A. N. Fedotov, Method for exploitability estimation of program bugs,
Proceedings of ISP RAS, 2016, Volume 28, Issue 4, 137–148

https://www.mathnet.ru/eng/tisp57

Метод оценки эксплуатируемости программных дефектов

А.Н. Федотов <fedotoff@ispras.ru>
Институт системного программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. В статье рассматривается метод оценки эксплуатируемости программных дефектов. Применение данного подхода позволяет осуществить приоритезацию найденных ошибок в программах. Это даёт возможность разработчику программного обеспечения исправлять ошибки, которые представляют наибольшую угрозу безопасности в первую очередь. Метод представляет собой совместное применение предварительной классификации аварийных завершений и автоматическую генерацию эксплойтов. Предварительная классификация используется для фильтрации неэксплуатируемых ошибок. Если аварийное завершение считается потенциально эксплуатируемым, то выбирается соответствующий алгоритм генерации эксплойта. В случае успешной генерации эксплойта происходит проверка работоспособности посредством эксплуатации анализируемой программы в эмуляторе. Для поиска программных дефектов можно использовать различные методы. В качестве таких методов можно выделить фаззинг и активно развивающийся в настоящее время подход к поиску ошибок на основе динамического символьного выполнения. Главным требованием для использования предлагаемого метода является возможность получения входных данных, на которых проявляется найденный дефект. Разработанный подход применяется к бинарным файлам программ и не требует дополнительной отладочной информации. Реализация метода представляет собой совокупность программных средств, которые связаны между собой управляющими скриптами. Метод предварительной классификации и метод автоматической генерации эксплойтов реализованы в виде отдельных программных средств, которые могут работать независимо друг от друга. Метод был апробирован на анализе 274 аварийных завершений, полученных в результате фаззинга. В результате анализа удалось обнаружить 13 эксплуатируемых дефектов, для которых в последствии успешно сгенерированы работоспособные эксплойты.

Ключевые слова: уязвимость; переполнение буфера; символьное выполнение; эксплойт; бинарный код.

DOI: 10.15514/ISPRAS-2016-28(4)-8

Для цитирования: А.Н. Федотов. Метод оценки эксплуатирумости программных дефектов. Труды ИСП РАН, том28, вып. 4, 2016 г., стр. 137-148. DOI: 10.15514/ISPRAS-2016-28(4)-8

1. Введение

В настоящее время проблема безопасности программного обеспечения становиться всё более актуальной. Растёт количество программных продуктов, которые применяются в различных отраслях промышленности. Кроме того, повседневная жизнь человека тесно связана с использованием программ. Современные бытовые приборы, телефоны, автомобили используют в своей работе программное обеспечение, которое может иметь выход в интернет. Сбои в работе могут приводить к серьёзным последствиям, а эксплуатация уязвимостей может нанести непоправимый ущерб. Крупные корпорации, такие как Google, Apple, Microsoft, Cisco и т.д. уделяют большое внимание разработке средств, которые обеспечивают поиск дефектов, а также внедряют их в цикл разработки ПО.

Существуют различные подходы к поиску ошибок, которые применяются в цикле разработке программного обеспечения. Одним из таких подходов является фаззинг. Главным преимуществом фаззинга – является возможность получения входных данных для воспроизведения ошибки. Современные фаззеры находят достаточно большое количество аварийных завершений во время своей работы [1,2,3,4]. Кроме того, в настоящее время перспективным направлением поиска ошибок является динамическое символьное выполнение, которое также как и фаззинг способно обнаруживать ошибки и снабжать входными данными, на которых эти ошибки проявляются. Важно отметить, что эти подходы применяются к бинарному коду, что позволяет искать ошибки в программах, для которых отсутствуют исходные тексты. Для эффективного исправления ошибок разработчиками необходимо обеспечить приоритизацию найденных дефектов. Наиболее высокий приоритет следует назначить ошибкам, которые могут привести к возникновению уязвимостей, приводящих к выполнению произвольного кода. Исправление таких ошибок в первую очередь, позволит повысить безопасность исполняемого кода.

Наличие эксплойта, который активирует уязвимость – является однозначным подтверждением её присутствия в программе. Существующие подходы к автоматической генерации экслойтов, описанные в работах [5-7] позволяют без особого вмешательства аналитика получить входные данные для активации уязвимости. Кроме того, работы [5,6] помимо эксплуатации обеспечивают и поиск уязвимости. В основе этих подходов лежит динамическое символьное выполнение. При анализе большого количества дефектов, использование высокотехнологичных видов анализа, таких как динамическое символьное выполнение, влечёт за собой рост времени анализа и потребления ресурсов. Далеко не каждый дефект является уязвимостью, приводящей к выполнению произвольного кода. Поэтому, прежде чем использовать ресурсоёмкий анализ, стоит произвести легковесную предварительную фильтрацию ошибок, с целью исключить заведомо неэксплуатируемые ошибки. В качестве таких неэксплуатируемых ошибок можно выделить деление на ноль, разыменование нулевого указателя и т.д. В статье предлагается подход, который совмещает в

себе предварительную классификацию ошибок и автоматическую генерацию эксплойта. За основу предварительной классификации дефектов используется подход представленный в инструменте с открытым исходным кодом !exploitable [8]. В качестве метода генерации эксплойтов был выбран метод, описанный в работе [9]. Классифицируемый дефект будет иметь максимальный приоритет, если ошибка считается потенциально эксплуатируемой после предварительной стадии, и затем впоследствии удалось сгенерировать работоспособный эксплойт.

Статья организована следующим образом. Во втором разделе рассмотрены существующие инструменты, которые применяются для оценки эксплуатируемости ошибок. В третьем разделе представлен метод оценки эксплуатируемости программных дефектов. В четвёртом разделе приведены основные результаты применения метода. В последнем, шестом разделе рассматриваются полученные результаты и обсуждаются перспективные направления исследований.

2. Обзор существующих подходов к оценке эксплуатируемости программых дефектов

В качестве основных подходов к оценке эксплуатируемости программных дефектов можно выделить два направления: анализ аварийных завершений и автоматическая генерация эксплойта с помощью динамического символьного выполнения.

2.1 Анализ аварийных завершений

Во время анализа аварийных завершений изучается состояние программы (значения регистров, памяти, стек вызовов) на момент срабатывания исключения. К этому состоянию применяется набор правил, каждое правило которого описывает класс аварийного завершения. В качестве примера такого класса можно привести исключение, вырабатываемое процессором при попытке выполнить инструкцию по адресу, доступ к которому запрещён, а также срабатывание защиты от переполнения буфера, вовремя вызова безопасных функций копирования. Каждый класс имеет ранг, чем ниже ранг тем, тем выше вероятность, что нарушителю удастся проэксплуатировать лежащую в основе уязвимость. Кроме того, каждый класс принадлежит одной из четырёх групп: эксплуатируемые, возможно эксплуатируемые, возможно неэксплуатируемые и недостаточно изученные. В последний класс попадают аварийные завершения, при анализе которых было слишком мало информации, для того чтобы сделать выводы об эксплуатируемости данного аварийного завершения. Например, если единственное, что удалось узнать при возникновении исключения, что это исключение возникло при нарушении доступа к памяти, то основываясь только на этой информации невозможно сделать вывод об эксплуатируемости потенциальной уязвимости. Одному

аварийному завершению может соответствовать несколько классов. Вывод об эксплуатируемости производится на основе информации о классе, ранг которого минимальный.

Реализация описанного подхода представлена в инструментах [8, 10]. Инструменты являются схожими и имеют один и тот же набор правил. Инструмент [8] анализирует программы, работающие под управлением операционных систем семейства Windows, а инструмент [10] работает с Linux-программами. Одним из достоинств данного подхода стоит отметить скорость анализа. Высокая скорость работы достигается благодаря тому, что проводится анализ только состояния программы в момент аварийного завершения, а не исследуется всё выполнение программы. Как следствие, от этого могут возникнуть ложные срабатывания. Существует вероятность, что неэксплуатируемые ошибки оцениваются как эксплуатрируемые, но ошибки, распознанные как неэксплуатируемые, как правило, таковыми и являются. Поэтому данный метод хорошо подходит для фильтрации неэксплуатируемых дефектов.

2.2 Автоматическая генерация эксплойтов

Метод автоматической генерации эксплойтов позволяет получить набор входных данных, который активирует существующую уязвимость в программе. Таким образом, можно утверждать, что уязвимость является эксплутируемой. Данный метод основывается на использовании динамического символьного выполнения. Во время динамического символьного выполнения конкретные значения переменных, которые участвуют в обработке входных данных, заменяются символьными значениями. В качестве входных данных используются различные источники: аргументы командной строки, переменные окружения, файлы и сеть. Операции над символьными значениями представляются в виде формул для SMT-решателя. Каждое ветвление в программе, результат которого зависит от символьных данных, добавляет ограничение в общую систему. Это ограничение отражает прохождение управления по конкретной ветви программы. Набор таких ограничений называется предикатом пути. Предикат пути описывает одно выполнение программы. Если данный набор ограничений предоставить на вход SMT-решателю, то результатом его работы будет набор входных данных, для активации того сценария работы программы, на котором был получен набор ограничений.

Для успешной эксплуатации уязвимости, необходимо описать дополнительные ограничения, которые позволят её активировать. Формализация эксплуатации уязвимости является сама по себе отдельной сложной задачей. Для некоторых видов уязвимостей эта задача уже решена и описана в работах [5,7,9,11,12]. В качестве таких уязвимостей выступают переполнение буфера на стеке и уязвимость форматной строки.

Существующие инструменты автоматической генерации эксплойтов имеют несколько различий между собой. Инструменты [7,9,11] используют полносистемную эмуляцию, что позволяет анализировать не только пользовательские приложения, но и программы уровня ядра операционной системы, тогда как инструменты [5,6] анализируют только пользовательские программы. Инструменты [5,6] кроме эксплуатации осуществляют ещё и поиск уязвимости, но, к сожалению, эти инструменты недоступны. Важным достоинством инструментов является, то, что участие аналитика в их работе является минимальным. В качестве недостатков можно отметить значительный рост потребления ресурсов, при анализе больших программ. Таким образом, важно обеспечить правильную предварительную фильтрацию и классификацию дефектов, для обеспечения производительности.

3. Разработанный метод

Разработанный метод представляет собой совместное применение анализа аварийных завершений и автоматической генерации эксплойта. Метод разделён на три этапа (рис.1).

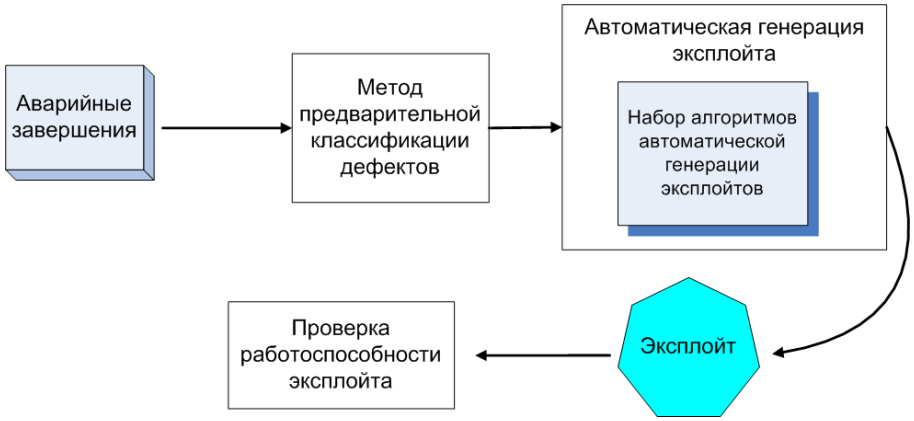

Рис. 1. Декомпозиция метода на три этапа

Fig. 1. The method decomposition into three stages

Анализ аварийных завершений применяется для осуществления предварительной классификации ошибок, с целью отфильтровать заранее неэксплуатируемые дефекты, такие как деление на ноль, разыменование нулевого указателя, срабатывание защиты во время копирования с использованием безопасных функций. Кроме того, предварительная классификация позволяет выбрать алгоритм для дальнейшей генерации эксплойта. Например, если после анализа аварийных завершений стало известно, что произошло исключение при попытке выполнить инструкцию по адресу, доступ к которому запрещён, а также существует повреждение стека, то для генерации эксплойта следует запустить алгоритм генерации эксплойта для

уязвимости переполнения буфера на стеке. После успешной генерации следует этап проверки работоспособности эксплойта. В случае успешной работы экспойта, можно однозначно утверждать о наличии эксплуатируемого дефекта в программе.

За основу метода предварительной классификации был выбран подход, реализованный в инструментах [8,10]. Для успешной интеграции метода потребовалось переработать ранги и группы для некоторых классов аварийных завершений. Так как в реализации инструмента некоторые классы аварийных завершений не подлежат эксплуатации в рамках текущей реализации автоматической эксплуатации. В качестве примера можно привести классы аварийных завершений, связанных переполнением буфера на куче. В настоящий момент автоматическая эксплуатация уязвимости переполнения буфера на куче отсутствует, поэтому эти классы перенесены в группу возможно не эксплуатируемых аварийных завершений. Также в эту группу перенесён класс, который отвечает за срабатывание защитных механизмов: таких как "канарейка" и безопасные функции работы со строками. Классы, отвечающие за исключения, возникающие при чтении из памяти или записи в память, перенесены в группу возможно эксплуатируемых, т.к. реализован подход к исправлению перезаписанных указателей во время переполнения буфера.

Проблема перезаписанных указателей возникает, если после перезаписи указателя осуществляется к нему доступ, в результате которого возникает исключение, которое, в свою очередь приводит к досрочному аварийному завершению. Технология исправления перезаписанных указателей базируется на методе, который описан в работе [9]. Этот метод использует динамическое символьное выполнение по бинарным трассам программ. В предлагаемом подходе происходит построение предиката пути от точки получения входных данных до точки срабатывания исключения доступа к памяти. Кроме этого добавляется ограничение на то, что указатель равен адресу, доступ по которому не вызывает исключения. Примером такого адреса может быть адрес из диапазона адресов секции данных анализируемого исполняемого файла. Результатом решения получается набор входных данных, который проходит по тому же пути и не вызывает срабатывания исключения. Затем этот набор входных данных вновь подвергается предварительной классификации, в результате которой возможны три случая. Первый случай, когда аварийное завершение классифицируется как эксплуатируемое, например, переход по неправильному адресу после выполнения инструкции передачи управления. В таком случае, будет происходить генерация эксплойта с учётом предыдущих ограничений на указатели. Второй случай возникает, если ещё срабатывает исключение при доступе к другому указателю. В этом случае процесс с исправлением указателя повторяется. Третий случай возникает, тогда, когда после исправления указателя аварийное завершение классифицируется как возможно неэксплуатируемое. Таким образом, обеспечивается исправление перезаписанных указателей. Также стоит отметить, что существуют класс

аварийных завершений, при котором перезаписан указатель стека в момент вызова инструкции возврата из функции. Исправление этой ситуации идентично с подходом к исправлению перезаписанных указателей. В качестве значения указателя стека используется значение, которое было на момент вызова текущей функции.

Применение вышеописанных методов позволяет произвести оценку эксплуатируемости программных дефектов.

4. Реализация метода и результаты практического применения

Предложенный метод был реализован в виде совместного использования нескольких программных средств. Метод предварительной классификации был реализован на основе утилиты gdb_exploitable [10]. Для автоматической генерации эксплойта был использован модуль-расширение из среды анализа бинарного кода[9]. Для проверки работоспособности использовался полносистемный эмулятор Qemu[13]. Механизм обеспечения совместной работы всех программных средств представлен в виде скриптов на языке Python.

Для получения набора аварийных завершений был использован подход, описанный в работе [5]. На основе этого подхода был реализован фаззер утилит, которые используют интерфейс командной строки. В качестве объекта для фаззинга были выбраны приложения из дистрибутива Debian6.0.10. В результате фаззинга были обнаружены 274 приложения, для которых есть входные данные, приводящие к аварийному завершению. Для этих приложений проводилась оценка найденных дефектов. Результаты предварительной классификации представлены в табл.1.

Табл. 1. Результаты предварительной классификации

Table 1. Results of the preliminary classification

Группа аварийных завершений	Класс аварийных завершений	Количество аварийных завершений
эксплуатируемые	Исключение при доступе к памяти, адрес которой совпадает со счётчиком команд	13
Возможно эксплуатируемые	Переполнение буфера на куче	23
Не достаточно изученные	Нарушение доступа	238

Далее для группы эксплуатируемых аварийных завершений применялась автоматическая генерация эксплойта. Так как класс аварийных завершений

соответствует ситуации, когда происходит передача управления по перезаписанному адресу во время выхода из функции, то применялся алгоритм автоматической генерации эксплойта для переполнения буфера на стеке. Для данных аварийных завершений были сгенерированы эксплойты. К сожалению, для успешной эксплуатации уязвимостей пришлось отключить рандомизацию адресного пространства, защита выполнения данных была отключена по умолчанию.

Так как нет алгоритма генерации эксплойтов для переполнения буфера на куче, данный класс аварийных завершений относится к группе возможно неэксплуатируемых дефектов. Для большинства аварийных завершений, единственной информацией, которую удалось извлечь, оказалась информация о том, что произошло только нарушение доступа. Основываясь только на этой информации невозможно сделать вывод о эксплуатируемости данного дефекта.

На тестирование всего набора примеров было потрачено примерно 10 часов. Среднее время проведения предварительной эксплуатации составило 1 минуту. Среднее время генерации эксплойта с последующей проверкой составило 21 минуту. Таким образом, использование предварительной классификации дефектов позволяет существенно сократить время, потраченное на оценку эксплуатируемости дефектов.

5. Заключение

В статье представлен метод оценки эксплуатируемости программных дефектов. Метод основан на совместном использовании предварительной классификации аварийных завершений и автоматической генерации эксплойта. Реализация метода представляет собой набор программных средств связанных между собой управляющими скриптами. Данный метод применяется непосредственно к исполняемым файлам. Применение этого метода позволяет разработчикам программного обеспечения понять, какие ошибки представляют наибольшую угрозу для безопасности ПО. Предварительная классификация позволяет отсеять неэксплуатируемые дефекты, а для потенциально эксплуатируемых запустить нужный алгоритм автоматической генерации эксплойтов. Представленные в данной работе результаты предлагают законченный метод, позволяющий оценивать эксплуатируемость найденных ошибок.

В качестве дальнейших работ стоит выделить автоматическую генерацию для других типов программных дефектов, например переполнение буфера на куче. В современных аллокаторах из библиотеки glibc, присутствует большое количество разного рода проверок, которые значительно усложняют эксплуатацию уязвимостей переполнения буфера на куче. Также отдельный интерес представляют ситуации, когда нарушитель может контролировать адрес, по которому происходит запись и данные, которые будут записаны по контролируемому адресу. Такие ситуации открывают массу возможностей для нарушителя: перезапись адреса возврата из функции или указателя на функцию

адресом в памяти, где располагается код полезной нагрузки. Эксплуатация такого рода уязвимостей позволяет обойти защитный механизм "канарейка", который призван защищать адрес возврата из функции от перезаписи во время переполнения буфера на стеке. Важным направлением исследования, является эксплуатация уязвимостей в рамках работы современных защитных механизмов уровня операционной системы. В качестве таких защит можно выделить DEP и ASLR. Для эксплуатации уязвимостей при включённых защитах следует использовать ROP- компиляцию, которая описана в работе [14].

Список литературы

[1]. Miller C. et al. Crash analysis with BitBlaze. At BlackHat USA, 2010.
[2]. American fuzzy lop fuzer. URL: http://lcamtuf.coredump.cx/afl//.
[3]. Peach fuzzer. URL: http://www.peachfuzzer.com/
[4]. Codenomicon fuzzer. URL: http://www.codenomicon.com/
[5]. T. Avgerinos, S. K. Cha, Alexandre Rebert, Edard J. Schwartz, Maverick Woo, and D.Brumley. AEG: Automatic exploit generation. *Commun. ACM,* 2014, №2.
[6]. Sang Kil Cha, Thanassis Avgerinos, Alexandre Rebert and David Brumley. Unleashing MAYHEM on Binary Code. IEEE Symposium on Security and Privacy, 2012
[7]. Huang S. K. et al. Crax: Software crash analysis for automatic exploit generation by modeling attacks as symbolic continuations.Software Security and Reliability (SERE), 2012 IEEE Sixth International Conference on. IEEE, 2012, pp. 78-87.
[8]. Инструмент !exploitable. URL: https://msecdbg.codeplex.com/.
[9]. Падарян В. А., Каушан В. В., Федотов А. Н. Автоматизированный метод построения эксплойтов для уязвимости переполнения буфера на стеке. Труды ИСП РАН, т. 26, вып. 3, стр. 127-144. DOI: 10.15514/ISPRAS-2014-26(3)-7.
[10]. Плагин exploitable для gdb. URL: https://github.com/jfoote/exploitable**.**
[11]. Вахрушев И. А. и др. Метод поиска уязвимости форматной строки //Труды Института системного программирования РАН, т. 27, вып. 4, 2015, стр. 23-38. DOI: 10.15514/ISPRAS-2015-27(4)-2.
[12]. Heelan S. Automatic generation of control flow hijacking exploits for software vulnerabilities. Master's thesis, University of Oxford, 2009.
[13]. Эмулятор Qemu. URL: http://wiki.qemu.org/Main_Page.
[14]. Schwartz E. J., Avgerinos T., Brumley D. Q: Exploit Hardening Made Easy //USENIX Security Symposium, 2011, pp. 25-41.

Method for exploitability estimation of program bugs

A.N. Fedotov <fedotoff@ispras.ru>
Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn str., Moscow, Russia, 109004

Abstract. The method for exploitability estimation of program bugs is presented. Using this technique allows to prioritize software bugs that were found. Thus, it gives an opportunity for a developer to fix bugs, which are most security critical at first. The method is based on combining preliminary classification of program bugs and automatic exploit generation. Preliminary classification is used to filter non-exploitable software defects. For potentially exploitable bugs corresponding exploit generation algorithm is chosen. In case of a successful exploit generation the operability of exploit is checked in program emulator. There are various ways that used for finding software bugs. Fuzzing and dynamic symbolic execution are often used for this purpose. The main requirement for the use of the proposed method is an opportunity to get input data, which cause program to crash. The technique could be applied to program binaries and does not require debug information. Implementation of the method is a set of software tools, which are interconnected with control scripts. The preliminary classification method and automatic exploit generation method are implemented as stand-alone tools, and could be used separately. The technique was used to analyze 274 program crashes, which were obtained by fuzzing. The analysis managed to detect 13 exploitable bugs, for which successfully workable exploits were generated.

Keywords: vulnerability; buffer overflow; symbolic execution; exploit; binary code.

DOI: 10.15514/ISPRAS-2016-28(4)-8

For citation: A.N. Fedotov. Method for exploitability estimation of program bugs. *Trudy ISP RAN/Proc. ISP RAS*, vol. 28, issue 4, 2016. pp. 137-148 (in Russian). DOI: 10.15514/ISPRAS-2016-28(4)-8

References

[1]. Miller C. et al. Crash analysis with BitBlaze. At BlackHat USA, 2010.
[2]. American fuzzy lop fuzer. URL: http://lcamtuf.coredump.cx/afl//.
[3]. Peach fuzzer. URL: http://www.peachfuzzer.com/
[4]. Codenomicon fuzzer. URL: http://www.codenomicon.com/
[5]. T. Avgerinos, S. K. Cha, Alexandre Rebert, Edard J. Schwartz, Maverick Woo, and D.Brumley. AEG: Automatic exploit generation. *Commun. ACM,* 2014, №2.
[6]. Sang Kil Cha, Thanassis Avgerinos, Alexandre Rebert and David Brumley. Unleashing MAYHEM on Binary Code. IEEE Symposium on Security and Privacy, 2012
[7]. Huang S. K. et al. Crax: Software crash analysis for automatic exploit generation by modeling attacks as symbolic continuations.Software Security and Reliability (SERE), 2012 IEEE Sixth International Conference on. IEEE, 2012, pp. 78-87.

[8]. !exploitable. URL: https://msecdbg.codeplex.com/.

[9]. Padaryan V.A., Kaushan V.V., Fedotov A.N.[Automated exploit generaton method for stack buffer overflow vulnerabilities]. *Trudy ISP RAN/Proc. ISP RAS*, vol. 26, issue 3, 2014, pp. 127-144. DOI: 10.15514/ISPRAS-2014-26(3)-7.

[10]. Exploitable plugin for gdb. URL: https://github.com/jfoote/exploitable.

[11]. Vakhrushev I. A. et al. [Search method for format string vulnerabilities]. *Trudy ISP RAN/Proc. ISP RAS,* vol. 27, issue 4, pp. 23-38. DOI: 10.15514/ISPRAS-2015-27(4)-2.

[12]. Heelan S. Automatic generation of control flow hijacking exploits for software vulnerabilities. Master's thesis, University of Oxford, 2009.

[13]. Qemu. URL: http://wiki.qemu.org/Main_Page.

[14]. Schwartz E. J., Avgerinos T., Brumley D. Q: Exploit Hardening Made Easy //USENIX Security Symposium, pp. 25-41, 2011.