УДК: 004.052.44; 004.054 DOI: 10.53816/20753608_2024_1_72

МЕТОДИКА ПОРТИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ В РЕЖИМ БЕЗОПАСНЫХ ВЫЧИСЛЕНИЙ АППАРАТНО-ПРОГРАММНОЙ ПЛАТФОРМЫ «ЭЛЬБРУС»

THE METHODOLOGY FOR PORTING A SOFTWARE INTO SECURE COMPUTING MODE OF «ELBRUS» HARDWARE AND SOFTWARE PLATFORM

По представлению академика РАРАН В.С. Ивановского

А.В. Морозов, Г.Е. Панамарев

Военный инновационный технополис «ЭРА»

A.V. Morozov, G.E. Panamarev

В статье представлено научное осмысление результатов прикладных исследований по созданию системы повышения доверия к программному обеспечению на основе аппаратных средств детального контроля. Ключевым элементом данной системы является методика портирования программного обеспечения в режим безопасных вычислений аппаратно-программной платформы «Эльбрус». Представлены методические аспекты механизма портирования, описан процесс его реализации на основе проведенных практических исследований. Методика формализует процесс портирования и позволяет рассматривать его как элемент технологического цикла.
Ключевые слова: режим безопасных вычислений, аппаратно-программная платформа «Эльбрус», портирование, архитектура E2K, дескрипторы, аппаратные средства детального контроля, методика портирования.

Paper presents scientific analysis of applied research on development of a system for software trustworthiness elevation based on hardware for detailed inspection. The key element of this system is the methodology for porting a software into secure computing mode of “Elbrus” hardware and software platform. Methodological aspects of porting mechanics are presented; the process of its implementation developed from practical studies is described. Presented methodology formalizes the porting process and allows including it into technological cycle.
Keywords: secure computing mode, «Elbrus» hardware and software platform, software porting, E2K architecture, descriptors, hardware for detailed inspection, porting methodology.

Введение

Процессоры семейства «Эльбрус» — это архитектурное решение, позволяющее эффективно выполнять вычислительные задачи за счет комплекса устройств, формирующих широкую команду ELBRUS (ExpLicit Basic Resources Utilization Scheduling, явное планирование использования основных ресурсов). Каждый процессор поддерживает явную спекулятивность, что повышает скорость выполнения команд при параллельных вычислениях. В арсенале Эльбруса имеется три аппаратных стека: пользовательский — для хранения данных пользователя; стек цепочек возврата — для хранения адресов возвратов функций; и стек регистров, содержащий параметры, через которые передаются адреса. Использование данной структуры позволяет повысить защищенность аппаратно-программной платформы по сравнению с зарубежными

аналогами за счет недопустимости модификации пользователем адресов перехода [1, 2].

Процессоры семейства «Эльбрус» функционируют как в обычном, так и в режиме безопасных вычислений. В случае режима безопасных вычислений используется расширенный набор команд процессора. Применение режима безопасных вычислений строится на основе аппаратной поддержки доступа к объектам исключительно через дескрипторы.

При этом эффективность системы достигается не только за счет архитектуры микропроцессора, но и за счет оптимизирующего компилятора. Процессор использует архитектуру «широкого командного слова», компилятор формирует группы команд, в которых отсутствуют взаимосвязи между элементами внутри группы. Высокий уровень параллелизма обеспечивается за счет параллельного исполнения групп команд [3].

Аппаратно-программная платформа (АПП) «Эльбрус» — это сложный уникальный инженерный комплекс, потенциал которого требует детальной проработки. Цель научной работы — формирование ключевых составляющих методики портирования программного обеспечения в режим безопасных вычислений и описание процесса ее реализации на основе результатов практических исследований.

Основная часть

Режим безопасных вычислений (РБВ) требует более строгого контроля типов данных. Там, где стандартом языка Си декларируется неопределенность поведения, РБВ в большинстве случаев вызовет аппаратное прерывание. Кроме того, РБВ накладывает определенные ограничения на используемый код, но в то же время увеличивает его защищенность. В качестве распространенных конструкций языка Си с неопределенным поведением, являющимися недопустимыми для РБВ, выступают [4]:

1. Отсутствие операции возврата (*return*) в функции, выдающей результат;
2. Частично-инициализированные буферы;
3. Использование неинициализированных переменных и структур;
4. Модификация указателей.

Существует ряд особенностей режима безопасных вычислений. Ключевой является использование дескрипторов. Дескриптор — это структура, содержащая базовый адрес, смещение от базового адреса и размер выделенной области. В РБВ к данным добавляются специальные биты, определяющие тип этих данных, — теги. При этом аппаратно поддерживаются теги размерами два бита на слово и четыре бита на двойное слово, а при приведении дескриптора к целочисленному типу данных (например, *unsigned long*) происходит преобразование в указатель. Обратная операция получения дескриптора из указателя вызовет аппаратное прерывание. Для портирования программного обеспечения (ПО) в РБВ архитектуры E2K АПП «Эльбрус» определено семь этапов [5, 6]:

1. Загрузка исходного кода программы;
2. Конфигурация исходного кода программы;
3. Проверка параметров сборки программы;
4. Сборка программы;
5. Контрольный запуск;
6. Тестирование программы;
7. Формирование протокола портирования программы.

К типовым действиям, выполняемым во время портирования ПО в РБВ архитектуры E2K АПП «Эльбрус», отнесем следующие [7, 8]:

– загрузка исходного кода портируемого ПО и выполнение сбора исходной информации об объекте портирования: определение перечня зависимостей и используемых библиотек;

– выполнение конфигурации исходного кода программы (в программах ОС Linux автоматизация выполнения процесса конфигурации осуществляется с помощью скриптов, поставляемых с исходным кодом проекта);

– задание требуемых параметров (в т.ч.: место установки, место расположения используемых библиотек) и флагов (выбор режима исполнения, оптимизация кода, добавление отладочной информации);

– проведение сборки программы (выполняется при помощи сгенерированных на этапе конфигурации сборочных скриптов);

– проведение контрольного запуска программы (открытие исполняемого файла портируемого ПО в ранее указанную директорию, в случае возникновения ошибки необходимо заново задать требуемые параметры);

– выполнение тестирования программы (исполнение набора автоматических тестов

прилагаемых к исходным кодам программы, если возникают сбои во время тестирования определить количество ошибок — их причину и внести корректирующие изменения в исходный код программы, тестирование должно быть выполнено как минимум для каждого метода и процедуры в коде портируемого ПО).

В результате каждое используемое программное средство и его отдельные модули должны быть испытаны. Эти испытания должны показать, что каждый модуль выполняет предназначенную ему функцию и не выполняет непредназначенных функций.

Также программные средства должны пройти испытания на отсутствие компьютерных вирусов. После чего необходимо проанализировать полученные результаты с точки зрения полноты портирования ПО в РБВ архитектуры Е2К АПП «Эльбрус» и запротоколировать информацию о используемых при портировании параметрах, флагах и внесенных изменениях в исходный программный код.

Испытания должны проводиться в нормальных климатических условиях в соответствии с ГОСТ 22261-94. В процессе их проведения присутствуют ресурсные ограничения по необходимым для сборки ПО библиотекам и ограничения по времени проведения испытаний, так как процесс сборки и отладки функционала занимает длительное время. Исходя из данных ограничений формируется алгоритм проведения испытаний.

Первый шаг, загрузка исходного кода портируемого ПО и проверка наличия вредоносного ПО с помощью антивируса. Сравниваются контрольные суммы скачанного архива с указанными в описании на официальной странице проекта. Вычисляется контрольная сумма архива по алгоритму *SHA1* при помощи утилиты *Rhash* (представлено на рис. 1). В случае отсутствия предупреждений антивирусного ПО и идентичности контрольных сумм возможно дальнейшее портирование ПО в РБВ архитектуры Е2К [9].

Второй шаг, конфигурация исходного кода портируемого ПО. Директория утилиты содержит скрипты автоматизации преобразования файлов *autogen.sh* и *configure*. Содержимое файла *configure* представлено на рис. 2.

```
user@localhost:~/serducov/RHash-1.4.2$ ./rhash --sha1 '/home/user/serducov/
gnupg-1.4.23.tar.bz2'
13747486ed5ff707f796f34f50f4c3085c3a6875  /home/user/serducov/gnupg-1.4.23.
tar.bz2
user@localhost:~/serducov/RHash-1.4.2$
```

Рис. 1. Вычисление контрольной суммы архива GnuPG

```
      ac_precious_vars='build_alias
host_alias
target_alias
CC
CFLAGS
LDFLAGS
LIBS
CPPFLAGS
CPP
CC_FOR_BUILD
CCAS
CCASFLAGS'


# Initialize some variables set by options.
ac_init_help=
ac_init_version=false
ac_unrecognized_opts=
ac_unrecognized_sep=
# The variables have the same names as the options, with
# dashes changed to underlines.
cache_file=/dev/null
exec_prefix=NONE
no_create=
no_recursion=
prefix=NONE
```

Рис. 2. Содержимое файла configure

На рис. 3 отображен вызов скрипта *configure* со следующими параметрами:

prefix — указывает директорию установки утилиты;

CC = lcc — использование выбранного компилятора языка Си;

CPP=l++ — использование выбранного компилятора языка Си++;

mptr128 — использование защищенного режима исполнения архитектуры Е2К;

g — создание отладочной информации;

O0 — выбор первого уровня оптимизации кода.

Третий шаг, осуществление проверки параметров сборки программы. В некоторых случаях указанные на этапе конфигурации параметры сборки программы могут быть проигнорированы скриптами автоматизации преобразования файлов. Во избежание ошибок, вызванных по данной причине, необходимо проанализировать содержимое сгенерированных *Makefile*-ов на предмет соответствия заданным на этапе конфигурации параметров [10]. В случае нахождения несоответствия, необходимо заменить параметры на требуемые вручную. Пример некорректных параметров представлен на рис. 4.

На этапе конфигурации был указан флаг использования режима безопасных вычислений *-mptr128*, следовательно, при сборке программы должны быть использованы 128-разрядные библиотеки *lib128*. По неизвестной причине скрипт *configure* проигнорировал требуемый параметр и использовал 64-разрядную библиотеку *lib64*. Данное несоответствие исправлено вручную.

Четвертый шаг, проведение сборки программы. Сборка ПО выполняется при помощи созданных на этапе конфигурации сборочных скриптов [11]. Если скрипт для автоматизированной сборки содержится в файле формата *Makefile*, то используется утилита *make*. Рекомендуется использовать утилиту *make* с правами администратора, т.е. применять команду *sudo make*, поскольку в некоторых случаях при сборке программы возникает ошибка, связанная с отсутствием у пользователя прав доступа к используемым директориям. После успешного исполнения команды *make* следует использовать команду *sudo make install* для установки ПО.

Сборка проекта может прерваться при возникновении следующих ошибок:

– отсутствие необходимых библиотек в РБВ;

– ошибка в исходном коде программы.

В случае возникновения ошибки отсутствия библиотеки в РБВ автоматизированный скрипт сообщит наименование требуемой библиотеки. Для портирования библиотеки в РБВ необходимо выполнить последовательность действий, идентичную портированию утилит:

1. Загрузить исходный код библиотеки с официального сайта;

2. Выполнить конфигурацию исходного кода;

3. Проверить параметры сборки и собрать библиотеку.

Существует вероятность, что портируемая библиотека потребует для сборки другие библиотеки в РБВ. В этом случае необходимо портировать все зависимые библиотеки, пока не будут удовлетворены требования сборки корневой библиотеки.

В случае возникновении ошибки в исходном коде программы на этапе сборки компилятор сообщит о файле проекта, прервавшем сборку (рис. 5). Для устранения ошибки необходимо проанализировать код прервавшего сборку файла.

Пятый шаг, произведение контрольного запуска программы. После успешного завершения этапа сборки ПО появляется

Рис. 3. Реализация конфигурации файлов в проекте GnuPG

```
GPGRT_CONFIG = /usr/bin/gpgrt-config --libdir=/usr/lib64
GPG_ERROR_CFLAGS =
GPG_ERROR_CONFIG = /usr/bin/gpgrt-config --libdir=/usr/lib64 gpg-error
```

Рис. 4. Некорректные параметры Makefile

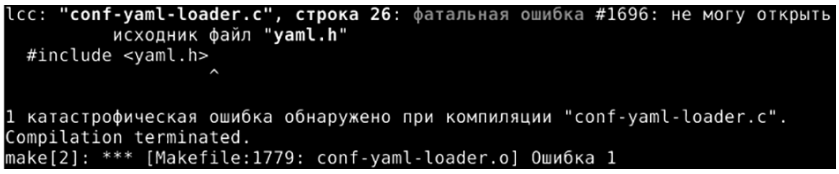

Рис. 5. Пример ошибки на этапе сборки проекта

возможность приступить к непосредственному запуску исполняемого файла проекта. Для этого необходимо запустить исполняемый файл с названием проекта, находящийся в директории по пути, установленном параметром *prefix* на этапе сборки программы. Контрольный запуск производится с минимально возможным количеством параметров и считается успешным, если не возникло каких-либо ошибок исполнения. В случае возникновения ошибки будет вызвано аппаратное прерывание и пользователь получит сообщение об ошибке. В некоторых случаях сообщение будет содержать код ошибки и место вызова прерывания, в случае отсутствия необходимой информации следует использовать стек вызовов и средства трассировки для локализации ошибок. Необходимо проанализировать код файлов, которые вызвали ошибку, и внести в них соответствующие изменения. Результат контрольного запуска утилиты *GnuPG* с параметрами *--gen-key* представлен на рис. 6.

В результате выполнения контрольного запуска возникла ошибка использование недопустимой инструкции. Поскольку вызываемая программа не предоставила сведения о возможной причине ошибки, следует произвести локализацию ошибок при помощи стандартной утилиты-отладчика *GDB*. Получить отладочную информацию позволил заданный на этапе конфигурации флаг *-g*. Для этого выполнен запуск исполняемого файла *GnuPG* с использованием утилиты *GDB* командой *gdb./gpg*. Далее, произведено исполнение файла командой *r --gen-key*. При исполнении кода с отладчиком получено сообщение *Program recieved signal SIGILL*, *Illegal instruction*. Также получена информация о строке исходного кода, при исполнении которой возникла ошибка. Для локализации ошибки использован стек вызовов с использованием инструкции *bt* (рис. 7).

Согласно информации, предоставленной стеком вызова, падение программы произошло при выполнении метода *mpi_set_cond* файла *mpiutil.c*. В строке кода, на которой произошло аппаратное прерывание, используются структуры *w* и *u*. Наиболее распространенной ошибкой при портировании в РБВ архитектуры E2K являются неинициализированные переменные. Можно предположить, что причиной падения является одна из приведенных выше структур. Фрагмент кода программы, в котором возникла ошибка недопустимой инструкции, представлен на рис. 8.

Ошибка возникла на строке *x = mask & (w->nbits ^ u->nbits)*. Согласно коду метода *mpi_set_cond*, в нем не происходит инициализация структур *w* и *u*, метод получает структуры *w* и *u* типа *MPI* в качестве входных параметров. Для определения места инициализации рассматриваемых структур принято решение исследовать следующий элемент в стеке вызовов — метод *mpi_powm*. При анализе кода метода *mpi_powm*, фрагмент которого представлен на рис. 9, обнаружена инициализация элементов структур *w* и *u*.

В приведенном коде нулевыми значениями инициализированы следующие элементы структур *w* и *u*: *sign, flags*. Инициализация элементов *nbits*, используемых в методе *mpi_set_cond*, отсутствует. Фрагмент кода метода *mpi_powm* с внесенными исправлениями представлен на рис. 10.

После исправления найденной ошибки необходимо повторно осуществить процесс сборки и

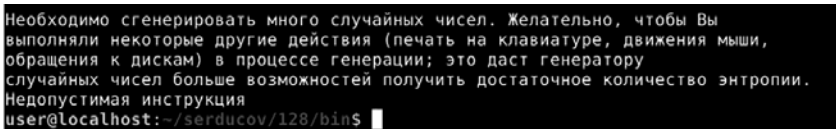

Рис. 6. Пример ошибки при контрольном запуске GnuPG

выполнить контрольный запуск. В случае возникновения прерывания требуется вернуться к поиску и исправлению ошибки в коде проекта. Также проблему неинициализированных полей структур можно решить сразу при объявлении структуры, «обнулив» ее. Для этого используется функция *memset(*u,0,size(u))*, где *u* — необходимая структура. В результате все поля структуры *u* станут равны значению 0.

Шестой шаг, выполнение тестирования программы. В большинстве случаев в исходных файлах проектов портируемых программ содержатся автоматические тесты. Тесты находятся в директориях под названием *check*, *tests* и вызываются при помощи утилиты *make* командой *sudo make check*. Скрипты автоматизированного тестирования предоставят информацию о пройденных тестах. В большинстве случаев

```
Program received signal SIGILL, Illegal instruction

exc_illegal_operand at 0x5055f5d0 ALS0

0x0000000005055f5d0 in mpi_set_cond (w=0xc25fffffcdc0, u=0xc25fffffcdf0, set=0)
    at mpiutil.c:455
455         x = mask & (w->nbits ^ u->nbits);
(gdb) bt
#0  0x0000000005055f5d0 in mpi_set_cond (w=0xc25fffffcdc0, u=0xc25fffffcdf0,
    set=0) at mpiutil.c:455
#1  0x0000000005044360 in mpi_powm (res=0x50f72620, base=0x50f72500,
    expo=0x50f72660, mod=0x50f726a0) at mpi-pow.c:627
#2  0x0000000004c50c0 in gen_prime (nbits=1024, secret=1, randomlevel=2)
    at primegen.c:368
#3  0x0000000004c1200 in generate_secret_prime (nbits=1024) at primegen.c:70
#4  0x0000000004bbab8 in generate (sk=0xc25fffffd060, nbits=2048)
    at rsa.c:122
#5  0x0000000004bf018 in rsa_generate (algo=1, nbits=2048,
    skey=0xc25fffffd200, retfactors=0xc25fffffd260) at rsa.c:405
#6  0x00000000050430ed8 in pubkey_generate (algo=1, nbits=2048,
    skey=0xc25fffffd200, retfactors=0xc25fffffd260) at pubkey.c:437
#7  0x0000000003cea80 in gen_rsa (algo=1, nbits=2048, pub_root=0x50f71600,
    sec_root=0x50f71670, dek=0x5065f630, s2k=0x5065f590,
    ret_sk=0xc25fffffd4d0, timestamp=1650441491, expireval=0, is_subkey=0)
    at keygen.c:1272
#8  0x0000000003d3a10 in do_create (algo=1, nbits=2048, pub_root=0x50f71600,
    sec_root=0x50f71670, dek=0x5065f630, s2k=0x5065f590, sk=0xc25fffffd4d0,
    timestamp=1650441491, expiredate=0, is_subkey=0) at keygen.c:2046
#9  0x0000000003e3cd0 in do_generate_keypair (para=0x50f71410,
    outctrl=0xc25fffffd670, card=0) at keygen.c:3131
#10 0x0000000003e09c8 in proc_parameter_file (para=0x50f71410,
--Type <RET> for more, q to quit, c to continue without paging--
```

Рис. 7. Стек вызовов в отладчике GDB

```
void
mpi_set_cond( MPI w, MPI u, unsigned long set)
{
    mpi_size_t i;
    mpi_size_t nlimbs = u->alloced;
    mpi_limb_t mask = ((mpi_limb_t)0) - !!set;
    mpi_limb_t x;

    if (w->alloced != u->alloced)
      log_bug ("mpi_set_cond: different sizes\n");

    for (i = 0; i < nlimbs; i++)
      {
        x = mask & (w->d[i] ^ u->d[i]);
        w->d[i] = w->d[i] ^ x;
      }

    x = mask & (w->nlimbs ^ u->nlimbs);
    w->nlimbs = w->nlimbs ^ x;

    x = mask & (w->nbits ^ u->nbits);
    w->nbits = w->nbits ^ x;

    x = mask & (w->sign ^ u->sign);
    w->sign = w->sign ^ x;
}
```

Рис. 8. Фрагмент кода метода mpi_set_cond

```
int c0;
mpi_limb_t e0;
struct gcry_mpi w, u;
w.sign = u.sign = 0;
w.flags = u.flags = 0;
w.nbits = u.nbits = 0;
w.d = base_u;
```

Рис. 9. Исходный код инициализации структур w и u

```
int c0;
mpi_limb_t e0;
struct gcry_mpi w, u;
w.sign = u.sign = 0;
w.flags = u.flags = 0;
w.d = base_u;
```

Рис. 10. Измененный код инициализации структур w и u

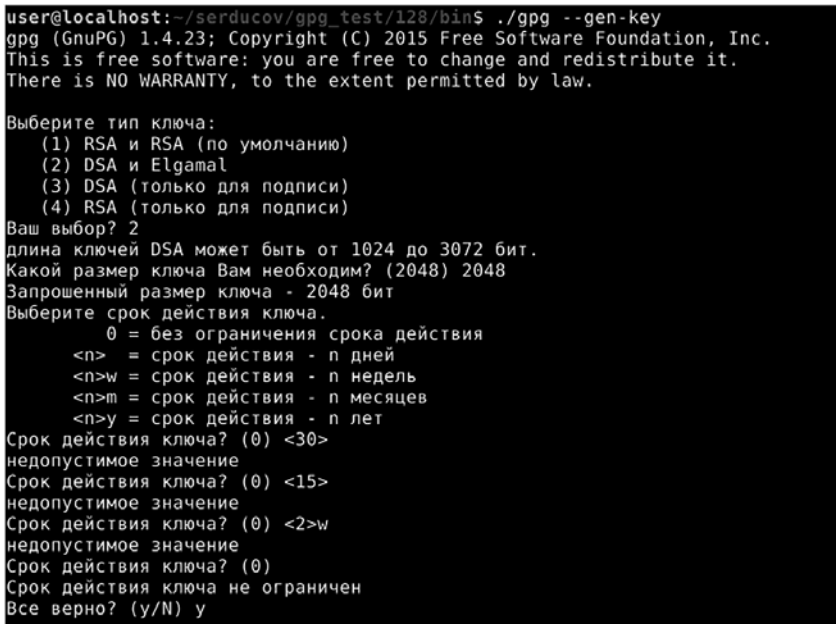

Рис. 11. Проверка функционирования GnuPG

автотесты проверяют соответствие входных и выходных данных по алгоритмам, разработанными, авторами портируемого ПО, поэтому в случае провала теста следует искать ошибку не в коде тестов, а в исходном коде портируемой программы.

Помимо использования поставляемых с ПО автотестов следует использовать модульное тестирование программы, например с применением библиотеки языка Си *assert.h*. Тестирование должно быть выполнено как минимум для каждой процедуры и функции в коде портируемого программного обеспечения.

После программного тестирования осуществляется тестирование функционала ПО. С полным перечнем параметров и команд программы можно ознакомиться при вызове команды *--help (-h)* или в текстовом файле *Readme*, входящем в состав проекта. Пример проверки функционала утилиты *GnuPG* приведен на рис. 11.

Портирование можно проводить как для программных пакетов, так и для отдельных библиотек. Процесс сборки крупных проектов может занимать довольно длительное время. В случае пересборки процесс идет быстрее, поскольку утилита-сборщик взаимодействует только с измененными файлами и их зависимостями. Если процесс пересборки программы завершается некорректно, то следует очистить директорию командой *make clean*.

В документирование испытаний формируется протокол портирования ПО в режим безопасных вычислений архитектуры Е2К аппаратно-программной платформы «Эльбрус». В данном протоколе отображаются следующие этапы: проверка исходного кода портируемого ПО на наличие вредоносного, вирусного ПО; проверка контрольных сумм; конфигурация проекта; сборка проекта; тестирование работоспособности.

В целях оптимизации процесса переноса портированного кода на другие версии машин с архитектурой Е2К, либо исправления текущих сборок в протокол портирования заносится информация о всех флагах и параметрах, используемых на стадии конфигурации проекта, а также на стадии проверки параметров сборки.

При использовании на этапе тестирования работоспособности портированного ПО модульных тестов необходимо предоставлять их исходный код, а также документировать используемые при тестировании входные и выходные данные.

В случае возникновения ошибки при портировании ПО необходимо зафиксировать в протоколе входные данные и последовательность действий, совершенных пользователем, приведших к некорректной работе ПО. Полученная информация должна быть использована для локализации и устранения скрытых ошибок при работе с архитектурой Е2К.

Заключение

Резюмируя результаты исследования, следует отметить, что РБВ — одна из ключевых возможностей, заложенных в архитектуру Е2К АПП «Эльбрус»; это особый режим работы процессора «Эльбрус», в котором аппаратура контролирует доступ в память не только к отдельным страницам, но и к отдельным объектам программы, что позволяет не допускать эксплуатацию злоумышленниками ряда уязвимостей, в том числе исключать «переполнение буфера». При этом РБВ значительно ускоряет процесс отладки разрабатываемого ПО за счет того, что аппаратура процессора сама детектирует ряд ошибок и прерывает работу программы в момент возникновения ошибки (нарушения правил доступа к объектам в памяти), а не в момент проявления последствий этой ошибки.

Предлагаемая методика позволяет формализовать процесс портирования ПО в РБВ и реализовать на основе АПП «Эльбрус» систему детального контроля над работой ПО на аппаратном уровне.

Литература

1. Морозов А.В., Панамарев Г.Е. Исследование технологий повышения доверия к специальному программному обеспечению с применением инструментальных средств, реализующих фаззинг — тестирование // Известия Российской академии ракетных и артиллерийских наук. 2022. № 3 (123). С. 129–136.

2. Панамарев Г.Е. Исследование новых подходов к разработке методов обнаружения уязвимостей в автоматизированных системах военного назначения // Военная мысль. 2023. № 3. С. 82–89.

3. Русяев Р.М., Нейманзаде М.И., Ермолицкий А.В., Волконский В.Ю. Программно-аппаратные средства выявления ошибок обращения к памяти для архитектуры «Эльбрус» // Вопросы радиоэлектроники. 2017. № 3. С. 33–38.

4. Григорьев П.В., Киржаев Д.А. Анализ недопустимых конструкций на языке Си при портировании программного обеспечения в защищенный режим исполнения на базе процессора «Эльбрус» / Сборник трудов II Всероссийской научно-технической конференции «Состояние и перспективы развития современной науки по направлению ИТ-технологии». Анапа, 2023. С. 34–50.

5. Девянин П.Н., Тележников В.Ю., Хорошилов А.В. Формирование методологии разработки безопасного системного программного обеспечения на примере операционных систем // Труды ИСП РАН. 2021. № 5. С. 25–40.

6. Вареница В.В., Марков А.С., Савченко В.В., Цирлов В.Л. Практические аспекты выявления уязвимостей при проведении сертификационных испытаний программных средств защиты информации // Вопросы кибербезопасности. 2021. № 5 (45). С. 36–44.

7. Панамарев Г.Е., Малинин К.С., Сердюков П.С., Леонтьев З.Д. Перенос в защищенный режим исполнения «Эльбрус» утилиты для шифрования ccrypt / Сборник статей I научно-технической конференции «Состояние и перспективы развития современной науки по направлению «ИТ-технологии». Анапа, 2022. С. 109–118.

8. Струков М.С., Григорьев П.В. Портирование в ЗРИ Е2К программы pdfcrack / Сборник трудов II Всероссийской научно-технической конференции «Состояние и перспективы развития современной науки по направлению ИТ-технологии». Анапа, 2023. С. 18–26.

9. Кондратьев Б.В., Поддубный М.И., Леонтьев З.Д. Портирование утилиты для вычисления контрольных сумм rhash в защищенный режим исполнения архитектуры Е2К / Сборник статей I научно-технической конференции «Состояние и перспективы развития современной науки по направлению «ИТ-технологии». Анапа, 2022. С. 6–16.

10. Буланый Р.И., Хиков С.П., Малинин К.С. Компиляция и компоновка программ и библиотек в защищенном режиме исполнения семейства «Эльбрус» / Сборник статей I научно-технической конференции «Состояние и перспективы развития современной науки по направлению «ИТ-технологии». Анапа, 2022. С. 91–100.

11. Кравцунов Е.М., Гисматов А.Р., Антипов А.С., Копылов Н.М. Исследование актуальных уязвимостей cve на платформе «Эльбрус» / Сборник статей конференции «Информатика и вычислительная техника». 2019. С. 176–179.