An Overview of DNS Cache Poisoning

Quirin Wiggerich
TU München
Garching, Germany

***Abstract*—In today's world everyone who uses the internet makes use of the domain name system (DNS). DNS is the backbone of the internet as it maps from domain names that are easy for humans to understand to IP addresses. DNS poisoning now describes the process of mapping the wrong IP address to a requested domain name. Sending the user to a website under the attacker's control. In the following we will explain the process of DNS cache poisoning as well as presenting use cases and how to protect against it.**

I. Introduction

The Domain Name System (DNS) was invented to map from a name (example.com.) that is easy for humans to understand to an IP address (192.0.2.0) which the computer can understand. The computer then uses the IP address to connect to the website. To lookup the corresponding IP address for a website name the stub resolver on the end device sends the request to a DNS forwarder (router). Which will then forward the request to a DNS resolver. In general a domain server can either directly respond to the resolver with the IP address, an A record in the DNS response would mean the answer is an IPv4 address and an AAAA record would mean an IPv6 address. Or the domain server could respond with a referral to a different domain server where the resolver will get the answer to his query. The DNS resolver then asks the root server (RS) for the IP address. The root server does not know this, but will direct the resolver to a top level domain (TLD) server for the domain name (in this case .com). The TLD will then redirect the resolver to the Name Server. The name server will then respond with the IP address for the website. To not have to go through this whole process again when another request for the same website name is made. The resolver, as well as the DNS forwarder and the device that made the request will store the IP address in its cache. Unfortunately the DNS was not designed with security in mind and there exist many different attacks against it. One of the most prominent attacks is the DNS cache poison attack. DNS-Cache Poisoning now describes the process of "poisoning" this DNS cache in any of the devices in the DNS chain.

osi model?

figure to explain

II. The first DNS poison attack

The simplest form of the attack had been known since 2002 and works as follows. An off-path adversary that is able to send spoofed packets (but not to intercept, modify or block packets) will send a target device a number of queries for a domain name. This domain name is the domain the attacker tries to compromise. When the resolve does not have the domain in its cache the resolver will send a request to the authoritative name server. While the resolver is waiting for the real answer to arrive the attacker will send a large number of forged responses to the target resolver. These forged responses contain an IP address that is under the attacker's control. The target resolver will of course not just accept any response. A number of parameters have to be correct to be accepted. These include UDP source port number and transaction ID (TXID). The UDP protocol is typically used for DNS queries and the UDP port number is a 16 bit field. The TXID is also a 16 bit field that identifies the response. The attacker has to guess both of these fields so that the response will be accepted and poison the DNS cache of the resolver. But at the time the UDP source port was a fixed value and not random. The resolver simply used the same source port for every transaction it made. Which means the UDP port can be easily predicted by the attacker and they just had to guess the TXID field which equals only 2 to the power of 16 possible combinations. Interestingly due to the "birthday paradox" an attacker only needs 2 to the power of 8 tries on average to generate a matching TXID header.

udp?

what

The attack works because once a response has been accepted by the resolver all future responses will be ignored. That means when the real response comes back after the forged one has been accepted it will be ignored by the resolver. Additionally there is often a large time to live (TTL) value attached to the spoofed response, so that the false IP address will stay in the resolvers cache as long as possible [1].

why

III. Kaminsky Attack

In 2008 Dan Kaminsky found three observations that drastically improved the usefulness of this attack in the real world. While the basic mechanism remains the same as the conventional DNS cache poisoning attack. The first observation is that the attacker can force the resolver to initiate a query to an authoritative name server of their choice. The second one, that the growing network bandwidth enables attackers to send even more forged responses with different TXID fields to the resolver. And lastly to not provide an answer for the queried domain name, but to provide info in the additional section in the response. With all that Kaminsky's attack works as follows. First the attacker chooses a domain that they want to poison, like www.example.com. Even when www.example.com is already in the resolver cache, the attacker simply generates non-existent subdomain names, for example wyzyw.example.com, which ensures that the resolver will send a query to the authoritative name server. After that the attacker floods the resolver with fake responses. But this time the attacker does not send an actual answer to the query but instead a referral to an authoritative name server where the resolver can get the answer. This authoritative server then is under the attackers control. The resolver will cache the information and each time a new query gets initiated for a subdomain of example.com, like www.example.com, the resolver will ask the authoritative name server under the attacker's control, allowing the attacker to poison not only a single domain but multiple at the same time. Another advantage of this approach is, should the attacker fail to be faster than the actual response they can just try again with a

how

threat model

new randomly generated subdomain. In the previous version of the attack this wasn't possible since when the actual answer comes back, the resolver will have the IP address in its cache and won't start another query until the TTL runs out.

All of these improvements to the attack made it much more likely to succeed in the real world and thus the people behind DNS resolvers felt the need to respond to this threat. Some of the fast "security fixes" that were deployed included the randomization of the UDP source port, significantly increasing the amount of possible combinations and reducing the probability that an attacker would guess both correctly. But none of these patches would be able to completely stop DNS cache poisoning attacks from being possible.

The IETF also started the development of DNSSEC, a set of DNS security extensions that would make DNS cache poisoning impossible due to the use of digital signatures. More on this later [2].

IV. Uses of DNS cache poisoning

A DNS cache poisoning attack is often used as an initialization of another attack. Such an attack could be a phishing attack, where the attacker would steal credentials from users. The user would be directed to a phishing site impersonating the real one. Once the user enters their credentials the attacker could then even forward the login information to the real site and redirect the user to the real site after successfully stealing their information.

Alternatively the attacker could send the victim to a malicious web page that tries to launch exploits in the web browser or in browser add ons. With the goal of getting malicious code onto the target device. examples

Another use case could be to launch a distributed denial of service (DDOS) attack. In that case the attacker would poison the cache so that a popular website renders unavailable. Or redirect users from a popular website to a smaller one that can't handle the incoming traffic. Essentially putting both websites out of service [3].

V. Prevent DNS cache Poisoning

The simplest fix to DNS cache poisoning would of course be if domain names would no longer be used but only IP addresses. But this approach is of course not feasible for our current world.

to late explained

The current and more logical approach is the deployment of DNS security extension (DNSSEC). DNSSEC adds authenticity and integrity to the DNS using digital signatures. DNSSEC does this by three primary records types.

- DNSKEY records, which are public keys. Typically each zone uses two keys. A Key Signing Key (KSK) and a Zone Signing Key (ZSK). The KSK is usually only used to create the RRSig for the DNSKEY records. The ZSK in contrast is used to create the RRSigs for all other records
- RRSig (Resource Record Signature) records which are cryptographic signatures of other records. They are created with the private Key of the domain. Each RRSIG is a signature over all records of a given type for a certain name. So would all A records (IPv4) for example.com be authenticated by a single RRSig. And all of those IPv4 address records are called the RRSet
- DS (Delegation Signer) records which are hashes of the Key Signing Keys. These hashes are then uploaded to the parent zone, which creates the "chain of trust". The DS records in the parent zone are authenticated by a RRSig, and signed with the parent zones ZSK. So would example.com upload their DS record to the .com zone. The .com zone would then upload their DS record to the root zone (.).

More security of course comes with a cost, which means that an operator that wants to deploy DNSSEC on his domain needs to perform additional tasks compared to regular DNS. Which might explain DNSSECs lack of deployment. Even though DNSSEC was first defined in 1999 by the IETF, the rootzone of the DNS network, where every request begins, was first signed in 2010 and the public rootzone key was published. From this year on, domain name server hoster could deploy DNSSEC in their zone. But progress has been slow, to this day over 90% of top level domains, which include .com, .org. But on domain level deployment has been slow only on roughly 10% of all domains DNSSEC has been deployed. On top of that comes the problem that even if DNSSEC has been deployed in a large number of cases it is setup incorrectly. About 31% of all domains fail to publish their records correctly that are needed for validation and about 39% use weak Key Signing Keys and 90% use weak Zone Signing Keys that are not secure enough by today's standard. Even though DNSSEC is supposed to be protected against stolen or weak keys, by frequent rotation of the keys, many domains do not rollover their keys often enough or not at all. All of these mentioned problems are on the DNS server side, but the client side also features major issues. While over 82% of resolvers request DNSSEC records from the domain servers, only 12% actually try to validate them, making all security benefits from DNSSEC irrelevant and making the resolver vulnerable to attacks like DNS cache poison attacks and even opening the door for new attacks, that exploit the partial deployment of DNSSEC. With all this said, the deployment of DNSSEC might be slow, but it has been steadily growing for the past 12 years and will most likely continue to do so in the coming years. On the other hand there have also been ideas to completely abandon UDP as the protocol of choice for DNS but instead use TCP or even TLS. [4][5][6]

what new attacks give discourse

VI. Brute Force

which 32bit field

One method to guess the 32 bit field required to poison the DNS cache is to simply try every possible combination. This method has its limitations since there is only a short time period where the attacker can send "false" answers to the target device. As soon as the legitimate answer comes back from the resolver and the device accepts it, the attack window is over. The attacker then has to wait until the device sends another request, either for a new domain name or until the cache is flushed and the device has to request again. Another limiting factor is the network speed of the network the target is in. The slower the network, the less spoofed answers can be sent to the target in the same amount of time. Which of course means that you can try even less combinations before the real answer comes back. If the attacker has access to the target device and the device is running Windows there is a workaround. In Windows the DNS cache is shared with all users. And every user can flush the DNS cache. So every user without administrator access can make a DNS request and then try to poison the cache. If the attack fails the user can simply flush the DNS cache and

threat model

try again. When they succeed, they simply wait until an administrator uses the device and waits until the user accesses the malicious website. When this happens the attacker can run malicious code on the target.

arent we already on the target ?

As can be seen, this method of poisoning the DNS cache can take a long time and thus is not really suitable for real world application. But there are other method that are based on the brute force approach and improve on it in different ways.

VII. Predicting the Random Generator

One method to poison the DNS cache is to build upon the brute force approach by not trying to predict the 16 bit field without any knowledge.The Linux kernel random generator consists of four registers that xor'ed with one another. All four registers are randomly seeded at startup. Every time an operation that requires a random number is executed one is added to each register. On top of that the first register is randomly seeded again every 30 to 60 seconds. For an attack to succeed the attacker has to find out the state that the generator is in and make a new request. Then send the spoofed answer to the device before the real answer comes back. By first sending 100.000's of requests to the target device it is possible to extract the position the random generator is currently in. This process worked, because the range the random numbers are in is known if the attacker knows the version of linux the target is running. The numbers the generator produces are then used for the UDP port generation and the TXID label. There are obviously challenges to this technique. First of all every physical core in the CPU has its own random generator. So if the attacker finds out the random generator of core one but when the attacker makes their move to send their request core two might answer that request and the information previously gained is useless. Which means that the amount of request the attacker first has to send to the target depends on how many cores the target has. The more cores the more requests must be sent. But since requests can nowadays be handled very fast this isn't a problem for devices up to ten cores. And is well possible within 30 seconds. Should the guess still be incorrect or should another core answer than anticipated. The attacker can just repeat the process as long as they want [7].

general introduction to rng on pcs

long time when rng is changed every 30s

after prediction what happens?

VIII. IP Fragmentation

Another approach that can also be used with the previous described methods is IP fragmentation. IP fragmentation describes the process of very long answers being split up into multiple fragments. This is helpful for DNS cache poisoning since only the first fragment needs the UDP port and the TXID label to be correct to be accepted by the device. Every other fragment afterwards only needs the correct UDP port number to be accepted. This helps the attacker immensely since they can just wait until the legitimate response comes back and the first fragment is accepted by the target. Then the attacker needs to intercept the remaining fragments and prevent them from reaching the system. Then the attacker can send their fake fragments to the target which can contain anything the attacker wants. And all the attacker needs to guess is the correct port number which of course makes it way more likely to guess the correct one [8][9].

example? intercepting

IX. Malware

The maybe simplest approach to DNS cache poisoning is to simply run a malicious program on the target resolver. This is of course also the unlikeliest since the target needs to be already compromised in order for this method to work. But to poison the DNS cache, the user that runs the malware does not even need admin access to the system. Since on operating systems like Windows the DNS cache can be written and flushed by every user.

repeat from before ?

[1] Zheng Wang, A Revisit of DNS Kaminsky Cache Poisoning Attacks

[2] Sooel Son and Vitaly Shmatikov, The Hitchhiker's Guide to DNS Cache Poisoning

[3] Team Cymru, Incident Response Guide to the Kaminsky DNS Cache Poison Exploit

[4] Taejoong Chung, Roland van Rijswijk-Deij, Balakrishnan Chandrasekaran, David Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove and Christo Wilson, A Longitudinal, End-to-End View of the DNSSEC Ecosystem

[5] Tho Le, Roland van Rijswijk-Deij, Luca Allodi and Nicola Zannone, Economic Incentives on DNSSEC Deployment: Time to Move from Quantity to Quality

[6] Wilson Lian, Eric Rescorla, Hovav Shacham and Stefan Savage, Measuring the practical impact of DNSSEC Deployment

[7] Amit Klein, Cross Layer Attacks and How to Use Them (for DNS Cache Poisoning, Device Tracking and More)

[8] Amir Herzberg and Haya Shulman, Fragmentation Considered Poisonous

[9] K. Fujiwara, Measures against cache poisoning attacks using IP fragmentation in DNS