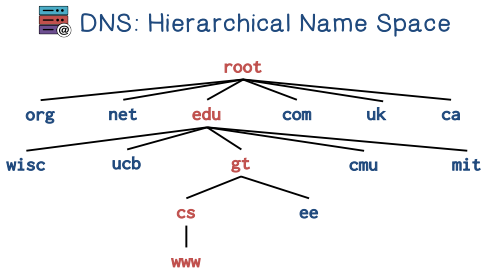
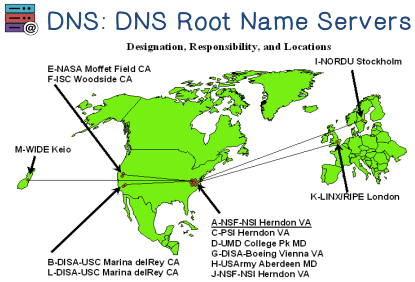
The recommended readings for this lesson are:

Steve Friedl's Unixwiz.net Tech Tips: An Illustrated Guide to the Kaminsky DNS Vulnerability

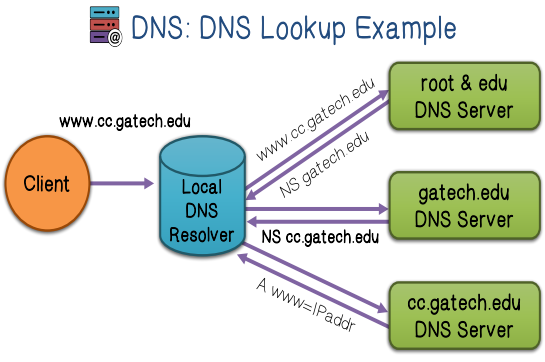
In this lesson, we will discuss domain name systems, or, DNS. We will explore there weaknesses and the security measures that have been implemented to protect the Internet.



DNS, or, the domain name system is a hierarchical database. There are root servers, top level domains, second level domains, third level domains, and so on.

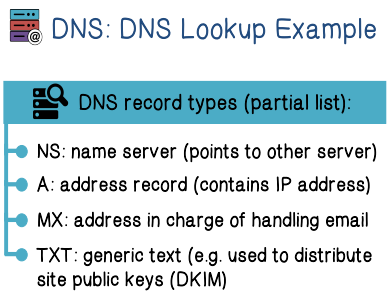


Just for your information there are 13 DNS root name servers.



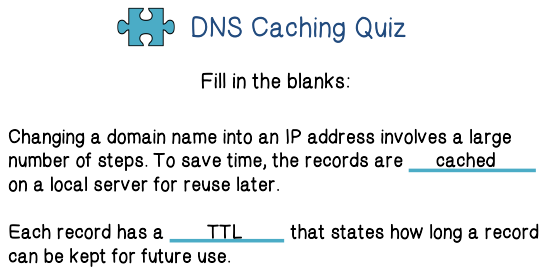
A DNS Lookup is an iterative, or, recursive, process querying the hierarchy code database. For example, suppose your browser is looking up www.cc.gatech.edu. The local DNS service is handling this request of looking up the IP address of this domain name. The query will start on the root or top level domain servers. That is, the local DNS server asks the root and top level domain servers what the IP address of www.cc.gatech.edu is. And these servers say, I do not know, but I know the main server of

gatech.edu. So the local DNS server then asks gatech.edu DNS server what the IP address of www.cc.gatech.edu is. And the gatech.edu DNS server says, I do not know but I know the name server cc.gatech.edu. So finally, the name server cc.gatech.edu says, I know the IP address of www.cc.gatech.edu and here it is. So that is an example of looking up a domain name in most iterative or recursive queries to DNS servers.

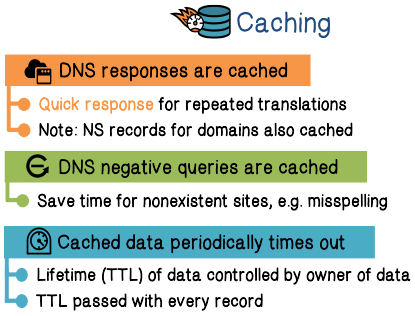


There are several types of DNS records in the response to DNS query. One is the NS record. This points to a name server. That is, this record contains the IP address of a name server such as gatech.edu. And then there is A record. This contains the address of the domain name in the original query. For example, www.cc.gatech.edu. And then there is MX record. This contains the address of the mail server for the domain. For example, mail.gatech.edu. And finally, there is a TXT record. It contains all the useful information about a domain. For example, it can be used to distribute public keys.

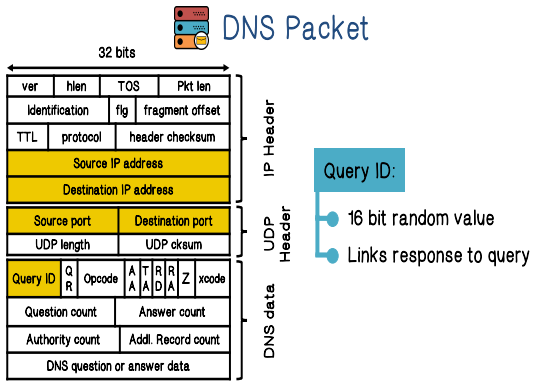
Now let us do a quiz.



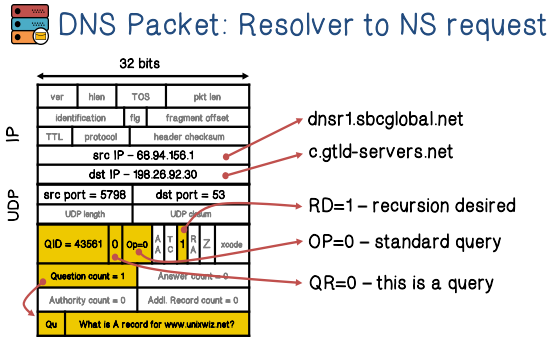
As illustrated in the previous example, querying the IP address of a domain name can involve a number of steps. To save time, the records are cached on a local server for reuse later. For example, when the IP address of www.cc.gatech.edu has been obtained, the mapping of this IP address and the domain is cached so that the next time the browser is looking up www.cc.gatech.edu, the DNS server does not have to go out to look it up again because the mapping is already stored in the cache. Obviously, this saves time. On the other hand, each record, meaning the mapping of IP address and domain that is being stored in the DNS cache, has a TTL, or time-to-live. And, when this TTL expires, the cache entry is invalid. Which means, if the browser looks up www.cc.gatech.edu, after the TTL of the cache entry has expired, the DNS server then has to go out and look it up again. TTL is useful because a server, say www.cc.gatech.edu, may be moved to a new IP address. So, you want the TTL to expire, so that the DNS servers can look up for the new mapping.



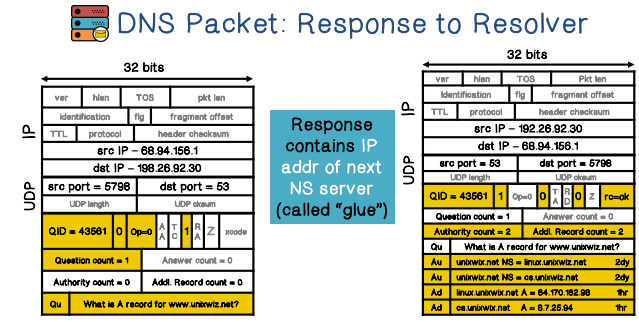
As we have discussed, the DNS responses and in particular, the mapping between the IP address and a domain are cached so that we can save time on repeated queries. The NS records of the domains are also cached. Therefore, if the browser looks up www.ece.gatech.edu, the local DNS server only needs to start with gatech.edu instead of the root. DNS servers can also cache the negative results such as a domain does not exist. For example, if somebody misspelled gatech.edu to say gatech.ed, the DNS query response will be: this domain does not exist. And this result is cached. So that, next time if somebody mistype again the same way, gatech.ed, the DNS cache can just say, hey, this domain does not exist, without having to query it. And all cache data, whether it is positive or negative response has a TTL.



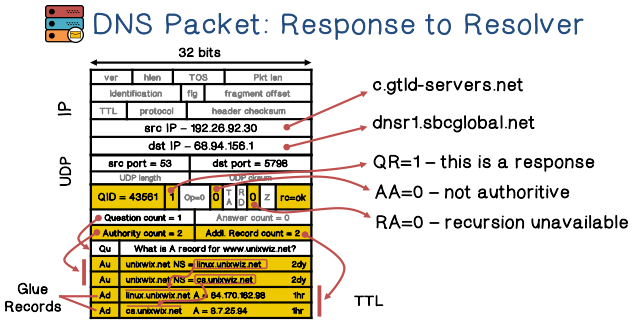
Now let us take a closer look at the internals of DNS query and response. Let us start with the format of a DNS packet. There is the usual IP header. There is a UDP header because DNS uses UDP, and the UDP payload is the actual DNS data. One of the most important fields in DNS data is the query ID, which is a 16-bit random value. A DNS query contains a query ID. And a response also carries the ID. Therefore, even though a DNS server may send out many, many queries at the same time, it can use the ID to link a response to a query. That is, the response to a query will have the same ID of the query ID of the original query.



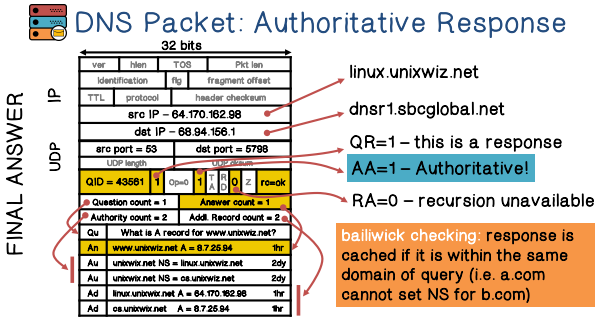
Now, let us look through an example. Suppose a local DNS server is looking up for the domain www.unixvis.net on behalf of a browser. So, this is the local DNS server. It is going to send the query to one of the root services. It sends a flag to say recursion desired. That means it is asking the destination DNS server to perform recursive queries on its behalf.



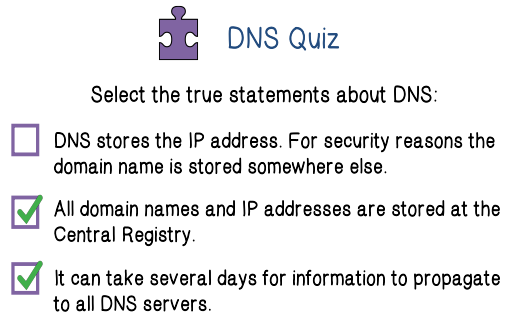
Here is the response from the root server. The root server does not know the IP address of www.unixwiz.net. So it provides the IP addresses of the next NS service. And these are called glue records.



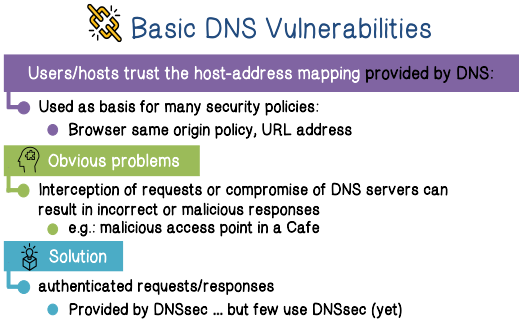
Let us look at this in more detail. The response comes from the root server, back to the local DNS server. It says that it is a response, the root server does not know the IP address of www.unixwiz.net, but it knows where to ask. It responds with a series of NS records that should know how to handle the original query. Notice that both the domain names and IP addresses of these named servers are provided. Because this response is not the final answer to the original query, it sets this flag to zero to indicate that this is not authoritative. And also because the root server is busy, it is not going to perform because of queries on behalf of the local DNS server. That is why it sets this flag to indicate that it is not going to perform because of queries. That is, the local DNS server should contact these named servers instead.



The final, or authoritative, response comes from the named server, linux.unixwiz.net. And notice that now the authoritative flag is set to 1. Notice that this final response contains the IP address of the domain www.unixwiz.net, which is the domain name contained in the original query. And the TTL is one hour, which means this record will be cached and be valid for one hour. And since these NS records are in the same second level domain, unixwiz.net, of the original queried domain, they are also cached.

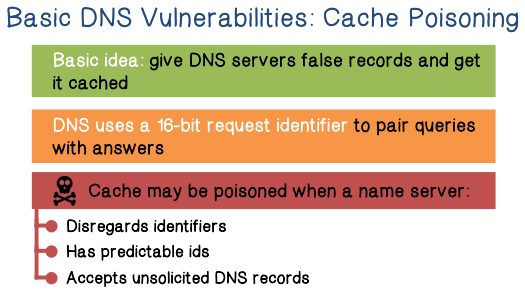


Now let us do a quiz. Select the true statements about DNS. The first statement, DNS stores IP address. For security reasons, the domain name is stored somewhere else. This is false. DNS stores both the IP address and the domain name. That is the whole point of providing the mapping between domain name and IP address. The second statement, all domain names and IP addresses are stored at the Central Registry. This is true. When a new domain name is registered at a local DNS, it will also be copied to the Central Registry. The third statement, it can take several days for information to propagate to all DNS servers. This is true. It takes anywhere from 12 to 36 hours to propagate information to all DNS servers worldwide.

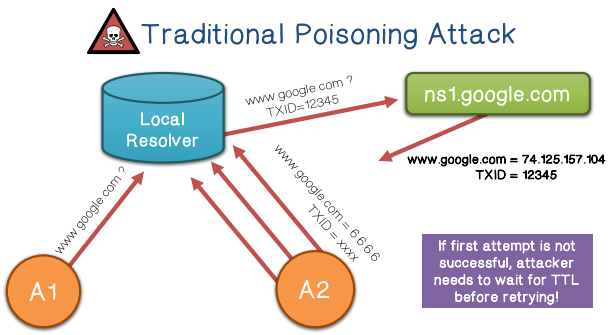


Now let us discuss the main vulnerabilities of DNS. First of all, we must be able to trust the domain name and address mapping provided by DNS. In fact, many security policies depend on this. For example, the same origin policy in browsers, or URL-based filtering. Obviously, if the host address mapping provided by DNS can be forged, then the traffic intended for the original legitimate host is now destined to the wrong or malicious host. This means that the wrong or forged host can now intercept the legitimate traffic.

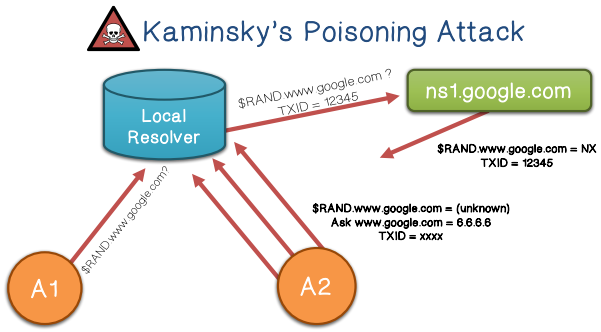
What if the host address mapping provided by DNS can be forged? For example, instead of getting the IP address of gmail.com, the browser instead gets the IP address of evil.com. The result is that, traffic will be routed to evil.com instead, which means evil.com can intercept traffic to gmail.com. There are several ways to forge the host address mapping. For example, the attackers can compromise the DNS servers, including cache poisoning, which we are going to discuss shortly, or the attackers can control the access point or gateway and intercept DNS queries and forge a response. A solution is to authenticate each request and response using cryptography. And DNSSEC is such a solution.



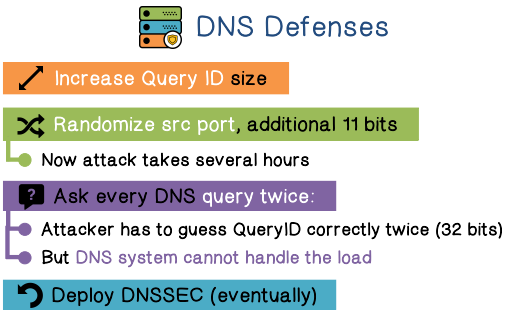
Now let us discuss attacks on the inner server, in particular, cache poisoning. The basic idea is that the attacker would provide to the local DNA server some false records and get the records cached. For example, if the local DNS server queries the domain gmail.com and the attacker is able to inject a response with the IP address of evil.com and have that IP address of evil.com cached by the local DNS server and then subsequent traffic to gmail.com will be routed to evil.com. The existing defense in DNS is the use of a 16-bit request ID to link a response with a query. That is, the attacker's response must have the ID that matches the ID of the original query and we will discuss how an attacker can overcome this defense. A DNS cache can be easily poisoned if the DNS server does not use the IDs properly, or the IDs are predictable.



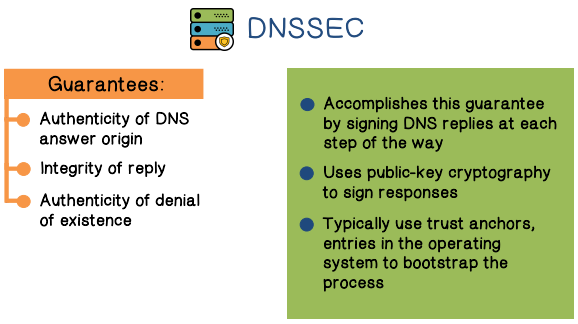
Now let us discuss DNS cache poisoning attacks in more detail. Let us start with traditional poisoning attack. Suppose the attacker wants to poison the cache of the DNS server at gatech.edu. In particular, the attacker wants to forge the IP address of www.google.com. The attacker first sends a query to the local DNS server. This can be done, for example, through a compromised machine within gatech.edu. The local DNS server is now going to perform a recursive query with the query ID 12345. And at some point the main server, ns1.google.com, is going to provide an authoritative answer with the IP address of www.google.com with a matching query ID. Now the attacker knows that this recursive query is taking place, because he has a compromised node inside the network that initiated the original lookup. So, the attacker is going to forge a response claiming it is from ns1.google.com. And in that response, it's going to use the IP address of evil.com instead of the real IP address of www.google.com. But the attacker does not know the real query ID, so all he can do is send a flood of responses, each with a guessed query ID. So, this is a matter of the attacker being able to guess the correct query ID and reach the local DNS server faster than the legitimate response from the real DNS server of Google. If the attack succeeds, then the incorrect answer will be cached resulting in cache poisoning. But if the attacker's attempt fails, the legitimate IP will be cached, and the attacker has to wait for TTL to expire before launching the whole attack again.



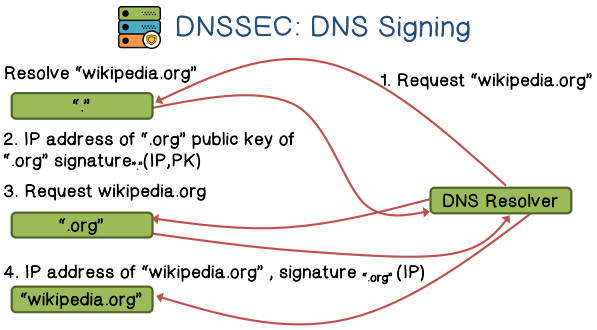
As you can see, the traditional poisoning attack is hard to succeed. But then Kaminsky found an approach that is drastically more effective than the traditional attack. The general approach is the same as the traditional attack. But the key difference is the nature of the forged payload. The intention of cache poisoning is to poison the final answer that is the A record with the IP address. But what Kaminsky discovered is that we can go up one level and hijack the authority records instead. As in the previous example, let us assume that the attacker wants to poison the cache of the DNS server at gatech.edu. And he wants to forge the IP address of www.google.com. But this time the inside help is going to send a query of a random domain within www.google.com. For example, 12345678.www.google.com. And as before the DNS server is going to perform a recursive query. Now the legitimate response will say that this random domain does not exist but will provide the IP address of www.google.com. The attacker is attempting to do the same thing. The goal is to have the DNS server cache his forged IP address of www.google.com. And you may ask, what is new here? Isn't the attacker facing the same challenge of guessing correctly the query ID before the response from Google? In the traditional attack, when the first attempt fails, the attacker has to wait for the TTL to expire. What is new here in the Kaminsky's Poisoning Attack is that when the first attempt fails, the attacker can start immediately again. That is, it doesn't have to wait for the TTL to expire and the reason is he can simply use a different random domain and that will immediately result in another query. So, he can flood the DNS server again. That is, the attacker can repeatedly and continuously force the local DNS server to query a random domain and keep flooding the local DNS server until the poisoning attack succeeds. Such Kaminsky's Poisoning Attack can succeed in mere ten seconds.



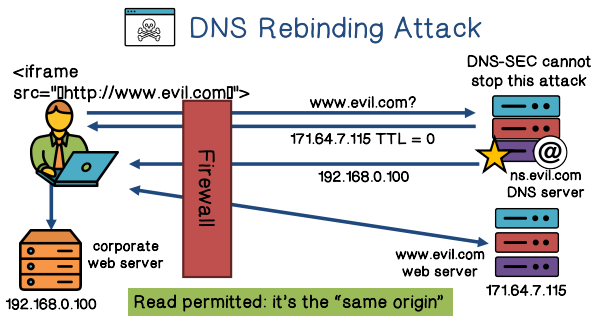
So what are the defenses against cache poisoning attack? The first few here simply make the attackers do a lot more work in order to succeed. For example we can increase the query ID size, we can randomize the source port or we can query twice. More fundamentally, we can use cryptography to provide authenticity of DNS records, and that is the idea behind DNSSEC.



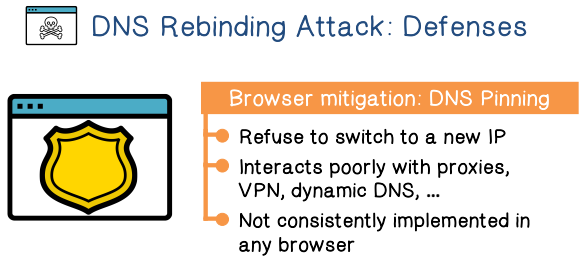
So let us discuss DNSSEC. The goals of DNSSEC is to provide guarantees of the authenticity of the DNS servers as well as the integrity of their responses. These guarantees are accomplished by having the DNS servers sign responses every step of the way. It is also assumed that the DNS servers themselves can be secured.



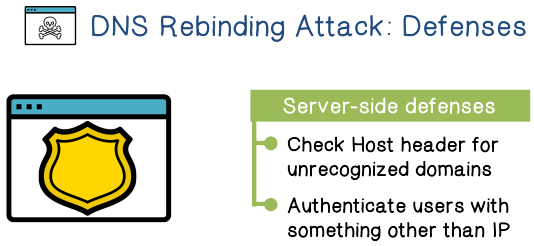
Here is an example of the DNS signing process. Suppose a local DNS server looks up wikipedia.org. They first query the root server. The root server provides the IP address of .org and signs it. The signature is based on the private key of the root server. The DNS server performs recursive query, in this case sending the request to .org. And the response contains the IP address of wikipedia.org signed with the private key of .org. The local DNS server can verify all these signatures and be confident that the IP address that it receives is correct.



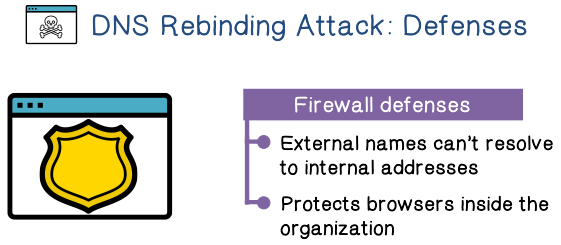
Even DNSSEC cannot prevent all DNS attacks. The DNS Rebinding Attack is one such example. To launch a DNS rebinding attack, the attacker needs to only register a domain name, such as evil.com. And attract web traffic, for example, by running an advertisement in a frame. In this attack, when evil.com is looked up, the attacker answers with the IP address of his own server and use a very short TTL value. The attacker's server, evil.com, also serves the browser a malicious JavaScript. To circumvent the firewall, when the malicious JavaScript issues a query to evil.com, the TTL has expired. The attacker then rebinds the host name, evil.com, to an IP address of an internal server. That is, now the firewall thinks that evil.com is internal. The browser now believes that these two servers belong to the same origin, because they share the same host name, www.evil.com. So it will allow the malicious script to read back the response. Therefore, the malicious script can easily exfiltrate information from the server to evil.com. That is now the attacker is able to read arbitrary documents from the internal server.



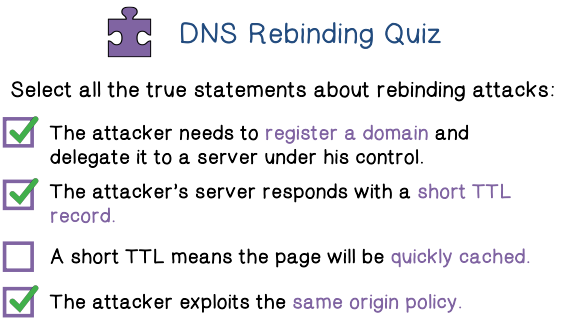
To mitigate such attack, the browser should use DNS Pinning. Meaning that, you should refuse the switching to new IP address for domain. On the other hand, this means that it may break proxies, VPNs, dynamic DNS, and so on. Therefore, it is not consistently implemented in all browsers.



For the internal servers, they should check Host headers for unrecognized domains such as evil.com. It should also provide stronger authentication of users.



For the firewall, it should implement a policy such that external domain names cannot resolve to internal IP address. It should provide stronger protection of browsers within the network.



Now, let's do a quiz. Select all true statements about rebinding attacks. The first statement, the attacker needs to register a domain and delegate it to a server under his control, this is true. The second statement, the attacker's server responds with a short TTL record, this is true. The third statement, a short TTL means that the page will be quickly cached, this is false. The fourth statement, the attacker exploits the same origin policy, this is true.

An Illustrated Guide to the Kaminsky DNS Vulnerability

The goal of this reading is to understand a major vulnerability in DNS. To learn about DNS basics before understanding the vulnerability, refer the "Terminology" section as well as the section "Resource Record Types" (scroll down to find it).

The working of a simple recursive DNS query is described in the section "Following a simple DNS query". This section describes the distributed nature of DNS. The client (user PC) requests its ISP's local nameserver for the A record (representing the IP address) for a domain like www.website.net. The nameserver picks one of 13 root servers at random and sends the query to it. The root server gives the NS records (as well as IPs) of the GTLD (global top-level domain) servers responsible for .net domain. This continues, i.e., the recursive nameserver follows a chain of referrals. Finally, the nameserver gets an authoritative response with the A record for the domain. The nameserver sends a (non-authoritative) response back to the user.

To understand the vulnerability, it is important to know about some fields in a DNS packet – most importantly the source and destination IP addresses and ports, and the Query ID field (also called transaction ID). Refer to the section "What's in a DNS packet?" The query ID is a unique identifier created in the query packet, allowing the server to associate answers with questions. To observe how the fields are used in a DNS query flow, refer the diagrams in "Drilling down to a real query".

Caching and cache poisoning: A local cache is used to satisfy future queries faster. Every cached value has a TTL (time to live) after which it expires. A records, NS records etc. are all cached. With DNS cache poisoning, hostname-to-IP lookups are no longer legitimate and cannot be trusted. DNS only responds to pending queries, so one way to poison the cache is to guess the query ID – since query ID increments by one in older nameservers, it is easy to guess. One method of mitigation is to randomize the query ID. Refer the "Poisoning the cache" section as well as "Shenanigans, Version 1" for a detailed explanation of this traditional poisoning attack. This method is used to poison a single record.

The vulnerability highlighted in this paper was discovered by Dan Kaminsky. He discovered that it was possible to go up a level and hijack the authority records. The attacker first configures an authoritative nameserver for their own malicious domain, say, baddomain.com. Then the evil client requests a random name within the target domain, of the sort random1234www.baddomain.com. A stream of forged packets is sent to the victim as in the traditional attack, but this time instead of valid A records, these responses delegate to another nameserver via Authority records. This step points those nameservers to the evil IPs. The attacker

will typically set a high TTL so that the bogus data is stored for a long time in the cache. Mitigation techniques include increasing the size of the query ID and randomizing the source port. For more information refer to the "Dan's Shenanigans" section in the document, as well as the linked document in the summary section – "Dan's presentation at Black Hat".