УДК 004.43

К. Е. Израилов

Санкт-Петербургский государственный университет телекоммуникаций им. проф. М. А. Бонч-Бруевича

РАСШИРЕНИЕ ЯЗЫКА «*C*» ДЛЯ ОПИСАНИЯ АЛГОРИТМОВ КОДА ТЕЛЕКОММУНИКАЦИОННЫХ УСТРОЙСТВ

современное общество, информационное пространство, телекоммуникационные устройства, язык С, расширение языка, эффективность кода, восстановление кода, язык описание алгоритмов, безопасность, уязвимость

Введение

Современное общество, присущее большинству высокоразвитых стран, без преувеличения можно считать потребительским. Оставляя обсуждение аргументов в защиту и против такого общества, рассмотрим следующие его черты, имеющие отношение к информационным технологиям [1] и их безопасности.

Во-первых, массовое потребление продуктов в сфере вычислительной техники (с целью самовыделения потребителя без объективной необходимости) привело к развитию компьютерных технологий, и, как следствие, к стремительному росту производительности платформ последних. Это, в свою очередь, сместило баланс от написания эффективного исходного кода программного обеспечения (далее – ПО) в сторону удобство и скорости процесса его разработки. Например, такая тенденция поспособствовала появлению языков, выполняемых интерпретаторами (*Perl*, *Ruby*) и на виртуальных машинах (*C#*, *Java*).

Во-вторых, революция в сфере коммуникаций привела к популяризации информационного пространства (далее – ИП), используемого для общения социумов. Популяризация же явилась одной из причин роста количество индивидуумов, вовлеченный в процесс общения, что автоматически дало серьезную нагрузку на телекоммуникационные устройства (далее – ТКУ), обслуживающие ИП. Для поддержания возросшей нагрузки простого усиления мощностей оборудования уже не достаточно, и задача высокой производительности его ПО (в противовес распространенному мнению) становится крайне актуальной.

За популяризацией также последовала доля критичных для компрометации данных, проходящих через ИП (например, передаче промышленных секретов или документов под грифом секретности через открытые сети, хотя и защищенным способом). Это актуализировало

требования к ПО, ужесточающие безопасность механизмов работы с данными.

И в-третьих, развитие различных направлений в области программирования (например, парадигма объектно-ориентированного программирования, оптимизация компилируемого кода) до сих пор не нашло достаточного применения в сфере ТКУ – многие промышленные устройства имеют ПО на процедурном языке *C*, построенное без оптимизации. Это, скорее всего, связано с настолько высокой значимостью правильной работы алгоритмов кода, что из-за этого производители готовы отказаться от использования сложных компиляторов и оптимизаций, не лишенных ошибок в генерируемом коде.

Резюмируя вышесказанное, сложилась ситуация, когда используемое ИП требует обслуживание с помощью ПО, обладающего, как высокой эффективностью (с точки зрения производительности), так и возможностями по защите данных. При этом создание новых языков пошло по иному пути развития, а именно в сторону эффективной (с точки зрения скорости) разработки ПО. Таким образом, разработка специализированного языка программирования, адаптированного для выполнения указанных общих требований к ПО, позволила бы снизить риски информационной безопасности, возникающие от угроз в ИП, связанных с экспансией информационных технологий.

Специализированный язык программирования

В качестве основы для предлагаемого специализированного языка выберем являющийся достаточно популярным и априори считающийся эффективным – язык *C*. А выполнение общих требований (увеличение производительность получаемого кода и дополнительные возможности защиты данных) реализуем с помощью расширения его синтаксиса (и, как следствие, семантики) – подобный процесс часто называется «наклонением языка». При этом нововведения должны хорошо «ложиться» на типовые процессоры, быть более удобны для применения программистами и не переводить язык в разряд более низкоуровневых; в ином случае суммарный эффект от использования нового языка может оказаться хуже первоначального.

С точки зрения разработки компилятора, новый язык не будет заметно сложнее *C*, а значит и качество генерируемого кода с точки зрения ошибок не изменится.

Преследуя цели лаконичности (сокращение до букв) и традиционности в названиях предшествующих языков (линейка языков *C*, *C++*, *C++11*, *C#*), дадим имя новому как *CPL*. Оно расшифровывается на английском языке как «*C with Performance and Security*», что переводится как «язык *C*, дополненный возможностями производительности и безопасности». Далее в статье будем использовать именно это название.

Предпосылками к определению требований можно считать такие особенности инструкций типовых процессоров, как битовый доступ к данным, многообразие выполнения итераций, особенности реального размещения переменных на регистрах процессора, менее абстрактная работа с памятью и другие. Также язык должен иметь конструкции, описывающие обеспечение безопасности данных.

Ну а поскольку *CPL* служит для написания программ обработки данных ИП, то и способ описания самих алгоритмов на нем должен быть удобным для применения человеком. В качестве противопоставления приведем область разработки сайтов, где основной задачей языков, как правило, считается создание эргономичного *Web*-интерфейса для конечного потребителя.

На основании предпосылок составим уточненный список требований (каждое из которых реализуется с помощью соответствующего языкового расширения) к *CPL*, включая их идентификаторы (*ID*).

1) Битовый доступ – возможность доступа к отдельным битам переменной.

2) Проверка значения бита – неявное введение типа *bool* (появившееся в языке *C++* и хранящее значение *true* или *false*) для проверки статуса бита.

3) Байтовый доступ – возможность доступа к отдельным байтам переменной.

4) Битово-диапазонный доступ – возможность доступа к диапазонам бит переменной.

5) Универсализация циклов – использование более однотипных по сравнению с языком *C* конструкций описания циклов, а именно:

   a) бесконечный цикл – определение бесконечного цикла с возможностью выхода только по командам *return*, *break* или *goto*;

   b) условие входа в цикл – условие выполнения итерации цикла, которое проверяется перед ее началом (соответствует условию входа в цикл);

   c) условие повторения цикла – условие повторения итерации цикла, которое проверяется после ее окончания (соответствует условию выхода из цикла);

   d) повторение цикла – команда повторения текущей итерации без изменения значения итерационной переменной.

6) Многокаскадный выход из цикла – выход из вложенного цикла на несколько уровней наверх.

7) Выходные параметры – возвращение нескольких значений функции (имеет смысл, когда использование указателя на структуру вместо набора регистров в качестве значения возврата неоправданно).

8) Директивы оптимизации – директивы, используемые пользователем для улучшения эффективности работы оптимизации (задается поль-

зователем на свой страх и риск, поскольку строго используется компилятором), а именно:

a) диапазон значений переменной – принудительная установка возможных значений переменной;
b) диапазон возвращаемых значений функции – принудительная установка возможных возвращаемых значений функции.

9) Адресное размещение объектов – задание точного адреса размещения переменной в памяти.

10) Безопасность объектов – указание объектов программы, содержащих или обрабатывающих критически важную информацию. Состоит из следующих типов:

a) безопасность переменных и памяти – указание переменных и областей памяти с данными, необходимых для защиты от доступа злоумышленника (например, путем хранения в защищенных областях памяти, по случайным адресам и «затирании» по окончанию использования);
b) безопасность функций – указание функций, реализующие механизмы работы с данными, алгоритмы которых необходимо скрыть от доступа злоумышленника (например, путем обфускации).

11) Инвариантность объектов – указание объектов, изменение которых недопустимо (как злоумышленником, так и в результате внутренних ошибок кода). Реализация требования может быть вынесена в специальный сервис проверки целостности объектов в случае разработки операционной системы. Состоит из следующих типов:

a) инвариантность переменных и памяти – указание переменных и областей памяти с данными, хранящих неизменную с точки зрения механизмов информацию (например, закрытые ключи шифрования);
b) инвариантность функций – указание функций, реализующих критически-важные с точки зрения выполнения механизмы (например, алгоритмы криптографического модуля).

Пункты с идентификаторами 9–11 удовлетворяют общему требованию по увеличению производительности получаемого кода, остальные – требованию безопасности данных.

Следует отметить, что большинство указанных расширений языка можно реализовать и с помощью классического языка *C*, а остальная часть – путем применения специализированных библиотек и оптимизаций компилятора. Тем не менее, это не является удовлетворительным решением: во-первых, такой исходный код будет менее удобным для написания и восприятия, а во-вторых, работа библиотек и компиля-

тора является достаточно непредсказуемой и не гарантирующей ожидаемый эффект.

Реализация новых возможностей *CPL* может ложиться, как на специфику самого процессора, так и на используемые программой библиотеки. Для разъяснения этого момента (являющегося достаточно важной особенностью языка), приведем следующие примеры.

Пример 1. Чтение байта памяти по адресу (*0xA000*) с занесением в переменную *b*.

```
addr = 0xA000;
b = (*addr).[0];
```

Данный пример может быть реализован с помощью генерации соответствующего ассемблерного кода для процессора, поддерживающего побайтный доступ к памяти. Так, в случае процессора *PowerPC* ассемблер будет иметь вид

```
lbz b, 0(addr)
```

Пример 2. Вычисление длины строки, хранящейся в защищаемой переменной *str*.

```
int strlen(str) secure;
char * secure str = “password”;
len = strlen(str);
```

Вычисление может быть реализовано с помощью библиотеки, функции которой в случае переменной с пометкой *secure* имеют специальные алгоритмы, не хранящие данные в легкодоступном виде. Так, для данного примера функция *strlen()* в процессе обработки может хранить части строки *str* в случайных адресах.

В примере пометка *secure* влияет на работу с переменной *str* на всем периоде ее жизни: блок памяти переменной должен шифроваться, а по окончанию использования (может определяться компилятором, средой выполнения или пользовательской директивой) быть принудительно «затерт».

Практика программирования

Для обоснования применимости нового языка, сравним его возможности с реализацией каждого из уточненных требований на классическом языке *C*, используя введенные ранее идентификаторы *ID*.

Сравнение проведем по следующим показателям: Л – исходный код на *CPL* иметь более лаконичный вид; Э – генерируемый на *CPL* код будет более производительным; Б – генерируемый на *CPL* код будет обладать элементами защиты данных.

Таблица сравнения с упрощенными примерами, которую также можно использовать для введения в синтаксис возможностей нового языка, приведена ниже.

ТАБЛИЦА. Сравнение возможностей языков *CPL* и *C*

ID	Реализация на языке *C*	Предлагаемая реализация на языке *CPL*	Результат сравнения
1)	получение значения 3-го бита		Л, Э
	(x & ~((1<<2))) >> 2;	*(x.3);*	
	установка 3-го бита		
	x |=(1<<2);	*(x.3)!true;*	
2)	*if((x & ~(0x4)))*	*if((x.3)?true)*	Л, Э
3)	получение значения 3-го байта		Л, Э, Б
	(x & ~(0xFF<<2)) >> 2;	*x.[3];*	
	установка значения 3-го байта в *0xAB*		
	x |= (0xAB<<2);	*x.[3] = 0xAB;*	
4)	получение значения числа, состоящего из 3-го и 4-го битов		Л, Э
	(x & ~((1<<2)|(1<<3))) >> 2;	*x.[3-4];*	
	установка 2-го и 3-го битов		
	x |= ((1<<2)|(1<<3));	*x.[3-4]!true;*	
5)			
5.a)	*while(1){}* или *do{…}while(1)*	*loop{…}*	Л
5.b)	*while(cond){…}*	*loop(cond){…}*	Л
5.c)	*while{…}(cond)*	*loop{…}(cond)*	Л
5.d)	*while(cond){* *again:* *…* *goto again;* *…* *}*	*loop(cond){* *…* *repeat;* *…* *}*	Л, Э
6)	выход из цикла на 2 уровня вверх		Л, Э
	(с помощью дополнительного флага выхода *fExit*) *int fExit = 0;* *while(cond1){* *…* *while(cond2){* *…* *fExit = 1;* *break;* *…* *}* *if(fExit)* *break;* *…* *}*	(с помощью аргумента команды *break*) *loop(cond1){* *…* *loop(cond1){* *…* *break(2);* *}* *…* *}*	

ID	Реализация на языке *C*	Предлагаемая реализация на языке *CPL*	Результат сравнения
7)	возврат пары значений из функции		Л, Э
	(с помощью структуры *S_R12*) *struct S_R12{* *int r1, r2;* *} r12, r12_ret;* *r12 = funct();* *funct() { return r12_ret; }* *или* *funct(&r12);* *funct(S_R12 *r12_) {* *(*r12_) = r12_ret;* *}*	*(r1, r2) = funct();* *funct(){* *return (r1_ret, r2_ret);* *}*	
8)			
8.a)	**Директива Отсутствует**, требование может быть частично удовлетворено оптимизацией компилятора (далее – ДО)	переменная *x* может принимать значение 10 или диапазон значений от 20 до 30 *int [10, 20-30]x;*	Л, Э
8.b)	ДО	функция *funct()* может возвращать значение 10 или диапазон значений от 20 до 30 *int [10, 20-30] funct();*	Л, Э,
9)	возможно лишь с помощью настроек линкера	принудительное размещение переменной *x* по адресу 0xA000 *int x &0xA000;*	Л, Э,
10)			
10.a)	возможно лишь с применением **Специальных Архитектур**, подходов и библиотек (далее – СА)	защита переменной *x* *int secure x;*	Л, Э, Б
10.b)	СА	защита функции *funct()* *int f() secure;*	Л, Э, Б
11)			
11.a)	СА; использование ключевого слова «const» языка *C* не решает задачу, поскольку оно имеет значение лишь для синтаксиса кода (хотя иногда и влияет на распределение переменных в бинарном коде)	установка неизменности переменной *key* *int const(secure) key;*	Л, Э, Б
11.b)	СА	установка неизменности функции *encrypt()* *int encrypt () const(secure);*	Л, Э, Б

Достоинства и недостатки нового языка

С уверенностью можно утверждать, что любой язык (кроме, естественно, группы эзотерических [2], лишенных особого смысла в применении) обладает множеством достоинств и недостатков. Это вытекает из того, что изначально любой язык предназначается для одних целей, а в меру дальнейшего распространения и отсутствия аналогов начинает использоваться и для иных. Таким образом, несмотря на то, классический язык *C* для ряда задач предпочтительнее, однако для задач написания эффективных низкоуровневых алгоритмов с поддержкой механизмов безопасности данных он будет заметно уступать языку *CPL*. И хотя, ряд особенностей языка *CPL* и может быть описано на языке *C* (впрочем, как и особенности последнего реализуемы на ассемблере), тем не менее, отсутствие в этом случае удобства при разработке программ может привести к ухудшению качества кода.

Также, нововведения в язык *CPL* не уменьшают набор возможностей *C*-языка, а лишь добавляют новые. Следовательно, *CPL* можно считать именно специализированным расширением функционала *C*.

Применение к задаче восстановления алгоритмов

Хотя изначально выдвигалось требование обеспечения возможности написания кода, имеющего высокую производительность выполнения (без потери удобства программирования), *CPL* также имеет непосредственное отношение к обратной задаче – реверс-инжинирингу.

Рассмотрим *CPL* с позиции языка для описания алгоритмов кода, восстановленных из машинного представления. Данная тема, включающая в себя метод и автоматизированную утилиту восстановления машинного кода ТКУ, неоднократно поднималась в предыдущих статьях [3, 4]. При этом, упростим язык с помощью исключения из него основной информации о типах объектов, поскольку она не сильно влияет на суть алгоритмов. Такую новую модификацию языка будем именовать как *AD* («*Algorithms Description*», означающее в переводе с английского «язык описания алгоритмов»).

Язык *AD* с точки зрения уровней абстракции (где самым низким является машинный код, а самым высоким – объекты сверхуровневых языков программирования), располагается между языком *C* и ассемблером, при этом обладая более лаконичным, чем у обоих, синтаксисом. Следовательно, описание алгоритмов машинного кода на нем предпочтительнее.

Поддержка конструкций безопасности данных позволит в процессе восстановления кода выявить потенциальные уязвимости ПО ТКУ – как собственно утилитой восстановления, так и при ручном анализе. Также,

аналитик безопасности может указать утилите наиболее приоритетные для проверки данные. В этом случае утилита пометит места в коде, в которых их безопасность может быть нарушена.

Прикладное применение

Комплексное сравнение синтаксисов и возможностей языков проведем на одном из вариантов исходного кода функции *inteN2A()* (переводящей 4-х байтную запись *IP*-адреса в текстовый формат «*dd.dd.dd.dd*»). Для введения функционала безопасности примем легенду, по которой обрабатываемая функцией переменная хранит *IP*-адрес, значение которого должно защищаться от доступа злоумышленника (например, адрес интернет-шлюза по умолчанию).

Листинг 1. Функция *inteN2A()* на языке *C*

```
char *inetN2A_ClassicC(long ina){
   static char buff[4 * sizeof("dd.")];
   unsigned char *ucp = (unsigned char *)&ina;

   sprintf(buff, "%d.%d.%d.%d",
      ucp[0] & 0xff,
      ucp[1] & 0xff,
      ucp[2] & 0xff,
      ucp[3] & 0xff);
   return buff;
}
```

Листинг 2. Функция *inteN2A()* на языке *CPL*

```
char secure *inetN2A_CPL(long secure ina){
   secure static char buff[4 * sizeof("dd.")];
   secure unsigned char *ucp = (unsigned char *)&ina;

   sprintf(buff, "%d.%d.%d.%d",
      ucp.[0],
      ucp.[1],
      ucp.[2],
      ucp.[3];
   return buff;
}
```

Листинг 3. Функция *inteN2A()* на языке *AD*

```
secure *inetN2A_AD(secure ina){
   secure buff[];
   secure *ucp = &ina;

   sprintf(buff, "%d.%d.%d.%d",
      ucp.[0],
      ucp.[1],
      ucp.[2],
      ucp.[3]);
   return buff;
}
```

Как можно увидеть из листингов, код на языке *CPL* выглядит более компактно и понятнее, чем на *C*. А ключевое слово *secure* сигнализирует о том, что генерируемый код должен быть максимально защищен от компрометации значения обрабатываемого *IP*-адреса.

Листинг же на языке *AD,* хотя и нельзя считать подходящим для компиляции (из-за отсутствия типов переменных), однако его лаконичность и «заостренность» на принципах работы алгоритмов функции (без соблюдения корректности работы с данными), позволяет использовать его для описания алгоритмов кода. В Листинге 3 сгенерированное ключевое слово *secure* сигнализирует аналитику кода о потенциальном нарушении безопасности данных в теле функции.

Заключение

Предложенный язык «среднего уровня» *CPL* позволяет разрабатывать программы, предназначением которых является эффективное и безопасное выполнение кода на ТКУ. При этом резкого снижения актуальности его применения в ближайшее время не ожидается. А удобство записи конструкций языка позволяет использовать его упрощенную версию (язык *AD*) и в обратной задаче – восстановлении алгоритмов кода ТКУ с целью поиска уязвимостей.

Библиографический список

1. **Социология** потребления: основные подходы / В. В. Радаев // Социологические Исследования. – 2005. – № 6. – С. 5–18.

2. **Основные** концепции языков программирования / У. С. Роберт. – М. : «Вильямс», 2001. – 672 с.

3. **Метод** алгоритмизации машинного кода телекоммуникационных устройств / М. В. Буйневич, К. Е. Израилов // Телекоммуникации. – 2012. – № 12. – С. 2–6.

4. **Автоматизированное** средство алгоритмизации машинного кода телекоммуникационных устройств / М. В. Буйневич, К. Е. Израилов // Телекоммуникации. – 2013. – № 6. – С. 2–9.

Аннотация

Статья посвящена оценке соответствия возможностей современных языков программирования потребностям общества. В качестве логичного продолжения результата сравнения предлагается расширение существующего С-языка с целью увеличения эффективности и безопасности кода для телекоммуникационных устройств. Также, обосновывается возможность применения расширения в обратной задаче по восстановления кода, используемой для поиска уязвимостей. Приводится комплексное сравнение синтаксисов и возможностей языков на примере кода реальной функции.

K. Izrailov

The Bonch-Bruevich Saint-Petersburg State University of Telecommunications

C-LANGUAGE EXTENSION FOR ALGORITHM DESCRIPTION OF TELECOMMUNICATION DEVICES CODE

Annotation

The article is devoted to the assessment of compliance with the capabilities of modern programming languages needs of society. As a logical continuation of the comparison proposed expansion of the existing C-language in order to increase efficiency and security code for telecommunication devices. Also, the possibility of extension is justified in the inverse problem for reconstruction of code used to find vulnerabilities. There is presents a comprehensive comparison of syntax and language capabilities at the code of real functions.

Key words: modern society, information space, telecommunication devices, the language *C*, the expansion of the language, code efficiency, restore the code, the language description of the algorithms, security, vulnerability.

References

1. **Sociologija** potreblenija: osnovnye podhody / V. V. Radaev // Sociologicheskie Issledovanija. – 2005. – № 6. – S. 5–18.
2. **Osnovnye** koncepcii jazykov programmirovanija / U. S. Robert. – M. : «Vil'jams», 2001. – 672 s.
3. **Metod** algoritmizacii mashinnogo koda telekommunikacionnyh ustrojstv / M. V. Bujnevich, K. E. Izrailov // Telekommunikacii. – 2012. – № 12. – C. 2–6.
4. **Avtomatizirovannoe** sredstvo algoritmizacii mashinnogo koda telekommunikacionnyh ustrojstv / M. V. Bujnevich, K. E. Izrailov // Telekommunikacii. – 2013. – № 6. – C. 2–9.

Израилов Константин Евгеньевич – аспирант ФГОБУ ВПО «Санкт-Петербургский государственный университет телекоммуникаций им. проф. М. А. Бонч-Бруевича», konstantin.izrailov@mail.ru