DOI: 10.15514/ISPRAS–2020–32(4)–6

CASR: анализ coredump файлов в ОС Linux и составление отчётов об ошибках

А.Н. Федотов, ORCID: 0000-0002-8838-471X <fedotoff@ispras.ru>
Ш.Ф. Курмангалеев, ORCID: 0000-0002-0558-2850 <kursh@ispras.ru>
Институт системного программирования им. В.П. Иванникова РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. Несмотря на то, что при разработке программного обеспечения используются различные технологии и подходы, позволяющие диагностировать ошибки на ранних этапах разработки и тестирования, часть ошибок обнаруживается во время эксплуатации. Для пользователя ошибки часто выглядят как аварийное завершение программы во время работы. Для сбора отчётов об аварийных завершениях программ в операционную систему встраивается специальный компонент анализа. Такой компонент присутствует как в ОС Windows, так и в ОС на базе Linux, в частности в Ubuntu. Важным параметром является степень критичности найденной ошибки, причем данная информация полезна как разработчику дистрибутива, так и пользователю. В частности, пользователи, имея такую диагностику, могут принять организационно-технические меры до выхода исправления ошибки от разработчика программного обеспечения. В статье представлен CASR: инструмент анализа образа памяти в момент завершения процесса (coredump) и составления отчётов об ошибках. Инструмент позволяет проводить оценку критичности обнаруженного аварийного завершения путём анализа образа памяти, а также собирать необходимую информацию для разработчика, которая поможет исправить дефект. В качестве такой информации выступают версия дистрибутива ОС, версия пакета, карта памяти процесса, состояние регистров, значения переменных среды, стек вызовов, номер сигнала, который привёл к аварийному завершению, и т.д. Оценка критичности даёт возможность разработчику программного обеспечения в первую очередь исправить те ошибки, которые являются наиболее опасными. CASR позволяет обнаружить файлы и сетевые соединения, которые были открыты в момент аварийного завершения. Эта информация поможет воспроизвести ошибку, а также принять меры пользователям и администраторам в случае атаки на систему. Инструмент предназначен для работы на ОС Linux, поддерживает архитектуры x86/64, armv7 и может поставляться в виде пакета для дистрибутивов на базе Debian. Инструмент был успешно протестирован на нескольких ошибках, сведения о которых были получены из открытых источников.

Ключевые слова: оценка критичности; дамп памяти; отчёты об ошибках

Для цитирования: Федотов А.Н., Курмангалеев Ш.Ф. CASR: анализ coredump файлов в ОС Linux и составление отчётов об ошибках. Труды ИСП РАН, том 32, вып. 4, 2020 г., стр. 89–96 DOI: 10.15514/ISPRAS–2020–32(4)–6

CASR: core dump analysis and severity reporter tool

A.N. Fedotov, ORCID: 0000-0002-8838-471X <fedotoff@ispras.ru>
Sh.F.Kurmangaleev, ORCID: 0000-0002-0558-2850 <kursh@ispras.ru>
Ivannikov Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. Despite the fact that software development uses various technologies and approaches to diagnose errors in the early stages of development and testing, some errors are discovered during operation. To the user, errors often look like a program crash while running. To collect reports on program crashes, a special analysis component is built into the operating system. Such a component is present in both Windows OS and Linux-based OS, in particular Ubuntu. An important parameter is the severity of the error found, and this information is useful to both the developer of the distribution kit and the user. In particular, users with such diagnostics can take organizational and technical measures before the release of a bug fix from the software developer. The article introduces CASR: a tool for analyzing a memory image at the time of a process termination (coredump) and reporting errors. The tool allows you to assess the severity of the detected crash by analyzing the memory image, as well as collect the necessary information for the developer to help fix the defect. Such information is: OS distribution version, package version, process memory card, state of registers, values of environment variables, call stack, signal number that led to abnormal termination, etc. Severity assessment enables the software developer to correct errors, which are the most dangerous in the first place. CASR can detect files and network connections that were open at the time of the crash. This information will help reproduce the error, and will help users and administrators take action in the event of an attack on the system. The tool is designed to work on Linux OS and supports x86 / 64, armv7 architectures and can be supplied as a package for Debian-based distributions. The tool has been successfully tested with several open source bugs.

Keywords: coredump; crash; error reports; critical estimation of software defects

For citation: Fedotof A.N., Kurmangaleev Sh.F. CASR: core dump analysis and severity reporter tool. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 4, 2020, pp. 89–96 (in Russian). DOI: 10.15514/ISPRAS–2020–32(4)–6

1. Введение

Безопасный цикл разработки программного обеспечения активно применяется как в России [1], так и за рубежом. Статический и динамический анализ позволяют обнаруживать ошибки, в том числе и критические ещё на стадии разработки. Тем не менее, уже в готовом программном обеспечении могут быть ошибки, которые будут обнаружены лишь на стадии эксплуатации программного продукта.

Для сбора отчётов об аварийных завершениях программ в операционную систему встраивается специальный компонент анализа. Такой компонент присутствует как в ОС Windows [2], так и в ОС на базе Linux, в частности в Ubuntu [3]. Таким образом, все обнаруженные аварийные завершения на машине пользователя могут быть проанализированы и отправлены разработчикам в Microsoft или сообществом разработчиков дистрибутива Ubuntu. Также компоненты создания и отправки отчётов об ошибках встроены в некоторое программное обеспечение напрямую, например, в браузеры или почтовые клиенты [4].

Таким образом, разработчики стороннего ПО вынуждены либо разрабатывать свои компоненты для сбора информации о сбоях в программном обеспечении, либо совсем отказаться от получения отчётов об ошибках. Кроме того, важную роль играет оценка критичности обнаруженного аварийного завершения [5]. Наиболее критичные аварийные завершения разработчики будут стараться исправить в первую очередь, а пользователи примут меры, чтобы обеспечить защиту своей системы, пока не выйдет исправление.

В работе рассматривается casr – инструмент анализа образа памяти в момент завершения процесса (coredump) в системах ОС Linux, который позволяет проводить оценку критичности найденного дефекта, а также собирать информацию, которая поможет разработчику при анализе аварийного завершения. Метод был протестирован на нескольких известных уязвимостях в системе Astra Linux 1.6 «Орёл».

2. Обзор схожих работ

2.1 Windows Error Reporting (WER)

WER [6] – система сбора отчётов об ошибках в операционных системах на базе Windows.

Миллионы устройств на сегодняшний день работают управлением ОС Windows, и на каждом из них присутствует WER. Основная задача этой системы собирать необходимую информацию для разработчиков компании Microsoft, которую они могут использовать для исправления ошибок: образ памяти, значения переменных среды, файлы логов и т.д. Основная сложность для WER – это огромный поток ошибок, который не возможно обработать человеку без автоматизации. Один из подходов это кластеризация отчётов. Отчёты, которые соответствуют одинаковым ошибкам помещаются в одну «корзину» и разработчик может смотреть на несколько отчётов, разбираясь с одной ошибкой. Кластеризация происходит на основе схожести стеков вызовов. При таком подходе могут возникать неточности, если в одной функции несколько ошибок, а каждая ошибка проявляется с одинаковым стеком вызовов.

По заявлениям авторов [6], разработчики обнаружили и исправили более 5000 ошибок, выявленных из отчётов WER в ОС Windows Vista. Основной недостаток данной системы в том, что она недоступна, а все отчёты, даже о сторонних программах отправляются в Microsoft, а не разработчикам этого ПО.

2.2 Ubuntu Apport

Apport [3] – система сбора отчётов об ошибках дистрибутивах Ubuntu, начиная с 6.10. Как и WER от Microsoft, Apport преследует те же цели – формирование отчётов об ошибках и отправка их разработчикам, в данном случае в поддержку дистрибутива Ubuntu. Использование данной системы имеет ряд преимуществ:

- отчёт формируется сразу при сбое, который не всегда легко воспроизвести;
- вся необходимая информация для отчёта собирается автоматически.

Информация об дистрибутиве, версия пакета, карта памяти процесса, состояние регистров, значения переменных среды, стек вызовов, номер сигнала, который привёл к аварийному завершению и т.д. – всё это входит в состав отчёта об ошибке. Важной особенностью, является то, что отправление отчёта в поддержку дистрибутива Ubuntu является необязательным. Таким образом, пользователь может генерировать отчёты для разработчиков стороннего ПО. Apport – система с открытым исходным кодом на языке Python. Одним из недостатков системы можно считать отсутствие оценки критичности аварийного завершения.

3. Casr

Casr позволяет анализировать образ памяти в момент завершения процесса (coredump), оценивать критичность аварийного завершения, а также формировать отчёты об ошибках. Анализ образа памяти в момент аварийного завершения стал возможен в ОС Linux начиная с ядра 2.6.19. Для этого, системе можно указать полный путь программы-анализатора, и система в момент аварийного завершения какой-либо программы направит содержимое образа памяти на стандартный поток ввода программы-анализатора. Из полученного образа памяти, а также от процесса исследуемой программы (он ещё живой в момент анализа) casr получает всю необходимую информацию для отчёта и оценки критичности.

Оценка критичности основывается проведении классификации аварийного завершения. Каждый класс в свою очередь, может принадлежать одной из трёх групп: эксплуатируемые классы аварийных завершений, возможно эксплуатируемые классы аварийных завершений, неэксплуатируемые классы аварийных завершений.

К эксплуатируемым ситуациям относятся такие аварийные завершения, которые максимально простые для перехвата потока управления программы. Например, аварийное завершение при выполнении инструкции вызова или возврата из функции, или при попытке доступа на выполнение по текущему адресу в указателе инструкций.

К потенциально эксплуатируемым относятся такие ошибки, которые требуют незначительных действий от атакующего для перехвата потока управления. Например, аварийное завершение при загрузке значения из памяти на регистр. Загруженное значение может контролироваться нарушителем и использоваться для передачи управления. К неэксплуатируемым аварийным относятся такие аварийные завершения, эксплуатация которых мало вероятна, например, деление на ноль или разыменование нулевого указателя.

Рис. 1. Схема работы инструмента casr
Fig. 1. Casr workflow scheme

На рис. 1 представлена общая схема работы инструмента. Для оценки критичности casr оценивает состояние программы в момент аварийного завершения. Для этого из образа памяти, который сохранён в формате elf необходимо извлечь карту процесса, сохранённые регистры для всех потоков (prpsinfo), информацию о сигнале из-за которого возникло аварийное завершение (siginfo). Соответствующие структуры расположены в разделе записей (notes) core-файла. Затем анализируется номер сигнала, рассматриваются следующие сигналы: SIGSEGV, SIGABRT, SIGILL, SIGFPE.

При анализе SIGABRT, анализируется стек вызовов, на основе которого делается вывод о классе аварийного завершения.

При анализе SIGSEGV, SIGFPE необходимо провести дизассемблирование инструкции, на которой произошло аварийное завершение, выяснить причины (доступ к памяти чтение/запись/выполнение, семантика инструкции и т.д.). Затем отнести инструкцию к какому-то классу аварийного завершения.

Для аварийного завершения при загрузке значения из памяти в регистр проводится дополнительный анализ помеченных данных. Его цель – уточнить класс аварийного завершения. Рассмотрим работу легковесного анализа помеченных данных на примерах. Ниже приводится фрагмент базового блока для вызова виртуальной функции по таблице.

```
1. mov eax, dword ptr [eax] // аварийное завершение при чтении из памяти
2. mov eax, dword ptr [eax]
3. call eax //потенциальный перехват потока управления
```

Изначально множество помеченных регистров инициализируется загруженным регистром на шаге №1 (*eax*). За тем на шаге №2 происходит доступ к памяти на чтение по помеченному регистру. В этом случае, считается, что значение памяти тоже контролируется. Таким образом, загруженный регистр (*eax*) на шаге №2 тоже становится помеченным. На шаге №3 происходит инструкция вызова. Алгоритм проверяет, есть ли регистры из множества помеченных значений в операндах инструкции. В данном случае регистр *eax* помеченный,

что означает возможность перехвата потока управления при инструкции вызова. Из этого следует, что можно уточнить класс аварийного завершения и присвоить новый: аварийное завершение при инструкции вызова, обнаруженное легковесным анализом помеченных данных, который уже относится к эксплуатируемым классам.

Рассмотрим следующий пример.

```
1. mov esi , dword ptr [ebx]  // аварийное завершение при чтении из памяти
2. add esi , 4
3. mov dword ptr [esi] , eax // запись по контролируемому адресу
```

В этой ситуации алгоритм работает схожим образом. На шаге №3 регистр esi является помеченным, что свидетельствует о потенциально возможной ситуации записи контролируемого значения по контролируемому адресу (CWE-123). В связи с этим, можно уточнить класс аварийного завершения и присвоить новый: аварийное завершение на инструкции записи в память, обнаруженное легковесным анализом помеченных данных, который уже относится к эксплуатируемым классам.

Анализ проводится в рамках базового блока, в котором произошло аварийное завершение.

Сигнал SIGILL однозначно определяется классом аварийного завершения, который соответствует попытке выполнить неправильно сформированную инструкцию.

Кроме классификации аварийных завершений casr, как и apport собирает важную информацию для разработчика: информацию об дистрибутиве, версию пакета, карта памяти процесса, состоянии регистров, значения переменных среды, стек вызовов и т. д.

Важной особенностью casr является то, что в отличии от apport он также собирает дополнительную информацию об открытых файлах и сетевых соединениях в момент аварийного завершения. Эта информация может помочь при попытке воспроизведения аварийного завершения или при ответной реакции на сам факт аварийного завершения (блокировка сетевых соединений указанных адресов).

Вся эта информация формируется в итоговый отчёт, который может быть передан разработчику или специалисту по безопасности.

4. Детали реализации инструмента casr

Инструмент разработан на языке Rust и в своём составе имеет компоненты с открытым исходным кодом. Основными компонентами являются: библиотека для парсинга исполняемых файлов *libgoblin* [7], библиотека для дизассемблирования *capstone* [8,9], библиотека для получения стека вызовов *libunwind* [10]. В рамках разработки для библиотеки libunwind сделаны интерфейсы для вызова функций библиотеки из языка Rust (bindings), которые доступны в открытом доступе [11].

Поддерживаются следующие архитектуры для работы и анализа: x86/64, armv7. Также инструмент может быть собран статически, что упрощает развёртывание, если в системы отсутствуют необходимые библиотеки. Результаты работы (отчёты) хранятся в файлах формата json. Инструмент может поставляться в виде deb-пакета для дистрибутивов на базе Debian. Для просмотра отчётов реализована утилита casr-cli, предоставляющая терминальный пользовательский интерфейс просмотра.

5. Тестирование разработанного инструмента

Для тестирования инструмент был установлен в систему Astra Linux Common Edition 1.6 «Орёл». Для пакетов *imagemagick-6*, *ffmpeg*, *popler-utils* был сформирован набор входных данных, позволяющих воспроизвести ранее известные уязвимости, информация о которых была получена из открытых источников. Также была установлена уязвимая версия сервера nginx для проверки работоспособности инструмента при анализе CVE-2013-2028: уязвимость

переполнения буфера на стеке, которая возникает из-за целочисленного переполнения. На рис. 2 представлен фрагмент отчёта для уязвимости CVE-2013-2028.

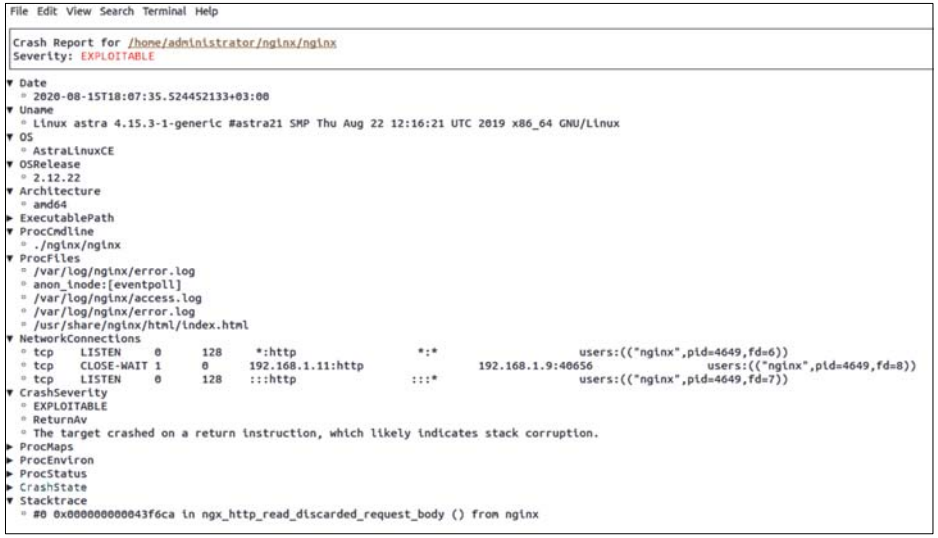

Рис. 2. Пример отчёта об ошибке инструмента casr
Fig. 2. Casr report example

Из основных полей можно отметить поле с оценкой критичности, которое указывает на то, что данное аварийное завершение произошло на инструкции возврата из функции и является эксплуатируемым. Так же стоит обратить внимание на сетевые соединения, активные момент аварийного завершения, из них видно, что с машины по адресу 192.168.1.9 пришёл http-запрос. Из полезных вещей можно отметить строку запуска программы, стек вызовов (в данном случае это одна функция, где произошло аварийное завершение, так как стековые кадры перезаписаны в результате переполнения буфера), открытые файлы.

Ниже приводится краткое описание отчётов об аварийных завершениях из пакетов, которые были установлены в системе.

Ffmpeg (CVE 2019-13390). Ошибка вызвана делением на ноль в функции *avio_enum_protocols*. Инструмент оценил данное завершение как неэксплуатируемое.

Ffmpeg (bug tracker id 8252) аварийное завершение произошло в функции *avfilter_transform* из-за обращения по нулевому указателю. Инструмент оценил данное завершение как неэксплуатируемое.

Imagemagick (bug tracker id 923). Ошибка произошла в функции *ExportQuntomPixels* – программа вызвала функцию *abort()*. Инструмент оценил данное завершение как неэксплуатируемое.

Popler-utils (bugs.fredesktop.org 85276). Ошибка произошла из-за деления на ноль. Инструмент оценил данное завершение как неэксплуатируемое.

Popler-utils (https://gitlab.freedesktop.org/poppler/poppler/-/issues/750). Ошибка возникает из-за нарушения доступа к памяти в функции *Splash::blitTransparent*. Инструмент оценил данное завершение как неэксплуатируемое.

Корректность работы инструмента проверялась путём анализа сохранённых образа памяти через отладчик *gdb*. Инструмент правильно оценил аварийное завершение для уязвимости CVE-2013-2028 и правильно установил адрес откуда пришёл http-запрос. Ошибки системных пакетах также были проверены посредством анализа образа памяти с помощью gdb. Casr

правильно установил класс аварийного завершения, стек вызовов, открытые файлы в момент падения, строку запуска программы.

6. Заключение

Сбор и анализ информации об аварийных завершениях используется как на уровне операционных систем (WER Microsoft, apport Ubuntu), так и на уровне отдельных программ (Firefox Mozila). В этих системах есть два основных недостатка: стороннему разработчику программного обеспечения невозможно получить отчёт о работе его ПО (WER отправляет отчёты в Microsoft), и отсутствует оценка критичности аварийных завершений (apport Ubuntu).

В статье представлен инструмент Casr, который формирует отчёты об ошибках и оценивает критичность обнаруженных аварийных завершений. Инструмент предназначен для ОС Linux. Поддерживаемые процессорные архитектуры – x86/64, armv7. Отчёт об ошибках ормируется на основе анализа образа памяти (coredump) и информации о процессе в системе.

Сгенерированные отчёты могут быть помочь разработчикам исправить ошибки, возникшие в результате эксплуатации ПО, причём оценка критичности позволяет определить, какие ошибки (экслпуатируемые) нужно исправлять в первую очередь. Кроме того, casr потенциально может быть использован в качестве агента для хостовой системы обнаружения вторжений. Рассмотрим пример уязвимости CVE-2013-2028. Здесь casr определил аварийное завершение как эксплуатируемое и указал открытые сетевые соединения в момент аварийного завершения. Такую ситуацию можно оценить как атаку и автоматически заблокировать все входящие соединения с указанного адреса и порта до дальнейших действий от пользователя системы.

В качестве направления развития инструмента можно отметить интеграцию с хостовой системой обнаружения вторжений.

Список литературы / References

[1]. ГОСТ Р 56939-2016 Защита информации. Разработка безопасного программного обеспечения. Общие требования. / GOST R 56939-2016 Information protection. Secure software development. General requirements (in Russian).
[2]. Dang Y., Wu Rongxin, Zhang H., Zhang D., Nobel P. Rebucket: A method for clustering duplicate crash reports based on call stack similarity. In Proc. of the 34th International Conference on Software Engineering (ICSE), 2012, pp. 1084-1093.
[3]. Apport. URL: https://wiki.ubuntu.com/Apport, accessed 25.08.2020.
[4]. Mozilla crash reporter. URL: https://support.mozilla.org/en-US/kb/mozillacrashreporter, accessed 25.08.2020.
[5]. Gdb ‘exploitable’ plugin. URL: https://github.com/jfoote/exploitable, accessed 25.08.2020.
[6]. Glerum K., Glerum K., Kinshumann K., Greenberg S., Aul G., Orgovan V., Nichols G., Grant D., Loihle G. Debugging in the (very) large: ten years of implementation and experience. In Proc. of the ACM SIGOPS 22nd Symposium on Operating Systems Principles, 2009, pp. 103-116.
[7]. libgoblin. URL: https://github.com/m4b/goblin, accessed 25.08.2020.
[8]. Capstone. URL: https://github.com/aquynh/capstone, accessed 25.08.2020.
[9]. Capstone-rust. URL: https://github.com/capstone-rust, accessed 25.08.2020.
[10]. Libunwind. URL: https://www.nongnu.org/libunwind/, accessed 25.08.2020.
[11]. libunwind-rs. URL: https://github.com/xcoldhandsx/libunwind-rs, accessed 25.08.2020.

Информация об авторах / Information about authors

Шамиль Фаимович КУРМАНГАЛЕЕВ – кандидат физико-математических наук, старший научный сотрудник. Сфера научных интересов: Статический анализ бинарного кода, динамический анализ, фаззинг, обфускация, символьное выполнение.

Shamil Faimovich KURMANGALEEV – Ph.D. of Physical and Mathematical Sciences, Senior Researcher. Research interests: Static analysis of binary code, dynamic analysis, fuzzing, obfuscation, symbolic execution.

Андрей Николаевич ФЕДОТОВ – кандидат технических наук, младший научный сотрудник. Сфера научных интересов: динамический анализ, символьное выполнение, поиск ошибок в программах.

Andrey Nikolaevich FEDOTOV – Ph.D. in Engineering sciences, junior researcher. Research interests: dynamic analysis, symbolic execution, search for errors in programs.