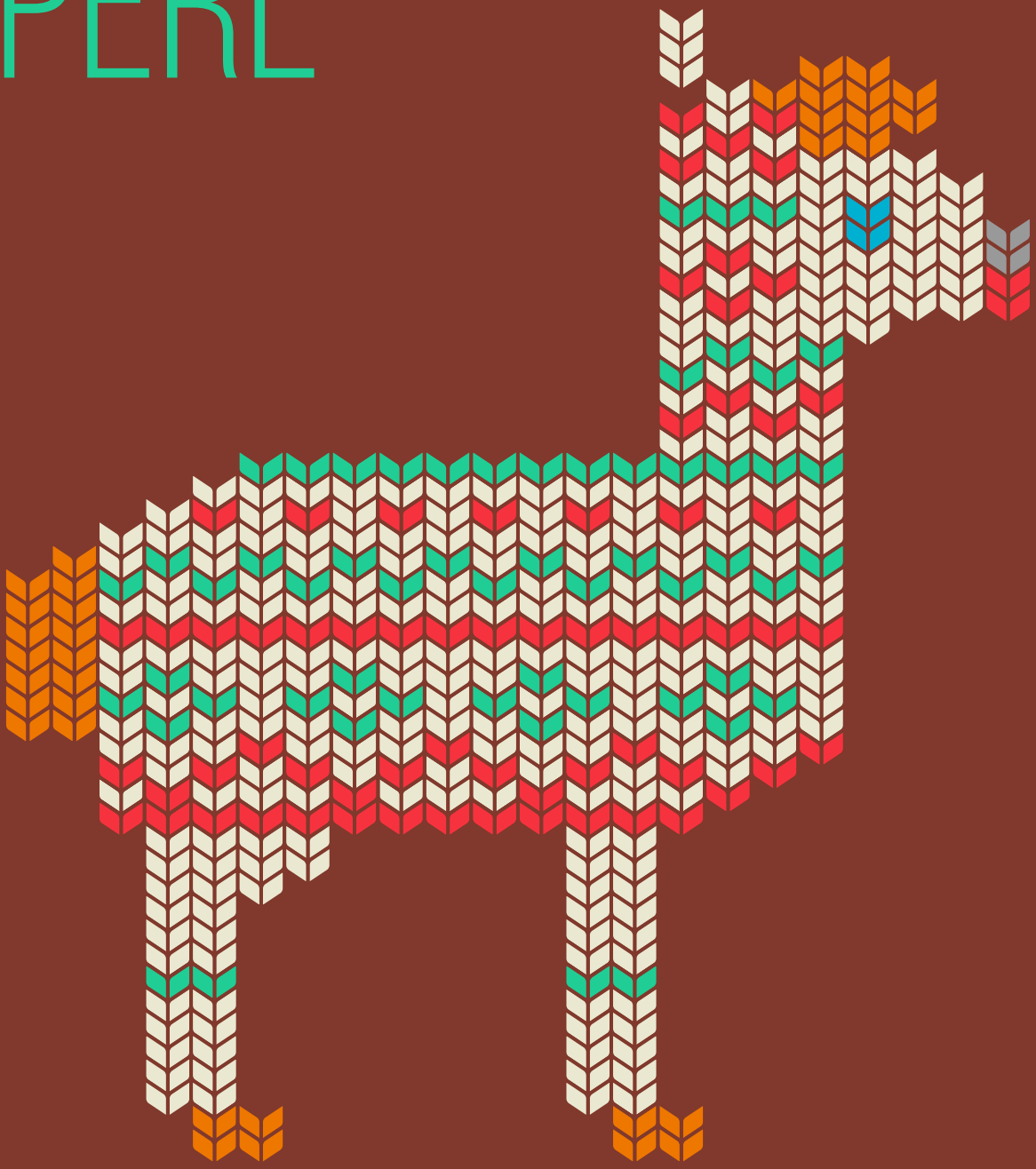
PRAGMATIC PERL

23



01/2015

pragmaticperl.com

Pragmatic Perl 23

pragmaticperl.com

Выпуск 23. Январь 2015

Другие выпуски и форматы журнала всегда можно загрузить с pragmaticperl.com. С вопросами и предложениями пишите на почту editor@pragmaticperl.com.

Комментарии к каждой статье есть в html-версии. Подписаться на новые выпуски можно по ссылке pragmaticperl.com/subscribe.

Авторы статей: Андрей Шитов, Сергей Бронников, Алексей Варяник, Владимир Леттиев

Обложка: Марко Иванык

Корректор: Андрей Шитов

Выпускающий редактор: Вячеслав Тихановский

Ревизия: 2015-10-09 11:27

© «Pragmatic Perl»

Оглавление

1 От редактора. Опрос 1

2 Взгляд на 2014 г. 3

3 Тестирование с помощью Mock-объектов 7

4 Tapper — система тестирования ПО полного цикла 19

5 Как нанять Perl-программиста 27

6 Использование TLS в Perl 40

7 Обзор CPAN за декабрь 2014 г. 69

8 Интервью с Рикардо Сигнесом (Ricardo Signes) 77

1. От редактора. Опрос

Промежуточные результаты опроса читателей журнала

С праздниками, дорогие друзья!

У нас появился онлайн irc-чат, где можно пообщаться на Perl-темы и не только. А также OPDS-каталог для тех, кто пользуется электронными книжками.

С 11 декабря прошлого года мы проводим опрос «Как сделать журнал лучше?». Опрос будет еще открыт до следующего 24-го выпуска. А пока промежуточные результаты.

На текущий момент ответили на вопросы 112 человек.

1. *Оцените журнал в целом*

    - Отлично — 58% (65)
    - Хорошо — 42% (47)
    - Плохо — 0% (0)

2. *Оцените сложность статей*

    - Все сложные — 3% (4)
    - Разного уровня — 87% (97)
    - Слишком легко — 10% (11)

3. *Оцените глубину статей*

    - Довольно подробно — 16% (18)
    - Вполне адекватно — 73% (82)
    - Очень поверхностно — 11% (12)

4. *Оцените объем выпусков*

    - Слишком много статей — 0% (0)
    - Вполне нормально — 65% (72)
    - Мало статей — 35% (38)

5. *Чего не хватает?*

    - Картинок в статьях — 46% (34)
    - Оценок статьям — 32% (23)
    - Форума на сайте — 22% (16)

Некоторые комментарии уже выполнены:

- работающее оглавления в PDF;
- веб-клиент для IRC (http://irc.pragmaticperl.com).

6. *Хотели бы вы стать автором?*

    - Да, обязательно стану — 26% (27)
    - Сомневаюсь — 43% (45)
    - Нет, только читаю — 31% (32)

По мере возможностей будем освещать подсказанные вами темы. Спасибо за ваши добрые пожелания и отзывы!

Друзья, журнал ищет новых авторов. Не упускайте такой возможности! Если у вас есть идеи или желание помочь, пожалуйста, свяжитесь с нами.

Приятного чтения.

■ *Вячеслав Тихановский*

2. Взгляд на 2014 г.

Краткий обзор заметных событий в мире Perl за прошедший год

1 января

Выпущена версия *1.0 beta* компилятора RPerl (Restricted Perl) — проекта по созданию компилятора с трансляцией кода на Perl в C++ и последующим возможным исполнением через Inline::CPP в обычном перле.

8 января

Вышли из строя несколько дисков на сервере PAUSE. Сервис был недоступен в течение трех дней.

21 января

Взломан сайт blogs.perl.org. В открытый доступ выложен дамп MySQL-таблицы `mt_author`, содержащей электронные адреса и зашифрованные пароли 3135 зарегистрированных участников. В качестве защитной меры отключены все CGI-скрипты.

25 января появился апдейт с некоторыми объяснениями: взломщик загрузил что-то через Movable Type, что и дало ему доступ. Движок платформы переустановлен и пропатчен; шифрование пароля усилено; ищется замена Movable Type.

1 февраля

В девелоперскую версию 5.19.9 добавлена реализация сигнатур функций, созданная Zefram'ом (Andrew Main). Сигнатуры появляются в качестве экспериментальной возможности и позднее вошли в четную версию 5.20.

24 марта

Конференция YAPC::Russia 2014, проведение которой было запланировано еще в конце 2013 года в Киеве, из-за нестабильной обстановки перенесена в Москву.

11 апреля произошел еще один перенос. В итоге YAPC::Russia 2014 прошла не в Москве, а в Санкт-Петербурге.

Отчеты о конференции, опубликованные в журнале:

- Отчет о конференции от организатора
- Еще один отчет о конференции

5 мая

Вышел очередной релиз компилятора Perl 6 Rakudo Star, впервые с поддержкой виртуальной машины MoarVM.

27 мая

Вышел Perl 5.20. Одновременно с этим начата работа над 5.21 и объявлено, что версия 5.22 будет выпущена в мае 2015 года.

Статьи в журнале о новых возможностях Perl 5.20:

- Что нового в Perl 5.20.0
- Синтаксические новинки в Perl 5.20
- Сигнатура функции в Perl 5.20

16 августа

Впервые прошел день CPAN. Многие годы день рождения CPAN отмечали 26 октября, по дате официального анонса CPAN. *Neil Bowers* вместе с *Philippe Bruhat* предложили считать дату 16 августа *днем CPAN*, поскольку именно 16 августа 1995 года *Andreas König* загрузил на CPAN первый модуль (`Symdump` 1.20) через интерфейс *PAUSE*.

22 августа

Объявлено место проведения конференции YAPC::Europe 2015: Гранада, Испания. На победу претендовало две группы — Гранада из Испании и Клуж из Румынии. Не исключено, что Клуж подаст заявку и на 2016 год.

2 октября

Booking.com выделил 60 000 долларов в фонд поддержки Perl 5.

12 октября

Исполнилось 20 лет модулю DBI. В этот день вышел публичный релиз проекта, работа над которым началась еще двумя годами ранее.

29 октября

Объявлены даты проведения конференции YAPC::Asia 2015 (20–22 августа). Эта конференция — десятая по счету и последняя, которую планирует организовать нынешняя команда.

YAPC::Asia собирала до тысячи человек (что втрое больше европейских конференций), в том числе из-за того, что организаторы старались привлечь участников и за пределами Perl-сообщества. Помещение для конференции 2015 года способно вместить 2000 участников.

конец октября — начало ноября

На сайте конференции FOSDEM появился анонс о том, что 31 января Ларри Уолл объявит, что Perl 6 станет доступен для использования в продакшне в 2015 году.

Статья в журнале про текущее положение Perl 6:

- Perl 6 XXI века

16 декабря

Открыт сайт конференции YAPC::Europe 2015: yapc.eu/2015.

18 декабря

Языку программирования Perl исполнилось 27 лет.

24 декабря

Объявлен конкурс 2015 CPAN Pull Request Challenge. Желающие принять в нем участие в течение 2015 года будут раз в месяц получать название случайного модуля (находящегося не только на CPAN, но и на GitHub), в который они должны сделать пул-реквест с любыми разумными изменениями в коде, тестах или документации.

■ *Андрей Шитов*

3. Тестирование с помощью Mock-объектов

Рассмотрены основные задачи Mock-объектов и их разновидности в Perl

Продолжаем цикл статей про тестирование: Тестирование в Perl. Лучшие практики, Тестирование в Perl. Практика.

Mock-объекты обычно применяются в тестировании и представляют собой аналогичные по поведению и интерфейсу имитаторы настоящих объектов, позволяют просто подставлять возвращаемые данные, проверять, какие данные были переданы в методы и т.д. Mock-объекты удобны, когда реализация настоящего объекта крайне сложна, его не так просто инициализировать, он требует особого окружения и т.п. Также они применяются при начальной разработке, когда некоторые подсистемы еще не разработаны, но их интерфейс и поведение в целом понятны.

Рассмотрим пример тестирования модуля, который делает запрос на получение списка товаров через API некоего сервиса. Интерфейс модуля будет выглядеть примерно так:

```
1 package API;
2 use strict;
3 use warnings;
4 sub new {
5     my $class = shift;
6
7     my $self = {};
8     bless $self, $class;
9
10    return $self;
11 }
12
13 sub list_orders {
14    ...
15 }
```

В качестве модуля HTTP-клиента воспользуемся `HTTP::Tiny`, он с некоторых пор идет вместе с ядром Perl. Допустим, что мы ничего не знаем о тестировании и сразу приступили к реализации модуля.

```
1 package API;
2 use strict;
3 use warnings;
```

```
4
5 use JSON ();
6 use HTTP::Tiny;
7
8 sub new {
9     my $class = shift;
10
11     my $self = {};
12     bless $self, $class;
13
14     return $self;
15 }
16
17 sub list_orders {
18     my $self = shift;
19
20     my $response = HTTP::Tiny->new->get('http://api.example.com/
          orders/');
21     die "Failed!\n" unless $response->{success};
22     my $content = $reponse->{content};
23
24     return JSON::decode_json($content);
25 }
```

У этой реализации много проблем. Во-первых, ее невозможно протестировать без доступа к интернету и работающему API, во-вторых, нет проверок на возникновение различного рода ошибок.

Имитация поведения `HTTP::Tiny`

Если мы приступим к тестированию, то сразу заметим, что нужно как-то подставить результат GET-запроса, чтобы сэмулирвать различные сценарии. Здесь нам и пригодятся Mock-объекты. Имитация объекта `HTTP::Tiny` должна как-то попасть внутрь метода `list_orders`.

Есть как минимум два способа это сделать:

- инъекция зависимости;
- фабричный метод.

Инъекция зависимости

В этом случае объект `HTTP::Tiny` передается в конструктор класса API. Таким образом мы можем передавать любой объект, который реализует метод `get` и возвращает соответствующий HASH.

```
package API;
use strict;
use warnings;

sub new {
    my $class = shift;
    my (%params) = @_;

    my $self = {};
    bless $self, $class;

    $self->{ua} = $params{ua};

    return $self;
}

sub list_orders {
    my $self = shift;

    my $ua = $self->{ua};
    ...
}
```

Тест может выглядеть следующим образом:

```
use stict;
use warnings;
use Test::More;

subtest 'returns orders' => sub {
    my $api = API->new(ua => TestUA->new)

    my $orders = $api->list_orders;

    is scalar @$orders, 1;
    is $orders->[0]->{foo}, 'bar';
};

done_testing;

package TestUA;
```

```
17 sub new {
18     my $class = shift;
19     my (%params) = @_;
20 
21     my $self = {};
22     bless $self, $class;
23 
24     return $self;
25 }
26 sub get {
27     return {
28         success => 1,
29         content => '[{"foo":"bar"}]'
30     }
31 }
```

В этом тесте создается имитатор `HTTP::Tiny`, объект класс `TestUA`, который при вызове метода `get` возвращает тестовый набор данных.

Вручную создавать Mock-объекты может быть довольно утомительно. На CPAN существует несколько модулей, который позволяют создавать Mock-объекты с различными свойствами. Здесь мы воспользуемся `Test::MonkeyMock`. После применения этого модуля тест будет выглядеть следующим образом:

```
1 use stict;
2 use warnings;
3 use Test::More;
4 use Test::MonkeyMock;
5 
6 subtest 'returns orders' => sub {
7     my $ua = Test::MonkeyMock->new;
8     $ua->mock(get => sub { {success => 1, content => '[{"foo":"
        bar"}]'}});
9 
10    my $api = API->new(ua => $ua);
11 
12    my $orders = $api->list_orders;
13 
14    is scalar @$orders, 1;
15    is $orders->[0]->{foo}, 'bar';
16 };
17 
18 done_testing;
```

С помощью метода `mock` класса `Test::MonkeyMock` можно эмулировать пове-

дение метода `get` оригинального класса `HTTP::Tiny`.

Фабричный метод

В этом случае объект `HTTP::Tiny` создается внутри API с помощью фабричного метода, который во время тестирования можно заменить своим методом, который будет возвращать Mock-объект.

```
1 package API;
2 use strict;
3 use warnings;
4
5 sub new {
6     my $class = shift;
7     my (%params) = @_;
8
9     my $self = {};
10    bless $self, $class;
11
12    return $self;
13 }
14
15 sub list_orders {
16    my $self = shift;
17
18    my $ua = $self->build_ua;
19    ...
20 }
21
22 sub _build_ua { HTTP::Tiny->new }
```

Модуль `Test::MonkeyMock` позволяет заменить методы в объектах нужными, тест будет выглядеть следующим образом:

```
1 use stict;
2 use warnings;
3 use Test::More;
4 use Test::MonkeyMock;
5
6 subtest 'returns orders' => sub {
7     my $ua = Test::MonkeyMock->new;
8     $ua->mock(get => sub { {success => 1, content => '[{"foo":"
        bar"}]'}});
9
10    my $api = API->new;
```

```
    $api = Test::MonkeyMock->new($api);
    $api->mock(_build_ua => sub { $ua });

    my $orders = $api->list_orders;

    is scalar @$orders, 1;
    is $orders->[0]->{foo}, 'bar';
};

done_testing;
```

Какой вариант подстановки использовать, зависит от задачи. В нашем случае удобнее использовать передачу через конструктор. Поэтому дальше будет использоваться первый вариант. Однако, это не значит, что он лучше или хуже. Все зависит от конкретной реализации и возможностей.

Углубленное тестирование

Рассмотрим, какие случаи нужно протестировать:

1. Модуль должен бросать исключение в случае неуспешного ответа от сервера.
2. Модуль должен бросать исключение в случае неправильного JSON.
3. Модуль должен преобразовывать ответ из JSON в Perl-структуру.
4. Модуль должен правильно вызывать метод `get` у объекта `HTTP::Tiny`.

Проверка поведения при неуспешном ответе сервера

Исключения удобно проверять с помощью `Test::Fatal`:

```
use Test::Fatal;

subtest 'throws exception when answer is not successful' => sub {
    my $ua = Test::MonkeyMock->new;
    $ua->mock(get => sub { {success => 0}});

    my $api = API->new(ua => $ua);

    ok exception { $api->list_orders };
```

```
};
```

Просто проверить, что было исключение, обычно не достаточно. Нужно проверить и ошибку. Конечно, лучше всего бросить специальное исключение и проверять имя класса, а не сообщение.

```
use Test::Fatal;

subtest 'throws exception when answer is not successful' => sub {
    my $ua = Test::MonkeyMock->new;
    $ua->mock(get => sub { {success => 0}});

    my $api = API->new(ua => $ua);

    like exception { $api->list_orders }, qr/Failed!/;
};
```

Если проверяются тексты исключений, то лучше пользоваться `like`, так как там обычно добавляется в конце информация об источнике исключения, которая может меняться при рефакторинге и ломать тесты.

Проверка поведения при неправильном JSON

```
subtest 'throws exception when answer is not valid JSON' => sub {
    my $ua = Test::MonkeyMock->new;
    $ua->mock(
        get => sub {
            {
                success => 1,
                content => 'invalid JSON'
            };
        }
    );

    my $api = API->new;
    $api = Test::MonkeyMock->new($api);
    $api->mock(_build_ua => sub { $ua });

    like exception { $api->list_orders }, qr/JSON error!/;
};
```

В реальной жизни лучше обернуть парсинг JSON в свой собственный `eval` и бросать исключение со своей ошибкой. Здесь для наглядности нам достаточно ошибки, которую бросает JSON.

Проверка, что JSON правильно преобразуется в Perl-структуру

Здесь все просто. Воспользуемся фунцией `is_deeply` для проверки получаемой структуры.

```
subtest 'correctly parses JSON response' => sub {
    my $ua = Test::MonkeyMock->new;
    $ua->mock(
        get => sub {
            {
                success => 1,
                content => '[{"foo":"bar"}]'
            };
        }
    );

    my $api = API->new(ua => $ua);

    my $orders = $api->list_orders;

    is_deeply $orders, [{foo => 'bar'}];
};
```

Проверка, что метод `get` вызывается с правильными параметрами

В этом тесте мы проверяем, что `get` вызывается с правильным `url`. `Test::MonkeyMock` позволяет это делать с помощь метода `mocked_call_args`. Например:

```
subtest 'calls get with correct arguments' => sub {
    my $ua = Test::MonkeyMock->new;
    $ua->mock(
        get => sub {
            {
                success => 1,
                content => '[{"foo":"bar"}]'
            };
        }
    );

    my $api = API->new(ua => $ua);

    $api->list_orders;

    my ($url) = $ua->mocked_call_args('get');
```

```
17
18      is $url, 'http://example.com/orders/';
19  };
```

На данном этапе полностью протестировано поведение модуля. Но есть еще что улучшить в самих тестах. Как видно, создание тестируемого объекта везде одинаково, и его можно выделить в отдельный фабричный метод. Например:

```
1  subtest 'returns orders' => sub {
2      my $ua = Test::MonkeyMock->new;
3      $ua->mock(
4          get => sub {
5              {
6                  success => 1,
7                  content => '[{"foo":"bar"}]'
8              };
9          }
10     );
11
12     my $api = _build_api(ua => $ua);
13
14     my $orders = $api->list_orders;
15
16     is scalar @$orders, 1;
17     is $orders->[0]->{foo}, 'bar';
18 };
19
20 sub _build_api {
21     my (%params) = @_;
22
23     my $ua = delete $params{ua};
24
25     return API->new(ua => $ua);
26 }
```

Также можно выделить и создание Mock-объекта `HTTP::Tiny`, который по умолчанию будет возвращать успешный ответ, но при необходимости можно изменить поведение.

```
1  subtest 'returns orders' => sub {
2      my $ua = _mock_ua();
3
4      my $api = _build_api(ua => $ua);
5
6      my $orders = $api->list_orders;
7
```

```
    is scalar @$orders, 1;
    is $orders->[0]->{foo}, 'bar';
};

sub _mock_ua {
    my (%params) = @_;

    my $ua = Test::MonkeyMock->new;
    $ua->mock(
        get => sub {
            {
                success => 1,
                content => '[{"foo":"bar"}]',
                %params
            };
        }
    );

    return $ua;
}

sub _build_api {
    my (%params) = @_;

    my $ua = delete $params{ua};

    return API->new(ua => $ua);
}
```

В итоге тест приобретает более чистый и понятный вид.

```
use stict;
use warnings;
use Test::More;
use Test::Fatal;
use Test::MonkeyMock;

subtest 'returns orders' => sub {
    my $ua = _mock_ua();
    my $api = _build_api(ua => $ua);

    my $orders = $api->list_orders;

    is scalar @$orders, 1;
    is $orders->[0]->{foo}, 'bar';
};

subtest 'throws exception when answer is not successful' => sub {
```

```
18      my $ua = _mock_ua(success => 0);
19      my $api = _build_api(ua => $ua);
20
21      ok exception { $api->list_orders };
22  };
23
24  subtest 'throws exception when answer is not valid JSON' => sub {
25      my $ua = _mock_ua(content => 'invalid JSON');
26      my $api = _build_api(ua => $ua);
27
28      like exception { $api->list_orders }, qr/JSON error!/;
29  };
30
31  subtest 'correctly parses JSON response' => sub {
32      my $ua = _mock_ua();
33      my $api = _build_api(ua => $ua);
34
35      my $orders = $api->list_orders;
36
37      is_deeply $orders, [{foo => 'bar'}];
38  };
39
40  subtest 'calls get with correct arguments' => sub {
41      my $ua = _mock_ua();
42      my $api = _build_api(ua => $ua);
43
44      $api->list_orders;
45
46      my ($url) = $ua->mocked_call_args('get');
47
48      is $url, 'http://example.com/orders/';
49  };
50
51  sub _mock_ua {
52      my (%params) = @_;
53
54      my $ua = Test::MonkeyMock->new;
55      $ua->mock(
56          get => sub {
57              {
58                  success => 1,
59                  content => '[{"foo":"bar"}]',
60                  %params
61              };
62          }
63      );
64
65      return $ua;
```

```
66 }
67
68 sub _build_api {
69     my (%params) = @_;
70
71     my $ua = delete $params{ua};
72
73     return API->new(ua => $ua);
74 }
```

Заключение

Таким образом осуществляется тестирование с помощью Mock-объектов. Не стоит однако сильно увлекаться Mock-объектами, когда они совершенно ни к чему. Опасность заключается в том, что при изменении интерфейса или поведения оригинального модуля тесты будут и дальше проходить, и вы будете уверены, что ничего не нужно исправлять. Такие ошибки должны проявиться в интеграционном тестировании, когда тестируется вся система в сборе как черный ящик. Об этом читайте в следующих выпусках.

■ *Вячеслав Тихановский*

4. Tapper — система тестирования ПО полного цикла

Рассмотрена экосистема для автоматического тестирования

В мире открытого ПО не так уж и много систем для организации тестирования ПО в промышленных масштабах, которые бы охватывали полный цикл автоматического тестирования: планирование тестирования, подготовка тестовых окружений, запуск тестов, анализ тестовых данных и т.д. Если бы меня попросили перечислить такие системы, то я бы вспомнил проект autotest, разрабатываемый компаниями RedHat, Google, IBM и проект Tapper.

Про последний я сегодня вам и расскажу. Кстати, несмотря на то, что Tapper это обычный opensource-проект, он отличается хорошей документацией. Поэтому если вы захотите внедрить систему или разобраться с ней более детально, то документация вам сильно в этом поможет. Я же, чтобы не пересказывать документацию, расскажу, какие задачи можно решать с помощью Tapper, его наиболее интересном функционале и попробую сделать так, чтобы вам захотелось попробовать его самим.

Tapper — это система, разработанная для обеспечения работы инфраструктуры, ориентированной на выполнение всех аспектов тестирования программного обеспечения. Используя Tapper, группы по контролю качества программного обеспечения могут организовать проведение всего жизненного цикла тестирования, от планирования и выполнения тестовых заданий до генерации отчетов. Tapper предназначен для тестирования больших проектов: гипервизоров, операционных систем и т.д. Просто потому, что разворачивание такой большой инфраструктуры для меньших проектов может быть неоправданным.

Проект был создан двумя разработчиками из Центра исследования операционных систем компании AMD для тестирования систем виртуализации. В 2011 году исходный код системы был опубликован под лицензией BSD. Проект разрабатывается до сих пор и используется для тестирования программного обеспечения в компании Amazon.

Что внутри

Архитектура Tapper состоит из нескольких сервисов. Каждый сервис реализован в виде набора CPAN-модулей. И это даёт гибкость в настройке всей системы: если какой-то компонент не нужен, то его можно и не устанавливать. Некоторые сервисы являются общими для всей системы и запускаются в одном экземпляре, а некоторые запускаются в нескольких экземплярах, по одному на каждом сервере. Поэтому потенциально система может неплохо масштабироваться.

Основные сервисы:

- Master Control Program — это центральный сервис, контролирующий запуск *всех* тестов в инфраструктуре. Может запускаться в единственном экземпляре для всей инфраструктуры;
- Program Run Control — процесс, с помощью которого тесты запускаются непосредственно на серверах. Необходим на каждом сервере, где будут запускаться тесты;
- Reports Receiver — сервис, принимающий отчёты о тестировании;
- Reports API — сервис для приёма дополнительной информации к отчётам о тестировании и формирования отчётов с результатами;
- Reports Web — веб-интерфейс для отображения результатов и запуска новых тестов, RSS-оповещения о результатах и т.д.

Все данные хранятся в базе данных, в роли которой может выступат как MySQL, так и sqlite.

Больше подробностей — докладах автора Tapper с конференций YAPC::EU 2009, YAPC:EU 2011 и LinuxCon EU 2011.

Как это работает

Вместо общего описания Tapper я пошагово расскажу, как происходит запуск теста, чтобы не наскучить описанием возможностей Tapper.

Перед началом тестирования все серверы должны быть зарегистрированы в Tapper. Каждому серверу можно назначить набор свойств (имя, количество оперативной памяти, производитель, количество процессорных ядер и т.д.). Тогда появится возможность запускать тесты только на тех машинах, которые обладают нужным тесту набором параметров.

Тестирование на голом железе имеет один недостаток - в случае багов в ПО вы можете остаться без доступа к ОС на сервере. Во избежание этого Tapper поддерживает интеграцию Power Distribution Unit. Но на данный момент поддерживается только одна модель, хотя поддержка других моделей не должна быть сложной. Это всего лишь небольшой модуль, который обращается к PDU удалённо. Поэтому если вы хотите построить тестирование, минимально зависимое от ручного вмешательства, то это ваш выбор.

Итак, добавляем новый сервер *pragmatic-perl*, делаем его активным и добавляем его в очередь *update8*:

```
$ tapper-testrun newhost --name pragmatic-perl --active --queue
    update8
```

Посмотрим список всех серверов:

```
$ tapper-testrun listhost --verbose

ID |      Name      | Active | TestrunID |                Comment
    | Queues
==========================================================================

30 | pragmatic-perl | active |     24976 |
                           | update8
31 | ps18           | active |     24962 |
                           | update7
27 | ps17           | active |     24929 |  testplanexperimenting
    | update7
22 | ts49           | active |      free |
                           | update7
```

После добавления серверов нужно для каждого теста подготовить сценарий подготовки сервера. В терминах Tapper это называется precondition. Это может быть установка операционной системы, копирование архива с тестом на сервер, монтирование удалённой файловой системы и т.д.

В простейшем случае precondition может выглядеть так:

```
# tapper-mandatory-fields: kernel_version
```

```
# tapper-optional-fields: kernelpkg
---
precondition_type: image
arch: linux64
image: suse/suse_sles10_64b_smp_raw.tar.gz
mount: /
partition: vz
---
precondition_type: copyfile
```

Для тестирования в промышленных масштабах подготовка тестовых окружений — обязательное требование. Tapper позволяет автоматически разворачивать ОС на сервере, подготавливать ОС к запуску тестов и осуществлять непосредственный их запуск. Установка ОС реализована за счёт интеграции с сервисами NFS, TFTP, PXE, DHCP.

Установка ОС может происходить как с использованием готовых образов установленной системы, так и штатными средствами с задействованием таких технологий, как RHEL Kickstart, Debian Preseed или OpenSUSE AutoYast.

Добавляем precondition:

```
$ tapper-testrun new [all usual options] \
    --macroprecond=FILENAME \
     -Did=value1 \
     -Dkernel=2.6.37
```

Когда пул серверов подготовлен, вы можете составить тест-план, который будет содержать список тестов, которые вы хотите запустить, серверов, на которых они будут запускаться, и ссылок на сценарии для подготовки серверов.

Обычно тест-план выглядит как шаблон, в переменные которого подставляются настоящие значения при запуске тестов:

```
### Allowed params:
###  - -Dqueue=QUEUE
###  - -Dtests=TEST1,TEST2,TEST3
###  - -Dmachines=HOST1,HOST2
[%# define default values %]
[%- IF queue == '' %][% queue = 'update8' %][% END -%]

[%- IF tests == '' %][% tests = 'tcpbench' %][% END -%]
[%- IF machines == '' %][% machines = 'pragmatic-perl' %][% END
   -%]
```

```
[% AllTests = tests.split(',') %]
[% AllDistros = distros.split(',') %]
[% AllMachines = machines.split(',') %]
[%- FOREACH machine = AllMachines %]
[%- FOREACH test = AllTests %]
---
type: multitest
description:
  shortname: [% test %]
  topic: Topic-[% AllTests.join('-') %]
  queue: [% queue %]
  requested_hosts_all:
  - [% machine %]
  preconditions:
  - ...
  -
    precondition_type: testprogram
    program: /opt/tapper/bin/tapper-testsuite-autotest
    parameters:
--test
      - [% test %]
  - ...
```

После составления тест-плана его нужно добавить в планировщик:

```
$ tapper-testrun newtestplan --guide --file pragmatic-perl-
   testplan
  This is an example testplan
  Allowed params:
   - -Dqueue=QUEUE
   - -Dtests=TEST1,TEST2,TEST3
   - -Dmachines=HOST1,HOST2
```

Просмотр очереди тестов:

```
$ tapper-testrun listqueue -v
11 | update8    |  1000
17 | update7    |   100
8  | update7_1  |   100
4  | update7_2  |   100
```

где первая колонка это номер очереди, вторая — название и третья — приоритет очереди по сравнению с другими очередями.

Посмотрим детально на нашу новую очередь:

```
$ tapper-testrun listqueue --name update8
```

```
Id: 10
Name: update8
Priority: 1000
Active: yes
Bound hosts: pragmatic-perl
Queued testruns(ids): 146, 138
```

Детали запланированного теста:

```
$ tapper-testrun list --id 146
id: 146
topic: track-workload-stress-opensuse_11.4_32
state: schedule
queue: update8
requested hosts: pragmatic-perl
auto rerun: no
precondition_ids: 15
```

Получили тест, запущенный на машине *pragmatic-perl*, в очереди *update8*.

Результаты тестирования

Как выше было сказано, Tapper позволяет агрегировать все результаты в одном месте для последующего анализа и формирования отчётов о тестировании.

Все тестовые результаты сохраняются в стандартном формате TAP (Test Anything Protocol), популярном в Perl-сообществе. Формат очень удобен для понимания даже неподготовленному человеку:

```
1..15
ok 1  TestSnapshotsCT.hostname_check
ok 2  TestSnapshotsCT.diskspace_check
ok 3  TestSnapshotsCT.veth_IP_check
ok 4  TestSnapshotsCT.veth_MAC_check
ok 5  TestSnapshotsCT.venet_IP_check
ok 6  TestSnapshotsCT.physpages_check
ok 7  TestSnapshotsCT.swappages_check
ok 8  TestSnapshotsCT.cpu_limit_check
ok 9  TestSnapshotsCT.cpu_units_check
ok 10 TestSnapshotsCT.Check_Quotas
ok 11 TestSnapshotsCT.resize_increase_ve_start
```

```
ok 12 TestSnapshotsCT.resize_decrease_ve_start
ok 13 TestSnapshotsCT.resize_increase_ve_stop
ok 14 TestSnapshotsCT.resize_decrease_ve_stop
ok 15 TestSnapshotsCT.Leak_checks
```

Но из-за своей простоты у формата есть недостаток — формат никак не регламентирует поля для дополнительной информации о тестовом окружении. Tapper в этих целях использует комментарии:

```
1..1
ok 1 - tapper-suite-meta
# Tapper-suite-name:    Pragmatic Perl Testsuite
# Tapper-suite-version: 1.0
# Tapper-machine-name:  pragmatic-perl
# Tapper-uname:         Linux 2.6.32-042stab102.8
# Tapper-osname:        Parallels Cloud Server
# Tapper-cpuinfo:       Intel(R) Xeon(R) CPU E5-2403 0 @ 1.80GHz
# Tapper-ram:           32776
```

Интеграция с другими сервисами и инструментами

Проект поддерживает интеграцию с другими инструментами для тестирования и анализа результатов:

- запуск тестов в уже упомянутой системе autotest, предназначенной для тестирования ядра Linux;
- выгрузку тестовых данных в Codespeed, веб-приложение для анализа результатов тестирования производительности;
- возможность планировать тесты с помощью TaskJuggler, инструмента для управления проектами;
- возможность установки операционной системы с помощью Cobbler.

Заключение

К сожалению разработка проекта продвигается не так быстро, как хотелось бы. Steffen Schwigon, разработчик Tapper, обещал выложить сделанные в Amazon изменения в репозиторий на Github, но пока этого не случилось.

Несмотря на это, существующего функционала уже достаточно для построения рабочей системы тестирования. Если вы заинтересовались проектом, то приглашаю поучаствовать в разработке и тестировании. Исходный код расположен на Github, модули для установки — на CPAN. Если хотите попробовать Tapper в деле, то есть несколько вариантов установки: с помощью CPAN-модулей и с помощью пакетного менеджера в ОС. Установочные пакеты есть для Fedora, ALT Linux, и есть OpenBSD-порт (не добавлены в официальное дерево портов, поэтому пакеты пока не собираются).

■ *Сергей Бронников*

5. Как нанять Perl-программиста

Про один удачный эксперимент по найму программистов

Если вы один из тех, кто когда-либо собеседовал человека на работу, вам наверняка приходилось сталкиваться с проблемой подбора **хороших** кадров. В статье я постараюсь опубликовать максимум всех материалов, которые были задействованы во время постановки эксперимента.

Но прежде чем начать — позвольте представиться.

Об авторе

Меня зовут Алексей Варяник, друзья зовут меня cono (как, в принципе, и многие коллеги).

Занимаюсь Perl-разработкой начиная с 4-го курса университета (2004 год). В стенах comScore.com подрос до TeamLead. После того, как проект отдали на поддержку в Кувейт, перешёл работать в компанию Provectus на проект livenation.com (поддержка сайта ticketmaster.com). В скором времени на этом проекте вырос до Software Engineering Manager, а как вам всем наверняка известно, одной из обязанностей software manager является подбор и собеседование «кадров» :)

Собеседования

По началу набирать программистов не составляло особого труда. HR-отдел находил «бесценных» Perl-разработчиков, мы их собеседовали, ну и, если уж совсем понравился, брали на работу. Команды состояли из back-end- (BE, perl), front-end- (FE, js) и QA-инженеров. Именно в такой последовательности и хотели нанимать. Но к сожалению, всё выходило с точностью до наоборот. Сначала появлялись QA, после чего FE, и только лишь после гигантских усилий BE. Постепенно проблема подбора BE становилась всё сложнее и сложнее. То ли разработчики «заканчивались», то ли люди становились более избалованные, и Чёрное море больше никого не привлекало :)

Слышу звон, не знаю где он

Кололись, плакали, но продолжали есть кактусы. Одним из первых «спасательных» решений было — нанимать людей из других областей (язык, технологии). И вот тут началось самое интересное. Наконец-то пошли люди и довольно много, но в то же время достаточно интересные «кадры».

> — Опишите пожалуйста процесс передачи GET параметров? — Они появляются в переменной `$_GET`.

Бывали и очень серьёзные ребята, которые пишут на «плюсах» под кастомную архитектуру:

> — Объясните пожалуйста, что такое Template в C++? — Шаблон! — Хорошо, а для чего они используются? — Эээ...

Бывали и просто хорошие программисты:

> — Напишите пожалуйста функцию, которая принимает целое число и возвращает 0, если число чётное и 1 в противном случае. — Я, наверное, вам не подойду.

Нагромоздив некое количество костылей (предварительное тестовое задание, технические вопросы от HR, ...), нам удалось снизить поток шарлатанов и бездарей. В целом, мы смогли нанять ещё парочку смышлёных ребят.

И вот тут-то звоночек и зазвонил. Решили рассматривать другие варианты найма.

Предыстория

В году эдак 2008-м, когда я был программистом на banner.kiev.ua, соседи по офису набирали Perl-разработчиков на проект mail.ua (уже mail.ru), и встретили те же самые проблемы, что и мы сейчас. Тогда было принято решение:

набирать смышлённых мальчуганов и девчонок и обучать их великому делу! (программировать на Perl).

В тот раз этот эксперимент закончился очень удачно, мы набрали замечательных ребят (и девочку, привет Янка!). Некоторые из них, даже открыли собственную компанию: WebbyLab.

Набор

Тогда я участвовал как преподаватель, но в этот раз пришлось поучаствовать и в организации процесса. Списался с одним из организаторов прошлых курсов (Олег Савчук aka oSa), попросил перечень тем, по которым вели прошлые курсы:

1. Обзор курсов. Обзор возможностей Perl. Настройка среды.
2. Введение в Perl. Скалярные данные. Основные операции с числами.
3. Массивы и хеши. Цикл foreach. Специальные переменные (`$_`). Операторы `next`, `last`, `redo`.
4. Операции над строками. Регулярные выражения.
5. Процедуры и функции. Пакеты и модули.
6. Ссылки и объекты.
7. CGI.
8. DBI.

Темы равномерно распределили между разработчиками, уже работающими у нас. Попросил каждого подготовить доклад по теме (не более чем на полтора часа) и тестовые задания (о них более детально расскажу чуть позже).

Далее вступили в бой наши HR и PR. Отдельное спасибо хочется сказать Оле Ковальской, она провела титанический труд по общению с кандидатами, а их было уж **очень** много.

Было сформировано следующее объявление о наборе:

> Время учить Perl

> Perl устарел? Нет, не слышали… Perl не просто жив и процветает, он еще и достаточно востребован!
>
> Provectus IT объявляет набор на курсы Perl с возможностью последующего трудоустройства.
>
> Обучение будет включать в себя лекционные и практически занятия с акцентом на самостоятельное обучение. Такой подход предоставит учащимся возможность в дальнейшем быстрее вникнуть в проект, справляться со сложными задачами, оперативно принимать решения.
>
> Курс рассчитан на 3 недели (по 2 занятия в неделю). Всех желающих ждет строгий отбор ведущих Perl-истов команды Live Nation/Ticketmaster.
>
> Требования:
>
> 1. Предпочтительно образование в области компьютерных наук или математики.
> 2. Логическое и аналитическое мышление.
> 3. Способность быстро обучаться.
> 4. Наличие опыта работы с любым объектным языком программирования.
> 5. Базовое знание Perl будет плюсом.
> 6. Английский язык (письменный и устный).
> 7. Опыт работы с MySQL (предпочтительно) либо с другими СУБД.
> 8. Знания алгоритмов и прикладной математики.
>
> Обучение будет проходить на бесплатной основе!

На выходе мы получили огромное количество резюме (более сотни). И люди попадались абсолютно разные: студенты, бывалые программисты, моряки дальнего плавания и даже преподаватель :)

Из всего списка мы отобрали 54 человека и разослали им пригласительное письмо. Надеясь хоть как-то отсеять народ, мы придумали три тестовых задания, которые нужно было выполнить, чтобы прийти на собеседование. Но до собеседования дошли все 54 человека. Было принято решение вести собеседование по проделанным тестовым заданиям и не более 20 минут (такой

вариант конечно даёт очень много false-negative-результатов, но пришлось чем-то жертвовать).

Пригласительное письмо

> Привет!
>
> В ответ на Ваше резюме на позицию trainee Perl, спешим пригласить Вас на собеседование, которое состоится в *день_недели*, *дата* июля в *время* в виде группового отбора.
>
> В прикреплённом файле Вы найдёте тестовое задание. Мы рассчитываем на Ваше выполнение этого задания на любом понятном нам языке программирования (BASIC, C, C++, Java, Pascal, Perl, Python, PHP) или просто в виде алгоритма. Собеседование будет включать Ваше устное описание выполненного задания и ответы на наши вопросы. Выполненное тестовое задание просим подготовить по возможности в виде распечатки (если с этим возникнут сложности — приносите в электронном виде, и мы поможем распечатать).
>
> Как будет проходить собеседование? Мы попросим Вас заполнить анкету, а затем побеседовать с нашими Software Engineers на тему выполненного задания. Даже если Вы затрудняетесь выполнить часть задания — это не беда: нам будет интересно пообщатсья с Вами о ходе мысли и в целом на программистские темы. Так как мы работаем на международную компанию, то просим Вас подготовиться рассказать о себе на английском языке.
>
> Как будет проходить обучение? Курс рассчитан на 7 тем, по каждой планируется 1-2 лекции и практические задания. Лекции будут происходить в первой половине дня (ориентировочно с 10:00) дважды в неделю в нашем офисе. Там же Вы регулярно сможете пообщаться с опытными разработчиками и задать им интересующие вопросы. Курс продлится около 5 недель. По окончании курса все успешные студенты будут допущены к собеседованию, по результатам которого происходит трудоустройство. Мы хотим взять на работу всех. Нет, честно.
>
> Как мы сообщали ранее, курсы бесплатные, а значит — мы оставляем за собой право выбора учеников. Благодарим за понимание.

Тестовые задания

Про тестовые задания хотелось поговорить более детально и упомянуть, на что обращали внимание, какие вопросы задавали. Итак, первое задание:

> В цикле от 1 до 71 вывести `c*10` для каждого `c`, кратного 3, и `c*100` для каждого `c`, кратного 7. Если `c` не кратно 3 и `c` не кратно 7 – не выводить ничего. Посчитать и вывести две суммы выведенных чисел.

Первое задание очень простое, больше для мотивации, чем для проверки каких-либо знаний, чтобы дать человеку понимание что он может выполнить эти задания.

Вопросы задавали по code-flow и оптимизации:

- а что случится если переставить эти строки местами;
- как модифицировать программу так, чтобы выводить `c` в случае некратности 3 и 7 (числа не просто так были выбраны простыми, дабы увидеть вариант: `c % 21`);
- как прооптимизировать программу (хотели добиться ухода от перебора 1 .. 71 и приходу к простому суммированию).

Второе задание немного сложнее:

> Написать функцию, которая сгенерирует пагинацию (список номеров страниц с указанием текущей страницы).
>
> Входные параметры:
>
> a: общее количество записей b: количество записей на странице c: стартовая страница
>
> Пагинация имеет вид:
>
> < 1 2 [3] 4 5 ... >

> Всегда отображается 5 страниц и троеточие в стороне, в которой есть неотображённые страницы. Если текущая страница приближается к первой или последней — всё равно выводим 5 номеров страниц, приближая пагинатор к нужной стороне.
>
> < … 6 7 8 [9] 10>

Эта задача для определения крайних случаев (edge-case'ов):

- общее количество записей 0;
- стартовая страница вне диапазона;
- страниц меньше 5 (многоточие должно было исчезнуть с обеих сторон).

Если какой-либо критерий не был выполнен, спрашивали, как нужно модифицировать программу чтобы добиться нужного эффекта.

И последняя задача (две в одной):

> На входе – бинарный файл. Представив его как последовательность символов в кодировке ASCII, подсчитать количество символов z и Y в этом файле. Вывести оба числа в системах счисления от 2 до 16.

В этом задании преследовалось две цели, но во время проверки нарисовалась и третья:

- capacity testing (*нагрузочное тестирование* — прим. ред.) — понимает ли кандидат, как нужно изменить программу, если на вход подаётся файл размером в несколько террабайт;
- проверка математических знаний (системы счисления);
- соответствие техническим требованиям, почему-то именно эта задача вызвала большие сложности с пониманием технического задания (Y & y — это разные буквы, от 2 до 16, это и 3, и 4, и 5, …).

Начало

В итоге мучительных интервью-сессий были отобраны 20 человек (планировалось 10, очень не хотелось прощаться с некоторыми). И одного попросили добавить по блату (мы же в Украине живём, куда ж без этого :)

ИТОГО: 21

Первая лекция была вводной, мы просто собрались вместе, познакомились. Ученики увидели своих будущих учителей, учителя — учеников. Одним из важных на мой взгляд моментов было расстановка всех точек над И. А именно:

1. Это бесплатные курсы.
2. Лекции будут читать простые программисты (и далеко не все из них с даром преподавателя).

Т.е. на первом же этапе мы определили простые правила приличия: относиться с уважением друг к другу, ведь никто никому ничего не должен. Любые несогласные могли покинуть данные курсы в любой момент. Ребята оказались все взрослые, и курсы проходили «без сучка и задоринки».

С технической стороны всем были выданы виртуальные машинки для VirtualBox (дабы снизить риск ошибок environment и архитектуры). Базовые инструкции: как установить виртуальную машину, как зайти, как писать код, как тестировать, проблемы с сетью и т.д., и т.п.

Домашнее задание

Старт дан, и на этом этапе мы начали задумываться о контроле успеваемости и о том, как узнать результаты в конце эксперимента. Материал хотелось закрепить практическими задачами. Поэтому каждый из восьми преподавателей взял на себя обязанность придумать (подсмотреть) 1-3 задания по своей теме.

1. Написать программу нахождения корней квадратного уровнения: `ax2 +bx+c=0`.
2. Перевод числа из десятичной системы счисления в другую и обратно.
3. Подсчитать частоты возникновения слова в тексте.
4. Нахождение циклов в циклической структуре данных.
5. Замены подстрок в строке (хеш соответствий).
6. Написать простейший shell.
7. Произведение двух матриц.
8. Аналог `Data::Dumper`.
9. Написать классы Date & Calendar.
10. SQL-executor.
11. CGI-приложение.

Более детальное описание заданий вы можете найти в этом документе.

К каждому тестовому заданию был написан некий набор тестов, о которых мы поговорим далее.

Проблема №1: Проверка

Дальнейший процесс развивался спонтанно, эволюционно и по мере возникновения проблем. И первой такой проблемой явилась проверка выполненных заданий.

Т.к. environment был задекларирован ещё на первой встрече, среда тестирования уже была определена. VirtualBox обладает замечательным механизмом snapshot-ов. При каждом тестировании востанавливаем виртуальную машину к исходному (не испорченому) состоянию и запускаем тесты.

Запускать 21 программу * N тестов вручную было очень накладно, поэтому на коленке были написаны shell-скрипты для тестирования.

На первых заданиях делались всяческие поблажки: вывод подправить, shebang исправить, и т.д. Не все были готовы к олимпиадному подходу (строгие входные данные, строгие выходные данные), но после первых результатов всё быстро стало на круги своя.

Вариант приёма исходников тоже по-началу был не идеален (кто в скайп, кто в почту: — А можно я исправленную версию пришлю?). На коленке была написан небольшой скрипт загрузки результатов, через которую можно было отправить решение хоть миллион раз до определённого времени. После этого времени — только один раз.

Все результаты складывались в git (на тот момент не публичный). В этом же репозитории находились и номинальные решения от «преподавателей».

Тестируемому скрипту передавалось имя файла с входными данными (тестовые данные определены в `model/*/dat`).

Проблема №2: Статистика

Статистики хотелось получить как можно больше — а значит и объективнее. В погоне за статистикой первыми были получены результаты прогона тестов, они имели следующий вид:

mark	n/n	00_example	01_large_linked_list	02_hash_sort	03_array_sort	04_get_
0,00	aleksandr.proydenko	-	-	-	-	-
5,55	aleksandr.zhogolko	-	+	+	+	-
0,00	alexander.sofontsev	-	-	-	-	-
0,00	alexey.konovalenko	-	-	-	-	-
0,00	andrej.kudjurov	-	-	-	-	-
0,00	denis.belov	-	-	-	-	-

Рис. 1: Таблица результатов

+/- означают прошёл тест или нет, и «T» — прерван по таймауту (если алгоритм зацикливался).

Этой статистики показалось мало, а так как результаты загружались через Web, можно было подсчитать среднее время, затраченное на решение (планировалось использовать эту статистику, если кто-то будет с одинаковыми оценками, но так и не пригодилась :)

Результирующая таблица получилась следующей:

average mark	n/n	00	01	02	03	04	05	06	07	08	09
3,25	aleksandr.proydenko	5,00	9,06	2,85	8,33	7,33	0,00	0,00	0,00	0,00	0,00
6,65	aleksandr.zhogolko	5,00	0,00	7,14	6,66	7,33	8,82	10,00	5,55	10,00	6,00
3,18	alexander.sofontsev	5,00	8,83	7,14	1,66	8,66	0,58	0,00	0,00	0,00	0,00
4,24	alexey.konovalenko	5,00	0,46	7,14	1,66	3,33	8,23	1,66	0,00	10,00	5,00
6,07	andrej.kudjurov	8,75	0,46	4,28	6,66	7,33	8,23	10,00	0,00	10,00	5,00
6,12	denis.belov	10,00	1,16	10,00	7,50	9,33	8,23	10,00	0,00	0,00	5,00
7,22	dima.mukaeliants	2,50	1,16	8,57	6,66	10,00	8,82	6,66	8,88	10,00	9,00
6,18	dmitriy.poltavchenko	6,25	0,46	8,57	6,66	9,33	8,23	10,00	3,33	5,00	4,00
4,69	dmitry.solodukhin	1,25	7,44	7,14	1,66	6,66	7,64	0,00	1,11	10,00	4,00

Рис. 2: Общая таблица результатов

Статистика выкладывалась постепенно и соревновательный дух не прекращался ни на секунду.

Проблема №3: Апелляция

Как бы не приноравливались наши «ученики» к оллимпиадному режиму, всё равно допускалось много ошибок, связанных с выводом (и даже ошибки в номинальных решениях). Ввиду этого было принято решение соблюсти все каноны Олимпиадного режима и добавить этап апелляции. Как говорилось ранее, исходники складывались в git, поэтому с точки зрения реализации решили использовать github + merge request. Патчи принимались только те, которые не изменяли логику работы программы, а только ошибки вывода.

В итоге наши студенты ещё и научились простейшим операциям с git (commit, push, merge-request, ...)

Английский

В конце курса обучения было обещано собеседование с присутствием заказчика (американец), поэтому одно из изначальных требований было умение объясниться технически на английском. У нас в компании проводятся разго-

ворные встречи клуба английского языка (бесплатные), поэтому всем участникам эксперимента было предложено бесплатное посещение занятий в течение всего курса обучения.

Собеседование

Собеседование решили проводить от высшей оценки к низшей. Статистически вышло так, что мы пособеседовали только тех, у кого бал был выше пяти.

На собеседованиях ребята очень волновались, хотя на английском их сильно и не спрашивали, но присутствие иностранного заказчика наводило некий ужас :)

В результате собеседований мы наняли пять человек на ставку Junior BE и ещё двух человек без зарплаты на позицию trainee. К сожалению, с нами на текущий момент остались только лишь четверо.

Выводы

Во всём есть свои «+» и «-», именно их и хотелось бы опубликовать как выводы.

Достоинства

- Мы получили четырех SUPER-Junior разработчиков. Которые в течение месяца смогли показать отличную обучаемость. Как и в течение дальнейшего периода работы на компанию.
- Интересный опыт для «преподавателей».
- Последующее отношение «преподавателей» к своим успешным «ученикам» — очень тёплое и способствует хорошему командному духу (все заинтересованы в дальнейшем их успехе).
- Вхождение в проект этих разработчиков в разы быстрее, чем просто нанятых посредством интервью (опять же ввиду скорости обучаемости).

- Есть маленький шанс получить на выходе сразу Middle-разработчиков (если входные данные были опытными программистами :)
- Имидж компании.
- Новые люди в Perl-сообществе.

Недостатки

- Таким образом практически невозможно набрать Senior-разработчиков (но при должном подходе junior очень быстро могут стать senior-разработчиками).
- Затраченное время (хотя вы получите целеустремлённых разработчиков, которые в разы лучше «зажравшихся» професионалов).
- Затраченное время. И всё же, как бы я не убеждал вас в успешности эксперимента, четыре раза подумайте о затраченном времени на него. Есть ли в этом смысл конкретно для вас. Не стоит относиться ко всему как к панацее, переложите изложеный опыт на свои реалии.

■ *Алексей Варяник*

6. Использование TLS в Perl

Ликбез по криптографии и практика применения TLS (Transport Layer Security — безопасность транспортного уровня) в Perl.

В последнее время в интернете обозначился тренд по использованию защищённой передачи данных. Откровения Эдварда Сноудена о работе NSA, стремление спецслужб различных стран прослушивать каждый бит информации, а также всевозможные атаки по подмене содержимого передаваемого по сети контента заставляют всех более серьёзно подходить к защите коммуникаций.

Основной способ защиты соединений — это использование криптографического протокола TLS, позволяющего транслировать данные приложения в зашифрованном виде. TLS выполняет несколько защитных функций: аутентификация сторон обмена, шифрование и контроль целостности передаваемого трафика. Таким образом исключаются атаки по перехвату трафика между сторонами коммуникации с попыткой подменить сервер для клиента и клиента для сервера, обеспечивается конфиденциальность и неизменность передаваемой информации.

Важно также отметить, что предшественник TLS — протокол SSL, разработанный корпорацией Netscape, больше не считается безопасным. Разработаны методологии атак, которые позволяют расшифровывать части передаваемых сообщений, поэтому использование SSL должно быть исключено в принципе.

Прежде чем приступить к детальному рассмотрению протокола TLS, отметим самый главный недостаток TLS и криптографических алгоритмов — это невероятная сложность теории и математика 80-го уровня. Это отталкивает от изучения и приводит к заучиванию команд и строчек конфигураций протокола как магических мантр. В свою очередь, тотальное невежество приводит к выраженному консервативному развитию протоколов: не дай бог изменение сломает какую-нибудь древнюю систему на SSL 2.0, которую никто не умеет настраивать, давайте-ка оставим всё обратно-совместимым. Удивительно, но обратная совместимость — это именно то, что нужно злоумышленникам — вы можете использовать самые передовые технологии безопасности, но оставляете лазейку для старых и уязвимых протоколов и шифров. Косвенно на такое положение дел влияют государства, стандартизируя бо-

лее слабые алгоритмы или параметры шифрования, чем существуют на сегодняшний день.

К счастью, браузерная гонка, развернувшаяся в последние годы, меняет положение к лучшему, предоставляя клиенту самые свежие и сильные протоколы шифрования, стимулируя и развитие серверных средств шифрования. Например, в Firefox при использовании протоколов SPDY/HTTP2 по умолчанию могут использоваться только самые строгие профили шифрования: TLS последней версии 1.2 и шифры с использованием эллиптических кривых. Поисковик Google стал учитывать при ранжировании сайтов наличие https, чтобы стимулировать веб-мастеров обзаводиться защищённым доступом к сайтам. Ну и новость последних дней — в 2015 г. браузер Chrome будет показывать обычные http-сайты как небезопасные так же, как показывает https-сайты с невалидным сертификатом.

Словарик криптографа

До изложения основной темы статьи необходимо кратко описать основные используемые термины, чтобы облегчить понимание базовых принципов работы защищённых коммуникаций.

Алиса, Боб и другие персонажи

При построении схем обмена данными, криптографы придумали различные имена для различных участников обмена информации. Например, вместо указания, что участник **А**, передаёт данные стороне **Б**, им дали более благозвучные имена: Алиса и Боб. Поэтому если вы слышите упоминания об Алисе и Бобе, то на 100% речь идёт о процедуре передаче данных между двумя сторонами, без привязки к конкретным личностям.

Также могут упоминаться Ева (от слова *eavesdropper* — подслушивающий), участник, который может иметь возможность прослушивать трафик обмена между другими сторонами. Ну и Мэллори (от слова *malicious* — злонамеренный), который может не только прослушивать, но и изменять данные, передаваемые другими участниками; сама атака по подмене данных в этом случае называется MITM (man in the middle — человек посередине).

Шифрование

Шифрование — это процесс изменения открытого сообщения в шифрокод по определённому алгоритму. Дешифрование — это обратный процесс преобразования шифрокода в исходное сообщение.

Как правило, сами алгоритмы шифрования всегда известны, а их стойкость ко взлому определяется большей частью выбранным размером ключа, который обычно измеряется в битах. Ключ шифрования — это некоторое секретное число, которое используется в процессе шифрования.

Сами алгоритмы шифрования делятся на два вида: симметричные и асимметричные.

Симметричные алгоритмы

Симметричные используют один и тот же ключ для шифрования и дешифрования, этот ключ должен держаться сторонами обмена в секрете. Также различают потоковые и блочные шифры. Потоковые шифры могут кодировать данные побайтно, например алгоритм RC4. Блочные шифры кодируют данные блоками, например по 64, 128 бит. Один из самых первых стандартизированных и широко применяемых алгоритмов блочного шифрования стал алгоритм DES, который использовал ключи длиной 56 бит и кодировал блоки в 64 бита. В 2001 году был стандартизирован алгоритм AES, также известный под именем *Rijndael*, который и сейчас активно используется.

Симметричные алгоритмы, как правило, работают очень быстро. Многие современные процессоры имеют расширенный набор инструкций для ускорения выполнения операций шифрования. Основной недостаток симметричных алгоритмов — обе стороны должны иметь общий секретный ключ. Если одной стороне потребуется передать ключ, то канал передачи должен быть защищён от прослушивания/изменения.

Асимметричные алгоритмы, или алгоритмы с открытым ключом

Асимметричные алгоритмы используют разные ключи для процессов шифрования и дешифрования, один из этих ключей называют открытым (публичным), а другой закрытым (приватным). Закрытый ключ всегда держится в секрете, в то время как открытый может свободно распространяться и не составляет секрета.

Например, если Алиса хочет передать зашифрованное сообщение Бобу, она воспользуется публичным ключом Боба для шифрования. Расшифровать шифротекст сможет только Боб, у которого есть приватный ключ.

Основное преимущество асимметричного шифрования в том, что публичный ключ для шифрования может передаваться по незащищённому каналу. Слабость алгоритма заключена в том, что он основывается на некой сложной вычислительной задаче, и, до тех пор пока вычислительная сложность сохраняется, алгоритм может использоваться. Современные процессоры непрерывно совершенствуются, всё это приводит к тому, что требуются всё большие длины ключей, чтобы противостоять натиску прогресса.

Один из первых и остроумных алгоритмов с открытым ключом, который до сих пор широко используется является алгоритм RSA.

Ключи RSA генерируются по следующему алгоритму:

- Выбираются два больших простых числа p и q.
- Вычисляется произведение n = p * q.
- Вычисляется φ(n)= (p−1)(q−1).
- Выбирают публичную экспоненту e, взаимно простую с φ(n) (например, 3, 257, 65537).
- Находят секретную экспоненту d, такую что d * e 1 (mod φ(n)).

Теперь значения e и n публикуются как публичный ключ, а значения d, n становятся секретным ключом.

Чтобы зашифровать сообщение m, число m возводится в степень e и находится остаток от деления на число n:

```
1 m^e mod n = c
```

Чтобы расшифровать сообщение, шифротекст возводится в секретную экспоненту d и находится остаток от деления на n:

```
c^d mod n = (m^e mod n)^d mod n = m^(e*d) mod n = m
```

Действительно, по условию выбора экспоненты d мы получим исходное зашифрованное сообщение.

Вся сложность расшифровки посторонним лицом сводится к сложности вычисления чисел p и q по известному n, т.е. нахождения разложения на множители очень большого числа. Известны успешные операции по взлому RSA-ключей длиной в 768 бит, не за горами и взлом 1024-битного ключа, поэтому на сегодняшний день минимальный рекомендуемый размер ключа RSA — 2048 бит.

Но увеличение ключа приводит к дополнительной проблеме — требуется большие вычислительные ресурсы для шифрования. Например, на типичном современном процессоре (Intel Xeon X5460) можно выполнить порядка 500 операций шифрования в секунду ключом RSA 2048 бит, в то время как AES-128 может шифровать поток в 100 Мбайт/сек. Поэтому обычно RSA используют только для передачи секретного сеансового ключа симметричного шифрования, который затем уже используется для шифрования данных.

Аутентификация и целостность

Аутентификация, в широком смысле, — это процедура проверки подлинности. Может проверяться подлинность одной или обеих сторон обмена — в этом случае в данной статье и будет использоваться данный термин. Может проверяться подлинность сообщений, когда к оригинальному сообщению добавляется код MAC (*message authentication code* — код аутентификации сообщения), в этом случае будет применяться термин целостность сообщения, чтобы не запутаться.

Цифровая подпись

Цифровая подпись — это, в принципе, всё тот же MAC — код, который подтверждает авторство данного сообщения и его целостность.

Например, если Боб решит передать открытое сообщение Алисе и подтвердить, что это именно его сообщение. Он шифрует сообщение с помощью закрытого ключа и добавляет полученный шифротекст к оригинальному сообщению. Теперь Алиса (и вообще кто-угодно) может расшифровать код с помощью публичного ключа Боба и убедиться, что полученный текст совпадает с оригиналом.

Основные используемые алгоритмы цифровой подписи на сегодняшний день — это RSA, DSA и ECDSA.

Криптографическая хеш-функция

Создавать цифровую подпись или MAC-код размером равными самому сообщению бывает непрактично. Поэтому подписью защищается не сам текст, а результат хеш-функции от текста. Криптографическая хеш-функция — это необратимое преобразование входных данных по определённому алгоритму в битовую строку определённой длины. Криптографические хеш-функции должны быть стойки к коллизиям, т.е. создаются таким образом, чтобы невозможно было создать вычислительный алгоритм, позволяющий подобрать два сообщения с одинаковым значением хеша или подобрать текст, который имеет заданное значение хеша.

Ранее широко используемые алгоритмы хеш-функции MD5 и SHA-1 теперь не рекомендуются. Предпочтение отдаётся SHA-2, который включает вариации SHA-224, SHA-256, SHA-384 и SHA-512 (по длине ключа). Также недавно был выбран претендент на звание SHA-3, которым стала хеш-функция Keccak. Скоро ожидается завершение процедуры его стандартизации.

Сертификат, центры сертификации и инфраструктура публичных ключей (PKI)

Сертификат — это стандартизированный формат, используемый для обмена публичными ключами, защищённый цифровой подписью. По сути, это структура данных, которая описывает: чей это публичный ключ, для чего может использоваться этот ключ, когда начал действовать этот публичный ключ и когда он устареет и т.д. Сертификат обязательно подписан цифровой подписью, чтобы исключить его подделку. Если сертификат подписан

с использованием своего же секретного ключа, то такой сертификат называют самоподписанным, и его часто используют в тестовых целях. Но важно понимать, что подделать самоподписанный сертификат не составляет никакого труда, поэтому самоподписанный сертификат бесполезен для любых практических задач.

Чтобы решить проблему с доверием к сертификатам, была создана целая инфраструктура по поддержке обмена публичными ключами, так называемые центры сертификации. Центр сертификации создаёт свою пару ключей (публичный и приватный), пользователи получают публичный ключ центра сертификации и считают его доверенным. Например, во многих дистрибутивах Linux есть пакет `ca-certificates`, который содержит сертификаты с публичными ключами всех доверенных центров сертификации. Каждый веб-браузер, как правило, также несёт с собой узелок с сертификатами, которым он доверяет.

Теперь если ваш сертификат подписан цифровой подписью центра сертификации, которому доверяет пользователь, то в этом случае пользователь может быть уверен в том, что вы тот, за кого себя выдаёте, т.к. это подтверждает центр сертификации.

Конечно, если центр сертификации потеряет свой закрытый ключ, то это моментально сделает все подписанные им сертификаты скомпрометированными. Чтобы минимизировать подобные риски, обычно строят иерархию сертификационных центров. Верхний уровень — это корневые сертификаты, их не так много. Они подписывают сертификаты сертификационных центров более нижнего уровня, которые затем также могут сертифицировать другие центры или конечные сертификаты пользователей. Таким образом образуются цепочки доверия.

Например, цепочка доверия для сайта google.ru:

```
Equifax Secure Certificate Authority
|
++ GeoTrust Global CA
 |
 ++ Google Internet Authority G2
  |
  + *.google.com.ru
```

Важно, чтобы при проверке сертификата проверялся каждый сертификат цепочки. Если хотя бы один из сертификатов подписан недоверенным цен-

тром, то доверять сертификату нельзя. Это, кстати, также важно знать и владельцам сертификатов: если ваш сертификат заверен каким-то новым центром сертификации, есть вероятность, что сертификата этого центра ещё нет у браузера пользователя. Поэтому стоит передавать клиенту связку сертификатов промежуточных сертификационных центров (как правило, просто объединяют файлы сертификатов в порядке цепочки доверия в один файл).

Основными характеристиками криптостойкости сертификата являются выбранный алгоритм и длина публичного ключа, а также криптографическая функция, которая была выбрана для цифровой подписи сертификата. Можно выбрать хороший ключ RSA-2048, но если цифровая подпись сертификата выполнена с применением алгоритма SHA-1, то сертификат, по современным меркам, слабо защищён. Google и Microsoft заявили, что с 2016 года сайты с сертификатами SHA-1 будут считаться невалидными, а большинство центров сертификации начали бесплатный перевыпуск существующих сертификатов с использованием SHA-2.

Отзыв сертификата, CRL

Сертификаты могут отзываться в случае, если произойдёт, например, компрометация закрытого ключа. Для этих целей создаётся файл в формате CRL, в который добавляются серийные номера отозванных сертификатов. Файл подписывается цифровой подписью центра сертификации.

Поскольку сертификат может быть отозван в любой момент, следует проверять не только корректность сертификата, но и его отсутствие в CRL-списках.

OCSP, OCSP stapling

Чтобы решить проблему с огромными CRL-списками, которые надо каким-то образом регулярно обновлять, был создан протокол OCSP (Online Certificate Status Protocol — протокол онлайн запросов статуса сертификата). Теперь Алисе достаточно выполнить запрос с серийным номером сертификата Боба к нужному центру сертификации, чтобы получить информацию о статусе сертификата. OCSP-ответы всегда подписаны цифровой подписью для защиты от подделки.

OCSP-запрос вносит дополнительную задержку в процедуре обмена, поэтому для оптимизации может применяться техника OCSP stapling, когда вместе со своим сертификатом Боб может отправить прикреплённый (stapled) ответ OCSP-сервера. Такой ответ всегда содержит время, когда был сделан запрос и время истечения валидности ответа и заверен цифровой подписью центра сертификации, поэтому ему можно доверять.

Протокол TLS

После того, как базовые понятия определены, можно приступить к рассмотрению работы протокола TLS. Протокол TLS традиционно относят к сессионному уровню в сетевой модели OSI, между транспортным уровнем (tcp) и уровнем приложения (http, smtp, imap, ...). Действительно, до начала передачи данных уровня приложения стороны должны согласовать параметры протокола: используемая версия, шифр, сжатие и т.д. После выполнения согласования (TLS Handshake) данные приложения передаются в рамках определённых правил — TLS-сессии.

Существуют две процедуры согласования: полная и сокращённая.

Полная процедура согласования

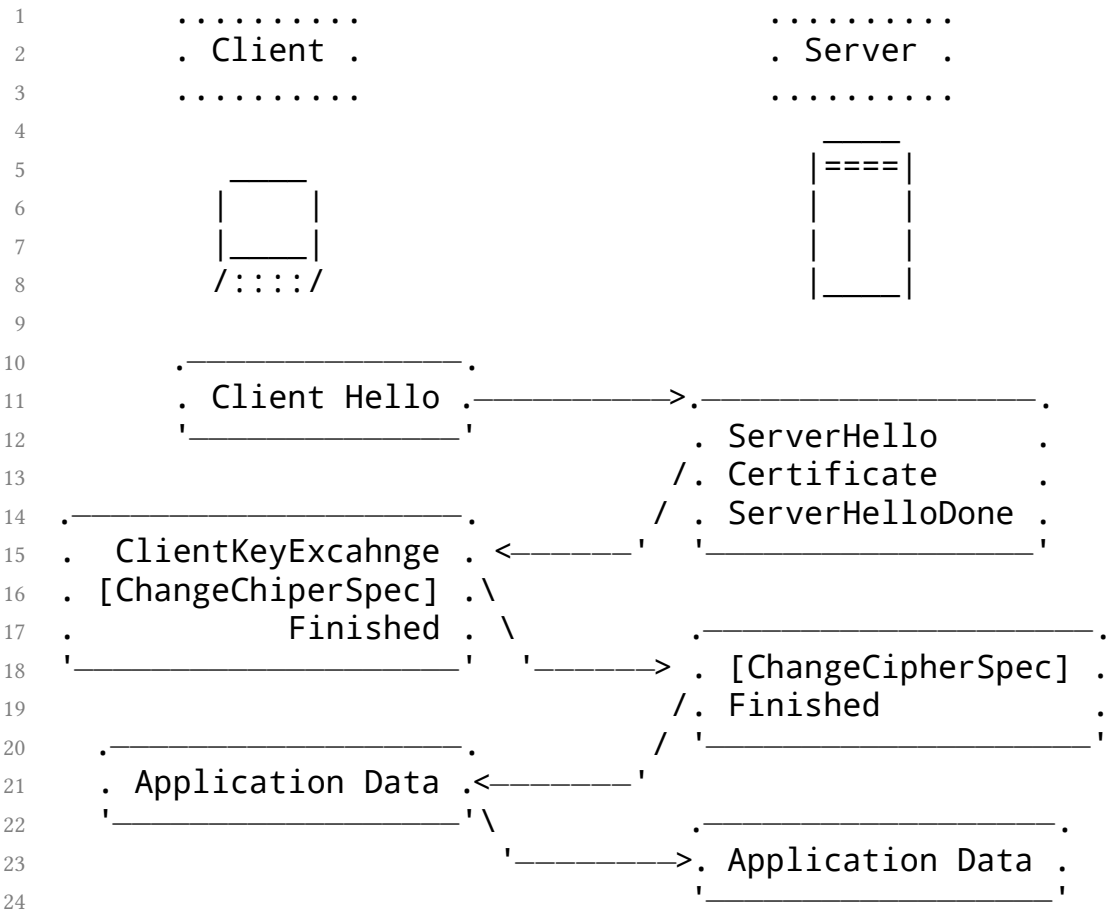
При полной процедуре клиент отправляет сообщение `ClientHello`, в котором сообщает серверу, какую версию протокола он предпочитает, список поддерживаемых шифров, поддерживаемые методы сжатия и список поддерживаемых расширений протокола. Сервер на это должен ответить своим `ServerHello`, в котором уведомляет клиента о том, какую он выбрал версию протокола, шифр, метод сжатия и расширения из того, что прислал клиент. Далее он должен отправить свой сертификат `Certificate` и сигнализировать об окончании передачи с помощью `ServerHelloDone`.

Теперь клиент проверяет сертификат сервера (можно ли ему доверять, подписан ли он доверенным центром), если используется алгоритм RSA, то клиент извлекает из сертификата публичный ключ и с его помощью шифрует некоторый случайный секрет. Этот случайный секрет затем используется для создания ключей шифрования и контроля целостности для данной сессии.

После чего клиент отправляет этот зашифрованный секрет серверу в сообщении `ClientKeyExchange`. Далее с помощью сообщения `ChangeChiperSpec` сигнализирует серверу, что он начал применять новые ключи шифрования и контроля целостности, и самое первое зашифрованное сообщение `Finished` содержит в себе хеш от всех предыдущих сообщений согласования в данной сессии.

Сервер, получив от клиента зашифрованный секрет, расшифровывает своим приватным ключом, генерирует по такому же алгоритму ключи шифрования и проверки подлинности. Теперь сервер может расшифровать сообщение `Finished` от клиента, в котором он проверяет хеш с тем, что получилось у него. Если всё совпало, то сервер сигнализирует, что новые шифры действуют и в направлении сервер — клиент: шлёт `ChangeCipherSpec` и свой вариант `Finished` с хешем.

Клиент расшифровывает и проверят хеш от сервера — если всё совпало, то начинают передаваться данные приложения, зашифрованные и подписанные сгенерированными при согласовании ключами.



Как видно, протокол обеспечивает три вида защиты:

1. Аутентичность. Сервер отправляет сертификат в формате X.509, который клиент проверяет на доверие. Зашифровав некий секретный ключ, клиент может быть уверен, что расшифровать его сможет только легитимный сервер, имеющий закрытый ключ сертификата.

2. Конфиденциальность. После выработки общего секрета по известному алгоритму формируются ключи для шифрования. Алгоритм шифрования выбирается из того, что согласовали клиент и сервер. Это может быть потоковый шифр RC4 или блочный 3DES или AES. Блочные шифры как правило используют режим CBC — блочное сцепление, когда следующий блок XOR'ится с предыдущим зашифрованным блоком, чтобы повысить стойкость шифрования. Данные приложения передаются зашифрованными, и нет никакой возможности третьей стороне их расшифровать.

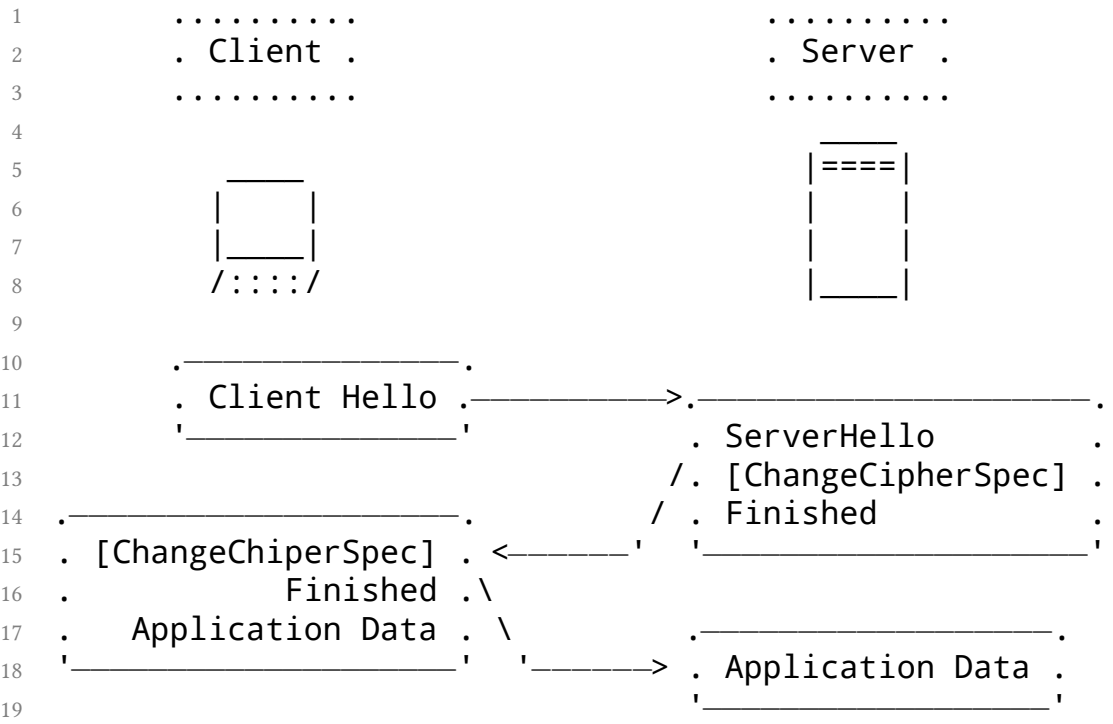
3. Целостность. Клиент и сервер согласуют функцию для хеширования данных. Это могут быть функции HMAC_SHA или HMAC_MD5, когда генерируется хеш от защищаемого текста и некой секретной соли. Таким образом гарантируется, что сообщение не было искажено при передаче. Кроме того, защищена от подмены и начальная стадия согласования (когда ещё не применяется шифрование) благодаря тому, что в сообщении `Finished` стороны обмениваются хешем всех предыдущих сообщений, чтобы убедиться, что каждая сторона получила одинаковые данные.

Здесь видна и основная проблема полной процедуры согласования: требуется два полных цикла отправки-приёма. Таким образом, протокол добавляет задержку при подключении, равную двум RTT.

Сокращённая процедура согласования

Чтобы решить проблему с задержкой, была создана сокращённая процедура согласования. При первом соединении клиента и сервера происходит обычная полная процедура, при этом сервер в сообщении `ServerHello` сообщает клиенту уникальный номер сессии и сохраняет контекст (главный секрет,

версия протокола, шифр и т.д.). Клиент может запомнить номер сессии и связанный контекст и при следующем подключении в `ClientHello` сразу передать номер сессии. Если сервер сохранил данные об этой сессии, то он уведомляет об этом клиента в `ServerHello` и сразу же применяет этот контекст, отправляя `ChangeCipherSpec` и `Finished` клиенту. Клиент также применяет контекст шифрования и отправляет свои `ChangeCipherSpec` и `Finished`. Сразу же могут начать передаваться данные приложения.



Как видно, в сокращённой схеме согласования дополнительная задержка составляет всего один RTT.

С другой стороны, это накладывает требования к серверу для хранения контекста сессий для каждого подключаемого клиента. К счастью, на этот случай было разработано специальное расширение для протокола Session Ticket (RFC5077). Сервер отправляет клиенту не только номер сессии, но и билет, в котором только серверу известным способом зашифрованы параметры сессии. Если клиент хочет начать сокращённую процедуру согласования, то с `ClientHello` отправляется билет, который сервер расшифровывает и использует. Это аналог зашифрованных cookie для хранения данных сессии в HTTP-протоколе.

Совершенная прямая секретность (Perfect forward secrecy, PFS)

Представим себе страшное происшествие: хакеры выкрали секретный RSA-ключ сервера, и теперь, имея записи шифрованного трафика и приватный ключ сервера, они смогут выполнить расшифровку всех предшествующих сессий пользователей сервера. Для того, чтобы исключить подобный сценарий, были разработаны алгоритмы PFS. Суть всех подобных алгоритмов в том, что начальный секретный ключ не передаётся от клиента к серверу, а формируются клиентом и сервером совместно.

Алгоритм Диффи-Хеллмана

Один из самых первых и популярных алгоритмов, обеспечивающих обмен ключами по незащищённым каналам, стал алгоритм Диффи-Хеллмана. Вкратце суть алгоритма такова:

Алиса и Боб выбирают числа `g` и `p` (они публичны). Затем Алиса формирует секретное число `a`, а Боб число `b`. Затем они обмениваются числами `A` и `B` — результатами следующих вычислений:

```
A = g^a mod p
B = g^b mod p
```

Теперь если Алиса возведёт число Боба `B` в степень `a` и найдёт остаток от деления на `p`, то получит число `K`:

```
B^a mod p = (g^b mod p)^a mod p = g^(b*a) mod p = K
```

Нетрудно заметить, что если Боб возьмёт число Алисы `A`, возведёт его в степень `b` и вычислит остаток от деления на `p`, то получит тоже самое число `K`:

```
A^b mod p = (g^a mod p)^b mod p = g^(a*b) mod p = K
```

Это и будет общим секретным ключом. При этом Ева, даже имея возможность наблюдать за обменом Алисы и Боба, не сможет рассчитать число `K`, поскольку перед ней возникает задача вычисления дискретного логарифма, и в зависимости от выбранных чисел `p`, `a`, `b` это может представлять собой неразрешимую вычислительную задачу. На практике, в качестве `g` обычно

берут число 2, поскольку это основание степени удобно для вычислений на компьютерах, ну а число p должно быть простым, и его размер является определяющим для стойкости алгоритма, как правило минимальное приемлемое значение имеет длину 1024 бит.

Алгоритм Диффи-Хеллмана на эллиптических кривых

Алгоритм Диффи-Хеллмана также замечательно может применяться на эллиптических кривых. Не вдаваясь в довольно сложную теорию эллиптических кривых, рассмотрим базовые принципы. Предположим у нас есть эллиптическая кривая

```
y^2=x^3+ax+b
```

Мы определяем поле как конечный набор точек, удовлетворяющих уравнению кривой на эллиптической кривой по модулю p, где p — это простое число:

```
y^2=x^3+ax+b (mod p)
```

На эллиптической кривой определены операции сложения и умножения точек.

Таким образом, Алиса передаёт Бобу известные параметры: выбранное уравнение эллиптической кривой, число p и некоторую базовую координатную точку G на кривой. Алиса выбирает некоторое случайное целое число (не превышающее число элементов в поле) da и передаёт результат умножения:

```
A=da*G
```

Боб выполняет ту же операцию:

```
B=db*G
```

Теперь общий секрет вычисляется как

```
B*da = db*G*da = K
A*db = da*G*db = K
```

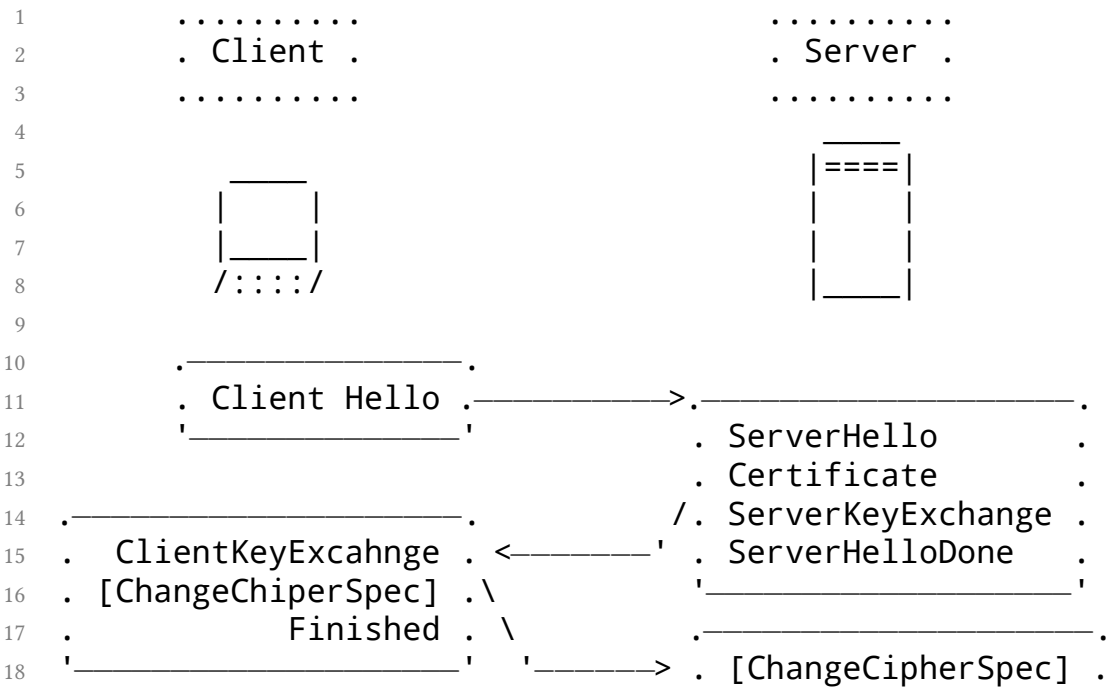
Существуют множество вариантов выбора эллиптических кривых; помимо описанного поля нечётной характеристики используются также поля характеристики 2. Но смысл алгоритма остаётся прежним, и криптографическая

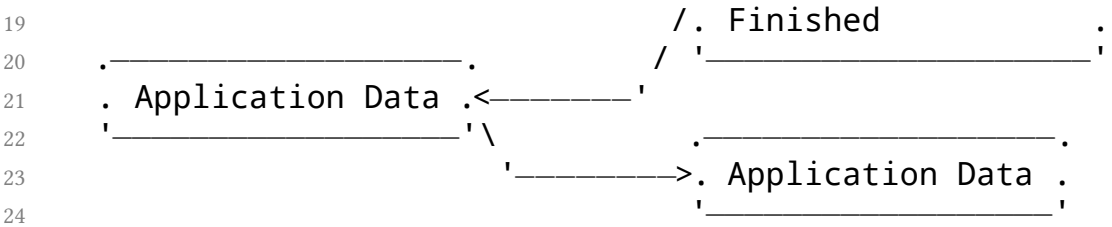
стойкость достигается за счёт вычислительной сложности дискретного логарифма на эллиптической кривой.

Основное достоинство эллиптических кривых — это значительно меньше требования к размеру ключа. Например, 2048-битный ключ RSA соответствует по стойкости ключу в 224 бита для эллиптических кривых. Кроме того, некоторые подобранные кривые требует меньше вычислительных мощностей по сравнению с классическими RSA/DSA. Например, nistp224 (ECDSA 224 бита) выдаёт порядка 10000 операций шифр/сек против 500 у RSA-2048 на современном процессоре.

Диффи-Хеллман в TLS

Различные модификация алгоритма Диффи-Хеллмана получили соответствующие обозначения DHE и ECDHE. Использовать их самостоятельно нельзя, поскольку они уязвимы к атакам MITM. Поэтому их используют совместно с RSA/DSA/ECDSA. В этом случае схема согласования немного изменена. После отправки сертификата сервер также отправляет сообщение `ServerKeyExchange`, в котором передаёт параметры Диффи-Хеллмана и подписывает их с помощью цифровой подписи RSA/DSA/ECDSA. Т.е. клиент обязательно проверяет валидность цифровой подписи по присланному сертификату сервера, убеждаясь, что он получил корректные параметры, и они не были подделаны по пути к нему:





Таким образом, секретный ключ, который будет использоваться в TLS-сессии, всегда будет различный, и его невозможно будет восстановить, даже имея на руках приватный ключ сертификата сервера, поскольку сервер не сохраняет параметры DH сессии (отсюда и буква E в названиях DHE/ECDHE — эфемерные).

Шифры

Как уже было выяснено, безопасность протокола TLS базируется на четырёх компонентах: аутентичность, конфиденциальность, целостность и совершенная прямая секретность. Каждый из этих компонентов может обеспечиваться различными криптографическими алгоритмами, например, шифрование данных может выполняться с помощью алгоритма AES, а целостность — алгоритмом SHA. Таким образом, шифр, используемый в TLS-сессии, в свою очередь задан из четырёх компонент. Один или несколько компонентов могут быть пустыми, т.е. соответствующий алгоритм не используется, но это снижает (и зачастую делает бесполезной) защищённость.

Например, рассмотрим название шифра, определяемого IANA как TLS_DHE_RSA_WITH_AES_128_CBC_SHA256. Как видно, название шифра включает в себя комбинацию из четырёх алгоритмов, каждый из которых используется для своих задач. Фрагмент `TLS` обозначает, что это шифр протокола TLS, далее строка `DHE` указывает на алгоритм PFS, `RSA` — алгоритм публичного ключа, используемый в сертификате сервера. После секции `WITH` следуют название алгоритма блочного шифрования: `AES`, указывается также длина ключа — 128 бит, подстрока `CBC` уточняет режим шифрования — сцепление блоков шифротекста для повышения стойкости ко взлому. Последним следует алгоритм криптографической хеш-функции SHA256.

Комбинаций из четырёх компонент можно получить достаточно много, поэтому существующих шифров достаточно много (судя по кодированию в

uint16 — потолок составляет 65536 вариаций). Но к выбору шифра стоит подходить со вниманием, т.к. существуют подобные шифры:

- `TLS_NULL_WITH_NULL_NULL` — никакой защиты нет;
- `TLS_RSA_WITH_NULL_SHA` — есть аутентификация и проверка целостности, но контент открыт и не шифруется;
- `TLS_ECDH_anon_WITH_AES_128_CBC_SHA` — отсутствует аутентификация, шифр уязвим к атаке MITM.

Поэтому основные рекомендацию по выбору шифрования на сегодняшний день такие:

- Использовать только протокол TLS, SSL должен быть отключён.
- Для обеспечения секретности рекомендуется использование ECDHE (с меньшим приоритетом — DHE).
- Шифрование — использование AEAD шифров, т.е. шифров, обеспечивающих одновременно и целостность, и секретность, например AES с длиной ключа 128 бит и режимом GCM: `AES_128_GCM_SHA256`. Ни при каких обстоятельствах не использовать NULL, RC4, DES.
- Аутентификация — использование сертификатов RSA с ключом 2048 бит с алгоритмом хеша SHA256. Использование цифровой подписи ECDSA пока ещё не очень популярно. В случае клиента — всегда обязательно проверять сертификат на валидность.
- Возобновляемые сессии удобны для сокращения задержки, но для улучшения секретности рекомендуется устанавливать небольшое время жизни для сессий (несколько минут).

Расширения TLS-протокола

В протоколе TLS предусмотрена возможность для создания и использования расширений, которые могут так или иначе влиять как на саму процедуру согласования, так и на передачу данных в рамках TLS-сессии. Выбор используемых расширений определяется клиентом, который в `ClientHello` передаёт список поддерживаемых расширений. Это касается даже серверных расширений, чтобы дать понять серверу, какие расширения поддерживаются кли-

ентом. Сервер может начать использовать только те расширения, которые поддерживает клиент. Рассмотрим некоторые популярные расширения.

Сессионные билеты

О сессионных билетах уже было упомянуто в главе о сокращённой процедуре согласования. Сервер может сохранить зашифрованные параметры сессии на стороне клиента, чтобы не хранить их у себя. Когда клиент подключается повторно, он прикладывает сессионный билет в `ClientHello` и, если расшифровка билета пройдёт успешно, сервер может начать сокращённую процедуру согласования.

Указание имени сервера (SNI)

Server Name Indication — указание имени сервера определено в RFC6066. Как известно, протокол HTTP 1.1 позволял клиенту путём указания HTTP-заголовка `Host` сообщить серверу, к какому именно сайту нужно выполнить подключение. Это позволило в условиях дефицита ipv4-адресов размещать на одном ip-адресе десятки и даже сотни виртуальных сайтов. Но с применением шифрования TLS такая возможность была потеряна, поскольку на этапе TLS-согласования сервер не знал сертификат какого виртуального сайта следует направить клиенту.

Расширение SNI даёт возможность клиенту в `ClientHello` обозначить, по какому доменному имени он хочет получить доступ. Таким образом, сервер на этапе согласования имеет возможность отправить нужный сертификат виртуального сайта.

Запрос статуса сертификата

Certificate Status Request или более известный как OCSP stapling, также уже был описан выше. Позволяет клиенту попросить сервер приложить ответ OCSP-сервера о статусе сертификата, чтобы сэкономить на времени одного запроса.

Поддержка шифров на эллиптических кривых (EC)

Действительно, протокол TLS изначально не содержал поддержки EC, поэтому, чтобы клиент и сервер могли использовать шифры с применением криптографии на эллиптических кривых, должно быть задействовано расширение, описанное в RFC4492.

ALPN и NPN

Application-Layer Protocol Negotiation — согласование протокола на уровне приложения. Расширение определено в RFC7301 и позволяет клиенту и серверу согласовать, какой протокол будет использоваться на уровне приложения. Например, клиент может отправить список протоколов уровня приложения в порядке приоритета, которые он поддерживает: “h2-16”, “http/1.1”. А сервер может выбрать из этого списка один и указать его в `ServerHello`. Данное расширение было специально создано для новой версии протокола http2, чтобы на переходном периоде клиент и сервер могли договориться, какую именно версию они хотят использовать. Данное расширение позволит отказаться от использования механизма http Upgrade.

Расширение ALPN было создано на основе TLS-расширения NPN, которое было разработано в Google для протокола, предшественника HTTP2 — SPDY.

Стоит отметить, что ALPN и NPN достаточно новые расширения. NPN появился в openssl 1.0.0d, а поддержка ALPN появится только в версии 1.0.2, которая ещё в статусе беты.

Heartbeat

Heartbeat — это расширение, опубликованное в RFC6520, реализующее функциональность keep-alive в TLS. Большую известность расширение получило благодаря багу Heartbleed в реализации openssl.

Использования сжатия в TLS

Протоколом TLS предусмотрена возможность сжатия данных. Алгоритм сжатия выбирается на этапе согласования, а затем применяется на данных до их шифрования. Данная возможность имеет больше недостатков, чем преимуществ. Поскольку сжатие производится независимо от типа контента, то это может быть неэффективно в случае сжатия медиаданных и приводит к бесполезному расходованию ресурсов процессора.

Но крест на использовании сжатия в TLS поставила успешная атака CRIME, которая позволяла путём отправки запросов и наблюдению за изменением длины передаваемого шифротекста определять части передаваемых данных, например, http-заголовка Cookie.

Таким образом, сжатие должно быть отключено, и не только для клиента, но и для сервера, чтобы исключить возможность применения подобных атак.

Обзор модулей для использования TLS в Perl

На сегодняшний день свободной, популярной и полной реализацией TLS-протокола и сопутствующих криптофункций является библиотека OpenSSL. Нельзя сказать, что это идеальная реализация: код безумно сложен, очень характерная история безопасности и неповоротливость развития. Но это лучше, чем ничего. Для работы с openssl в Perl разрабатывается и активно развивается низкоуровневая обертка — модуль Net::SSLeay.

Помимо `Net::SSLeay` существуют и другие обертки к OpenSSL, например `Crypt::SSLeay`. Но на сегодняшний день он не развивается и его использование не рекомендуется, поскольку в нём используется только протокол SSLv3. Также входящий в состав дистрибутива модуль `Net::SSL` не умеет проводить верификацию сертификата сервера.

Существуют также экзотические имплементации других SSL/TLS-библиотек: Crypt::NSS — обертка к крипто-библиотеке NSS, используемой в Firefox, Crypt::MatrixSSL — обертка к крипто-библиотеке MatrixSSL. Но их возможности достаточно ограниченные, на CPAN нет зависимых от них модулей, т.е. по всей видимости их никто не использует.

Таким образом, большинство модулей на CPAN, которые используют сокеты с использованием шифрования TLS, базируются на `Net::SSLeay` или производных от него модулях. Поскольку интерфейс библиотеки довольно низкоуровневый и тяжеловат для понимания, существует модуль-обёртка IO::Socket::SSL, который имеет более привычный интерфейс `IO::Socket::INET`. Кроме `IO::Socket::SSL` существуют модули `AnyEvent::TLS`, `IO::Async::SSL`, `POE::Filter::SSL` и т.п., которые применяются для работы с TLS-сокетами в соответствующих IO-фреймворках.

На практике, `Net::SSLeay` напрямую используется не часто. Детальному разбору API модуля можно посвятить отдельную статью. Гораздо интереснее с практической точки зрения модули, построенные на его основе.

IO::Socket::SSL

`IO::Socket::SSL` — это удобный и отличный выбор для применения. Основное его достоинство конечно же в простом интерфейсе, который наследуется от знакомого и привычного `IO::Socket::INET`. Второе достоинство — активная поддержка и развитие.

Клиент Рассмотрим пример тривиального https-клиента:

```
1 my $cl = IO::Socket::SSL->new('www.google.com:443')
2     or die "error=$!, ssl_error=$SSL_ERROR";
3 print $cl "GET / HTTP/1.0\r\n\r\n";
4 print <$cl>;
```

В зависимости от версий `IO::Socket::SSL`/`Net::SSLeay` этот код может вести себя по-разному. Начиная с версии *1.950* `IO::Socket::SSL` стал проверять сертификат сервера по умолчанию. В версии *1.971* также начинает использоваться TLS-расширение SNI для того, чтобы передать требуемое имя сервера. В версии *1.984* появилась поддержка OCSP stapling, по умолчанию анонсируется поддержка расширения и запрашивается статус сертификата. В версии *2.000* исключается использование версии протокола SSLv3 и SSLv2. Ну и наконец в версии *2.001* по умолчанию включается поддержка ECDHE/DHE для активации PFS.

Конструктор с опциями по умолчанию для последней версии модуля выглядит так:

```
my $cl = IO::Socket::SSL->new(
    'www.google.com:443',

    # Список шифров в порядке предпочтения
    # Шифры с предшествующим знаком '!' ––– запрещены
    SSL_cipher_list => '
        ECDHE-ECDSA-AES128-GCM-SHA256
        ECDHE-ECDSA-AES128-SHA256
        ECDHE-ECDSA-AES256-GCM-SHA384
        ECDHE-ECDSA-AES256-SHA384
        ECDHE-ECDSA-AES128-SHA
        ECDHE-ECDSA-AES256-SHA
        ECDHE-RSA-AES128-SHA256
        ECDHE-RSA-AES128-SHA
        ECDHE-RSA-AES256-SHA
        DHE-DSS-AES128-SHA256
        DHE-DSS-AES128-SHA
        DHE-DSS-AES256-SHA256
        DHE-DSS-AES256-SHA
        AES128-SHA256
        AES128-SHA
        AES256-SHA256
        AES256-SHA
        EDH-DSS-DES-CBC3-SHA
        DES-CBC3-SHA
        RC4-SHA
        !EXP !LOW !eNULL !aNULL !DES !MD5 !PSK !SRP',

    # Полностью исключены SSLv3 и SSLv2
    SSL_version => 'SSLv23:!SSLv3:!SSLv2',

    # Обязательная проверка сертификата сервера
    SSL_verify_mode => SSL_VERIFY_PEER,

    # SNI имя сервера
    SSL_hostname => 'www.google.com',

    # запрос статуса сертификата
    SSL_ocsp_mode => SSL_OCSP_TRY_STAPLE
)
```

Таким образом, рекомендуемая для работы клиента версия `IO::Socket::SSL` должна быть не меньше *2.001*. Конечно, такая политика изменения опций по умолчанию ломает обратную совместимость, зато обеспечивает высокий уро-

вень безопасности. Поэтому рекомендуется без необходимости не указывать явно параметры, связанные с безопасностью, например, список шифров, т.к. в будущем эти параметры могут оказаться слабыми, а параметры по умолчанию всегда будут изменяться в пользу большей защищённости.

Сервер Рассмотрим пример простого TLS-сервера.

```
my $srv = IO::Socket::SSL->new(
    LocalAddr     => '0.0.0.0:1234',
    Listen        => 10,
    SSL_cert_file => 'server-cert.pem',
    SSL_key_file  => 'server-key.pem',
);
$srv->accept;
```

TLS-сервер должен обязательно иметь сертификат и соответствующий приватный ключ. Для включения PFS `IO::Socket::SSL` активирует параметры DHE и ECDHE в `Net::SSLeay`. Кроме того, при выборе шифра сервер будут ориентироваться не по приоритетам, заданным клиентом, а по собственным приоритетам. Таким образом, по умолчанию добавляются следующие опции:

```
my $srv = IO::Socket::SSL->new(
    ...,

    # Группы шифров в порядке предпочтения
    # Шифры с предшествующим знаком '!' --- запрещены
    SSL_cipher_list => '
        EECDH+AESGCM+ECDSA EECDH+AESGCM
        EECDH+ECDSA +AES256 EECDH
        EDH+AESGCM EDH ALL +SHA +3DES +RC4
        !LOW !EXP !eNULL !aNULL !DES !MD5 !PSK !SRP',

    # Сервер использует свои приоритеты при выборе шифра
    SSL_honor_cipher_order => 1,

    # Заранее сгенированные параметры Диффи-Хеллмана
    # длиной 2048 бит
    SSL_dh => "...",

    # Задаются параметры эллиптической кривой для ECDHE
    # длина 256 бит
    SSL_ecdh_curve => 'prime256v1',
);
```

В случае, если сервер должен поддерживать несколько виртуальных хостов (т.е. выбирать сертификат по имени хоста в случае использования SNI), используется следующий конструктор:

```
my $srv = IO::Socket::SSL->new(
    LocalAddr     => '0.0.0.0:1234',
    Listen        => 10,
    SSL_cert_file => {
        'www1.example.com' => 'server-cert-www1.pem',
        'www2.example.com' => 'server-cert-www2.pem',
        ''                 => 'default-cert.pem',
    },
    SSL_key_file => {
        'www1.example.com' => 'server-key-www1.pem',
        'www2.example.com' => 'server-key-www2.pem',
        ''                 => 'default-key.pem',
    },
);
```

Для каждого хоста задаётся своя пара сертификатов/ключей, а также указывается хост по-умолчанию в случае, если клиент не поддерживает SNI или указывает имя, отсутствующее в списке.

Инъекция параметров IO::Socket::SSL Возможно вы никогда не используете `IO::Socket::SSL` напрямую, но вам приходится сталкиваться с модулями, которые используют этот модуль. Если данные модули не предоставляют возможности изменить параметры подключения, но при этом вам требуется их изменить, то для этих случаев создана специальная функция `set_args_filter_hack`.

Рассмотрим, для примера, модуль `Mail::POP3Client`, который служит для подключения к POP3-серверу, в том числе и с использованием TLS-подключения. Если POP3-сервер находится в локальной сети и использует сертификат, подписанный неизвестным центром CA или вообще самоподписанный сертификат, то при использовании последней версии `IO::Socket::SSL` модуль больше не сможет осуществлять подключение. Как один из вариантов решения, можно повлиять на параметры создаваемого сокета с помощью `set_args_filter_hack`:

```
use Mail::POP3Client;
use IO::Socket::SSL;

# установка хака
```

```
IO::Socket::SSL::set_args_filter_hack( sub {
    my ($is_server, $args) = @_;

    # Явное указание отпечатка сертификата сервера
    $args->{SSL_fingerprint} = 'SHA1$BE:EF:CA:FE:CO:DE:...';
} );

my $pop = Mail::POP3Client->new(
    HOST     => "pop3.server.local",
    USER     => "user",
    PASSWORD => "password",
    USESSL   => 1,
);

printf "Count: %d\n", $pop->Count();
```

В данном примере мы задаём функцию, которая будет вызываться при создании сокета. В этой функции имеется возможность переопределить аргументы, которые были переданы в конструктор `IO::Socket::SSL`. Здесь мы указываем ожидаемый отпечаток сертификата в опции `SSL_fingerprint`. Отпечаток сертификата можно получить, например, с помощью команды openssl:

```
$ openssl x509 -noout -fingerprint < pop3.server.cert

SHA1 Fingerprint=BE:EF:CA:FE:CO:DE:BE:EF:CA:FE:CO:DE:BE:EF:CA:FE
```

Это лишь один из вариантов, который можно использовать для самоподписанных сертификатов. Если сертификат заверен неизвестным центром сертификации, можно указать путь к файлу сертификата этого центра в опции `SSL_ca_file`.

Таким образом можно переопределить и другие опции, которые по тем или иным причинам вас не устраивают. Функция `set_args_filter_hack` появилась в версии `IO::Socket::SSL` 1.969.

AnyEvent::TLS

Модуль `AnyEvent::TLS` может неявно использоваться в других модулях пространства имён `AnyEvent`, если у сокета указываются параметры TLS.

Клиент Например, при создании объекта `AnyEvent::Handle` клиента TLS:

```
my $h = AnyEvent::Handle->new(
    fh  => $fh,
    tls => 'connect',
);
```

В этом случае автоматически создаётся объект `AnyEvent::TLS`, который загружает и инициирует модуль `Net::SSLeay`. К сожалению, автор `AnyEvent::TLS` не так активно следит за прогрессом `Net::SSLeay` по сравнению с разработчиками `IO::Socket::SSL`, поэтому чтобы получить сравнимый уровень защищённости, необходимо задавать некоторые параметры. Например, для клиента:

```
my $h = AnyEvent::Handle->new(
    fh      => $fh,
    tls     => 'connect',
    tls_ctx => {

        # Отключаем поддержку SSLv3
        sslv3 => 0,

        # Включаем проверку сертификата сервера
        verify => 1,

        # Включаем поддержку сессионых билетов
        session_ticket => 1,
    }
);
```

К сожалению, ни OCSP stapling, ни SNI включить невозможно без переопределения метода `AnyEvent::TLS::_get_session`.

Сервер В случае TLS-сервера создаётся такой конструктор:

```
use AnyEvent;
use Net::SSLeay;

my $h = AnyEvent::Handle->new(
    fh      => $fh,
    tls     => 'accept',
    tls_ctx => {

        cert_file => 'server-cert.pem',
        key_file  => 'server-key.pem',
```

```
11
12          # Отключаем поддержку SSLv3
13          sslv3 => 0,
14
15          # Включаем поддержку сессионых билетов
16          session_ticket => 1,
17
18          # Функция вызываемая после создания
19          # объекта AnyEvent::TLS
20          prepare => sub {
21              my $tls = shift;
22
23              # Контекст Net::SSLeay
24              my $ctx = $tls->ctx;
25
26              # Инициализация ECDHE
27              # Требуется Net-SSLeay >= 1.56 и openssl >= 1.0.0
28              if ( exists &Net::SSLeay::CTX_set_tmp_ecdh ) {
29                  my $curve = Net::SSLeay::OBJ_txt2nid('prime256v1'
                        );
30                  my $ecdh  = Net::SSLeay::EC_KEY_new_by_curve_name
                        ($curve);
31                  Net::SSLeay::CTX_set_tmp_ecdh( $tls->ctx, $ecdh )
                        ;
32                  Net::SSLeay::EC_KEY_free($ecdh);
33              }
34          }
35      }
36  );
```

В функции-колбеке `prepare` можно получить контекст `Net::SSLeay` и произвести с ним некоторые необходимые манипуляции. Например, установить параметры для алгоритма Диффи-Хеллмана на эллиптических кривых. Как мы уже знаем, это позволит использовать шифры с эллиптическими кривыми, которые снижают процессорную нагрузку и обеспечивают лучшую прямую секретность.

По умолчанию `AnyEvent::TLS` использует заранее сгенерированные параметры обычного DH длиной 1539 бит. Если требуется более надёжное значение, сгенерируйте параметры DH с помощью openssl:

```
1 $ openssl dhparam -out dh-2048.pem 2048
```

Или выберите заранее сгенированные значения в `AnyEvent::TLS` нужной длины:

```
tls_ctx => {
    ...
    # Путь к файлу с DH
    dh_file => 'dh-2048.pem',

    # Или встроенные значения со странными названиями
    # skip512, skip1024, skip2048, skip4096
    # schmorp1024, schmorp1539, schmorp2048, schmorp4096,
       schmorp8192

    dh => 'schmorp2048',
}
```

Настроить поддержку SNI в AnyEvent простыми методами пока затруднительно.

Заключение

Надеюсь, что эта сжатая статья поможет заполнить пробелы в знаниях о защищённом протоколе TLS и тех принципах безопасности, на которых он основан. Для подробного изучения темы рекомендую следующие источники:

- Криптография:
- «Практическая криптография» Нильс Фергюсон, Брюс Шнайер
- «Прикладная криптография» Брюс Шнайер
- TLS:
- «RFC 5246»
- «High Performance Browser Networking»
- «SSL/TLS Deployment Best Practices»
- Насколько быстр TLS
- Тестирование SSL/TLS-серверов

P.S. Protocol::TLS

В качестве Post Scriptum информация к размышлению о небольшом, но амбициозном проекте Protocol::TLS.

Рассмотрев работу протокола TLS, можно сделать достаточно простой вывод: логика протокола (машина состояний/конечный автомат) отделена от криптографических функций. Именно поэтому и возникла идея создания модуля `Protocol::TLS`, который должен стать реализацией протокола TLS на чистом Perl. При этом реализация набора криптографических функций может выделена в отдельные плагины, которые могут быть основаны на различных криптобекендах в зависимости от требуемого функционала или каких-то других требований. Своеобразный аналог `DBI`, который подключает нужный драйвер СУБД при создании подключения, при этом предоставляя единый интерфейс приложениям.

На сегодняшний день `Protocol::TLS` частично реализует RFC5246 (TLS v1.2), поддерживает несколько шифров, в том числе обязательный `TLS_RSA_WITH_AES_128_C`. В качестве пока единственного криптографического плагина используется модуль CryptX, который является самодостаточным модулем без внешних зависимостей, который несёт в себе C-библиотеку libtomcrypt. Данный криптографический бекенд содержит практически всю необходимую базу современных криптографических алгоритмов, включаю криптографию эллиптических кривых. Библиотека LibTomCrypt имеет двойную лицензию: общественное достояние (*Public Domain*) и *DO WHAT THE FUCK YOU WANT TO PUBLIC LICENSE* (делайте, что вам вздумается). Модули `CryptX` и `Protocol::TLS` имеют обычную Perl-лицензию (Artistic || GPL). Поэтому с точки зрения лицензирования такой модуль имеет преимущество по сравнению с OpenSSL, лицензия которого несовместима с GPL.

Также, в качестве эксперимента, создан модуль IO::Socket::TLS, который эмулирует API `IO::Socket::SSL`, но при этом под капотом использует модуль `Protocol::TLS`. Такая обёртка может быть удобна для тестирования существующих приложений, основанных на `IO::Socket::SSL`. Модуль пока не на CPAN из-за неполноты реализации и довольно наглого названия.

В планах реализация поддержки TLS 1.1/1.0, дополнительного криптобекенда на основе модулей на чистом Perl, чтобы обеспечить возможность fatpack-упаковки модуля и всех его зависимостей.

■ *Владимир Леттиев*

7. Обзор CPAN за декабрь 2014 г.

Рубрика с обзором интересных новинок CPAN за прошедший месяц

Статистика

- Новых дистрибутивов — 296
- Новых выпусков — 909

Новые модули

Mojo::JSON_XS

Модуль `Mojo::JSON_XS` позволяет использовать в Mojolicious более быструю XS-реализацию JSON-кодировщика, основанного на модуле `Cpanel::JSON::XS`. Модуль должен быть загружен раньше, чем будет загружен `Mojo::JSON` и использованы какие-либо из его функций:

```
use Mojo::JSON_XS;
use Mojo::JSON qw(to_json from_json);
```

Следует, однако, отметить, что `Cpanel::JSON::XS` и `Mojo::JSON` в некоторых случаях могут производить различный вывод. Примеры таких расхождений перечислены в документации модуля.

App::Prove::Watch

Утилита `provewatcher` позволяет наблюдать за изменениями файлов в каталоге и, в случае появления таких изменений, запускать заданную команду, например `prove` или `dzil test`. К сожалению, утилита реагирует на изменения в любых файлах, например, в своп-файлах vim. Возможно, в более новых версиях появится файловый фильтр.

AnyEvent::GnuPG

`AnyEvent::GnuPG` — это обёртка к утилите gpg, позволяющая выполнять различные операции с шифрованием и цифровой подписью как синхронно, так и асинхронно. В настоящий момент модуль работает только с gnupg версий 1.x.

Alien::Web

`Alien::Web` — это модуль, который определяет пространство имён для распространения всевозможных JavaScript/CSS библиотек через CPAN, по аналогии с пространством имён `Alien`, которое используют для сборки/конфигурации, например, C/C++-библиотек. При инсталляции модуля, соответствующие файлы помещаются в share-каталог дистрибутива модуля (определяется как `File::ShareDir::dist_dir()`).

Это позволяет, например, следующим образом собирать веб-приложение:

```
use Plack::Builder;
use Plack::App::Directory;
use File::ShareDir;

builder {
    mount "/js/extjs" => Plack::App::Directory->new({
            root => dist_dir('Alien-Web-ExtJS-V3')
        })->to_app;
    ...
};
```

Thrust

`Thrust` — это байндинг к одноимённому кросс-платформенному фреймворку приложений, основанному на Chromium. Модуль `Thrust` позволяет создавать приложения с интерфейсом, созданным на основе привычных HTML/CSS/JavaScript. Своеобразный симбиоз браузера и веб-сервера в одном приложении.

Debug::Statements

Модуль `Debug::Statements` упрощает написание отладочных сообщений. Типичный отладочный код может выглядеть так:

```
my $DEBUG = 1;

print "\$x = '$x', \$array[2] = '$array[2]'\n" if $DEBUG;
```

Это выведет сообщение:

```
$x = 'some value', $array[2] = 'three'
```

С помощью `Debug::Statements` отладочный код можно существенно упростить:

```
use Debug::Statements;

# $d — задает уровень отладки
my $d = 1;

d  '$x $array[2]';  # Печатается при уровне отладки $d >= 1
d2 '@array';        # Печатается при уровне отладки $d >= 2
```

Отладочный вывод:

```
DEBUG:  $x = 'some value'
DEBUG:  $array[2] = 'three'
DEBUG2:  @array = [
  'one',
  'two',
  'three'
]
```

Как видно, печатается не только значение переменной, но и её название. Правда при использовании модуля `Debug::Statements` придётся пожертвовать переменной `$d`, которая задаёт уровень отладки (максимум три уровня).

Crypt::Rijndael::PP

`Crypt::Rijndael::PP` — это реализация криптографического алгоритма Rijndael (больше известного как AES) на чистом Perl. Реализация совместима

с Crypt::Rijndael и может использоваться в Crypt::CBC для 256-битных ключей.

SQL::Interpol

Выпущен форк модуля SQL::Interp. Модуль позволяет интерполировать Perl-переменные в SQL-выражения, например:

```
my %item = ( column1 => "value1", "column2" => "value2" );
my ($sql, @bind) = sql_interp 'INSERT INTO table', \%item;

#  $sql = "INSERT INTO table (column1, column2) VALUES(?, ?)"
# @bind = ( "value1", "value2" )
```

Выполнен рефакторинг кода парсера, удалён код DBIx::Interp, удалена поддержка строгого (strict) режима.

Обновлённые модули

perl 5.21.7

В декабре был выпущен очередной релиз Perl для разработчиков 5.21.7. Значимым этот релиз делают несколько оптимизаций, появившихся в данном релизе:

- Новая операция OP_MULTIDEREF, которая выполняет одно или несколько разыменований, если индексы/ключи являются константами или простыми скалярами, что позволяет сократить общее число операций. Например, получение значения $a[0]{$k}[$i] вместо четырёх операций теперь требует только одну.
- Ускорен вызов методов как с явным (Class->method()), так и динамическим именем (Class->$dynamic_method) в среднем на 30%. Вызов метода родительского класса (SUPER::method()) ускорен на 50%. Также ускорены вызовы вида $obj->Class::method(), поскольку теперь во время исполнения не требуется сканировать имя метода на наличие ::. Важно отметить, что автором и инициатором включения этих изменений

> в Perl стал Олег Пронин (*syberrus*) из компании Crazy Panda. Очень надеемся, что в будущих номерах журнала мы увидим его статью об этих изменениях, а также тех, которые ещё планируется сделать для Perl.

Помимо прочего, в perldelta вскользь упомянуто об обновлении модуля `PerlIO::scalar` с исправлением бага №123443. На самом деле это очень важное исправление, поскольку исправляет проблему безопасности при работе с данными в скалярах через файловые дескрипторы. Исправление касается операций seek и read. Применение операции seek для задания смещения в файловом дескрипторе к скаляру позволяло задать смещение далеко за пределами скаляра. А последующее чтение read с такой позиции могло приводить к краху приложения. Например:

```
my $buf = "hello";
open my $fh, '<', \$buf or die $!;
seek( $fh, 2**32, SEEK_SET );   # смещение на большую позицию
read( $fh, my $tmp, 1 );        # чтение 1 байта с этого смещения
```

Perl при чтении попытается расширить скаляр до нужного смещения, что приведёт к ошибке сегментации на Perl ≥ 5.16 и к попытке выделения огромного сегмента памяти на более ранних версиях. Также seek, хоть и с предупреждением, но может устанавливать отрицательное смещение. И если ваш код не проверяет ошибку `seek`, то последующее чтение позволяет читать произвольные участки памяти вашего процесса. Если в памяти процесса содержатся переменные с паролями, приватными ключами, то все они могут уплыть при чтении за пределами скаляра. Получили аналог Heartbleed для Perl.

Таким образом, до исправления бага во всех поддерживаемых стабильных релизах Perl лучше избегать использования `PerlIO::scalar`, воспользовавшись альтернативой: `IO::Scalar`. Если это невозможно, то необходимо обязательно самостоятельно проверять аргументы seek, чтобы смещение не выходило за пределы скаляра.

Email::MIME::Kit 3.000001

Вышел новый мажорный релиз генератора почтовых сообщений из шаблонов `Email::MIME::Kit`. Модуль не просто компонует сообщение из отдельных фрагментов текста, но и полностью отвечает за подготовку

multipart-сообщений, текстовой и html-альтернатив сообщения, кодирования, внутренних связей контента и проверки параметров. Новый релиз имеет важное несовместимое изменение: при чтении шаблонов с диска они декодируются в UTF-8 только если формат шаблона указан как текстовый (`text/*`).

Want 0.25

Новая версия модуля `Want`, который является развитием функции `wantarray`. Обновлённый модуль поддерживает новую операцию `OP_MULTIDEREF`, появившуюся в perl 5.21.7, позволяя корректно определять тип первой операции разыменования. Например:

```
sub foo {
    if (wantref eq 'ARRAY') {
        ...
    }
}

foo()->[1]{key}
```

Первая из двух операций разыменования – доступ к элементу массива с индексом 1, поэтому `wantref`, как и ожидается, вернёт ‘ARRAY’.

Class::Accessor::Inherited::XS 0.07

Продолжает активно развиваться модуль `Class::Accessor::Inherited::XS` для создания акцессоров/мутаторов, совместимых с `Class::Accessor::Grouped`. В представленных бенчмарках данный модуль работает на порядок быстрее `Class::Accessor::Grouped`.

XML::RSS 1.56

Новый релиз `XML::RSS` исправляет потенциальную проблему безопасности при обработке RSS-лент. Злоумышленник может создать RSS-файл, включив в него элемент с внешней сущностью. Если результат обработки может быть доступен, то злоумышленник может получить содержимое любого файла с

сервера. Например, получить содержимое `/etc/passwd` можно, указав такую сущность в XML-файле:

```
<!DOCTYPE title [ <!ELEMENT title ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
```

А затем вывести его в тело документа, сославшись на имя сущности:

```
<title>&xxe;</title>
```

Эксплойт и информация о других уязвимых продуктах доступны по ссылке. Важно отметить, что уязвимы и другие модули, которые используют `XML::RSS`, такие как `XML::Feed` и другие

PadWalker 2.0

Вышел новый мажорный релиз `PadWalker`, позволяющий инспектировать и даже менять лексические переменные внутри текущей области видимости. Новый релиз теперь полностью совместим с последними изменениями в blead perl.

Log::Any 1.02

Представлен первый мажорный релиз универсального API для логирования `Log::Any`. В новом релизе были объединены дистрибутивы `Log::Any` и `Log::Any::Adapter`, вычищены все зависимости, не входящих в базовую поставку Perl (по состоянию для Perl 5.8.1).

Gearman 1.12

После многолетнего перерыва обновлена Perl-реализация клиента `Gearman::Client`, рабочего `Gearman::Worker` и сервера задач `Gearman::Server`. Новый релиз содержит множество исправлений, касающихся процедуры соединения и таймаутов.

HTTP::Parser::XS 0.17

Выпущена новая версия самого быстрого парсера HTTP-заголовков `HTTP::Parser::XS`. В новом релизе обновлена входящая в состав модуля C-библиотека picohttpparser, которая также используется в высокопроизводительном HTTP-сервере H2O, превосходящим по скорости даже nginx.

PerlPowerTools 1.003

Обновлённый выпуск `PerlPowerTools` — набор классических UNIX-утилит, написанных на Perl. Проект был возрождён брайном ди фоем, один из выпусков был специально подготовлен для доклада на конференции Saint Perl 2014.

Image::Info 1.37

Обновлён модуль `Image::Info` для извлечения информации об изображении. В новом релизе исправлены проблемы при чтении битых TIFF-файлов. Также вместо использования встроенного в Perl средства для работы с файловым дескриптором к скалярной переменной `PerlIO::scalar` теперь используется `IO::Scalar`. Связано это с обнаруженным критическим багом в `PerlIO::scalar` < 0.21.

autodie 2.26

Обновлена прагма `autodie`, заменяющая системные функции аналогами, которые генерируют исключение в случае ошибки. В новой версии исключены функции `umask` и `fileno`, так как если они возвращают `undef`, это не является ошибкой. Также обновились зависимости модуля: вместо `base` теперь используется `parent`.

■ *Владимир Леттиев*

8. Интервью с Рикардо Сигнесом (Ricardo Signes)

Рикардо Сигнес (Ricardo Signes) — американский Perl-программист, в настоящее время pumpking, ответственный за выпуск релизов perl

Когда и как научился программировать?

В детстве у меня в семье был TI-99/4A, современник Commodore 64. Мне было четыре или пять, когда я научился писать программы на BASIC по отгадыванию чисел или по подсчету возраста, если ввести день рождения. В то время я только учился писать, и компьютер тоже был особым пишущим устройством, где пишешь и что-то происходит. Я до сих пор думаю о компьютерах в этом ключе.

Программировал лишь изредка, пока не закончил университет. Мне просто нравилось решать небольшие задачки. Я писал для BBS-серверов на Pascal и C, писал небольшие утилиты для фильтрации USENET и интернета. Оно работало, но кое-как. Я не изучал, как писать качественный профессиональный код, — если согласиться, что этим я занимаюсь сейчас — пока не закончил университет и не получил работу.

Какой редактор используешь?

Первым на памяти редактором был KEDIT, клон XEDIT. Мой отец работал с 3270, часто пользовался XEDIT, а дома был KEDIT. Редактор был странным, но конечно лучше, чем EDIT.COM, поэтому я его и использовал. Позже я переехал на elvis, потому что сидел на Slackware, затем JED, о котором остались самые нежные воспоминания. Помню, что в 1995 году весь софт, написанный на библиотеке S-Lang, выглядел впечатляюще. Все мои сегодняшние схемы подстветки наследуются из JED и slrn.

Сейчас я использую Vim. Я не Vim-фанатик или даже не эксперт, но я кое-что знаю и написал достаточно Vim-скриптов, чтобы больше не хотеть этого делать. Мне нравится пробовать новые редакторы, но сомневаюсь, что я его сменю когда-нибудь снова.

Когда и как познакомился с Perl?

Когда я только пересел на Slackware в 1995 году, мне нужно было написать

несколько небольших программ для автоматизации таких задач как объединение USENET-постов и их декодирования. Я написал (ужасный) Си-код, и, наверное, смог бы закончить утилиты, но подозревал, что это будет мучительно. В Slackware было полно Perl-кода, и я решил выучить Perl.

Но это был Perl 4. Slackware, насколько я помню, очень медленно переходило на Perl 5. В конце концов он был доступен, но нужно было ставить отдельный пакет, указывающий на `/usr/bin/perl5`. Меня это не сильно волновало. Perl 4 мог делать все! И мне было тогда невдомек, зачем нужны модули и ссылки.

Я не начинал учить Perl 5 до 2001 года. У меня была первая полноценная работа, и я самостоятельно писал на PHP 4. Все отлично работало, но мне было понятно, что язык не помогает мне писать софт, который я смогу поддерживать длительное время. Мне нужно было что-то дисциплинирующее, и мне досталась в наследство книга Camelbook. Я ее прочел, переписал все на Perl 5 и никогда об этом не пожалел.

Наверное, я единственный человек, который начал использовать Perl потому, что он дисциплинирует.

С какими еще языками интересно работать?

Большинство серьезной работы я делаю на Perl 5, поэтому с другими языками я только играюсь. Иногда я работаю над большими проектами и всегда ищу причины использовать другие языки.

Есть много языков, похожих на Perl: PHP, Python, Ruby, JavaScript. Конечно, больше всего мне нравится Perl. Люди переоценивают разницу между ними.

В прошлом году я серьезно занялся Forth. В прошлом я пытался учить ассемблер и всегда его ненавидел. Forth мне показался ассемблером, который я всегда хотел. Он был простой, прямой, близкий к машине и укладывался у меня в голове. На конференциях я, бывает, убеждаю людей больше писать на Forth.

Мне действительно нравится Smalltalk (язык, не его странно работающее окружение), и мне нравится решать небольшие задачки на Prolog для разминки. Хотелось бы найти причины, чтобы больше писать на языке Io.

Я с удовольствием вспоминаю все языки, на которых по разным причинам

когда-либо доводилось писать. Кроме Visual Basic 5.

Что, по-твоему, является самым большим преимуществом Perl?

Есть объектно-ориентированные языки, функциональные языки, логические языки и языки других парадигм. Парадигма же Perl — ориентированность на результат. Perl это рабочий язык. Если работа будет выполнена один раз, можно набросать нечто монстроподобное и никогда об этом не говорить. Если работа должна выполняться каждый день в течение десяти лет, можно написать красивый надежный код и быть уверенным, что программа будет работать на будущих версиях Perl. Это не свойство языка, но это принцип по которому он эволюционирует с течением времени. Мы стараемся помочь вам сделать работу, никак не влияя на то, что уже работает.

Между прочим, можно многое узнать, просто наблюдая за Perl-сообществом, беря во внимание описанную выше идею.

Что, по-твоему, является самой важной особенностью языков будущего?

Многозадачность. Действительно, это одна из самых важных особенностей языков настоящего. Очевидно, что множество программ, которые мы пишем, могут быть гибче и производительней, если бы были спроектированы с акцентом на многозадачность. Perl 5 не ужасен в многозадачности. Есть множество фреймворков для разных видов многозадачности. Это частая практика у современных языков.

Мне кажется, что действительно важной особенностью языков будущего является простота и повсеместная абстракция для многозадачности во всем языке. В этой области на такие языки как Go и Erlang стоит обратить внимание.

Почему так много модулей? Когда это все закончится?

Разве это уже не закончилось? Мне страшно посмотреть на график своей активности!

Когда я решаю какую-то проблему так, что это можно использовать и в другом проекте, я выделяю решение в модуль. Иногда у модуляризации есть свои проблемы, поэтому возникает еще один уровень модуляризации. Через некоторое время это все накапливается, и в конце концов у меня получаются

модули из других модулей для управления другими модулями своих модулей. Вот!

Уже некоторое время я чувствую, что у меня достаточно утилит. Пока я не столкнусь с какими-то странными задачами, я буду выкладывать модули небыстро и стабильно. Но опять же, у меня есть список идей, которые бы хотелось реализовать: замена `URI.pm`, `Mason`-подобный шаблонизатор, `zasm`-компилятор...

Какая твоя текущая роль в разработке perl?

Я «pumpking», что-то вроде «временный великодушный диктатор». Эта должность переходит от человека к человеку. Со своей стороны я стараюсь, чтобы список рассылки perl5-porters не стагнировал и пинаю тогда, когда нужно запустить остановившийся процесс. Так же я пытался укрепить цивилизованное поведение в perl5-porters. Не хочу работать в условиях грубости; предполагаю, что и другие люди тоже.

Что ожидать от 5.22?

На данный момент 5.22 сильно отличается от 5.20. В 5.20 мы добавили множество захватывающих фишек, на которые можно показать пальцем и сказать: «Круто, я хочу это использовать!». 5.22 в основном это улучшение производительности — и их очень много! Вызовы методов стали быстрее, глубокое разыменование стало быстрее, есть несколько других оптимизаций. Конечно, есть также исправленные ошибки и другие хорошие небольшие фишки.

Если бы мне нужно было выбрать самое интересное, я бы выбрал новые экспериментальные псевдонимы ссылок:

```
\$x = \$y
```

...и обе будут одинаковые переменные, будучи псевдонимами друг друга. Это можно использовать и в циклах:

```
my @input = ( [ 1, 2, 3 ], [ 8, 9, 10 ] );
for \my $x (\ (@input) ) {
  $x++;
}
```

...и получить список ([2,3],[9,10]). Другими словами, можно получить такое же поведение, как и при использовании `$_` без использования этой перемен-

ной.

Какое сейчас направление в разработке perl самое главное?

Не думаю, что было бы правильным утверждать, что таковое есть. Есть несколько людей, который работают над perl в зависимости от своих собственных интересов. Я пытаюсь поддерживать то, что будет полезно и отговаривать от редких изменений, которые, по-моему, никогда не будут внедрены.

Безусловно, есть определенное количество людей, работающих над увеличением производительности, но есть также и много других областей разработки: улучшение генерации кода, чисел с плавающей запятой, оптимизация памяти и даже возобновление совместимости с EBCDIC (*кодировка* — прим. перев.). Мое видение в том, что если больше людей работают и делятся своей работой и планами, то все будет идти хорошо. Пытаться направлять людей туда, где им не нравится, никак не поможет ни Perl, ни тому как сейчас обстоят дела.

Есть некоторые, которые говорят, что обратная совместимость уже не в приоритете. Что можешь ответить на это?

Это явно не наш случай. Когда вопросы по изменению языка обсуждаются в perl5-porters, всегда поднимается вопрос об обратной совместимости, и это весомый фактор при принятии решения о том, что будет хорошо для Perl. Да и как я уже упоминал, мы не хотим ломать ваш существующий код, который работает.

Разные люди хотят разного размера изменений, и они хотят поменять разные части языка, поэтому часто возникают споры. Но тем не менее, и это должно быть явно видно из наблюдений за процессом в perl5-porters, обратная совместимость в приоритете. В прошлом году мы решили явно указывать причины некоторых редких нарушений обратной совместимости, которые войдут в документ «направление perl» до 5.22.

Есть ли что-нибудь из внутренней разработки perl, о чем ты хотел бы, чтобы узнали пользователя языка?

Это очень интересно! Еще задолго до того, как я начал принимать участие в разработке perl, я читал рассылку p5p для удовольствия. Много я удалял

не читая, так как мне не было интересно, но очень многое было захватывающим. Я познакомился с различными странными хитростями языка, некоторые из которых были намеренными. У меня появилось понимание причин разных приоритетов, назначенных разных частям языка разными людьми. Я узнал много об операционных системах, обмене данными, алгоритмах, структурах данных и динамике социальных групп. Кроме того, просто круто наблюдать, как умные люди работают вместе.

Даже когда мне не было интересно работать над perl, было захватывающим наблюдать все это, и это хорошо меня подготовило в дальнейшем, когда я захотел внести некоторые советы и изменения и быть более вовлеченным в разработку.

Что думаешь по поводу Perl 6?

Я осторожно оптимистичен. Я пытался не спускать глаз с разработки Perl 6, что было очень интересным. Каждый год, или вроде того, я писал небольшую программу на Perl 6, и это было весело. Perl 6 один из немногих языков, знакомых мне, который пытаются предоставить абстракции для многозадачности, оставаясь в то же время динамическим языком. Я точно буду продолжать пробовать что-нибудь писать на нем. Но этого языка так много, и до сих пор у меня не получалось все удержать в своей голове. Хотя это может сильно измениться, если я буду использовать его чаще, чем раз в год!

Где сейчас работаешь? Сколько времени проводишь за написанием Perl-кода?

Я работаю в Pobox.com уже около десяти лет. Мы предоставляем сервисы по использованию вашего email-адреса, и мы занимаемся этим уже двадцать лет, что меня сильно потрясает, учитывая продолжительность жизни других сервисов, которые я пытался использовать.

Наш внутренний код практически полностью написан на Perl. Есть немного Ruby, Python, C и PHP, и, конечно же, JavaScript, но где-то 95% своего времени я пишу на Perl. Мы пытаемся выкладывать на GitHub или CPAN наиболее полезный для общих задач код, поэтому большая часть моего кода на CPAN была написана для работы.

Стоит ли сейчас советовать молодым программистам учить Perl?

Мы должны советовать молодым программистам не забывать, что компьютеры это инструменты общего назначения для решения разного рода задач. Это может означать, что некоторые задачи могут требовать программирования. Не думаю, что выбор языка имеет значение, в той мере, что он помогает им решать текущую задачу и позволяет им чувствовать возможность решить и следующую.

Для меня Perl идеален. Мне кажется, что таким он будет и для многих других людей. С другой стороны, если бы я увидел ребенка, решающего задачи тем способом, которым он научился в «JavaScript для детей», я не буду ему говорить: «Тебе стоило написать это на Perl».

Для тех, кто хочет стать серьезным программистом, я хочу порекомендовать Forth! Дайте-ка я немного расскажу вам о нем…

Вопросы от читателей

Поедешь на FOSDEM?

Конечно! Я слышал много позитивного о FOSDEM, и всегда хотел поехать. В этом году я достаточно продвинулся с реализацией своей идеи и купил билет! Я буду в Perl-аудитории, но также и в других местах, надеюсь. Также предвкушаю дегустацию всего, что может предложить Брюссель, а также встречу со многими людьми, которые никогда не были на конференциях в Штатах.

Чем увлекаешься?

Последнее время я пытаюсь доказать себе, что независишу от сладостей. Перестал употреблять колу, кондитерские изделия, шоколад и много других вещей, кроме некоторых исключений. Это не делает меня несчастным, но и не легко.

Кроме этого я стараюсь увлекаться некоторым списком недорогих хобби. Я мог бы бросить одно из них в любое время, но после этого мне нужно найти новое. Мне не очень удается просто убивать время. Может быть, я завишу от желания быть всегда занятым.

Удалось ли тебе наконец восстановить все резервные копии?

Ух! Это до сих пор кусается! Наверное это о моей миграции музыки в июне. Самой большой проблемой было то, что я удалил копии, которые думал, что восстановил. В итоге я потерял огромное количество файлов. Мне удалось восстановить большинство, но теперь у меня нет треков всех моих любимых альбомов. Альбомы у меня не пропали целиком, поэтому мне тяжело заставить себя спуститься в подвал, распаковать CD-диски и снова их «рипнуть». На сегодняшний день отсутствуют 1598 файлов. Я должен что-то сделать. В большинстве случаев это «что-то» значит «слушать больше Spotify».

■ *Вячеслав Тихановский*