УДК 004.056.5 DOI:10.31854/1813-324X-2020-6-1-77-85

Идентификация архитектуры процессора выполняемого кода на базе машинного обучения. Часть 1. Частотно-байтовая модель

М.В. Буйневич[1, 2], **К.Е. Израилов**[1*]

[1]Санкт-Петербургский государственный университет телекоммуникаций им. проф. М.А. Бонч-Бруевича, Санкт-Петербург, 193232, Российская Федерация
[2]Санкт-Петербургский университет государственной противопожарной службы МЧС России, Санкт-Петербург, 196105, Российская Федерация
[*]Адрес для переписки: konstantin.izrailov@mail.ru

Информация о статье
Поступила в редакцию 05.03.2020
Принята к публикации 20.03.2020

Ссылка для цитирования: Буйневич М.В., Израилов К.Е. Идентификация архитектуры процессора выполняемого кода на базе машинного обучения. Часть 1. Частотно-байтовая модель // Труды учебных заведений связи. 2020. Т. 6. № 1. С. 77–85. DOI:10.31854/1813-324X-2020-6-1-77-85

Аннотация: *Изложены результаты исследования способа идентификации архитектуры процессора исполняемого кода на базе машинного обучения. В первой части статьи производится обзор существующих решений по идентификации машинного кода и делается предположение относительно нового способа. Рассматриваются особенности инструкций машинного кода и строится его частотно-байтовая модель. На базе последней предлагается схема идентификации архитектуры процессора. Также приводятся частотные сигнатуры для следующих Топ-10 процессорных архитектур: amd64, arm64, armel, armhf, i386, mips, mips64el, mipsel, ppc64el, s390x.*

Ключевые слова: *информационная безопасность, машинный код, архитектура процессора, машинное обучение, частотно-байтовая модель, сигнатура кода*

Введение

Информационная безопасность, актуальность которой в современном мире безусловна, затрагивает множество областей, которые относятся не только к противодействию соответствующим угрозам, но и к исследованию их последствий. Так, после свершившегося факта атаки на информационные ресурсы требуется изучение того, как эта атака была проведена, кем проведена и с использованием каких программно-аппаратных средств. При этом необходимо учитывать тот факт, что следы киберпреступления могут скрываться злоумышленником, затрудняя тем самым его поиск.

Подобными вопросами занимается соответствующая наука по расследованию киберпреступлений – *форензика,* основными задачами которой является построение сценария атаки и хронологии событий, сбор следов нарушителя, составление его образа и непосредственная идентификация, предложение превентивных мер защиты и т. п.

Один из ее начальных этапов, очевидно, заключается в сборе информации о системе (далее – Система), как той, на которую была направлена угроза (жертвы), так и той, с которых была проведена атака (нарушителя). Большинство атак на информационные ресурсы производится с помощью выполняемых файлов (типичным примером которых являются *вирусы, эксплойты и логические бомбы*) [1]. Исследование же машинного кода (далее – МК) выполняемых файлов позволит понять логику их работы, что гипотетически даст возможность обнаружить вредоносный код [2].

Существует достаточное количество методологических подходов и способов анализа МК. Однако первостепенной задачей остается определение архитектуры процессора – ЦП (именно не концепции системы команд – RISC, CISC и др., а набора инструкций – x86, PowerPC, MIPC и др.), для выполнения на котором данный код предназначен. Это необходимо для выбора соответствующих средств, предназначенных для анализа только определенного набора машинных команд. Кроме того, знания о процессорной архитектуре позволят дополнить информацию об аппаратной составляющей Системы (так, некоторые из них имеют в своем составе процессоры различных архитектур, часть

из которых как раз могут быть использованы нарушителем для злонамеренных действий).

1. Постановка задачи и схема исследования

Проведенный авторами анализ выявил полное отсутствие высокоэффективных специализированных способов добывания информации о программно-аппаратных средствах атакующей/атакуемой Системы – налицо противоречие с острой потребностью в такой информации в интересах предотвращения и/или раскрытия киберпреступлений. С учетом вышеизложенного, определим **целью** настоящего исследования получение дополнительной информации об «орудиях преступления», а именно – установление архитектуры процессора нарушителя/жертвы, имея в распоряжении только МК вредоносной программы, который закономерно является **объектом** исследования.

Логика, заложенная в МК (содержание), выполнима на различных типах процессоров в разной байтовой кодировке (форме) для каждого типа. Тогда **предметом** исследования могут выступать информационные характеристики МК, ассоциированные с характерными особенностями (признаками) той или иной процессорной архитектуры.

Для достижения цели в рамках выбранных объекта и предмета исследования потребуется решить следующую **научно-техническую задачу** – *разработать способ однозначной (в практическом приложении) идентификации архитектуры процессора машинного кода*.

Исследование будет проведено по канонической схеме «анализ → синтез → оценка» и предполагает следующую ее детализацию:

– обоснование практических значимых условий и допущений на процессорные архитектуры и корректность МК;

– обзор и сравнение существующих решений по идентификации МК для формирования научно-обоснованных требований к «идеальному» решению;

– анализ предметной области и создание ее модели, в терминах которой можно выдвинуть гипотезу, лежащую в основе нового способа идентификации;

– выбор и обоснование механизма создания способа идентификации на базе методов искусственного интеллекта, как главного тренда современной IT-науки;

– синтез нового способа идентификации МК и проектирование архитектуры средства его реализации;

– программная реализация и оценка эффективности разработанного средства;

– исследование границ применимости нового способа для различных вариантов использования.

Исходя из значительного (свыше 50) количества архитектур процессоров [3], их модификаций и поколений, введем разумное **ограничение на задачу исследования**, выбрав только наиболее часто применяемые. Для этого воспользуемся набором сборок популярной операционной системы Debian (дистрибутива GNU/Linux) версии 10.3.0 [4], которые содержат установочный бинарный код для следующих архитектур (и соответствующих процессоров):

1) amd64 – для 64-битного Intel/AMD;
2) arm64 – для 64-битного ARM;
3) armel – для 32-битного ARM EABI;
4) armhf – для 32-битного ARM с аппаратной поддержкой плавающей запятой;
5) i386 – для 32-битного Intel;
6) mips – для 32-битного MIPS с порядком байтов от старшего к младшему;
7) mips64el – для 64-битного MIPS с порядком байтов от младшего к старшему;
8) mipsel – для 32-битного MIPS с порядком байтов от младшего к старшему;
9) ppc64el – для 64-битного PowerPC с порядком байтов от младшего к старшему;
10) s390x – для 32-битного IBM System Z.

Таким образом, сформирован Top-10 архитектур процессоров, идентификацию МК которых необходимо производить. Будем далее называть множество МК, относящегося к одной из указанных архитектур – *Классом МК*.

Введем **дополнительное условие к задаче исследования**, основанное на практическом опыте специалистов по компьютерной криминалистике и отражающее реальную ситуацию в области в части подготовленности нарушителей к проводимым атакам. Так, последние для недопущения как раскрытия деталей атаки, так и собственного обнаружения, будут стремиться «замести следы» преступной деятельности путем уничтожения примененных программных средств. И если ситуация их полного уничтожения выходит за рамки **предметной области** – то есть *определения архитектуры процессора машинного кода*, то в случае частичного повреждения файлов решение задачи должно давать удовлетворительный результат. Очевидно, что будет некоторая зависимость вероятности корректной идентификации МК от количества стертых (или замененных на случайные значения) его областей. Хотя стоит отметить, что чисто теоретически идентификация МК возможна даже по одному установленному биту – например, если этот бит используется для кодирования команд единственной архитектурой из списка «подозреваемых».

2. Обзор существующих решений

Проведем обзор существующих решений, применимых в интересах задачи исследования.

Базовым способом определения Класса МК можно считать использование информации в заголовке выполняемого файла. Существует 2 наиболее

распространенных формата заголовков: ELF (*от англ.* Executable and Linkable Format – исполняемый и компонуемый формат), используемый, как правило, в операционных системах FreeBSD, Linux, Solaris и др. [5]; и PE (*от англ*. Portable Executable – переносимый исполняемый), применяемый в операционной системе Microsoft Windows [6, 7]. В формате первого заголовка присутствует поле, как раз определяющее архитектуру аппаратной платформы, для которой создан файл (со значениями 3 для Intel 80386, 8 для MIPS R3000, 20/21 для PowerPC 32/64-битности, 40 для ARM и т.д.). В формате второго заголовка также есть поле, ответственное за процессор выполнения (со значениями 0x014C для Intel 80386, 0x0160 для MIPS R3000, 0x01C0 для ARM, 0x01F0 для PowerPC и т.п.). Однако полностью полагаться на данные поля нельзя, поскольку некоторые производители сознательно изменяют их значения на некорректные как раз в интересах защиты своего ПО от анализа.

Еще одним способом решения задачи идентификации МК может считаться использование «brute-force» подхода, пришедшего из непосредственной практики исследования программно-аппаратных комплексов (как правило, с позиции их информационной безопасности) и направленного на определение состава и принципов работы таких систем путем анализа кода служебных утилит, механизмов их выполнения, взаимодействия с окружением и др. Так, последовательные попытки дизассемблирования МК под разные архитектуры будут иметь различный успех по причине того, что некоторые комбинации бит в коде не будут соответствовать ни одной из инструкций – это может стать критерием отсечения использованной архитектуры из списка допустимых для выполнения МК [8–10].

Тем не менее, у такого способа будет достаточное количество ограничений. Например, при отсутствии информации о точке входа в код не ясно, с какого момента (а точнее адреса байта) необходимо начать дизассемблирование (напомним, что инструкция большинства архитектур процессоров кодируется не одним, а несколькими байтами). Кроме того, МК небольшого размера теоретически может быть дизассемблирован под различные архитектуры, особенно если множества кодирований из наборов инструкций значительно пересекается.

«Классикой» идентификации файлов является сравнение значений их хэш-кодов с записями в эталонной базе данных для заранее собранных файлов [11]. Развитием такого подхода можно считать применение Fuzzy Hash (*пер. с англ.* – нечеткий хэш) [12] и хэш индекса подобия [13], позволяющих определять сходство между файлами, а не их тождественность. Однако идентификация архитектуры процессора возможна лишь в ограниченном числе случаев, при которых отличия исследуемого МК от заложенного в базе, даже при условии соответствия их архитектур, не являются значительными. Также, остается открытым вопрос полноты собранной эталонной базы.

Существует ряд работ, решающих задачу идентификации файлов путем сравнения частотной характеристики появления определенных значений байт МК – *сигнатуры программы* – с базой заранее подготовленных сигнатур. При этом способе для сравнения применяется как критерий Фишера и превышение частоты появления байта некоторого предела, заданного экспертным путем [14, 15], так и использование наивного классификатора Байеса (*от англ*. – Naive Bayes) [16]. Способ обладает схожими с предыдущим ограничениями, хотя отличия исследуемых файлов от эталонных на его успешность влияют и менее сильно.

Произведем сравнение рассмотренных решений по следующим критериям:

– *К_1. Полнота*, определяемая охватом файлов, поддающихся идентификации (потребность решения больших эталонных баз влияет негативно по причине крайней сложности их сбора в необходимом объеме, в особенности без какой-либо автоматизации);

– *К_2. Трудоэкономность*, соответствующая степени ручного труда экспертов (ручной анализ отдельных файлов и их сбор в базу данных влияет негативно по причине невозможности избежать человеческого фактора, повышающего количество ошибок, часть из которых будет значительно ухудшать качество идентификации);

– *К_3. Обход защиты*, обозначающий применимость решения для файлов, защищаемых от идентификации МК (неработоспособность решения при незначительном изменении заголовка файла влияет негативно, т. к. применение простейшей защиты МК от анализа полностью нейтрализует попытку идентификации);

– *К_4. Стойкость к повреждению*, оценивающая, насколько решение может показывать высокий результат для поврежденных файлов (снижение вероятности корректной идентификации при пропаже некоторых байт МК влияет негативно, поскольку в этом случае код, разработанный под одну архитектуру, может быть отнесен к иному Классу МК);

– *К_5. Интеллектуальность*, обозначающая возможность решения самостоятельно синтезировать и использовать новые знания, не заданные экспертом (применение лишь ограниченного количества строгих правил идентификации влияет негативно вследствие практической невозможности описать вручную такими правилами все многообразие комбинаций байт МК).

Согласно результатам проведенного критериального сравнения (таблица 1), у каждого из решений есть свои достоинства и недостатки.

ТАБЛИЦА 1. Результаты критериального сравнения решений, применимых для идентификации МК

TABLE 1. The Results of Criteria-Based Comparison of Solutions Used to Identify Machine Code

Решение	К_1	К_2	К_3	К_4	К_5
Заголовки файлов [5–7]	+	+	–	+/–	–
Успешность дизассемблирования МК [8–10]	+	+/–	+/–	–	+/–
Хэши файлов [11–13]	–	+/–	+	+/–	–
Частотные сигнатуры файлов [14–16]	+/–	+/–	+	+/–	+

Так, анализ заголовков файлов достаточно результативен в идеальных ситуациях – для файлов без повреждений. Попытка дизассемблирования под разные процессоры будет иметь высокий уровень ошибок (как I-го, так и II-го рода) и потребует ручной валидации результатов. Сравнение хэшей исследуемых файлов с эталонной базой работоспособно только при наличии последней в полном объеме. Наиболее результативным может считаться решение с применением частотных сигнатур, хотя и оно соответствует большинству критериев не полностью – в части размера и заполнения эталонной базы, а также стойкости к повреждению МК. Таким образом, интересным с научной и важной с практической точки зрения является разработка решения по идентификации МК в виде способа, обладающего следующими характеристиками: использование эталонной базы минимально возможного размера; полная автоматизация работы; применение для МК, который может быть защищен от анализа и/или быть частично разрушен; построение на принципах работы интеллектуальных систем (например, с применением методов машинного обучения), позволяющих самостоятельно строить новые связи и алгоритмы работы [17–19].

3. Анализ машинного кода

Проанализируем структуру МК и метаданные, применимые для его идентификации. Структура МК определяется его заголовком, наиболее популярными примерами которого являются ELF и PE; оба задают способ отображения программы в памяти. Наиболее интересной в рамках исследования является секция выполняемого кода (именуемая как «.text»). Инструкции процессора, архитектуру которого необходимо определить, располагаются именно в данной секции, и поэтому другие секции не рассматриваются.

Все процессорные архитектуры могут быть разбиты на некоторый набор типов, имеющих существенные различия. Так, важная особенность типа CISC (*от англ.* Complex Instruction Set Computer – сложный набор команд компьютера) состоит в нефиксированной длине команды, что требует при дизассемблировании каждой последующей команды распознать предыдущую (а точнее, ее длину); типичными представителями являются процессоры Intel, AMD и IBM System Z. Альтернативой данному типу можно считать RISC (*от англ.* Reduced Instruction Set Computer – сокращенный набор команд компьютера), которая отличается использованием инструкций одной длины и структуры; типичными представителями являются процессоры Arm, MIPS и PowerPC.

Исходя из вышеотмеченных особенностей, моделью МК, применимой для идентификации всех Классов МК, может быть частотно-байтовое распределение его инструкций – частота «встречания» байт инструкций во всем выполняемом коде, содержащемся в файле. Такое распределение, по аналогии с известными работами, назовем – *частотной сигнатурой МК*. Отталкиваясь от упомянутых различий CISC и RISC, необходимо оценивать частоту именно каждого байта, который может быть как началом целой инструкции, так и ее серединой или концом. В формальном виде сигнатура МК в файле может быть записана, как массив частот значений его байт из $2^8 = 256$ элементов:

$$Signature_{File} = Freq[256](File),$$

где *File* – файл с МК, а частота $Freq[i]$ отнормирована по максимальному значению, т. е. определяется как количество байт со значением i ($Count[i]$) по отношению к максимальному количеству байт:

$$Freq[i] = \frac{Count[i]}{\max(Count[0], \dots, Count[255])}.$$

В интересах решения задачи исследования выдвинем следующую *гипотезу*: *«Множество файлов с машинным кодом, выполняемым на одной из выбранных процессорных архитектур, обладает собственной уникальной частотной сигнатурой, отличной от остальных»*. Данная гипотеза может быть записана в формальном виде следующим образом:

$$\begin{cases} Signature^{Class_i} = Signature_{\{File1_{Class_i}, \dots, FileN_{Class_i}\}} \\ Signature^{Class_i} \neq\neq Signature^{Class_j} \text{ при } i \neq j \\ Class_i \in \{amd64, arm64, armel, armhf, i386, mips, mips64el, mipsel, ppc64el, s390x\} \end{cases},$$

где $Signature^{Class_i}$ – сигнатура множества файлов МК $\{File1_{Class_i}, \dots, FileN_{Class_i}\}$ одного класса $Class$, при этом сигнатура одного класса ($Class_i$) существенно отлична ($\neq\neq$) от сигнатуры другого ($Class_j$). В соответствии с ограничением на задачу исследования, количество классов равно количеству архитектур.

Для верификации гипотезы получим усредненные частотно-байтовые распределения МК всех классов – т. е. *сигнатуры классов МК*, воспользовавшись модулем построения частотно-байтовой модели, описание которого будет приведено далее – при рассмотрении архитектуры разработанного средства идентификации. Таким образом, данный модуль был разработан изначально в интересах анализа предметной области, а затем вошел в состав средства, разработанного для решения поставленной задачи (далее – Средство).

В качестве наборов файлов с МК использовались сборки Debian 10.3.0 для каждой процессорной архитектуры, имеющими вид ISO-образов и размер в распакованном виде от 360 Мб (для armel) до 880 Мб (для i386).

Помимо существенного размера выборки (даже с учетом того, что не все файлы в сборках содержат машинный код), ее можно считать достаточно разнообразной с точки зрения всевозможных вариантов сигнатур, поскольку Debian представляет из себя операционную систему, а, следовательно, содержит реализацию достаточно широкого набора функционала. Извлечение файлов из сборки может быть осуществлено другим модулем, также вошедшим в состав Средства.

Размеры всех выполняемых файлов (в байтах) с машинным кодом (в форматах ELF и PE) для каждой из Top-10 архитектур составили значения, приведенные в таблице 2; данный *датасет* (*от англ.* dataset – набор данных) может быть получен из Интернет-ресурса по ссылке http://demono.ru/projects/BinArhType/.

ТАБЛИЦА 2. Размеры файлов

TABLE 2. File Sizes

ЦП	amd64	arm64	armel	armhf	i386	mips	mips64el	mipsel	ppc64el	s390x
Файл	201 167 165	157 478 312	99 131 132	161 106 696	255 544 343	173 863 012	110 959 016	177 905 464	209 457 484	100 392 820

В среднем на одно значение байта приходится примерно $\frac{(100+250)/2}{256}$ Мб = ~700 000 байт данных (от ~100 Мб для s390x до ~250 Мб для i386), что можно считать достаточным для выявления общей закономерности в частотно-байтовом распределении Класса МК.

Полученные частотные сигнатуры классов МК (т. е. для каждого типа архитектуры ЦП) приведены на рисунке 1.

Графический анализ частотных сигнатур классов (см. рисунок 1) позволяет сделать следующие выводы. Во-первых, сигнатура каждого класса обладает четко выделенным пиком максимальной частоты при значении байта, равного 0, что объясняется использованием данного значения в качестве стандартного во множестве случаев – например, при первоначальной инициализации памяти. Во-вторых, для каждого класса характерна высокая частота появления байта со значением 255 – видимо, также вследствие его частого использования для служебных нужд – например, в качестве значения –1, соответствующего в беззнаковой форме записи числу 255. В-третьих, частота для значения 0 примерно в 4 раза превосходит частоту для значения 255, что можно объяснить общим соотношением использований этих чисел в коде программ. В-четвертых, достаточно близкие архитектуры mips и mipsel, отличающиеся лишь порядком байт, имеют и схожие сигнатуры классов (пики при значениях 2, 16, 32, 36, 37, 143 и 255), что также предсказуемо. Исходя из этого, можно предположить, что возможно ошибочное отношение МК одной архитектурой к классу другой архитектуры. В-пятых, также близкая к этим архитектура mips64el, которая является 64-битной версией mipsel, обладает отличной от них сигнатурой – пик в точке 233 и бóльшая частота в точке 255, хотя часть пиков и совпадает – в точках 16, 32, 36, 37. И, в-шестых, аналогично mips и mipsel, близкие архитектуры armel и armhf, вторая из которых поддерживает плавающую запятую на уровне инструкций, имеют схожие сигнатуры классов – пики в точках 16, 32, 48, 160, 224 – 227 и 229; тем не менее, их внешний «облик» более различим, чем для mips и mipsel.

Подводя предварительные итоги, можно сделать общий вывод, что частотная сигнатура МК для заданной архитектуры отличается от сигнатур МК других архитектур, и, следовательно, выдвинутая гипотеза может считаться подтвержденной. Схожесть частотных сигнатур архитектур mips VS mipsel, а также armel VS armhf не является критической, поскольку их можно объединить в одну архитектуру – mips и arm соответственно, т. к. они, по сути, представляют один и тот же набор инструкций процессора.

На основании поставленной задачи исследования, а также модели и подтвержденной гипотезы касательно частотных сигнатур классов, предложим *схему идентификации архитектуры процессора выполняемого кода*, состоящую из следующих последовательно выполняемых этапов.

Этап 1. Формирование большой выборки МК для всех архитектур, обладающей полнотой относительно разнообразия реализуемого функционала.

Этап 2. Построение частотных сигнатур классов МК для всех архитектур.

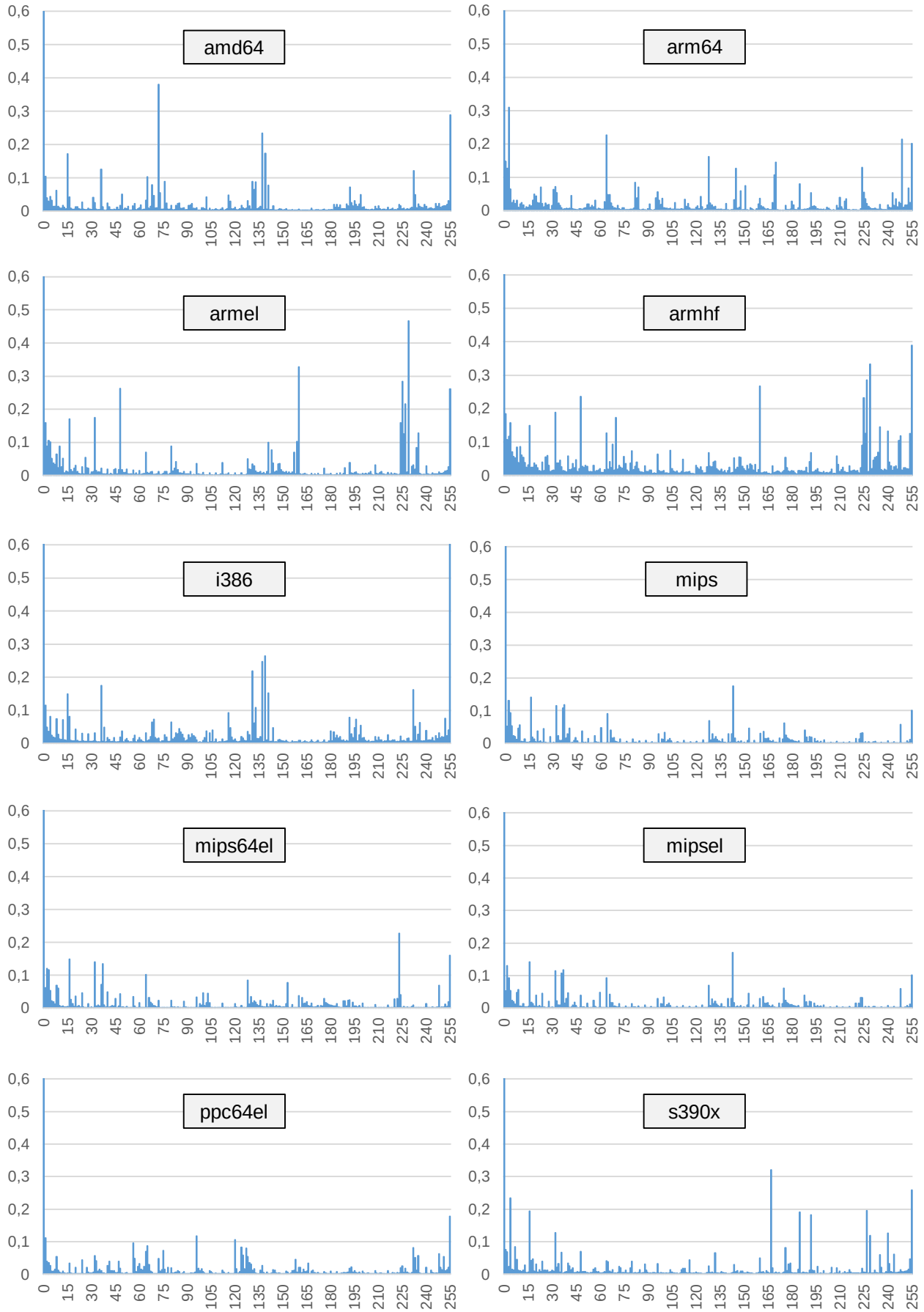

Рис. 1. Частотные сигнатуры классов МК для Top-10 архитектур процессоров

Fig. 1. Machine Code Classes Frequency Signatures for Top-10 Processor Architectures

Этап 3. Реализация алгоритма сравнения тестовой выборки МК с сигнатурами классов, позволяющей делать выбор наиболее близкого. Целесообразным будет также вычисления степени принадлежности МК из тестовой выборки к каждому из классов – для определения степени достоверности результата.

Этап 4. Выбор тестовой выборки МК для оценки функционирования процесса идентификации.

Этап 5. Проведение эксперимента над тестовой выборкой и вычисление эффективности идентификации. Под последней здесь понимается совокупность Результативности – количества верно определенных классов, Оперативности – скорости определения классов, Ресурсоэкономности – количества затраченных человеческих и программно-аппаратных ресурсов (человеко-машинных часов).

Используя введенные выше формальные записи, схема идентификации архитектуры процессора выполняемого кода может быть представлена в следующем виде:

$$\begin{cases} ClassModel = BuildModel\left(\sum_{i=i}^{N}\langle Signature^{Class_i}|Class_i\rangle\right) \\ \sum_{i=1}^{N}\langle Class_i|Probability_i\rangle = IdentificationAlgorithm(File, Model) \\ Class^{File} = Class_i, \text{для } i\colon Probability_i = \max(Probability_1, \dots, Probability_N) \end{cases},$$

где $ClassModel$ – модель классов МК, описывающая взаимосвязь между частотно-байтовыми распределениями файлов и степенью их принадлежности к каждому классу из числа N; $BuildModel()$ – функция для построения модели на основании пар $\langle Signature^{Class_i}|Class_i\rangle$, соответствующих сигнатуре класса и его названию; $\langle Class_i|Probability_i\rangle$ – пара названия класса и вероятности отнесения к нему идентифицируемого файла; $IdentificationAlgorithm()$ – алгоритм определения вероятностей ($Probability_i$) отнесения файла ($FIle$) к каждому из классов ($Class_i$) на основании построенной модели ($Model$); $Class^{File}$ – результат идентификации в виде класса ($Class_i$), вероятность ($Probability_i$) отнесения к которому файла с МК максимальна.

Таким образом, проанализировано множество вариаций МК для Топ-10 процессорных архитектур и получены соответствующие им сигнатуры на базе единой частотно-байтовой модели. Как следствие, подтверждена гипотеза, которая легла в основу схемы идентификации архитектуры процессора МК.

Следуя канонам исследования и опираясь на результаты, полученные в первой части статьи, последующие части будет посвящены непосредственному синтезу способа идентификации на базе машинного обучения, а также его программной реализации и оценке.

Продолжение следует ...

Список используемых источников

1. Buinevich M., Izrailov K., Vladyko A. The life cycle of vulnerabilities in the representations of software for telecommunication devices // 18th International Conference on Advanced Communications Technology (ICACT-2016, Pyeongchang, South Korea, 31 January–3 February 2016). IEEE, 2016. PP. 430–435. DOI:10.1109/ICACT.2016. 7423420

2. Buinevich M., Izrailov K., Vladyko A. Method and prototype of utility for partial recovering source code for low-level and medium-level vulnerability search // Proceedings of the 18th International Conference on Advanced Communication Technology (ICACT-2016, Pyeongchang, South Korea, 31 January–3 February 2016). IEEE, 2016. PP. 700–707. DOI:10.1109/ICACT.2016.7423603

3. Dake L., Zhaoyun C., Wei W., Trends of communication processors // China Communications. 2016. Vol. 13. Iss. 1. PP. 1–16. DOI:10.1109/CC.2016.7405699

4. Файлы образов Debian версии 10.3.0 // Debian. URL: https://www.debian.org/distrib/netinst.ru.html (дата обращения: 20.03.2020)

5. Штеренберг С.И., Красов А.В. Варианты применения языка ассемблера для заражения вирусом исполнимого файла формата ELF // Информационные технологии и телекоммуникации. 2013. Т. 1. № 3. С. 61–71.

6. Штеренберг С.И., Андрианов В.И. Варианты модификации структуры исполнимых файлов формата PE // Перспективы развития информационных технологий. 2013. № 16. С. 134–143.

7. Юрин И.Ю. Способы установления первоначального имени PE-файла // Теория и практика судебной экспертизы. 2008. № 3(11). С. 200–205.

8. Касперски К., Рокко Е. Искусство дизассемблирования. СПб. БХВ-Петербург, 2009. 896 с.

9. Sulaiman A., Ramamoorthy K., Mukkamala S., Sung A.H. Disassembled code analyzer for malware (DCAM) // Proceedings of the International Conference on Information Reuse and Integration (IRI, Las Vegas, USA, 15–17 August 2005). IEEE, 2005. PP. 398–403. DOI:10.1109/IRI-05.2005.1506506

10. Krishnamoorthy N., Debray S., Fligg K. Static Detection of Disassembly Errors // Proceedings of the 16th Working Conference on Reverse Engineering (Lille, France, 13–16 October 2009). IEEE, 2009. PP. 259–268. DOI:10.1109/WCRE.2009.16

11. Антонов А.Е., Федулов А.С. Идентификация типа файла на основе структурного анализа // Прикладная информатика. 2013. № 2(44). С. 068–077.
12. Израилов К.Е., Гололобов Н.В., Краскин Г.А. Метод анализа вредоносного программного обеспечения на базе Fuzzy Hash // Информатизация и связь. 2019. № 2. С. 36–44. DOI:10.34219/2078-8320-2019-10-2-36-44
13. Choi S., Kim Y. Kim J. Similarity Hash Index // Proceedings of the 9th International Conference on Information and Communication Technology Convergence (ICTC 2018, Jeju Island, Korea, 17–19 October 2018). IEEE, 2018. PP. 1298–1300. DOI:10.1109/ICTC.2018.8539650
14. Salakhutdinova K., Lebedev I., Krivtsova I., Bazhayev N., Sukhoparov M., Smimov P. et al. A Frequency Approach to Creation of Executable File Signatures for their Identification // Proceedings of the 11th International Conference on Application of Information and Communication Technologies (AICT, Moscow, Russia, 20–22 September 2017). IEEE, 2017. PP. 1–7. DOI:10.1109/ICAICT.2017.8687105
15. Кривцова И.Е., Салахутдинова К.И., Юрин И.В. Метод идентификации исполняемых файлов ПО их сигнатурам // Вестник государственного университета морского и речного флота им. адмирала С.О. Макарова. 2016. № 1(35). С. 215–224. DOI:10.21821/2309-5180-2016-8-1-215-224
16. Мищенко Н.К. Способ идентификации ELF-файлов на основе классификатора Байеса // XLVIII научная и учебно-методическая конференция Университета ИТМО. Альманах научных работ молодых ученых Университета ИТМО. 2019. Том 2. С. 38–42.
17. Dhingra M., Jain M., Jadon R.S. Role of artificial intelligence in enterprise information security: A review // Proceedings of the Fourth International Conference on Parallel, Distributed and Grid Computing (PDGC, Waknaghat, India, 22–24 December 2016). IEEE, 2016. PP. 188–191. DOI:10.1109/PDGC.2016.7913142
18. Yousaf M.S., Durad M.H., Ismail M. Implementation of Portable Executable File Analysis Framework (PEFAF) // Proceedings of the 16th International Bhurban Conference on Applied Sciences and Technology (IBCAST, Islamabad, Pakistan, 8–12 January 2019). IEEE, 2019. PP. 671–675. DOI:10.1109/IBCAST.2019.8667202
19. Markel Z., Bilzor M. Building a machine learning classifier for malware detection // Proceedings of the Second Workshop on Anti-malware Testing Research (WATeR, Canterbury, UK, 23–23 October 2014). IEEE, 2014. PP. 1–4. DOI:10.1109/WATeR.2014.7015757

* * *

Identification of Processor's Architecture of Executable Code Based on Machine Learning. Part 1. Frequency Byte Model

M. Buinevich[1, 2], **K. Izrailov**[1]

[1]The Bonch-Bruevich Saint-Petersburg State University of Telecommunications,
St. Petersburg, 193232, Russian Federation
[2]Saint-Petersburg University of State Fire Service of Emercom of Russia,
St. Petersburg, 195105, Russian Federation

Article info
DOI:10.31854/1813-324X-2020-6-1-77-85
Received 5th March 2020
Accepted 20th March 2020

For citation: Buinevich M,. Izrailov K. Identification of Processor's Architecture of Executable Code Based on Machine Learning. Part 1. Frequency Byte Model. *Proc. of Telecom. Universities.* 2020;6(1):77–85. (in Russ.) DOI:10.31854/1813-324X-2020-6-1-77-85

Abstract: *This article shows us the study results of a method for identifying the processor architecture of an executable code based on machine learning. In the first part of the article we see an overview of existing solutions for machine code identifying and we see how the author makes a new method assumption. The author considers features of the machine code instructions and build its frequency-byte model. There is a processor architecture identification scheme, which is based on this model. Apart from that we see the frequency signatures which are provided for the following Top 10 processor architectures: amd64, arm64, armel, armhf, i386, mips, mips64el, mipsel, ppc64el, s390x.*

Keywords: *information security, machine code, processor architecture, machine learning, frequency-byte model, code signature*

References

1. Buinevich M., Izrailov K., Vladyko A. The life cycle of vulnerabilities in the representations of software for telecommunication devices. *Proceedings of the 18th International Conference on Advanced Communications Technology, ICACT-2016, 31 January–3 February 2016, Pyeongchang, South Korea*. IEEE; 2016. p.430–435. DOI:10.1109/ICACT.2016.7423420
2. Buinevich M., Izrailov K., Vladyko A. Method and prototype of utility for partial recovering source code for low-level and medium-level vulnerability search. *Proceedings of the 18th International Conference on Advanced Communication Technology, ICACT-2016, 31 January–3 February 2016, Pyeongchang, South Korea*. IEEE; 2016. p.700–707. DOI:10.1109/ICACT.2016.7423603
3. Dake L., Zhaoyun C., Wei W. Trends of communication processors. *China Communications*. 2016;13(1):1–16. DOI:10.1109/CC.2016.7405699
4. *Debian*. Debian Version 10.3.0 Image Files Available from: https://www.debian.org/distrib/netinst.en.html [Accessed 20th March 2020]
5. Shterenberg S.I., Krasov A.V. Methods of Using Assembly Language for Infection the Virus Executable File Format .ELF. *TelecomIT*. 2013;1(3): 61–71. (in Russ.)
6. Shterenberg S.I., Andrianov V.I. Options for Modifying the Structure of PE Executable Files. Perspektivy razvitiia informatsionnykh tekhnologii. 2013;16:134–143. (in Russ.)
7. Yurin I.Yu. Ways to Set the Initial PE File Name. *Theory and Practice of Forensic Science*. 2008;3(11):200–205. (in Russ.)
8. Kaspersky K., Rocco E. The Art of Disassembly. St. Petersburg. BHV Publ.; 2009. 896 p. (in Russ.)
9. Sulaiman A., Ramamoorthy K., Mukkamala S., Sung A.H. Disassembled code analyzer for malware (DCAM). *Proceedings of the International Conference on Information Reuse and Integration, IRI, 15–17 August 2005, Las Vegas, USA*. IEEE; 2005. p.398–403. DOI:10.1109/IRI-05.2005.1506506
10. Krishnamoorthy N., Debray S., Fligg K. Static Detection of Disassembly Errors. *Proceedings of the 16th Working Conference on Reverse Engineering, 13–16 October 2009, Lille, France*. IEEE; 2009. p.259–268. DOI:10.1109/WCRE.2009.16
11. Antonov A.E., Fedulov A.S. File Type Identification Based on Structural Analysis. *Applied informatics*. 2013;2(44):068–077. (in Russ.)
12. Izrailov K.E., Gololobov N.V., Kraskin G.A. Method of MALwARE Analysis Based on Fuzzy Hash. *Informatizatsiia i sviaz*. 2019;2:36–44. DOI:10.34219/2078-8320-2019-10-2-36-44
13. Choi S., Kim Y. Kim J. Similarity Hash Index. *Proceedings of the 9th International Conference on Information and Communication Technology Convergence, ICTC 2018, 17–19 October 2018, Jeju Island, Korea*. IEEE; 2018. p.1298–1300. DOI:10.1109/ICTC.2018.8539650
14. Salakhutdinova K., Lebedev I., Krivtsova I., Bazhayev N., Sukhoparov M., Smimov P. et al. A Frequency Approach to Creation of Executable File Signatures for their Identification. *Proceedings of the 11th International Conference on Application of Information and Communication Technologies, AICT, 20–22 September 2017, Moscow, Russia.* IEEE; 2017. p.1–7. DOI:10.1109/ICAICT.2017.8687105
15. Krivtsova I.E., Salakhutdinova K.I., Yurin I.V. Method of Executable Filts Identification by their Signatures. *Vestnik Gosudarstvennogo Universiteta Morskogo i Rechnogo Flota Imeni Admirala S.O. Makarova*. 2016;1(35):215–224. DOI:10.21821/2309-5180-2016-8-1-215-224
16. Mishchenko N.K. Method for Identifying ELF Files Based on Bayes Classifier. *Proceedings of the XLVIII Scientific and Educational Conference of ITMO University. Almanac of Scientific Works of Young Scientists of ITMO University*. 2019. vol.2. p.38–42 (in Russ.)
17. Dhingra M., Jain M., Jadon R.S. Role of artificial intelligence in enterprise information security: A review. *Proceedings of the Fourth International Conference on Parallel, Distributed and Grid Computing, PDGC, 22–24 December 2016, Waknaghat, India*. IEEE; 2016. p.188–191. DOI:10.1109/PDGC.2016.7913142
18. Yousaf M.S., Durad M.H., Ismail M. Implementation of Portable Executable File Analysis Framework (PEFAF). *Proceedings of the 16th International Bhurban Conference on Applied Sciences and Technology, IBCAST, 8–12 January 2019, Islamabad, Pakistan*. IEEE; 2019. p.671–675. DOI:10.1109/IBCAST.2019.8667202
19. Markel Z., Bilzor M. Building a machine learning classifier for malware detection. *Proceedings of the Second Workshop on Anti-malware Testing Research, 23–23 October 2014, WATeR, Canterbury, UK*. IEEE; 2014. PP. 1–4. DOI:10.1109/WATeR.2014.7015757

Сведения об авторах:

БУЙНЕВИЧ Михаил Викторович — доктор технических наук, профессор, профессор кафедры безопасности информационных систем Санкт-Петербургского государственного университета телекоммуникаций им. проф. М.А. Бонч-Бруевича, профессор кафедры прикладной математики и информационных технологий Санкт-Петербургского университета государственной противопожарной службы МЧС России, bmv1958@yandex.ru
https://orcid.org/0000-0001-8146-0022

ИЗРАИЛОВ Константин Евгеньевич — кандидат технических наук, доцент кафедры защищенных систем связи Санкт-Петербургского государственного университета телекоммуникаций им. проф. М.А. Бонч-Бруевича, konstantin.izrailov@mail.ru
https://orcid.org/0000-0002-9412-5693