операции криптографического преобразования потока медиаданных, что делает возможным повышение качества сеанса связи без задействования дополнительных вычислительных мощностей.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Кравченко П.П.* Основы теории оптимизированных дельта-преобразований второго порядка. Цифровое управление, сжатие и параллельная обработка информации: Монография. – Таганрог: Изд-во ТРТУ, 1997.

5. *Кравченко П.П., Хусаинов Н.Ш., Погорелов К.В., Хаджинов А.А., Шкурко А.Н.* Программная система аудиовидеоконференцсвязи для локальных и корпоративных IP-сетей. Программные продукты и системы (Software & Systems). 2004. №1. – С.27–30.

В.С. Несов
Россия, г. Москва, ИСП РАН

ИСПОЛЬЗОВАНИЕ ПОБОЧНЫХ ЭФФЕКТОВ ФУНКЦИЙ ДЛЯ УСКОРЕНИЯ АВТОМАТИЧЕСКОГО ПОИСКА УЯЗВИМОСТЕЙ В ПРОГРАММАХ

Введение

В Институте системного программирования РАН разработана среда обнаружения уязвимостей [1][2] в исходном коде программ на языке C, позволяющая обнаруживать уязвимости и дефекты следующих типов:

- переполнение буфера (buffer overflow);
- неконтролируемая форматная строка (format string);
- разыменование нулевого указателя (null pointer) ;
- утечка памяти (memory leak).

Перед средой ставилась задача не пропускать уязвимости. Для этого предупреждения о возможных дефектах выводятся во всех местах анализируемой программы, в которых при помощи анализа не удалось доказать отсутствие дефектов.

Среда использует межпроцедурный итеративный анализ потока данных. Высокая точность выполняемого анализа требуется для снижения количества ложных предупреждений, число которых в данной постановке задачи непосредственно от нее зависит.

При межпроцедурном анализе информация о значениях переменных распространяется через точки вызовов функций. Для того, чтобы информация о значении некоторой переменной распространилась от одной функции к другой, она проходит через все промежуточные функции на графе вызовов.

Так как каждая функция программы может вызываться из нескольких мест, среднее количество информации, проходящей через каждый вызов в графе вызовов, увеличивается с увеличением размера анализируемой программы.

В статье описывается метод, позволяющий сократить количество распространяемой по графу вызовов информации о значениях переменных, тем самым сокращая время анализа и требуемое количество памяти. Метод основан на нахождении побочных эффектов функций одновременно с выполнением основного анализа, и ограничении распространяемых значений переменных с использованием этой информации.

Отавшаяся часть статьи организована следующим образом. В разделе 2 приводится обзор процесса анализа. В разделе 3 подробно описывается анализ инструкции вызова функции, передающей данные между процедурами. В разделе 4 описано использование побочных эффектов функции для ограничения распро-

странения значений переменных через точки вызова функций. Раздел 5 содержит результаты проведенных экспериментов.

Краткое описание процесса анализа

В этом разделе кратко описывается процесс автоматического анализа программ в среде обнаружения уязвимостей. Более подробное описание, включающее особенности реализации, можно найти в [2].

Анализируемая программа собирается с использованием модифицированной версии компилятора gcc, дополнительно выдающей файлы, содержащие внутреннее представление для каждого скомпилированного модуля. Набор модулей, составляющий анализируемую программу, объединяется в проект.

Среда обнаружения уязвимостей считывает файлы проекта. Для каждой функции строится граф потока управления (в вершинах которого расположены отдельные инструкции внутреннего представления), для проекта в целом строится статический граф вызовов. Впоследствии граф вызовов дополняется ребрами, соответствующими вызовам по указателям.

Каждому ребру графов потока управления сопоставляются (изначально пустые) контексты. Также контексты сопоставляются точкам входа и выхода из функций. Каждый контекст хранит информацию о значениях переменных, определенную для данной точки программы.

Каждой точке определения переменной (включая точки выделения памяти в куче) сопоставляется абстрактная ячейка памяти (АЯП). Контекст сопоставляет каждой АЯП набор атрибутов.

Используются следующие атрибуты:

- интервал возможных значений (для целочисленных переменных);
- множество АЯП, указываемых данным указателем, со смещением указателя относительно каждой указываемой АЯП; для смещения указывается интервал его возможных значений;
- флаг, указывающий факт удаления переменной из кучи (для областей памяти в куче);
- флаг, указывающий наличие зависимости значения переменной от пользовательского ввода.

При межпроцедурном анализе граф потока управления обходится топологически, попеременно в прямом и обратном направлениях. В каждой точке обхода для текущей функции выполняется внутрипроцедурный анализ.

При внутрипроцедурном анализе граф потока управления топологически обходится в прямом направлении. Для каждой инструкции применяется функция преобразования потока данных, находящая значения контекстов на исходящих из инструкции ребрах графа потока управления по значениям контекстов на входящих в данную инструкцию ребрах.

Анализ вызова функции

Межпроцедурная передача информации происходит в функциях преобразования потока данных, сопоставленных инструкциям вызова.

При анализе инструкции вызова часть входного контекста вызова передается вызываемой функции. Собранные таким образом из разных мест вызова контексты впоследствии объединяются, составляя входной контекст вызываемой функции.

Часть выходного контекста вызываемой функции используется для замены атрибутов АЯП входного контекста вызова для определения выходного контекста вызова.

Каждая функция принимает часть АЯП извне (из разных точек вызова), часть АЯП являются локальными для функции, и оставшиеся АЯП обозначают созданные в результате выполнения функции области памяти в куче. Передаваемые вы-

зываемой функции АЯП (часть входного контекста инструкции вызова) ограничиваются доступными из вызываемой функции. Для формирования выходного контекста используются АЯП, созданные внутри функции, и АЯП, переданные из *данного* вызова. Таким образом, ограничивается распространение АЯП по нереализуемым путям, когда переданное в вызываемую функцию в одной точке вызова значение в результате неточности анализа передается другой точке вызова той же функции.

Будем обозначать множество АЯП, атрибуты которых определены в контексте C, как Def(C). Будем обозначать Pt(x,C) множество АЯП, указываемых АЯП x в контексте C (points-to атрибуты). Если x не является указателям, или $x \notin C$, положим Pt(x,C)=∅. Для АЯП, обозначающих переменные структурных типов, поддерживаются отдельные АЯП, обозначающие их поля. Будем обозначать Fi(x) множество полей АЯП x.

При вызове функции происходит копирование фактических параметров в формальные, а также внутреннего возвращаемого функцией значения в переменную, принимающую его при вызове. Для данного вызова обозначим ActToForm функцию, сопоставляющую фактическим параметрам формальные (определенную на множестве фактических), ActSet множество фактических параметров, RetToRes функцию, отображающую внутреннее возвращаемое функцией значение Ret на принимающую его переменную вызывающей функции.

Обозначим Close(C,S) операцию замыкания множества АЯП S в контексте C относительно преобразования $F_C(S) = S \cup NextPtGen(C,S) \cup Fi(S)$,

где $NextPtGen(C,S) = \bigcup_{s \in S} Pt(s,C)$.

Close(C,S) есть минимальное множество такое, что $S \subseteq Close(C,S)$ и $Close(C,S)=F_C(Close(C,S))$.

Обозначим Retain (C,S) операцию, возвращающую контекст C, в котором удалены все АЯП, не включенные в S (в том числе и из points-to атрибутов). Наконец обозначим Replace(C,F) преобразование, возвращающее контекст C с ключами (АЯП), замененными в соответствии с частично определенной функцией F: АЯП→АЯП (на ее области определения).

Пусть входной контекст вызова обозначен In, предикат принадлежности к множеству всех глобальных переменных isGlob. Псевдокод нахождения входного контекста вызова функции с удаленными АЯП, не достижимыми из вызываемой функции, приведен на рис. 1. Полученный трансляцией фактических переменных в формальные контекст PassedToCallee передается вызываемой функции.

```
InGlob = {s∈Def(In) | isGlob(s)}
; множество непосредственно доступных
; вызываемой функции АЯП
InAnchorSet = InGlob∪ActSet
; множество доступных из InAnchorSet АЯП
InAccSet = Close(In, InAnchorSet)
; суженный контекст, содержащий
; только АЯП из InAnchorSet
FilteredInput = Retain(In, InAccSet)
; результат трансляции фактических параметров
; в формальные
PassedToCallee = Replace(FilteredInput, ActToForm)
```

Рис. 1. Псевдокод алгоритма передачи информации к вызываемой функции

Пусть выходной контекст функции обозначен FExit. Псевдокод нахождения копируемой в выходной контекст вызова части FExit приведен на рис. 2.

Использование побочных эффектов

Каждой функции сопоставляется множество АЯП побочных эффектов. Побочные эффекты каждой функции складываются из побочных эффектов каждой инструкции, входящей в функцию. При анализе каждой инструкции определяется множество АЯП, атрибуты которых читаются либо изменяются [2].

```
; множество доступных из выходного контекста функции АЯП
ExitAccSet = Close(FExit, Def(FExit))
; множество глобальных АЯП в ExitAccSet
; (на которые вызываемая функция могла сослаться независимо
; от того, передавались ли они ей извне)
ExitGlobSet = {s∈ExitAccSet | isGlob(s)}
; множество АЯП в ExitAccSet, выделенных в куче
; (которые могли быть созданы в результате вызова функции)
ExitHeapSet = {s∈ExitAccSet | isHeap(s)}
IndirectlyAccActSet = ActSet ∩ NextPtGen(In, InAccSet)
; множество АЯП, доступных из вызываемой функции
; через переданные данным вызовом
AccFromCallee =  (InAccSet \ ActSet) ∪ IndirectlyAccActSet
; множество АЯП, доступных из AccFromCallee
; в выходном контексте вызываемой функции
ExitPassSet = Close(FExit, AccFromCallee)
; ограничение на возвращаемое множество АЯП
; для данного контекста вызова
RetrSet = ExitGlobSet ∪ ExitHeapSet ∪ ExitPassSet ∪ {Ret}
PassedFromCalleeInt = Restrict(FExit, RetrSet)
PassedFromCallee = Replace(PassedToCalleeInt, RetToRes)
```

Рис. 2. Псевдокод алгоритма передачи информации к вызывающей функции

Пусть множество побочных эффектов вызываемой функции обозначено ChangeSet. Измененный псевдокод вычисления множества InAccSet, ограничивающего передаваемую к вызываемой функции часть входного контекста вызова, приведен на рис. 3.

```
InGlob = {s∈ChangeSet | isGlob(s)}
; множество непосредственно доступных
; вызываемой функции АЯП, включенных
; в множество побочных эффектов
InAnchorSet = InGlob∪ActSet
; множество доступных из InAnchorSet АЯП,
; включенных в множество побочных эффектов
InAccSet = Close(In, InAnchorSet) ∩ ChangeSet
```

Рис. 3. Псевдокод измененного алгоритма передачи информации к вызываемой функции

Так как не все из входящих в ChangeSet АЯП обязательно определены в контексте In, для распространения множества побочных эффектов по графу вызовов InAccSet добавляется в множество побочных эффектов вызывающей функции полностью. АЯП распространяется от вызывающей функции к вызываемой только по требованию через множество побочных эффектов. За счет топологического по-

рядка обхода графа вызовов такое требование достигает точки назначения за одну итерацию и межпроцедурный анализ замедляется незначительно.

Экспериментальные результаты

В табл.1 показаны результаты анализа 14 пакетов свободно распространяемого ПО.

Таблица 1

Результаты работы среды

	LOC	NOF	без учета побочных эффектов		с учетом побочных эффектов	
			time (сек)	avg	time (сек)	avg
polymorph-0.4.0	400	28	4	33	3	2
surfboard-1.1.8	675	27	6	16	4	4
lhttpd-0.1	814	19	6	21	5	5
ssmtp-2.60	1818	41	21	57	12	7
troll-ftpd-1.26	2279	95	106	72	59	6
pound-1.0	1796	49	34	149	27	2
muh-2.05d	2863	184	123	139	46	10
bftpd-1.0.24	3023	181	141	145	46	38
pgp4pine-1.76	3340	106	231	281	37	8
cfingerd-1.4.3	3699	233	313	373	50	4
pcre-3.9	5625	80	343	52	211	10
gzip-1.2.4	5832	111	246	196	75	29
sharutils-4.2.1	6368	101	88	103	26	5
thttpd-2.23beta1	7955	191	602	305	120	33

Для каждого пакета приведены его название и номер версии, количество строк исходного кода пакета без учета пустых строк (колонка LOC), количество функций в пакете (колонка NOF).

Для каждого варианта реализации среды указаны время анализа (колонки time) и средний размер входных контекстов функций (количество определенных АЯП), установившийся в конце анализа (колонки avg).

Как видно из результатов, использование побочных эффектов многократно снижает количество элементов контекстов функций (в 10-50 раз, в зависимости от проекта). Такое снижение даже с учетом возникновения издержек на вычисление побочных эффектов приводит к снижению времени анализа в 2-6 раз. Для больших размеров анализируемых программ выиграш во времени от применения оптимизации увеличивается.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Маликов О.Р.* Автоматическое обнаружение уязвимостей в исходном коде программ, Известия ТРТУ №4, Материалы VII Международной научно-практической конференции «Информационная безопасность». – С. 48–53, июль 2005.

2. *Несов В.С., Маликов О.Р.* Автоматический поиск уязвимостей в больших программах, Известия ТРТУ №7, Материалы VIII Международной научно-практической конференции «Информационная безопасность». – С. 38–44, июль 2006.

3. *Bush W.R., Pincus J.D., and Sielaff D.J.* A static analyzer for finding dynamic programming errors. In Proceedings of Software Practice and Experience, P. 775–802, 2000.

4. *Donglin Liang and Mary Jean Harrold.* Efficient Computation of Parameterized Pointer Information for Interprocedural Analyses. Lecture Notes in Computer Science, v. 2126, p. 279, 2002.

5. *Ramkrishna Chatterjee, Barbara G. Ryder.* Relevant Context Inference. Symposium on Principles of Programming Languages, p. 133-146, 1999.

6. *Patrick Cousot, Radhia Cousot.* Abstract interpretation frameworks. *Journal of Logic and Computation,* 2(4):511-547, 1992.

7. *Dor N., Rodeh M., and Sagiv M.* CSSV: Towards a Realistic Tool for Statically Detecting All Buffer Overflows in C. In Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation, pages 155–167, San Diego, California, 2003.

8. *Wagner D., Foster J. S., Brewer E. A., and Aiken A.* A First Step Towards Automated Detection of Buffer Overrun Vulnerabilities. In Proceedings of 7th Network and Distributed System Security Symposium, Feb. 2000.

9. *Emami M., Ghiya R., and Hendren L.* Context-Sensitive Interprocedural Points-to Analysis in the Presence of Function Pointers. In Proceedings of the SIGPLAN '94 Conference on Program Language Design and Implementation, Orlando, US, 1994.

А.С. Моляков
Россия, г. Москва, ОАО «ИнфоТеКС»,

ИССЛЕДОВАНИЕ СКРЫТЫХ МЕХАНИЗМОВ УПРАВЛЕНИЯ ЗАДАЧАМИ ЯДРА WINDOWS NT 5.1

В предисловии хочу выразить благодарность научному руководителю д.ф.-м.н., профессору Грушо А.А.В данной статье я наглядно покажу коллизии в сопряжении проектов верхнего и нижнего уровней на платформе Windows NT.

Актуальность. Острота проблемы обеспечения безопасности субъектов информационных отношений, защиты их законных интересов при использовании информационных и управляющих систем, хранящейся и обрабатываемой в них информации все более возрастает. Проблема защиты вычислительных систем становится еще более серьезной и в связи с развитием и распространением вычислительных сетей, территориально распределенных систем и систем с удаленным доступом к совместно используемым ресурсам. Увеличение числа компьютеров в организации приводит к потере контроля над ними.

Объект исследования: программное обеспечение штатных средств защиты на базе платформы Windows NT 5.1(объекты микроядра , модули диспетчера оборудования , подсистема управления вводом-выводом).

Практическая значимость работы

Результаты выполненных исследований могут быть использованы при построении ответственных программных систем , для прогнозирования и оценки надежности ПО и при расчете надежности систем в целом. Метод систематического поиска уязвимостей позволяет обнаруживать скрытые процессы как в третьем кольце защиты , так и в нулевом , используя неявные механизмы в сопряжении проектов верхнего и нижнего уровней.

Научная новизна

- разработана методология анализа надежности ПО как процесс возникновения и устранения ошибок в ПО на всех этапах жизненного цикла ПО, предложена методика тестирования ,реализован новый метод – метод систематического поиска уязвимостей ОС Windows;

- представлены таблицы состояний ЦП(логических переходов), механизмов сопряжения проектов верхнего и нижнего уровней и процессов выявления недекларированных уязвимостей ядра Windows NT 5.1.

Архитектура памяти NT — это система виртуальной памяти, использующая 32-разрядные адреса в линейном адресном пространстве.Каждой прикладной про-