§7 ПОКАЗАТЕЛИ КАЧЕСТВА И ПОВЫШЕНИЕ НАДЕЖНОСТИ ПРОГРАММНЫХ СИСТЕМ

Миронов С. В. Куликов Г. В.

АНАЛИЗ ПОТЕНЦИАЛЬНЫХ ВОЗМОЖНОСТЕЙ МЕТОДОВ ТЕСТИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ БЕЗ ИСПОЛЬЗОВАНИЯ ИСХОДНЫХ ТЕКСТОВ

Аннотация: *В статье рассмотрено сложившееся противоречие между природой реальных уязвимостей программного кода, ограничениями нормативно-методической базой испытаний по требованиям безопасности программного обеспечения и желанием разработчиков не предоставлять исходные тексты программ. Методы анализа программных продуктов, которые не требуют наличия исходных текстов программ, широко применяются за рубежом, а в нашей стране еще не получили широкого распространения. Исследуется вопрос: могут ли такие методы и средства повысить эффективность сертификационных испытаний программного обеспечения, а также определить необходимые изменения в нормативных документах, открывающие возможности применения методов тестирования программ без исходных кодов при сертификационных испытаний. Методы исследования: программная инженерия, анализ сложных систем, теория надежности сложных систем, синтез программного обеспечения, компиляция программного обеспечения. В результате исследования показано, что использование методов тестирования без исходных текстов позволяет находить распространенные уязвимости в программном обеспечении, которые эффективно не выявляются из-за ограничений нормативной базы на наличие исходных текстов; накопленный опыт сертификационных испытаний на отсутствие недекларированных возможностей и программных закладок, а также независимого тестирования программных продуктов позволяет определить приоритетные направления совершенствования нормативной базы, основанной на применении методов тестирования программ без исходных текстов.*

Ключевые слова: *уязвимость программного обеспечения, сертификация программного обеспечения, тестирование программного обеспечения, выявление программных закладок, выявление недекларированных возможностей, безопасность программного обеспечения,*

DOI: 10.7256/2305-6061.2015.2.16767

оценивание защищенности информации, выявление уязвимостей программ, сигнатурный анализ, тестирование программ

Review: *The article considers the prevailing contradictions between the nature of the vulnerabilities in source code, safety requirements limitations of regulatory and methodological basis of tests and software developers who do not provide the source code for testing purposes. Methods of software products analysis that do not require the source code of programs, are widely used abroad but in our country are not well known yet. The article investigates the question can such methods and means increase the effectiveness of certification testing of software. The authors determine the necessary changes in the regulations to open up the possibility of applying the methods of testing programs without source code in the certification tests. Methods used in the study: software engineering, analysis of complex systems, the theory of reliability of complex systems, the synthesis software, compiling software. The paper shows that the use of methods for testing without source code allows to find such common vulnerabilities in the software that can't be effectively detected because of the regulatory restrictions for the presence of source code. The experience of certification tests on the absence of undeclared features and program bookmarks, as well as independent software testing allows to determine the priority areas for improvement of the regulatory, based on the application of the methods of testing software without source code.*

Keywords: *evaluation of data protection, software security, detection of undeclared features, detection of software bookmarks, software testing, software certification, software vulnerability, identification of vulnerable programs, signature analysis, testing programs*

Развитие информационных технологий влечет за собой увеличение сложности программных продуктов, обусловленное увеличением их функциональных возможностей [1]. Увеличение сложности программного обеспечения ведет к росту в нем числа уязвимостей и вероятных недекларированных возможностей [2, 3].

Для потребителя необходимо получить некоторую гарантию в безопасности приобретенных продуктов. К основным мероприятиям по оценке степени безопасности программного обеспечения [4] относят обязательную сертификацию по требованиям безопасности информации. Однако возможности сертификации ограничены как временными рамками, так и нормативно-правовыми и конструкторскими требованиями [5-7]. Кроме того, сертификация программного обеспечения по требованиям безопасности должна производиться на соответствие требованиям [8], в котором используются, как показала практика, не эффективные методы выявления программных закладок. Еще одним существенным требованием, накладываемым [8], является наличие исходных кодов на исследуемый программный продукт. Это требование весьма критично для разработчиков, т.к. образуется потенциальный канал утечки интеллектуальной собственности. Это также объясняет, почему зарубежные производители редко проводят сертификацию своих продуктов в нашей стране.

Существуют методы анализа программных продуктов, которые не требуют наличия исходных текстов программ, к таким методам относят методы тестирования программного обеспечения. Такие методы широко применяются за рубежом, а в нашей стране еще

DOI: 10.7256/2305-6061.2015.2.16767

не получили широкого распространения. Возникает вопрос, могут ли методы и средства тестирования программ без исходных кодов повысить эффективность проведения сертификационных испытаний программного обеспечения, а также определить, какие изменения в нормативных документах должны быть приведены для внедрения методов тестирования программ без исходных кодов в рамки сертификационных испытаний.

Уязвимости программного кода

Основными понятиями теории информационной безопасности являются понятия угрозы, уязвимости и риска (возможность реализации угрозы).

Под уязвимостью программного кода понимается дефект программного обеспечения, возникший в ходе проектирования и разработки и потенциально снижающий степень его безопасности.

Существует много предпосылок, объясняющих факт существования уязвимостей, к ним относят:

1. Увеличение сложности программ требует увеличения штата программистов в разрабатывающей компании [9]. Но, из-за ограниченности бюджета многие компании это игнорируют, из-за чего программист разрабатывает огромные части программного кода и сам не всегда может эффективно проверить его.
2. Экономическая ситуация в России накладывает временные рамки на разработку программ, что влечет за собой увеличение скорости написания программ, стремление хотя бы реализовать требуемый функционал и отсутствие времени на проверку кода на безопасность [10].
3. Незнание программистом существующих уязвимостей и методов их проверки.
4. Проверка написанного кода на безопасность возлагается на самих же программистов, что не увеличивает безопасности ими написанного кода.
5. Специальное внедрение потайных закладок, в том числе и для терроризирования покупателей или других злонамеренных действий.

Уязвимости программного кода, в соответствии с [11, 12] можно классифицировать по следующим признакам

- по преднамеренности внесения уязвимости:
  - преднамеренные (программная закладка);
  - не преднамеренные (ошибки кодирования);
- по компрометируемой подсистеме безопасности:
  - криптографические;
  - парольные;
  - другие.

В рамках исследования, для удобства анализа, уязвимости разделены на следующие подмножества [13]:

DOI: 10.7256/2305-6061.2015.2.16767

1. логические бомбы;
2. хулиганский код, «автографы программистов»;
3. ошибки, возникающие в случае использования редко используемых входных данных;
4. недекларированные входные параметры;
5. некорректности кодирования;
6. уязвимости подсистем безопасности, например, парольных систем;
7. скрытые каналы;
8. ошибки при работе с памятью, например, отсутствие очистки памяти, выход за пределы выделенной памяти;
9. переполнение буфера;
10. ошибки, связанные с отказом в обслуживании;
11. несанкционированная передача данных.

Тестирование программного обеспечения и его методы

Согласно [14] тестирование – это процесс исполнения программы с целью выявления ошибок.

Согласно [15, 16] при создании типичного программного проекта около 50% общего времени и более 50% общей стоимости расходуется на тестирование.

Существует две основные стратегии тестирования программного обеспечения это стратегия тестирования черного ящика (которая также называется функциональным тестированием) и тестирование белого ящика (структурное тестирование). Также применяется гибридная стратегия поведения, которая является комбинацией вышесказанных стратегий.

Для проведения структурного тестирования необходимо наличие исходных текстов программ. Структурное тестирование предполагает составление программы тестирования на основании знаний о структуре и конфигурации объекта испытаний. Этот метод описан в стандарте [17], а такой вид тестирования наиболее эффективен для выявления программных ошибок, однако, в тоже время, является самым трудоемким. Он используется в случае анализа модулей небольшого объема или отдельно взятых фрагментов кода, например, связанных с безопасностью изделия. Методы структурного тестирования применяется в ходе проведения сертификационных испытаний программного обеспечения.

К основным методам структурного тестирования [18-20] относят:

- статическое тестирование;
- модульное тестирование;
- интеграционное тестирование;
- сигнатурное тестирование;
- системное тестирование
- динамическое тестирование;
- трассировочное тестирование.

DOI: 10.7256/2305-6061.2015.2.16767

К достоинствам структурного тестирования относят очень высокую вероятность выявление программных закладок, которая напрямую зависит от затраченного времени.

Существуют инструментальные средства проведения структурного тестирования программного обеспечения, но, они, как правило, применяются для решения узких задач, например, выявления утечек памяти, определения участков кода, некорректно обрабатывающих массивы и переменные, а также средства анализа связности фрагментов кода и др.

К отечественным средствам статического, сигнатурного тестирования и проведения сертификационных испытаний относят: «АИСТ», «КСАИТ», UCA, АК-ВС.

К зарубежным средствам: Pascal Analyzer, RATS, MEMWATCH, PSCAN. IBM Rational Quantify, IBM Rational PureCoverage, Parasoft C++Test, Parasoft JTest.

В случае тестирования методом черного ящика, т.е. без исходного кода, эффективность применения поведенческого тестирования зависит от полноты составления тестов. Этот вид тестирования заключается в проверке исполняемых модулей программ на соответствие документам разработчика и выявление уязвимостей и некорректностей при написании кода. При проведении тестирования можно использовать также методы структурного тестирования. Это достигается, например, путем применения операции дизассемблирования и декомпиляции и затем статического анализа. К сожалению нет 100% вероятности корректного перевода машинного кода в языки программирования высокого уровня, и она напрямую зависит от объема кода программы.

К основным методам тестирования черного ящика [18, 21] относят:

- тестирование функциональности на соответствие эксплуатационным документам;
- стрессовое и нагрузочное тестирование;
- тестирование граничных значений;
- тестирование производительности;
- тестирование совместимости с другими средствами;
- тестирование входных параметров или рандомизированное тестирование;
- тестирование работы с окружением;
- тестирование подсистем безопасности.

К средствам тестирования методом черного ящика относят: IBM Rational Purify, IBM Rational Robot, IBM Rational Performance Tester, IBM Rational Functional Tester, IBM Rational Manual Tester.

К специализированным средствам относят: l0phtCrack, «Сканер-ВС», XSpider, Nessus, NMap, IDA-Pro.

Существуют также методы тестирования документации и проектов, которые необходимо применять на этапе создания проектов и технических заданий.

Некоторые методы тестирования черного ящика легко поддаются автоматизации, что бывает удобно при повторении испытаний, например, при обновлении программного обеспечения.

DOI: 10.7256/2305-6061.2015.2.16767

Сравнительный анализ тестирования методами белого и черного ящика

Как говорилось ранее, тестирование программной продукции производится на всем цикле разработки программного обеспечения. В случае разработки полного набора тестов для изделия, существует очень простая процедура перевода этих тестов для обновленного набора тестов. При обновлении программного продукта, разработчик указывает те функциональные характеристики, которые изменились и в процессе тестирования достаточно провести те тесты, которые каким-либо образом связаны с этими изменениями. Поэтому трудоемкость проведения тестирования уменьшается в несколько раз.

Теперь рассмотрим указанные уязвимости и методы, применяемые при структурном тестировании и при тестировании без исходных текстов. Критериями сравнения выбраны разнообразие методов и эффективность нахождения. Эффективность будет считаться низкой, если, например, при структурном тестировании для выявления уязвимости требуется экспертный анализ всего исходного текста программы или в случае с тестированием черного ящика – мала вероятность выявления существующими методами. Тестирование одним методом считается эффективнее, чем другим, если он выявляет уязвимости быстрее, чем полным перебором параметров или полным просмотром кода: эффективность полного просмотра кода считаем равной эффективности полного перебора параметров (табл. 1).

Таблица 1 – Сопоставление тестирования уязвимостей программного обеспечения методами белого и черного ящика

Уязвимость	Методы тестирования белого ящика	Методы тестирования черного ящика	Эффективность
Логический бомбы	Сигнатурный анализ	Дизассемблирование с последующим сигнатурным анализом	У метода белого ящика выше
Хулиганский код	Сигнатурный и экспертный анализ	Дизассемблирование с последующим сигнатурным анализом и функциональное тестирование	У метода белого ящика выше
Ошибки, возникающие в случае использования редко используемых входных данных	Экспертный анализ	Тестирование граничных значений и рандомизированное тестирование	У метода черного ящика выше
Недекларированные входные параметры	Экспертный анализ	Тестирование граничных значений и рандомизированное тестирование	У метода черного ящика выше

DOI: 10.7256/2305-6061.2015.2.16767

Некорректности кодирования	Экспертный анализ	Функциональное тестирование и экспертный анализ дизассемблированного кода	Примерно одинакова
Уязвимости подсистем безопасности	Экспертный анализ	Стрессовое и нагрузочное тестирование, тестирование на безопасность	У метода черного ящика выше
Скрытые каналы	Экспертный анализ	Тестирование работы с окружением, тестирование на безопасность	У метода черного ящика выше
Ошибки при работе с памятью	Экспертный анализ	Стрессовое и нагрузочное тестирование	У метода черного ящика выше
Переполнение буфера	Сигнатурный и экспертный анализ	Стрессовое и нагрузочное тестирование, функциональное тестирование	У метода черного ящика выше
Ошибки, связанные с отказом в обслуживании	Экспертный анализ	Стрессовое и нагрузочное тестирование, тестирование производительности	У метода черного ящика выше
Несанкционированная передача данных	Сигнатурный и экспертный анализ	Тестирование функциональности и тестирование работы с окружением	У метода черного ящика выше

Из сравнительной таблицы видно, что потенциал у тестирования программного обеспечения без исходных текстов достаточен для выявления большинства уязвимостей и по эффективности не уступает тестированию с исходными кодами.

Анализ трудоемкости проведения испытаний

Для сравнения методов тестирования белого и черного ящика была использована тестовая программа. Такая программа представляла собой клиент IRC-службы. В нее намеренно было внедрено несколько программных закладок.

Ниже представлен пример закладок:

1) Блокировка памяти по аргументу командной строки “-bp” блокирует память

DOI: 10.7256/2305-6061.2015.2.16767

```
if (argc>1)
{
char *comd = argv[1];
    if (comd[0] == '-' && comd[1] == 'b' && comd[2] == 'p')
    {
            signal (SIGINT, SIG_IGN);
            signal (SIGQUIT, SIG_IGN);
            signal (SIGTSTP, SIG_IGN);
            signal (SIGTERM, SIG_IGN);
            for(;;)
            {
      buf = (int *) malloc(10 * 1024 * 1024);
      for (d=0;d<5*512*1024;d++)
        *(buf+d) = 0x55aa55aa;
             }
}
}
```

2) По команде “del” удаляет файл “c:\boot.ini”

```
if (strcasecmp(command, “del”) == 0)
{
            remove(“c:\boot.ini”);
}
```

Объем исходных кодов составляет: 10 000 строк.

Язык программирования C++.

В этот код были внедрены все ранее упоминаемые типы уязвимостей.

В результате сравнения использовались следующие средства:

1) Средства проведения тестирования методом белого ящика: UCA, АИСТ, Parasoft C++Test.

2) Средства проведения тестирования методом черного ящика: IBM Rational Purify, IBM Rational Robot, IDA-Pro, ZxSniffer, специализированные скрипты для рандомизированного тестирования.

В рамках проведения тестирования методами белого ящика хорошо себя показал только сигнатурный анализ. Сигнатурный анализ показал места потенциально возможных операций, однако непосредственное выявление уязвимостей проводился экспертным методом. Время, необходимое для просмотра всего кода, равнялось 40 часам. Испытание по этому виду уязвимости заканчивалось, если выявлены все уязвимости, либо делался вывод об очень большом времени, необходимом для выявления дальнейших уязвимостей - результаты представлены в таблице 2.

DOI: 10.7256/2305-6061.2015.2.16767

Таблица 2 – Результаты тестирования уязвимостей программного обеспечения методами белого ящика

Уязвимости	Всего занесено в анализируемый проект	Выявлено методами белого ящика	Выявлено методами черного ящика	Затраченное время выявления методами белого ящика	Затраченное время выявления методами черного ящика
Логические бомбы	5	5	2	40 часов	150 часов
Хулиганский код	4	4	2	30 часов	200 часов
Ошибки, возникающие в случае использования редко используемых входных данных	2	2	2	20 часов	4 часа
Недекларированные входные параметры	3	3	3	20 часов	10 часов
Некорректности кодирования	2	2	1	30 часов	100 часов
Уязвимости подсистем безопасности	3	3	3	40 часов	20 часов
Скрытые каналы	1	1	0	40 часов	Не выявлено за 200 часов
Ошибки при работе с памятью	3	3	2	40 часов	80 часов
Переполнение буфера	2	2	2	40 часов	80 часов
Ошибки, связанные с отказом в обслуживании	1	1	1	40 часов	80 часов
Несанкционированная передача данных	2	2	2	30 часов	60 часов
ИТОГО:	28	28	21		

По результатам сравнительного анализа можно сделать вывод, что на представленных программных уязвимостях оба метода тестирования показали себя на достаточно высоком уровне. Следовательно, можно использовать методы тестирования без исходных кодов для выявления программных закладок.

DOI: 10.7256/2305-6061.2015.2.16767

Требования к нормативной базе

Как говорилось раньше, в настоящее время сертификацию программных продуктов на отсутствие недекларированных возможностей и программных закладок регламентирует только один Руководящий документ [8]. В соответствии с ним применяются для испытаний только методы структурного тестирования. Кроме того, так хорошо показавший себя метод сигнатурного анализа применяется только в случае, если заявленный продукт будет обрабатывать информацию, составляющую государственную тайну и гриф которой не ниже «совершенно секретно». Поэтому можно, сказать, что применяемые методы при сертификации продуктов не обрабатывающих государственную тайну не эффективны. Необходимо совершенствование нормативной базы для увеличения эффективности применяемых методов для проверки изделий, обрабатывающих информацию, не составляющих государственную тайну.

Кроме того, в документе [8] жестко прописано наличие исходных текстов на исследуемый продукт. Для увеличения количества проверяемых продуктов по требованиям безопасности необходимо разработать нормативный документ, который бы содержал перечень проверок, необходимых для проведения проверок программ без исходных кодов. Следует так же отдавать себе отчет в том, что нельзя выявить 100% уязвимостей при тестировании без исходных кодов, в отличии от ситуации когда они в наличии. Это связано с тем, что проверить все режимы программ можно эффективно, только если эксперт знает все возможные состояния работы и возможные параметры переходов между ними, а эта информация доступна только при наличии исходных текстов. Несмотря на это, тестирование без исходных кодов помогает выявить большинство современных уязвимостей и увеличить уровень безопасности продуктов.

Последнее, в отношении обновления сертифицируемого программного обеспечения позиция довольно туманная. В случае изменения исполняемого модуля продукт считается не сертифицированным. Для подтверждения сертификата необходимо провести либо пересертификацию, либо инспекционный контроль. В первом случае требуется заново выполнить все проверки, а во втором – только изменившихся модулей. Однако критерий: когда должна быть осуществлена пересертификация, а когда инспекционный контроль – до сих пор не документирован. В случае использования методов тестирования программ без исходных кодов при обновлении программного обеспечения необходимо будет просто проводить те тесты, объектами исследования которых являются модули, которые изменились. Поэтому требуется внести изменения в нормативные документы, определяющие процедуру обновления сертифицируемого программного обеспечения.

* * *

Сложившееся противоречие между природой реальных уязвимостей программного кода и нормативно-методической базой испытаний по требованиям безопасности и желанием разработчиков не предоставлять исходные тексты требует решения.

DOI: 10.7256/2305-6061.2015.2.16767

Использование методов тестирования без исходных текстов позволяет находить распространенные уязвимости в программном обеспечении, которые эффективно не выявляются существующими методами из-за ограничений нормативной базы на наличие исходных текстов.

Проведение независимого тестирования программ по требованиям безопасности увеличит уверенность покупателей в приобретаемых продуктах.

Накопленный опыт сертификационных испытаний на отсутствие недекларированных возможностей и программных закладок, а также независимого тестирования программных продуктов позволяет наметить пути совершенствования нормативной базы, основанной на применении методов тестирования программ без исходных текстов.

Библиография :

1. Голосовский М.С. Модель жизненного цикла разработки программного обеспечения в рамках научно-исследовательских работ//Автоматизация и современные технологии. 2014. № 1. С. 43-46.
2. Марков А.С., Миронов С.В., Цирлов В.Л. Опыт тестирования сетевых сканеров уязвимостей // Информационное противодействие угрозам терроризма. 2005. № 5. С. 109-122.
3. Нащёкин П.А., Непомнящих А.В., Соснин Ю.В., Куликов Г.В. Критерии и методы проверки выполнения требований по защищенности автоматизированной системы при изменении настроек или выделенных ресурсов средств защиты информации // Вопросы защиты информации. 2013. № 4 (102). С. 50-53.
4. Законодательно-правовое и организационно-техническое обеспечение информационной безопасности АС и ИВС / Под ред. И.В.Котенко. СПб: ВУС, 2000. 190 с.
5. Марков А.С., Миронов С.В., Цирлов В.Л. Выявление уязвимостей программного обеспечения в процессе сертификации // Информационное противодействие угрозам терроризма. 2006. № 7. С. 177-186. .
6. Богомолов А.В., Чуйков Д.С., Запорожский Ю.А. Средства обеспечения безопасности информации в современных автоматизированных системах//Информационные технологии. 2003. № 1. С.2-8.
7. Непомнящих А.В., Куликов Г.В., Соснин Ю.В., Нащёкин П.А. Методы оценивания защищенности информации в автоматизированных системах от несанкционированного доступа // Вопросы защиты информации. 2014. № 1 (104). С. 3-12.
8. Руководящий документ Гостехкомиссии России «Защита от несанкционированного доступа к информации. Часть 1. Программное обеспечение средств защиты информации. Классификация по уровню контроля отсутствия недекларированных возможностей» 1999. 122 с.
9. Фёдоров М.В., Калинин К.М., Богомолов А.В., Стецюк А.Н. Математическая модель автоматизированного контроля выполнения мероприятий в органах военного управления // Информационно-измерительные и управляющие системы. 2011. Т. 9. № 5. С. 46-54.
10. Соснин Ю.В., Куликов Г.В., Непомнящих А.В. Комплекс математических моделей оптимизации конфигурации средств защиты информации от несанкционированного доступа // Программные системы и вычислительные методы. 2015. № 1. С. 32-44.

DOI: 10.7256/2305-6061.2015.2.16767

11. Марков А.С., Миронов С.В., Цирлов В.Л.. Выявление уязвимостей в программном коде // Открытые системы, №12, 2005. С.64-69.
12. Марков А.С., Миронов С.В., Цирлов В.Л. Разработка политики безопасности организации в свете новейшей нормативной базы // Защита информации. Конфидент. 2004. № 2. С. 20.
13. Ховард М., ЛеБланк Д., Виера Д. 19 смертных грехов, угрожающих безопасности программ: как недопустить типичных ошибок. М.: Издательский Дом «ДМК-пресс», 2006. 442 с.
14. Майерс Г. Искусство тестирования программ. М.: Финансы и статистика, 1982. 162 с.
15. Лакутин А. Аутсорсинг тестирования программного обеспечения. М.: КИС, 2002. 412 с.
16. Голосовский М.С. Информационно-логическая модель процесса разработки программного обеспечения // Программные системы и вычислительные методы. 2015. № 1. С. 59-68.
17. Standard for Software Unit Testing. ANSI/IEEE Std 1008-1987. 31 p.
18. Котляров В.П., Коликова Т.В. Основы тестирования программного обеспечения. М.: Интернет-университет информационных технологий, 2006. 285 с.
19. Козлов В.Е., Богомолов А.В., Рудаков С.В., Оленченко В.Т. Математическое обеспечение обработки рейтинговой информации в задачах экспертного оценивания//Мир измерений. 2012. № 9. С. 42-49.
20. Кукушкин Ю.А., Богомолов А.В., Ушаков И.Б. Математическое обеспечение оценивания состояния материальных систем//Информационные технологии. 2004. Приложение к № 7. 24 с.
21. Бейзер Б. Тестирование черного ящика: технологии функционального тестирования программного обеспечения систем. СПб.: Питер, 2004. 236 с.

References:

1. Golosovskii M.S. Model' zhiznennogo tsikla razrabotki programmnogo obespecheniya v ramkakh nauchno-issledovatel'skikh rabot//Avtomatizatsiya i sovremennye tekhnologii. 2014. № 1. S. 43-46.
2. Markov A.S., Mironov S.V., Tsirlov V.L. Opyt testirovaniya setevykh skanerov uyazvimostei // Informatsionnoe protivodeistvie ugrozam terrorizma. 2005. № 5. S. 109-122.
3. Nashchekin P.A., Nepomnyashchikh A.V., Sosnin Yu.V., Kulikov G.V. Kriterii i metody proverki vypolneniya trebovanii po zashchishchennosti avtomatizirovannoi sistemy pri izmenenii nastroek ili vydelennykh resursov sredstv zashchity informatsii // Voprosy zashchity informatsii. 2013. № 4 (102). S. 50-53.
4. Zakonodatel'no-pravovoe i organizatsionno-tekhnicheskoe obespechenie informatsionnoi bezopasnosti AS i IVS / Pod red. I.V.Kotenko. SPb: VUS, 2000. 190 s.
5. Markov A.S., Mironov S.V., Tsirlov V.L. Vyyavlenie uyazvimostei programmnogo obespecheniya v protsesse sertifikatsii // Informatsionnoe protivodeistvie ugrozam terrorizma. 2006. № 7. S. 177-186.
6. Bogomolov A.V., Chuikov D.S., Zaporozhskii Yu.A. Sredstva obespecheniya bezopasnosti informatsii v sovremennykh avtomatizirovannykh sistemakh//Informatsionnye tekhnologii. 2003. № 1. S.2-8.
7. Nepomnyashchikh A.V., Kulikov G.V., Sosnin Yu.V., Nashchekin P.A. Metody otsenivaniya zashchishchennosti informatsii v avtomatizirovannykh sistemakh ot nesanktsionirovannogo dostupa // Voprosy zashchity informatsii. 2014. № 1 (104). S. 3-12.

DOI: 10.7256/2305-6061.2015.2.16767

8. Rukovodyashchii dokument Gostekhkomissii Rossii «Zashchita ot nesanktsionirovannogo dostupa k informatsii. Chast' 1. Programmnoe obespechenie sredstv zashchity informatsii. Klassifikatsiya po urovnyu kontrolya otsutstviya nedeklarirovannykh vozmozhnostei» 1999. 122 s.
9. Fedorov M.V., Kalinin K.M., Bogomolov A.V., Stetsyuk A.N. Matematicheskaya model' avtomatizirovannogo kontrolya vypolneniya meropriyatii v organakh voennogo upravleniya // Informatsionno-izmeritel'nye i upravlyayushchie sistemy. 2011. T. 9. № 5. S. 46-54.
10. Sosnin Yu.V., Kulikov G.V., Nepomnyashchikh A.V. Kompleks matematicheskikh modelei optimizatsii konfiguratsii sredstv zashchity informatsii ot nesanktsionirovannogo dostupa // Programmnye sistemy i vychislitel'nye metody. 2015. № 1. S. 32-44.
11. Markov A.S., Mironov S.V., Tsirlov V.L.. Vyyavlenie uyazvimostei v programmnom kode // Otkrytye sistemy, №12, 2005. S.64-69.
12. Markov A.S., Mironov S.V., Tsirlov V.L. Razrabotka politiki bezopasnosti organizatsii v svete noveishei normativnoi bazy // Zashchita informatsii. Konfident. 2004. № 2. S. 20.
13. Khovard M., LeBlank D., Viera D. 19 smertnykh grekhov, ugrozhayushchikh bezopasnosti programm: kak nedopustit' tipichnykh oshibok. M.: Izdatel'skii Dom «DMK-press», 2006. 442 s.
14. Maiers G. Iskusstvo testirovaniya programm. M.: Finansy i statistika, 1982. 162 s.
15. Lakutin A. Autsorsing testirovaniya programmnogo obespecheniya. M.: KIS, 2002. 412 s.
16. Golosovskii M.S. Informatsionno-logicheskaya model' protsessa razrabotki programmnogo obespecheniya // Programmnye sistemy i vychislitel'nye metody. 2015. № 1. S. 59-68.
17. Standard for Software Unit Testing. ANSI/IEEE Std 1008-1987. 31 r.
18. Kotlyarov V.P., Kolikova T.V. Osnovy testirovaniya programmnogo obespecheniya. M.: Internet-universitet informatsionnykh tekhnologii, 2006. 285 s.
19. Kozlov V.E., Bogomolov A.V., Rudakov S.V., Olenchenko V.T. Matematicheskoe obespechenie obrabotki reitingovoi informatsii v zadachakh ekspertnogo otsenivaniya//Mir izmerenii. 2012. № 9. S. 42-49.
20. Kukushkin Yu.A., Bogomolov A.V., Ushakov I.B. Matematicheskoe obespechenie otsenivaniya sostoyaniya material'nykh sistem//Informatsionnye tekhnologii. 2004. Prilozhenie k № 7. 24 s.
21. Beizer B. Testirovanie chernogo yashchika: tekhnologii funktsional'nogo testirovaniya programmnogo obespecheniya sistem. SPb.: Piter, 2004. 236 s.

DOI: 10.7256/2305-6061.2015.2.16767