Основы веб-хакинга

Как зарабатывать деньги этичным хакингом

Анализ более чем 30 оплаченных отчетов!

Питер Яворски

Перевод: Евгений Бурмакин

Основы веб-хакинга

Более 30 примеров уязвимостей

Peter Yaworski и Eugene Burmakin

Эта книга предназначена для продажи на
http://leanpub.com/white-hat-hacking-ru

Эта версия была опубликована на 2020-03-19

Это книга с Leanpub book. Leanpub позволяет авторам и издателям участвовать в так называемом Lean Publishing - процессе, при котором электронная книга становится доступна читателям ещё до её завершения. Это помогает собрать отзывы и пожелания для скорейшего улучшения книги. Мы призываем авторов публиковать свои работы как можно раньше и чаще, постепенно улучшая качество и объём материала. Тем более, что с нашими удобными инструментами этот процесс превращается в удовольствие.

Оглавление

Вступительное слово **1**

Введение **3**

Необходимые знания **15**

Уязвимости Открытого Редиректа (Open Redirect) . . **19**
Описание 19
Примеры 20
Итоги 26

HTTP Parameter Pollution **28**
Описание 28
Примеры 33
Итоги 40

Вступительное слово

Лучший способ научиться чему-либо — просто начать этим заниматься. Именно так мы — Майкл Принс и Джоберт Абма — научились хакингу.

Мы были молодыми. Как и все хакеры до нас, и все, кто будет после нас, нас вело неконтролируемое, жгучее желание понимать, как работают вещи. В основном мы играли в компьютерные игры, а к 12 годам решили научиться создавать софт самостоятельно. Мы научились программировать на Visual Basic и PHP по библиотечным книгам и на практике.

Из нашего понимания разработки программного обеспечения мы быстро обнаружили, что эти навыки позволяют нам находить ошибки других разработчиков. Мы перешли от создания к разрушению и с тех пор хакинг стал нашей страстью. Чтобы отпраздновать наш выпуск из старшей школы, мы захватили эфир телеканала, пустив в него поздравительный ролик для нашего выпускного класса. Хотя в то время это было весело, мы быстро поняли, что последствия неизбежны и это не тот вид хакеров, которые нужны миру. Телеканал и школа не были в восторге и мы провели все лето за мытьем окон, которое стало нашим наказанием. В колледже мы обратили наши навыки в жизнеспособный консалтинговый бизнес, который, на своем пике, имел клиентов в публичном и частном секторах по всему миру. Наш хакинговый опыт привел нас к HackerOne, компании, которую мы вместе основали в 2012. Мы хотели позволить каждой компании во вселенной успешно работать с хакерами и это продолжает быть миссией HackerOne и по сей день.

Если вы читаете это, значит в вас есть то же любопытство, необходимое для того, чтобы быть хакером и искателем багов.

Мы верим, что эта книга будет потрясающим руководством на протяжении всего вашего пути. Она полна полноценных примеров отчетов об уязвимостях из реального мира, эти отчеты принесли их авторам реальные деньги. Так же в этой книге вы найдете полезный анализ и обзор от Питера Яворски, автора и хакера. Он ваш помощник на пути обучения, и это бесценно.

Еще одна причина, по которой эта книга так важна, заключается в том, что она фокусируется на том, как стать этичным хакером. Освоение искусства хакинга может быть чрезвычайно мощным навыком, который, мы надеемся, будет использован во благо. Самые успешные хакеры умеют во время хакинга балансировать на тонкой линии между правильным и неправильным. Многие люди могут ломать вещи, и даже пытаются извлечь из этого быструю выгоду. Но только представьте, вы можете сделать Интернет безопаснее, работать с потрясающими компаниями со всего мира и даже получать за все это деньги. Ваш талант потенциально может сохранить миллиарды людей и их данные в безопасности. Мы надеемся, что вы вдохновляетесь именно этим.

Мы бесконечно благодарны Питу за время, которое он потратил на то, чтобы задокументировать все описанное с такой тщательностью. Нам хотелось бы иметь подобный источник знаний в те дни, когда мы только начинали свой путь. Книгу Пита приятно читать благодаря информации, которая поможет успешно начать путь хакера.

Приятного чтения и успешного хакинга!

И не забывайте применять свой навык с ответственностью.

Майкл Принс и Джоберт Абма, основатели HackerOne

Введение

Спасибо за покупку этой книги, я надеюсь, вы испытаете такое же удовольствие от чтения, какое испытывал я при проведении исследований и её написании.

“Основы веб-хакинга” — моя первая книга, и она предназначена помочь вам ступить на путь хакера. Я начал писать её как самиздатовское объяснение 30 уязвимостей, побочный продукт моего собственного обучения. Она быстро превратилась в нечто значительно большее.

Я надеюсь, что эта книга, как минимум, откроет ваши глаза на огромный мир хакинга. В лучшем случае, мне хочется надеяться, что это будет вашим первым шагом по направлению к тому, чтобы сделать веб более безопасным местом и заодно получить за это некоторое вознаграждение.

Как все это началось

В конце 2015 я наткнулся на книгу Парми Олсон *We Are Anonymous: Inside the Hacker World of LulzSec, Anonymous and the Global Cyber Insurgency* и прочитал её за неделю. Однако, закончив чтение, я так и не понял, как эти хакеры начали свой путь.

Я жаждал большего, но я не просто хотел знать **ЧТО** делают хакеры, я хотел знать, **КАК** они это делают. Поэтому я продолжил читать. Но каждый раз, когда я заканчивал читать следующую книгу, по-прежнему оставались без ответов следующие вопросы:

- Как другие хакеры узнают об уязвимостях, которые они находят?

- Где люди находят уязвимости?
- Как хакеры начинают процесс хакинга целевого сайта?
- Хакинг — это просто использование автоматических инструментов?
- Как я могу начать находить уязвимости?

Но в поисках ответах лишь открывались все новые и новые двери.

Примерно в то же время я проходил на Coursera курс по разработке для Android и искал другие интересные курсы. Мне на глаза попался курс Coursera Cybersecurity, в частности, Курс 2, Software Security. К моей удаче, курс только начался (на февраль 2016 он в состоянии “Coming Soon”) и я записался на обучение.

После нескольких лекций я наконец понял, что такое переполнение буфера и как его использовать. Я полностью ухватил принцип эксплуатации SQL-инъекций, о которых я раньше знал лишь то, что они опасны. Короче говоря, меня зацепило. До этого момента я всегда подходил к безопасности веб-приложений с точки зрения разработчика, понимая необходимость экранирования значений и избегая нефильтрованного пользовательского ввода. Теперь я начал понимать, как все это выглядело с точки зрения хакера.

Я продолжал искать информацию о том, как взламывать и попал на форумы Bugcrowd. К сожалению, активность там была довольно слабой, но кто-то упомянул хакерскую активность на HackerOne и дал ссылку на отчет. Пройдя по ссылке, я испытал восторг. Я читал описание уязвимости, написанное для компании, которая раскрыла его миру. Возможно, еще более важным было то, что компания заплатила хакеру за то, что он нашел уязвимость и описал её!

Это было поворотной точкой, я стал одержим. Особенно, когда узнал, что Shopify, компания из моей родной Канады, была

лидером по раскрытию отчетов об уязвимостях на тот момент. Посмотрев профиль Shopify, я увидел, что их профиль усыпан публичными отчетами. Я никак не мог начитаться. Уязвимости включали межсайтовый скриптинг (XSS), баги в аутентификации, CSRF, и это лишь пара примеров.

Признаю, в тот момент я пытался понять, что описывали отчеты. Некоторые уязвимости и методы их эксплуатации было тяжело понять.

Поиски в Google в попытках понять один конкретный отчет привели меня на дискуссию на Github, посвященную одной старой уязвимости, связанной с дефолтным параметром Ruby on Rails (это описано в главе "Уязвимости логики приложений"), о которой сообщил Егор Хомяков. Это имя привело меня на блог Егора, который содержит описание некоторых действительно сложных уязвимостей.

Читая о его опыте, я понял, что мир хакинга может получить пользу от объяснения реальных уязвимостей простым языком. И так уж получилось, что я учусь лучше, когда учу других.

Так появились "Основы веб-хакинга".

Всего 30 примеров и моя первая продажа

Я решил начать с простой цели, найти и объяснить простым языком 30 веб-уязвимостей, легких для понимания.

Я понял, что в худшем случае, исследование и написание текстов об уязвимостях поможет мне изучить хакинг. В лучшем случае, я продам миллион копий, стану гуру самиздата и рано уйду на пенсию. Последнее пока не случилось, а первое временами кажется неосуществимым до конца.

Примерно после 15 объясненных уязвимостей я решил опубликовать свой черновик, чтобы его можно было купить - платформа, которую я выбрал, Leanpub (через которую вы,

скорее всего, и приобрели книгу) позволяет вам публиковать итеративно, предоставляя покупателям доступ ко всем обновлениям. Я отправил твит, чтобы поблагодарить HackerOne и Shopify за их открытые отчеты и чтобы рассказать миру о своей книге. Я не ожидал ничего особенного.

Но через считанные часы первый покупатель приобрел мою книгу.

Окрыленный мыслью, что кто-то по-настоящму заплатил за мою книгу (что-то, созданное мной и чему я отдал тонны усилий), я вошел на Leanpub, чтобы узнать, могу ли я что-то узнать о своем таинственном покупателе. Ничего. Но затем мой телефон завибрировал, я получил твит от Майкла Принса, в котором он говорил, что ему понравилась книга и попросил оставаться на связи.

Кто блин такой Майкл Принс? Я проверил его профиль на Twitter и узнал, что он один из со-основателей HackerOne. **Черт**. Часть меня думала, что ребята из HackerOne не будут рады тому, что я полагаюсь на содержимое их сайта. Я попытался оставаться позитивным, Майкл казался доброжелательным и попросил оставаться на связи, что, вероятно, не должно ничем грозить.

Вскоре после первой продажи состоялась вторая и я понял, что что-то происходит. Так совпало, что примерно в то же время я получил уведомление с Quora о вопросе, который, возможно, мог бы меня заинтересовать, *Как мне стать успешным этичным хакером*?

Благодаря своему опыту начинающего, я знал, каково это, и, также ведомый эгоистичным желанием рассказать о своей книге, я решил написать ответ. Примерно на полпути я понял, что единственный ответ, кроме моего, оставил Джоберт Абма, второй из двух со-основателей HackerOne. Довольно авторитетный голос в хакинге. **Черт**.

Я уже думал не отправлять свой ответ, но решил переписать

его таким образом, чтобы основываться на ответе Джоберта, поскольку я не мог соревноваться с ценностью его совета. Я нажал “Отправить” и больше об этом не думал. Но затем я получил интересное письмо:

> Привет, Питер, я видел твой ответ на Quora и видел, что ты пишешь книгу об этичном хакинге. Я был бы рад узнать об этом больше.
>
> С наилучшими пожеланиями,
>
> Мартен CEO, HackerOne

Тройное Черт. Куча мыслей пронеслась в моей голове в этот момент, ни одна из которых не была позитивной, и практически все были нерациональными. В двух словах, я понял, что единственная причина, по которой Мартен мог написать мне, была в том, чтобы опустить кувалду на мою книгу. К счастью, это было невероятно далеко от истины.

Я ответил ему, объяснив, кто я такой и что я делаю - что я пытаюсь научиться хакингу и помочь другим в этом непростом деле. Оказалось, что ему очень нравится эта идея. Он объяснил, что HackerOne заинтересован в росте сообщества и поддерживает хакеров в их обучении, поскольку это выгодно для всех вовлеченных. В общем, он предложил помощь. И, черт, он помог. Эта книга, вероятно, не была бы в том состоянии, в котором она сегодня, или включала бы половину содержимого без постоянной поддержки и мотивации со стороны Мартена и HackerOne.

С того первого письма, я продолжал писать и Мартен продолжал интересоваться прогрессом. Майкл и Джоберт читали черновики, предлагали изменения и даже написали некоторые части текста. Мартен даже покрыл расходы на профессионально оформленную обложку (прощай, простая желтая обложка с белой ведьминской шляпой, выглядевшая так, словно тебя

рисовал четырехлетний ребенок). В мае 2016, Адам Бакус присоединился к HackerOne и на свой пятый день в компании он прочитал книгу, предложил внести правки и объяснил, каково быть по другую сторону — получателем отчетов об уязвимостях, и теперь это описано в главе о написании отчетов.

Я пишу обо всем этом потому, что на протяжении всего пути HackerOne никогда не просили ничего взамен. Они просто хотели поддержать соообщество и эта книга оказалась хорошим способом это сделать. Как для новичка в хакерском сообществе, это тронуло меня и я надеюсь, что тронет и вас. **Я, лично, предпочитаю быть частью отзывчивого и поддерживающего сообщества**.

Итак, с тех пор эта книга значительно увеличилась, став намного больше, чем я изначально рассчитывал. И с этой переменой изменилась и целевая аудитория.

Для кого написана эта книга

Я написал эту книгу, помня о тех, кто только начинает путь хакинга. Не важно, являетесь ли вы веб-разработчиком, вебдизайнером, домохозяйкой, вам 10 лет или 75. Я хочу, чтобы эта книга была надёжным справочником для понимания различных типов уязвимостей, того, как их обнаруживать, как сообщать о них, как получать за это деньги, и даже как писать код, позволяющий их предотвратить.

Таки образом, я не пишу эту книгу, чтобы обратиться к массам. На самом деле это книга о совместном обучении. А значит, я делюсь успехами **И** некоторыми своими заметными (и постыдными) неудачами.

Эта книга так же не обязательно должна быть прочитана от корки до корки, если вы нашли интересующую вас часть, свободно читайте её первой. В некоторых случаях я ссылаюсь на описанные ранее главы, но делая это, я пытаюсь связать их,

чтобы вы могли пролистать вперед и назад. Я хочу, чтобы эта книга стала чем-то, что вы держите открытым, пока занимаетесь хакингом.

Каждая глава, посвященная типам уязвимостей, структурирована одинаково:

- Начало с описанием типа уязвимости;
- Обзор примеров уязвимости; и
- Заключение с подведением итогов.

Подобным образом, каждый пример внутри этих глав структурирован в едином стиле и включает:

- Мои оценки сложности обнаружения уязвимости
- url, связанный с местом, где была обнаружена уязвимость
- Ссылку на отчет или описание
- Дату публикации отчета об уязвимости
- Сумму, выплаченную за отчет
- Понятное описание уязвимости
- Выводы, которые вы можете использовать в собственных исследованиях

Наконец, хотя это и не является обязательным требованием для хакинга, вероятно, будет хорошей идеей иметь хотя бы беглое знакомство с HTML, CSS, Javascript и, возможно, иметь некоторый опыт программирования. Это не значит, что вы должны быть способны с нуля создавать страницы, но понимание базовой структуры веб-страницы, как CSS определяет внешний вид и ощущения, и чего можно достичь с помощью Javascript, помогут вам в нахождении уязвимостей и в осознании потенциальной опасности, которую они могут нести. Опыт в программировании полезен, когда вы ищете уязвимости в логике приложения. Если вы можете поставить себя на

место программиста и предположить, как он мог реализовать что-либо или прочитать его код, если он доступен, вы будете иметь преимущество в этой игре.

Для этого я рекомендую посмотреть бесплатные курсы Udacity **Intro to HTML and CSS** и **Javascript Basics**, ссылки на которые я включил в главу "Ресурсы". Если вы не знакомы с Udacity, их миссия заключена в предоставлении доступного, недорогого, увлекательного и высокоэффективного высшего образования всему миру. Они имеют партнерство с такими компаниями, как Google, AT&T, Facebook, Salesforce, и многими другими, и это партнерство позволяет им создавать программы и предлагать курсы онлайн.

###Обзор глав

Глава 2 является введением в то, как работает Интернет, включая HTTP-запросы и ответы, а так же HTTP-методы.

Глава 3 описывает Open Redirects, интересную уязвимость, которая включает в себя использование целевого сайта для перенаправления пользователей на другой сайт, что позволяет хакеру воспользоваться доверием пользователя на этом уязвимом сайте.

Глава 4 описывает загрязнение параметров HTTP (HTTP Parameter Pollution) и в ней вы научитесь находить системы, которые уязвимы к передаче небезопасного ввода сайтам третьих сторон.

Глава 5 описывает уязвимости, позволяющие подмену межсайтовых запросов, или CSRF (Cross-Site Request Forgery), а примеры покажут, как можно обмануть пользователей, заставив их (без их ведома) отправить информацию на сайты, на которых они залогинены.

Глава 6 описывает HTML-инъекции (HTML Injections) и в ней вы научитесь внедрять HTML в веб-страницу и использовать его в своих целях. Один из наиболее интересных выводов -

то, как вы можете использовать закодированные значения, чтобы обмануть сайт, заставить принять их и отрендерить отправленный вами HTML, минуя фильтры.

Глава 7 описывает инъекции CLRF (Carriage Return Line Feed Injections) и в ней рассмотрены примеры отправки символов переноса строки сайтам и их влияние на отображаемое содержимое.

Глава 8 описывает межсайтовый скриптинг (XSS), крупную тему с огромным количеством способов найти уязвимость. Межсайтовый скриптинг предоставляет множество возможностей и ему одному можно посвятить целую книгу. Здесь будет куча примеров и я попытаюсь сосредоточиться на самых интересных и полезных для изучения.

Глава 9 описывает Server Side Template Injection, а так же инъекции со стороны клиента. Эти типы уязвимостей используют невнимательность разработчиков, которые вставляют пользовательский ввод напрямую в страницы при отправке с использованием синтаксиса шаблона. Опасность этих уязвимостей зависит от того, где они находятся, но часто могут вести к удаленному исполнению кода.

Глава 10 описывает SQL-инъекции, которые включают манипулирование запросами баз данных для извлечения, изменения или удаления информации с сайта.

Глава 11 описывает серверную подмену запроса (SSRF, Server Side Request Forgery), которая позволяет хакеру использовать удаленный сервер для отправки последующих HTTP-запросов от имени хакера.

Глава 12 описывает XML External Entity уязвимости, появляющиеся в результате работы сайтов, парсящих расширяемый язык разметки (XML). Этот тип уязвимостей может включать такие вещи, как чтение приватных файлов, удаленное исполнение кода и многие другие.

Глава 13 описывает удаленное выполнение кода (RCE, Remote Code Execution), или ситуацию, когда атакующий может выполнить произвольный код на сервере жертвы. Этот тип уязвимости — один из самых опасных, поскольку хакер может получить контроль над выполняемым кодом. Вознаграждается, соответственно, высоко.

Глава 14 описывает уязвимости, относящиеся к памяти, этот тип уязвимости может быть непросто найти и, как правило, он относится к низкоуровневым языкам программирования. Однако, обнаружение этого типа багов может привести к некоторым весьма серьезным уязвимостям.

Глава 15 описывает захват поддоменов, кое-что, о чем я много узнал, работая над исследованиями для этой книги, за что я благодарен Матиасу, Франсу и команде Dectetify. Суть в том, что сайт ссылается на поддомен, который размещен на сервисе третьей стороны, при этом не требуя корректного адреса от этого сервиса. Это позволяет атакующему зарегистрировать адрес со стороны этого третьего сервиса и весь траффик, который поступает на домен жертвы, на деле поступает на домен злоумышленника.

Глава 16 описывает Race Conditions, уязвимость, которая включает два или более процесса, выполняющих действие, основанное на условиях, которые должны позволять выполниться лишь одному процессу. Например, представьте перевод денег между банковскими счетами, у вас не должно быть возможности осуществить два перевода по $500, когда ваш баланс равен всего $500. Однако, уязвимость race condition (или “состояние гонки”) позволяет выполнить такой перевод.

Глава 17 описывает уязвимости Insecure Direct Object Reference (небезопасная прямая ссылка на объект), где хакер может прочитать или обновить объекты (записи в базе данных, файлы, и так далее), к которым он не должен иметь доступа.

Глава 18 описывает уязвимости, основанные на логике при-

ложений. Эта глава является собранием всех уязвимостей, которые я рассматриваю как связанные с недостатками логики программирования. Я считаю, что нахождение этого типа уязвимостей может быть несложным для новичков, по крайней мере, проще, чем попытки найти странный и креативный способ отправить вредные значения на сайт.

Глава 19 описывает, то, с чего стоит начинать. Эта глава призвана помочь вам рассмотреть возможные точки атаки и поиска уязвимостей. Она основана на моем опыте и на том, как я подхожу к поиску уязвимостей.

Глава 20 справедливо считается одной из самых важных глав в книге, поскольку содержит советы по тому, как написать эффективный отчет. Весь хакинг в мире не значит ничего, если вы не можете надлежащим образом сообщить о найденной уязвимости соответствующей компании. Таким образом, я обратился к нескольким крупным компаниям, выплачивающим вознаграждение за найденные уязвимости, спросил их совета, как лучше всего сообщить об уязвимости, и получил ответ от HackerOne. **Убедитесь, что уделили этой главе должное внимание**.

Глава 21 посвящена инструментам. Здесь мы узнаем о рекомендуемых хакерских инструментах. В первый черновик этой главы внес значительный вклад Майкл Принс из HackerOne. С тех пор это растущий и изменяющийся список полезных инструментов, которые я нашел и использую сам.

Глава 22 посвящена тому, чтобы помочь вам вывести ваш навык хакинга на следующий уровень. Здесь я расскажу о некоторых замечательных ресурсах, которые помогут продолжить обучение. Опять же, рискну показаться заевшей пластинкой, но поблагодарю Майкла Принса за вклад в этот список.

Глава 23 завершает книгу и описывает некоторые ключевые термины, которые вы должны знать, занимаясь хакингом. Хотя большинство из них обсуждаются в других главах, неко-

торые вы увидите впервые, так что рекомендую все же прочитать эту главу.

Слово предупреждения и просьба

Прежде, чем вы отправитесь в восхитительный мир хакинга, я хочу кое-что прояснить. Пока я учился, читая публичные отчеты, глядя на деньги, которые люди получали (и получают), мне казалось, что это легкий способ быстро разбогатеть. Это не так. Хакинг может быть чрезвычайно прибыльным, но непросто найти истории о неудачах, постигающих на этом пути (разве что здесь, где я делюсь некоторыми довольно постыдными историями). В результате, поскольку вы будете слышать в основном истории успеха, вы можете выработать нереалистичные ожидания в отношении успеха. И может быть вы быстро его добьетесь. Но если нет, продолжайте работать! Все станет проще, а принятый отчет об уязвимости приносит несравненное чувство удовлетворения.

Теперь я хочу попросить вас об услуге. По мере чтения, пожалуйста, пишите мне в Twitter @yaworsk и расскажите мне, как ваши дела. Успешно или неуспешно, я хотел бы узнать об этом. Поиск багов может быть одинокой работой, если вы застряли, но это также приятно праздновать друг с другом. И может быть вы найдете что-то, что мы сможем включить в следующее издание.

Удачи!!

Необходимые знания

Если вы начинаете с нуля, как начинал я, и эта книга — один из первых совершенных вами шагов в мир хакинга, для вас важно будет понимать, как работает интернет. Прежде, чем вы перевернете эту страницу, я хочу сказать, что имею ввиду то, как URL, который вы набираете в адресной строке, связывается с доменом, который направляется на IP-адрес, и так далее.

Одним предложением: Интернет — это множество систем, которые связаны вместе и отправляют друг другу сообщения. Некоторые принимают только определенные типы сообщений, другие принимают сообщения только от ограниченного списка других систем, но каждая система в интернете имеет адрес, чтобы люди могли отправлять ей сообщения. Каждая система решает, что ей делать с сообщением и как она будет отвечать.

Чтобы определить структуру этих сообщений, люди задокументировали то, как некоторые из этих систем должны общаться в Requests for Comments (RFC). Например, взгляните на HTTP. HTTP определяет протокол того, как ваш интернет-браузер общается с веб-сервером. Поскольку ваш интернет-браузер и веб-сервер действуют в соответствии с одним и тем же протоколом, они могут общаться.

Когда вы вводите http://www.google.com в адресной строке своего браузера и нажимаете enter, следующие шаги описывают то, что происходит на высшем уровне:

- Ваш браузер извлекает имя домена из URL, www.google.com.
- Ваш компьютер отправляет DNS запрос к DNS-серверам, описанным в конфигурации вашего компьютера. DNS

может помочь определить IP-адрес для доменного имени, в этом случае он равен 216.58.201.228. Подсказка: вы можете использовать dig A www.google.com из своего терминала, чтобы узнать IP-адрес для домена.

- Ваш компьютер пытается установить TCP-соединение с IP-адресом на порту 80, который используется для передачи и получения HTTP-трафика. Подсказка: вы можете установить TCP-соединение, выполнив nc 216.58.201.228 80 из своего терминала
- Если соединение успешно установлено, ваш браузер отправит HTTP-запрос, подобный этому:

```
GET / HTTP/1.1
Host: www.google.com
Connection: keep-alive
Accept: application/html, */*
```

- Теперь он будет ждать ответа от сервера, который будет выглядеть примерно так:

```
HTTP/1.1 200 OK
Content-Type: text/html

<html>
  <head>
    <title>Google.com</title>
  </head>
  <body>
    ...
  </body>
</html>
```

- Ваш браузер прочтет и отрисует возвращенный HTML, CSS и Javascipt. В этом случае, на экране появится главная страница Google.com.

Теперь, когда мы закончили с браузером, интернетом и HTML, как упомянуто ранее, существует сообщение о том, как эти сообщения будут отправляться, включая конкретные используемые методы, и требования к заголовку-запросу для всех HTTP/1.1 запросов, как обозначено в пункте 4. Описанные методы включают GET, HEAD, POST, PUT, DELETE, TRACE, CONNECT и OPTIONS.

Метод **GET** означает запрос произвольной информации, обозначенной URI (Uniform Request Identifier). Термин URI может быть непонятным, особенно в сочетании с указанным ранее URL, но для целей этой книги просто знайте, что URL — это как адрес человека, и является типом URI, который подобен имени человека (спасибо, Wikipedia). Хотя HTTP-полиции не существует, обычно GET-запросы не должны быть ассоциированы с какими-либо функциями, изменяющими данные, они просто извлекают и предоставляют информацию.

Метод **HEAD** идентичен GET, с единственным отличием: сервер не должен возвращать тело сообщения в ответе. Обычно вы не встретите случаев, когда он применяется, но он нередко служит для тестирования гипертекстовых ссылок на валидность, доступность и недавние изменения.

Метод **POST** используется для вызова некоторой функции, которая будет выполнена на сервере способом, определенным этим сервером. Другими словами, обычно на бэкенде будет выполнено некоторое действие, такое, как создание комментария, регистрация пользователя, удаление аккаунта, и так далее. Действие, выполняемое сервером в ответ на POST, может варьироваться и не обязательно вызывается в результате запроса. Например, если в процессе обработки запроса возникла ошибка.

Метод **PUT** используется при вызове какой-либо функции, но относится к уже существующей сущности. Например, при обновлении вашего аккаунта, обновлении поста в блоге, и так далее. Опять же, выполняемое действие может варьироваться и не обязательно вызывается в результате запроса.

Метод **DELETE**, как несложно догадаться, используется для вызова запроса на удаление ресурса, идентифицированного через URI, обращенного к серверу.

Метод **TRACE** — еще один необычный метод, в этот раз используемый для отражения сообщения запроса к тому, кто его запросил. Это позволяет отправителю увидеть, что было получено сервером и использовать эту информацию для тестирования и диагностики.

Метод **CONNECT** зарезервирован для использования с прокси (прокси обычно является сервером, который передает запросы к другим серверам).

Метод **OPTIONS** используется для запроса с сервера информации о доступных способах общения. Например, запрос OPTIONS может показать, что сервер принимает GET, POST, PUT, DELETE и OPTIONS, но не HEAD или TRACE.

Теперь, вооруженные базовым пониманием того, как работает интернет, мы можем ознакомиться с разными типами уязвимостей, которые могут быть в нем найдены.

Уязвимости Открытого Редиректа (Open Redirect)

Описание

Уязвимость открытого редиректа возникает, когда жертва посещает определенный URL сайта и этот сайт перенаправляет браузер жертвы на совершенно другой URL, на отдельном домене. Например, представьте, что Google перенаправляет пользователей на Gmail с помощью следующего URL:

```
https://www.google.com?redirect_to=https://www.gmail.com
```

Посетив этот URL, Google получит GET HTTP-запрос и использует значение параметра redirect_to для определения, куда должен быть перенаправлен браузер пользователя. После этого Google вернет ответ 302, заставляя браузер пользователя совершить GET-запрос на `https://www.gmail.com`, значение параметра redirect_to. Теперь представьте, что мы изменили исходный URL на этот:

```
https://www.google.com?redirect_to=https://www.attacker.com
```

Если бы Google не проверял, является ли значение параметра redirect_to одним из его собственных сайтов, куда Google мог бы отправлять пользователей (в нашем примере `https://www.gmail.com`), он мог бы быть уязвим к открытому редиректу и вернул бы HTTP-ответ, заставляющий браузер жертвы совершить GET-запрос на https://www.attacker.com.

Open Web Application Security Project (OWASP), чье комьюнити

занимается безопасностью приложений и курирует список самых критичных брешей в безопасности веб-приложений, поместил эту уязвимость в свой список 10 самых опасных уязвимостей 2013 года. Открытые редиректы используют доверие к определенному домену, в нашем примере `https://www.google.com/`, чтобы заманить жертву на вредоносный сайт. Это может быть использовано для фишинговых атак, чтобы заставить пользователя поверить, что он вводит свои данные на сайте, которому можно доверять, хотя на деле он отправился бы на вредоносный сайт. Это так же позволяет хакерам распространять вирусы с вредоносного сайта или красть токены OAuth (эту тему мы затронем в одной из последующих глав).

При поиске уязвимостей этого типа стоит смотреть на GET-запросы с параметром, отвечающим за направление редиректа, отправляемые с тестируемого сайта.

Примеры

1. Открытое перенаправление при установке темы оформления для Shopify

Сложность: Низкая

Url: app.shopify.com/services/google/themes/preview/supply–blue?domain_-name=XX

Ссылка на отчет: https://hackerone.com/reports/101962[1]

Дата отчета: 25 ноября 2015

Выплаченное вознаграждение: $500

Описание:

[1] https://hackerone.com/reports/101962

Наш первый пример — открытый редирект, найденный на Shopify, платформе, которая позволяет пользователям создавать онлайн-магазины для продажи товаров. Платформа Shopify позволяет администраторам настраивать внешний вид своих магазинов, и одним способом кастомизации является установка новых тем оформления. Ранее Shopify предоставлял возможность предпросмотра темы с использованием URL, который включал в себя параметр редиректа. URL переадресации похож на следующий, я отредактировал его для читабельности:

```
https://app.shopify.com/themes/preview/blue?domain_name=example.com/admin
```

Часть URL для предпросмотра темы оформления включала в себя параметр domain_name в конце ссылки, и этот параметр содержал другой URL, который использовался для переадресации. Shopify не валидировал этот параметр, так что его значение можно было использовать для перенаправления жертвы на `http://example.com/admin`, где хакер мог разместить фишинговую форму.

Выводы

Не все уязвимости являются сложными. Открытая переадресация в данном случае просто требовала изменения параметра domain_name на внешний сайт, что могло привести у перенаправлению пользователя на этот внешний сайт.

2. Открытое перенаправление входа в Shopify

Сложность: Средняя

Url: http://mystore.myshopify.com/account/login

Ссылка на отчет: https://hackerone.com/reports/103772[2]

Дата отчета: 6 декабря 2015

Выплаченное вознаграждение: $500

Описание:

Этот случай очень похож на первый пример с Shopify, параметр Shopify не редиректил пользователя на домен, указанный в параметре URL, но вместо этого добавлял значение параметра в конец поддомена Shopify. Обычно это использовалось бы редиректа пользователя на страницу этого магазина. После того, как пользователь логинился в Shopify, платформа использует параметр checkout_url для перенаправления пользователя. Например, если жертва посещает:

```
http://mystore.myshopify.com/account/login?checkout_url=.attacker.com
```

то она будет перенаправлена на

```
http://mystore.myshopify.com.attacker.com
```

который не является доменом Shopify, поскольку заканчивается на .attacker.com. DNS ищет самый правый домен, `.attacker.com` в этом примере. Поэтому, когда браузер пытается перейти на

```
http://mystore.myshopify.com.attacker.com
```

он попадет на attacker.com, который не принадлежит Shopify или myshopify.com.

Поскольку Shopify соединял URL магазина, в этом случае `http://mystore.myshopify.com`, с параметром checkout_url, хакер не мог свободно отправить жертву куда ему вздумается. Но хакер мог отправить пользователя на другой домен, если он знает, что URL редиректа имеет тот же самый поддомен.

[2] https://hackerone.com/reports/103772

Выводы

Параметры редиректа не всегда очевидны, поскольку параметры будут именоваться по-разному на разных сайтах, или даже в рамках одного сайта. В некоторых случаях вы можете обнаружить, что параметры имеют имя, состоящее всего из одного символа, вроде r= или u=. При поиске открытых редиректов обращайте внимание на параметры URL, которые содержат слова URL, redirect, next, и так далее, и которые могут определять путь, по которому сайт направит пользователей.

Кроме того, если вы можете контролировать лишь часть конечного URL, возвращаемого сайтом, например, только значение параметра checkout_url, и видите, что параметр комбинируется с захардкоженным URL на бэкенде сайта, например, с URL магазина `http://mystore.myshopify.com`, попробуйте добавить специальные символы URL, такие, как точка или @ чтобы изменить значение URL и перенаправить пользователя на другой домен.

3. Промежуточное перенаправление HackerOne

Сложность: Средняя

Url: Недоступен

Ссылка на отчет: https://hackerone.com/reports/111968[3]

Дата отчета: 20 января 2016

Выплаченное вознаграждение: $500

[3] https://hackerone.com/reports/111968

Описание:

Промежуточная страница — это страница, которая показывается перед ожидаемым контентом. Использование таких страниц — известная практика для защиты против уязвимости открытого редиректа, поскольку каждый раз при редиректе пользователя на URL вы можете показывать промежуточную страницу с сообщением, объясняющим пользователю, что он покидает домен, на котором он находится. Таким образом, если страница редиректа показывает фейковую форму логина или пытается притвориться доверенным доменом, пользователь узнает о том, что был перенаправлен. Этот подход используется HackerOne при переходе на большинство URL вне сайта HackerOne, например, при переходе по ссылкам в отправленых отчетах. Хотя промежуточные страницы используются для устранения уязвимостей, связанных с редиректом, сложности в том, как сайты взаимодействуют друг с другом все еще могут привести к переходу на вредоносные сайты.

HackerOne использует Zendesk, систему поддержки пользователей, на их поддомене. Когда следом hackerone.com идет /zendesk_session, пользователи перенаправляются с платформы HackerOne на платформу HackerOne Zendesk без промежуточной страницы, поскольку HackerOne доверяет ссылкам, содержащим hackerone.com. Кроме того, Zendesk позволяет пользователям переходить на другие аккаунты Zendesk с помощью параметра `/redirect_to_account?state=` так же без применения промежуточной страницы.

Махмуд Джамал в своем отчете описывает следующее: он создал аккаунт в Zendesk с поддоменом, `http://compayn.zendesk.com`, и добавил следующий Javascript код в заголовок файла с помощью редактора тем Zendesk, который позволяет администраторам настраивать внешний вид их сайтов Zendesk:

```
<script>document.location.href = "http://evil.com";</script>
```

Здесь Махмуд использует Javascript, чтобы сообщить браузе-

ру, что он должен перейти на `http://evil.com`. Детали работы Javascript здесь описаны не будут, но тег `<script>` используется, чтобы обозначить код в HTML, `document`означает весь HTML-документ, возвращаемый Zendesk и являющийся содержимым страницы. Точки и имена, следующие за `document` являются его свойствами. Свойства содержат информацию и значения, которые либо описывают объект, чьими свойствами они являются, или могут быть использованы для изменения объекта. Свойство `location` может быть использовано для контроля страницы, котору вы видите в своем браузере, а подсвойство `href` (которое является свойством `location`) перенаправляет браузер на определенный сайт. Таким образом, посещение ссылки перенаправил жертву на поддомен Махмуда на Zendesk, что заставит браузер жертвы выполнить скрипт Махмуда и перенаправит его на `http://evil.com` (обратите внимание, URL был отредактирован для читабельности):

```
https://hackerone.com/zendesk_session?return_to=https://support.hackerone.
```

Поскольку ссылка содеожит домен hackerone, промежуточная страница не будет показана и пользователь не узнает, что страница, которую он посещает, небезопасна. Что интересно, Махмуд изначально описал эту проблему Zendesk, но они не сочли её важной и не пометили как уязвимость. Поэтому он, разумеется, продолжил искать применение этой уязвимости.

Выводы

Как и при поиске уязвимостей, примите к сведению, что каждый из сторонних сервисов, используемых сайтом, представляет собой новый вектор атаки. При этом, указанная в примере уязвимость была возможна благодаря сочетанию использования HackerOne с Zendesk и знанием о том, какие перенаправления они позволяют совершать.

Кроме того стоит учесть, что когда вы находите ошибки и уязвимости, до момента их исправления может пройти много времени, прежде чем ваш отчёт о найденной уязвимости прочитают, поймут, и на него отреагируют. Вот почему у меня есть глава “Отчеты об уязвимостях”, которая описывает детали, которые нужно включить в отчет, как построить отношения с компаниями и содержит другую информацию. Более тщательная работа, а также детализированность и вежливость в вашем отчёте поможет обеспечить более глубокое понимание вопроса тем, кто занимается вопросами защиты информации, и как следствие более быструю реакцию.

Но некоторые компании даже не смотря на то, о чём я говорил, будут с вами не согласны. Если это так, смело продолжайте копать, как это сделал Махмуд. Вы можете доказать существование реальной угрозы безопасности эксплуатацией и демонстрацией возможности этой уязвимости на деле.

Итоги

Открытое перенаправление - интересная уязвимость. Она позволяет злоумышленнику перенаправлять ничего не подозре-

вающих людей на вредоносные сайты. Нахождение этих уязвимостей, как показывают примеры, часто требуют наблюдательности. Иногда параметры редиректа легко найти по именам вроде redirect_to=, domain_name=, checkout_url=, и подобным. Однако, иногда они будут иметь мене очевидные имена, такие, как r=, u= и другие.

Этот тип уязвимости полагается на использование доверия, когда жертвы посещают сайт хакера, думая, что они посетят знакомый сайт. Встретив потенциально уязвимые параметры, тщательно протестируйте их и попробуйте добавить специальные символы, такие, как точка, если часть URL поступает с бэкенда.

Кроме того, промежуточное перенаправление от HackerOne показывает важность того, что и инструменты и сервисы вебсайтов могут содержать уязвимости, и что иногда необходимо проявить настойчивость, наглядно демонстрировать уязвимость, прежде чем она будет признана и принята для выплаты вознаграждения.

HTTP Parameter Pollution

Описание

Уязвимостью HTTP Parameter Pollution, или HPP, называется манипулирование тем, как сайт обрабатывает параметры, получаемые им в процессе обработки HTTP-запросов. Уязвимость возникает, когда параметры внедряются и считаются безопасными уязвимым сайтом, что ведет к отклонению от ожидаемого поведения. Это может произойти на бэкенде, серверной стороне, где сервер сайта, который вы посещаете, обрабатывает информацию, не показывая вам процесс. А может — на клиентской стороне, где вы можете увидеть эффект своих действий на клиенте, которым обычно является ваш браузер.

Серверная уязвимость HPP

Когда вы делаете запрос к сайту, сервер этого сайта обрабатывает запрос и возвращает ответ, как было описано в Главе 1. В некоторых случаях сервер не просто возвращает страницу, но так же и выполняет некоторый код, руководствуясь информацией, полученной вместе с URL, по которому вы перешли. Этот код выполняется только на сервере и процесс невидим для вас, вы можете лишь увидеть отправленную вами информацию и результат, который получаете в ответ. Поскольку вы не можете увидеть, как функционирует серверный код сайта, серверная уязвимость HPP зависит от определения потенциально уязвимых параметров и экспериментирования с ними.

Пример серверной HPP: она может произойти, если ваш банк

инициирует трансфер через свой сайт, который работает на его серверах, обрабатывая параметры URL. Например, вы можете отправить деньги, заполнив три значения URL: откуда, куда и сколько вы хотите отправить, определив номер счета отправителя, номер счета получателя и переводимую сумму, в конкретном порядке. URL с этими параметрами, делающий запрос на перевод $5000 со счета 12345 на счет 67890 может выглядеть так:

```
https://www.bank.com/transfer?from=12345&to=67890&amount=5000
```

Возможно, банк мог бы предположить, что он будет получать только один параметр “откуда”. Но что произойдет, если вы отправите два, как в следующем URL:

```
https://www.bank.com/transfer?from=12345&to=67890&amount=5000&from=ABCDEF
```

Этот URL изначально структурирован так же, как и первый, но к нему добавлен дополнительный параметр from, который устанавливает другой счет-отправитель, ABCDEF. Как вы могли предположить, если приложение уязвимо к HPP, взломщик сможет совершить трансфер со счета, который ему не принадлежит в случае, если банк доверяет последнему параметру from, который он получает. Вместо перевода $5000 со счета 12345 на счет 67890, серверный код использует второй параметр и отправит деньги с ABCDEF на 67890.

И клиентская и серверная уязвимости HPP зависят от того, как сервер ведет себя при получении нескольких параметров с одним и тем же именем. Например, PHP/Apache использует последний параметр, Apache Tomcat использует первый параметр, ASP/IIS использует все параметры, и так далее. В результате, не существует одного гарантированного процесса обработки нескольких отправленных параметров с одним именем и нахождение HPP требует некоторого количества экспериментов, чтобы узнать, как именно работает сайт, который вы проверяете.

Хотя наш пример пока использовал очевидные параметры,

иногда уязвимости HPP являются результатом скрытого серверного поведения, которое может быть не видимым для вас напрямую. Например, скажем, наш банк переработал свой способ обработки трансферов и изменил серверный код так, чтобы он не принимал параметр from из URL, а вместо этого принимал список, который содержит в себе несколько значений.

На этот раз наш банк принимает два параметра: номер счета-получателя и сумму перевода. Номер счета-отправителя у него уже есть. Пример ссылки может выглядеть так:

```
https://www.bank.com/transfer?to=67890&amount=5000
```

Обычно серверный код для нас загадка, но нам повезло и мы украли часть их исходного кода и знаем, что их (совершенно ужасный, просто для этого примера) серверный код на Ruby выглядит так:

```
1  user.account = 12345
2
3  def prepare_transfer(params)
4    params << user.account
5    transfer_money(params) #user.account (12345) becomes pa\
6  rams[2]
7  end
8
9  def transfer_money(params)
10   to = params[0]
11   amount = params[1]
12   from = params[2]
13   transfer(to,amount,from)
14 end
```

Этот код создает две функции, `prepare_transfer` и `transfer_money`. Функция `prepare_transfer` получает массив (список) под названием **params**, который содержит параметры **to** и

amount, полученные из URL. Массив будет выглядеть как [67890,5000], его значения помещены между квадратными скобками и разделены запятой. Первая строка функции добавляет номер счета пользователя, который был определен ранее, в конец массива, и в результате получается массив [67890,5000,12345], в затем он отправляется в функцию `transfer_money`.

Вы заметите, что в отличии от параметров, массивы в Ruby не имеют имен, ассоциированных со значениями, так что код зависит от того, чтобы все значения располагались в определенном порядке, который выглядит так: первым идет номер счета-получателя, затем сумма перевода, и последний — номер счета-отправителя. В `transfer_money` это становится очевидно, функция назначает каждое значение в массиве соответствующей переменной. Нумерация элементов в массиве начинается с 0, поэтому params[0] содержит первое значение массива, которое в этом случае равно 67890, и назначает его переменной **to**. Другие значения так же назначаются переменным в следующих двух строках и затем эти переменные передаются функции трансфера, которая не показана в нашем коде, но она принимает значения и осуществляет перевод денег.

В идеале параметры URL всегда будут форматированы так, как ожидает код. Однако, взломщик может изменить результат этой логики, отправив параметр **from** в params, как в следующем URL:

```
https://www.bank.com/transfer?to=67890&amount=5000&from=ABCDEF
```

В этом случае параметр **from** так же включен в массив params, переданный функции `prepare_transfer`, и этот массив становится равен [67890,5000,ABCDEF], и добавление счета отправителя уже превратит его в такой: [67890,5000,ABCDEF,12345]. В результате, в функции `transfer_money`, вызванной в `prepare_transfer`, переменная **from** примет третий параметр, ожидая значение user.account равным 12345, но получит переданное

хакером значение ABCDEF.

Клиентская уязвимость HPP

С другой стороны, клиентские уязвимости HPP позволяют внедрять параметры в URL, что, в свою очередь, отражается на странице, которую видит пользователь.

Лука Кареттони и Стефано ди Паола, два исследователя, рассказывавшие об этой уязвимости в 2009, продемонстрировали это поведение в своей презентации, используя теоретический URL `http://host/page.php?par=123%26action=edit` и следующий серверный код:

```
<? $val=htmlspecialchars($_GET['par'],ENT_QUOTES); ?>
<a href="/page.php?action=view&par='.<?=$val?>.'">View Me\
!</a>
```

Здесь код генерирует новый URL, основываясь на введённом пользователем. Сгенерированный URL включает параметры **action** и **par**, второй определён пользовательским URL. В теоретическом URL хакер передаёт значение 123%26action=edit как значение для **par** в URL. %26 в URL является закодированным значением, которое интерпретируется как &. Это добавляет дополнительный параметр к сгенерированному адресу ссылки без добавления явного параметра **action**. Используй они вместо этого 123&action=edit, это было бы интерпретировано как два отдельных параметра, так что **par** было бы равно 123, а параметр **action** был бы равен edit. Но поскольоку этот сайт ищет и использует только параметр **par** чтобы сгенерировать новый URL, параметр **action** будет проигнорирован. Чтобы обойти это, используется %26, в результате, **action** изначально не распознаётся как отдельный параметр, а значение параметра **par** становится равным 123%26action=edit.

Теперь **par** (с & закодированным как %26) будет передан в функцию htmlspecialchars. Эта функция конвертирует специ-

альные символы вроде %26 в их кодированные значения HTML и %26 становится &. Конвертированное значение далее сохраняется в $val. Затем генерируется новая ссылка с добавлением к значению href параметра $val. Сгенерированная ссылка теперь выглядит так:

```
<a href="/page.php?action=view&par=123&amp;action=edit">
```

Проделав это, хакер сумел добавить дополнительный параметр action=edit в целевой URL, который может вести к уязвимости в зависимости от того, как сервер обрабатывает получение двух параметров action.

Примеры

1. Кнопки социальных сетей HackerOne

Сложность: Низкая

Url: https://hackerone.com/blog/introducing-signal-and-impact

Ссылка на отчет: https://hackerone.com/reports/105953[4]

Дата отчета: 18 декабря 2015

Выплаченное вознаграждение: $500

Описание:

В блоге HackerOne есть ссылки для шаринга контента через популярные социальные сети, такие как Twitter, Facebook и прочие. Эти ссылки создают контент, который пользователь может опубликовать в социальных сетях, с обратной ссылкой на оригинальный пост в блоге. Ссылки на создание постов содержат параметры, которые при клике перенаправляют пользователя на этот пост.

[4] https://hackerone.com/reports/105953

Обнаруженная уязвимость позволяла хакеру подставить другие параметры URL в ссылку, что отразилось бы на расшариваемой ссылке и привело бы к тому, что ссылка, которой поделились в социальной сети, вела бы произвольный сайт. сайт. Пример, используемый в этом отчёте об уязвимости включает ссылку:

```
https://hackerone.com/blog/introducing-signal
```

куда затем добавляется следующее

```
&u=https://vk.com/durov
```

Если бы посетители HackerOne нажали на обновленную таким образом ссылку, пытаясь поделиться контентом через социальные сети, вредоносная ссылка выглядела бы так:

```
https://www.facebook.com/sharer.php?u=https://hackerone.com/blog/introduci
```

Если на обновленную таким образом ссылку нажал бы посетитель HackerOne, пытаясь поделиться контентом через социальные сети, последний параметр **u** имел бы приоритет над первым и, соответственно, был бы использован в публикации на Facebook. Это привело бы к тому, что пользователи Facebook при клике на ссылку были бы перенаправлены на https://vk.com/durov вместо HackerOne.

Кроме того, при публикации в Twitter, предложенный стандартный текст также мог бы быть изменен. Это достигается добавлением **&text=** в url:

```
https://hackerone.com/blog/introducing-signal?&u=https://vk.com/durov&text
site:https://vk.com/durov
```

При клике на эту ссылку появился бы попап твита, который имел бы текст **another_site: https://vk.com/durov** вместо того, что предоставляет целевой пост на HackerOne.

Выводы

Обращайте внимание на случаи, когда сайты принимают контент и взаимодействуют с другим веб-сервисом, таким, как сайты социальных сетей.

В этих ситуациях может быть возможна отправка переданного содержимого без надлежащих проверок его безопасности.

2. Уведомления об отмене подписки в Twitter

Сложность: Низкая

Url: twitter.com

Ссылка на отчет: blog.mert.ninja/twitter-hpp-vulnerability[5]

Дата отчета: 23 августа 2015

Выплаченное вознаграждение: $700

Описание:

В августе 2015 хакер Мерт Таски, отменяя подписку на получение уведомлений от Twitter, заметил интересный URL:

```
https://twitter.com/i/u?iid=F6542&uid=1134885524&nid=22+26
```

(Я сократил его немного для книги). Вы заметили параметр UID? Оказалось, что это UID пользовательского аккаунта в Twitter. Заметив это, он сделал то, что, как я полагаю, сделали бы большинство хакеров, он попытался изменить UID на чужой и… ничего. Твиттер вернул ошибку.

[5] http://www.blog.mert.ninja/twitter-hpp-vulnerability

Многие сдались бы, но Мерт был настроен решительно и попробовал добавить второй параметр UID к URL, который теперь выглядел так (опять же, я сократил):

```
https://twitter.com/i/u?iid=F6542&uid=2321301342&uid=1134885524&nid=22+26
```

и... УСПЕХ! Он сумел отменить подписку на уведомления для другого пользователя. Оказалось, Твиттер был уязвим к HPP при отмене этой подписки.

Выводы

Хоть описание и короткое, попытки Мерта демонстрируют важность настойчивости и знаний. Если бы он оставил попытки после неудачи с подстановкой чужого UID в качестве единственного параметра, или если бы он не знал об уязвимостях типа HPP, он бы не получил вознаграждение в 700 долларов.

Также, внимательно относитесь к параметрам вроде UID, которые включены в HTTP-запросы, поскольку за время моих исследований я видел множество отчетов, которые включали манипулирование их значениями и веб-приложения делали неожиданные вещи.

3. Twitter Web Intents

Сложность: Низкая

Url: twitter.com

Ссылка на отчет: Parameter Tamperting Attack on Twitter Web Intents[6]

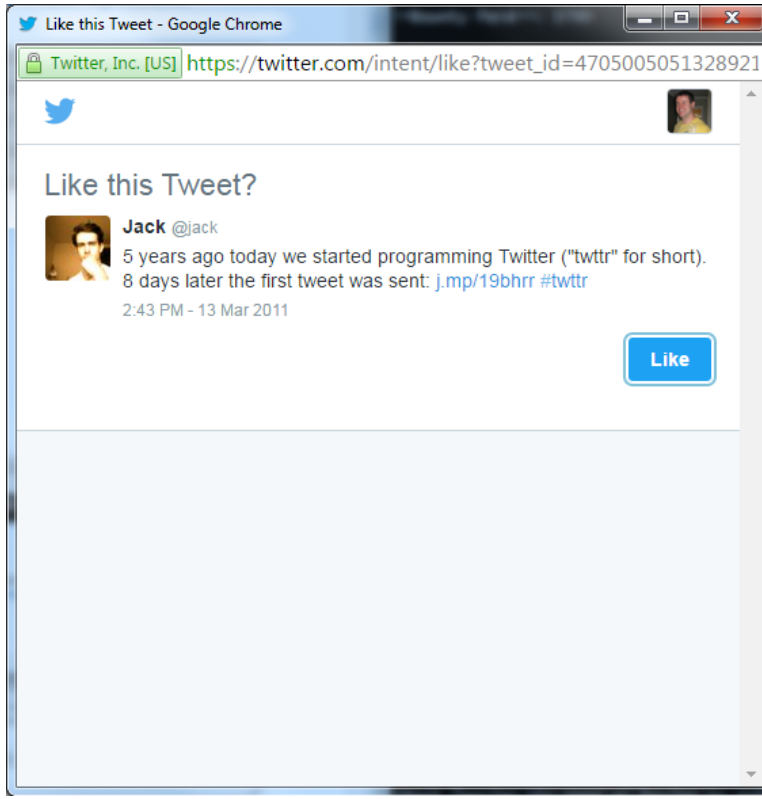
Дата отчета: Ноябрь 2015

[6] https://ericrafaloff.com/parameter-tampering-attack-on-twitter-web-intents

Выплаченное вознаграждение: Не раскрыто

Описание:

Twitter Web Intents предоставляет попапы для работы с пользовательскими твитами, ответами, ретвитами, лайками и кнопками Follow в контексте сторонних сайтов. Они позволяют пользователям взаимодействовать с контентом Твиттера, не покидая страницу или не требуя авторизоваться в приложении для простого взаимодействия. Вот как выглядит пример такого попапа:



Twitter Intent

Тестируя эту возможность, хакер Эрик Рафалофф обнаружил,

что все четыре типа “интентов”, фолловинг пользователя, лайк твита, ретвит и твит, были уязвимы к HPP. Twitter создавал каждый такой “интент” с помощью GET запроса с использованием параметров, подобным следующим:

```
https://twitter.com/intent/intentType?paramter_name=paramterValue
```

Этот URL будет включать **intentType** и один или более параметров с парами ключ-значение, например, пользовательский юзернейм в Твиттере и id твита. Twitter будет использовать эти параметры, чтобы создать попап интента, который будет показан пользователю, чтобы он мог лайкнуть твит или зафолловить пользователя. Эрик обнаружил, что если он создаст URL с двумя параметрами **screen_name** для интента на follow, то вместо одиночного **screen_name**, вроде

```
https://twitter.com/intent/follow?screen_name=twitter&screen_-
name=ericrtest3
```

Твиттер обработает запрос, отдав приоритет при генерировании кнопки второму значению **screen_name** (ericrtest3), а не первому, а пользователь, пытаясь зафолловить официальный аккаунт Твиттера, обманется, зафолловив вместо этого тестовый аккаунт Эрика. Посещение URL, созданного Эриком, создаст следующую HTML-форму с двумя параметрами **screen_name**, сгенерированную бэкендом Твиттера:

```
<form class="follow" id="follow_btn_form" action="/intent\
/follow?screen_name=ericrtest3" method="post">
  <input type="hidden" name="authenticity_token" value=".\
..">
  <input type="hidden" name="screen_name" value="twitter">
  <input type="hidden" name="profile_id" value="783214">
  <button class="button" type="submit" >
    <b></b><strong>Follow</strong>
  </button>
</form>
```

Твиттер подтянул бы информацию с первого параметра **screen_name**, который ассоциирован с официальным аккаунтом Твиттера, поэтому жертва увидит корректный профиль пользователя, которого она собиралась зафолловить, потому что первый параметр **screen_name** использовался для заполнения двух значений в полях ввода. Однако, по клику на кнопку жертва зафолловила бы аккаунт ericrtest3, потому что параметр action в форме использовал бы значение второго **screen_name**, переданное в исходный URL:

```
https://twitter.com/intent/follow?screen_name=twitter&screen_name=ericrtest3
```

Аналогично, экспериментируя с интентами для лайков, Эрик обнаружил, что может включить второй параметр **screen_name**, несмотря на то, что он не имеет никакого отношения к лайку твита. Например, он мог создать URL:

```
https://twitter.com/intent/like?tweet_id=661625230297821184 5&screen_name=ericrtest3
```

Нормальный лайк нуждается лишь в параметре tweet_id, но Эрик вставил параметр **screen_name** в конец URL. Лайк такого твита привёл бы к ситуации, в которой жертва поставила бы лайк правильному твиту, но кнопка Follow, расположенная рядом с корректным твитом и профилем, вела бы на некорректный профиль пользователя ericrtest3.

Выводы

Это похоже на предыдущую уязвимость в Твиттере, касающуюся UID. Неудивительно, что когда сайт уязвим к вещам вроде HPP, это может быть индикатором более широкой системной проблемы. Иногда, если вы находите подобную уязвимость, стоит потратить время на исследование платформы в целом, чтобы убедиться, что вы не обнаружите (или обнаружите) другие области, в которых можно будет использовать похожий сценарий.

Итоги

Риски, которые открывает использование HTTP Parameter Pollution сильно зависят от действий, выполняемых бэкендом сайта и тем, куда будут отправлены вредоносные параметры.

Обнаружение уязвимостей такого вида очень зависит от экспериментов, больше, чем другие уязвимости, поскольку действия бэкенда сайта могут быть полностью неизвестны хакеру. Чаще всего, исследуя наличие этой уязвимости, вы будете иметь очень малое представление о том, какие действия принимает сервер после получения отправленных вами данных.

Через трудности и ошибки, вы сможете обнаружить подобные уязвимости. Ссылки на социальные сети обычно являются хорошим первым шагом, но не забывайте продолжать поиски и думать о HPP при тестировании других уязвимостей вроде замены параметров.