контроля, а обнаружение скрытого канала, реализованного подобным образом, также затруднительно, так как им используется только стандартная функциональность, предоставляемая API сервиса. Теоретическая пропускная способность такого скрытого канала составляет 1 бит за $(L + S)$ секунд, где L — время, необходимое s_3 для выполнения запроса к e_3, а S — время, уходящее на обработку запроса на серверах хостинга.

Рассмотрим пример реализации предложенного метода на примере облачного сервиса хранения файлов Google Drive. Субъект s_1 через заданные промежутки времени осуществляет чтение одного бита из сущности e_1 и, в зависимости от полученных данных, совершает или не совершает POST-запрос e_2 по адресу `www.googleapis.com/drive/v2/files/`**fileId**`/touch`, где **fileId** — идентификатор сущности e_3. Данный запрос обновляет время последней модификации сущности-ресурса e_3. В тот же временной интервал субъект s_3 выполняет GET-запрос по адресу `www.googleapis.com/drive/v2/files/`**fileId** и получает ответ e_4, содержащий значение entity-tag сущности e_3. Субъект s_3 записывает в e_5 бит 1, если ресурс был модифицирован с момента последнего запроса, и бит 0 в противном случае.

В результате тестирования реализации скрытого канала через сервис Google Drive достигнута пропускная способность 3 бит/с при точности передачи 99,8 %, где под точностью понимается отношение числа правильно переданных бит к общему числу бит. Установлено, что основным фактором, влияющим на пропускную способность реализации, является время обработки запроса серверами сервиса. Таким образом, можно говорить о достижении в эксперименте максимальной пропускной способности канала.

ЛИТЕРАТУРА

1. *Колегов Д. Н., Брославский О. В., Олексов Н. Е.* Об информационных потоках по времени, основанных на заголовках кэширования протокола HTTP // Прикладная дискретная математика. Приложение. 2014. № 7. С. 89–91.
2. *Колегов Д. Н., Брославский О. В., Олексов Н. Е.* Исследование скрытых каналов по времени на основе заголовков кэширования протокола HTTP // Прикладная дискретная математика. 2015. № 2. С. 71–85.

УДК 004.94 DOI 10.17223/2226308X/8/32

НЕИНВАЗИВНЫЙ МЕТОД КОНТРОЛЯ ЦЕЛОСТНОСТИ COOKIE В ВЕБ-ПРИЛОЖЕНИЯХ

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

Предлагается метод контроля целостности cookie в веб-приложениях, построенный на основе криптографических протоколов с ключевыми хеш-функциями. Метод может быть использован для реализации неинвазивных механизмов защиты от атак на веб-приложения через cookie.

Ключевые слова: *криптографические протоколы, хеш-функции, веб-приложения, web cookie.*

Термином *cookie* в протоколе HTTP обозначается набор данных, хранимый веб-клиентом (веб-браузером) и отправляемый на сервер в специальном заголовке Cookie в HTTP-запросах. Cookie первоначально могут быть сгенерированы на веб-сервере и переданы веб-клиенту в заголовке Set-Cookie в HTTP-ответе либо сгенерированы

на стороне веб-клиента с помощью переданного веб-сервером Javascript. Cookie, как правило, используются веб-приложением для идентификации и аутентификации пользователей, а также для хранения произвольных данных, настроек и т. п. Таким образом, если веб-приложение имеет уязвимость, связанную с недостаточной обработкой данных, передаваемых в cookie (например, уязвимость к атаке внедрения операторов SQL), то злоумышленник может модифицировать полученные cookie для эксплуатации данной уязвимости. Примеры атак на веб-приложения через механизм cookie описаны, например, в [1].

Одним из методов защиты от атак на веб-приложения является использование ключевых хеш-функций для аутентификации HTTP сообщений. Первоначально данный метод использовался лишь для контроля целостности атрибутов HTML (например, атрибутов href, src, action) и полей форм [2, 3]. В работе [4] метод обобщён и показано, как он может быть применён для защиты веб-приложений от широкого класса атак (например, CSRF, утечка токенов, HPP и др.). Одним из достоинств данного метода является возможность его неинвазивной реализации на уровне веб-приложения — реализации, не требующей изменения исходного кода веб-приложения.

Ввиду существенных отличий в структуре данных и принципах функционирования механизма cookie метод [4] не может быть применен к последнему в том же виде. В данной работе исследуется возможность неинвазивного контроля целостности cookie на основе хеш-фунций.

Структура данных cookie может быть представлена в виде набора $c = (k_c, v_c, p_c, d_c, e_c, f_c)$, где k_c — ключ (имя cookie); v_c — значение cookie; p_c — значение атрибута path; d_c — значение атрибута domain; e_c — значения атрибута expires; f_c — список флагов (например, *secure* — передача cookie возможна только по HTTPS; *session* — cookie является сессионной; *httponly* — доступ к cookie средствами Javascript запрещён).

При установке cookie веб-сервер отправляет веб-клиенту в заголовках HTTP-ответа весь необходимый набор атрибутов, которые сохраняются веб-клиентом, после чего больше никогда не отправляются в HTTP-запросах к серверу. Таким образом, обеспечение целостности всех атрибутов cookie представляется более сложной задачей, чем тривиальное хеширование всех контролируемых атрибутов. Алгоритмы обеспечения целостности cookie, предложенные в [5], решают данную задачу лишь частично, обеспечивая целостность только для значения cookie и времени жизни. Подходы, описанные в работах [6, 7], решают задачи, связанные с реализацией защищённого контейнера на основе cookie для «sessionless»-веб-приложений.

Предлагаемый метод контроля целостности cookie обладает следующими свойствами:

— обеспечение целостность значения cookie;
— защита cookie от удаления или продления, т. е. от изменения атрибута expires или установки флага *session*;
— обеспечение целостности значений атрибутов path и domain;
— контроль передачи cookie по защищённому соединению при установленном флаге *secure*;
— возможность неинвазивной реализации.

Будем использовать следующие обозначения: $x|y$ — конкатенация строк x и y; $hmac$ — ключевая хеш-функция, построенная по алгоритму HMAC; $hmac(k, s)$ — результат применения алгоритма $hmac$ с ключом k к строке s.

Пусть C — множество всех cookie, используемых веб-приложением, а $S \subseteq C$ — множество защищаемых (контролируемых) cookie. Для каждой защищаемой cookie $c = (k_c, v_c, p_c, d_c, e_c, f_c) \in S$ построим парную ей cookie $c\_s = (k_s, v_s, p_s, d_s, e_s, f_s)$, где:

1) e_s — максимально возможное значение, что позволяет получать из cookie c актуальную информацию о cookie c_s, даже если последняя устарела;
2) $p_s = p_c$ и $d_s = d_c$, что обеспечивает одновременную отправку веб-клиентом cookie c и c_s до истечения времени expires;
3) $v_s = hmac(k, v_c|e_c)|e_c$, где k — ключ, уникальный для каждой сессии веб-приложения; данное построение позволяет контролировать целостность значений v_c и e_c и, кроме того, проверять, не истекло ли время жизни cookie c;
4) $f_s = f_c \cup \{httponly\}$.

Аутентификатором множества S будем называть строку $auth$, полученную конкатенацией значений k_c, p_c, d_c, f_c для всех cookie c из S.

Для определения легитимности отправки каждой защищаемой cookie по значениям атрибутов path и domain дополнительно введём вспомогательную cookie $\alpha = (k_\alpha, v_\alpha, p_\alpha, d_\alpha, e_\alpha, f_\alpha)$, в которой:

1) $v_\alpha = hmac(k, auth)|auth$, где $auth$ — аутентификатор S;
2) p_α и d_α выбираются наиболее общими для рассматриваемого веб-приложения, что гарантирует отправку cookie α при любом запросе к веб-севрверу;
3) e_α — максимально возможное значение;
4) $f_\alpha = \{httponly\}$.

Рассмотрим основные действия протокола взаимодействия веб-клиента и веб-сервера. При первом запросе к веб-серверу пользователю устанавливается α с некоторым начальным значением. Все последующие запросы к веб-приложению обрабатываются в соответствии с приведённым ниже методом. Если в ответе веб-сервера устанавливается одна или несколько защищаемых cookie, то для каждой такой cookie c в HTTP-ответ добавляется заголовок Set-Cookie, устанавливающий парную ей cookie c_s, а данные о c заносятся в α. После обработки всех защищаемых cookie в HTTP-ответе значение $hmac$ в α пересчитывается и в HTTP-ответ добавляется заголовок Set-Cookie, устанавливающий обновлённую α. При необходимости выставляются заголовки, удаляющие отмеченные на удаление cookie.

Метод обработки запроса к веб-приложению

Условия применения метода: задано множество S защищаемых cookie веб-приложения.

Для каждого HTTP-запроса h выполнить следующие шаги:

1) Если в HTTP-запросе h отсутствует cookie α, то удалить из запроса h все защищаемые cookie $c \in S$ и отправить модифицированный запрос h веб-серверу.
2) Проверить целостность значения cookie α. Для этого по значению v_α вычислить $hmac' = hmac(k, auth')$. При несовпадении $hmac'$ и $hmac$ запрос следует считать запрещённым.
3) Удалить из запроса h все защищаемые cookie $c \in S$, которые отсутствуют в α.
4) Для каждой защищаемой cookie $c \in S$, присутствующей в запросе h, проверить выполнение следующих условий:
   а) запрос h содержит для c парную cookie c_s;
   б) атрибут p_c, полученный из cookie α, допускает отправку cookie c в запросе h;

в) атрибут d_c, полученный из cookie α, допускает отправку cookie c в запросе h;
г) для cookie c с флагом *secure* запрос h передан по HTTPS;
д) выполняется равенство $v_s = hmac(k, v_c|e_c)$ и e_c меньше текущего времени.

При невыполнении любого из условий запрос следует считать запрещённым.

5) Пометить на удаление все cookie c_s, для которых в запросе h отсутствуют парные им cookie c.

В случае, если HTTP-запрос к веб-приложению признан запрещённым, его обработка приложением может привести к эксплуатации уязвимости, а потому нежелательна. В связи с этим предлагаются следующие варианты обработки таких HTTP-запросов:

1) Блокирование HTTP-запроса.
2) Перенаправление пользователя на некоторый URL-адрес.
3) Завершение сессии пользователя веб-приложения путём отправления на веб-сервер специального HTTP-запроса и последующего удаления парных cookie и выставления начального значения cookie α.
4) Модификация HTTP-запроса путём удаления из него всех некорректных cookie и последующей отправки этого запроса на веб-сервер.

Таким образом, описанный метод контролирует целостность всех атрибутов cookie, защищает их от несанкционированного удаления на клиентской стороне, позволяет определить необходимость отправки cookie на заданный путь и домен веб-приложения, а также обеспечивает аутентичность cookie. В то же время данный метод предполагает добавление $|S| + 1$ дополнительных cookie. Количество последних можно существенно сократить, отказавшись от введения парных cookie и вычисляя значение v_α следующим образом: $v_\alpha = hmac(k, m)|m$, где m — строка, полученная конкатенацией значений $domain_i$, $path_{ij}$, $hmac_{ij}$. Значение $hmac_{ij} = hmac(k, m_{ij})$, где m_{ij} — конкатенация значений k_c и v_c для всех cookie c, которые могут быть отправлены на домен $domain_i$ и путь $path_{ij}$. В данном варианте реализации для всех защищаемых cookie $c \in S$ должны быть известны p_c и d_c.

В этом случае обработка HTTP-запроса h происходит следующим образом:

1) Если в HTTP-запросе h отсутствует cookie α, то удалить из запроса h все защищаемые cookie $c \in S$ и отправить модифицированный запрос h веб-серверу.
2) Проверить целостность значения cookie α. Для этого по значению v_α вычислить $hmac' = hmac(k, m')$. При несовпадении $hmac'$ и $hmac$ запрос следует считать запрещённым.
3) Вычислить $hmac(k, m)'$ для v'_α, используя для формирования m значения защищаемых cookie, полученных в h. При несовпадении $hmac(k, m)'$ и соответствующего текущему пути и домену значения $hmac_{ij}$ в v_α запрос следует считать запрещённым.

Обработка HTTP-ответа происходит следующим образом:

1) Если HTTP-запрос был запрещён, то выставить в ответе заголовки, удаляющие на стороне клиента все защищаемые cookie $c \in S$ и α.
2) Если в HTTP-ответе присутствуют заголовки, устанавливающие защищаемые cookie, то следует обновить значение $hmac_{ij}$ для всех пар $(domain_i; path_{ij})$, соответствующих текущему пути и домену.

Данный метод обеспечивает целостность значений cookie, требует меньшего количества служебных cookie, но накладывает следующие существенные ограничения на веб-приложение:

— невозможность выставления cookie c с атрибутами path и domain, отличными от пути и домена HTTP-запроса, в котором выставляется cookie c;
— невозможность пересчёта значения $hmac$ по истечении времени жизни или при удалении cookie;
— инициализация новой сессии при любом некорректном запросе.

Реализация данных методов не требует изменения исходного кода веб-приложения и может быть выполнена на уровне гибридных WAF (например, ModSecurity), модульных фрэймворков (например, Django, Ruby on Rails) и сетевых WAF (например, F5 BIG-IP ASM). Прототип системы, реализующей эти методы, разработан на базе Django Middleware [8].

ЛИТЕРАТУРА

1. *Barnett R.* The Web Application Defender's Handbook: Battling Hackers and Protecting Users. Indianapolis: John Wiley & Sons, 2013. 522 p.
2. Reducing Web Application Attack Surface. `http://blog.spiderlabs.com/2012/07/reducing-web-apps-attack-surface.html`
3. ModSecurity Advanced Topic of the Week: HMAC Token Protection. `http://blog.spiderlabs.com/2014/01/modsecurity-advanced-topic-of-the-week-hmac-token-protection.html`
4. *Колегов Д. Н.* Общий метод аутентификации HTTP-сообщений в веб-приложениях на основе хеш-функций // Прикладная дискретная математика. Приложение. 2014. № 7. С. 85–89.
5. *Fu K., Sit E., Smith K., and Feamster N.* Dos and Don'ts of client authentication on the Web // Proc. 10th USENIX Security Symp., Washington, 2001. P. 251–268.
6. *Liu A., Kovacs J., Huang C., and Gouda M.* A secure cookie protocol // Proc. 14th Intern. Conf. Computer Communications and Networks, 2005. P. 333–338.
7. *Murdoch S.* Hardened Stateless Session Cookies. `http://www.cl.cam.ac.uk/~sjm217/papers/protocols08cookies.pdf`
8. Прототип модуля неинвазивного контроля целостности cookie на базе Django. `https://github.com/tsu-iscd/django-HTTPauth`

УДК 004.94 DOI 10.17223/2226308X/8/33

НЕИНВАЗИВНАЯ РЕАЛИЗАЦИЯ МАНДАТНОГО УПРАВЛЕНИЯ ДОСТУПОМ В ВЕБ-ПРИЛОЖЕНИЯХ НА УРОВНЕ СУБД

Д. Н. Колегов, Н. О. Ткаченко

Предлагается неинвазивный метод (метод, не изменяющий исходный код самого приложения) устранения уязвимостей в механизмах логического управления доступом и информационными потоками в веб-приложениях на уровне СУБД. Задача ставится следующим образом: имеется многоуровневое веб-приложение, в котором реализована подсистема базового управление доступом (как правило, ролевого), возможно, имеющая некоторое множество уязвимостей. Необходимо устранить как можно более широкий класс данных уязвимостей без изменения исходного кода самого приложения или обеспечить реализацию новой политики мандатного управления доступом. Метод включает выполнение следующих