- черновой стандарт WebRTC обрел наибольшую стабильность;
- отсутствует необходимость установки плагинов для браузера;
- возможность использования встроенных возможностей HTML5 для воспроизведения потоковых аудио и видео данных.

Недостатки технологии WebRTC:

- стандарт находится в процессе разработки;
- поддерживается небольшой перечень браузеров.

Принцип реализации программного средства, использующего WebRTC, представлен на рисунке 1:

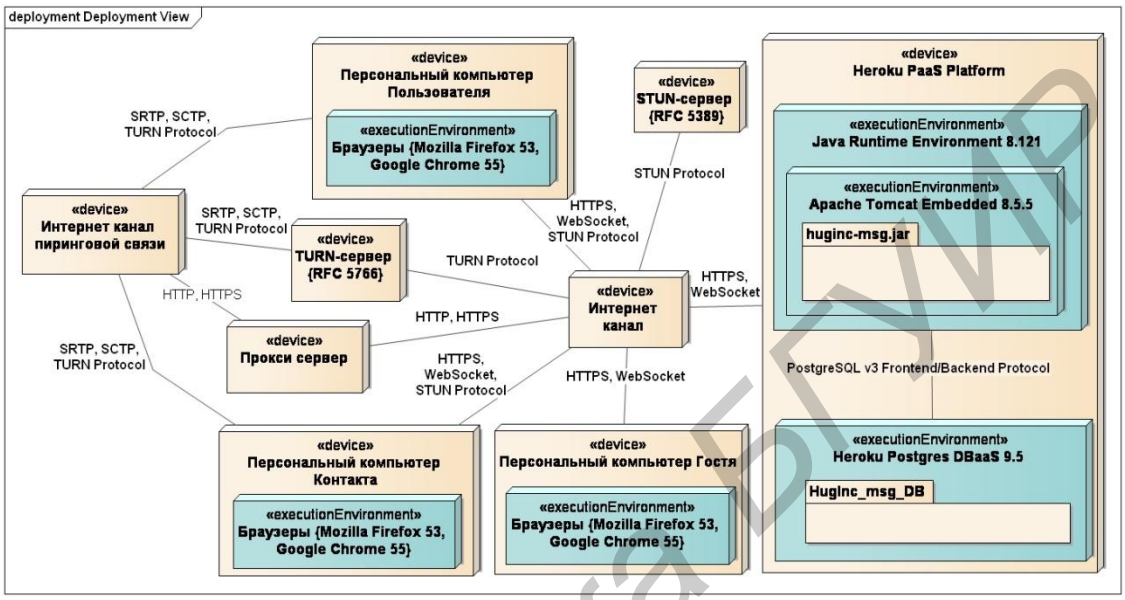

Рис. 1 – Диаграмма развертывания ПС мультимедийной связи браузеров по принципу пиринговой связи

Диаграмма развертывания отражает основные компоненты программного средства мультимедийной связи браузеров по принципу пиринговой связи. Схема разработана с использование принципов ICE из RFC 5245. Основные преимущества данной схемы:

а) масштабируемость;
б) уменьшение нагрузки на сеть благодаря использованию WebSocket;
в) возможность обеспечения контроля доступа к системе;
г) безопасная и качественная передача данных благодаря WebRTC;
д) поддержка последних версий браузеров;
е) возможность создания более гибких и удобных пользовательских интерфейсов.

Переход от JavaAppletAPI и AdobeFlash к WebRTC является залогом успешного будущего браузерных программных средств, работающих с передачей потоковых и обычных мультимедиа по сети.

Список использованных источников:

1. RFC 5245 Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols Interactive Connectivity Establishment [Электронныйресурс]: – Электронныеданные. – Режимдоступа: https://tools.ietf.org/pdf/rfc5245.pdf
2. RFC 5389 SessionTraversalutilitiesforNAT [Электронный ресурс]: – Электронные данные. – Режим доступа: https://tools.ietf.org/pdf/rfc5389.pdf
3. RFC 5766 Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN) [Электронныйресурс]: – Электронныеданные. – Режимдоступа: https://tools.ietf.org/pdf/rfc5766.pdf
4. Технология WebRTC [Электронный ресурс]: – Электронные данные. – Режим доступа: https://webrtc.org

ОБФУСКАЦИЯ ПРОГРАММНОГО КОДА

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Быцкевич Ю. И.

Прохорчик Р. В. – м.т.н., ассистент

В современном мире программное обеспечение является продуктом, который наравне с другими товарами массово производится, продаётся, покупается, а также нуждается в защите от различных угроз со

стороны злоумышленников. Одним из повсеместно распространенных механизмов защиты является использование лицензии несвободного использования. Однако раз за разом злоумышленники доказывают, что взлом этого механизма не является невозможным.

Одной из уязвимостей, которые позволяют взламывать лицензионные продукты, может являться доступность исходного кода. Владея исходным кодом, можно обойти лицензионный механизм, произвести обратную разработку, произвольно модифицировать программу, определить узкие места в безопасности.

Для противодействия этому есть несколько способов, среди которых – обфускация (запутывание) исходного кода.

Сущность обфускации заключается в том, чтобы различными способами изменить код так, чтобы при сохранении своей функциональности он имел максимально сложный для понимания вид [1].

В первую очередь в обфускации нуждаются исходные коды, которые распространяются в Интернете в открытом виде. Речь идет о клиентской (front-end) части веб-приложений – например, html, css, javascript код разнообразных сайтов может быть просмотрен без труда из браузера. Исходные коды бизнес-логики (back-end) веб-приложений как правило защищены от злоумышленника, так как сами веб-приложения разворачиваются на удалённых серверах, а исходные коды хранятся приватно у разработчиков.

Что касается десктопных приложений, здесь есть особенности. Несмотря на то, что исходные коды разработчиками не распространяются, они могут быть восстановлены из распространяемых исполняемых модулей, если те представлены не машинным, а промежуточным кодом. Такой опасности подвержены, в частности, программы платформ .NET и Java. Поэтому в случае, если есть необходимость защитить логику и структуры данных этих программ от использования злоумышленником, произведение обфускации является хорошей практикой.

Обычно в отношении программ, компилируемых в промежуточный код (.NET, Java), производится запутывание самого промежуточного кода (то есть исполняемых модулей, а не исходников). Из такого кода извлечь с помощью декомпиляции оригинальные исходные коды не удастся.

Причина такого подхода в том, что промежуточный код – финальный продукт, который и будет подлежать распространению, в отличие от исходного кода, которому еще предстоит пройти компиляцию. Компилятор производит над кодом ряд оптимизаций, поэтому если запутывать исходный код, некоторые из преобразований могут быть ликвидированы (например, недостижимый/мертвый код).

Но обфускация промежуточного кода имеет и минусы. Самый главный из них – то что чем объёмнее и сложнее программа, тем больше шанс того, что преобразования обфускатора внесут ошибки в её работу. Причем эти ошибки будет сложно обнаружить, так как речь идет о готовом исполняемом модуле (.exe). Также их сложно будет исправить, так как далеко не все разработчики столь же хорошо знакомы с промежуточным языком, как с исходным (например, далеко не все C#-разработчики смогут отладить IL-код). Может случиться и так, что программа после обработки обфускатором не запустится вовсе. Все зависит от качества написания используемого обфускатора, а также от набора и сложности преобразований, которые он производит.

В этом отношении запутывание исходных кодов имеет преимущества: можно производить более изощрённые преобразования (например, структур данных), оперируя более сложными абстракциями, и производить отладку запутанного кода, пользуясь средствами среды разработки. Неработающий код просто не скомпилируется, а среда разработки даст подсказки относительно мест и причин ошибок.

Проведя анализ плюсов и минусов подходов обфускации исходного и промежуточного кода, можно сделать вывод, что подходы не противоречат, а дополняют друг друга. В сочетании они дают возможность использовать достоинства обоих решений.

Таким образом, в качестве актуального и эффективного обфускатора может рассматриваться программа, реализующая двухступенчатое запутывание: сначала исходного кода, затем – промежуточного. Первый этап позволяет производить семантически более сложные преобразования, второй – преобразования по внесению дополнительной информационной нагрузки, которые нельзя выполнить в первом этапе из-за оптимизаций компилятора.

Примерный список запутывающих преобразований, который можно выделить для реализации в двухступенчатом обфускаторе:

1. Преобразования для исходного кода:
   - удаление всех комментариев из кода;
   - удаление отступов, переносов строки, пробелов (где это возможно);
   - вставку дополнительных фигурных и круглых скобок (без нарушения хода выполнения программы);
   - изменение названий пространств имен, классов, переменных, свойств, методов;
   - вставка кодов функций в место их вызова;
   - развёртка циклов;
   - переплетение (слияние) функций;
   - внесение кода, который может быть упрощён или удален вовсе (избыточного кода);
   - шифрование константных строк;
   - в рамках одного класса объединение переменных одного типа в массив;
   - изменение иерархии классов;
   - внесение фиктивных классов в проект и создание их экземпляров в исходном коде проекта, а также в других фиктивных классах [2,3].
2. Преобразования для промежуточного кода:

- реструктуризация графа потока управления (разбиение кода на блоки, перемешивание блоков, управление посредством элемента-диспетчера);
- нарушение порядка выполнения программы с использованием инструкций безусловного перехода;
- внесение кода, который никогда не будет выполняться, а только увеличивает информационную нагрузку (недостижимого кода);
- внесение кода, который может быть упрощён или удален вовсе (избыточного кода) [2,3].

Список использованных источников:

1. Wikipedia [Электронный ресурс]. – Электронные данные. – Режим доступа: https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%84%D1%83%D1%81%D0%BA%D0%B0%D1%86%D0%B8%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5)
2. Christian Collberg, Clark Thomborson, Douglas Low. A Taxonomy of Obfuscating Transformations. – Department of Computer Science, The University of Auckland, 1997 – 36 с.
3. Чернов А. В. Анализ запутывающих преобразований программ – Институт Системного программирования РАН, 2003 – 34 с.

ПРОГРАММНОЕ СРЕДСТВО СОПРОВОЖДЕНИЯ УЧЕБНОГО ПРОЦЕССА

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Василенко П.И.

Данилова Г.В., м.т.н.

XXI век проходит под знаменем глобальной информатизации общества. Информационные технологии повсеместно внедряются в самые разнообразные сферы жизнедеятельности человека, начиная с науки, продолжая в финансах и заканчивая в сфере услуг и развлечений. Однако стоит отметить, что такое глобальное явление обошло стороной одну из важнейших отраслей человеческой жизни – это образование.

В современном мире, и, в особенности, в сфере высоких технологий огромную роль для каждого специалиста играет самообразование. Благодаря стремительному развитию всемирной сети, появлению новых моделей накопления и распространения знаний каждый человек, имея доступ к Интернету, имеет также и доступ к огромному количеству информации.

К сожалению, несмотря на наличие достаточного количества недостатков, материал, составленный квалифицированным сотрудником престижных научных учереждений, всё ещё является наиболее достоверным источником информации. Именно это и есть один из самых серьёзных преимуществ обучения в специальных заведениях по сравнению с самообразованием.

Очевидной является потребность в усовершенствовании передачи информации от преподавателя к студенту. Удовлетворить эту потребность призвано программное средство, которое позволит автоматизировать определенные этапы обучения, расширить возможности для получения знания и повысить эффективность оного процесса.

Основные возможности, предоставляемые программным средством:

- Для преподавателя:
  - Возможность проведения лекционных занятий, как для присутствующих студентов, так и для отсутствующих в лекционной аудитории.
  - Возможность подготовки дополнительного учебного материала, которая не требует дополнительного участия от преподавателя и студента.
  - Возможность многократно увеличить эффективность учебного материала и его переиспользуемость.
  - Возможность без особых усилий консультировать студентов по волнующим их вопросам в рамках курса.
- Для студента:
  - Возможность получить знания, будучи непривязанным географически к какому-либо месту.
  - Возможность повторного разбора материала без необходимости участия преподавателя.
  - Возможность получить консультацию преподавателя.
- Для учебного заведения:
  - Возможность повысить престиж учебного заведения.
  - Возможность подготовки отдельных частей курсов без постоянного участия преподавателя.
  - Возможность подготовки материала, который можно распространять по платной подписке.