Д. Н. Колегов, О. В. Брославский, Н. Е. Олексов, Исследование возможности управления веб-браузерами на основе фрэймворка BeEF и облачного сервиса Google Drive, *ПДМ*, 2015, номер 4, 72–76

DOI: http://dx.doi.org/10.17223/20710410/30/6

МАТЕМАТИЧЕСКИЕ ОСНОВЫ КОМПЬЮТЕРНОЙ БЕЗОПАСНОСТИ

УДК 004.94

ИССЛЕДОВАНИЕ ВОЗМОЖНОСТИ УПРАВЛЕНИЯ ВЕБ-БРАУЗЕРАМИ НА ОСНОВЕ ФРЭЙМВОРКА BeEF И ОБЛАЧНОГО СЕРВИСА Google Drive[1]

Д. Н. Колегов, О. В. Брославский, Н. Е. Олексов

Национальный исследовательский Томский государственный университет, г. Томск, Россия

Текущая реализация средства для анализа защищённости веб-приложений Browser Exploitation Framework (BeEF) предполагает прямое сетевое взаимодействие контролируемых браузеров с сервером управления, что не позволяет обеспечить высокий уровень неотслеживаемости и необнаруживаемости такого взаимодействия в компьютерной сети. В работе показывается, как может быть реализован механизм сетевого взаимодействия между сервером управления BeEF и контролируемыми браузерами с помощью скрытого канала через облачный файловый сервис Google Drive, позволяющий выдать сетевые потоки управления контролируемыми браузерами за стандартные пользовательские запросы к файловому сервису Google Drive.

Ключевые слова: *компьютерная безопасность, HTTP, скрытые каналы, безопасность веб-приложений, безопасность веб-браузеров, бот-сети.*

DOI 10.17223/20710410/30/6

HOOKED-BROWSER NETWORK WITH BeEF AND Google Drive

D. N. Kolegov, O. V. Broslavsky, N. E. Oleksov

National Research Tomsk State University, Tomsk, Russia

E-mail: d.n.kolegov@gmail.com, o.v.broslavsky@gmail.com, n.e.oleksov@gmail.com

At the present time, Browser Exploitation Framework (BeEF) supports experimental WebRTC-based mechanism for implementing a hooked browser meshed-network. The main purpose of this solution is to avoid tracking post-exploitation communication back to BeEF command and control server. We propose an alternate method to provide more anonymity and undetectability for BeEF hooked browser communications. The main idea is to use covert channel communications over known and popular cloud web services, for example Google Drive, by using it as shared resources between BeEF server and hooked browsers. In this case, there is no direct communication between BeEF server and hooked browsers, all of them communicate only with Google API servers. The implementation is based on Google Drive file system primitives and

[1]Работа представлялась на международной конференции по практическим аспектам информационной безопасности «Zero Nights 2015».

its API. We consider practical issues of this implementation and show how this can be implemented in BeEF.

Keywords: *computer security, HTTP, covert channels, web application security, web browsers security, botnets.*

Введение

Одним из наиболее известных средств для инструментального анализа защищенности веб-приложений, использующих методы реализации ботнетов для сетевого взаимодействия с исследуемыми браузерами, является Browser Exploitation Framework [1].

После получения контроля над браузером в результате эксплуатации какой-либо уязвимости веб-приложения одной из основных задач яляется обеспечение постоянного коммуникационного канала между браузером и командным сервером (command and control server) [2]. В случае внедрения контролирующего JavaScript-сценария в браузер жертвы коммуникационный канал, как правило, реализуется с помощью механизмов XMLHttpRequest, WebSocket или WebRTC. Вместе с тем ни один из этих механизмов, применённый для связи контролируемых браузеров напрямую с командным сервером, не обеспечивает решения задачи неотслеживаемости и необнаруживаемости командного сервера. На конференции Kiwicon в 2014 г. Кристианом Фричатом (Christian Frichot), одним из главных разработчиков BeEF, был представлен механизм построения сети контролируемых браузеров на основе технологии WebRTC [3]. В этом случае контролируемые браузеры могут взаимодействовать не только с сервером BeEF, но и друг с другом. При этом часть контролируемых браузеров все равно ваимодействует с сервером BeEF напрямую.

Другим подходом к решению данной задачи является реализация взаимодействия с командным сервером не напрямую, а через промежуточные узлы популярных сервисов (например, Google, Twitter, Yandex и др.). Как правило, крупные облачные хостинги, такие, как Google Drive и Dropbox, имеют высокий уровень доверия и, что важно, предоставляют программный интерфейс (API) для доступа к функционалу хостинга средствами JavaScript. В случае реализации такого сетевого взаимодействия средства анализа смогут фиксировать лишь сетевые информационные потоки между браузерами пользователей и сервисами, неотличимые от разрешённых потоков. Даный подход используется, например, в прототипах Gcat [4] и Twittor [5].

В данной работе демонстрируется возможность реализации механизма сетевого взаимодействия между сервером управления BeEF и контролируемыми браузерами с помощью скрытого канала через облачный файловый сервис Google Drive, позволяющий выдать сетевые потоки управления контролируемыми браузерами за стандартные пользовательские запросы к файловому сервису Google Drive.

1. Принципы работы BeEF

Для более детального понимания задачи ознакомимся с основными элементами, используемыми в BeEF. *Зомби* — вредоносный JavaScript-код, выполняемый в браузере жертвы. *Командный сервер* — основная часть инструментального средства, ответственная за отправку команд браузерам-зомби, а также за получение, обработку и хранение результатов выполнения команд. *Идентификатор «Hook session»* — уникальный идентификатор зомби, присваиваемый ему командным сервером.

Процесс захвата контроля над браузером, который моделирует действия злоумышленника, происходит следующим образом: в результате эксплуатации уязвимости бра-

узер жертвы загружает вредоносный JavaScript-код, который немедленно начинает исполняться в контексте заражённой страницы. В первую очередь данный вредоносный код собирает информацию о браузере жертвы и отправляет её командному серверу, после чего вредоносным кодом инициируются процедуры получения команд от сервера BeEF и логирования событий.

Для получения команды браузер жертвы с определённой периодичностью совершает HTTP-запросы к командному серверу и в теле HTTP-ответа получает JavaScript-код команд, поставленных в очередь на исполнение. О результатах выполнения должна сообщить сама команда, используя механизм отправки результатов командному серверу через функцию *beef.net.send*. Механизм логирования записывает все события, происходящие на заражённой странице, и с определённой периодичностью отправляет записанные события командному серверу.

Командный сервер BeEF неоднороден по своей структуре и состоит из множества обработчиков запросов вида /, */init*, */event*, а также обработчиков, уникальных для каждого конкретного модуля или расширения BeEF. Первый из запросов заражённого браузера, содержащий информацию о нём, предназначен для обработчика */init*. В данном обработчике полученная от браузера информация анализируется и заносится в базу данных. С этого момента заражённый браузер становится зомби, а нарушитель получает возможность отправлять этому зомби команды, используя интерфейс командного сервера. Обработчики / и */event* ответственны за выдачу команд зомби-браузерам и за сохранение результатов работы модуля регистрации событий.

2. Реализация коммуникационного канала через сервис Google Drive

Исходя из описанного процесса функционирования, первым шагом в решении задачи реализации коммуникационного канала между зомби-браузерами и сервером BeEF является переход от их прямого сетевого взаимодействия к непрямому сетевому взаимодействию с использованием функциональности облачных хостингов для временного размещения команд для зомби и результатов их выполнения для командного сервера.

Для минимально приемлемой реализации такого хранения команд и ответов достаточно разработать систему уникальных имён для файлов и хранить их в одном каталоге облачного хостинга. В некоторых случаях уникальности данных имён не требуется. Например, в Google Drive файлы различаются при помощи идентификаторов, самостоятельно генерируемых сервисом, а не их именами. Однако такое представление не является достаточно наглядным, поэтому предлагается разделить все хранимые файлы на несколько логических групп и использовать для каждой из групп отдельную директорию. Так, директория *Init* содержит информацию о только что заражённых браузерах, ожидающих обработки отосланной информации и подключения к серверу BeEF; директория *Answers* содержит данные выполнения команд в следующем формате:

— идентификатор команды в базе данных командного сервера;
— имя обработчика, ответственного за дальнейшую обработку результатов;
— непосредственно данные, предназначенные для этого обработчика;
— уникальный идентификатор зомби, выполнившего команду.

Кроме того, для каждого зомби создаётся отдельная директория с именем, соответствующим его идентификатору *hook_session*. Данная директория содержит команды, отправленные командным сервером BeEF и ожидающие загрузки их зомби-браузером (рис. 1).

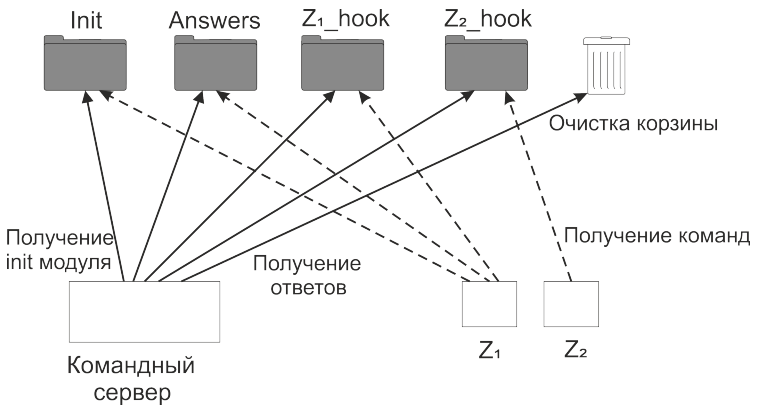

Рис. 1. Схема размещения ресурсов на Google Drive

Для авторизации доступа к сервису Google Drive необходим ряд ключей доступа. Ключ *api_key* используется зомби и позволяет запрашивать данные у программного интерфейса Google Drive, авторизуя доступ на чтение. Этот ключ используется для регулярного получения актуального значения ключа *auth_key*. Ключ *auth_key* используется командным сервером и зомби-браузерами для создания и удаления файлов, которые хранятся на облачном сервисе. Данный ключ действителен в течение ограниченного интервала времени и требует регулярного обновления. Для обновления данного ключа командный сервер использует ключ *master_key*, который может быть получен при помощи аутентификации по протоколу *OAuth* на сервисах Google. Обновлённый *auth_key* помещается сервером BeEF в специальную сущность-файл *keychain*, откуда он может быть загружен браузером-зомби при помощи ключа *api_key*.

Так как зомби в классической архитектуре BeEF для получения команд от сервера BeEF использует метод *polling*, то есть периодически опрашивает командный сервер о наличии новых команд, необходимые изменения на клиенте минимальны. Помимо реализации механизмов взаимодействия с программным интерфейсом Google Drive, необходимо, чтобы зомби осуществлял следующую логику. После инфицирования и сбора сведений о браузере полученные результаты загружаются в директорию *Init* и зомби начинает периодически проверять наличие новых команд в своей директории с именем, соответствующим идентификатору *hook_session*.

Для проверки обновлений используется стандартный механизм проверки обновлений с модифицированной функцией запроса. При обнаружении новых файлов в командной директории их содержимое выгружается и добавляется в очередь на исполнение, а сами файлы с командами помещаются в корзину.

Результаты выполнения команд помещаются в директорию *Answers* по готовности в формате, описанном выше. Результаты логирования событий загружаются в директорию *Answers* по мере накопления так, что в одном файле содержатся события, произошедшие в зомби-браузере за определённый интервал времени. Важным моментом является то, что одним из идентификаторов зомби на командном сервере является IP-адрес. В оригинальной архитектуре значение IP-адреса получается из соответствующего поля HTTP-запроса. Очевидно, при непрямом сетевом взаимодействии такой подход невозможен, поэтому в этом случае IP-адрес опрделяется запросом к внешнему сервису, а затем добавляется к первичной информации о заражённом браузере.

Командный сервер лишён какой бы то ни было активности и работает в классической клиент-серверной архитектуре. Таким образом, для работы в новой архитектуре

командный сервер нуждается в схожем с зомби *polling*-механизме. Опрос директорий Google Drive реализован в виде расширения BeEF и происходит в трёх независимых потоках выполнения. Поток *init_server* с заданной периодичностью проверяет содержимое директории *Init*, выполняет предварительную обработку полученных данных и передаёт их одноимённому обработчику командного сервера. Поток *command_server* занимается обработкой очереди команд, оформляя элементы очереди необходимым образом и сохраняя их в командных директориях зомби-браузеров. Поток *data_server* отвечает за получение и обработку результатов выполнения команд: один раз в течение заданного временного интервала *data_server* выгружает все накопившиеся в директории *Answers* ответы, записывает информацию о полученных ответах в базу данных и передаёт результаты соответствующим обработчикам.

Ещё один служебный поток выполнения описываемого расширения занимается регулярным обновлением ключа *auth_key* при помощи ключа *master_key* и интерфейса, предоставляемого сервисами Google. Отдельные функции расширения следят за созданием директорий для зомби-браузеров и за очисткой корзины, в которую помещаются все файлы, «считанные» зомби-браузерами и командным сервером.

Таким образом, в работе продемонстрирована возможность реализации сетевого взаимодействия между сервером управления BeEF и контролируемыми браузерами с помощью скрытого канала через облачный файловый сервис Google Drive.

ЛИТЕРАТУРА

1. The Browser Exploitation Framework Project. `http://beefproject.com/`
2. *Alkorn W., Frichot C., and Orru M.* The Browser Hacker's Handbook. Indianapolis: John & Wiley Sons, 2014. 648 p.
3. Hooked-Browser Meshed-Networks with WebRTC. `http://blog.beefproject.com/2015/01/hooked-browser-meshed-networks-with.html`
4. The Gcat Project. `https://github.com/byt3bl33d3r/gcat`
5. The Twittor Project. `https://github.com/PaulSec/twittor`

REFERENCES

1. The Browser Exploitation Framework Project. `http://beefproject.com/`
2. *Alkorn W., Frichot C., and Orru M.* The Browser Hacker's Handbook. Indianapolis, John & Wiley Sons, 2014. 648 p.
3. Hooked-Browser Meshed-Networks with WebRTC. `http://blog.beefproject.com/2015/01/hooked-browser-meshed-networks-with.html`
4. The Gcat Project. `https://github.com/byt3bl33d3r/gcat`
5. The Twittor Project. `https://github.com/PaulSec/twittor`