УДК 004.056.5 DOI:10.31854/1813-324X-2020-6-3-48-57

Идентификация архитектуры процессора выполняемого кода на базе машинного обучения. Часть 3. Оценка качества и границы применимости

М.В. Буйневич[1, 2], **К.Е. Израилов**[1, 3*]

[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
[3]Санкт-Петербургский федеральный исследовательский центр Российской академии наук, Санкт-Петербург, 199178, Российская Федерация
[*]Адрес для переписки: konstantin.izrailov@mail.ru

Информация о статье
Поступила в редакцию 10.07.2020
Принята к публикации 08.09.2020

Ссылка для цитирования: Буйневич М.В., Израилов К.Е. Идентификация архитектуры процессора выполняемого кода на базе машинного обучения. Часть 3. Оценка качества и границы применимости // Труды учебных заведений связи. 2020. Т. 6. № 3. С. 48–57. DOI:10.31854/1813-324X-2020-6-3-48-57

Аннотация: *В статье изложены результаты тестирования авторского способа идентификации архитектуры процессора выполняемого кода на базе машинного обучения. В третьей заключительной части цикла определены его качественные показатели: точность, полнота и F-мера для выполняемых файлов сборки Debian. Исследованы границы применимости способа идентификации архитектуры для четырех условий: отсутствие заголовка файлов, различные размеры машинного кода, частичное разрушение кода и наличие инструкций нескольких архитектур. Указаны обнаруженные недостатки предлагаемого способа и пути их устранения, а также дальнейшее направление развития.*

Ключевые слова: *информационная безопасность, машинный код, архитектура процессора, машинное обучение, сигнатура кода, способ идентификации, показатели качества, матрица ошибок, точность, полнота, F-мера, разрушение кода.*

Введение

Одной из задач информационной безопасности является исследования файлов информационной системы, наиболее важными из которых являются выполняемые – содержащие машинный код (далее – МК) [1]. Первоначальным шагом такого исследования служит определение архитектуры процессора МК, что даст общую картину о составе информационной системы, а также позволит выбрать наиболее подходящий инструментарий для ее анализа [2–4]; притом даже в случае разрушения файловой структуры [5]. Существуют различные способы решения данной задачи, рассмотренные в 1-ой части цикла статей [6]. Недостатки каждого из них привели к необходимости создания частотно-байтовой модели кода, а затем и к разработке авторского способа идентификации на базе машинного обучения, описанного во 2-ой части цикла [7].

Продолжая следовать канонической схеме «анализ → синтез → оценка», в данной статье будет произведена оценка качества [8] способа идентификации, а также определены границы его применимости. В интересах первого действия будут получены точность, полнота и *F-мера* способа, а в интересах второго – исследованы работоспособность способа при отсутствии заголовка файлов [9, 10], зависимость от размера кода и степени его разрушения [11], а также влияние нескольких процессорных архитектур в коде. Затем потребуется обсуждение полученных результатов, из чего последуют предложения касательно дальнейшего усовершенствования способа.

Оценка качества классификации

Показатели качества

Наиболее часто используемыми показателями качества классификации (применимыми к множеству задач, например, сетевой безопасности [12]

или обработки текста [13]) являются *точность* и *полнота*. Применительно к задаче определения архитектуры процессора МК, первая определяет долю файлов, для которых архитектура была определена верно, по сравнению со всеми файлами, отнесенными способом к данной архитектуре. Вторая определяет долю файлов с верно определенной архитектурой по сравнению со всеми истинными файлами данной архитектуры. Естественно, идеальным случаем будет максимальное значение точности и полноты, что в реальных условиях практически недостижимо; по этой причине, как правило, вводится *F-мера*, представляющая собой некий компромисс между точностью и полнотой в виде среднего гармонического. Более же полную картину о проведенной идентификации архитектур для некоторой тестовой выборки можно увидеть по *матрице ошибок*, отражающей в численном виде взаимосвязь между реальными архитектурами процессора и теми, которые определил способ. Все остальные показатели качества могут быть напрямую получены из этой матрицы, поэтому целесообразно получить вначале именно ее.

Матрица ошибок

Для получения матрицы ошибок авторское программное средство идентификации архитектуры процессора МК было модернизировано добавлением 4-го режима работы – «Вычисление метрик», в котором, помимо сигнатур и архитектур процессора МК, средству указывались директории с выполняемыми файлами и реальными архитектурами процессоров. Таким образом, производится сравнение архитектур, полученных с помощью разработанного способа, и реальных.

Так, после запуска программного средства в данном режиме (1-ый аргумент *MetricMode*) с помощью командной строки:

```
> BinArchId.exe MetricMode 3 amd64=D:\Debian\Amd64\ i386=D:\Debian\I386\
mipsel=D:\Debian\Mipsel\ amd64=D:\Debian\amd64.sig i386=D:\Debian\i386.sig
                          mipsel=D:\Debian\mipsel.sig
```

оно выполнит 5 действий. Во-первых, установит число исследуемых архитектур, равное 3 (2-ой аргумент). Во-вторых, вычислит сигнатуры файлов из директорий, вложенных в D:\Debian\, указав для них верные архитектуры – *amd64*, *i386* и *mipsel* (с 3-го по 5-ый аргументы). В-третьих, аналогично режиму «Идентификация МК» (при аргументе *TestMode*), загрузит из директории «D:\Debian\» сигнатуры «*amd64.sig*», «*i386.sig*» и «*mipsel.sig*» для соответствующих архитектур (аргументы с 6-го по 8-ой). В-четвертых, произведет идентификацию всех файлов согласно сигнатурам. И, в-пятых, построит матрицу ошибок, где столбцами является реальная архитектура процессора МК, строками – идентифицированная архитектура, а значениями в ячейках – количество файлов с обеими архитектурами. В случае первого аргумента *MetricModeX*, файлы будут интерпретироваться как полностью состоящие из МК (т. е. без учета заголовка и секций). Для получения более корректных оценок режим использует одинаковое количество файлов каждой из архитектур – путем вычисления их минимального значения из всех.

Для вычисления матрицы ошибок воспользуемся сборкой *Debian* для различных архитектур [14], которая использовалась ранее для тестирования программного средства. Также сделаем некоторое упрощение, а именно, опустим идентификацию архитектуры *mipsel*, поскольку она достаточно близка к архитектуре *mips* и будет вносить неоправданные (и уже показанные во 2-ой части цикла [7]) неточности, как в саму матрицу, так и в вычисляемые показатели. Результат применения режима *MetricMode* представлен в таблице 1; число файлов каждой из архитектур составило 1554.

ТАБЛИЦА 1. Матрица ошибок при идентификации машинного кода файлов сборки Debian

TABLE 1. Error Matrix for Machine Code Identification of Debian Build Files

Идентифицируемые архитектуры процессора	Реальные архитектуры процессора								
	amd64	*arm64*	*armel*	*armhf*	*i386*	*mips*	*mips64el*	*ppc64el*	*s390x*
amd64	1147	5	0	5	43	5	0	6	0
arm64	0	1449	0	7	0	0	0	0	0
armel	0	0	1552	294	0	0	0	0	0
armhf	1	0	2	1066	1	0	0	0	0
i386	223	0	0	4	1421	0	0	0	0
mips	1	9	0	10	3	1319	26	0	0
mips64el	0	0	0	7	0	224	1528	0	0
ppc64el	182	91	0	160	86	6	0	1548	0
s390x	0	0	0	1	0	0	0	0	1554

Примечание. Зеленым фоном в таблице помечено наибольшее количество файлов в каждой строке, желтым – количество файлов более 10, серым – количество менее 10 (кроме 0).

Как хорошо видно (см. таблицу 1), все архитектуры были определены корректно – диагональ матрицы, указывающая на количество совпадения реальных архитектур с идентифицированными (зеленый фон), содержит максимальные значения. Однако, все же некоторое количество файлов было идентифицировано ошибочно (серый и желтый фон – в зависимости от превышения порога в 10).

Точность, полнота и F-мера

Используя матрицу ошибок, вычислим точность и полноту для идентификации каждой из архитектур. Точность (*Precession*) может быть получена, как отношение значения в диагональном элементе к сумме значений в соответствующей строке:

$$Precission_x = \frac{M_{x,x}}{\sum_{k=1..N} M_{k,x}},$$

где x – порядковый номер архитектуры; $M_{i,j}$ – значения матрицы ошибок на пересечении i-го столбца и j-ой строки; N – размер матрицы (т. е. число архитектур).

Аналогичным образом, полнота (*Recall*) может быть получена, как отношение диагонального элемента к сумме значений в столбце следующим образом:

$$Recall_x = \frac{M_{x,x}}{\sum_{k=1..N} M_{x,k}}.$$

Используя данные показатели качества, *F-мера* ($F_{measure}$) получается следующим образом:

$$F_{measure} = 2 \times \frac{Precession \times Recall}{Precession + Recall}. \quad (1)$$

Тривиальные вычисления позволяют получить следующие значения для точности, полноты и *F-меры* архитектур процессора (таблица 2).

Анализ полученных показателей (см. таблицу 2) демонстрирует, что большинство из них входят в типовой диапазон требований – больше 0,9, что является свидетельством достаточно хорошего качества работы способа и программного средства. *F-мера* считается универсальным показателем работы классификаций (включая идентификацию), и поэтому далее для определения границ применимости способа будем использовать именно ее. Предельное же значение точности, полноты и *F-меры* (т. е. 1,0) при идентификации архитектуры достигается только для архитектуры *s390x*, что говорит о ее отличительной уникальности среди остальных.

2. Границы применимости

Оценим границы применимости способа для различных условий, определяя статическое или динамическое значение *F-меры*.

Отсутствие заголовка

Для проверки работоспособности способа в режиме, не учитывающем заголовки файлов (как ELF [15], так и [16], что, соответственно, ведет к построению сигнатур не только по секциям выполняемого кода, но и по секциям данных и служебной информации), необходимо запустить средство с первым аргументом *MetricModeX*. Полученные результаты представлены в таблице 3.

Исследование условия без чтения заголовка файлов показало, что в общем случае способ имеет низкую работоспособность. Так, бóльшая часть файлов была идентифицирована, как относящаяся к архитектуре процессора *ppc64el* (серый фон), что, естественно, не верно. Это также подтверждается низким значением показателей качества – *F-мера* не превышает 0,5, при этом большинство архитектур имеет ее значение менее 0,3. Исключением, помимо *ppc64el*, является архитектура *mips* (см. соответствующий столбец таблицы 3), поскольку 2/3 реальных файлов этой архитектуры все же будет отнесено к верной архитектуре. Впрочем, в каждой строке таблицы диагональные элементы также имеют максимальное значение (зеленый фон), что говорит о том, что отдельно точность идентификации все также остается высокой.

Удаление из тестирования архитектуры *ppc64el* не повышает результативность идентификации остальных, поскольку тогда основная масса файлов начинает относиться к *mips64el*, затем к *mips* и т. д. Данный эффект возникает по причине преобладания в байтах файла (как области кода, так и данных) числа 0, что вносит сильный «шум» в процесс идентификации.

ТАБЛИЦА 2. Показатели качества идентификации машинного кода файлов сборки Debian

TABLE 2. Debian Build Files Machine Code Identification Quality Indicators

Показатели	Архитектуры процессора								
	amd64	*arm64*	*armel*	*armhf*	*i386*	*mips*	*mips64el*	*ppc64el*	*s390x*
Точность	0,95	1,00	0,84	1,00	0,86	0,96	0,87	0,75	1,00
Полнота	0,74	0,93	1,00	0,69	0,91	0,85	0,98	1,00	1,00
F-мера	0,83	0,96	0,91	0,81	0,89	0,90	0,92	0,85	1,00

Примечание. Зеленым фоном в таблице помечены ячейки со значением больше 0,9, желтым – больше 0,8, синим – больше 0,7, красным – все остальные

ТАБЛИЦА 3. Матрица ошибок и показателей качества при идентификации машинного кода файлов сборки Debian без учета заголовков файлов

TABLE 3. Error and Quality Indicators Matrix for Machine Code Identification of Debian Build Files without Regard to File Headers

Идентифицируемые архитектуры процессора	Реальные архитектуры процессора									Показатели качества		
	amd64	*arm64*	*armel*	*armhf*	*i386*	*mips*	*mips64el*	*ppc64el*	*s390x*	Точность	Полнота	*F-мера*
amd64	60	0	0	0	15	0	0	0	0	0,80	0,04	0,07
arm64	0	43	1	0	0	0	0	0	0	0,98	0,03	0,05
armel	0	0	186	0	0	0	0	0	0	1,00	0,12	0,21
armhf	1	1	2	5	1	2	1	0	1	0,36	0,00	0,01
i386	13	4	8	6	116	4	8	1	10	0,68	0,07	0,13
mips	5	8	326	76	9	962	899	2	115	0,40	0,62	0,49
mips64el	1	0	0	8	1	1	310	0	0	0,97	0,20	0,33
ppc64el	1474	1498	1031	1459	1412	585	336	1551	1287	0,15	1,00	0,25
s390x	0	0	0	0	0	0	0	0	14	1,00	0,09	0,17

Примечание. Зеленым фоном в таблице помечено наибольшее количество файлов в каждой строке, иначе серым – наибольшее количество файлов в каждом столбце.

Размер кода

Крайне интересным, как с научной, так и с практической точки зрения, является исследование зависимости *F-меры* от размера идентифицируемого МК, поскольку это покажет, насколько способ работоспособен для отдельных (небольших) участков кода. Для проведения такого эксперимента средство было временно модифицировано так, чтобы циклически брать лишь определенное количество байт из секции кода, проводить их идентификацию и получать матрицу ошибок (из которой потом вычислялись показатели качества). Результаты в виде графика зависимости *F-меры* каждой из архитектур от размера МК показаны на рисунке 1 (шаг приращения длины кода равен 50 байт).

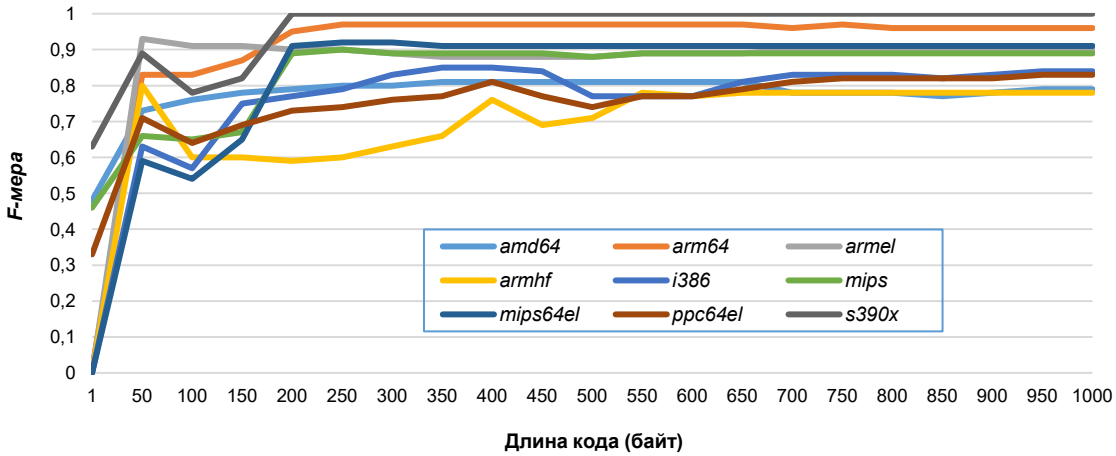

Рис. 1. Зависимость *F-меры* идентификации архитектур машинного кода от его размера

Fig. 1. The F-Measure Identification Dependence of Machine Code Architectures on its Size

На рисунке 1 в качестве оси абсцисс выбрано число байт в секции кода, по которым производилась идентификация архитектуры МК, а в качестве оси ординат – *F-мера*. При этом, поскольку вычисление *F-меры* для нулевой длины не имеет смысла, то в качестве точки отсчета по горизонтали выбран 1 байт. Также, для ряда архитектур (*armel*, *armhf*, *i386* и *mips64el*) значение точности и полноты оказалось равным нулю (при единичной длине), и, следовательно, *F-мера* по формуле (1) не может быть вычислена; в этом случае, мера принималась равной нулю.

Анализ рисунка 1 позволяет сделать вывод, что после длины секции кода примерно в 1000 байт *F-меры* достигают своих, близких к типовым, значений, указанных в таблице 2. Следовательно, данное значение можно считать нижней границей, которая позволяет получить удовлетворительные результаты работы предложенного способа идентификации.

Также, достаточно интересная картина на графике наблюдается в области длины кода 50 и 100 байт. В первом случае *F-мера* для большинства архитектур имеет ярко выраженный локальный максимум, а во втором – локальный минимум. Затем картина нормализуется, и *F-мера* начинает достаточно равномерно приближаться к типовым значениям. Такое поведение можно объяснить

тем, что на малых длинах МК происходит «конкуренция» среди различных архитектур за принадлежность к ним секций кода. При этом, данных оказывается недостаточно даже для теоретического построения полноценной сигнатуры (которая состоит из распределения 256 байт). Таким образом, вследствие ошибок идентификации на области абсциссы до 250 байт график имеет существенные колебания большинства *F-мер*.

Разрушение кода

В случае, если хранилище информации, содержащее выполняемые файлы, было подвержено частичному разрушению [17] (например, физическим воздействием на HDD или SSD, или же программным воздействием на файловую систему), идентификация архитектуры МК будет затруднена. Во-первых, если будут уничтожены заголовки файлов, то, как показали предыдущие исследования (см. таблицу 3), это может катастрофически сказаться на результативности способа. Во-вторых, если заголовки файлов не были затронуты, то МК в секциях файлов будет содержать «шум» в виде случайных байт – эффект разрушения. Такая ситуация представляет особый интерес с точки зрения определения границ применимости способа для частично разрушенных файлов. Для этого средство идентификации было временно модифицировано добавлением алгоритма, производящего замену определенного количества случайных байт МК на произвольные значения – имитируя тем самым различные степени разрушения файла. Из всех файлов бралась часть МК размером 1000 байт ($Length_{All}$, *от англ.* «длина всего»), поскольку, как показали предыдущие эксперименты (см. рисунок 1), этого размера достаточно, чтобы выйти на типовые значения *F-меры*. В качестве же количества заменяемых байт выбирались значения от 0 до 1000 ($Length_{Noize}$, *от англ.* «длина шума») с шагом 50, что позволило динамически оценить влияние разрушения файла на идентификацию:

$$K_{Destroy} = \frac{Length_{Noice}}{Length_{All}} \in [0,1].$$

Результатом исследования стала зависимость *F-меры* каждой из идентификаций архитектур МК в процессе его последовательного разрушения (рисунок 2).

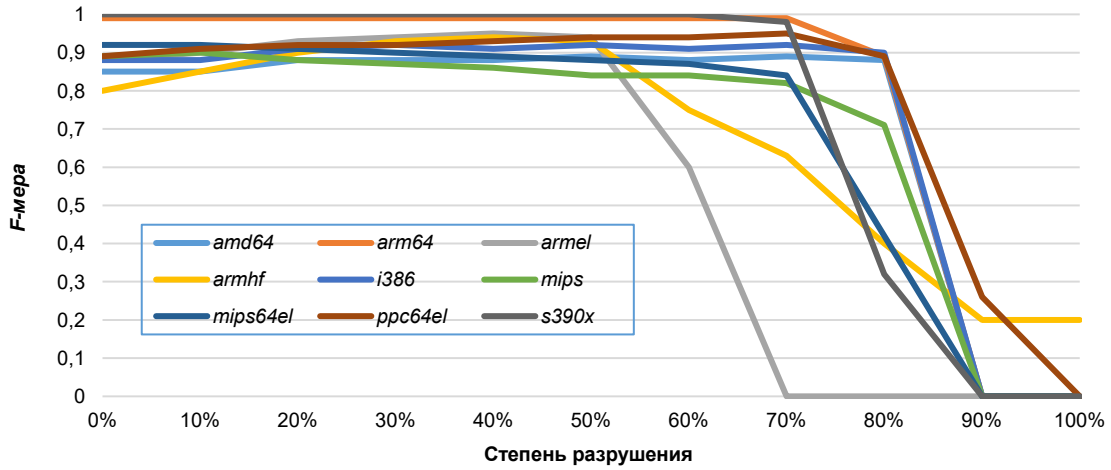

Рис. 2. Зависимость *F-меры* идентификации архитектур машинного кода от степени его разрушения

Fig. 2. The F-Measure Identification Dependence of Machine Code Architectures on the Degree of its Destruction

На рисунке 2 в качестве оси абсцисс выбрана $K_{Destroy}$ – степень разрушения (*Destroy*), выраженная в процентах, а в качестве оси ординат – *F-мера*. При этом, поскольку для ряда измерений точность и полнота оказались равными 0 (область более 70 % по оси абсциссы), то для них *F-мера* принималась равной 0.

Анализ полученных результатов (см. рисунок 2) позволяет сделать следующие выводы. Во-первых, значение *F-меры* для большинства архитектур скачкообразно падает с 50 до 90 % разрушения МК файла; при этом условно критической точкой можно считать 70 %. Во-вторых, при 100 %-ном разрушении (состояние «белого шума») любой МК идентифицируется, как *armhf*. И, в-третьих, на всем процессе разрушения *F-мера* архитектур испытывает колебание в обе стороны – т. е. отсутствует строго снижающаяся динамика. Последнее видимо объяснимо особенностями распределения байт в МК различных архитектур процессора (см. рисунок 1 первой части цикла статей [6]), а точнее, их близости к случайному распределению. Также, частотная сигнатура *armhf* визуально имеет меньшее количество ярко выраженных пиков (или большую площадь распределения), что, видимо и влияет на то, что МК с преобладающим количеством случайных байт оказывается более близок именно к этой архитектуре.

Код разных архитектур

В практике крупного телекоммуникационного оборудования возможна ситуация, когда в одном МК содержатся инструкции сразу нескольких процессорных архитектур; например, присутствует

как загрузчик, выполняемый на основном оборудовании, так и код прошивки периферийного устройства другой архитектуры. Следовательно, необходимо исследовать работу способа в такой ситуации. Для этого можно воспользоваться простым экспериментом, заключающимся не в вычислении сигнатур множества файлов с кодом двух архитектур, а в сложении частотных сигнатур двух классов (архитектур) МК – т. е. тех, которые используются для машинного обучения. Результаты идентификации файлов, которым соответствуют попарные комбинации сигнатур различных архитектур, представлены в таблице 4; последней строкой в таблице идет комбинация всех девяти архитектур.

ТАБЛИЦА 4. Вероятности отнесения комбинированных файлов к архитектурам процессоров

TABLE 4. Probabilities of Attributing Combined Files to Processor Architectures

Тестовые файлы		Архитектуры процессора								
Архитектура 1	*Архитектура 2*	*amd64*	*arm64*	*armel*	*armhf*	*i386*	*mips*	*mips64el*	*ppc64el*	*s390x*
amd64	*arm64*	0,26	0,13	0,07	0,07	0,06	0,10	0,10	0,12	0,08
amd64	*armel*	0,22	0,08	0,12	0,11	0,07	0,10	0,10	0,11	0,10
amd64	*armhf*	0,20	0,09	0,11	0,13	0,08	0,09	0,10	0,10	0,10
amd64	*i386*	0,15	0,08	0,07	0,07	0,37	0,06	0,07	0,07	0,07
amd64	*mips*	0,13	0,06	0,05	0,06	0,09	0,30	0,12	0,11	0,08
amd64	*mips64el*	0,28	0,08	0,07	0,07	0,06	0,12	0,13	0,11	0,08
amd64	*ppc64el*	0,15	0,06	0,06	0,06	0,09	0,07	0,07	0,37	0,08
amd64	*s390x*	0,13	0,08	0,06	0,06	0,10	0,09	0,09	0,11	0,28
arm64	*armel*	0,09	0,13	0,24	0,12	0,07	0,09	0,09	0,09	0,09
arm64	*armhf*	0,10	0,24	0,11	0,13	0,07	0,08	0,09	0,09	0,09
arm64	*i386*	0,12	0,13	0,08	0,08	0,21	0,09	0,09	0,11	0,10
arm64	*mips*	0,12	0,13	0,06	0,06	0,06	0,29	0,11	0,10	0,08
arm64	*mips64el*	0,12	0,13	0,06	0,06	0,06	0,08	0,30	0,10	0,08
arm64	*ppc64el*	0,13	0,27	0,06	0,06	0,07	0,10	0,10	0,13	0,08
arm64	*s390x*	0,08	0,13	0,07	0,07	0,07	0,11	0,11	0,11	0,26
armel	*armhf*	0,09	0,11	0,35	0,14	0,06	0,06	0,06	0,06	0,07
armel	*i386*	0,12	0,08	0,18	0,10	0,12	0,09	0,09	0,10	0,11
armel	*mips*	0,09	0,11	0,23	0,07	0,07	0,12	0,11	0,11	0,09
armel	*mips64el*	0,09	0,11	0,20	0,08	0,08	0,12	0,12	0,11	0,09
armel	*ppc64el*	0,10	0,11	0,12	0,11	0,06	0,08	0,07	0,26	0,09
armel	*s390x*	0,07	0,07	0,14	0,12	0,06	0,08	0,08	0,08	0,30
armhf	*i386*	0,07	0,08	0,10	0,12	0,20	0,10	0,10	0,11	0,11
armhf	*mips*	0,08	0,10	0,11	0,11	0,06	0,21	0,11	0,10	0,11
armhf	*mips64el*	0,09	0,10	0,11	0,19	0,07	0,11	0,12	0,10	0,11
armhf	*ppc64el*	0,10	0,11	0,10	0,12	0,06	0,06	0,07	0,24	0,12
armhf	*s390x*	0,06	0,07	0,11	0,28	0,07	0,09	0,09	0,09	0,13
i386	*mips*	0,12	0,07	0,06	0,07	0,12	0,24	0,12	0,11	0,09
i386	*mips64el*	0,12	0,07	0,06	0,07	0,24	0,11	0,12	0,11	0,09
i386	*ppc64el*	0,12	0,06	0,06	0,07	0,14	0,07	0,07	0,31	0,10
i386	*s390x*	0,11	0,08	0,06	0,07	0,13	0,08	0,09	0,10	0,28
mips	*mips64el*	0,12	0,08	0,07	0,06	0,06	0,30	0,13	0,06	0,12
mips	*ppc64el*	0,13	0,06	0,05	0,05	0,05	0,13	0,10	0,32	0,12
mips	*s390x*	0,08	0,08	0,06	0,06	0,07	0,13	0,12	0,10	0,29
mips64el	*ppc64el*	0,13	0,06	0,05	0,05	0,05	0,11	0,13	0,31	0,11
mips64el	*s390x*	0,08	0,08	0,06	0,06	0,07	0,12	0,13	0,10	0,28
ppc64el	*s390x*	0,08	0,09	0,06	0,06	0,07	0,11	0,10	0,13	0,28
Все архитектуры		0,11	0,11	0,10	0,10	0,10	0,11	0,11	0,15	0,10

Примечание. Зеленым фоном в таблице помечены наибольшие вероятности, соответствующие идентифицированным архитектурам процессора МК, желтым – ближайшая альтернатива, серым – первая корректно идентифицированная архитектура из комбинации, голубым – вторая корректно идентифицированная архитектура из комбинации.

Результаты идентификации МК, содержащего несколько архитектур процессора (см. таблицу 4) позволяют сделать следующие выводы. Во-первых, для всех комбинаций пар архитектур обе из них были определены верно, поскольку имеют максимальную и вторую по величине вероятности (в двух левых столбцах архитектура отмечена синим или голубым фоном). Заметим, что в некоторых строках желтый фон имеют сразу несколько ячеек, поскольку численные вероятности идентификации архитектур совпали; причина этого, скорее всего, кроется в ограниченном выводе знаков после запятой. Во-вторых, различия между максимальной вероятностью и второй по величине можно считать значимыми (минимально, около 30 %), и, следовательно, идентификация хотя бы одной из архитектур является однозначной. И, в-третьих, идентификация «смеси» из всех возможных архитектур (последняя строка таблицы) показала примерно одинаковое распределение вероятностей (0,10 и 0,11) с некоторым «перекосом» в сторону архитектуры *ppc64el*, что необходимо учитывать при практическом использовании.

3. Обсуждение результатов

Исследование показателей качества способа идентификации и границ его применимости позволили определить следующее.

Во-первых, *F-мера* для всех архитектур процессора превосходит 0,8, а для половины и – 0,9, что является достаточно высоким показателем. Еще большего их повышения можно достигнуть усложнением частотно-байтовой модели, например, введением в нее частоты «встречания» двух (или более) определенных байт подряд – *N*-грамм [18, 5], или использованием специальных семантик машинных инструкций [19].

Во-вторых, интерпретация выполняемых файлов, как последовательности байт без учета их заголовков (и, следовательно, без разбиения на секции кода, данных и служебную информацию) показывает низкую работоспособность из-за отнесения большинства таких файлов исключительно к одной из архитектур. Впрочем, тот факт, что отдельно показатель точности остается высоким, позволяет предположить, что модернизация способа позволит успешно его применять и в этом случае (аналогично работам по поиску упакованных PE-файлов [20–22]), в том числе противодействуя сложным системам сокрытия информации [23].

В-третьих, разрушение 70 % и более МК путем замены некоторого количества байт на случайные в конечном итоге приводит к неверной идентификации архитектуры процессора, как *armhf*, вместо ожидаемого равномерного распределения среди архитектур. Такое поведение отражает особенности набора инструкций МК рассматриваемых архитектур и способов их байтового кодирования [24]; поэтому, повышение результативности такой идентификации представляется мало реалистичным.

И, в-четвертых, применение средства для выполняемых файлов, МК которых содержит инструкции сразу двух процессорных архитектур, показало высокую работоспособность – обе архитектуры идентифицируются приемлемым образом; это, очевидно, расширяет область применения способа; например, для поиска дублируемого функционала [25].

Заключение

Таким образом, в третьей части цикла проведены полноценное тестирование и количественная оценка способа и средства идентификации с определением соответствующих показателей качества: точности, полноты и *F-меры*. Последние показали высокие значения, доказывающие применимость способа на практике; в том числе, в качестве предварительного этапа для авторского *метода алгоритмизации* [26, 27] (преобразующего МК в вид, подходящий эксперту для поиска уязвимостей), а также при начальном этапе интеллектуального статического анализа [28]. Были определены границы применимости способа для наиболее типовых условий работы: отсутствие заголовков файлов, влияние размера и степени разрушения МК, наличие инструкций процессора нескольких архитектур. Несмотря на ограниченность применимости в некоторых случаях, предполагается, что дальнейшая и более «тонкая» доработка способа позволит частично устранить эти недостатки.

Основное развитие способа, с точки зрения авторов, должно быть направлено именно на обеспечение его работоспособности в условиях отсутствия заголовков файлов, которые позволяют однозначно выделить секции с инструкциями МК из другой информации. Данное усовершенствование способа может быть осуществлено внедрением предварительного этапа выделения областей с МК, в том числе с применением интеллектуальных методов выявления аномалий. В этом случае будет возможным поиск и точная идентификация процессорной архитектуры участков с выполняемым кодом в любых бинарных областях независимо от окружающего контекста [29].

Окончание цикла.

Список используемых источников

1. Буйневич М.В., Васильева И.Н., Воробьев Т.М., Гниденко И.Г., Егорова И.В. и др. Защита информации в компьютерных системах: монография. СПб.: Санкт-Петербургский государственный экономический университет, 2017. 163 с.

2. Kim J., Youn J.M. Malware behavior analysis using binary code tracking // Proceedings of the 4th International Conference on Computer Applications and Information Processing Technology (CAIPT, Kuta Bali, Indonesia, 8–10 August 2017). IEEE, 2017. DOI:10.1109/CAIPT.2017.8320724

3. Elhadi A.A.E., Maarof M.A., Barry B.I.A. Improving the Detection of Malware Behaviour Using Simplified Data Dependent API Call Graph // International Journal of Security and Its Applications. 2013. Vol. 7. Iss. 5. PP. 29–42. DOI:10.14257/ijsia.2013.7.5.03

4. Anwar Z., Sharf M., Khan E., Mustafa M. VG-MIPS: A dynamic binary instrumentation framework for multi-core MIPS processors // Proceedings of the International Conference on Multi Topic (INMIC, Lahore, Pakistan, 19–20 December 2013). 2013. IEEE, 2013. PP. 166–171. DOI:10.1109/INMIC.2013.6731344

5. Erozan A.S.A. File fragment type detection by neural network // Proceedings of the 26th Signal Processing and Communications Applications Conference (SIU, Izmir, Turkey, 2–5 May 2018). IEEE, 2018. DOI:10.1109/SIU.2018.8404380

6. 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

7. Буйневич М.В., Израилов К.Е. Идентификация архитектуры процессора выполняемого кода на базе машинного обучения. Часть 2. Способ идентификации // Труды учебных заведений связи. 2020. Т. 6. № 2. С. 104–112. DOI:10.31854/1813-324X-2020-6-2-104-112

8. Шунина Ю.С., Алексеева В.А., Клячкин В.Н. Критерии качества работы классификаторов // Вестник Ульяновского государственного технического университета. 2015. № 2(70). С. 67–70.

9. Трофименков А.К., Трофименков С.А., Пимонов Р.В. Алгоритмизация обработки файлов для их идентификации при нарушении целостности данных // Системы управления и информационные технологии. 2020. № 2(80). С. 82–85.

10. Антонов А.Е., Федулов А.С. Идентификация типа файла на основе структурного анализа // Прикладная информатика. 2013. № 2(44). С. 068–077.

11. Касперски К. Как спасти данные, если отказал жесткий диск // Системный администратор. 2005. № 9(34). С. 80–87.

12. Кажемский М.А., Шелухин О.И. Многоклассовая классификация сетевых атак на информационные ресурсы методами машинного обучения // Труды учебных заведений связи. 2019. Т. 5. № 1. С. 107–115. DOI:10.31854/1813-324X-2019-5-1-107-115

13. Попков М.И. Автоматическая система классификации текстов для базы знаний предприятия // International Journal of Open Information Technologies. 2014. Т. 2. № 7. С. 11–18.

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

15. 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.

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

17. Жилин В.В., Сафарьян О.А. Искусственный интеллект в системах хранения данных // Вестник Донского государственного технического университета. 2020. Т. 20. № 2. С. 196–200. DOI:10.23947/1992-5980-2020-20-2-196-200

18. Al-Kasassbeh M., Mohammed S., Alauthman M., Almomani A. Feature Selection Using a Machine Learning to Classify a Malware // Gupta B., Perez G., Agrawal D., Gupta D. (eds) Handbook of Computer Networks and Cyber Security. Springer, Cham, 2020. PP. 889–904. DOI:10.1007/978-3-030-22277-2_36

19. Падарян В.А., Соловьев М.А., Кононов А.И. Моделирование операционной семантики машинных инструкций // Труды Института системного программирования РАН. 2010. Т. 19. С. 165–186.

20. Wang T.-Y., Wu C.-H. Detection of packed executables using support vector machines // Proceedings of the International Conference on Machine Learning and Cybernetics (Guilin, China, 10–13 July 2011). IEEE, 2011. PP. 717–722. DOI:10.1109/ICMLC.2011.6016774

21. Hubballi N., Dogra H. Detecting Packed Executable File: Supervised or Anomaly Detection Method? // Proceedings of the 11th International Conference on Availability, Reliability and Security (ARES, Salzburg, Austria, 31 August–2 September 2016). IEEE, 2016. PP. 638–643. DOI:10.1109/ARES.2016.18

22. Choi Y.-S., Kim I.-K., Oh J.-T., Ryou J.-C. PE File Header Analysis-Based Packed PE File Detection Technique (PHAD) // Proceedings of the International Symposium on Computer Science and its Applications (Hobart, Australia, 13–15 October 2008). IEEE, 2008. PP. 28–31. DOI:10.1109/CSA.2008.28

23. AL-Nabhani Y., Zaidan A.A., Zaidan B.B., Jalab H.A., Alanazi H.O. A new system for hidden data within header space for EXE-File using object oriented technique // Proceedings of the 3rd International Conference on Computer Science and Information Technology (Chengdu, China, 9–11 July 2010). IEEE, 2010. PP. 9–13. DOI:10.1109/ICCSIT.2010.5564461

24. Соловьев М.А., Бакулин М.Г., Макаров С.С., Манушин Д.В., Падарян В.А. Декодирование машинных команд в задаче абстрактной интерпретации бинарного кода // Труды Института системного программирования РАН. 2019. Т. 31. № 6. С. 65–88. DOI:10.15514/ISPRAS-2019-31(6)-4

25. Wang M., Tang Y., Lu Z. Massive Similar Function Searching for Cross-Architecture Binaries // Proceedings of the 18th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES, Wuhan, China, 8–10 November 2019). IEEE, 2019. PP. 195–198. DOI:10.1109/DCABES48411.2019.00055

26. Буйневич М.В., Израилов К.Е. Метод алгоритмизации машинного кода телекоммуникационных устройств // Телекоммуникации. 2012. № 12. С. 2–6.

27. Буйневич М.В., Израилов К.Е. Автоматизированное средство алгоритмизации машинного кода телекоммуникационных устройств // Телекоммуникации. 2013. № 6. С. 2–9.

28. Буйневич М.В., Израилов К.Е. Обобщенная модель статического анализа программного кода на базе машинного обучения применительно к задаче поиска уязвимостей // Информатизация и Связь. 2020. № 2. С. 143–152. DOI:10.34219/2078-8320-2020-11-2-143-152

29. Поддубный В.А., Коркин И.Ю. Средство обнаружения скрытого исполнимого кода в памяти ОС Windows // Вопросы кибербезопасности. 2019. № 5(33). С. 75–82. DOI:10.21681/2311-3456-2019-5-75-82

* * *

Identification of Processor's Architecture of Executable Code Based on Machine Learning. Part 3. Assessment Quality and Applicability Border

M. Buinevich[1, 2], **K. Izrailov**[1, 3]

[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
[3]St. Petersburg Federal Research Center of the Russian Academy of Sciences,
St. Petersburg, 199178, Russian Federation

Article info
DOI:10.31854/1813-324X-2020-6-3-48-57
Received 10th July 2020
Accepted 8th September 2020

For citation: Buinevich M., Izrailov K. Identification of Processor's Architecture of Executable Code Based on Machine Learning. Part 3. Assessment Quality and Applicability Border. *Proc. of Telecom. Universities.* 2020;6(3): 48–57. (in Russ.) DOI:10.31854/1813-324X-2020-6-3-48-57

Abstract: *The article presents the author's method testing results for identifying the processor architecture of the executable code based on machine learning. In the third final part of the cycle, its qualitative indicators are determined: accuracy, completeness and F-measure for the executable files of the Debian build. There are investigated the applicability limits of the architecture identification method for four conditions: the file header absence, different sizes of machine code, partial code destruction, and the presence of instructions from several architectures. We can observe the identified disadvantages of the proposed method and ways to eliminate them, as well as the further direction of its development.*

Keywords: *information security, machine code, processor architecture, machine learning, code signature, identification method, quality indicators, error matrix, accuracy, completeness, F-measure, code destruction*

References

1. Buynevich M.V., Vasilieva I.N., Vorobyov T.M., Gnidenko I.G., Egorova I.V. et al. *Information Security in Computer Systems*. St. Petersburg: Saint Petersburg Electrotechnical University "LETI" Publ.; 2017. 163 p. (in Russ.)

2. Kim J., Youn J.M. Malware behavior analysis using binary code tracking. *Proceedings of the 4th International Conference on Computer Applications and Information Processing Technology, CAIPT, 8–10 August 2017, Kuta Bali, Indonesia*. IEEE; 2017. DOI:10.1109/CAIPT.2017.8320724

3. Elhadi A.A.E., Maarof M.A., Barry B.I.A. Improving the Detection of Malware Behaviour Using Simplified Data Dependent API Call Graph. *International Journal of Security and Its Applications*. 2013;7(5):29–42. DOI:10.14257/ijsia.2013.7.5.03

4. Anwar Z., Sharf M., Khan E., Mustafa M. VG-MIPS: A dynamic binary instrumentation framework for multi-core MIPS processors. *Proceedings of the International Conference on Multi Topic, INMIC, 19–20 December 2013, Lahore, Pakistan*. 2013. IEEE; 2013. p.166–171. DOI:10.1109/INMIC.2013.6731344

5. Erozan A.S.A. File fragment type detection by neural network. *Proceedings of the 26th Signal Processing and Communications Applications Conference, SIU, 2–5 May 2018, Izmir, Turkey*. IEEE; 2018. DOI:10.1109/SIU.2018.8404380

6. 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

7. Buinevich M., Izrailov K. Identification of Processor's Architecture of Executable Code Based on Machine Learning. Part 2. Identification method. *Proc. of Telecom. Universities.* 2020;6(2):104–112. (in Russ.) DOI:10.31854/1813-324X-2020-6-2-104-112

8. Shunina Ju. S., Alekseeva V.A., Klyachkin V.N. Criteria of Quality of Qualifiers Work. *Vestnic of UlSTU*. 2015;2(70):67–70. (in Russ.)

9. Trofimenkov A.K., Trofimenkov S.A., Pimonov R.V. Algorithmization of File Processing for their Identification in Case of Violation of Data Integrity. *Sistemy upravleniya i informatsionnyye tekhnologii*. 2020;2(80):82–85. (in Russ.)
10. Antonov A., Fedulov A. File type identification based on structural analyses. *Journal of Applied Informatics*. 2013;2(44):068–077. (in Russ.)
11. Kaspersky K. How to Save Data if the Hard Drive Fails. *Sistemnyy administrator*. 2005;9(34):80–87.
12. Kazhemskiy M., Sheluhin O. Multiclass Classification of Attacks to Information Resources with Machine Learning Techniques. *Proc. of Telecom. Universities*. 2019;5(1):107–115. (in Russ.) DOI:10.31854/1813-324X-2019-5-1-107-115
13. Popkov M.I. Text Analytics for Enterprise Knowledge Base. *International Journal of Open Information Technologies*. 2014;2(7):11–18. (in Russ.)
14. *Debian.* Debian Image Files Version 10.3.0. Available from: https://www.debian.org/distrib/netinst.ru.html (in Russ.) [Accessed 20th March 2020]
15. 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.)
16. Yurin I.Yu. Ways to Set the Initial PE File Name. *Theory and Practice of Forensic Science*. 2008;3(11):200–205. (in Russ.)
17. Zhilin V.V., Safar'yan O.A. Rtificial Intelligence in Data Storage Systems. *Vestnik of Don State Technical University*. 2020; 20(2):196–200. DOI:10.23947/1992-5980-2020-20-2-196-200
18. Al-Kasassbeh M., Mohammed S., Alauthman M., Almomani A. Feature Selection Using a Machine Learning to Classify a Malware. *In: Gupta B., Perez G., Agrawal D., Gupta D. (eds) Handbook of Computer Networks and Cyber Security*. Springer, Cham; 2020. p.889–904. DOI:10.1007/978-3-030-22277-2_36
19. Padaryan V.A., Soloviev M.A., Kononov A.I. Modeling the operational semantics of machine instructions. *Proceedings of the Institute for System Programming of the RAS*. 2010;19:165–186. (in Russ.)
20. Wang T.-Y., Wu C.-H. Detection of packed executables using support vector machines. *Proceedings of the International Conference on Machine Learning and Cybernetics 10–13 July 2011, Guilin, China*. IEEE; 2011. p.717–722. DOI:10.1109/ICMLC.2011.6016774
21. Hubballi N., Dogra H. Detecting Packed Executable File: Supervised or Anomaly Detection Method? *Proceedings of the 11th International Conference on Availability, Reliability and Security, ARES, 31 August–2 September 2016, Salzburg, Austria*. IEEE; 2016. p.638–643. DOI:10.1109/ARES.2016.18
22. Choi Y.-S., Kim I.-K., Oh J.-T., Ryou J.-C. PE File Header Analysis-Based Packed PE File Detection Technique (PHAD). *Proceedings of the International Symposium on Computer Science and its Applications, 13–15 October 2008, Hobart, Australia*. IEEE; 2008. p.28–31. DOI:10.1109/CSA.2008.28
23. AL-Nabhani Y., Zaidan A.A., Zaidan B.B., Jalab H.A., Alanazi H.O. A new system for hidden data within header space for EXE-File using object oriented technique. *Proceedings of the 3rd International Conference on Computer Science and Information Technology, 9–11 July 2010, Chengdu, China*. IEEE; 2010. p.9–13. DOI:10.1109/ICCSIT.2010.5564461
24. Solovev M.A., Bakulin M.G., Makarov S.S., Manushin D.V., Padaryan V.A. Decoding of Machine Instructions for Abstract Interpretation of Binary Code. *Proceedings of the Institute for System Programming of the RAS*. 2019;31(6):65–88. DOI: 10.15514/ISPRAS-2019-31(6)-4 (in Russ.)
25. Wang M., Tang Y., Lu Z. Massive Similar Function Searching for Cross-Architecture Binaries. *Proceedings of the 18th International Symposium on Distributed Computing and Applications for Business Engineering and Science, DCABES, 8–10 November 2019, Wuhan, China*. IEEE; 2019. p.195–198. DOI:10.1109/DCABES48411.2019.00055
26. Buinevich M.V., Izrailov K.E. Algorithmization Method for Machine Code of Telecommunication Devices. *Telekommunikatsii (Telecommunications)*. 2012;12:2–6. (in Russ.)
27. Buinevich M.V., Izrailov K.E. Automated Tool for Algorithmic Machine Code of Telecommunication Devices. *Telekommunikatsii (Telecommunications)*. 2013;6:2–9. (in Russ.)
28. Buinevich M., Izrailov K. A Generalized Model of Static Analysis of Program Code Based on Machine Learning for the Vulnerability Search Problem. *Informatizatsiya i Svyaz'*. 2020;2:143–152 (in Russ.) DOI:10.34219/2078-8320-2020-11-2-143-152
29. Poddubnyy V., Korkin I. Advanced Rootkit Detection Using Memory Forensics. Voprosy Kiberbezopasnosti. 2019;5(33): 75–82. DOI:10.21681/2311-3456-2019-5-75-82 (in Russ.)

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, старший научный сотрудник Санкт-Петербургского федерального исследовательского центра Российской академии наук, konstantin.izrailov@mail.ru
https://orcid.org/0000-0002-9412-5693