СПИСОК ЛИТЕРАТУРЫ:

1. *Ахо А., Ульман Дж.* Компиляторы: принципы, технологии и инструменты. М.: Издательский дом «Вильямс», 2003. — 768 с.
2. *Ахо А., Ульман Дж.* Теория синтаксического анализа, перевода и компиляции. М.: Мир, 1978.
3. *Холанд Г., Мак-Гроу Г.* Взлом программного обеспечения анализ и использование кода. М.: Издательский дом «Вильямс», 2005. — 400 с.: ил.
4. *Матросов А. А.* Курс лекций «Защита программного обеспечения». URL: http://aktivco.ru/course/lecture/.
5. *Краснопевцев А. А.* Разработка средств автоматизации для переноса байт-кода во внешний аппаратный модуль // Технологии Microsoft в теории и практике. М.: Вузовская книга, 2008. С. 139—141.
6. *Краснопевцев А. А., Букасов В. А.* Разработка средств автоматической защиты приложений, содержащих байт-код // Материалы 10-й Международной научно-практической конференции, Таганрог, 24—27 июня 2008. С. 15—16.

А. А. Станкевичус
Московский инженерно-физический институт (государственный университет)

ИНСТРУМЕНТАЛЬНОЕ СРЕДСТВО ПРОВЕРКИ БЕЗОПАСНОСТИ КОДА, ПРЕДНАЗНАЧЕННОГО ДЛЯ ИСПОЛНЕНИЯ В ГРИД-СЕТЯХ

В работе анализируются методы и средства защиты Грид-сетей от вредоносного кода, их преимущества и недостатки. Предлагается новое инструментальное средство, позволяющее в автоматическом режиме подтверждать безопасность программ на языке Си, предназначенных для Грид-сетей.

Введение

При работе с Грид-сетями существует угроза распространения вредоносного кода производителем прикладного ПО Грид. Вредоносный код может нарушать работу отдельных ресурсов Грид, а также Грид в целом, что определяет актуальность разработки средств защиты Грид от вредоносного кода. Задачей настоящей работы является разработка инструментального средства проверки безопасности кода, предназначенного для Грид-сетей.

1. Инструментальные средства защиты Грид-сетей от вредоносного кода

Проверка безопасности кода для Грид-сетей нередко производится вручную, однако многие специалисты считают целесообразным применение инструментальных средств.

Для защиты Грид-сетей от вредоносного кода могут быть применены антивирусные программы, такие как [1], но современные антивирусные средства во многих случаях не выявляют новые виды вирусов, а также вредоносное ПО, не являющееся вирусом или сетевым червем.

Для защиты Грид от вредоносного кода могут быть использованы верификаторы, например [2, 3]. Однако для успешного применения многих средств формальной верификации необходима формализация верифицируемых свойств для конкретной программы, что само по себе может представлять сложную задачу, требующую участия человека, заинтересованного в обеспечении безопасности узла Грид.

Для защиты Грид от вредоносного кода могут быть применены безопасные языки, такие как [4], и инструментальные средства на их основе. К сожалению, производительность программ на безопасных языках часто оказывается ниже производительности аналогичных программ на небезопасных языках, таких как язык Си. Применение разработчиками ПО безопасных языков требует их предварительного изучения. Перевод существующего кода на безопасный язык может быть длительной и дорогостоящей процедурой.

Для защиты Грид от вредоносного кода могут быть применены виртуальные машины [5, 6]. При этом производительность прикладного ПО, выполняемого на виртуальной машине, снижается. Кроме того, безопасность многих виртуальных машин может быть поставлена под сомнение [7].

Для защиты Грид от вредоносного кода применяются компоненты отдельных систем более широкого назначения, а также организационно-правовые санкции совместно со средствами аутентификации разработчиков и пользователей ПО [8]. Однако такие методы позволяют лишь выявить и покарать виновного в распространении вредоносного ПО, но не предотвратить атаки.

Существует потребность в специализированных средствах защиты Грид от вредоносного кода, их разработано относительно мало по сравнению с прочими средствами. Существующие средства не покрывают все необходимые требования, предъявляемые к средству проверки кода для узла Грид. Например, многие средства ориентированы на ПО на языке Java, в то время как существует потребность в верификации кода на других языках, в частности Си и Си++. Кроме того, существующие средства не учитывают специфику программ для вычислительного узла Грид. Многие существующие средства не позволяют проводить верификацию в автоматическом режиме.

2. Концепция инструментального средства

На рис. 1 приведена концептуальная схема работы средства проверки безопасности кода, предназначенного для исполнения в Грид-сети. Средство проверки содержит транслятор, верификатор и компилятор, основанный на безопасной библиотеке. Транслятор преобразует исходный код прикладного программного обеспечения, предназначенный для Грид, на языке Си в код на подмножестве Си, использующий безопасную библиотеку. Полученный код проверяется верификатором. Результатом работы верификатора является безопасный код на подмножестве языка Си, использующий защищенную библиотеку, либо сообщение о том, что безопасность кода не может быть подтверждена с указанием причины затруднения. Безопасный код компилируется компилятором в безопасный исполнимый код, который может быть исполнен на вычислительном узле Грид.

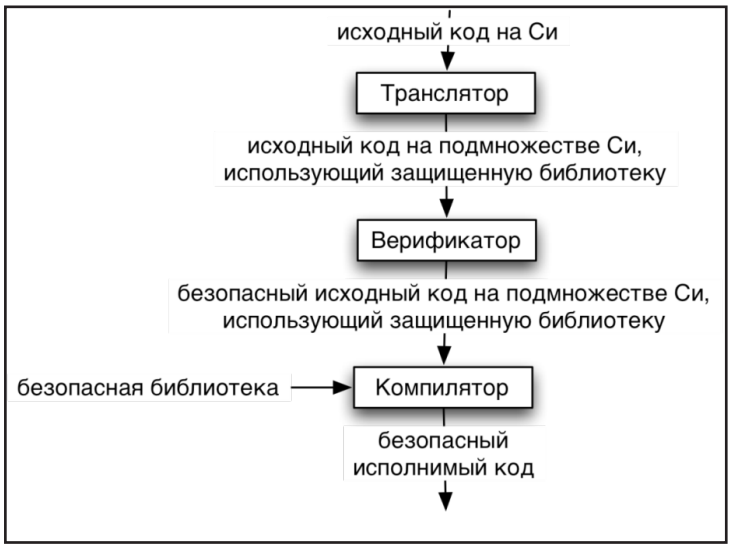

Рис. 1. Концептуальная схема работы средства проверки безопасности кода

Безопасная библиотека — это библиотека функций, которые могут быть использованы в программном обеспечении для Грид и использование которых гарантированно безопасно для вычислительного узла Грид. Безопасная библиотека содержит функции, позволяющие производить: остановку работы программы, выделение и освобождение памяти в общем пуле, выделение, чтение, запись и освобождение памяти в безопасном пуле, открытие, чтение, запись, закрытие и перемещение по файлу указателя чтения-записи, получение и передачу данных диспетчеру Грид. На рис. 2 приведена общая схема взаимодействия с безопасной библиотекой. Безопасная библиотека реализуется на языке Си и для повышения производительности результирующего ПО компилируется и компонуется вместе с исходным кодом прикладного ПО Грид. Проверка

безопасности библиотеки производится в автоматизированном режиме с применением методов и существующих инструментов формальной верификации.

3. Алгоритм выявления потенциально вредоносного ПО

Выявление потенциально вредоносного ПО производится путем абстрактной интерпретации [9] программы на языке Си и проверки невозможности нарушения этим ПО требований безопасности при произвольных входных данных. Проверяемая программа переводится в представление в виде направленного графа, узлы которого соответствуют линейным участкам программы, а дуги — ветвлениям потока управления. Переменные языка Си представляются в виде описания диапазона значений, которые данная переменная может принимать при различных путях исполнения и различных входных данных. При этом используется абстрактное описание множества значений, включающего все принимаемые переменной значения. В начале работы алгоритма все переменные, используемые в каждом из линейных участков программы, предполагаются способными иметь произвольные значения. Далее над диапазонами вероятных значений производятся операции, соответствующие операциям, выполняемым над переменными программы. В случае невыполнения требований безопасности на каком-либо участке участок помечается как потенциально опасный. Далее рассматриваются все дуги графа, ведущие в потенциально опасные участки, и производится уточнение диапазонов вероятных значений переменных на входе каждого из таких потенциально опасных участков. В случае, когда в результате такого уточнения потенциально опасный участок остается потенциально опасным, рассматриваются все дуги графа, ведущие в данный узел, и процесс повторяется. Завершение процесса производится в случае достижения заданной глубины анализа либо при превышении заданного времени, отводимого на проверку. Участки, потенциальную опасность которых не удается опровергнуть, снабжаются проверками времени исполнения. Таким образом, результирующая программа, снабженная в случае необходимости проверками времени выполнения, удовлетворяет требованиям безопасности.

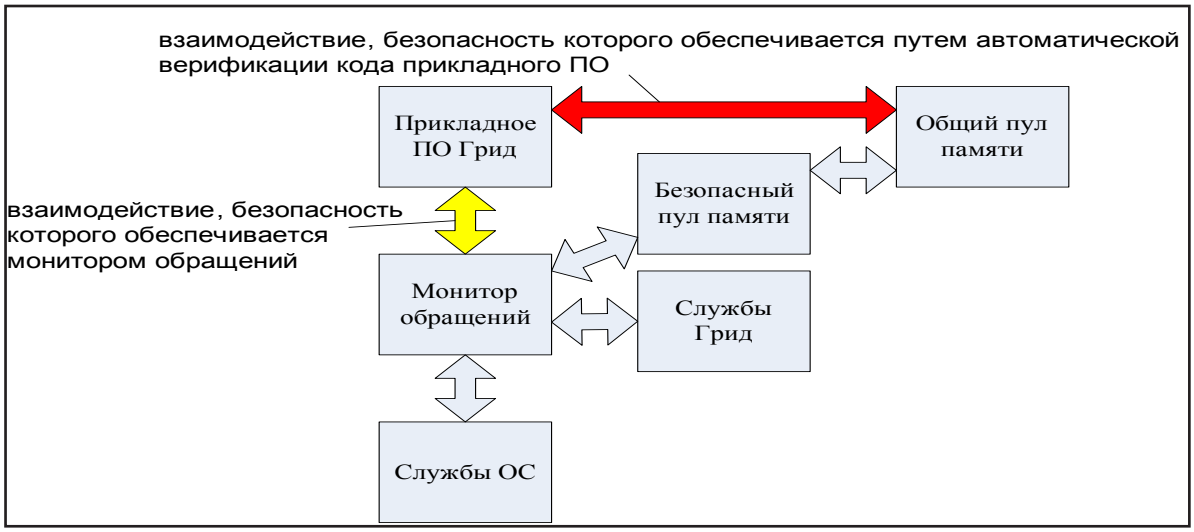

Рис. 2. Общая схема взаимодействия с безопасной библиотекой

4. Динамическая проверка безопасности ПО

В случаях, когда статическая проверка не позволяет убедиться в безопасности работы с выделенной памятью, производится трансляция выделения и освобождения памяти в вызов функций безопасной библиотеки, производящих выделение памяти из безопасного пула. Трансляция чтения и записи в память необходима в случаях, когда выделение памяти произведено в безопасном пуле. При этом операция чтения/записи данных преобразуется в вызов функции безопасной библиотеки. Трансляция создания переменных в стеке, трансляция чтения стековых переменных, трансляция

записи в стековые переменные не рассматриваются в настоящей работе и могут послужить темами для отдельных работ. Таким образом, за счет применения динамической проверки может быть обеспечена возможность безопасного применения многих программ, статическая проверка безопасности которых по тем или иным причинам не может быть произведена.

Заключение. Предложенное средство позволяет в автоматическом режиме подтверждать безопасность программ на языке Си, предназначенных для Грид-сетей. Такое средство, в отличие от своих аналогов, позволяет автоматически верифицировать код на языке Си, учитывая специфику программ для Грид. Полученные в работе новые результаты могут послужить основой для разработки системы защиты Грид-сетей от вредоносного кода.

СПИСОК ЛИТЕРАТУРЫ:

1. Clam AntiVirus. URL: http://www.clamav.net/.
2. The ASTRÉE Static Analyzer. URL: http://www.astree.ens.fr/.
3. OWASP SWAAT Project. URL: http://www.owasp.org/index.php/Category:OWASP_SWAAT_Project.
4. CCured documentation. URL: http://manju.cs.berkeley.edu/ccured/.
5. *Calder Brad, Chien Andrew A., Wang Ju, Yang Don.* The entropia virtual machine for desktop grids // Proceedings of the 1st ACM/USENIX International Conference on Virtual Execution Environments, June 11–12, 2005, Chicago, IL, USA.
6. *Barham P., Dragovic B., Fraser K., Hand S., Harris T., Ho A., Neugebauer R., Pratt I., Warfield A.* Xen and the art of virtualization // SOSP '03: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles. NY, USA, 2003. ACM Press. P. 164–177.
7. *Saraswat Vijay.* Java is not type-safe. URL: http://www.cis.upenn.edu/~bcpierce/courses/629/papers/Saraswat-javabug.html.
8. Kerberos Leveraged PKI. URL: http://www.citi.umich.edu/projects/kerb_pki/.
9. *Cousot Patrick.* Abstract interpretation // ACM Computing Surveys (CSUR). Vol. 28. № 2. June 1996. P. 324–328.