РАЗРАБОТКА СКАНЕРА ПОТЕНЦИАЛЬНЫХ CSRF УЯЗВИМОСТЕЙ

Коровин Кирилл Сергеевич

студент Московского института электроники и математики НИУ ВШЭ, РФ, г. Москва

CSRF (Cross Site Request Forgery – Межсайтовая подделка запроса) – один из самых распостраненных типов атак на веб-приложения, использующий клиентов этих приложений в качестве вектора атаки. Эта уязвимость входит в авторитетный список наиболее распространённых уязвимостей OWASP Top 10 под номером 8. первые рассуждения на тему возможности подобных атак возникли еще в 1988 году, а первые подобные уязвимости были обнаружены в 2000 году. Одна из опасностей, которую несет с собой CSRF уязвимость – это то, что данная атака основана не на ошибках в протоколе HTTP или реалиации приложения, а на недочётах этого протокола, то есть, с точки зрения протокола и поведения браузера все выглядит легально.

CSRF атака становится возможной при отсутствии проверкой самим приложением валидности приходящего запроса, гарантирующей, что запрос к серверу пришел из соответствующего места. Эта уязвимость критична, так как при отсутствии защитных механизмов со стороны сервера эксплатируемого приложения достаточно лишь того, чтобы пользователь, подвергнутый атаке был авторизован на этом сервере (например, через файлы куки). Наиболее часто приводимыми примерами является несанционированное отправление электронных писем без ведома пользователя или операции с банковским счётом. Несмотря на то, что абсолютное большинство как почтовых сетей, так и банковских, сейчас так или иначе защищены от подобного рода атак, все же это является наиболее наглядным примером. Для того, чтобы полностью представить картину, предлагаю рассмотреть этот пример подробнее.

Типичный пример, запроса, отправляемого на сервер при перечислении средств, приведен на рисунке 1:

```
http://bank.example.com/withdraw?account=<victim acc id>&amount=1000&to<attacker acc id>
```

Рисунок 1. Пример GET-запроса при перечислении средств

Подразумевается, что жертва уже авторизована на сервере банковского приложения, и, скорее всего, эта информация сохранена в файлах куки. В таком случае, злоумышленнику даже не нужно вынуждать пользователя открывать ссылку, достаточно лишь, чтобы пользователь открыл страницу, содержащую эту ссылку например в теге <img> в качестве значения аттрибута «src» (рис. 2)

```
Эй, смотри какой котик! <img src='http://bank.example.com/withdraw?account=<victim acc id>&amount=1000&to<attacker acc id>'/>
```

Рисунок 2. Текст вредоносного e-mail

Несмотря на то, что по этой ссылке не будет изображения, при открытии жертвой письма, браузер попытается его загрузить и сделает запрос по ссылке указанной в аттрибуте «src» (рис. 3). Предполагая, что никаких защитных механизмов нет и пользователь уже авторизован на сервере банка, запрос будет исполнен.

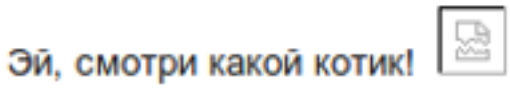

Рисунок 3. Пример отображения вредносного e-mail

Есть несколько распостраненных способов защиты от подобного рода атак:

· Проверка поля HTTP Refferer.

· Некоторые тесты Тьюринга, например, CAPTCHA.

· Выдача CSRF токена (уникального ключа) на каждый запрос к серверу и проверка по нему.

Первый способ довольно опасен, так как значение заголовка легко подделать вместе с запросом, поэтому он практически не используется. Второй способ намного надежнее первого, однако в наше время появились сервисы, предлагающие автоматическую разгадку «капчи», и злоумышленнику ничего не стоит ими воспользоваться для того, чтобы добавлять в свои запросы код подтверждения капчи. Остается третий способ, при котором сервер добавляет на страницу среди прочих параметров формы скрытое поле с уникальным токеном. При каждом новом запросе к странице, этот токен тоже обновляется. При отправке формы, этот ключ отправляется вместе с остальными данными на сервер, который проверяет, валиден ли ключ, и только после этого обрабатывает данные. Данный способ в большинстве случаев является стопроцентной защитой от CSRF атаки при условии правильной реализации.

К сожалению, все популярные и используемые методы обнаружения CSRF уязвимостей являются активными, то есть подразумевают попытку эксплуатирования самой уязвимости для того, чтобы её обнаружить. Во многих случаях это не является безопасным методом, так как если уязвимость есть, то её эксплуатирование может привести к неожиданным и иногда даже непоправимым последствиям. Поэтому сканер должен обнаруживать формы, для которых нет явного способа защиты. Безусловно, это значит, что может быть большой процент ложных срабатываний, но безопасность сканирования стоит больше. Сам же сканер реализован в виде скрипта написанного на языке Python с использованием библиотек urllib и BeautifulSoup. Сканер также должен иметь возможность полного сканирования сайта и поддержку файлов куки.

Основываясь на выводах из вышесказанного, было выявлено три основных ситуации, когда форма может быть уязвима к CSRF атакам:

1. Отсутствие CSRF токена вообще.

2. Отсутствие обновления этого токена при новом запросе.

3. Возможность повторного использования токена. Это один из случаев неправильной реализации механизма защиты, который был обнаружен мной случайно во время одного из исследований, и о нём будет сказано ниже.

Итак, в этом скрипте есть три основных функции, проверяющие три вышеописанных ситуации. Первая довольно проста: создается HTTP запрос, ответ которого обрабатывается парсером (BeautifulSoup) и для каждой формы идет проверка наличия токена. Проверка производится по следующим условиям:

1. У формы есть скрытое поле.

2. В этом скрытом поле есть что-то, чья длина не меньше 10 символов.

3. Это значение проверяется на то, что оно состоит из букв и цифр, посредством соответствия регулярному выражению.

Вторая функция делает такой же запрос, находит токен по тем же критериям, что и выше. Затем, она записывает это значение во временную переменную и делает новый запрос, после чего проверяет значения этих токенов. Если он не поменялся, то программа выдает соответствующее сообщение.

Третий случай же довольно интересен. Во время тестирования одного из сайтов мной была замечена следующая ситуация: токен менялся и казалось, что его нельзя никак эксплуатировать, так как внешне он работал как обычно. Однако, начав экспериментировать с запросами по этому URL из командной строки с помощью curl было замечено, что если отключить перенаправление после отправки формы, то данные всё равно отправлялись и сообщения об ошибке не было, в то время как токен не обновлялся, так как не было нового запроса к странице с этой формой. Это означает, что на сервере при использовании токена он не удаляется из базы данных и остаётся валидным. Таким образом, я сделал вывод, что это можно попробовать эксплуатировать и это случай неправильной конфигурации выдачи CSRF токена

Именно это ситуацию и воспроизводит третья функция. Она создает два шаблона запросов – первый из них обычный, а второй не поддерживает перенаправление. Сначала отправляется обычный запрос, ответ обрабатывается и токен запоминается. После этого делается несколько одинаковых запросов с «шаблонными» неверными данными через интерфейс, не поддерживающий перенаправление, с указанным ранее найденным токеном.

После этого делается обычный запрос и снова обрабатывается с помощью парсера, чтобы найти поля с ошибками (так как данные были неверные). Если они все находятся в одном блоке и имеют одинаковый текст, значит, что данные были посланы несколько раз в разных запросах и были доставлены. С чего делается вывод, что токен может быть использован повторно и выдается соответствующее сообщение.

И в конце концов, есть общая main() функция, которая принимает входные данные (в том числе из командной строки) и по очереди передает их каждой из функций, собирая ответы и возвращая пользователям. Куки могут быть посланы с помощью значения аргумента – cookie в виде строки.

Так же реализована возможность сканирования всего сайта. Для этого был написан “crawler”, ищущий рекурсивно все ссылки, относящиеся к тому же домену. Эта функция ищет все ссылки, связанные с тем же самым доменом, который указан во входном URL и добавляет их в список, предварительно проверяя, не была ли эта ссылка уже проверена и добавлена, затем сканирует следующую ссылку из этого списка, и так далее по рекурсии. В связи с рекурсивным характером этой опции, не рекомендуется использовать её на больших сайтах с динамическим контентом, о чём сказано в разделе “help” сканера напротив этой команды.

Пример конечного результата сканирования предоставлен на рисунке 4:

Рисунок 4. Результат работы сканера

В итоге проделанной работы был реализован сканер, позволяющий пассивно просканировать узел на наличие потенциально уязвимых к CSRF атакам, реализован рекурсивный режим и поддержка куков. На данный момент – это единственный сканер CSRF уязвимостей такого рода.

Исходный код можно найти на github по следующей ссылке: https://github.com/skvoter/OVWAScanner/blob/master/utils/crosser.py.

Список литературы:

1. Низамутдинов М.Ф. Тактика защиты и нападения на Web-приложения. СПб: БХВ-Петербург, 2005.

2. The Cross-Site Request Forgery (CSRF/XSRF) FAQ. URL: http://www.cgisecurity.com/csrf-faq.html (дата обращения 15.06.2017).

3. Kali Linux Tools Listing. URL: http://tools.kali.org/tools-listing (дата обращения 15.06.2017).

4. Open Web Application Security Project (OWASP). URL: https://www.owasp.org (дата обращения 19.06.2017).