ресурсоемких средств (хэш-функций и цифровой подписи). При идентификации файла оптимизация возможна только в случае его несоответствия списку разрешенных. Т. е. первыми должны быть проверены наиболее простые характеристики: имя, размер и т. п., и только в случае успеха должен быть проверен хэш-код. Вообще, для выбора метода контроля для различных файлов можно разделить все исполняемые файлы, сценарии и библиотеки на три группы:

— часто загружаемые в память;

– исполняемые файлы ОС и другие исполняемые файлы, изменение которых (перемещение, удаление, обновление) происходит редко;

– исполняемые файлы, непосредственно связанные с производственной деятельностью сотрудников, и прочие исполняемые файлы.

Таким образом, при создании политики запуска приложений следует начинать с контроля на основе доверенных каталогов. Этим методом, прежде всего, должны быть охвачены стандартные библиотеки и другие файлы первой и второй групп. В силу того, что этот подход к проверке наиболее производителен, его нужно использовать настолько широко, насколько это возможно, но не в ущерб удобству работы с системой безопасности, учитывая возможность изменения структуры каталогов и доверия к исполняемым файлам. Для оставшейся части файлов должна быть создана политика, основанная на перечне разрешенных к запуску файлов.

Таким образом, модель разграничения доступа и идентификации исполняемых файлов определяет архитектуру системы предотвращения несанкционированного запуска ПО. Предложенный метод идентификации сочетает в себе преимущества существующих подходов и в то же время позволяет избежать некоторых неудобств, связанных с ними.

СПИСОК ЛИТЕРАТУРЫ:

1. Инсайдерские угрозы в России 2009. URL: http://perimetrix.ru/downloads/rp/PTX_Personal_Data_2009.pdf.
2. СТО БР ИББС-1.0-2008. Обеспечение информационной безопасности организаций банковской системы Российской Федерации. Общие положения.

Д. М. Тимовский

СОЗДАНИЕ СИСТЕМЫ АВТОМАТИЧЕСКОЙ ЗАЩИТЫ ИСПОЛНЯЕМЫХ ФАЙЛОВ ПОД ОС LINUX

Целью данной работы было создание системы автоматической защиты исполняемых файлов для ОС Linux. В процессе работы были исследованы существующие системы автоматической защиты исполняемых файлов (результаты были оформлены в виде сравнительной таблицы):

· ElfCrypt;
· UPX;
· Burneye;
· Shiva.

На основании полученных результатов были выявлены 2 работающие на современных ядрах системы защиты: ElfCrypt и UPX, к недостаткам которых можно отнести следующие:

· отсутствие антиотладочных приемов;

· отсутствие приемов против трассировки процесса;

· отсутствие противодействия отладке уже запущенного процесса;

· наличие всего лишь 1 метода противодействия статическому дизассемблированию, к тому же поддающегося обходу в автоматическом режиме (изменение значения символа _start для ElfCrypt).

Был проведен подробный анализ известных в открытых источниках антиотладочных приемов и приемов против статического дизассемблирования, а также разработаны антиотладочные приемы для ОС Linux, ранее не реализованные в открытых источниках печати:

· манипуляции с отладочными регистрами;

· манипуляции с флагом трассировки TF;

· цепочки самотрассирующихся процессов;

· поиск отладчика в памяти;

· сброс регистра gs;

· эмуляция инструкций;

· расшифрование по требованию.

Проанализировано время выполнения отдельных приемов при отладке и при нормальном исполнении и реализован антиотладочный прием с использованием rdtsc.

Создана система автоматической защиты исполняемых файлов под ОС Linux с расширяемой модульной архитектурой, реализованы в виде подключаемых модулей к разработанной системе исследованные в данной работе антиотладочные приемы и приемы против статического дизассемблирования. С целью выявления преимуществ и недостатков созданной системы была разработана методика тестирования, в соответствии с которой было выполнено тестирование разработанной системы и существующих работающих аналогов на нескольких примерах, в том числе на утилитах, входящих в стандартный набор поставки дистрибутива Debian Lenny. Все исполняемые модули, переданные на защиту разработанной системе, были успешно защищены. Критерии успешности:

· сохранение первоначальной функциональности защищаемой программы;

· затруднение статического и динамического анализа исполняемого модуля.

К преимуществам разработанной системы по сравнению с существующими решениями следует отнести:

· противостояние отладке уже запущенного процесса;

· противодействие снятию дампа запущенного процесса путем использования технологии расшифровки по требованию;

· эмуляция инструкций;

· возможность расширяемости путем подключения новых модулей;

· большое количество уже разработанных подключаемых модулей с антиотладочными приемами и приемами против статического дизассемблирования.

К недостаткам разработанной системы относятся:

· сложность создания новых подключаемых модулей (процесс требует серьезных знаний в области низкоуровневого системного программирования под ОС Linux);

· малое количество устанавливаемых пользователем параметров защиты;

· отсутствие удобного графического интерфейса.

Разработанная система автоматической защиты исполняемых файлов может применяться в следующих областях:

· защита от копирования и нелегального распространения ПО;

· защита от реверс-инжиниринга ПО.

Планы на дальнейшее развитие:

· добавление в систему модуля профилирования выделенных в результате дизассемблирования функций;

· упрощение процесса создания новых подключаемых модулей (создание модулей на языках высокого уровня, добавление скриптового движка);

· увеличение задаваемых пользователем параметров, влияющих на процесс защиты ПО;

· добавление модулей для защиты других форматов исполняемых файлов (PE, Mach-O);

· создание удобного графического интерфейса.

СПИСОК ЛИТЕРАТУРЫ:

1. *Стивенс У. Р., Рago С. А.* UNIX. Профессиональное программирование. 2-е изд. СПб.: Символ-Плюс, 2007.
2. *Лав Р.* Разработка ядра Linux. 2-е изд. М.: ООО «И.Д. Вильямс», 2006.
3. *Щелкунов Д. А.* Разработка методик защиты программ от анализа на основе запутывания кода и данных. М.: МГТУ им. Н. Э. Баумана, 2009 (рукопись).
4. *Лившиц Ю.* Запутывание (обфускация) программ. Обзор. 2004. URL: http://logic.pdmi.ras.ru/~yura/of/survey1.pdf.
5. *Тимовский Д.* Исследование механизмов защиты от динамического анализа для ОС Linux. М.: МИФИ, 2008 (рукопись).
6. Tool Interface Standart (TIS) Executable and Linking Format (ELF) Specification Version 1.2. URL: http://www.x86.org/ftp/manuals/tools/elf.pdf.
7. *Barak B., Goldreich O., Impagliazzo R., Rudich S., Sahai A., Vadhan S., Yang K.* On the (Im)possibility of Obfuscating Programs. LNCS, 2001, 2139. P. 1–18.
8. *Collberg, Thomborson C., Townsend G. M.* Dynamic graph-based software watermarking. Technical Report TR04-08, April 2004.
9. *Lynn B., Prabhakaran M., Sahai A.* Positive Results and Techniques for Obfuscation // Eurocrypt, 2004.
10. *Varnovsky N. P. and Zakharov V. A.* On the Possibility of Provably Secure Obfuscating Programs // In Andrei Ershov Fifth International Conference. July, 2003. P. 91–102. Springer LNCS 2890, 2003.
11. *Chow S., Gu Y., Johnson H., Zakharov V.* An approach to the obfuscation of control-flow of sequential computer programs // LNCS, 2001, 2200. P. 144–155.
12. Ilo. Process Dump and Binary Reconstruction. URL: http://www.phrack.org/issues.html?issue=63&id=12#article.
13. ELFsh crew. Embedded ELF Debugging. URL: http://www.phrack.org/issues.html?issue=63&id=9#article.
14. *Vanegue J., Garnier T., Auto J., Roy S., Lesniak R.* Next generation debuggers for reverse engineering. URL: http://www.blackhat.com/presentations/bh-europe-07/ERSI/Whitepaper/bh-eu-07-ersi-WP-apr19.pdf.

М. В. Тимонин

ОПТИМИЗАЦИЯ СТРАТЕГИИ ИНВЕСТИРОВАНИЯ В СИСТЕМУ ЗАЩИТЫ ИНФОРМАЦИИ В МОДЕЛЯХ, ОСНОВАННЫХ НА ТЕОРИИ НЕЧЕТКОЙ МЕРЫ

Риск, связанный с информационной безопасностью (ИБ) организации, является многомерным сложным понятием, включающим множество связанных друг с другом переменных. Основой построения модели риска является его декомпозиция на логические составляющие, их оценка и агрегация информации снизу вверх для расчета величины риска. В качестве математического аппарата, позволяющего моделировать взаимосвязь между компонентами, возможно использовать