильтрации.

Заключение

Требования регуляторов по выявлению НДВ ПО и опыт аудита безопасности ПО обуславливают широкое толкование статического анализа, включающего в том числе сигнатурные техники выявления уязвимостей и дефектов безопасности. Более того, сигнатурный подход в области ИБ остается основным, т.к. практика выявления критических уязвимостей, таких как: встроенные ключи, парольные константы, условия компрометации механизмов защиты, возможность подмены модулей, логические бомбы и т.п. показала, что они были выявлены именно сигнатурными подходами.

Сравнительный анализ различных методик и алгоритмов статического анализа показал, что сигнатурный анализ может быть реализован на различном уровне представления кода, выбор которого может зависеть от эффективности и особенностей используемого парсера, соответствующего конкретному языку программирования.

К достоинству статического сигнатурного анализа можно отнести относительную простоту реализации анализаторов и баз сигнатур, высокую скорость работы (для уровня исходных текстов и AST-дерева), легкость модификации сигнатур и их портирования на различные платформы и системы программирования. В тоже время при сигнатурном анализе необходимость учитывать синтактическую «вариативность» различных языков программирования, сложность реализации качественного межпроцедурного анализа, сложность обработки конструкций «распределенных» по модулям ПО. Это означает, что для проведения эффективных проверок безопасности кода необходим комплексный подход, сочетающий различные методики статического анализа и динамического тестирования.

Для решения задач выявления уязвимостей ПО разработано специализированное средство испытаний ПО на отсутствие НДВ, включающее модуль сигнатурного анализа, - «АК-ВС», которое успешно апробированного в ходе сотен испытаний программных средств защиты информации.

Библиографический список

1. Методы оценки несоответствия средств защиты информации / Под.ред. А.С.Маркова. М.: Радио и связь, 2012. 192 с.

2. Аветисян А., Белеванцев А., Бородин А., Несов В. Использование статического анализа для поиска уязвимостей и критических ошибок в исходном коде программ. // Труды Института системного программирования РАН. 2011. Т. 21. С. 23-38.

3. Беляев М.А., Цесько В.А. Статический анализ с использованием систем типов и эффектов на основе LLVM // Моделирование и анализ информационных систем. 2011. Т. 18. № 4. С. 45-55.

4. Глухих М.И., Ицыксон В.М. Программная инженерия. Обеспечение качества программных средств методами статического анализа. СПб.: Изд-во Политехн. ун-та, 2011. 150 с.

5. Заборовский Н.В., Тормасов А.Г. Моделирование многопоточного исполнения программы и метод статического анализа кода на предмет состояний гонки // Прикладная информатика. 2011. № 4. С. 105-110.

6. Карпов Ю. Новая жизнь верификации // Открытые системы. СУБД. 2012. № 3. С. 50-53.

7. Кириллин А.В. Инкрементальный статический анализ кода на основе разложения отношений системное // Программирование. 2006. Том 2. № 1. С. 5-24.

8. Колосов А.П., Рыжков Е.А. Применение статического анализа при разработке программ // Известия Тульского государственного университета. Серия: Технические науки. 2008. № 3. С. 185-190.

9. Моисеев М.Ю. Итеративный алгоритм статического анализа для обнаружения дефектов в исходном коде программ // Информационно-управляющие системы. 2009. № 3. С. 34-39.

10. Пучков Ф.М., Шапченко К. А. Статический метод анализа программного обеспечения на наличие угроз переполнения буферов // Программирование. 2005. Том 31. № 4. С. 19-34.

11. Марков А.С., Миронов С.В., Цирлов В.Л. Выявление уязвимостей в программном коде. // Открытые системы. СУБД. 2005. № 12. С.64-69.

12. Марков А.С., Фадин А.А. К вопросу о выявлении дефектов безопасности методом статического сигнатурного анализа. // Материалы XII Международной научно-практической конференции «ИБ-2012». Ч.II. – Таганрог: Изд-во ТТИ ЮФУ, 2012. С.39-45.

Сведения об авторе:

Марков Алексей Сергеевич: канд. техн. наук, доцент кафедры «Информационная безопасность» МГТУ им.Н.Э.Баумана, 105005, Москва, **2-я Бауманская** ул., д.5. a.markov@cnpo.ru, +74959712298.

Фадин Андрей Анатольевич: CISSP. Руководитель департамента ЗАО «НПО «Эшелон». 107023, г. Москва, ул. Электрозаводская, д. 24 стр. 1. E-mail: mail@cnpo.ru, +74956453810.

About the author:

Aleksey Markov: Ph.D., assistant professor of «Information Security» Bauman MSTU, 105005, Moscow, 2nd Bauman Str., 5. a.markov@cnpo.ru, +74959712298.

Andrey Fadin: CISSP. Head of Department, JSC «NPO» Echelon «. 107023, Moscow, ul. Electrozavodskaya, 24 -1. E-mail: mail@cnpo.ru, +74956453810.

Индекс по каталогу «Пресса России»: 10339

ПРОГРАММНАЯ ИНЖЕНЕРИЯ И ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ

Научно-технический журнал

ISSN

Издаётся с 2013 г.

Журнал зарегистрирован Федеральной службой по надзору в сфере связи и массовых коммуникаций.
Свидетельсво о регистрации средства массовой информации ПИ № ФС77-50623 от 19.07.2012.

Учредитель
ООО «Научно-техническое предприятие «Вираж-Центр»

Издательство, редакция
ООО НТП «Вираж-Центр»
Россия, 105264, Москва, ул. Верхняя Первомайская, д. 49, корп. 1 офис 401.
Почтовый адресс: Россия, 105043, Москва, а/я 29
Тел.: 8 495 780-94-73
http://www.machizdat.ru
e-mail: virste@dol.ru

Директор журнала
М.А.Мензуллов

Вёрстка
А.А.Мензуллов

Отпечатано: ООО «РПЦ ОФОРТ»
г. Москва, пр-кт Будённого, 21

Заказ №
Тираж 100
Цена – договорная