СИСТЕМЫ И СРЕДСТВА ИНФОРМАТИКИ, 2013. Т. 23. № 1. С. 69–79

ОСОБЕННОСТИ РЕАЛИЗАЦИИ АНАЛИЗАТОРА СЕТЕВОГО ТРАФИКА С ЦЕЛЬЮ ОБНАРУЖЕНИЯ ВРЕДОНОСНОГО ИСПОЛНИМОГО КОДА НА РЕКОНФИГУРИРУЕМОМ ВЫЧИСЛИТЕЛЕ

***М. Н. Самойлов*[1], *Д. Ю. Гамаюнов*[2], *С. О. Беззубцев*[3], *М. А. Булгаков*[4]**

Аннотация: Рассмотрен один из методов улучшения существующего алгоритма фильтрации вредоносного сетевого трафика для использования на высокоскоростных каналах передачи данных. Алгоритм Racewalk используется как базовый. Главная идея улучшения состоит в переносе части вычислений на специализированное устройство. Приведены результаты экспериментального тестирования на оборудовании с FPGA (field-programmable gate array — прграммируемая пользователем вентильная матрица) Virtex 6 для канала с пропускной способностью 1 Гбит/с, которые дают основание полагать, что данный метод можно применять для более скоростных интерфейсов.

Ключевые слова: Racewalk; фильтрация трафика; ПЛИС; шеллкоды; сетевой трафик

1 Введение

Обеспечение безопасности сетевого обмена — одна из задач, возникающих в рамках сложных (в том числе и сетевых) вычислительных комплексов, а также в сети Интернет. Для ее решения применяется фильтрация и отсечение вредоносного сетевого трафика.

К одному из видов вредоносного трафика относятся так называемые шеллкоды (shellcodes) [1]. Шеллкод — это последовательность исполнимого машинного кода, эксплуатирующая уязвимость переполнения буфера ресивера программы, реализующей сетевой сервис, с целью нарушения конфиденциальности, целостности или доступности сервиса или вычислительной системы.

[1]Московский государственный университет им. М. В. Ломоносова, факультет вычислительной математики и кибернетики, samoylov@lvk.cs.msu.su

[2]Московский государственный университет им. М. В. Ломоносова, факультет вычислительной математики и кибернетики, gamajun@cs.msu.su

[3]Институт точной механики и вычислительной техники им. С. А. Лебедева Российской академии наук, stas.bezzubtsev@gmail.com

[4]Московский государственный университет им. М. В. Ломоносова, факультет вычислительной математики и кибернетики, bulgakov@lvk.cs.msu.su

Применяемые методы фильтрации шеллкодов сложны в вычислительном, а некоторые и в пространственном, плане (см., например, [2, 3]). Обнаружение шеллкодов для сетевых каналов с пропускной способностью более 1 Гбит/с с использованием вычислителей общего назначения [4, 5] затруднено или невозможно в силу особенностей организации внутренних шин и их пропускных возможностей, особенностей организации доступа процессоров к оперативной памяти, а также сопутствующих накладных расходов, связанных с работой операционной системы (ОС) и организацией ввода–вывода.

В данной статье рассматривается один из методов решения задачи обнаружения шеллкодов и фильтрации сетевого трафика и его реализация на программируемых логических интегральных схемах (ПЛИС) Xilinx Virtex-6. Разработанный фильтр может быть приспособлен для фильтрации вредоносных исполняемых последовательностей в таких процессорных архитектурах, как IA-64, MIPS, ARM, SPARC, JVM, что позволит реализовать фильтрацию вредоносного трафика для мобильных устройств операторами мобильной связи.

2 Фильтрация сетевого трафика

При реализации фильтра сетевого трафика на базе электронной вычислительной машины (ЭВМ) общего назначения возникают следующие проблемы:

1. Прием и передача сетевого трафика на ЭВМ реализуется периферийными устройствами ввода–вывода. Обслуживание периферийного устройства со стороны ОС сводится к передаче данных из буфера ресивера сетевого устройства в оперативную память ЭВМ, организации доступа к данным из прикладной программы анализатора, анализу полученных данных, а также к связанным с этим операциям ОС по планированию процессов и переключению между ними, обработке данных.
2. Вычислители общего назначения жестко привязаны к определенному (пусть и полному с точки зрения вычислений) набору команд, который физически не может предоставить команды для эффективной реализации специфической обработки данных в прикладных программах (например, подсчет контрольной суммы за одну операцию процессора). Не менее существенными ограничениями служат жесткая структура и небольшой объем встроенной памяти (регистров) процессоров общего назначения. Данная особенность не позволяет эффективно реализовать обработку потоковых данных из-за высокого процента кеш-промахов, приводящих к падению фактической эффективной производительности процессора до производительности оперативной памяти.

Из вышеизложенного можно заключить, что естественное желание реализовать анализ сетевого трафика в реальном масштабе времени на скорости канала трудно реализуемо на архитектурах ЭВМ общего назначения по причине высо-

ких накладных расходов и недетерминизма поведения программно-аппаратного комплекса, компонентов ОС.

Нельзя не отметить, что системы на базе вычислителей общего назначения занимают физически больше места, а их энергопотребление, как правило, в разы выше, чем у встроенных систем, ориентированных на решение конкретной задачи и показывающих характеристики производительности при их решении не ниже, чем ЭВМ.

3 Обзор

Типичный шеллкод реализует атаку на уязвимость памяти в некотором приложении, например атаку на переполнение стека, кучи, и, как следствие, имеет вполне определенную структуру, предопределенную типом используемой уязвимости. Современные ОС содержат специализированные механизмы защиты от атак на переполнение буфера, такие как ASLR (address space layout randomization), запрет исполнения стека, рандомизация адресов функций библиотек. В силу этого для конкретной уязвимости чаще всего нельзя точно предсказать адрес, по которому будет загружено тело шеллкода. Для увеличения вероятности успешной эксплуатации уязвимости в шеллкод внедряют большие последовательности инструкций, которые «ничего не делают», — их единственное предназначение заключается в том, чтобы при передаче управления в любую точку внутри такой последовательности исполнение кода дошло до «полезной нагрузки» шеллкода. Такие последовательности называют NOP-следом. Как правило, NOP-след имеет значительную длину в сотни или даже тысячи байтов, корректно дизассемблируется с каждого байта (иными словами, последовательность байтов, полученная путем отбрасывания любого числа байтов из начала исходной последовательности, представляет собой корректную последовательность инструкций целевого вычислителя).

Задачу поиска вредоносных шеллкодов можно свести к задаче поиска подстрок специального вида во входной строке.

На практике используется три класса алгоритмов, решающих задачу обнаружения шеллкодов: статический анализ, динамический анализ и гибридные алгоритмы, сочетающие в себе элементы обоих подходов.

Суть статического анализа [1, 6] заключается в принятии решения о вредоносности проверяемой последовательности на основе анализа входной последовательности как текста программы на машинном языке (без исполнения). Как правило, в ходе статического анализа проверяются некоторые эвристики. Примером такой эвристики является наличие в анализируемой последовательности NOP-следа [6], т. е. некоторой подпоследовательности, которую можно корректно дизассемблировать и исполнить с любого смещения относительно начала подпоследовательности. Преимуществами данного подхода являются достаточно

низкая как вычислительная, так и пространственная сложность, явная ориентированность на поточную обработку данных без необходимости буферизации. К недостаткам можно отнести высокий уровень ложных срабатываний.

Суть динамического анализа [7] заключается в эмуляции или исполнении анализируемой последовательности на исполнителе целевой архитектуры и вычислении эвристик относительно поведения анализируемого кода. Данный способ является более точным, но и более вычислительно сложным по сравнению со статическим анализом.

Применение вычислителя, оптимизированного под нужды конкретного алгоритма анализа данных, позволяет снять большинство проблем, связанных с реализацией сетевых фильтров, ориентированных на поиск шеллкодов. Одной из технологий, обеспечивающих создание вычислителя, специально приспособленного для решения пользовательских задач, является технология ПЛИС.

Применение ПЛИС, в силу возможности создания полностью специализированной и ориентированной на конкретную задачу архитектуры, позволяет обеспечить высокий уровень параллелизма вычислений, компактность и сравнительно низкое энергопотребление по сравнению с «классическими» процессорами.

4 Формальная модель

Сетевой обмен представим в виде следующей модели.

Трафик — неограниченная последовательность байтов данных, следующих в одном направлении: от отправителя к получателю.

Фрагмент трафика — ограниченная последовательность байтов данных, являющаяся подпоследовательностью соответствующего **трафика** или **фрагмента трафика**.

Пустой фрагмент трафика — последовательность байтов данных, не содержащих ни одного элемента.

Сетевой пакет — фрагмент трафика, состоящий из служебных данных и полезной нагрузки. В рамках данной модели под служебными данными понимается фрагмент трафика сетевого пакета фиксированной длины, а под полезной нагрузкой понимается фрагмент трафика сетевого пакета, включающий в свой состав все байты данных сетевого пакета за вычетом байтов данных, принадлежащих фрагменту служебных данных.

Сетевой трафик — **трафик**, составленный из байтов полезной нагрузки сетевых пакетов, принадлежащих строго одному исходному трафику, причем байты данных полезной нагрузки упорядочены в соответствии с их порядком следования в исходном трафике, а также с порядком байтов в каждом соответствующем сетевом пакете. Каждый **фрагмент сетевого трафика** отвечает некоторому множеству сетевых пакетов. Каждый сетевой пакет (его полезная нагрузка) отображается на **фрагмент сетевого трафика**.

Вредоносный фрагмент трафика — это **фрагмент сетевого трафика** наибольшего размера, такой что его обработка на заданной вычислительной системе приводит к нарушению свойств целостности, безопасности и доступности вычислительной системы. Пример **вредоносного фрагмента сетевого трафика** — это шеллкод, отправленный в содержании сетевого пакета.

Легитимный (невредоносный) фрагмент (трафика) — это **фрагмент трафика** не являющийся **вредоносным**.

Безопасный сетевой трафик — **сетевой трафик**, не содержащий в себе **вредоносных фрагментов**.

Фильтр сетевого трафика — сущность, принимающая на вход **сетевой трафик** и преобразующая его в **чистый сетевой трафик** путем формирования **сетевого трафика**, не содержащего **фрагментов**, на которые **отображаются** сетевые пакеты, которым **отвечают обнаруженные вредоносные фрагменты сетевого трафика**, и содержащего остальные **фрагменты** исходного **сетевого трафика**.

В данной статье рассматриваются особенности реализации **фильтра сетевого трафика** на модуле ПЛИС.

Стоит отметить, что ограничение на фиксированность длины служебных данных не является жестким и может быть снято. Оно введено лишь с целью упрощения описания задачи и, как следствие, не накладывает ограничений на область применимости предложенного в данной работе метода.

5 Алгоритм решения

В качестве алгоритма решения поставленной задачи в данной работе был выбран алгоритм статического анализа сетевого трафика Racewalk [6]. Основная идея этого алгоритма состоит в поиске и выделении NOP-эквивалентных последовательностей из входящего потока и последующая классификация каждой выявленной последовательности с помощью методов машинного обучения (рис. 1).

Поиск NOP-следа реализуется на специализированном вычислителе (на базе ПЛИС), как, например, в работах [2–5]. Для решения задачи поиска NOP-следа используется декодирование команд процессорной архитектуры IA-32 [8], в результате которого для каждого участка входной последовательности устанавливается тип инструкции. Если в результате последовательность заданного размера с каждого смещения декодируется в корректные инструкции, притом не входящие в число привилегированных инструкций процессора x86, то такая последовательность считается NOP-следом.

Классификация обнаруженной последовательности выполняется непосредственно на ЭВМ. Для классификации применяется алгоритм опорных векторов (SVM — support vector machine).

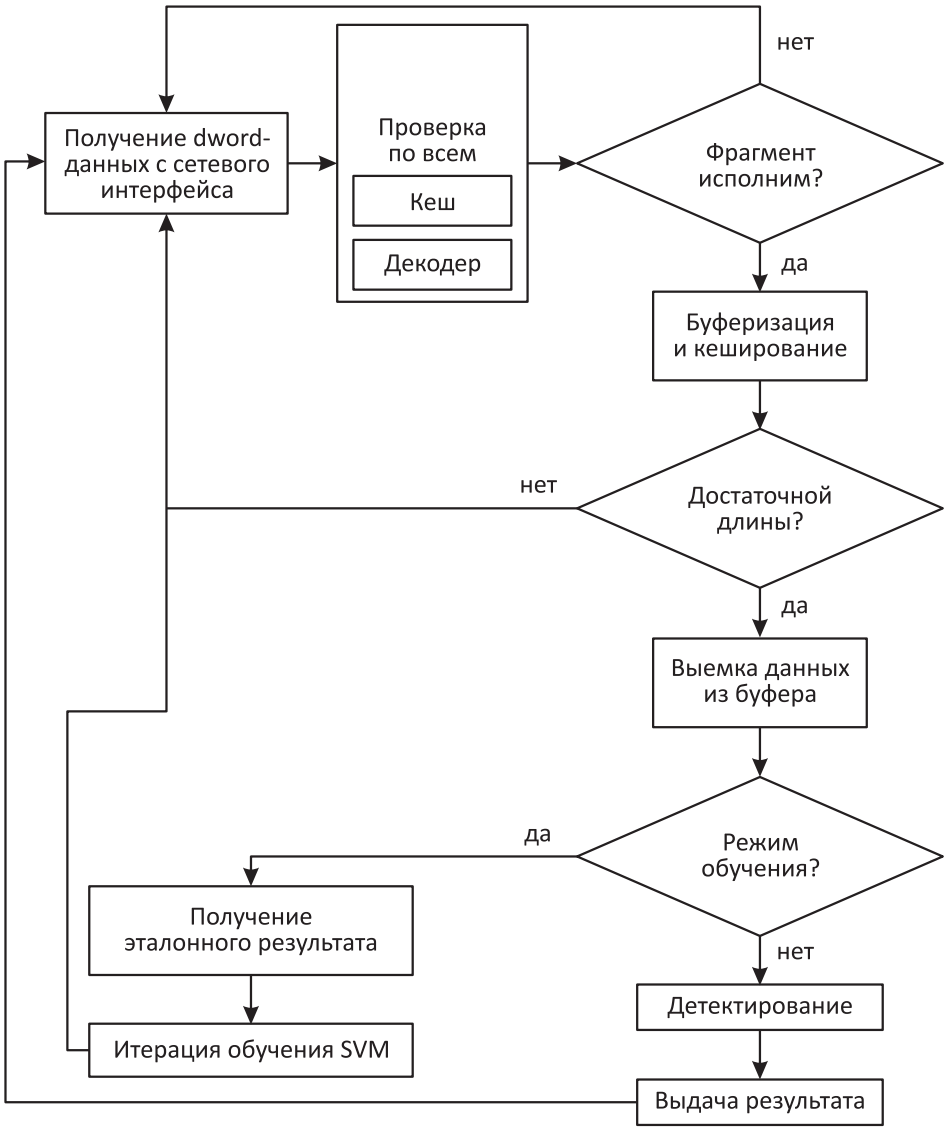

Рис. 1 Блок-схема алгоритма Racewalk

6 Описание реализации

В рамках данной работы была произведена профилировка программной реализации алгоритма Racewalk с целью выявления узких мест и «высоконагруженных» (т. е. наиболее часто выполняющихся) участков программной реализации. По результатам проведенного исследования была спроектирована и сконструирована программно-аппаратная реализация, призванная ускорить работу программы и оптимизировать процесс вычислений (рис. 2).

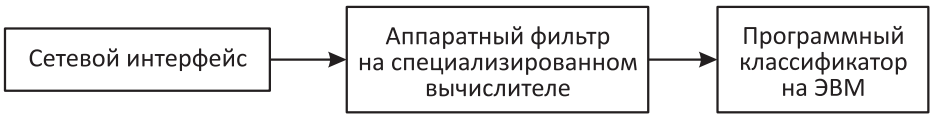

Рис. 2 Общая логическая схема

В качестве специализированного вычислителя использовалась плата Xilinx ML605 [9], построенная на базе ПЛИС Xilinx Virtex-V6-240LXT, снабженная сетевым интерфейсом стандарта Gigabit Ethernet и системным интерфейсом стандарта PCI-Express x8. В рамках работы был реализован драйвер для ОС Linux, предоставляющий интерфейсы сетевого устройства и вспомогательный интерфейс символьного устройства для управления фильтром и сбора статистики.

Разработанный для ПЛИС фильтр NOP-следов функционирует на частоте 250 МГц и реализует декодирование входящего потока данных. При этом определение типа одной инструкции занимает 1 такт работы устройства, т. е. на обнаружение 64-байтного шеллкода тратится 64 такта работы устройства (см. рис. 3 и 4).

Обнаруженный шеллкод помещается в буфер обмена, откуда по интерфейсу PCI-Express считывается приложением, выполняющимся на ЭВМ общего назначения.

Для реализации функции фильтра задействуется менее 1% ресурса макроячеек указанной ПЛИС и менее 1% ресурса внутренней памяти ПЛИС Virtex-6-240LXT.

7 Экспериментальное исследование реализации

Тестирование разработанного фильтра производилось на стенде, состоящем из ЭВМ фильтра и вспомогательной ЭВМ генератора, объединенных в сеть. Электронная вычислительная машина фильтра — это ЭВМ общего назначения на базе 6-ядерного процессора Intel Xeon E8350 с 8 ГБ оперативной памяти. Сетевой трафик формировался с помощью вспомогательной ЭВМ, оборудованной

Рис. 3 Схема функционирования аппаратной реализации

сетевой картой Gigabit Ethernet. На специализированном оборудовании фильтра был реализован один фильтрующий элемент. В качестве целевой процессорной архитектуры исполняемых данных использовалась только IA-32, рассматривалась минимальная длина последовательности — 64 байта.

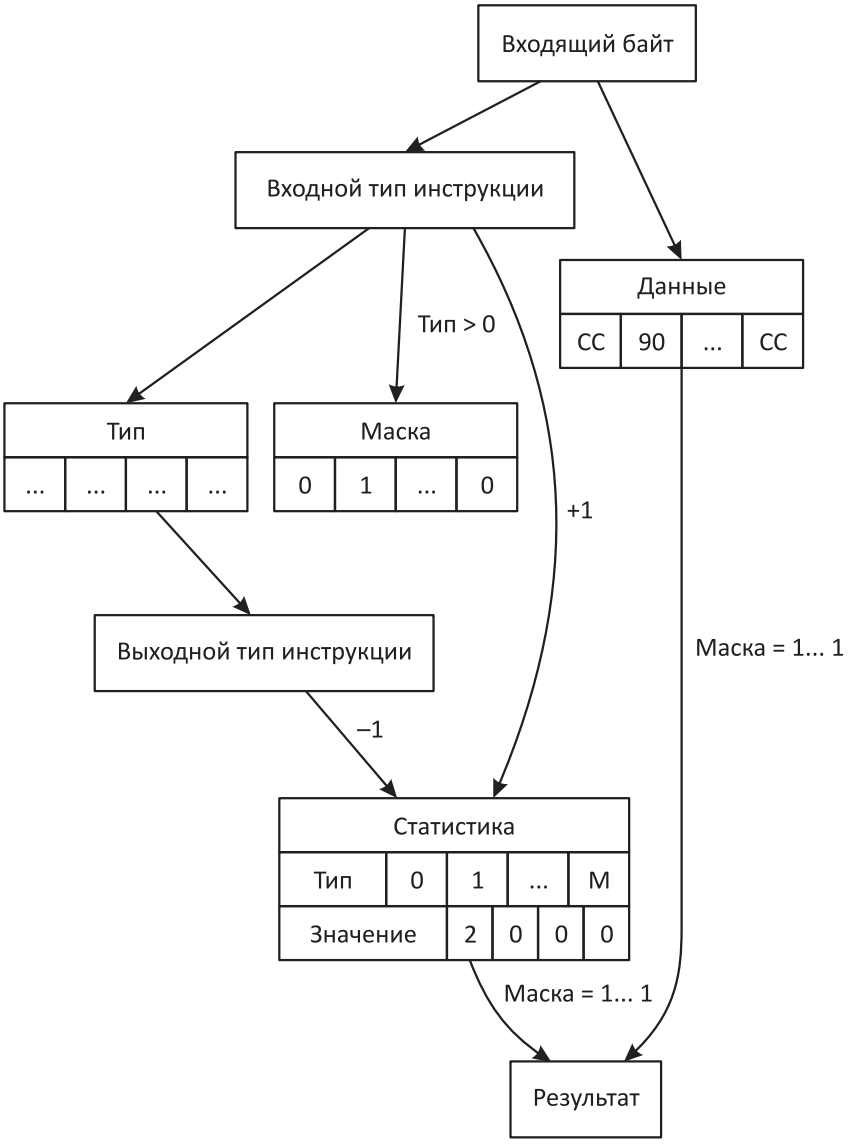

Рис. 4 Пример функционирования реализации

Для оценки эффективности полученной реализации было проведено тестирование и сравнение с программной реализацией. Сравнение проводилось по двум критериям — эквивалентности (т. е. степени сходства результатов выдачи на одинаковых входных последовательностях) и производительности (отношения числа обработанных пакетов к числу отправленных).

Тестовым набором служило несколько специально подготовленных дампов трафика достаточно большого размера (около 1,1 ГБ, 200 000 пакетов). Тестовые наборы составлялись с помощью набора утилит tcpdump [10] и включали в свой состав:

(1) вредоносный трафик — в каждом из пакетов (длиной 1400 байтов) содержалось по 3 различные последовательности шеллкодов длиной 64 байта, сгенерированные специальными средствами Metasploit Framework версии 4.1 [11] и ADMutate (в равных долях);

(2) легитимный трафик — дамп передачи по сети легитимных исполняемых файлов (exe-файлов), а также архивов с данными.

В канал было передано 600 000 шеллкодов, все они были обнаружены, фоновый трафик отсутствовал, тестирование проводилось на различных скоростях потока данных (от 4 до 1000 Мбит/с) со скоростью передачи данных интерфейса 1 Гбит/с, аппаратной реализацией были обнаружены все переданные примеры.

Производительность аппаратной реализации значительно превзошла показатели программной реализации алгоритма (600 000 обнаруженных шеллкодов против 19 000 на скорости 1 ГБ/с, рис. 5). На легитимном трафике обе реализации не выдавали ложных срабатываний.

Теоретическая пропускная способность совпала с пропускной способностью интерфейса Gigabit Ethernet. Оценка латентности фильтра дает право полагать, что наличие на канале сетевого фильтра, выполненного с применением использо-

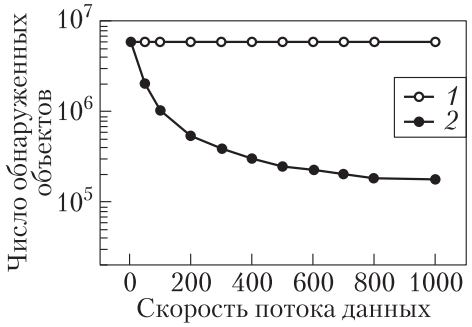

Рис. 5 Сравнение реализаций: *1* — аппаратной (ПЛИС FPGA-оборудование); *2* — программной (Racewalk)

ванных методов, не приведет к значительному снижению пропускной способности сетевого канала и, соответственно, скорости обмена данными.

8 Заключение

В рамках данной работы был разработан и реализован программно-аппаратный комплекс фильтра сетевого трафика на базе модуля ПЛИС, проведена его проверка и нагрузочное тестирование.

По результатам тестирования можно заключить, что на базе данной реализации возможна разработка фильтра для интерфейсов с большой пропускной способностью (10 Гб/с и выше) при сохранении основных показателей.

Литература

1. *Akritidis P., Markatos E. P., Polychronakis M., Anagnostakis K.* Stride: Polymorphic sled detection through instruction sequence analysis // 20th IFIP Information Security Conference (International). — Milano, 2008. P. 375–392.
2. *Madhusudan B., Lockwood J.* Design of a system for Real-TimeWorm Detection // IEEE Micro, 2005. Vol. 25. Iss. 1. P. 60–69.
3. *Chey Sh., Li J., Sheaffer W., Skadron K., Lach J.* Accelerating compute-intensive applications with GPUs and FPGAs // 6th IEEE Symposium on Application Specific Processors (SASP 2008). — Anaheim, 2008. P. 101–107.
4. *Loinig J., Wolkerstorfer J., Szekely A.* Packet filtering in gigabit networks using FPGAs // 15th Austrian Workshop on Microelectronics (Austrochip 2007). — Graz, 2007.
5. *Katashita T., Yamaguchi Y., Maeda A., Toda K.* FPGA-based intrusion detection system for 10 gigabit Ethernet // IEICE Trans. Information Systems, 2007. Vol. E90-D. No. 12. P. 1923–1931.
6. *Gamayunov D., Quan N. T. M., Sakharov F., Toroshchin E.* Racewalk: Fast instruction frequency analysis and classification for shellcode detection in network flow // 2009 European Conference on Computer Network Defense. — Milano, 2009. P. 4–12.
7. *Polychronakis M., Anagnostakis K. G., Evangelos P.* Network-level polymorphic shellcode detection using emulation // GI/IEEE SIG SIDAR Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA 2006). — Berlin, 2006. P. 54–73.
8. Intel 64 and IA-32 Architectures Software Developer's Manual. http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html.
9. Xilinx. http://www.xilinx.com.
10. Tcpdump & libpcap. http://www.tcpdump.org.
11. Metasploit. http://www.metasploit.com.