SECTION 1 - ARTICLE SUMMARY

The security and stability of Internet-based applications and web ecosystem fundamentally relies on the trustworthiness of Certificate Authorities (CAs), which ensures that these services are trusted and communication to them is secure. The CAs vouch for the reliability of a service by issuing a digital certificate that binds a domain name to a public key of the service. Upon receiving the request for a certificate for domain, a CA validates that the server issuing the request owns and controls the domain for which it requests the certificate. After a successful ownership validation, the CA issues the certificate. The certificate contains, among others, the public key of the requesting server and the requested domain and is signed by the private key of the CA. The server then uses this certificate to prove its identity to clients in the Internet. The clients use the server's public key in the certificate to establish a secure (encrypted and authenticated) connection to the server. Therefore, it is vital to correctly verify the domain's ownership by the CA during the certificate issuance process, such that clients and services will communicate with genuine site and not an impostor.

One way that CAs can use to verify ownership of domains is through Domain Validation (DV). The idea behind DV is that only the owner of the domain can receive the communication sent to the services within his domain and can respond to them, therefore can legitimately prove the ownership.

In this work, the researchers propose two main developments. The first is an offensive development which allows, for the first time, an off-path attacker to trick the CA into issuing a certificate to domain he does not own. The second is a new defensive mechanism called Domain Validation++ (DV++), an innovative distributed system of agents which preserves the benefits of standard DV while providing resilience against off-path and MitM attackers. We will now elaborate on these two developments.

The offensive development is based upon off-path attacker, that generates packets and spoofs IP addresses which can be used to leverage DNS cache poisoning of the CA's DNS cache resolver, and tricks it into issuing fraudulent certificates for domains the attacker does not legitimately own. Issuing a certificate requires a client to submit a Certificate Signing Request (CSR), which contains information such as organisation name, domain name, country, public key and more. A CA then uses the submitted CSR to authenticate the domain ownership by the applicant and subsequently to issue the certificate.

During the attack a spoofed DNS record is injected into CA's cache resolver, mapping a domain to an attacker controlled IP address. As a result, the DV process is performed against the attacker-controlled host, to which he can legitimately respond, being disguised as a domain's owner and eventually obtain a valid certificate.

The time window for DNS cache poisoning attack is initiated once a DNS request is issued from the CA's cache resolver to a nameserver. This window ends either when a timeout event occurs or when a correct response arrives. In order to craft a correct response, the attacker has to guess the challenge-response authentication parameters, such as source port and DNS Transaction Identifier (TXID) in the DNS request. The way in which an off-path attacker can spoof a record in the resolver's cache is not by attempting to guess these parameters, but instead by exploiting IPv4 packet fragmentation to overwrite the relevant fields in the genuine DNS response from the nameserver, with malicious values. The core idea of the attack is that the attacker 'convinces' the nameserver to fragment responses back to the CA cache resolver, by issuing ICMP fragmentation error, which enforces the nameserver to respond with a

fragmented packets. Before issuing this error, the attacker calculates the sizes of responses from the nameserver and the offset where the fragmentation should occur. He then sets the MTU value accordingly to ensure that the second fragment indeed contains all the required records. The second fragment contains either the additional or also the authority parts of a DNS response.
The attack utilizes fragmentation and tricks the receiving resolver into reassembling the first fragment from the genuine response from the nameserver with the second fragment generated by the attacker. This allows to bypass the challenge-response authentication fields used by the DNS resolvers, since they are echoed back by the nameserver in the first fragment.
Now, we will describe the full attack procedure: 1.The attacker sends the victim DNS resolver a spoofed second fragment that is cached by the receiving resolver in the IP defragmentation cache, waiting for the rest of the fragments to arrive. 2.The attacker uploads a CSR form requesting certification for the victim's domain. 3. The DNS resolver at the CA issues a DNS request to the nameserver of the victim's domain asking for an IP address of the victim's email server. 4. The attacker issues an ICMP fragmentation needed packet to ensure that the response is indeed fragmented. 5. The victim's nameserver sends a fragmented response. The first fragment contains the entropy (resolver's source port and TXID are copied by the nameserver to the response) and some of the DNS payload, such as the question, answer and part of the authoritative section, while the second fragment contains some (or complete) records of the authoritative section. The first fragment is then reassembled with the spoofed second fragment that was already in the cache of the DNS resolver and they both leave the cache, and are passed on to the UDP layer. The legitimate second fragment will not have any first fragment to reassemble with, since when it arrives, the legitimate first fragment have already been reassembled with the spoofed second fragment and left the buffer. At this point, the attacker can complete the DV process as if he legitimately owns the victim domain.
The researchers proposed a defensive mechanism against this attack called DV++, a decentralised and innovative system that utilizes the existing infrastructure to validate claims of domain ownership. DV++ makes use of distributed agents, which perform independently DV processes from multiple vantage points. The attacker model is a malicious ISP, that passively collects the traffic that traverses its networks. The attacker can also actively try to attract traffic from other networks by performing BGP prefix hijacking. The security against MitM attackers is achieved by placing the components in different networks, which does not traverse overlapping paths. To pass a DV++ validation, domain owners must prove their ownership to a majority of the certificate agents in a fully automated manner by responding to queries sent by the agents for the resource records in the domain. The agents are synchronised with an orchestrator module which is located on the CA network. During the domain validation process, all the agents receive from the orchestrator the domain and the record that need to be queried. The agents send DNS requests to the nameservers in their network, requesting the record. As soon as response arrives, the agent sends it to the orchestrator. When more than 50% of the responses arrive, and they match, the orchestrator returns success, otherwise failure. The number of correct responses from the agents is a parameter that the CA can configure. At last, as said before, to launch a successful attack, the attacker has to spoof correct responses for the majority of the DNS requests issued by the DV++ agents. It is highly

unlikely that even a strong nation state attackers, such as security agencies, could successfully conduct this kind of attack since even they do not control all Internet networks and paths.

SECTION 2 - CRITICAL READING

In this section, we will criticise the paper by listing points that we have found troublesome regarding the aforementioned offensive and defensive developments. For each point, we will start with a relevant theoretical explanation and then describe our critical view. First, we will focus on the proposed attack relevant points and then the defensive mechanism point.

MTU VALUE & ICMP FRAGMENTATION

One of the main components of the attack is that an off-path attacker sends the nameserver an ICMP fragmentation error needed message that indicates to fragment the packets it sends back to the CA cache resolver. Fragmentation is done by the network layer when the maximum size of a datagram is greater than the maximum size of data that can be held in a frame i.e., its Maximum Transmission Unit (MTU). The network layer divides the datagram received from transport layer into fragments, such that data flow is not disrupted. In this process, the packets break into smaller pieces (fragments), so that the resulting pieces can pass through a link with a smaller maximum transmission unit (MTU) than the original packet size. MTU can vary based on configurations and interfaces. If a router's interface MTU is smaller than a packet that must be forwarded, the router fragments the packets into smaller packets. Once reaching the desired destination, the fragments are reassembled by the receiving host.
ICMP (Internet Control Message Protocol) is an error-reporting protocol in which network devices like routers use to generate error messages to the source IP address, when network problems prevent delivery of IP packets. This is a network layer protocol that provides troubleshooting, control and error message services. ICMP is most frequently used in operating systems for networked computers, where it transmits error messages. The ICMP packet does not have source and destination port numbers because it was designed to communicate network-layer information between hosts and routers, not between application layer processes.
Thus, the attack model must have that the DNS resolver and the CA's network devices will allow fragmented responses, in order to succeed. However, a serious let down for this attack is that CA's servers can be configured to block fragmented traffic, which will cause the attack to fail. We will elaborate on blocking fragmentation and provide relevant simulation in section 3. In addition, the attack contains a preprocess phase, in which the attacker performs measurements and calculations, which will be used to set the values in the ICMP fragmentation needed error message. These calculations will be used to construct the spoofed second fragment, which will be sent to CA's cache resolver. In this phase, the attacker measures the responses' sizes from the nameserver and calculates the offset where the fragmentation should occur. Then, the MTU in the ICMP fragmentation needed error message is set accordingly in order to ensure that the records, which the attacker replaces with spoofed records, will be stored in the second fragment.
Therefore, we believe that constantly altering the MTU values randomly during a communication with the nameserver would prevent the attacker the possibility to calculate the required offset and values.

Therefore doing this would prevent the attacker from crafting a valid second fragment that will be accepted by the CA cache resolver, and will cause the attack to fail. We will elaborate on altering the MTU value and provide relevant simulation in section 3.
Moreover, an attacker which makes an attempt to measure the MTU on the path between the cache resolver to the nameserver using ICMP fragmentation error messages in order to craft his spoofed records into the second fragment, will face constantly and randomly updating MTU, which will prevent him from doing so.

UDP CHECKSUM CALCULATION

Since the spoofed second fragment contains different records than the genuine second fragment, the attacker needs to adjust the UDP checksum to the correct value. So, when calculating it using the first genuine fragment, it will have the same value as stored in the UDP header. If the values differ, then the UDP datagram is discarded. The attacker can ensure that the values remain the same as follows: he sends a request to the nameserver and receives a response in two fragments. Then he calculates the UDP checksum value of the second fragment, alters the second fragment and calculates the checksum of the altered second fragment. The difference value in the checksum of both second fragments is the value the attacker needs to add (or remove). This is a simple computation since the attacker knows the original value, know what bytes were modified, can therefore efficiently compute the difference and add (or remove) it to the value of UDP checksum of second fragment. The packet contains an IP header which consists of source address, destination address, protocol and UDP length. The UDP checksum is performed over the entire payload, other fields in the header and some fields from the IP header. A pseudo-header is constructed from the IP header in order to perform the calculation (which is done over this pseudo-header, the UDP header and the payload). The reason the pseudo-header is included is to catch packets that have been routed to the wrong IP address. Therefore, even if the attacker tries to use this approach to calculate the correct checksum value and inject it to the second spoofed fragment, using randomly changing MTU will cause him to face a significant challenge that is unlikely he will be able to overcome. This is since the checksum depends (among the rest) on the size of the data within the payload, which is constantly changing in a non-repeating pattern, for the reason the MTU value varies periodically. Thus, we argue that by using periodically alternating MTU values on the CA's routers, the attacker will not be able to make the DNS cache resolver reassemble the first genuine fragment without the second spoofed one being discarded because of the UDP checksum mismatch. Thus, the attack will fail. As said above, we will elaborate on altering MTU values and provide a simulation in section 3.

INACCURATE FRAGMENTATION LOCATION

Some servers fragments are not exactly at the location specified in the MTU in ICMP fragmentation needed error message. For instance, when signalling MTU of 512 bytes, some servers may fragment at 500 bytes. This might cause trouble to attacker which makes an attempt to figure out where the next fragmentation will occur and force him to make the measurements and calculations to figure out the patterns all over again, in cases of attacking new targets.

IP ID'S

In IPv4, the Identification (ID) field is a 16-bit value that is unique for every datagram for a given source address, destination address, and protocol, such that it does not repeat within the maximum datagram lifetime (MDL). As currently specified, all datagrams between a source and a destination of a given protocol must have unique IPv4 ID values over a period of this MDL, which is typically interpreted as two minutes and is related to the recommended reassembly timeout. This uniqueness is currently specified for all datagrams, regardless of fragmentation settings. The primary purpose of the ID Field is in support of fragmentation and reassembly. Upon arrival to the receiver, the fragments of an IP packet are stored in an IP defragmentation cache, by default for 30 seconds. The receiver uses the IP ID value in the fragments to identify all the fragments of the same original IP packet (they all have the same IP ID value). Then, uses the offset field to reassemble their payload together. The receiver knows that all the fragments of the original IP packet have arrived by checking that the fragment with the lowest offset has a "More Fragments" (MF) value of zero. When that attacker sends spoofed second fragment, it should contain the correct IP ID value, which is the same value as the genuine IP packet sent by the nameserver. The attacker must predict this value correctly, otherwise, when the first fragment and the spoofed second fragment have different IP ID values, the receiving OS will not reassemble them together. For this reason, we propose to use random IP ID's during the communication of the CA's cache resolver with the nameserver, in order to decrease the chances of the attacker to predict the correct IP ID value. We will discuss random IP ID's in section 3.

INITIATION TIMING OF DV PROCESS

Now we will discuss proposed attack's timing issues, which occur at the submission of the CSR in the CA server. When submitting a CSR for a new domain, an off-path attacker cannot observe the communication between the DNS resolver and the nameservers. This introduces potential challenges regarding attack's timing. The attacking host initiates the CSR process for a given victim domain and its subdomains, if exist. After submitting the CSR, the attacker selects the DV method that it wishes to pass (CAs typically support a number of DV methods). Since the attacker cannot control the timing when the validation process will be initiated, it must periodically transmit spoofed second fragment to the CA's cache resolver until it starts. Finally, if the attacker receives a signed certificate for the resource that was requested, he can deduce that the attack has been accomplished successfully. In section 3, we will propose a simple defensive mechanism which makes the attack harder based upon this timing issue, which involves random timing of the initiation of the DV process.

OPERATION OF DNS REQUESTS

We will now discuss the DNS request that the CA's cache resolver sends to the nameserver once the DV process is initiated and elaborate the issues that might occur in a realistic scenario during initiation of the proposed attack. For the attack to work, the request has to arrive to a nameserver which is controlled by the attacker, that should return a fragmented response to the cache resolver with the answer. The attacker will try to spoof the second fragment with a domain name that is mapped to an IP address controlled by him. Prior to this, the attacker has to perform measurements and calculations of the

responses sizes and statistics that are unique to his nameserver, in order to figure out the right offset and to understand where exactly the fragmentation would happen. That is, the attacker relies on the fact that the request indeed arrives to his nameserver. However, in reality, since we are discussing an off-path attacker which can not force traffic to traverse certain paths, he has no way of guaranteeing that the request will indeed arrive to his nameserver and not to a different one. If the request arrives to a nameserver which is not under attacker's control, he has no way of analyzing the responses sizes and the statistics he needs, since he will not be able to observe this traffic. In reality, it is highly likely that if the requested domain is a popular one that has reasonable amount of traffic (and it's record is not already stored in CA's resolver's cache), the DNS request will arrive a nameserver located within the CA's network, which the required record is already located in his cache. Thus, the request does not even leave the CA's network to an external nameserver that might be under the attacker's control. This is since reliable CA's networks typically contain several nameservers within them, which already contain records for popular domains in their cache. In case of which the required record is not in an internal nameserver, the DNS query process searches for an external nameserver which holds the answer. Even then, it is likely that the request arrives a nameserver which is not under attacker control, if the required domain is popular. This is since with high probability,there will be many nameservers which hold the answer, and it is unlikely that the attacker controls all of them or even a large portion of them. Because it is an off-path attacker, he can not even try to force the traffic to pass through his nameservers. In case the requested domain is a new or unfamiliar one, with a small amount of traffic in it, there are higher chances that the request will arrive to the attacker's nameservers, because it is unlikely that the record which contains it's details will be stored in any of the nameservers inside or outside the CA's network. In this case, the attacker will be able to analyze the traffic and get the statistics he needs. Of course, there are non-zero chances that even this domain's records will be stored in a nameserver which is not under attacker's control, but they are relatively low. To conclude, in a realistic scenario, it is unlikely that the resolver's request will arrive to any of the attacker nameservers if the domain is popular or familiar, therefore the attack will fail. If the domain is an unfamiliar one with low traffic, there are greater chances that the request will indeed arrive to the attacker's nameservers and he will be able to initiate the rest of the attack and try to poison the resolver's records. However, even that is not guaranteed, not to mention the fact that in this case the domain is a poor quality target for the attacker, since it is an unpopular domain with low users' traffic.

DV++ AGENTS AGREEMENT PARAMETER

We will now discuss and criticize the innovative defensive mechanism proposed by the researchers, known as DV++. This defensive mechanism preserves the benefits of current DV while attempting to provide resilience against MitM attackers. The developers aim to ensure that the integration of the new mechanism would not require any changes to the CA infrastructure and functionality, and that it should be easy to deploy by the CA's.

In the defensive scenario, the attacker obtains MITM capabilities and can be present on some networks but does not control the entire Internet, such as a malicious ISP, that passively collects the traffic that

traverses its networks. The attacker can also actively try to attract traffic from other networks by performing BGP prefix hijacking.

DV++ is a decentralised mechanism, that utilises the existing Internet infrastructure to validate claims of domain's ownership. In contrast to the centralised validation performed by the CAs with DV, DV++ is based on a flat hierarchy with several equally trusted certification agents. To pass a DV++ validation, domain owners must prove their ownership to a majority of the agents in a fully automated manner by responding to queries sent by the agents for the resource records in the domain. The agents are synchronised with an orchestrator module, which is located on the CA network and they use HTTPS for their communication. The agents are configured on different physical availability networks. The selection is done such that the agents are located in different networks, whose routes do not intersect. Similarly to DV, DV++ authentication is initiated by the CA following a submission of a CSR by the applicant. During this process, queries are sent to the agents, which perform look-ups in the target domain. Once the majority of the responses are received by the orchestrator, they are processed. The number of correct responses from the agents is a parameter that the CA can configure. We will now discuss this parameter's meaning and explain why it poses a security risk.

As described above, in order to launch a successful attack on DV++, the attacker must provide correct spoofed responses for more than 50% of the DNS requests issued by the DV++ agents. However, the number of correct responses from these agents is a parameter that the CA can configure and saved inside it's server. We believe that this is a major weakness in DV++ . If an attacker is able to access this parameter and reconfigure it to a much lower value, say 5%, he can easily bypass the security of DV++. This is since the core idea of DV++ is that even if the attacker can corrupt some agents and control some networks or path, it cannot control all the paths and networks on the Internet or corrupt all the agents within the DV++ system. Even strong nation state attackers, such as security agencies, do not control all Internet networks and paths. For this reason, changing this parameter to a low value will allow an attacker with MITM capabilities that corrupts relatively small number of agents or few traffic paths to complete the validation process and successfully issue fraudulent certificate.

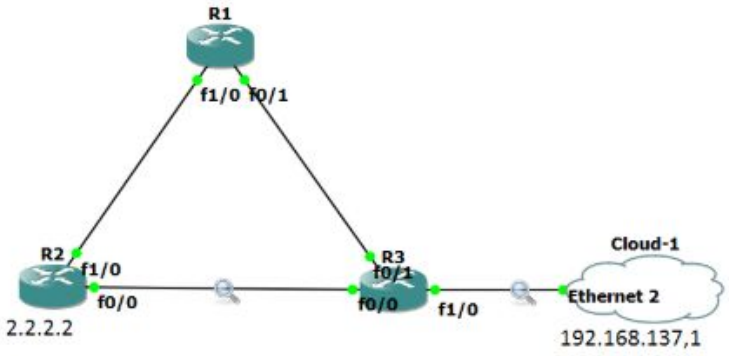
SECTION 3 - EPSILON STEP

In this section we will implement a small step towards several points that we raised in section 2, considering researchers' proposed attack. We will do so by explaining and simulating improvements / solutions that we believe can be useful concerning the critical view we raised towards these points.

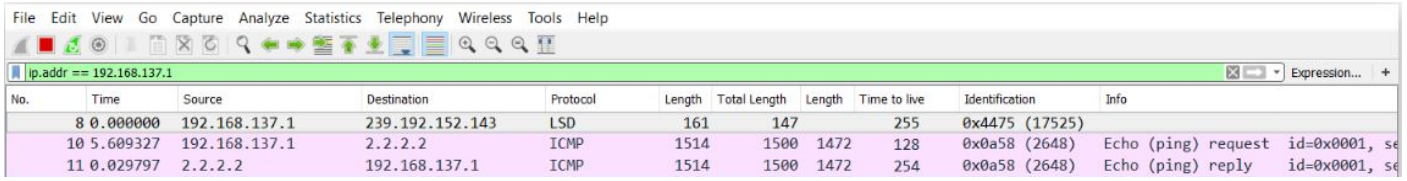
RANDOMLY CHANGING MTU VALUES

The MTU is set through router interface settings, and can be changed automatically using a script that interacts with it. Changing this value in one or more routers within the CA's network to random values between the available range periodically, enforces random changes in the sizes of fragments which traverses this path. As a result, the sizes of the fragmented responses alter periodically as well. Thus, an attacker which controls a nameserver and observes the outgoing traffic in order to figure out a pattern which is used to calculate the offset where the fragmentation should occur, will face a significant and complex challenge. If the path contains more than one router on the CA's network, such that each router

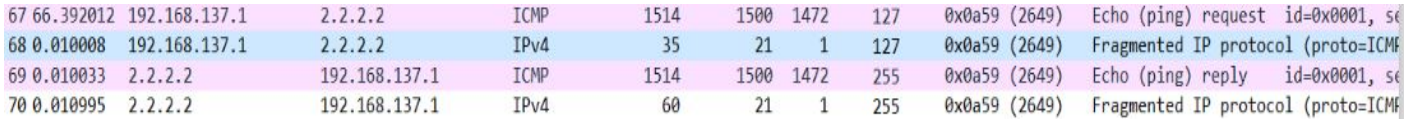
has a different random MTU which periodically alters, it will get even harder, as the fragmented packets keep fragmenting even further on each link on the path with lower MTU. Thus, the odds the attacker will succeed in injecting the spoofed records into the second fragment decreases, since its an off-path attacker that can not observe the traffic within the CA's network. In addition, as we explained in section 2, the UDP checksum calculation gets harder, as it depends on packets payload sizes' pattern, that randomly alters . Now, we will provide a simulation of changing MTU's sizes and observe the changes in packets' sizes. We will demonstrate it using the following architecture: Cloud 1 is where our PC is with ip address 192.168.137.1, a windows 10 machine, from which we ping router R2 which has an ip address of 2.2.2.2.



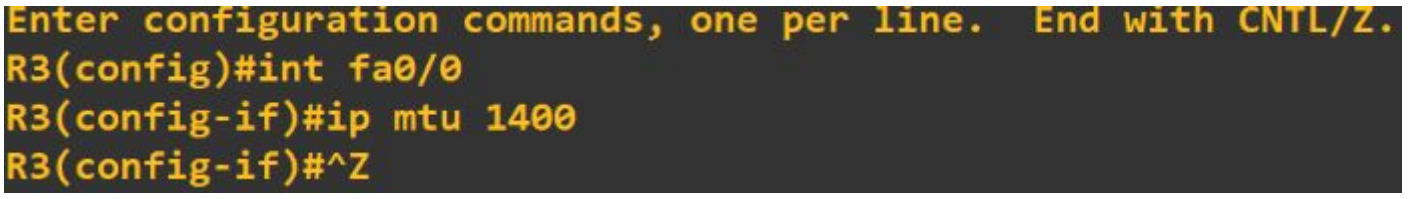
Using the command "netsh int ipv4 show subinterface" we see that our current MTU is 1500. If we ping 2.2.2.2 and sniff the traffic using Wireshark, we will get:



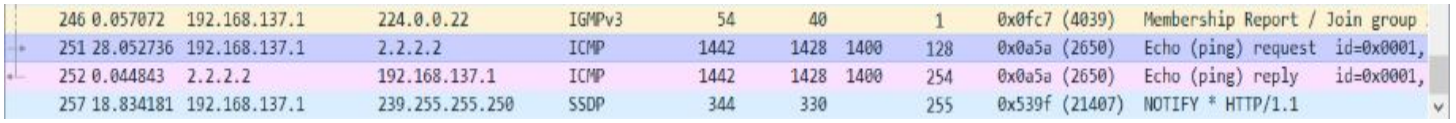
That is 1514 bytes for total packet's length and 1472 bytes for ICMP payload. If we ping 2.2.2.2 with 1 more byte for the payload, meaning 1473 bytes using "ping 2.2.2.2 -n 1 -l 1473" and sniff the traffic between R3 to R2, we will get:



We can observe the fragmentation: in ID 2649 first packet is 1514 bytes, 1472 payload and the second is 35 bytes, 1 byte payload. Now we will set the MTU in R3 to 1400:



So if we will ping with an ICMP payload of 1400 with "ping 2.2.2.2 -n 1 -l 1400", the packet will be fragmented when it reaches R3, because the total packet size will be 1442 bytes as shown below. First, before reaching R3 at ID 2650:



Second, after reaching R3 in same ID:

70	0.010995	2.2.2.2	192.168.137.1	IPv4	60	21	1	255	0x0a59 (2649)	Fragmented IP protocol (proto=ICMP
309	365.0900…	192.168.137.1	2.2.2.2	ICMP	1410	1396	1368	127	0x0a5a (2650)	Echo (ping) request id=0x0001, se
310	0.010876	192.168.137.1	2.2.2.2	IPv4	66	52	32	127	0x0a5a (2650)	Fragmented IP protocol (proto=ICMP
311	0.010996	2.2.2.2	192.168.137.1	ICMP	1442	1428	1400	255	0x0a5a (2650)	Echo (ping) reply id=0x0001, se

We observe that the packet splits into two packets, one 1396 bytes and the other 52 bytes for payload sizes, such that the total sizes are 1410 and 66 bytes. To conclude, we can see that randomly altering the MTU value will change the payload sizes as well, and thus makes it hard for an attacker to calculate where the fragmentation will occur and the correct UDP checksum for each possible MTU value, a fact which increases the chances of the proposed attack to fail.

BLOCKING ICMP FRAGMENTATION

Configuring the servers to block fragmented packets can be done easily by the admins of the CA's DNS resolver and will disrupt this attack, as spoofed records need to be injected into the second fragment in order for it to work. The way to block fragmentation is fairly simple and can be done by setting the 'DF' (Don't Fragment) bit on.

An example of how it can be done in C:

```
setsockopt(sockfd, IPPROTO_IP, IP_MTU_DISCOVER, &val, sizeof(val));

where 'val' is one of:
#define IP_PMTUDISC_DO      0
#define IP_PMTUDISC_DONT    1
#define IP_PMTUDISC_PROBE   2
```

Where, For datagram sockets:

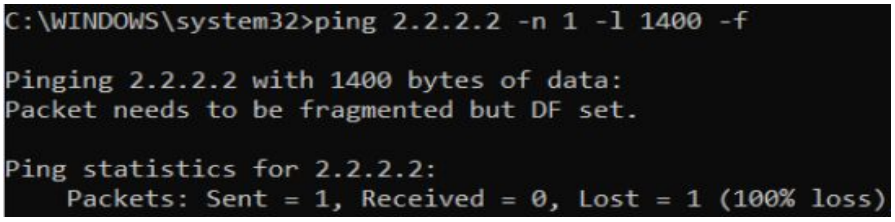
- **IP_PMTUDISC_DO** will force all outgoing packets to have the DF bit set and an attempt to send packets larger than path MTU will result in an error.
- **IP_PMTUDISC_DONT** will force all outgoing packets to have the DF bit not set, and packets will be fragmented according to interface MTU.
- **IP_PMTUDISC_PROBE** will force all outgoing packets to have the DF bit set, and an attempt to send packets larger than interface MTU will result in an error.

We will now simulate sending packets with the DF bit on, such that packet size is larger than the MTU:

296	67.966392	192.168.137.1	2.2.2.2	ICMP	1442	1428	1400	128	0x0a5c (2652)	Echo (ping) request id=0x0001, seq=151/38656, ttl=12
297	0.007001	192.168.137.2	192.168.137.1	ICMP	70	56,1428		255,127	0x020a (522),0…	Destination unreachable (Fragmentation needed)
307	49.020391	192.168.137.1	239.255.255.250	SSDP	216	202		1	0x53a8 (21416)	M-SEARCH * HTTP/1.1
308	1.002356	192.168.137.1	239.255.255.250	SSDP	216	202		1	0x53a9 (21417)	M-SEARCH * HTTP/1.1
309	1.000816	192.168.137.1	239.255.255.250	SSDP	216	202		1	0x53aa (21418)	M-SEARCH * HTTP/1.1
310	1.002009	192.168.137.1	239.255.255.250	SSDP	216	202		1	0x53ab (21419)	M-SEARCH * HTTP/1.1

```
> Frame 297: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface 0
> Ethernet II, Src: c0:03:13:64:00:10 (c0:03:13:64:00:10), Dst: 02:00:4c:4f:4f:50 (02:00:4c:4f:4f:50)
> Internet Protocol Version 4, Src: 192.168.137.2, Dst: 192.168.137.1
v Internet Control Message Protocol
    Type: 3 (Destination unreachable)
    Code: 4 (Fragmentation needed)
```

We get an ICMP unreachable error: "Code 4: fragmentation needed", because the next hop MTU is 1300 and our packet size is 1428. This error will get back to the application and note it to send smaller packets, while discarding the currently traversing packets:



Thus, if CA's servers are configured to block ICMP fragmentation, the proposed attack will fail.

RANDOM IP ID'S

Random IP ID's works by maintaining multiple counters by the server, and continually checking for collisions, i.e., that it does not select an IP ID that was already allocated. The success probability of the attack in case of a single fragment is $1/2^{16}$. The attacker can increase its success probability by sending multiple fragments, each with a different IP ID value. To deal with that, we can configure the server's OSes to limit the number of fragments that can be sent, e.g., Windows to 100 fragments, recent Linux versions to 64 and older allow several thousand fragments. The attacker can try to circumvent the OSes obstacle by sending multiple fragments from different IP addresses to have more chances to hit the correct IP ID. To cope with that, the CA's DNS servers can be configured to block connections to addresses that send unreasonable amount of packets with different IP IDs during the session of a single DV process with the resolver. Thus, we believe that configuring the CA's servers to use random IP ID's values will decrease the chances of initiating a successful attack.

RECOGNITION OF MALICIOUS FRAGMENTS BEFORE INITIATION OF DV PROCESS

We will propose a simple defensive mechanism which makes the attack harder based upon the fact that the attacker can not observe when the DV process will be initiated. Once the CA received a submission of a new CSR, the DV process will not start immediately, i.e. the CA's cache resolver will not issue DNS request to the nameserver right away, but at a random time afterwards. Meanwhile, the resolver will listen to incoming packet fragments that contain the same source IP addresses. These fragments will be constantly discarded, because of the missing appropriate sequence of fragments which should appear before and after them. If a relatively large amount of such fragments appear for no good reason before the DV process even began, the resolver can be configured to update the CA's servers to block this source IP address. Furthermore, the CA will review the domain name that appeared on the repeating packets (the victim domain that the attacker attempts to poison its record on the resolver), deduce which of the CSR's that have been submitted contains it and cancel its DV process before it even started. Because the attacker keeps sending spoofed second fragment over and over again from the moment of the CSR's submission, using this method will expose the attacker, block his IP address for future communication and cancel his DV process. Thus, the proposed attack will fail.