УДК 004.05
doi:10.21685/2307-4205-2021-1-14

ИСПОЛЬЗОВАНИЕ СИСТЕМЫ АУДИТА ОПЕРАЦИОННЫХ СИСТЕМ СЕМЕЙСТВА LINUX ПРИ ПРОВЕДЕНИИ СЕРТИФИКАЦИОННЫХ ИСПЫТАНИЙ ПРОГРАММНЫХ ИЗДЕЛИЙ

В. В. Самаров

ООО «16 НИИЦ», Мытищи, Россия
samarov_vladimir@mail.ru

Аннотация. Рассматривается общая проблематика выполнения работ по регистрации событий компиляции и сборки программных изделий, проходящих сертификационные испытания в ситеме сертификации Минобороны России, при выполнении этапа работ по контролю требований по полноте и отсутствию избыточности исходных текстов на уровне файлов. Для фиксации событий компиляции и сборки проекта из исходных текстов предлагается к использованию универсальный метод, не зависящий от схемы и средств сборки, основанный на механизмах регистрации, предоставляемых системой аудита операционных систем семейства Linux (далее – ОС Linux). Приведен алгоритм по настройке системы аудита ОС Linux и получения записей, фиксирующих события, описывающих процесс регистрации и сборки исследуемого проекта. Рассмотрены выходные данные, генерируемые системой аудита ОС Linux, и дано обоснование возможности и целесообразности использования полученных данных при проведении соответствующего этапа сертификационных испытаний.

Ключевые слова: сертификационные испытания программных изделий, контроль полноты и отсутствия избыточности на уровне файлов, система аудита ОС Linux

Для цитирования: Самаров В. В. Использование системы аудита операционных систем семейства Linux при проведении сертификационных испытаний программных изделий // Надежность и качество сложных систем. 2021. № 1. С. 144–150. doi:10.21685/2307-4205-2021-1-14

USE OF THE LINUX OPERATING SYSTEM AUDIT SYSTEM WHEN CONDUCTING CERTIFICATION TESTS OF SOFTWARE PRODUCTS

V. V. Samarov

LLC "16 NIITS", Mytishchi, Russia
samarov_vladimir@mail.ru

Abstract. The general problematics of the work on the registration of compilation and assembly events of software products undergoing certification tests in the certification system of the Ministry of Defense of Russia, when performing the stage of work to control the requirements for the completeness and absence of redundancy of source texts at the file level, is considered. To record the compilation and build events of a project from source texts, it is proposed to use a universal method that does not depend on the scheme and build tools, based on the registration mechanisms provided by the audit system of operating systems of the Linux family (hereinafter referred to as Linux OS). An algorithm for setting up a Linux OS audit system and obtaining records that record events describing the process of registration and assembly of the project under study is presented. The paper considers the output data generated by the Linux OS audit system and substantiates the possibility and expediency of using the obtained data during the corresponding stage of certification tests.

Keywords: certification testing of software products, control of completeness and lack of redundancy at the file level, Linux OS audit system

For citation: Samarov V.V. Use of the linux operating system audit system when conducting certification tests of software products. *Nadezhnost' i kachestvo slozhnykh sistem = Reliability and quality of complex systems.* 2021;1:144–150. (In Russ.). doi:10.21685/2307-4205-2021-1-14

При проведении сертификационных испытаний программных изделий испытательными лабораториями, аккредитованными в системе сертификации Минобороны России, выполняется этап работ по оценке полноты и отсутствию избыточности пакетов с исходными текстами контролируемого программного изделия на уровне файлов.

Структурно, согласно руководящему документу «Защита от несанкционированного доступа к информации. Часть 1. Программное обеспечение средств защиты информации. Классификация по уровню контроля отсутствия недекларированных возможностей» (Гостехкомиссия России, Москва, 1999 г.) [1], данный этап входит в группу испытаний «Статический анализ» и является обязательным к проведению для всех уровней контроля отсутствия недекларированных возможностей (далее – НДВ).

При проведении сертификационных испытаний контролируемого программного изделия по четвертому уровню контроля (конфиденциально) отсутствия НДВ, данный этап является одним из двух этапов проверок, проводимых непосредственно над исходными текстами программного изделия. Корректное выполнение проверок, регламентированных данным этапом, также крайне важно при испытаниях программных изделий на соответствие более высоким уровням контроля отсутствия недекларированных возможностей (третий и четвертый уровни контроля).

Действительно, необнаруженные на этапе проверки файловой избыточности файлы, в составе входящих в них функциональных объектов (процедур, функций) станут объектами исследования, выполняемого на последующих за этим этапом, стадиях испытаний в рамках статического и динамического анализа. При корректном проведении испытаний (при последующих испытаниях, проводимых в рамках статического и динамического анализа) файловая избыточность будет выявлена, но ресурсы испытательной лаборатории будут потрачены впустую.

Что касается важности установления полноты исходных текстов (как на уровне файлов, так и на уровне функциональных объектов), то необходимо отметить, что положительные результаты проверок по данному пункту являются критическим условием для дальнейших испытаний программного изделия и возможности формирования положительного заключения по результатам сертификационных испытаний.

Таким образом, важность этапа испытаний по установлению полноты и отсутствию избыточности пакетов с исходными текстами на уровне файлов не вызывает сомнений.

Статистика выполненных испытательной лабораторией ООО «16 НИИЦ» работ, а также участие сотрудников лаборатории в экспертизе материалов сертификационных испытаний, выполненных сторонними организациями, позволяет констатировать, что:

1) программные изделия, заявляемые на контроль соответствия отсутствия НДВ в системе сертификации Минобороны России, в подавляющем большинстве предназначены для функционирования в операционных системах семейства Linux (ОС «Astra Linux», ОС «МСВС», ОС «Заря», ОС «Роса» и некоторых других);

2) большая часть программных изделий разрабатывается на компилируемых языках C/C++/C# и их диалектов, а также на языке Java. При этом разработчиками используется все многообразие существующих средств разработки (в том числе средств компиляции и сборки пакетов с исходными текстами в дистрибутивные пакеты), а унификация этих средств отсутствует.

При проведении этапа контроля полноты на уровне файлов и отсутствия файловой избыточности на практике наиболее широко распространен метод, при котором сравниваются элементы (файлы) двух множеств (списков)[1]* по определенной маске (расширению), а результаты сравнения сохраняются в два (три) результирующих списка: 1) перечень файлов исходных текстов, имеющийся только в первом множестве; 2) перечень файлов исходных текстов, имеющийся только во втором множестве; 3) перечень совпадающих файлов исходных текстов.

Как было отмечено выше, при компиляции и сборке исходных текстов в бинарные дистрибутивные пакеты (*.deb, *.rpm, *.tar, *.tar.gz, *.cpio и др.) используются различные средства и системы сборки. В упрощенном виде схему сборки можно изобразить в следующем виде (рис. 1) [2]:

Исх. код → система сборки → компилятор → исполняемый (-е) файл (-ы)

Рис. 1. Упрощенная схема сборки проекта из исходных текстов

[1] Как правило, первое множество (список) представляет собой перечень файлов с исходными текстами, входящими в проект; второе множество – протокол, формируемый при компиляции исходных текстов из первого множества в соответствующие бинарные файлы и их последующей сборки в дистрибутивные пакеты.

При компиляции и сборке небольших проектов зачастую используются bash-скрипты или скрипты на иных интерпретируемых языках (php, python, ruby и др.), при сборке же более солидных проектов используются специализированные («front-end» и «back-end») средства (cmake, qmake, make, ninja, apache ant, apache maven и др.), автоматизирующие этот процесс (рис. 2), либо средства, интегрированные в различные среды разработки.

Исх. код → GNU MAKE → компилятор → исполняемый (-е) файл (-ы)

Исх. код → CMAKE → GNU MAKE или QMAKE или NINJA или… → компилятор → исполняемый (-е) файл (-ы)

Рис. 2. Варианты схем сборки при использовании автоматизированных средств сборки на примере средств сборки GNU MAKE и CMAKE

При использовании перечисленных схем сборки далеко не всегда возможно использовать полученные по результатам их работы протоколы (маршруты) компиляции (сборки) в целях контроля отсутствия файловой избыточности и подтверждения полноты на уровне файлов. Это связано с тем, что перед разработчиками программных изделий не стоит задача написания сборочных скриптов и инструкций (правил) сборочных систем, обеспечивающих детальное протоколирование процесса сборки (включая протоколирование всех обращений компилятора к файлам исходных текстов). Таким образом, зачастую протоколы сборки оказываются неинформативными с точки зрения фиксации факта непосредственной компиляции конкретного файла с исходными текстами (в некоторых случаях без использования специальных отладочных инструкций компилятора (как пример, ключ – *v* для компилятора gcc [3]) для каждого его вызова, проблематично достоверно установить факт компиляции).

Таким образом, для получения информативных с точки зрения компиляции протоколов экспертам, проводящим испытания, зачастую необходимо выполнить работы по модификации (доработке) сборочных скриптов, инструкций (правил) сборочных систем. Для солидных проектов данная задача может быть нетривиальной и трудоемкой. Далее, после получения протоколов компиляции, требуется их интерпретация и семантический анализ, что с учетом отсутствия унификации в средствах разработки, сборки и, как следствие, в формируемых выходных протоколах компиляции значительно увеличивает трудоемкость проводимых работ в целом.

Для решения обозначенной задачи предлагается использовать систему аудита (сервис auditctl), которая поддерживается операционными системами семейства Linux, начиная с ядра версии 2.6 [4]. Данная система предназначена для отслеживания критичных с точки зрения безопасности системных событий, вместе с тем она может быть использована для получения подробного протокола компиляции и сборки исходных текстов исследуемого проекта в соответствующие бинарные дистрибутивные пакеты. Основным доистоинством преставленного ниже алгоритма получения протоколов компиляции (сборки) с использованием системы аудита является его универсальность, т.е. представленный ниже алгоритм может быть использован для любых систем разработки и средств сборки.

Алгоритм получения лога аудита на примере команд, совместимых с операционными системами, основанными на дистрибутивах Debian GNU Linux и Mandriva Linux:

1) установка системы аудита (сервис auditctl).

А. Открыть терминал и выполнить команду

```
$ whereis auditd
```

В случае если в результате выполнения команды в терминал будет выведено:

– сообщение вида «`auditd:`», свидетельствующее о том, что компоненты системы аудита отсутствуют в операционной системе, то выполнить установку службы

```
$ sudo apt-get install auditd ;
```

– сообщение вида «`auditd: /sbin/audit.d…`», свидетельствующее о том, что компоненты системы аудита развернуты в операционной системе, то перейти к пункту 1 (Б).

Б. Проверка успешности установки службы аудита и контроль ее функционирования.

Выполнить команду

```
$ sudo service auditd status
```

В случае если по результатам выполнения команды в терминал будет выведено сообщение вида “`active (running)`”, перейти к следующему пункту действий (п. 2), в противном случае (“`inactive (dead)`”) активировать службу, для чего выполнить команду

```
$ sudo service auditd start
```

2) постановка на аудит каталога с анализируемыми исходными текстами
Выполнить команду

```
$ sudo auditctl - w /Dest/for/src -p rwxa
```

где */Dest/for/src* – абсолютный путь к каталогу с анализируемыми исходными текстами;

3) контроль настройки аудита

```
$ sudo auditdctl - l
```

4) выполнение компиляции и сборки программного изделия;
5) фиксация файла на съемный носитель для последующего анализа:

```
$ sudo cat /var/log/audit/audit.log > /media/aud_log.txt.
```

На примере простого тестового проекта (состоящего из двух Си файлов и makefile-а, описывающего схему их сборки в исполняемый файл (рис. 3), убедимся, что система аудита, настроенная в соответствии с приведенным выше алгоритмом, в полном объеме фиксирует последовательность сборки проекта в соответствии со схемой сборки, предствленной на рис. 1, 2 (в части GNU MAKE).

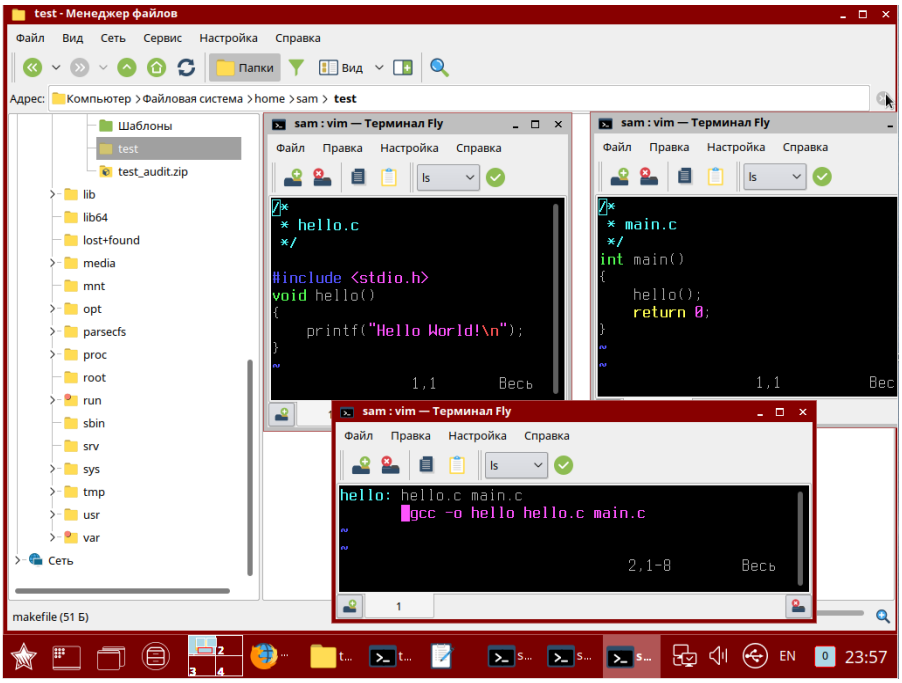

Рис. 3. Видеокадр содержимого файлов исходных текстов (файлы hello.c, main.c) и соответствующего им makefile-а, тестового проекта

Последовательность действий:

– в терминале перейти в каталог с тестовым проектом (предварительно выполнив действия п. 1–3 по настройке системы аудита, как описано выше):

```
$ cd /dest/for/src[1]
```

– произвести сборку тестового проекта, для чего выполнить команду

```
$ make
```

[1] Далее по тексту будут приведены данные для тестового каталога – */home/sam/test*.

– убедиться, что исполняемый файл сформирован и работоспособен, для чего выполнить команду

```
$ ./hello // проконтролировать появление
    в терминале надписи «Hello World»
```

– ознакомиться с логом аудита (aud_log.txt), для чего выполнить действия указанные в п. 5 вышеприведенного алгоритма, после чего открыть его для анализа в любом текстовом редакторе, например «Kate» (рис. 4).

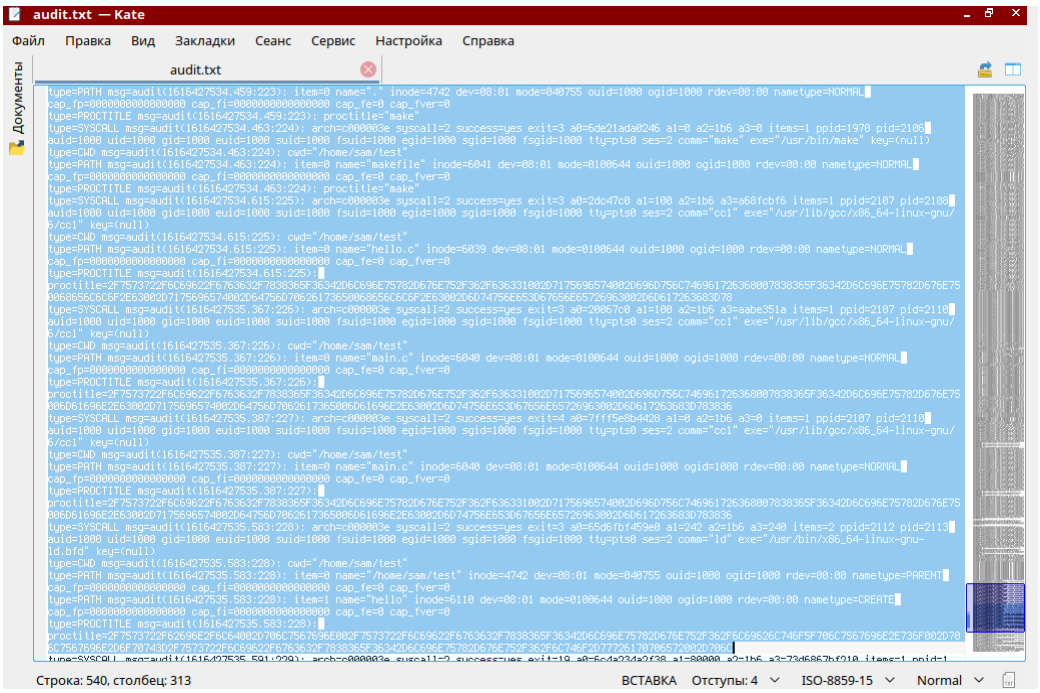

Рис. 4. Видеокадр файла аудита с выделенным фрагментом блоков записей, описывающих сборку

Из анализа содержимого лога аудита видно, что любое фиксируемое при сборке тестового проекта событие аудита состоит из четырех записей (`type= SYSCALL, CWD, PATH, PROCTITLE`) [5], имеющих одинаковую отметку времени и серийный номер (рис.4). В рассматриваемом примере сборка тестового проекта запротоколирована группами записей с серийными номерами 223–228.

Расшифровка записей:

223 – успешная инициализация системы сборки make *(/usr/bin/make)*;

224 – успешное связывание системы сборки с файлом инструкций makefile, находящимся в каталоге */home/sam/test*

225* – успешная компиляция (ассемблирование) файла */home/sam/test/hello.c*, компилятором gcc (команда */usr/lib/gcc/x86_64-linux-GNU/6/cc1 –quiet – imultiarch x86_64-linux-GNU hello.c –quiet – dumpbase hello.c –mtune=generic –march=x*);

226–227* – успешная компиляция (ассемблирование) файла */home/sam/test/main.c* компилятором gcc (команда */usr/lib/gcc/x86_64-linux-gnu/6/cc1 -quiet -imultiarch x86_64-linux-gnu main.c -quiet -dumpbase main.c -mtune=generic -march=x86*);

228, 232* – успешная линковка (связывание) объектных (временных) файлов в исполняемый файл hello, линковщиком ld (команда линковки – */usr/bin/ld –plugin usr/lib/gcc/x86_64-linux-GNU/6/liblto_plugin.so –plugin-opt=/usr/lib/gcc/x86_64-linux-gnu/6/lto-wrapper -pl*)

П р и м е ч а н и е: * Запись “ `PROCTITLE` ” частично представлена в виде строки в 16 СС.

Таким образом, фиксация системой аудита протокола сборки тестового проекта подтверждена. Вместе с тем представление информации в логе аудита избыточно, а сам он трудночитаем.

Для решения этой проблемы предлагается воспользоваться командой aureport [6]

`$ sudo aureport -f -i - success (1)// выборка из сформированного лога аудита` */var/log/audit/audit.log* `успешных записей обращения к файлам интерпретацией значений параметров`

Результат выполнения команды для тестового проекта представлен на рис. 5.

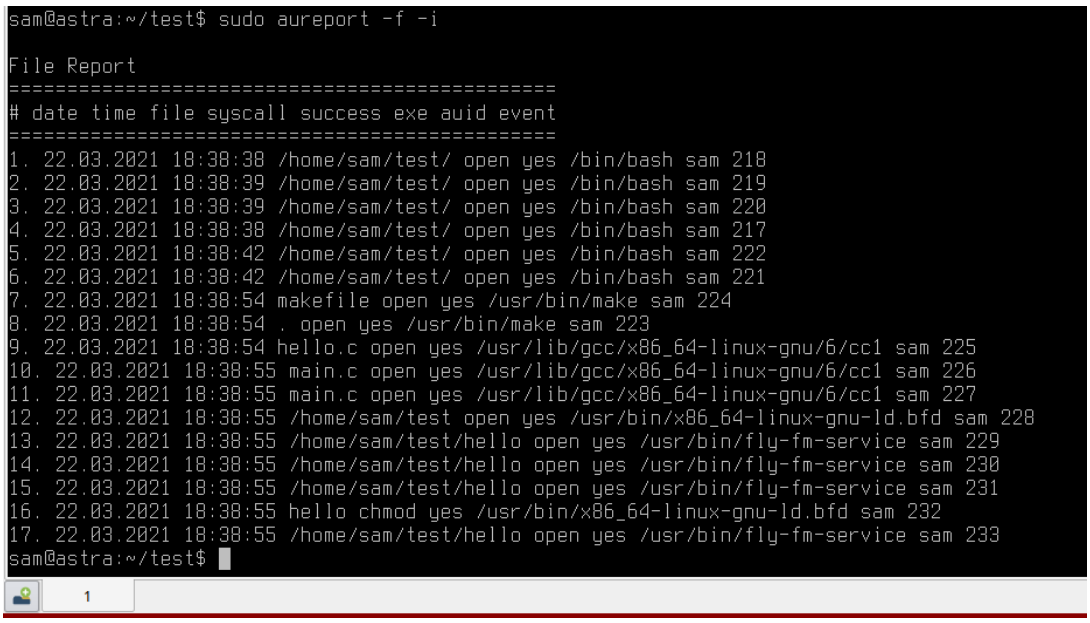

Рис. 5. Видеокадр выполнения утилиты aureport для просмотра успешных событий аудита

Как видно из результатов выполнения утилиты aureport, в целом регистрируемая при компиляции тестового проекта последовательность операций над контролируемыми файлами соответствует информации, регистрируемой системой аудита по умолчанию (файл */var/log/audit.log*). Полученный при выполнении утилиты результат менее детальный, но лучше читаемый и все еще достаточный для интерпретации процесса сборки. Еще одним достоинством применения утилиты aureport является объем регистрируемой информации (в среднем более чем в 6 раз меньше, чем генерируемый в части сборки рассматриваемого проекта лог файл (*/var/log/audit/audit.log*).

Таким образом, предложенный подход может быть использован специалистами испытательных лабораторий, аккредитованных в системе сертификации Минобороны России, при выполнении этапа работ по контролю требований по полноте и отсутствию избыточности исходных текстов на уровне файлов, и позволит в целом снизить трудоемкость проводимых работ. По мнению автора, основываясь на рассмотренных в данной статье результатах и данных, получаемых при использовании системы аудита *ОС Linux,* целесообразно проработать вопрос о возможности автоматизации соответствующего этапа сертификационных испытаний.

Список литературы

1. Защита от несанкционированного доступа к информации. Часть 1. Программное обеспечение средств защиты информации. Классификация по уровню контроля отсутствия недекларированных возможностей : руководящий документ. М. : Гостехкомиссия России, 1999. URL: https://fstec.ru/tekhnicheskaya-zashchita-informatsii/dokumenty/114-spetsialnye-normativnye-dokumenty/382-rukovodyashchij-dokument-prikaz-predsedatelya-gostekhkomissii-rossii-ot-4-iyunya-1999-g-n-114 (дата обращения: 26.02.2021).
2. Сборка упаковщика Cmake-ом или тренировка… URL: https://habr.com/ru/post/178839/ (дата обращения: 26.02.2021).
3. MAN gcc (FreeBSD и Linux). URL: https://www.opennet.ru/man.shtml?topic=gcc&russian=0&category=&submit=%F0%CF%CB%C1%DA%C1%D4%D8+man (дата обращения: 26.02.2021).
4. Пингвин под колпаком: Аудит системных событий в линукс // Хакер. URL: https://xakep.ru/2011/03/30/54897/ (дата обращения: 26.02.2021).

5. UNDERSTANDING AUDIT LOG FILES. RHEL. Security Guide. URL: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/security_guide/sec-understanding_audit_log_files, (дата обращения: 26.02.2021).
6. MAN Aureport. Команды системного администрирования. URL: https://www.opennet.ru/man.shtml?topic=aureport&category=8&russian=0 (дата обращения: 26.02.2021).

References

1. *Zashchita ot nesanktsionirovannogo dostupa k informatsii. Chast' 1. Programmnoe obespechenie sredstv zashchity informatsii. Klassifikatsiya po urovnyu kontrolya otsutstviya nedeklarirovannykh vozmozhnostey: rukovodyashchiy document = Protection against unauthorized access to information. Part 1. Information security software. Classification by level of control of the absence of undeclared opportunities : guidance document*. Moscow: Gostekhkomissiya Rossii, 1999. (In Russ.). Available at: https://fstec.ru/tekhnicheskaya-zashchita-informatsii/dokumenty/114-spetsialnye-normativnye-dokumenty/382-rukovodyashchij-dokument-prikaz-predsedatelya-gostekhkomissii-rossii-ot-4-iyunya-1999-g-n-114 (accessed 26.02.2021).
2. *Sborka upakovshchika Cmake-om ili trenirovka… = Cmake Packer assembly or training…* (In Russ.). Available at: https://habr.com/ru/post/178839/ (accessed 26.02.2021).
3. *MAN gcc (FreeBSD i Linux)*. Available at: https://www.opennet.ru/man.shtml?topic=gcc&russian=0&category=&submit=%F0%CF%CB%C1%DA%C1%D4%D8+man (accessed 26.02.2021).
4. Penguin under the hood: Audit of system events in Linux. *Khaker = Hacker*. (In Russ.). Available at: https://xakep.ru/2011/03/30/54897/ (accessed 26.02.2021).
5. *UNDERSTANDING AUDIT LOG FILES. RHEL*. Security Guide. Available at: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/security_guide/sec-understanding_audit_log_files, (accessed 26.02.2021).
6. *MAN Aureport. Komandy sistemnogo administrirovaniya = MAN Aureport. System Administration Commands*. (In Russ.). Available at: https://www.opennet.ru/man.shtml?topic=aureport&category=8&russian=0 (accessed 26.02.2021).

Информация об авторах / Information about the authors

Владимир Владимирович Самаров
заместитель начальника испытательной лаборатории,
ООО «16 НИИЦ»
(Россия, Московская обл., г. Мытищи,
Олимпийский просп., 29, вл. 2, 7А-4)
E-mail: samarov_vladimir@mail.ru

Vladimir V. Samarov
deputy head of test laboratory,
LLC "16 NIITS "
(7A-4, 2, 29 Olimpiyskiy avenue, Mytischi,
Moscow region, Russia)