Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»
Кафедра информатики

Отчет по лабораторной работе №4-5

Разработка защищенных приложений.

Выполнил:
Брычиков Д.Д.
Проверил:
Чернявский Ю. А.

Минск 2018

Введение

Web-приложения — наиболее распространенные программные сервисы, доступные через интернет, поэтому они являются лакомым куском для всяких злоумышленников, желающих получить доступ к вашей сети и похитить ценную информацию, испортить ваши данные или как-то иначе скомпрометировать вашу систему. Обеспечение безопасности Web-приложения — весьма серьезная задача, которой нужно уделять должное внимание на всех этапах — при проектировании, разработке, развертывании и эксплуатации. Не думайте, что для этого достаточно что-то приделать к существующему приложению или просто применить средства защиты, поддерживаемые платформой.

Даже когда разрабатываемое Web-приложение будет работать на относительно безопасной платформе, на этапах проектирования, разработки и развертывания нужно использовать передовой опыт в области безопасности, чтобы обеспечить максимальный уровень защиты от атак. Создать защищенное Web-приложение без знания специфики платформы в сочетании с методиками проектирования безопасных систем, моделированием угроз и тестированием системы защиты на проникновение невозможно.

Постановка задачи

- Познакомиться с концепцией ролевого управления доступом и способами защиты программного обеспечения от существующих угроз.

Научиться разрабатывать приложения, которые используют ролевое управление доступом для разграничения полномочий пользователей. Получить навыки защиты разработанной программы от несанкционированного копирования и других угроз, которым может подвергаться программное обеспечение.

Переполнение буфера

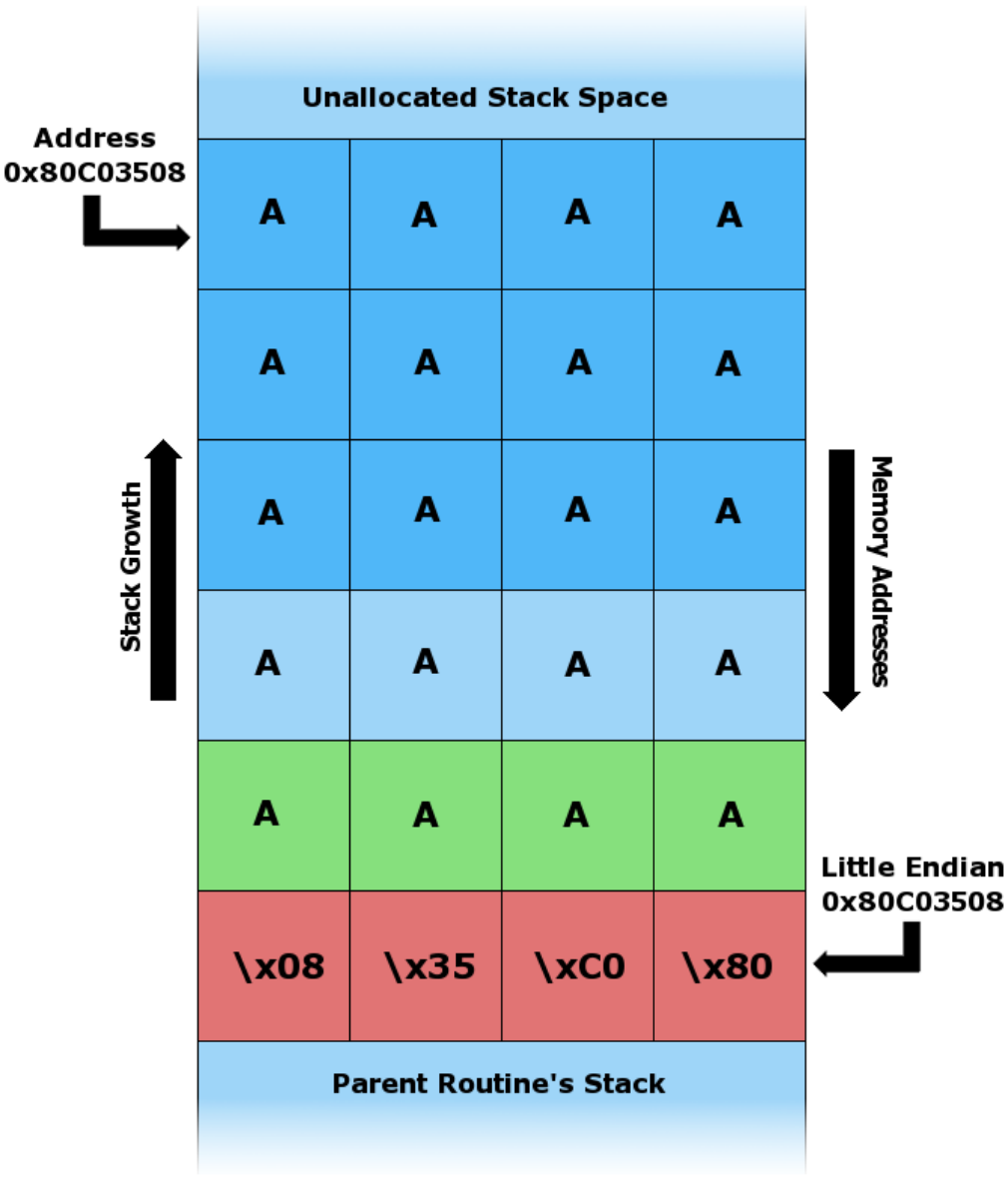
Переполнение буфера (англ. Buffer Overflow) — явление, возникающее, когда компьютерная программа записывает данные за пределами выделенного в памяти буфера.

Переполнение буфера обычно возникает из-за неправильной работы с данными, полученными извне, и памятью, при отсутствии жесткой защиты со стороны подсистемы программирования (компилятор или интерпретатор) и операционной системы. В результате переполнения могут быть испорчены данные, расположенные следом за буфером (или перед ним).

Переполнение буфера является одним из наиболее популярных способов взлома компьютерных систем, так как большинство языков высокого уровня используют технологию стекового кадра — размещение данных в стеке процесса, смешивая данные программы с управляющими данными (в том числе адреса начала стекового кадра и адреса возврата из исполняемой функции).

Переполнение буфера может вызывать аварийное завершение или зависание программы, ведущее к отказу обслуживания (denial of service, DoS). Отдельные виды переполнений, например переполнение в стековом кадре, позволяют злоумышленнику загрузить и выполнить произвольный машинный код от имени программы и с правами учетной записи, от которой она выполняется.

Известны примеры, когда переполнение буфера намеренно используется системными программами для обхода ограничений в существующих программных или программно-аппаратных средствах. Например, операционная система iS-DOS (для компьютеров ZX Spectrum) использовала возможность переполнения буфера встроенной TR-DOS для запуска своего загрузчика в машинных кодах (что штатными средствами в TR-DOS сделать невозможно).


Unallocated Stack Space
Address
0x80C03508
A
A
A
A
A
A
A
A
A
A
A
A
A
A
A
A
A
A
A
A
\x08
\x35
\xC0
\x80
Parent Routine's Stack
Stack Growth
Memory Addresses
Little Endian
0x80C03508

Внедрение SQL-кода

Внедрение SQL-кода (англ. SQL injection) — один из распространённых способов взлома сайтов и программ, работающих с базами данных, основанный на внедрении в запрос произвольного SQL-кода.

Внедрение SQL, в зависимости от типа используемой СУБД и условий внедрения, может дать возможность атакующему выполнить произвольный запрос к базе данных (например, прочитать содержимое любых таблиц, удалить, изменить или добавить данные), получить возможность чтения и/или записи локальных файлов и выполнения произвольных команд на атакуемом сервере.

Атака типа внедрения SQL может быть возможна из-за некорректной обработки входных данных, используемых в SQL-запросах.

Разработчик прикладных программ, работающих с базами данных, должен знать о таких уязвимостях и принимать меры противодействия внедрению SQL.

Допустим, серверное ПО, получив входной параметр id, использует его для создания SQL-запроса. Рассмотрим следующий PHP-скрипт:

```
$id = $_REQUEST['id'];
$res = mysqli_query("SELECT * FROM news WHERE id_news = " . $id);
```

Если на сервер передан параметр id, равный 5 (например так: http://example.org/script.php?id=5), то выполнится следующий SQL-запрос:

```
SELECT * FROM news WHERE id_news = 5
```

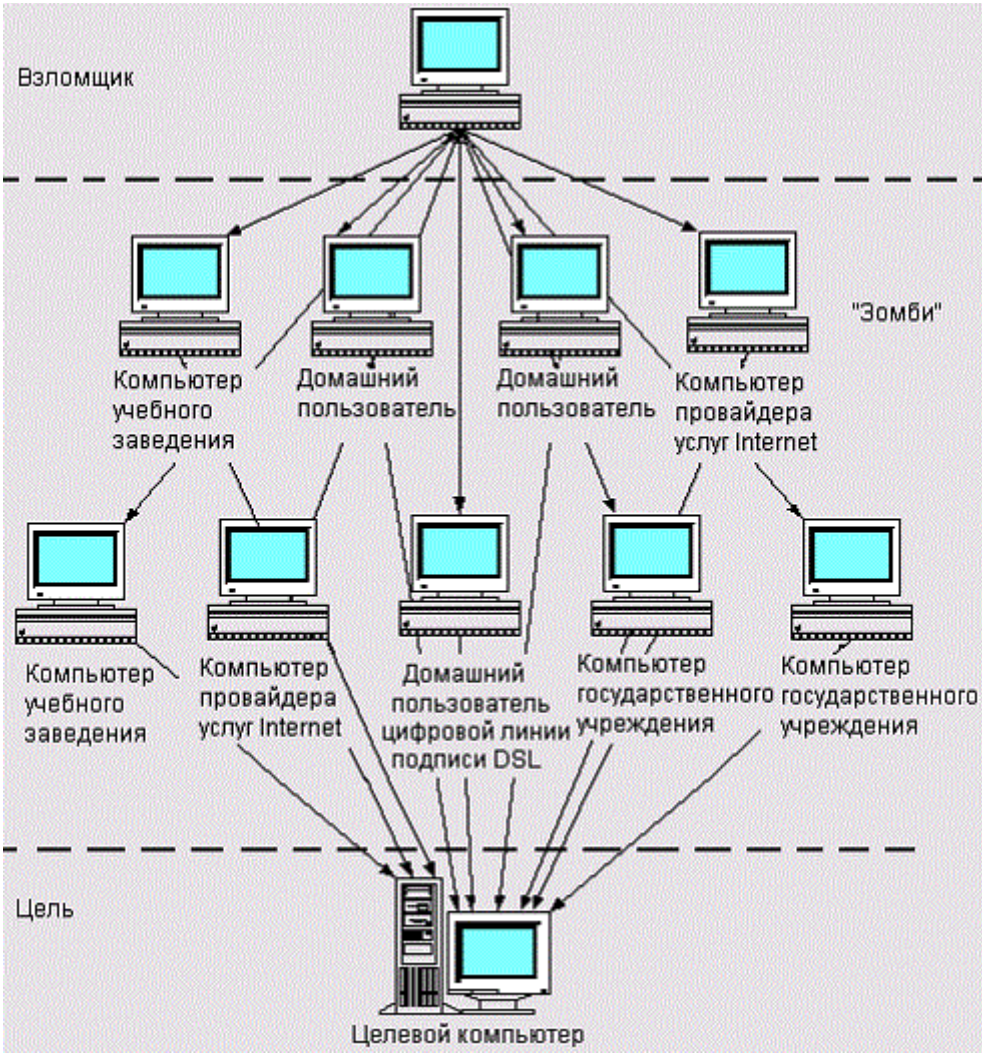
Но если злоумышленник передаст в качестве параметра id строку -1 OR 1=1 (например, так: http://example.org/script.php?id=-1+OR+1=1), то выполнится запрос:

```
SELECT * FROM news WHERE id_news = -1 OR 1=1
```

Таким образом, изменение входных параметров путём добавления в них конструкций языка SQL вызывает изменение в логике выполнения SQL-запроса (в данном примере вместо новости с заданным идентификатором будут выбраны все имеющиеся в базе новости, поскольку выражение 1=1 всегда истинно — вычисления происходят по кратчайшему контуру в схеме).

DoS-атака

DoS (аббр. Англ. Denial of Service «отказ в обслуживании») — хакерская атака на вычислительную систему с целью довести её до отказа, то есть создание таких условий, при которых добросовестные пользователи системы не могут получить доступ к предоставляемым системным ресурсам (серверам), либо этот доступ затруднён. Отказ «вражеской» системы может быть и шагом к овладению системой (если в нештатной ситуации ПО выдаёт какую-либо критическую информацию — например, версию, часть программного кода и т. д.). Но чаще это мера экономического давления: потеря простой службы, приносящей доход, счета от провайдера и меры по уходу от атаки ощутимо бьют «цель» по карману. В настоящее время DoS и DDoS-атаки наиболее популярны, так как позволяют довести до отказа практически любую систему, не оставляя юридически значимых улик.



Распределённая DoS-атака

Если атака выполняется одновременно с большого числа компьютеров, говорят о **DDoS-атаке** (от англ. *Distributed Denial of Service*, *распределённая атака типа «отказ в обслуживании»*). Такая атака проводится в том случае, если требуется

вызвать отказ в обслуживании хорошо защищённой крупной компании или правительственной организации.

Первым делом злоумышленник сканирует крупную сеть с помощью специально подготовленных сценариев, которые выявляют потенциально слабые узлы. Выбранные узлы подвергаются нападению, и злоумышленник получает на них права администратора. На захваченные узлы устанавливаются троянские программы, которые работают в фоновом режиме. Теперь эти компьютеры называются компьютерами-зомби, их пользователи даже не подозревают, что являются потенциальными участниками DDoS-атаки. Далее злоумышленник отправляет определенные команды захваченным компьютерам и те, в свою очередь осуществляют мощную DoS-атаку на целевой компьютер.

Существуют также программы для добровольного участия в DDoS-атаках.

В некоторых случаях к фактической DDoS-атаке приводит непреднамеренное действие, например, размещение на популярном интернет-ресурсе ссылки на сайт, размещённый на не очень производительном сервере (слэшдот-эффект). Большой наплыв пользователей приводит к превышению допустимой нагрузки на сервер и, следовательно, отказу в обслуживании части из них.

Защита

Для защиты от сетевых атак применяется ряд фильтров, подключенных к интернет-каналу с большой пропускной способностью. Фильтры действуют таким образом, что последовательно анализируют проходящий трафик, выявляя нестандартную сетевую активность и ошибки. В число анализируемых шаблонов нестандартного трафика входят все известные на сегодняшний день методы атак, в том числе реализуемые и при помощи распределённых бот-сетей.

Классификация DoS-атак

Хакерам гораздо легче осуществить DoS-атаку на систему, чем получить полный доступ к ней. Существуют различные причины, из-за которых может возникнуть DoS-условие, то есть такая ситуация, при которой пользователи не могут получить доступ к ресурсам, которые предоставляет сервер, либо доступ к ним существенно затруднен:

Насыщение полосы пропускания

В настоящее время практически каждый компьютер подключён к сети Internet либо к локальной сети. Это служит отличным поводом для осуществления DoS-атаки за счет переполнения полосы пропускания. Обычно злоумышленники пользуются флудом(англ. *flood* — «наводнение», «переполнение») — атака, связанная с большим количеством обычно бессмысленных или сформированных в неправильном формате запросов к компьютерной системе или сетевому оборудованию, имеющая своей целью или приведшая к отказу в работе системы из-за исчерпания системных ресурсов — процессора, памяти или каналов связи. Есть несколько разновидностей флуда.

HTTP-флуд и ping-флуд

Это самый примитивный вид DoS-атаки. Насыщение полосы пропускания можно осуществить с помощью обычных ping-запросов только в том случае, если канал атакующего (например 1,544 Мбит/с) намного шире канала компьютера-жертвы, скорость в котором 128 Кбит/с. Но такая атака бесполезна против сервера, так как тот, в свою очередь, обладает довольно широкой полосой пропускания. Для атаки на сервер обычно применяется HTTP-флуд. Атакующий шлёт маленький по объёму HTTP-пакет, но такой, чтобы сервер ответил на него пакетом, размер которого в сотни раз больше. Даже если канал сервера в десять раз шире канала атакующего, то все равно есть большой шанс насытить полосу пропускания жертвы. А для того, чтобы ответные HTTP-пакеты не вызвали отказ в обслуживании у злоумышленника, он каждый раз подменяет свой ip-адрес на ip-адреса узлов в сети.

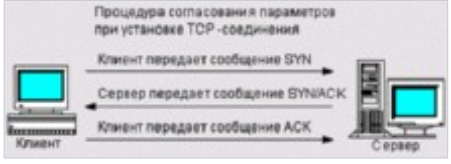
Smurf-атака (ICMP-флуд)

Атака Smurf или ICMP-флуд — один из самых опасных видов DoS-атак, так как у компьютера-жертвы после такой атаки произойдет отказ в обслуживании практически со 100 % гарантией. Злоумышленник использует широковещательную рассылку для проверки работающих узлов в системе, отправляя ping-запрос. Очевидно, атакующий в одиночку не сможет вывести из строя компьютер-жертву, поэтому требуется ещё один участник — это усиливающая сеть. В ней по широковещательному адресу злоумышленник отправляет поддельный ICMP пакет. Затем адрес атакующего меняется на адрес жертвы. Все узлы пришлют ей ответ на ping-запрос. Поэтому ICMP-пакет, отправленный злоумышленником через усиливающую сеть, содержащую 200 узлов, будет усилен в 200 раз. Поэтому для такой атаки обычно выбирается большая сеть, чтобы у компьютера-жертвы не было никаких шансов.

Атака Fraggle (UDP-флуд)

Атака Fraggle (осколочная граната)(от англ. *Fraggle attack*) является полным аналогом Smurf-атаки, где вместо ICMP пакетов используются пакеты UDP, поэтому её ещё называют UDP-флуд. Принцип действия этой атаки простой: на седьмой порт жертвы отправляются echo-команды по широковещательному запросу. Затем подменяется ip-адрес злоумышленника на ip-адрес жертвы, которая вскоре получает множество ответных сообщений. Их количество зависит от числа узлов в сети. Эта атака приводит к насыщению полосы пропускания и полному отказу в обслуживании жертвы. Если все же служба echo отключена, то будут сгенерированы ICMP-сообщения, что также приведёт к насыщению полосы.

Атака с помощью переполнения пакетами SYN (SYN-флуд)



Установка TCP — соединения

До появления атаки Smurf была широко распространена атака с помощью переполнения пакетами SYN, также известная под названием SYN-флуд. Для описания её действия можно остановиться на рассмотрении двух систем A и B, которые хотят установить между собой TCP соединение, после которого они смогут обмениваться между собой данными. На установку соединения выделяется

некоторое количество ресурсов, этим и пользуются DoS-атаки. Отправив несколько ложных запросов, можно израсходовать все ресурсы системы, отведённые на установление соединения. Рассмотрим подробнее, как это происходит. Хакер с системы A отправляет пакет SYN системе B, но предварительно поменяв свой IP-адрес на несуществующий. Затем, ничего не подозревая, компьютер B отправляет ответ SYN/ACK на несуществующий IP-адрес и переходит в состояние SYN-RECEIVED. Так как сообщение SYN/ACK не дойдет до системы A, то компьютер B никогда не получит пакет с флагом ACK.Данное потенциальное соединение будет помещено в очередь. Из очереди оно выйдет только по истечении 75 секунд. Этим пользуются злоумышленники и отправляют сразу несколько пакетов SYN на компьютер жертвы с интервалом в 10 секунд, чтобы полностью исчерпать ресурсы системы. Определить источник нападения очень непросто, так как злоумышленник постоянно меняет исходный IP-адрес.

CSRF

CSRF (англ. Cross Site Request Forgery — «межсайтовая подделка запроса», также известна как XSRF) — вид атак на посетителей веб-сайтов, использующий недостатки протокола HTTP. Если жертва заходит на сайт, созданный злоумышленником, от её лица тайно отправляется запрос на другой сервер (например, на сервер платёжной системы), осуществляющий некую вредоносную операцию (например, перевод денег на счёт злоумышленника). Для осуществления данной атаки жертва должна быть аутентифицирована на том сервере, на который отправляется запрос, и этот запрос не должен требовать какого-либо подтверждения со стороны пользователя, которое не может быть проигнорировано или подделано атакующим скриптом.

Данный тип атак, вопреки распространённому заблуждению, появился достаточно давно: первые теоретические рассуждения появились в 1988 году, а первые уязвимости были обнаружены в 2000 году. А сам термин ввёл Peter Watkins в 2001 году.

Основное применение CSRF — вынуждение выполнения каких-либо действий на уязвимом сайте от лица жертвы (изменение пароля, секретного вопроса для восстановления пароля, почты, добавление администратора и т. д.). Также с помощью CSRF возможна эксплуатация отражённых XSS, обнаруженных на другом сервере.

Пример
Атака осуществляется путём размещения на веб-странице ссылки или скрипта, пытающегося получить доступ к сайту, на котором атакуемый пользователь заведомо (или предположительно) уже аутентифицирован. Например, пользователь Алиса может просматривать форум, где другой пользователь, Мэллори, разместил сообщение. Пусть Мэллори создал тег <img>, в котором в качестве источника картинки указал URL, при переходе по которому выполняется действие на сайте банка Алисы, например:

```
Мэллори: Привет, Алиса! Посмотри, какой милый котик: <img
src="http://bank.example.com/withdraw?
account=Alice&amount=1000000&for=Mallory">
```

Если банк Алисы хранит информацию об аутентификации Алисы в куки, и если куки ещё не истекли, при попытке загрузить картинку браузер Алисы отправит куки в запросе на перевод денег на счёт Мэллори, чем подтвердит аутентификацию Алисы. Таким образом, транзакция будет успешно завершена, хотя её подтверждение произойдет без ведома Алисы.

Защищаться должны все запросы, изменяющие данные на сервере, а также запросы, возвращающие персональные или иные деликатные данные.

Наиболее простым способом защиты от данного типа атак является механизм, когда веб-сайты должны требовать подтверждения большинства действий пользователя и проверять поле HTTP_REFERER, если оно указано в запросе. Но этот способ может быть небезопасен, и использовать его не рекомендуется.

Другим распространённым способом защиты является механизм, при котором с каждой сессией пользователя ассоциируется дополнительный секретный уникальный ключ, предназначенный для выполнения запросов. Секретный ключ не должен передаваться в открытом виде, например, для POST запросов ключ следует передавать в теле запроса, а не в адресе страницы. Браузер пользователя посылает этот ключ в числе параметров каждого запроса, и перед выполнением каких-либо действий сервер проверяет этот ключ. Преимуществом данного механизма, по сравнению с проверкой Referer, является гарантированная защита от атак CSRF. Недостатками же являются требование возможности организации пользовательских сессий и требование динамической генерации HTML-кода страниц сайта.

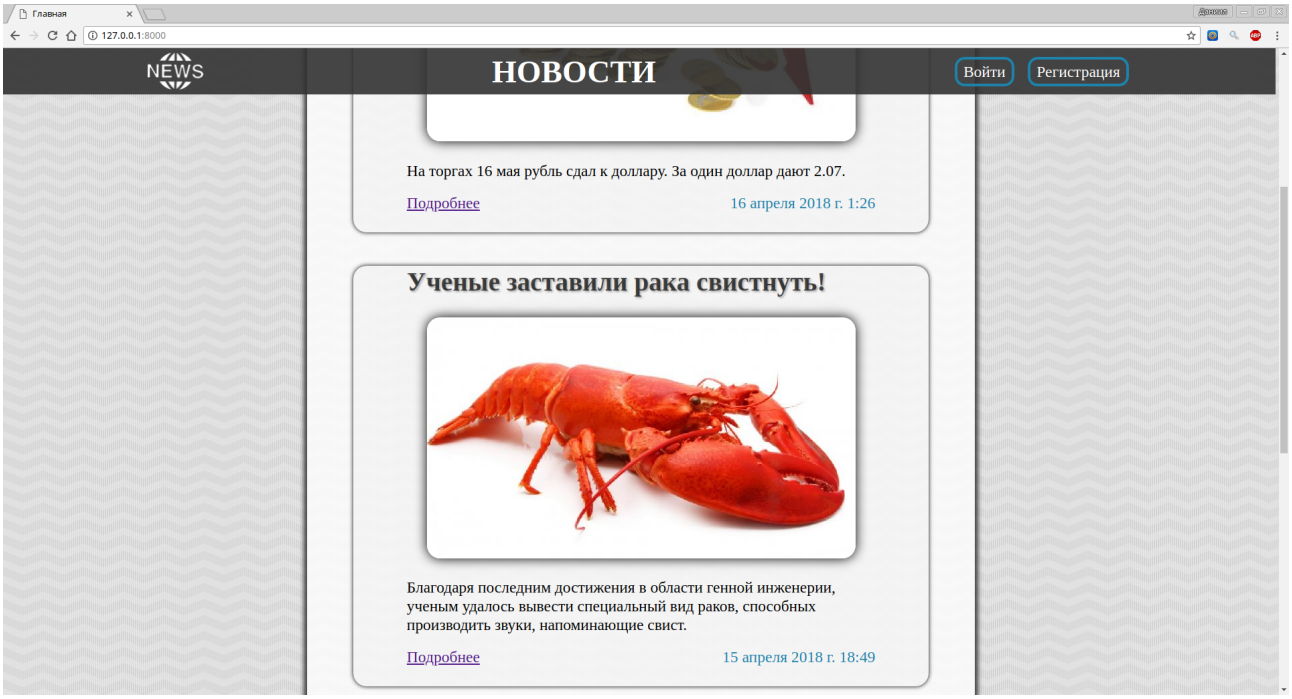
Спецификация протокола HTTP/1.1 определяет безопасные методы запросов, такие как GET, HEAD, которые не должны изменять данные на сервере. Для таких запросов, при соответствии сервера спецификации, нет необходимости применять защиту от CSRF.

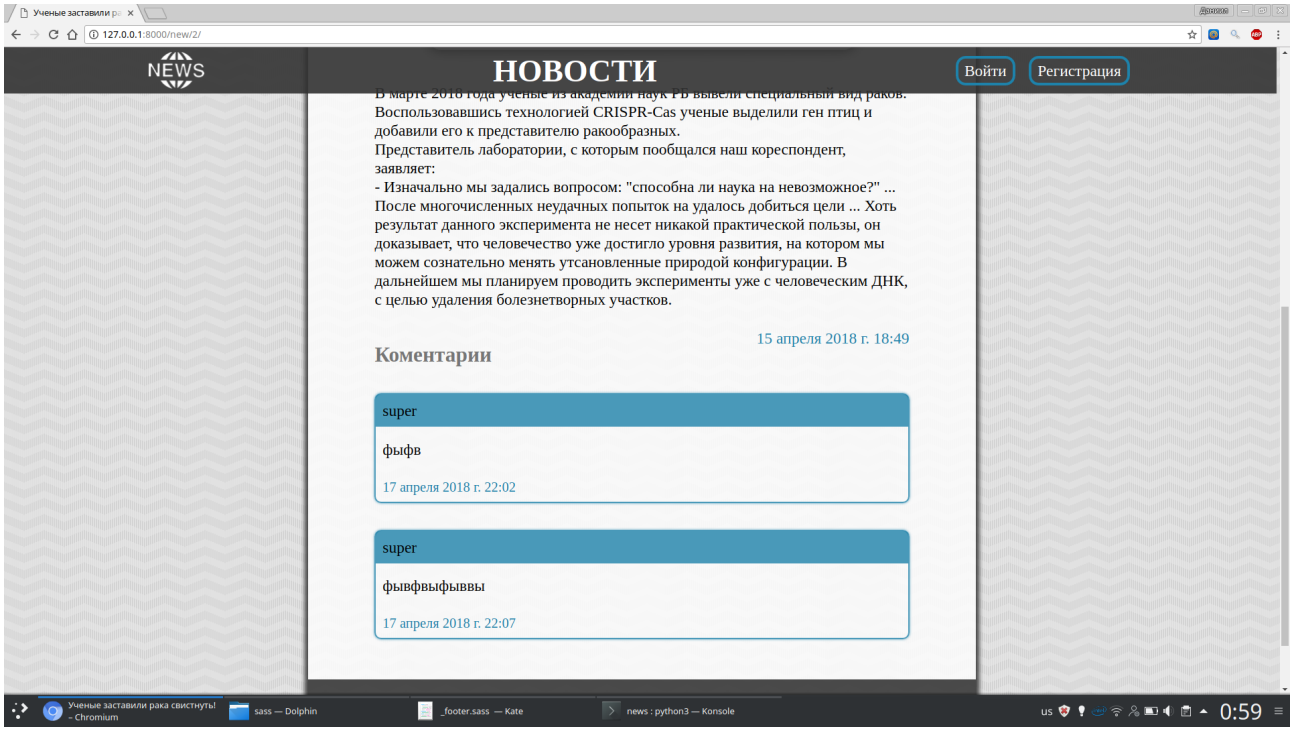
Может возникнуть желание подстраховаться и добавить ключ в каждый запрос, но следует иметь в виду, что спецификация HTTP/1.1[3] допускает наличие тела для любых запросов, но для некоторых методов запроса (GET, HEAD, DELETE) семантика тела запроса не определена, и должна быть проигнорирована. Поэтому ключ может быть передан только в самом URL или в HTTP заголовке запроса. Необходимо защитить пользователя от неблагоразумного распространения ключа в составе URL, например, на форуме, где ключ может оказаться доступным злоумышленнику. Поэтому запросы с ключом в URL не следует использовать в качестве адреса для перехода, то есть исключить переход по такому адресу клиентским скриптом, перенаправлением сервера, действием формы, гиперссылкой на странице и т. п. с целью сокрытия ключа, входящего в URL. Их можно использовать лишь как внутренние запросы скриптом с использованием XMLHttpRequest или обёрткой, например AJAX.

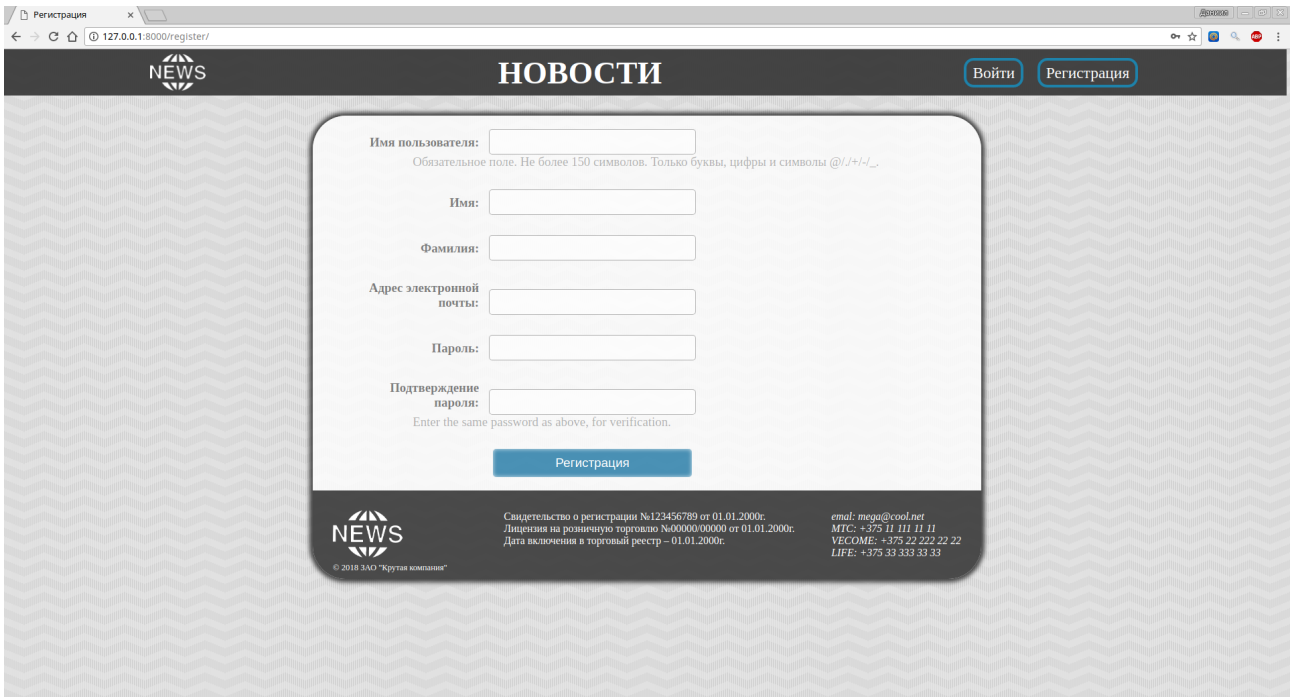
Существенен факт того, что ключ (CSRF токен) может быть предназначен не для конкретного запроса или формы, а для всех запросов пользователя вообще. Поэтому достаточно утечки CSRF токена с URL, выполняющим простое действие или не выполняющего действие вовсе, как защиты от подделки запроса лишается любое действие, а не только то, с которым связан ставший известным URL.

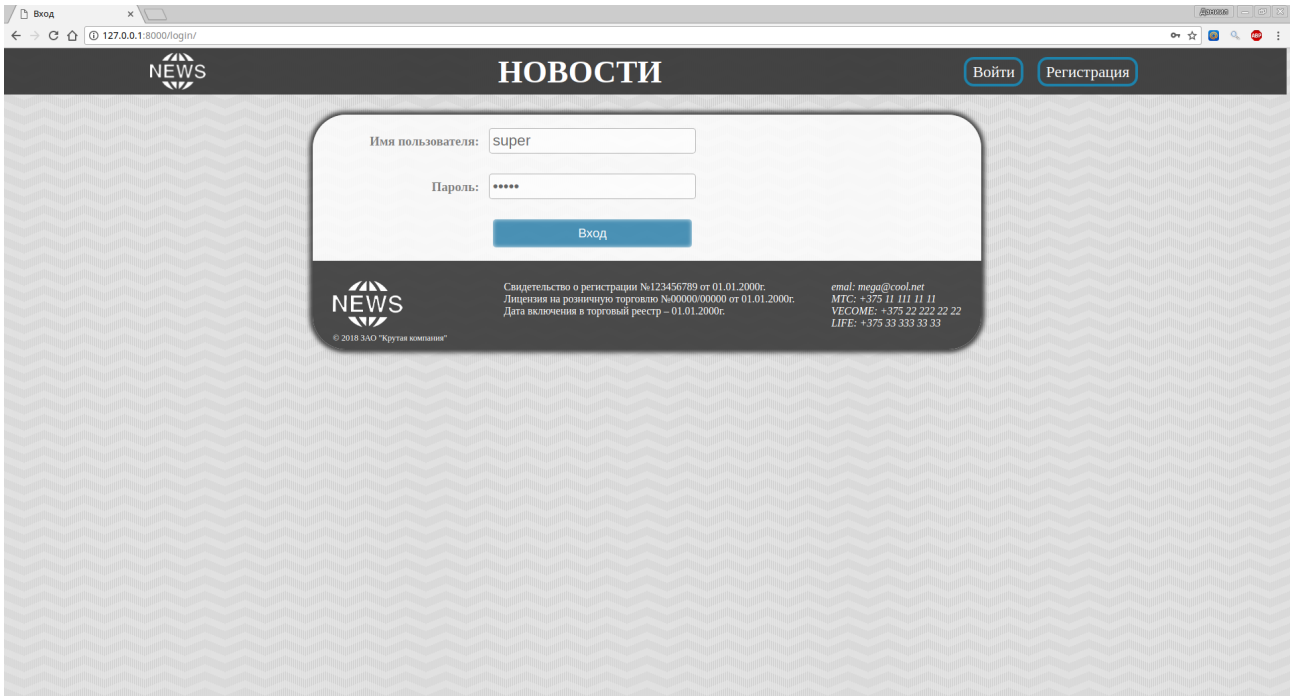
Существует более жёсткий вариант предыдущего механизма, в котором с каждым действием ассоциируется уникальный одноразовый ключ. Такой способ более сложен в реализации и требователен к ресурсам. Способ используется некоторыми сайтами и порталами, такими как Livejournal, Rambler и др.

Демонстрация работы приложения






Регистрация
127.0.0.1:8000/register/
NEWS
НОВОСТИ
Войти
Регистрация
Имя пользователя:
Обязательное поле. Не более 150 символов. Только буквы, цифры и символы @/./+/-/_.
Имя:
Фамилия:
Адрес электронной почты:
Пароль:
Подтверждение пароля:
Enter the same password as above, for verification.
Регистрация
NEWS
© 2018 ЗАО "Крутая компания"
Свидетельство о регистрации №123456789 от 01.01.2000г.
Лицензия на розничную торговлю №00000/00000 от 01.01.2000г.
Дата включения в торговый реестр – 01.01.2000г.
email: mega@cool.net
МТС: +375 11 111 11 11
VECOME: +375 22 222 22 22
LIFE: +375 33 333 33 33


Вход
127.0.0.1:8000/login/
NEWS
НОВОСТИ
Войти
Регистрация
Имя пользователя:
super
Пароль:
Вход
NEWS
© 2018 ЗАО "Крутая компания"
Свидетельство о регистрации №123456789 от 01.01.2000г.
Лицензия на розничную торговлю №00000/00000 от 01.01.2000г.
Дата включения в торговый реестр – 01.01.2000г.
email: mega@cool.net
МТС: +375 11 111 11 11
VECOME: +375 22 222 22 22
LIFE: +375 33 333 33 33

Права пользователей

Посетитель:

- список новостей
- читать новость
- читать комментарии

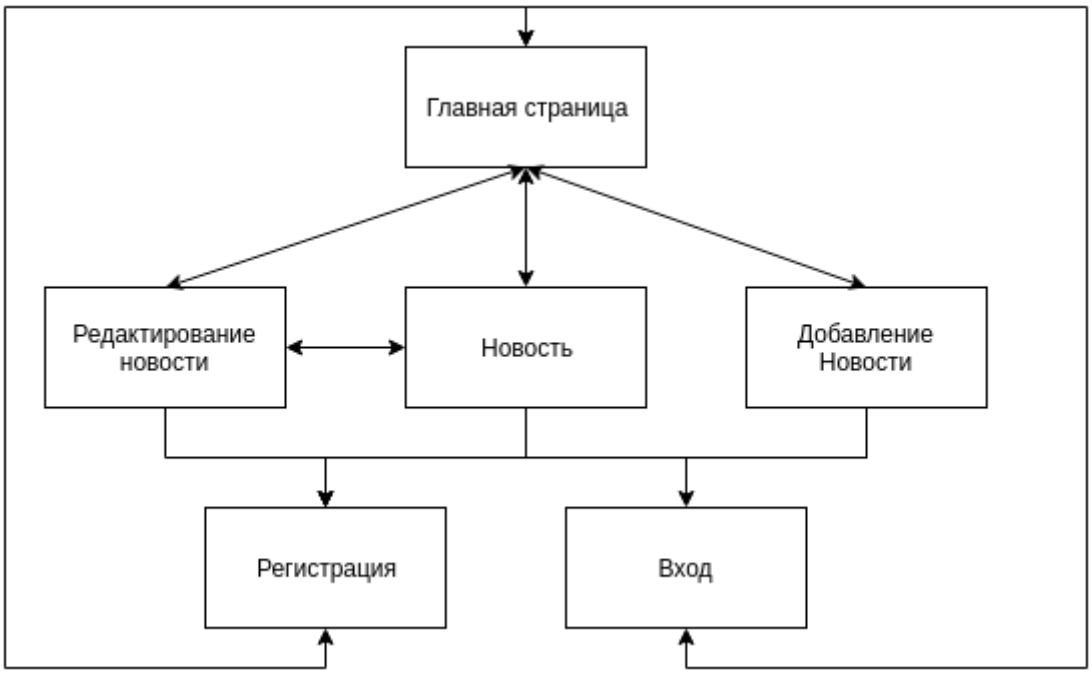
Обычный пользователь:

- все права посетителя
- оставлять комментарии

Администратор

- все права обычного пользователя
- создание новостей
- редактирование новостей
- удаление новостей
- удаление комментариев

Схема переходов сайта



Вывод

В ходе выполненной работы были рассмотрены основные методы атак на Web приложение.

Атака переполнения буфера ранее являлась весьма популярной. В настоящее время, с распространением более высокоуровневых языков программирования и улучшения методов управлению выделяемой памятью, распространенность данной уязвимости снизилась. Однако при разработке программного продукта забывать о ней не следует.

SQL инъекции несут серьезную угрозу конфиденциальности информации. Данный вид атаки легко воспроизводить на незащищенных приложения. Современные среды разработки, в качестве решения этой проблемы предлагают обращаться к данным через промежуточное представление данных в виде объектов. Данный подход называется объектно-реляционное отображение.

DoS атака является одной из самых сложных для зашиты методов атак. Тем не менее повысить надежность приложения можно путем устранения «узких мест».

Межсайтовая подделка запроса является одной из главных уязвимостью HTTP протокола. Благодаря ей злоумышленник может украсть аккаунт пользователя или осуществлять от его имени нежелательные действия. Защита от данного типа атаки яляется достаточно простой.

Кроме этого необходимо правильно организовывать внутреннюю логику сайта. Следует с осторожностью выдавать пользователям привилегии на выполнения действий на сайте. Поэтому на многих сайтах вводятся сложные системы прав, которые сводят риск повреждения данных к минимуму.

Также в ходе выполненной работы было реализовано приложение, обладающее защитой от данных видов атак.

Исходный код

```
import os

from django.core.wsgi import get_wsgi_application

os.environ.setdefault("DJANGO_SETTINGS_MODULE", "news2.settings")

application = get_wsgi_application()
"""news2 URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/2.0/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path, include
from django.conf.urls.static import static
from django.conf import settings


urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('main.urls')),
]

urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
"""
Django settings for news2 project.

Generated by 'django-admin startproject' using Django 2.0.4.

For more information on this file, see
https://docs.djangoproject.com/en/2.0/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/2.0/ref/settings/
"""

import os

# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/2.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'cna=lum&x)09x)-9n-gf#_v2!24ct$d@9f4pz*0hc&g2y#5dv7'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
```

```
    'main',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'news.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'news.wsgi.application'


# Database
# https://docs.djangoproject.com/en/2.0/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}


# Password validation
# https://docs.djangoproject.com/en/2.0/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]


# Internationalization
# https://docs.djangoproject.com/en/2.0/topics/i18n/

LANGUAGE_CODE = 'ru-ru'

TIME_ZONE = 'Europe/Minsk'

USE_I18N = True

USE_L10N = True

USE_TZ = True
```

```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/2.0/howto/static-files/

STATIC_URL = '/static/'

MEDIA_ROOT = os.path.join(BASE_DIR, 'uploads')
MEDIA_URL = '/media/'


EMAIL_HOST = 'smtp.mail.ru'
EMAIL_PORT = 2525
EMAIL_HOST_USER = "brychdaneel@mail.ru"
EMAIL_HOST_PASSWORD = "20DaNeEl07"
EMAIL_USE_TLS = True
#!/usr/bin/env python3
import os
import sys

if __name__ == "__main__":
    os.environ.setdefault("DJANGO_SETTINGS_MODULE", "news.settings")
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)
from django.db import models
from main.models.news import News
from django.contrib.auth.models import User

class Coments(models.Model):

    class Meta:
        db_table = 'comments'
        verbose_name = 'коментарий'
        verbose_name_plural = 'коментарии'
        ordering = ['pubtime']

    new = models.ForeignKey(News, editable=False, on_delete=models.CASCADE)

    user = models.ForeignKey(
                             User,
                             verbose_name='Пользователь',
                             editable=False,
                             on_delete=models.SET_NULL,
                             blank=True,
                             null=True
                            )

    text = models.TextField(verbose_name='Коментарий')

    pubtime = models.DateTimeField(
                                    auto_now_add=True,
                                    editable=False,
                                    verbose_name='Дата коментирования'
                                    )

    def __str__(self):
        return  '{} к новости "{}" ({:%d.%m.%Y %H:%M})'\
                .format(self.user, self.new, self.pubtime)
from django.db import models
from django.core.validators import FileExtensionValidator
from django.conf import settings
from PIL import Image
import os
import uuid
import datetime


IMAGE_W = 640
IMAGE_H = 360

def image_name():
    cfg = {}
    cfg['root'] = 'news'
```

```
    cfg['now'] = datetime.datetime.now()
    cfg['rnd'] = uuid.uuid4()
    cfg['ext'] = 'jpg'

    return  '{root}/{now:%Y/%m/%d}/{rnd}.{ext}'.format(**cfg)

class News(models.Model):

    class Meta:
        db_table = 'news'
        verbose_name = 'Новость'
        verbose_name_plural = 'Новости'
        ordering = ['-pubtime']

    title = models.CharField(max_length=50, verbose_name='Заголовок')

    image = models.ImageField(
                            upload_to='temp',
                            verbose_name='Изображение'
                           )

    short_text = models.TextField(verbose_name='Краткое описание')

    text = models.TextField(verbose_name='Тескт')

    pubtime = models.DateTimeField(
                                   auto_now_add=True,
                                   editable=False,
                                   verbose_name='Дата публикации'
                                   )

    def __str__(self):
        return self.title

    def save(self, *args, **kwargs):
        if News.objects.filter(pk=self.pk).exists():
            try:
                this_record = News.objects.get(pk=self.pk)
                if this_record.image != self.image:
                    this_record.image.delete(save=False)
            except:
                print("WARNING: Can't delete file {}".format(thisself_record.image.path))

        super(News, self).save(*args, **kwargs)

        im = Image.open(self.image.path)
        im = im.resize((IMAGE_W, IMAGE_H))
        new_path = image_name()
        full_path = os.path.join(settings.MEDIA_ROOT, new_path)
        os.makedirs(os.path.split(full_path)[0], exist_ok=True)
        im.save(full_path)
        os.remove(self.image.path)

        self.image.name = new_path
        super(News, self).save(*args, **kwargs)
        print('ok')


    def delete(self, *args, **kwargs):
        try:
            this_record.image.delete(save=False)
        except:
            pass
        super(News, self).delete(*args, **kwargs)
from django.db import models
from django.contrib.auth.models import User
from uuid import uuid4

class EmailConfirm(models.Model):

    user = models.OneToOneField(User, on_delete=models.CASCADE)

    token = models.CharField(default=uuid4, max_length=36)
from main.models.news import News
from main.models.coments import Coments
from main.models.email_confirm import EmailConfirm
from django.contrib import admin
```

```
from main.models import News
from main.models import Coments

admin.site.register(News)
admin.site.register(Coments)
from django.views.generic.edit import CreateView
from main.models.news import News
from django.urls import reverse_lazy

class CreateNewView(CreateView):
    model = News
    fields = ['title', 'image', 'short_text', 'text']
    template_name = 'create.html'
    success_url = reverse_lazy('main')
from django.views.generic.base import View
from django.contrib.auth import logout
from django.shortcuts import redirect


class CustomLogoutView(View):

    def get(self, request, *args, **kwargs):
        logout(request)
        return redirect('main')
from django.views.generic.edit import FormView
from django.urls import reverse_lazy
from django.urls import reverse
from django.conf import settings
from django.contrib.auth.models import Group
from django.contrib.auth import logout

from main.forms import RegisterForm

from main.models import EmailConfirm


EMAIL_SUBJECT = 'Подтвердение почты'
EMAIL_TEXT = '{}'
DEFAULT_GROUP = 'User'

class CreateUserView(FormView):
    form_class = RegisterForm
    template_name = 'register.html'
    success_url = reverse_lazy('main')

    def form_valid(self, form, request):
        user = form.save(commit=False)
        user.is_active = False
        confirm = EmailConfirm()

        link = reverse('email.confirm', kwargs={'token' : confirm.token})
        url = request.build_absolute_uri(link)
        user.email_user(
                        EMAIL_SUBJECT,
                        EMAIL_TEXT.format(url),
                        from_email=settings.EMAIL_HOST_USER
                       )
        user.save()
        confirm.user = user
        group = Group.objects.get(name=DEFAULT_GROUP)
        user.groups.add(group)
        confirm.save()
        return super(CreateUserView, self).form_valid(form)

    def post(self, request, *args, **kwargs):
        form_class = self.get_form_class()
        form = self.get_form(form_class)
        if form.is_valid():
            return self.form_valid(form, request)
        else:
            return self.form_invalid(form)

    def get(self, request, *args, **kwargs):
        logout(request)
        return super(CreateUserView, self).get(self, request, *args, **kwargs)
from django.views.generic.list import ListView

from main.models import News
```

```
class MainView(ListView):
    template_name = 'main.html'
    model = News
    context_object_name = 'news'
from django.views.generic.edit import DeleteView
from main.models import Coments
from django.urls import reverse

class DeleteComentView(DeleteView):
    model = Coments
    template_name = 'coments_confirm_delete.html'
    context_object_name = 'coment'

    def get_success_url(self):
        pk = self.get_object().new.pk
        return reverse('detail', kwargs={'pk' : pk})
from django.views.generic.base import TemplateView

from main.models import EmailConfirm


class EmailConfirmView(TemplateView):

    template_name = 'email_confirm.html'

    def get_context_data(self, token, **kwargs):
        context = super().get_context_data(**kwargs)

        confirm = EmailConfirm.objects.filter(token=token)
        exst = confirm.exists()
        context['valid'] = exst

        if exst:
            confirm = confirm.get()
            user = confirm.user
            user.is_active = True
            confirm.delete()
            user.save()

        return context
from django.views.generic.list import ListView

from django.contrib.auth.models import User

class UsersView(ListView):
    template_name = 'users.html'
    model = User
    context_object_name = 'users'
from django.contrib.auth.views import LoginView
from django.contrib.auth import logout
from django.urls import reverse


class CustomLoginView(LoginView):

    template_name = 'login.html'

    def get(self, request, *args, **kwargs):
        logout(request)
        return super(CustomLoginView, self).get(self, request, *args, **kwargs)

    def get_success_url(self):
        return reverse('main')
from django.views.generic.edit import UpdateView
from main.models.news import News
from django.urls import reverse_lazy

class UpdateNewView(UpdateView):
    model = News
    fields = ['title', 'image', 'short_text', 'text']
    success_url = reverse_lazy('main')
    template_name = 'update.html'
from django.views.generic.detail import DetailView
from django.urls import reverse_lazy
from django.shortcuts import redirect
from datetime import timedelta
from datetime import datetime
```

```
from main.models import News
from main.models import Coments
from main.forms import ComentForm


COMMENT_TIME = timedelta(seconds=30)

class NewDetailView(DetailView):
    template_name = 'new.html'
    model = News
    context_object_name = 'new'
    login_url = reverse_lazy('login')

    form = None

    def get_context_data(self, **kwards):
        context = super(NewDetailView, self).get_context_data(**kwards)
        context['coment_form'] = self.form if self.form else ComentForm()
        context['coments'] = Coments.objects.filter(new=self.get_object())
        return context

    def post(self, request, *args, **kwargs):

        if not request.user.has_perm('main.add_coments'):
            return redirect(self.login_url)

        coment_form = ComentForm(request.POST)

        if not coment_form.is_valid():
            self.form = coment_form
            return super(NewDetailView, self).get(request, *args, **kwargs)

        coment = coment_form.save(commit=False)
        coment.user = request.user
        coment.new = self.get_object()

        if Coments.objects.filter(user=coment.user,
                                  pubtime__gte=datetime.now()-COMMENT_TIME
                                 ).exists():
            self.form = coment_form
            self.form.add_error('text',
                                "Нельзя отправлять комментарии чаще 30 секунд"
                                )
            return super(NewDetailView, self).get(request, *args, **kwargs)

        coment.save()

        return super(NewDetailView, self).get(request, *args, **kwargs)
from django.views.generic.edit import DeleteView
from main.models.news import News
from django.urls import reverse_lazy

class DeleteNewView(DeleteView):
    model = News
    success_url = reverse_lazy('main')
    template_name = 'news_confirm_delete.html'
    context_object_name = 'new'
from main.views.main import MainView
from main.views.detail import NewDetailView
from main.views.create import CreateNewView
from main.views.delete import DeleteNewView
from main.views.update import UpdateNewView
from main.views.create_user import CreateUserView
from main.views.delete_coment import DeleteComentView
from main.views.users import UsersView
from main.views.email_confirm import EmailConfirmView
from main.views.login import CustomLoginView
from main.views.logout import CustomLogoutView
from django import forms
from django.contrib.auth.models import User
from django.contrib.auth.password_validation import MinimumLengthValidator
from django.contrib.auth.password_validation import UserAttributeSimilarityValidator
from django.contrib.auth.password_validation import CommonPasswordValidator
from django.contrib.auth.password_validation import NumericPasswordValidator

from django.utils.translation import gettext as _
```

```
class RegisterForm(forms.ModelForm):

    error_messages = {
        'password_mismatch': _("The two password fields didn't match."),
        'email_occupation' : 'Данный адрес почты уже используется',
    }

    class Meta:
        model = User
        fields = ['username', 'first_name', 'last_name', 'email']


    def __init__(self, *args, **kwargs):
        super(RegisterForm, self).__init__(*args, **kwargs)
        for key in self.fields:
            self.fields[key].required = True

    password1 = forms.CharField(label=_("Password"),
            widget=forms.PasswordInput,
            validators=[
                        MinimumLengthValidator(min_length=8).validate,
                        UserAttributeSimilarityValidator().validate,
                        CommonPasswordValidator().validate,
                        NumericPasswordValidator().validate
                       ])

    password2 = forms.CharField(label=_("Password confirmation"),
        widget=forms.PasswordInput,
        help_text=_("Enter the same password as above, for verification.")
        )

    def clean_password2(self):
        password1 = self.cleaned_data.get("password1")
        password2 = self.cleaned_data.get("password2")
        if password1 and password2 and password1 != password2:
            raise forms.ValidationError(
                self.error_messages['password_mismatch'],
                code='password_mismatch',
            )
        return password2

    def clean_email(self, *args, **kwargs):
        email = self.cleaned_data.get("email")
        if User.objects.filter(email=email).exists():
            raise forms.ValidationError(
                self.error_messages['email_occupation'],
                code='email_occupation',
            )
        return email

    def save(self, commit=True):
        user = super(RegisterForm, self).save(commit=False)
        user.set_password(self.cleaned_data["password1"])
        if commit:
            user.save()

        return user
from django.forms import ModelForm
from main.models import Coments

class ComentForm(ModelForm):
    class Meta:
        model = Coments
        fields = ['text']
from main.forms.coment import ComentForm
from main.forms.register import RegisterForm
from django.conf.urls import url

from django.contrib.auth.decorators import permission_required
from django.urls import reverse_lazy

from main.views import MainView
from main.views import NewDetailView
from main.views import CreateNewView
from main.views import DeleteNewView
from main.views import UpdateNewView
from main.views import CreateUserView
from main.views import DeleteComentView
```

```
from main.views import UsersView
from main.views import EmailConfirmView
from main.views import CustomLoginView
from main.views import CustomLogoutView


login_url = reverse_lazy('login')


urlpatterns = [
    url(r'^$', MainView.as_view(), name='main'),

    url(r'^new/(?P<pk>\d+)/$', NewDetailView.as_view(), name='detail'),

    url(r'^new/add',
        permission_required('main.add_news', login_url=login_url)(CreateNewView.as_view()),
        name='new.add'),

    url(r'^new/(?P<pk>\d+)/delete/$',
        permission_required('main.delete_news', login_url=login_url)(DeleteNewView.as_view()),
name='new.delete'),

    url(r'^new/(?P<pk>\d+)/update/$',
        permission_required('main.change_news', login_url=login_url)(UpdateNewView.as_view()),
        name='new.update'),

    url(r'^login/$', CustomLoginView.as_view(), name='login'),

    url(r'^logout/$', CustomLogoutView.as_view(), name='logout'),

    url(r'^register/$', CreateUserView.as_view(), name='register'),

    url(
        r'^comments/delete/(?P<pk>\d+)$',
        permission_required('main.delete_coments', login_url=login_url)(DeleteComentView.as_view()),
        name='coment.delete'
       ),

    url(r'^users/$', UsersView.as_view(), name='users'),

    url(r'^users/confirm/(?P<token>'
        r'[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}'
        r')$',
        EmailConfirmView.as_view(),
        name='email.confirm'),
]
from django.test import TestCase

# Create your tests here.


form
    font-size: 1.2em
    color: rgba(0, 0, 0, 0.3)

    p
        margin-bottom: 30px

    label
        display: inline-block
        width: 200px
        text-align: right
        margin-right: 10px
        font-weight: bold
        color: rgba(0,0,0,.5)

    input, textarea
        border-radius: 5px
        border: 1px solid rgba(0, 0, 0, 0.3)
        background-color: rgba(255,255,255,.5)
        color: rgba(0,0,0,0.6)
        font-size: 1.2em
        box-shadow: inset 0 0 10px rgba(255,255,255,.75)
        width: 300px
        padding: 5px

    input:focus, textarea:focus
        background-color: white
```

```
        color: black

    textarea
        width: 700px
        height: 200px
        margin: 0 30px
        margin-bottom: -20px

    .helptext
        display: block
        margin-left: 100px

    .errorlist
        color: red

    input[type="submit"]
        margin-left: 220px
        padding: 10px 25px
        font-size: 1em
        color: white
        background: rgba(0,102,153, 0.7)
        border-radius: 5px
        border: rgba(0,102,153, 0.6)
        width: 300px

        &:hover
            box-shadow: 0 0 3px 3px rgba(0,102,153, 0.2)

header
    display: flex
    color: white
    width: 100%
    justify-content: space-around
    align-items: center
    background: rgba(50, 50, 50, 0.9)
    position: fixed
    top: 0
    left: 0
    height: 70px

    .logo
        float: left
        box-sizing: border-box
        height: 80%


    h1
        text-transform: uppercase
        font-size: 3em

        a
            color: white
            text-decoration: none

    .login
        float: right
        margin: auto 0
        font-size: 1.5em

        span
            text-transform: uppercase
            margin-right: -10px

        a
            color: white
            border: 4px solid rgba(30,130,170, 1)
            border-radius: 15px
            padding: 5px 10px
            margin-left: 15px
            text-decoration: none

            &:hover
                background: rgba(30,130,170, 1)
body
    background: url("../images/background.png") repeat


.pageWrapper
```

```
    max-width: $page_heaght
    margin: 0 auto

    background: rgba(255, 255, 255, 0.8)
    border-radius: 50px
    box-shadow: 0 0 10px 5px rgba(30, 30, 30, 0.8)
footer

    background: rgba(0, 0, 0, 0.7)
    color: white
    border-radius: 0 0 50px 50px
    padding: 30px

    .info
        max-width: $page_heaght
        margin: 0 auto
        display: flex
        justify-content: space-between
        align-items: flex-start

    .logo
        height: 70px

    .logo img
        height: 100%
        margin-bottom: 5px

    .copy
        font-size: 0.8em

    small
        padding: 0 20px 0 20px
        margin-left: 30px

    .payments
        box-sizing: border-box
        clear: both
        display: block
        margin: 0 auto
        padding: 20px

main
    order: 1
    padding: 20px 50px
    margin-top: 100px


    .new
        border-radius: 20px
        padding: 0 80px 30px 80px
        margin: 50px 20px
        box-shadow: 0 0 10px
        background: rgba(200,200,200, 0.1)

        h2
            text-align: left
            font-size: 2.5em
            color: rgba(0, 0, 0, 0.7)
            margin-top: 10px
            text-shadow: 2px 2px 3px rgba(0,0,0,0.3)

        img
            margin: 30px auto
            border-radius: 20px
            display: block
            box-shadow: 0 0 30px

        .shorttext
            font-size: 1.5em
            margin-bottom: 20px

        .time
            float: right
            font-size: 1.5em
            color: rgba(30,130,170, 1)

        a
            font-size: 1.5em
```

```
    .details

        padding: 0 50px

        h1
            font-size: 5em
            color: rgba(100, 100, 100, 0.9)
            text-align: center

        img
            margin: 50px auto
            border-radius: 20px
            display: block
            box-shadow: 0 0 30px

        .text
            font-size: 1.5em
            margin-bottom: 30px

        .time
            float: right
            font-size: 1.5em
            color: rgba(30,130,170, 1)

        a
            font-size: 1.5em
            margin-right: 50px


    .comments

        padding: 0 50px
        margin-top: 50px

        h2
            font-size: 2em
            color: rgba(100, 100, 100, 0.9)
            text-align: left

        .comment

            margin: 40px 0
            border: 2px solid rgba(30,130,170, 0.8)
            border-radius: 10px
            box-shadow: 0 0 5px rgba(30,130,170, 0.8)

            .name
                font-size: 1.5em
                background: rgba(30,130,170, 0.8)
                border-radius: 5px 5px 0 0
                padding: 10px
                font-size: 1.5em

            .text
                font-size: 1.5em
                margin: 10px 0
                padding: 10px

            .time
                font-size: 1.3em
                color: rgba(30,130,170, 1)
                padding: 10px

            a
                font-size: 1.5em
                padding: 10px
                text-align: center
$page_heaght: 1000px

{% extends "base.html" %}

{% block title %} Delete::{{ new.title }} {% endblock %}


{% block content %}
```

```
<form action="" method="post">
    {% csrf_token %}
    <p>Подтверждаете удаление новости "{{ object }}"?</p>
    <input type="submit" value="Удалить" />
</form>

{% endblock %}
{% extends "base.html" %}

{% block title %} Delete::Comment {% endblock %}


{% block content %}

<p>Подтверждаете удаление коментария {{ coment }} :</p>
<p>{{ coment.text }} ?</p>

<form action="" method="post">
    {% csrf_token %}
    <input type="submit" value="Удалить" />
</form>

{% endblock %}
{% extends "base.html" %}

{% block title %}Регистрация{% endblock %}

{% block content %}

<form action="" method="post">
{% csrf_token %}
{{ form.as_p }}
<input type="submit" value="Регистрация">
</form>

{% endblock %}
{% extends "base.html" %}

{% block title %} Подтвердение почты {% endblock %}


{% block content %}

{%if valid %}
подтвержденно
{% else %}
неверный токен
{% endif %}

{% endblock %}
{% extends "base.html" %}

{% block title %}Вход{% endblock %}

{% block content %}

<form action="" method="post">
{% csrf_token %}
{{ form.as_p }}
<input type="submit" value="Вход">
</form>

{% endblock %}
{% extends "base.html" %}

{% block title %} {{ new.title }} {% endblock %}


{% block content %}

<form action="" method="post" enctype="multipart/form-data">

{{ form.as_p }}

{% csrf_token %}

<input type="submit" value="обновить">
```

```
</form>

{% endblock %}
{% extends "base.html" %}

{% block title %} Пользователи {% endblock %}

{% block content %}

{% for user in users %}
    <p>{{ user }}</p>
{% endfor %}


{% endblock %}

{% extends "base.html" %}

{% block title %} Главная {% endblock %}


{% block content %}

{% for new in news %}

<div class="new">
    <h2>{{ new.title }}</h2>
    <img src="{{ new.image.url }}">
    <div class="shorttext"> {{ new.short_text|linebreaks }} </div>
    <p class="time">{{ new.pubtime }}</p>
    <a href={% url 'detail' pk=new.pk %}>Подробнее</a>

    {% if perms.main.change_news %}
    <a href={% url 'new.update' pk=new.pk %}>Обновить</a>
    {% endif %}

    {% if perms.main.delete_news %}
    <a href={% url 'new.delete' pk=new.pk %}>Удалить</a>
    {% endif %}

</div>

{% endfor %}

{% if perms.main.add_news %}
<a href={% url 'new.add' %}>Добавить новость<a>
{% endif %}

{% endblock %}
<!doctype html>

<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>{% block title %}{% endblock %}</title>
        <link href="/static/css/styles.css" rel="stylesheet" media="all" />
    </head>
        <body>
            <div class="pageWrapper">

            <header>
            {% include "components/header.html" %}
            </header>

            <main>
            {% block content %}{% endblock %}
            </main>


            <footer>
            {% include "components/footer.html" %}
            </footer>

            </div>
        </body>
</html>
{% extends "base.html" %}
```

```
{% block title %} {{ new.title }} {% endblock %}


{% block content %}

<div class="details">

<h1>{{ new.title }}</h1>
<img src="{{ new.image.url }}">
<div class="text">{{ new.text|linebreaks }}</div>
<p class="time">{{ new.pubtime }}</p>

{% if perms.main.change_news %}
<a href={% url 'new.update' pk=new.pk %}>Обновить</a>
{% endif %}

{% if perms.main.delete_news %}
<a href={% url 'new.delete' pk=new.pk %}>Удалить</a>
{% endif %}
</div>


<div class="comments">
<h2>Коментарии</h2>
{% for comment in coments %}

<div class="comment">
<p class="name">{{ comment.user.get_username }}</p>
<p class="text">{{ comment.text }}</p>
<p class="time">{{ comment.pubtime }}</p>

{% if perms.main.delete_coments %}
<a href={% url 'coment.delete' pk=comment.pk %}>Удалить коментарий</a>
{% endif %}

</div>

{% endfor %}


{% if perms.main.add_coments %}
<form action="" method="post">
{% csrf_token %}
{{ coment_form.as_p }}
<input type="submit" value="Коментировать">
</form>
{% endif %}

</div>

{% endblock %}
<img src="/static/images/logo.png" class="logo" alt="logo">

<h1><a href="{% url "main" %}">Новости</a></h1>

<div class="login">
{% if not user.is_authenticated %}
<a href="{% url "login" %}">Войти</a>
<a href="{% url "register" %}">Регистрация</a>
{% else %}
<span>{{ user.username }}</span>
<a href="{% url "logout" %}">Выход</a>
{% endif %}

</div>
<div class="info">


<figure class="logo">
<img src="/static/images/logo.png">
<figcaption class="copy">
&copy; 2018 ЗАО "Крутая компания"
</figcaption>
</figure>

<small>
Свидетельство о регистрации №123456789 от 01.01.2000г.<br>
Лицензия на розничную торговлю №00000/00000 от 01.01.2000г.<br>
```

```
Дата включения в торговый реестр – 01.01.2000г.
</small>

<div class="contact">

<address>
emal: mega@cool.net<br>
MTC: +375 11 111 11 11<br>
VECOME: +375 22 222 22 22<br>
LIFE: +375 33 333 33 33
</address>

</div>

</div>
{% extends "base.html" %}

{% block title %} {{ new.title }} {% endblock %}


{% block content %}

<form action="" method="post" enctype="multipart/form-data">
{% csrf_token %}
{{ form.as_p }}
<input type="submit">
</form>

{% endblock %}
```