Раздел III
Защита информационных процессов в компьютерных системах

Н.П. Варновский, В.А. Захаров
Россия, г. Москва,
Московский государственный университет им. М.В. Ломоносова
Н.Н. Кузюрин, А.В. Шокуров
Институт системного программирования РАН

СОВРЕМЕННЫЕ МЕТОДЫ ОБФУСКАЦИИ ПРОГРАММ: СРАВНИТЕЛЬНЫЙ АНАЛИЗ И КЛАССИФИКАЦИЯ

В 1997 г. была опубликована работа [1], в которой проблема обфускации программ была впервые выделена как новая перспективная задача программирования, решение которой позволит получить простые и эффективные методы информационной защиты программного обеспечения. Результаты изучения этой задачи свидетельствуют о том, что за прошедшие десять лет вокруг нее сформировалась развитая проблематика и методология, и это позволяет с уверенностью говорить о том, что проблема обфускации программ стала самостоятельной отраслью исследования в области информационной безопасности. Цель настоящей статьи – дать краткий обзор наиболее значительных достижений в изучении задачи обфускации программ и наиболее важных направлений ее исследования.

Задача *обфускации программ* заключается в разработке таких преобразований, которые сохраняют функциональные характеристики программ, но при этом делают невозможным или чрезвычайно трудоемким извлечение из открытого текста программы ключевой информации об устройстве содержащихся в ней алгоритмов и структур данных. Именно сочетание этих двух противоположных качеств – общедоступного кода программы (синтаксиса) и защищенного ее содержания (семантики) – открывает обфускирующим преобразованиям программ широкие возможности применения в криптографии и компьютерной безопасности.

Обфускация программ может иметь многочисленные приложения в криптографии: ее можно использовать для преобразования криптосистем с секретным ключом в криптосистемы с открытым ключом [2-4], для проведения вычислений над зашифрованными данными (гомоморфных вычислений) [2,5], для замещения модели случайного оракула в криптографических протоколах [2,5], для обеспечения безопасности в иерархических системах распределенного доступа [5], для создания верифицируемых систем тайного голосования [6] и др.. Во всех перечисленных криптографических приложениях от обфускации программ требуется столь же высокая стойкость, как и от традиционных криптографических систем, используемых для решения подобных задач. Впервые строгое математическое определение стойкости обфускации программ было предложено в работе [2]: обфускация считается стойкой, если всякий противник может извлечь из текста обфускированной программы за разумное (полиномиальное) время не больше информации, чем можно было бы получить, проводя тестовые испытания этой программы как «черного ящика». В этой же работе было установлено существование таких программ, для которых подобная стойкость обфускации в принципе не достижима. Впоследствии в ряде работ [3,4,7-9] были предложены и другие, менее требовательные определения стойкости обфускации, однако, и для них была показана невозмож-

ность построения эффективного транслятора, гарантирующего стойкую обфускацию произвольных программ. Вместе с тем, в работах [4,5,8,10] было показано, что для отдельных классов функций (т.н. «точечных функций») стойкая обфускация вычисляющих их программ возможна при тех или иных стандартных криптографических предположениях. Наиболее значительное продвижение было достигнуто в работе [11]; ее авторы доказали осуществимость стойкой обфускации программ перешифрования сообщений. В целом, однако, результаты исследований в этом направлении не дают больших оснований для оптимизма: до сих пор не удалось обнаружить достаточно сложной криптографической функции, программы вычисления которой допускают доказуемо стойкую обфускацию.

Применение обфускации не ограничивается криптографией; другое направление исследований проблемы обфускации программ связано с ее применением в области компьютерной безопасности. Обфускирующие преобразования могут быть использованы для защиты интеллектуальной собственности на программное обеспечение [1,12], для информационной защиты мобильных агентов [13] и микроэлектронных схем на этапе проектирования [14], для обеспечения приватности информационного поиска [15]. К сожалению, оказалось, что техника обфускации программ может быть также использована и в злонамеренных целях для разработки трудно обнаруживаемых «вирусов» [16,17]. Перспективы успешной разработки и применения обфускирующих преобразований в этом направлении представляются более определенными: для многих коммерческих приложений проведение обфускации программ становится целесообразным, если издержки, которые вынужден понести противник для преодоления последствий обфускации, существенно превосходят издержки, вызванные применением обфускации. Так, например, значительное понижение производительности вычислительной системы, обусловленное продолжительной работой антивирусного сканера в попытке обнаружить скрытный метаморфный «вирус», может быть приравнено к ущербу, причиненному самим «вирусом».

Методы обфускации программ, применяемые для обеспечения информационной безопасности программного обеспечения, можно условно классифицировать по трем основным характеристикам в зависимости от того,

- допускается ли использование высоконадежного вычислительного устройства, недоступного для противника, или доступ к обфускированной программе считается неограниченным;
- в какой форме программа предполагается доступной для противника (в виде исходного текста на языке высокого уровня или в виде исполняемого кода);
- предполагается ли, что код программы должен оставаться неизменным на протяжении ее выполнения.

С математической точки зрения, наиболее хорошо обоснованными являются методы обфускации программ, рассчитанные на использование высоконадежных вычислительных устройств. В этом случае исходная программа P разбивается на две компоненты (C_1,C_2). На одну из этих компонент (например, C_1) возлагается основная вычислительная нагрузка, тогда как другая компонента (C_2) играет вспомогательную роль. Основная компонента C_1 устанавливается на высокопроизводительном вычислительном устройстве, не имеющем надежных средств защиты; считается, что противнику открыт неограниченный доступ к компоненте C_1. Вспомогательная компонента C_2 устанавливается на высоконадежном (но, быть может, значительно менее производительном) вычислительном устройстве, которое обеспечивает достаточный уровень защиты от любого противника. Таким устройством может быть электронная карточка (smartcard), защищенный модуль (tamper-proof device), и т.п. Предполагается, что противник не имеет доступа к

компоненте C_2. Разбиение (C_1,C_2) программы P должно быть устроено так, чтобы противник, имеющий доступ лишь к основной компоненте C_1 и не располагающий сведениями о компоненте C_2, не смог восстановить исходную программу P.

Впервые этот подход к информационной защите программ был исследован в работе [18], где было показано, что с небольшими вычислительными издержками доступ к открытой памяти (компоненте C_1) можно организовать так, что противнику по ходу вычисления не будет открыта никакая информация о выполняемой программе (компоненте C_2). В работах [19,20] описаны способы построения сечений программ; сечение представляет собой небольшой фрагмент программы, который, будучи скрытым от противника, не позволяет ему получить никакой информации об устройстве всей программы. В работах [21,22] показано, что для некоторых вычислительных схем можно выбрать подходящий способ шифрования данных, который позволяет проводить вычисления над зашифрованными данными. Шифрование и дешифрование данных проводится на защищенном устройстве (компоненте C_2), а сама схема вычислений над зашифрованными данными (компонента C_1) считается открытой. Показано, что в этом случае вероятность правильного восстановления программы P по открытой компоненте C_1 является пренебрежимо малой величиной.

Стойкость обфускации может зависеть также и от той формы, в которой программа оказывается доступной противнику. Если противнику открыт доступ лишь к исполняемому двоичному коду программы, то деобфускация программы начинается с преобразования ее двоичного кода в описание программы на языке высокого уровня. Эта процедура называется *обратной инженерией*; она состоит из двух этапов – *дизассемблирования* и *декомпиляции*. Некоторые методы обфускации программ используют особенности устройства двоичных программных кодов с целью затруднить дизассемблирование и декомпиляцию исполняемых файлов. Главная особенность двоичных кодов состоит в том, что машинное слово в зависимости от контекста может восприниматься как команда или как фрагмент данных. Обфускация исполняемого кода перемежает машинные команды с данными или несущественными словами («мусором»), и это затрудняет применение статических методов дизассемблирования. Метод обфускации машинных кодов, предложенный в работе [23], приводит к тому, что свыше 60% машинных команд обфускированной программы восстанавливаются дизассемблерами неправильно. В последующих работах [24-28] развернулось настоящее исследовательское соревнование между проектировщиками дизассемблеров и разработчиками обфускаторов двоичных машинных кодов. Систематическое описание методов обфускации двоичных машинных кодов представлено в работе [29]. Эти работы свидетельствуют о том, что обфускаторы способны эффективно противодействовать статическим методам дизассемблирования. Однако использование динамического дизассемблирования, при котором ассемблерный код программы восстанавливается в процессе ее выполнения, существенно снижает стойкость обфускации.

Для противодействия динамическим методам дизассемблирования программ применяются специальные методы обфускации программ, приводящие к тому, что код программы постоянно видоизменяется в процессе выполнения. Впервые этот подход к защите программного обеспечения был представлен в работе [30]. Суть метода состоит в том, что фрагменты программы постоянно расшифровываются и перешифровываются в процессе ее выполнения, не позволяя тем самым противнику увидеть весь код программы целиком. В работе [31] этот подход был усилен тем, что код программы в процессе выполнения не только постоянно шифруется, но и изменяет свою собственную структуру (мутирует), используя генераторы случайных чисел. Этот метод обфускации активно используется создателями

трудно диагностируемых полиморфных вирусов (см. [16,17]) для того, чтобы скрыть характерные признаки вирусов («подписи») от антивирусных сканеров.

Тем не менее, большая часть методов обфускации рассчитана на применение к описаниям программ на языках высокого уровня (Java, C, Basic и др.). Перечень простейших методов обфускации высокоуровневых программ представлен в основополагающей работе [1]. Эти методы разделяются на две группы.

1. *Обфускация потока управления программы.* Цель обфускации – скрыть наиболее существенные индивидуальные особенности программ. Для этого управляющие структуры программ (потоки управления) приводятся к единообразной форме. Тело программы (главная функция) представляет собой переключатель (*switch*), моделирующий работу конечного автомата. На вход автомата поступают значения логических условий, в зависимости от которых управление передается той или иной функции, выполняющей линейную последовательность простейших операторов. Таким образом, у всех программ, имеющих одно и то же число линейных участков, графы потоков управления выглядят одинаково. Это подход был впервые применен в работе [32] и развит в работах [33-37].

2. *Обфускация потоков данных программы.* Здесь целью обфускации является сокрытие зависимости по данным между переменными и функциями программы. Это достигается, например, за счет интенсивного использования переменных-указателей, функций-указателей и адресной арифметики. Поскольку даже простейшие задачи выявления синонимии переменных и оценки диапазона их значений имеют большую вычислительную сложность (см. [38-39]), методы статического анализа не способны выявить зависимости по данным между разными фрагментами обфускированных программ. Этот подход был впервые описан в работе [40] и был систематически исследован в работах [41,42].

Большую роль в успешном применении указанных методов обфускации играют т.н. «непроницаемые» (opaque) предикаты, значения которых известны в процессе обфускации программы, но трудно вычисляются на этапе ее анализа. Обычно «непроницаемые» предикаты строятся на основе алгебраических или теоретико-числовых тождеств (см. [43]). В работах [32,33] «непроницаемые» предикаты строились на основе труднорешаемых комбинаторных задач. В серии работ [44-46] было показано, каким образом можно использовать теорию абстрактных интерпретаций программ для автоматического построения «непроницаемых» предикатов. Дополнительные возможности, усиливающие скрытность «непроницаемых» предикатов, открываются в случае обфускации многонитевых или распределенных программ [47].

К сожалению, методы обфускации программ активно используются разработчиками компьютерных «вирусов» для того, чтобы затруднить работу антивирусных сканеров. Как известно, наиболее эффективным методом обнаружения «вирусов» является поиск подписи «вируса». Однако более «совершенные» полиморфные и метаморфные вирусы наделены способностью к маскировке. Полиморфизм заключается в формировании кода вируса «на лету» — уже во время его исполнения; при этом сама процедура, формирующая код, также не является постоянной и видоизменяется при каждом новом заражении. Метаморфные вирусы пытаются избежать обнаружения за счет применения в процессе репликации более изощренных обфускирующих преобразований. Поэтому наряду с разработкой новых методов обфускации исследователи этой проблемы уделяют большое внимание разработке методов деобфускации программ, которые можно применять для выявления чужеродных фрагментов кода [48-51]. В основу этих методов положены алгоритмы проверки функциональной эквивалентности программ, поскольку именно

функциональные характеристики вируса являются его подлинной «подписью», неизменно сохраняющейся при репликации.

В настоящее время создано несколько действующих развитых систем (де)обфускации программ [35,36,52,53], в которых используются различные комбинации описанных выше методов обфускации. Индивидуальные особенности каждой из этих систем определяются стратегией применения обфускирующих преобразований.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Collberg C., Thomborson C., Low D.* A taxonomy of obfuscating transformations // Tech. Report, N 148, Univ. of Auckland, 1997.

2. *Barak B., Goldreich O., Impagliazzo R., Rudich S., Sahai A., Vedhan S., Yang K.* On the (Im)possibility of obfuscating programs // Lecture Notes in Computer Science, v. 2139, 2001, p. 1-18.

3. *Goldwasser S., Tauman Kalai Y.* On the impossibility of obfuscation with auxiliary input // Proc. of the 46 th IEEE Symp. on Foundations of Computer Science, 2005, 553-562.

4. *Hofheinz D., Malone-Lee J., Stam M.* Obfuscation for cryptographic purposes // Lecture Notes in Computer Science, v. 4392, 2007, p. 214-232.

5. *Lynn B., Prabhakaran M., Sahai A.* Positive results and techniques for obfuscation // Lecture Notes in Computer Science, v. 3027, 2004, p. 20-39.

6. *Adida B., Wikström D.* Obfuscated ciphertext mixing // IACR. Eprint Archive, N 394, 2005.

7. *Varnovsky N.P.* A note on the concept of obfuscation // Труды Института системного программирования, т. 6, 2004.

8. *Wee H.* On obfuscating point functions // Proc. of 37th ACM Symp. on Theory of Computing, 2005, p. 523-532.

9. *Goldwasser, S., Rothblum G.* On best-possible obfuscation // Lecture Notes in Computer Science, v. 4392, 2007, p. 253-272.

10. *Varnovsky N.P., Zakharov V.A.* On the possibility of provably secure obfuscating programs // Lecture Notes in Computer Science, v. 2890, 2003, p. 91-102.

11. *Hohenberger S., Rothblum G. N., Shelat A., Vaikuntanathan V.* Securely obfuscating reencryption // Lecture Notes in Computer Science, v. 4392, 2007, p. 233-252.

12. *Collberg C, Thomborson C.* Watermarking, tamper-proofing, and obfuscation - tools for software protection // IEEE Transactions on Software Engineering, v. 28, N 6, 2002.

13. *D'Anna L., Matt B., Reisse A., Van Vleck T. , Schwab S., LeBlanc P.* Self-protecting mobile agents obfuscation report // Report #03-015, Network Associates Laboratories, 2003.

14. *Иванников В.П., Варновский Н.П., Захаров В.А., Кузюрин Н.Н., Шокуров А.В., Кононов А.Н., Калинин А.В.* Методы информационной защиты проектных решений при изготовлении микроэлектронных схем // Известия ТРТУ, 2005. N 4. – С. 112-119.

15. *Ostrovsky R., Skeith III W.E.* Private searching on streaming data // Lecture Notes in Computer Science, v. 3621, 2005, p. 223-240.

16. *Chess D., White S.* An undetectable computer virus. // 2000 Virus Bulletin Conference, 2000.

17. *Szor P., Ferrie P.* Hunting for metamorphic // 2001 Virus Bulletin Conference, 2001, 123-144.

18. *Goldreich O., Ostrovsky R.*. Software protection and simulation on oblivious RAMs.// Journal of the ACM, v. 43, N 3, 1996, p.431-473.

19. *Mana A., Pimentel A.* An efficient software protection scheme // Proc. of the 16th Int. Conf. on Information Security, 2001, p. 385-401.

20. *Zhang X. , Gupta* R. Hiding program slices for software security // Proc. of the First Annual IEEE/ACM Int. Symp. on Code Generation and Optimization, 2003, p. 325-336.

21. *Sander T., Tchudin C.* On software protection via function hiding. // Lecture Notes in Computer Science, v. 1525, 1998.

22. *Shokurov A.V.* An approach to quantitative analysis of resistance of equivalent transformations of algebraic schemes // Труды Института системного программирования, 2004, т. 6.

23. *Linn C., Debray S.* Obfuscation of executable code to improve resistance to static disassembly // Proc. of the 10th ACM Conf. on Computer and Communications Security, Oct. 2003.

24. *Stroulia E., Systa T.* Dynamic analysis for reverse engineering and program understanding // ACM SIGAPP Applied Computing Review, v. 10, 2002, N 1, p. 8-17.

25. *Heffner K., Collberg C.* The obfuscation executives // Tech. Rep. TR 04-03, Dep. of computer science, Univ. of Arizona, USA, 2003.

26. *Kruegel C., Robertson W., Valeur F., Vigna G.* Static disassembly of obfuscated binaries // Proc. of USENIX Security, 2004, p. 255-270.

27. *Udupa S. K., Madou M., Debray S.* Deobfuscation: reverse engineering obfuscated code // Proc. of the 12th Working Conf. on Reverse Engineering, 2005, p. 45-54.

28. *Madou M., Anckaert B., De Bus B., De Bosschere K., Cappaert J., Preneel B.* On the effectiveness of source code transformations for binary obfuscation // Proc. of the Int. Conf. on Software Engineering Research and Practice (SERP06), June. 2006.

29. *Wroblewski G.* General method of program code obfuscation // Proc. of the Int. Conf. on Software Engineering Research and Practice (SERP), 2002, p.153-159.

30. *Aushmith D.* Tamper resistant software: an implementation // Lecture Notes in Computer Science, v. 1174, 1996, p.317-333.

31. *Madou M., Anckaert B. , Moseley P., Debray S., De Sutter B., De Bosschere K.* Software protection through dynamic code mutation // Information Security Applications, 2005, p. 371-385.

32. *Wang C., Hill J., Knight J,. Davidson J.* Software tamper resistance: obstructing static analysis of programs // Tech. Rep., N 12, Dep. of Comp. Sci., Univ. of Virginia, 2000.

33. *Chow S., Gu Y., Johnson H., Zakharov V.A.* An approach to the obfuscation of control-flow of sequential computer programs // Lecture Notes in Computer Science, v. 2200, 2001, p.144-155.

34. *Чернов А.В.* Анализ запутывающих преобразований программ // Труды Института системного программирования, Т. 3, 2002, – С. 137-163.

35. *Чернов А. В.* Интегрированная инструментальная среда Poirot для изучения методов маскировки программ // Препринт Института системного программирования РАН. - М.: ИСП РАН, 2003.

36. *Ertaul L. Venkatesh S.* Jhide – a toolkit for code obfuscation // Proc. of the 8th IASTED Int. Conf. on Software Engineering and Applications, 2004.

37. *Madou M., van Put L., De Bosschere K.* Understanding obfuscated code // Proc. of the 14th IEEE International Conf. on Program Comprehension, 2006, p. 268-274.

38. *Hind M., Pioli A.* Which pointer analysis should I use // Proc. of the 2000 ACM SIGSOFT Int. Symp. on Software Testing and Analysis, 2000, p 113-123.

39. *Horwitz S.* Precise flow-insensitive may-alias analysis is NP-hard // University of Wisconsin-Madison, August 1996.

40. *Collberg C., Thomborson C., Low D. C.* Breaking abstractions and unstructuring data structures // Proc. of the 1998 Int. Conf. on Computer Languages, 1998, p. 28-38.

41. *Ivanov K. S., Zakharov V. A.* Program obfuscation as obstruction of program static analysis // Труды Института системного программировани, т. 6, 2004.

42. *Sakabe Y., Soshi M., Miyaji A.* Java obfuscation with a theoretical basis for building secure mobile agents // Proc. of the 7th IFIP Conf. on Communication and Multimedia Security, 2003.

43. *Collberg C., Thomborson C., Low D. C.* Manufacturing cheap, resilient and stealthy opaque constructs // Symp. on Principles of Programming Languages, 1998, p. 184-196.

44. *Cousot P., Cousot R.* An abstract interpretation based framework for software watermarking // Symp. on Principles of Programming Languages, 2003, p. 311-324.

45. *Dalla Preda M., Giacobazzi R.* Semantic-based code obfuscation by abstract interpretation // Lecture Notes in Computer Science, v. 3580, 2005, p. 1325-1336.

46. *Dalla Preda M., Giacobazzi R., Madou M., de Bosschere B.* Opaque predicate detection by means of abstract interpretations // Lecture Notes in Computer Science, v. 4019, 2006, p. 81-95.

47. *Majumdar A., Thomborson C.* Manufacturing opaque predicates in distributed systems for code obfuscation // Proc. of the 29th Australasian Computer Science Conf., 2006, p. 187-196.

48. *Christodorescu M., Jha S., Seshia S.A., Song D., Bryant R.E.* Semantic-aware malware detection // Proc. of the 2005 IEEE Symp.on Security and Privacy (Oakland 2005), 2005.

49. *Dalla Preda M., Christodorescu M., Jha S.,Debray S.* Semantic-based approach to malware detection // Proc. of the 34th Annual ACM SIGPLAN-SIGACT Symp.on Principles of Programming Languages, 2007, p. 377-388.

50. *Варновский Н.П., Захаров В.А., Кузюрин Н.Н., Шокуров А.В., Подловченко Р.И., Щербина В.С.* О применении методов деобфускации программ для обнаружения сложных компьютерных вирусов // Известия ТРТУ, Таганрог, Изд-во ТРГУ, 2006, т. 7, с. 66-72.

51. *Madou M., Anckaert B., De Sutter B., De Bosschere K.* Hybrid static-dynamic attacks against software protection mechanisms // Proc. of the 5th ACM Workshop on Digital Rights Management, 2005, p. 75-82.

52. *Collberg, C.S. Myles G., Huntwork A.* Sandmark - a tool for software protection research // IEEE Security & Privacy, v. 1, N 4, 2003, p. 40-49.

53. *Madou, M.; Van Put, L.; De Bosschere, K.* Loco: an interactive code (de)obfuscation tool // Proc. of ACM SIGPLAN 2006 Workshop on Partial Evaluation and Program Manipulation, 2006.

В.В. Анищенко, Ю.В. Земцов
Беларусь, г. Минск, ОИПИ НАН Беларуси

МЕТОДИКА ИСПЫТАНИЙ СИСТЕМ ОБНАРУЖЕНИЯ АТАК

В настоящее время многие исследовательские и коммерческие организации тестируют и оценивают системы обнаружения атак (СОА) для определения эффективности каждого продукта, сравнения их друг с другом и обоснования выбора лучшего из них. Многие системные администраторы и администраторы безопасности также производят свои собственные оценки таких продуктов, чтобы принять решение о целесообразности их использования в своих средах. Однако подобные оценки СОА обычно включают в себя неформальное сравнение заявляемых разработчиками функциональных возможностей и, в лучшем случае, проверку относительных признаков сигнатур программных продуктов, т. е. производится определенное количество запланированных атак на тестируемую сеть, а испытатель подсчитывает количество сигналов тревоги действительно соответствующих производимым атакам, а также количество ложных сигналов тревоги. Результаты такого теста могут дать общее представление о сигнатурах, но они не дают точной оценки и не обеспечивают доверительные данные для того, чтобы делать заключение о преимуществе в эффективности той или иной системы [1]. В условиях обилия предложений СОА и отсутствия объективных критериев их оценки крайне тяжело сделать обоснованный выбор. Таким образом, особенно актуальна задача разработки методики испытаний СОА, основанной на формальных моделях и позволяющей получить для функциональных возможностей СОА количественные оценки.

Для оценки функциональных возможностей СОА существует ряд методик. Например, методика, предназначенная для оценки функциональных возможностей СОА, встраиваемых в канал связи и перехватывающих весь сетевой трафик на входе в защищаемую сеть, т.е. анализ данных с целью обнаружения атак выполняется в СОА на основе информации, собираемой на уровне сети. Однако существуют СОА, в которых применяются не только датчики уровня сети, но и датчики уровня операционной системы, предназначенные для сбора данных с целью обнаружения атак, не выявляемых на сетевом уровне [2]. Сбор данных на уровне операционной системы подразумевает получение информации о событиях критичных для обеспечения безопасности в защищаемой сети на основе последовательностей системных вызовов, сведений об использовании ресурсов системы, записей из журналов аудита операционной системы и журналов приложений и т.д. Кроме то-