IPSecME Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 6, 2016

Y. Nir
Check Point
V. Smyslov
ELVIS-PLUS
July 5, 2015

Protecting Internet Key Exchange (IKE) Implementations from Distributed Denial of Service Attacks

draft-ietf-ipsecme-ddos-protection-02

Abstract

This document recommends implementation and configuration best practices for Internet-connected IPsec Responders, to allow them to resist Denial of Service and Distributed Denial of Service attacks. Additionally, the document introduces a new mechanism called "Client Puzzles" that help accomplish this task.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 6, 2016.

Table of Contents

1. Introduction

The IKE_SA_INIT Exchange described in section 1.2 of [RFC7296] involves the Initiator sending a single message. The Responder replies with a single message and also allocates memory for a structure called a half-open IKE SA (Security Association). This half-open SA is later authenticated in the IKE_AUTH Exchange, but if that IKE_AUTH request never comes, the half-open SA is kept for an unspecified amount of time. Depending on the algorithms used and implementation, such a half-open SA will use from around 100 bytes to several thousands bytes of memory.

This creates an easy attack vector against an Internet Key Exchange (IKE) Responder. Generating the Initial request is cheap, and sending multiple such requests can either cause the Responder to allocate too much resources and fail, or else if resource allocation is somehow throttled, legitimate Initiators would also be prevented from setting up IKE SAs.

An obvious defense, which is described in Section 5, is limiting the number of half-open SAs opened by a single peer. However, since all that is required is a single packet, an attacker can use multiple spoofed source IP addresses.

Section 2.6 of RFC 7296 offers a mechanism to mitigate this DoS attack: the stateless cookie. When the server is under load, the Responder responds to the Initial request with a calculated "stateless cookie" - a value that can be re-calculated based on values in the Initial request without storing Responder-side state. The Initiator is expected to repeat the Initial request, this time including the stateless cookie.

Attackers that have multiple source IP addresses with return routability, such as bot-nets can fill up a half-open SA table anyway. The cookie mechanism limits the amount of allocated state to the size of the bot-net, multiplied by the number of half-open SAs allowed for one peer address, multiplied by the amount of state allocated for each half-open SA. With typical values this can easily reach hundreds of megabytes.

The mechanism described in Section 3 adds a proof of work for the Initiator, by calculating a pre-image for a partial hash value. This sets an upper bound, determined by the attacker's CPU to the number of negotiations it can initiate in a unit of time.

1.1. Conventions Used in This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. The Vulnerability

If we break down what a responder has to do during an initial exchange, there are three stages:

1. When the Initial request arrives, the responder:

    * Generates or re-uses a D-H private part.

* Generates a responder SPI.

* Stores the private part and peer public part in a half-open SA database.

2. When the Authentication request arrives, the responder:

* Derives the keys from the half-open SA.

* Decrypts the request.

3. If the Authentication request decrypts properly:

* Validates the certificate chain (if present) in the auth request.

Yes, there's a stage 4 where the responder actually creates Child SAs, but when talking about (D)DoS, we never get to this stage.

Stage #1 is pretty light on CPU power, but requires some storage, and it's very light for the initiator as well. Stage #2 includes private-key operations, so it's much heavier CPU-wise, but it releases the storage allocated in stage #1. Stage #3 includes a public key operation, and possibly many of them.

To attack such a server, an attacker can attempt to either exhaust memory or to exhaust CPU. Without any protection, the most efficient attack is to send multiple Initial requests and exhaust memory. This should be easy because those Initial requests are cheap.

There are obvious ways for the responder to protect itself even without changes to the protocol. It can reduce the time that an entry remains in the half-open SA database, and it can limit the amount of concurrent half-open SAs from a particular address or prefix. The attacker can overcome this by using spoofed source addresses.

The stateless cookie mechanism from section 2.6 of RFC 7296 prevents an attack with spoofed source addresses. This doesn't solve the issue, but it makes the limiting of half-open SAs by address or prefix work. Puzzles do the same thing only more of it. They make it harder for an attacker to reach the goal of getting a half-open SA. They don't have to be so hard that an attacker can't afford to solve them - it's enough that they increase the cost of a half-open SAs for the attacker.

Reducing the amount of time an abandoned half-open SA is kept attacks the issue from the other side. It reduces the value the attacker

gets from managing to create a half-open SA. So if a half-open SA takes 1 KB and it's kept for 1 minute and the capacity is 60,000 half-open SAs, an attacker would need to create 1,000 half-open SAs per second. Reduce the retention time to 3 seconds, and the attacker needs to create 20,000 half-open SAs per second. Make each of those more expensive by introducing a puzzle, and you're likely to thwart an exhaustion attack against responder memory.

At this point, filling up the half-open SA database in no longer the most efficient DoS attack. The attacker has two ways to do better:

1. Go back to spoofed addresses and try to overwhelm the CPU that deals with generating cookies, or

2. Take the attack to the next level by also sending an Authentication request.

It seems that the first thing cannot be dealt with at the IKE level. It's probably better left to Intrusion Prevention System (IPS) technology.

On the other hand sending an Authentication request is surprisingly cheap. It requires a proper IKE header with the correct IKE SPIs, and it requires a single encrypted payload. The content of the payload might as well be junk. The responder has to perform the relatively expensive key derivation, only to find that the Authentication request does not decrypt. Depending on the responder implementation, this can be repeated with the same half-open SA (if the responder does not delete the half-open SA following an unsuccessful decryption - see discussion in Section 4).

Here too, the number of half-open SAs that the attacker can achieve is crucial, because each one of them allows the attacker to waste some CPU time. So making it hard to make many half-open SAs is important.

A strategy against DDoS has to rely on at least 4 components:

1. Hardening the half-open SA database by reducing retention time.

2. Hardening the half-open SA database by rate-limiting single IPs/ prefixes.

3. Guidance on what to do when an Authentication request fails to decrypt.

4. Increasing cost of half-open SA up to what is tolerable for legitimate clients.

Puzzles have their place as part of #4.

3. Puzzles

The puzzle introduced here extends the cookie mechanism from RFC 7296. It is loosely based on the proof-of-work technique used in BitCoins ([bitcoins]).

A puzzle is sent to the Initiator in two cases:

- The Responder is so overloaded, than no half-open SAs are allowed to be created without the puzzle, or

- The Responder is not too loaded, but the rate-limiting in Section 5 prevents half-open SAs from being created with this particular peer address or prefix without first solving a puzzle.

When the Responder decides to send the challenge notification in response to a IKE_SA_INIT request, the notification includes three fields:

1. Cookie - this is calculated the same as in RFC 7296. As in RFC 7296, the process of generating the cookie is not specified.

2. Algorithm, this is the identifier of a PRF algorithm, one of those proposed by the Initiator in the SA payload.

3. Zero Bit Count. This is a number between 8 and 255 (or a special value - 0, see Section 8.1.1.1) that represents the length of the zero-bit run at the end of the output of the PRF function calculated over the Keyed-Cookie payload that the Initiator is to send. Since the mechanism is supposed to be stateless for the Responder, either the same value is sent to all Initiators who are receiving this challenge or the value is somehow encoded in the cookie. The values 1-8 are explicitly excluded, because they create a puzzle that is too easy to solve for it to make any difference in mitigating DDoS attacks.

Upon receiving this challenge payload, the Initiator attempts to calculate the PRF using different keys. When a key is found such that the resulting PRF output has a sufficient number of trailing zero bits, that result is sent to the Responder in a Keyed-Cookie notification, as described in Section 3.1.

When receiving a request with a Keyed-Cookie, the Responder verifies two things:

- That the cookie part is indeed valid.

o That the PRF of the transmitted cookie calculated with the transmitted key has a sufficient number of trailing zero bits.

Example 1: Suppose the calculated cookie is fdbcfa5a430d7201282358a2a034de0013cfe2ae (20 octets), the algorithm is HMAC-SHA256, and the required number of zero bits is 18. After successively trying a bunch of keys, the Initiator finds that the key that is all-zero except for the last three bytes which are 02fc95 yields HMAC_SHA256(k, cookie) = 843ab73f35c5b431b1d8f80bedcd1cb9ef46832f799c1d4250a49f683c580000, which has 19 trailing zero bits, so it is an acceptable solution.

Example 2: Same cookie, but this time the required number of zero bits is 22. The first key to satisfy that requirement ends in 960cbb, which yields a hash with 23 trailing zero bits. Finding this requires 9,833,659 invocations of the PRF.

key	Last 24 Hex PRF Digits	# 0-bits	Time To Calculate
00	0cbbbd1e105f5a177f9697d4	2	0.000
08	34cdedf89560f600aab93c68	3	0.000
0b	6153a5131b879a904cd7fbe0	5	0.000
2b	0098af3e9422aa40a6f7b140	6	0.000
0147	c8bf4a65fc8b974046b97c00	10	0.001
06e2	541487a10cbdf3b21c382800	11	0.005
0828	48719bd62393fcf9bc172000	13	0.006
0204a7	3dce3414477c2364d5198000	15	0.186
185297	c19385bb7b9566e5fdf00000	20	2.146
69dc34	1b61ecb347cb2e0cba200000	21	9.416
960cbb	e48274bfac2b7e1930800000	23	13.300
01597972	39a0141d0fe4b87aea000000	25	30.749
0b13cd9a	00b97bb323d6d33350000000	28	247.914
37dc96e4	1e24babc92234aa3a0000000	29	1237.170
7a1a56d8	c98f0061e380a49e00000000	33	2726.150

Table 1: COOKIE=fdbcfa5a430d7201282358a2a034de0013cfe2ae

The figures above were obtained on a 2.4 GHz single core i5. Run times can be halved or quartered with multi-core code, but would be longer on mobile phone processors, even if those are multi-core as well. With these figures 20 bits is believed to be a reasonable choice for puzzle level difficulty for all Initiators, with 24 bits acceptable for specific hosts/prefixes.

3.1. The Keyed-Cookie Notification

To be added

3.2. The Puzzle-Required Notification

To be added

4. Retention Periods for Half-Open SAs

As a UDP-based protocol, IKEv2 has to deal with packet loss through retransmissions. Section 2.4 of RFC 7296 recommends "that messages be retransmitted at least a dozen times over a period of at least several minutes before giving up". Retransmission policies in practice wait at least one or two seconds before retransmitting for the first time.

Because of this, setting the timeout on a half-open SA too low will cause it to expire whenever even one IKE_AUTH request packet is lost. When not under attack, the half-open SA timeout SHOULD be set high enough that the Initiator will have enough time to send multiple retransmissions, minimizing the chance of transient network congestion causing IKE failure.

When the system is under attack, as measured by the amount of half-open SAs, it makes sense to reduce this lifetime. The Responder should still allow enough time for the round-trip, enough time for the Initiator to derive the Diffie-Hellman shared value, and enough time to derive the IKE SA keys and the create the IKE_AUTH request. Two seconds is probably as low a value as can realistically be used.

It could make sense to assign a shorter value to half-open SAs originating from IP addresses or prefixes from which are considered suspect because of multiple concurrent half-open SAs.

5. Rate Limiting

Even with DDoS, the attacker has only a limited amount of nodes participating in the attack. By limiting the amount of half-open SAs that are allowed to exist concurrently with each such node, the total amount of half-open SAs is capped, as is the total amount of key derivations that the Responder is forced to complete.

In IPv4 it makes sense to limit the number of half-open SAs based on IP address. Most IPv4 nodes are either directly attached to the Internet using a routable address or are hidden behind a NAT device with a single IPv4 external address. IPv6 networks are currently a rarity, so we can only speculate on what their wide deployment will

be like, but the current thinking is that ISP customers will be assigned whole subnets, so we don't expect the kind of NAT deployment that is common in IPv4. For this reason it makes sense to use a 64-bit prefix as the basis for rate limiting in IPv6.

The number of half-open SAs is easy to measure, but it is also worthwhile to measure the number of failed IKE_AUTH exchanges. If possible, both factors should be taken into account when deciding which IP address or prefix is considered suspicious.

There are two ways to rate-limit a peer address or prefix:

1. Hard Limit - where the number of half-open SAs is capped, and any further IKE_SA_INIT requests are rejected.

2. Soft Limit - where if a set number of half-open SAs exist for a particular address or prefix, any IKE_SA_INIT request will require solving a puzzle.

The advantage of the hard limit method is that it provides a hard cap on the amount of half-open SAs that the attacker is able to create. The downside is that it allows the attacker to block IKE initiation from small parts of the Internet. For example, if a certain purveyor of beverages resembling coffee provides Internet connectivity to its customers through an IPv4 NAT device, a single malicious customer can create enough half-open SAs to fill the quota for the NAT device external IP address. Legitimate Initiators on the same network will not be able to initiate IKE.

The advantage of a soft limit is that legitimate clients can always connect. The disadvantage is that a sufficiently resourceful (in the sense that they have a lot of resources) adversary can still effectively DoS the Responder.

Regardless of the type of rate-limiting used, there is a huge advantage in blocking the DoS attack using rate-limiting in that legitimate clients who are away from the attacking nodes should not be adversely affected by either the attack or by the measures used to counteract it.

6. Plan for Defending a Responder

This section outlines a plan for defending a Responder from a DDoS attack based on the techniques described earlier. The numbers given here are not normative, and their purpose is to illustrate the configurable parameters needed for defeating the DDoS attack.

Implementations may be deployed in different environments, so it is RECOMMENDED that the parameters be settable. As an example, most commercial products are required to undergo benchmarking where the IKE SA establishment rate is measured. Benchmarking is indistinguishable from a DoS attack and the defenses described in this document may defeat the benchmark by causing exchanges to fail or take a long time to complete. Parameters should be tunable to allow for benchmarking (if only by turning DDoS protection off).

Since all countermeasures may cause delays and work on the initiators, they SHOULD NOT be deployed unless an attack is likely to be in progress. To minimize the burden imposed on Initiators, the Responder should monitor incoming IKE requests, searching for two things:

1. A general DDoS attack. Such an attack is indicated by a high number of concurrent half-open SAs, a high rate of failed IKE_AUTH exchanges, or a combination of both. For example, consider a Responder that has 10,000 distinct peers of which at peak 7,500 concurrently have VPN tunnels. At the start of peak time, 600 peers might establish tunnels at any given minute, and tunnel establishment (both IKE_SA_INIT and IKE_AUTH) takes anywhere from 0.5 to 2 seconds. For this Responder, we expect there to be less than 20 concurrent half-open SAs, so having 100 concurrent half-open SAs can be interpreted as an indication of an attack. Similarly, IKE_AUTH request decryption failures should never happen. Supposing the the tunnels are established using EAP (see section 2.16 or RFC 7296), users enter the wrong password about 20% of the time. So we'd expect 125 wrong password failures a minute. If we get IKE_AUTH decryption failures from multiple sources more than once per second, or EAP failure more than 300 times per minute, that can also be an indication of a DDoS attack.

2. An attack from a particular IP address or prefix. Such an attack is indicated by an inordinate amount of half-open SAs from that IP address or prefix, or an inordinate amount of IKE_AUTH failures. A DDoS attack may be viewed as multiple such attacks. If they are mitigated well enough, there will not be a need enact countermeasures on all Initiators. Typical figures might be 5 concurrent half-open SAs, 1 decrypt failure, or 10 EAP failures within a minute.

Note that using counter-measures against an attack from a particular IP address may be enough to avoid the load on the half-open SA database and the amount of failed IKE_AUTH exchanges to never exceed the threshold of attack detection. This is a good thing as it

prevent Initiators that are not close to the attackers from being affected.

When there is no general DDoS attack, it is suggested that no Cookie or puzzles be used. At this point the only defensive measure is the monitoring, and setting a soft limit per peer IP or prefix. The soft limit can be set to 3-5, and the puzzle difficulty should be set to such a level (number of zero-bits) that all legitimate clients can handle it without degraded user experience.

As soon as any kind of attack is detected, either a lot of initiations from multiple sources or a lot of initiations from a few sources, it is best to begin by requiring stateless cookies from all Initiators. This will force the attacker to use real source addresses, and help avoid the need to impose a greater burden in the form of cookies on the general population of initiators. This makes the per-node or per-prefix soft limit more effective.

When Cookies are activated for all requests and the attacker is still managing to consume too many resources, the Responder MAY increase the difficulty of puzzles imposed on IKE_SA_INIT requests coming from suspicious nodes/prefixes. It should still be doable by all legitimate peers, but it can degrade experience, for example by taking up to 10 seconds to solve the puzzle.

If the load on the Responder is still too great, and there are many nodes causing multiple half-open SAs or IKE_AUTH failures, the Responder MAY impose hard limits on those nodes.

If it turns out that the attack is very widespread and the hard caps are not solving the issue, a puzzle MAY be imposed on all Initiators. Note that this is the last step, and the Responder should avoid this if possible.

6.1. Session Resumption

When the Responder is under attack, it MAY choose to prefer previously authenticated peers who present a session resumption [RFC5723] ticket. The Responder MAY require such Initiators to pass a return routability check by including the COOKIE notification in the IKE_SESSION_RESUME response message, as allowed by RFC 5723, Sec. 4.3.2. Note that the Responder SHOULD cache tickets for a short time to reject reused tickets (Sec. 4.3.1), and therefore there should be no issue of half-open SAs resulting from replayed IKE_SESSION_RESUME messages

7. Operational Considerations

[This section needs a lot of expanding]

The difficulty level should be set by balancing the requirement to minimize the latency for legitimate initiators and making things difficult for attackers. A good rule of thumb is for taking about 1 second to solve the puzzle. A typical initiator or bot-net member in 2014 can perform slightly less than a million hashes per second per core, so setting the difficulty level to n=20 is a good compromise. It should be noted that mobile initiators, especially phones are considerably weaker than that. Implementations should allow administrators to set the difficulty level, and/or be able to set the difficulty level dynamically in response to load.

Initiators should set a maximum difficulty level beyond which they won't try to solve the puzzle and log or display a failure message to the administrator or user.

8. Using Puzzles in the Protocol

8.1. Puzzles in IKE_SA_INIT Exchange

IKE initiator indicates the desire to create a new IKE SA by sending IKE_SA_INIT request message. The message may optionally contain COOKIE notification if this is a repeated request performed after the responder's demand to return a cookie.

HDR, [N(COOKIE),] SA, KE, Ni, [V+][N+] -->

According to the plan, described in Section 6, IKE responder should monitor incoming requests to detect whether it is under attack. If the responder learns that (D)DoS attack is likely to be in progress, then it either requests the initiator to return a cookie or, if the volume is so high, that puzzles need to be used for defense, it requests the initiator to solve a puzzle.

The responder MAY choose to process some fraction of IKE_SA_INIT requests without presenting a puzzle even being under attack to allow legacy clients, that don't support puzzles, to have chances to be served. The decision whether to process any particular request must be probabilistic, with the probability depending on the responder's load (i.e. on the volume of attack). Only those requests, that contain COOKIE notification, must participate in this lottery. In other words, the responder MUST first perform return routability check before allowing any legacy client to be served if it is under attack. See Section 8.1.3 for details.

8.1.1. Presenting Puzzle

If the responder takes a decision to use puzzles, then it includes two notifications in its response message - the COOKIE notification and the PUZZLE notification. The format of the PUZZLE notification is described in Section 10.1.

<-- HDR, N(COOKIE), N(PUZZLE), [V+][N+]

The presence of these notifications in an IKE_SA_INIT response message indicates to the initiator that it should solve the puzzle to get better chances to be served.

8.1.1.1. Selecting Puzzle Difficulty Level

The PUZZLE notification contains the difficulty level of the puzzle - the minimum number of trailing zero bits that the result of PRF must contain. In diverse environments it is next to impossible for the responder to set any specific difficulty level that will result in roughly the same amount of work for all initiators, because computation power of different initiators may vary by the order of magnitude, or even more. The responder may set difficulty level to 0, meaning that the initiator is requested to spend as much power to solve puzzle, as it can afford. In this case no specific number of trailing zero bits is required from the initiator, however the more bits initiator is able to get, the higher chances it will have to be served by the responder. In diverse environments it is RECOMMENDED that the initiator sets difficulty level to 0, unless the attack volume is very high.

If the responder sets non-zero difficulty level, then the level should be determined by analyzing the volume of the attack. The responder MAY set different difficulty levels to different requestd depending on the IP address the request has come from.

8.1.1.2. Selecting Puzzle Algorithm

The PUZZLE notification also contains identificator of the algorithm, that must be used by initiator to compute puzzle.

Cryptographic algorithm agility is considered an important feature for modern protocols ([ALG-AGILITY]). This feature ensures that protocol doesn't rely on a single build-in set of cryptographic algorithms, but has a means to replace one set with another and negotiate new set with the peer. IKEv2 fully supports cryptographic algorithm agility for its core operations.

To support this feature in case of puzzles the algorithm, that is used to compute puzzle, needs to be negotiated during IKE_SA_INIT exchange. The negotiation is done as follows. The initial request message sent by initiator contains SA payload with the list of transforms the initiator supports and is willing to use in the IKE SA being established. The responder parses received SA payload and finds mutually supported set of transforms of type PRF. It selects most preferred transform from this set and includes it into the PUZZLE notification. There is no requirement that the PRF selected for puzzles be the same, as the PRF that is negotiated later for the use in core IKE SA crypto operations. If there are no mutually supported PRFs, then negotiation will fail anyway and there is no reason to return a puzzle. In this case the responder returns NO_PROPOSAL_CHOSEN notification. Note that PRF is a mandatory transform type for IKE SA (see Sections 3.3.2 and 3.3.3 of [RFC7296]) and at least one transform of this type must always be present in SA payload in IKE_SA_INIT exchange.

8.1.1.3. Generating Cookie

If responder supports puzzles then cookie should be computed in such a manner, that the responder is able to learn some important information from the sole cookie, when it is later returned back by initiator. In particular - the responder should be able to learn the following information:

- Whether the puzzle was given to the initiator or only the cookie was requested.
- The difficulty level of the puzzle given to the initiator.
- The number of consecutive puzzles given to the initiator.
- The amount of time the initiator spent to solve the puzzles. This can be calculated if the cookie is timestamped.

This information helps the responder to make a decision whether to serve this request or demand more work from the initiator.

One possible approach to get this information is to encode it in the cookie. The format of such encoding is a local matter of responder, as the cookie would remain an opaque blob to the initiator. If this information is encoded in the cookie, then the responder MUST make it integrity protected, so that any intended or accidental alteration of this information in returned cookie is detectable. So, the cookie would be generated as:

```
Cookie = <VersionIDofSecret> | <AdditionalInfo> |
                  Hash(Ni | IPi | SPIi | <AdditionalInfo> | <secret>)
```

Alternatively the responder may continue to generate cookie as suggested in Section 2.6 of [RFC7296], but associate the additional information, that would be stored locally, with the particular version of the secret. In this case the responder should have different secret for every combination of difficulty level and number of consecutive puzzles, and should change the secrets periodically, keeping a few previous versions, to be able to calculate how long ago the cookie was generated.

The responder may also combine these approaches. This document doesn't mandate how the responder learns this information from the cookie.

8.1.2. Solving Puzzle and Returning the Solution

If initiator receives puzzle but it doesn't support puzzles, then it will ignore PUZZLE notification as unrecognized status notification (in accordance to Section 3.10.1 of [RFC7296]). The initiator also MAY ignore puzzle if it is not willing to spend resources to solve puzzle of requested difficulty, even if it supports puzzles. In both cases the initiator acts as described in Section 2.6 of [RFC7296] - it restarts the request and includes the received COOKIE notification into it. The responder should be able to distinguish the situation when it just requested a cookie from the situation when the puzzle was given to the initiator, but the initiator for some reason ignored it.

If the received message contains PUZZLE notification, but doesn't contain cookie, then this message is malformed, because it requests to solve the puzzle, but doesn't provide enough information to do it. In this case the initiator SHOULD resend IKE_SA_INIT request. If this situation repeats several times, then it means that something is wrong and IKE SA cannot be established.

If initiator supports puzzles and is ready to deal with them, then it tries to solve the given puzzle. After the puzzle is solved the initiator restarts the request and returns the puzzle solution in a new payload called Puzzle Solution payload (denoted as PS, see Section 10.2) along with the received COOKIE notification back to the responder.

```
HDR, N(COOKIE), [PS,] SA, KE, Ni, [V+][N+]   -->
```

8.1.2.1. Computing Puzzle

General principals of constructing puzzles in IKEv2 are described in Section 3. They can be summarized as follows: given unpredictable string S and pseudo-random function PRF find the key K for that PRF so that the result of PRF(K,S) has the specified number of trailing zero bits.

In the IKE_SA_INIT exchange it is the cookie that plays the role of unpredictable string S. In other words, in IKE_SA_INIT the task for IKE initiator is to find the key K for the agreed upon PRF such that the result of PRF(K,cookie) has sufficient number of trailing zero bits. Only the content of the COOKIE notification is used in puzzle calculation, i.e. the header of the Notification payload is not included.

8.1.3. Analyzing Repeated Request

The received request must at least contain COOKIE notification. Otherwise it is an initial request and it must be processed according to Section 8.1. First, the cookie MUST be checked for validity. If the cookie is invalid then the request is treated as initial and is processed according to Section 8.1. If the cookie is valid then some important information is learned from it or from local state based on identifier of the cookie's secret (see Section 8.1.1.3 for details). This information would allow the responder to sort out incoming requests, giving more priority to those of them, which were created spending more initiator's resources.

First, the responder determines if it requested only a cookie, or presented a puzzle to the initiator. If no puzzle was given, then it means that at the time the responder requested a cookie it didn't detect the (D)DoS attack or the attack volume was low. In this case the received request message must not contain the PS payload, and this payload MUST be ignored if for any reason the message contains it. Since no puzzle was given, the responder marks the request with the lowest priority since the initiator spent a little resources creating it.

If the responder learns from the cookie that puzzle was given to the initiator, then it looks for the PS payload to determine whether its request to solve the puzzle was honored or not. If the incoming message doesn't contain PS payload, then it means that the initiator either doesn't support puzzles or doesn't want to deal with them. In either case the request is marked with the lowest priority since the initiator spent a little resources creating it.

If PS payload is found in the message then the responder MUST verify the puzzle solution that it contains. The result must contain at least the requested number of trailing zero bits (that is also learned from the cookie, as well as the PRF algorithm used in puzzle solution). If the result of the solution contais fewer bits, than were requested, it means that initiator spent less resources, than expected by the responder. This request is marked with the lowest priority.

If the initiator provided the solution to the puzzle satisfying the requested difficulty level, or if the responder didn't indicate any particular difficulty level (by requesting zero level) and the initiator was free to select any difficulty level it can afford, then the priority of the request is calculated based on the following considerations.

- The higher zero bits the initiator got, the higher priority its request should achieve.
- The more consecutive puzzles the initiator solved (it must be learned from the cookie), the higher priority its request should achieve.
- The more time the initiator spent solving the puzzles (it must be learned from the cookie), the higher priority its request should achieve.

After the priority of the request is determined the final decision whether to serve it or not is made.

8.1.4. Making Decision whether to Serve the Request

The responder decides what to do with the request based on its priority and responder's current load. There are three possible actions:

- Accept request.
- Reject request.
- Demand more work from initiator by giving it a new puzzle.

The responder SHOULD accept incoming request if its priority is high - it means that the initiator spent quite a lot of resources. The responder MAY also accept some of low-priority requests where the initiators don't support puzzles. The percentage of accepted legacy requests depends on the responder's current load.

If initiator solved the puzzle, but didn't spend much resources for it (the selected puzzle difficulty level appeared to be low and the initiator solved it quickly), then the responder SHOULD give it another puzzle. The more puzzles the initiator solves the higher would be its chances ro be served.

The details of how the responder takes decision on any particular request are implementation dependant. The responder can collect all the incoming requests for some short period of time, sort them out based on their priority, calculate the number of alailable memory slots for half-open IKE SAs and then serve that number of the requests from the head of the sorted list. The rest of requests can be either discarded or responded to with new puzzles.

Alternatively the responder may decide whether to accept every incoming request with some kind of lottery, taking into account its priority and the available resources.

8.2. Puzzles in IKE_AUTH Exchange

Once the IKE_SA_INIT exchange is completed, the responder has created a state and is awaiting for the first message of the IKE_AUTH exchange from initiator. At this point the initiator has already passed return routability check and has proved that it has performed some work to complete IKE_SA_INIT exchange. However, the initiator is not yet authenticated and this fact allows malicious initiator to perform an attack, described in Section 2. Unlike DoS attack in IKE_SA_INIT exchange, which is targeted on the responder's memory resources, the goal of this attack is to exhaust responder's CPU power. The attack is performed by sending the first IKE_AUTH message containing garbage. This costs nothing to the initiator, but the responder has to do relatively costly operations of computing the Diffie-Hellman shared secret and deriving SK_* keys to be able to verify authenticity of the message. If the responder doesn't keep the computed keys after unsuccessful verification of IKE_AUTH message, then the attack can be repeated several times on the same IKE SA.

The responder can use puzzles to make this attack more costly for the initiator. The idea is that the responder includes puzzle in the IKE_SA_INIT response message and the initiator includes puzzle solution in the first IKE_AUTH request message outside the Encrypted payload, so that the responder is able to verify puzzle solution before computing Diffie-Hellman shared secret. The difficulty level of the puzzle should be selected so, that the initiator would spend substantially more time to solve the puzzle, than the responder to compute the shared secret.

The responder should constantly monitor the amount of the half-open IKE SA states, that receive IKE_AUTH messages, but cannot decrypt them due to the integrity check failures. If the percentage of such states is high and it takes an essential fraction of responder's computing power to calculate keys for them, then the responder can assume that it is under attack and can use puzzles to make it harder for attackers.

8.2.1. Presenting Puzzle

The responder requests the initiator to solve a puzzle by including the PUZZLE notification in the IKE_SA_INIT response message. The responder MUST NOT use puzzles in the IKE_AUTH exchange unless the puzzle has been previously presented and solved in the preceeding IKE_SA_INIT exchange.

<-- HDR, SA, KE, Nr, N(PUZZLE), [V+][N+]

8.2.1.1. Selecting Puzzle Difficulty Level

The difficulty level of the puzzle in IKE_AUTH should be chosen so, that the initiator would spend more time to solve the puzzle, than the responder to compute Diffie-Hellman shared secret and the keys, needed to decrypt and verify the IKE_AUTH request message. On the other hand, the difficulty level should not be too high, otherwise the legitimate clients would experience additional delay while establishing IKE SA.

Note, that since puzzles in the IKE_AUTH exchange are only allowed to be used if they were used in the preceeding IKE_SA_INIT exchange, the responder would be able to estimate the computing power of the initiator and to select the difficulty level accordingly. Unlike puzzles in IKE_SA_INIT, the requested difficulty level for IKE_AUTH puzzles MUST NOT be zero. In other words, the responder must always set specific difficulty level and must not let the initiator to choose it on its own.

8.2.1.2. Selecting Puzzle Algorithm

The algorithm for the puzzle is selected as described in Section 8.1.1.2. There is no requirement, that the algorithm for the puzzle in the IKE_SA INIT exchange be the same, as the algorithm for the puzzle in IKE_AUTH exchange, however it is expected that in most cases they will be the same.

8.2.2. Solving Puzzle and Returning the Solution

If the IKE_SA_INIT response message contains the PUZZLE notification and the initiator supports puzzles, it MUST solve the puzzle. Puzzle construction on the IKE_AUTH exchange differs from the puzzle in the IKE_SA_INIT exchange and is described in Section 8.2.2.1. Once the puzzle is solved the initiator sends the IKE_AUTH request message, containing the Puzzle Solution payload.

```
HDR, PS, SK {IDi, [CERT,] [CERTREQ,]
             [IDr,] AUTH, SA, TSi, TSr}   -->
```

The Puzzle Solution payload is placed outside the Encrypted payload, so that the responder would be able to verify the puzzle before calculating the Diffie-Hellman shared secret and the SK_* keys.

If IKE Fragmentation [RFC7383] is used in IKE_AUTH exchange, then the PS payload MUST be present only in the first IKE Fragment message, in accordance with the Section 2.5.3 of RFC7383. Note, that calculation of the puzzle in the IKE_AUTH exchange doesn't depend on the content of the IKE_AUTH message (see Section 8.2.2.1). Thus the responder has to solve the puzzle only once and the solution is valid for both unfragmented and fragmented IKE messages.

8.2.2.1. Computing Puzzle

The puzzle in the IKE_AUTH exchange is computed differently, than in the IKE_SA_INIT exchange (see Section 8.1.2.1). The general principle is the same, the difference is in constructing of the string S. Unlike the IKE_SA_INIT exchange, where S is the cookie, in the IKE_AUTH exchange S is a concatenation of Nr and SPIr. In other words, the task for IKE initiator is to find the key K for the agreed upon PRF such that the result of PRF(K,Nr | SPIr) has sufficient number of trailing zero bits. Nr is a nonce used by the responder in IKE_SA_INIT exchange, stripped of any headers. SPIr is IKE responder SPI in the SA being established.

8.2.3. Receiving Puzzle Solution

If the responder requested the initiator to solve puzzle in the IKE_AUTH exchange, then it SHOULD silently discard all the IKE_AUTH request messages without the Puzzle Solution payload.

Once the message containing solution for the puzzle is received the responder SHOULD verify the solution before performing computationly intensive operations - computing the Diffie-Hellman shared secret and the SK_* keys. The responder MUST silently discard the received message if the puzzle solution is not correct (has insufficient

number of trailing zero bits). If the puzzle is successfully verified and the SK_* key are calculated, but the message authenticity check fails, the responder SHOULD save the calculated keys in the IKE SA state while waiting for the retransmissions from the initiator. In this case the responder may skip verification of the puzzle solution and ignore the Puzzle Solution payload in the retransmitted messages.

If the initiator uses IKE Fragmentation, then it is possible, that due to packets loss and/or reordering the responder would receive non-first IKE Fragment messages before receiving the first one, containing the PS payload. In this case the responder MAY choose to keep the received fragments until the first fragment containing the solution to the puzzle is received. However in this case the responder SHOULD NOT try to verify authenticity of the kept fragments untill the first fragment with the PS payload is received and the solution to the puzzle is verified. After successful verification of the puzzle the responder would calculate the SK_* key and verify authenticity of the collected fragments.

9. DoS Protection after IKE SA is created

Once IKE SA is created there is usually no much traffic over it. In most cases this traffic consists of exchanges aimed to create additional Child SAs, rekey or delete them and check the liveness of the peer. With a typical setup and typical Child SA lifetimes there must be no more than a few such exchanges in a minute, often less. Some of these exchanges require relatively little resources (like liveness check), while others may be resource consuming (like creating or rekeying Child SA with Diffie-Hellman exchange).

Since any endpoint can initiate new exchange, there is a possibility that a peer would initiate too many exchanges, that could exhaust host resources. For example the peer can perform endless continuous Child SA rekeying or create overwhelming number of Child SAs with the same Traffic Selectors etc. Such behaviour may be caused by buggy implementation, misconfiguration or be intentional. The latter becomes more real threat if the peer uses NULL Authentication, described in [NULL-AUTH]. In this case the peer remains anonymous, that allow it to escape any resposibility for its actions.

The following recommendations for defense against possible DoS attacks after IKE SA is established are mostly intended for implementations that allow unauthenticated IKE sessions. However they may also be useful in other cases.

- If the IKEv2 window size is greater than one, then the peer could initiate multiple simultaneous exchanges, that would potentially

increase host resourse consumption. Since currently there is no way in IKEv2 to decrease window size once it was increased (see Section 2.3 of [RFC7296]), the window size cannot be dynamically adjusted depending on the load. For that reason if is NOT RECOMMENDED to ever increase IKEv2 window size above its default value of one if the peer uses NULL Authentication.

- If the peer initiates requests to rekey IKE SA or Child SA too often, implementations can respond to some of these requests with the TEMPORARY_FAILURE notification, indicating that the request should be retried after some period of time.

- If the peer creates too many Child SA with the same or overlapping Traffic Selectors, implementations can respond with the NO_ADDITIONAL_SAS notification.

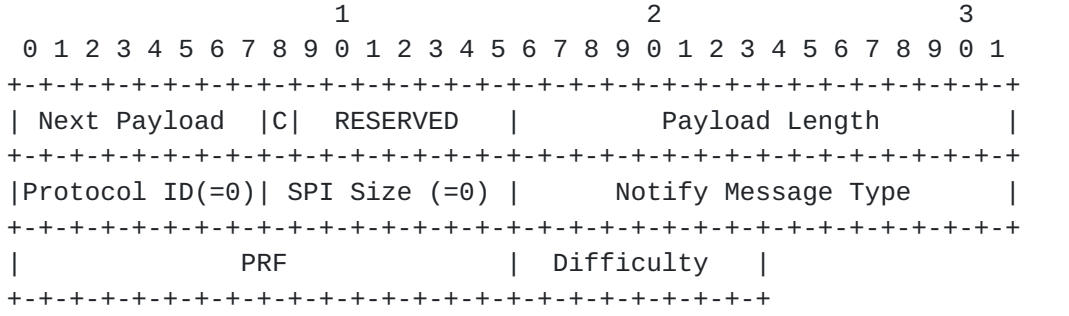
- If the peer initiates too many exchanges of any kind, implementations can introduce artificial delay before responding to request messages. This delay would decrease the rate the implementation need to process requests from any particular peer, making possible to process requests from the others. The delay should not be too long not to cause IKE SA to be deleted on the other end due to timeout. It is believed that a few seconds is enough. Note, that if the responder receives retransmissions of the request message during the delay period, the retransmitted messages should be silently discarded.

- If these counter-measures are inefficient, implementations can delete IKE SA with an offending peer by sending Delete Payload.

10. Payload Formats

10.1. PUZZLE Notification

The PUZZLE notification is used by IKE responder to inform the initiator about the necessity to solve the puzzle. It contains the difficulty level of the puzzle and the PRF the initiator should use.



- Protocol ID (1 octet) - MUST be 0.

- SPI Size (1 octet) - MUST be 0, meaning no Security Parameter Index (SPI) is present.

- Notify Message Type (2 octets) - MUST be <TBA by IANA>, the value assigned for the PUZZLE notification.

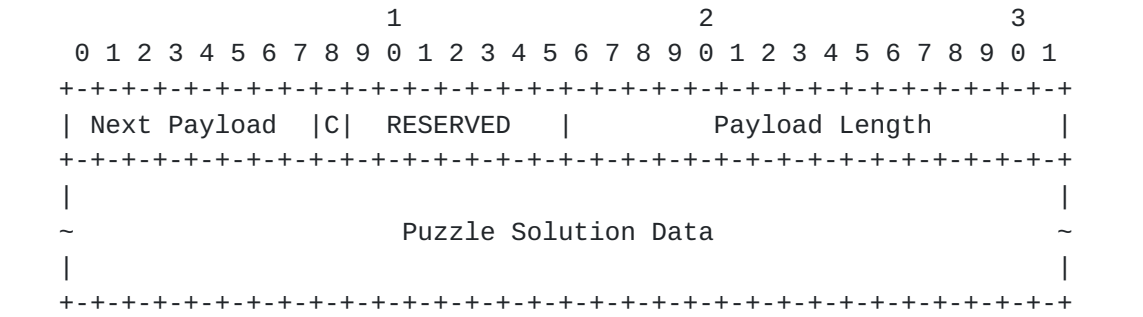
- PRF (2 octets) - Transform ID of the PRF algorithm that must be used to solve the puzzle. Readers should refer to the section "Transform Type 2 - Pseudo-random Function Transform IDs" in [IKEV2-IANA] for the list of possible values.

- Difficulty (1 octet) - Difficulty Level of the puzzle. Specifies minimum number of trailing zero bit, that the result of PRF must contain. Value 0 means that the responder doesn't request any specific difficulty level and the initiator is free to select appropriate difficulty level of its own (see Section 8.1.1.1 for details).

This notification contains no data.

10.2. Puzzle Solution Payload

The solution to the puzzle is returned back to the responder in a dedicated payload, called Puzzle Solution payload and denoted as PS in this document.



- Puzzle Solution Data (variable length) - Contains the solution to the puzzle - i.e. the key for the PRF. This field MUST NOT be empty. If the selected PRF has a fixed-size key, then the size of the Puzzle Solution Data MUST be equal to the size of the key. If the PRF agreed upon accepts keys of any size, then then the size of the Puzzle Solution Data MUST be between 1 octet and the preferred key length of the PRF (inclusive).

The payload type for the Puzzle Solution payload is <TBA by IANA>.

11. Security Considerations

To be added.

12. IANA Considerations

This document defines a new payload in the "IKEv2 Payload Types" registry:

<TBA>	Puzzle Solution	PS

This document also defines a new Notify Message Type in the "IKEv2 Notify Message Types - Status Types" registry:

<TBA>	PUZZLE

13. References

13.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, October 2014.

[RFC7383] Smyslov, V., "Internet Key Exchange Protocol Version 2 (IKEv2) Message Fragmentation", RFC 7383, November 2014.

[IKEV2-IANA] "Internet Key Exchange Version 2 (IKEv2) Parameters", <http://www.iana.org/assignments/ikev2-parameters>.

13.2. Informative References

[RFC5723] Sheffer, Y. and H. Tschofenig, "Internet Key Exchange Protocol Version 2 (IKEv2) Session Resumption", RFC 5723, January 2010.

[bitcoins] Nakamoto, S., "Bitcoin: A Peer-to-Peer Electronic Cash System", October 2008, <https://bitcoin.org/bitcoin.pdf>.

[ALG-AGILITY] Housley, R., "Guidelines for Cryptographic Algorithm Agility", draft-iab-crypto-alg-agility-05 (work in progress), December 2014.

[NULL-AUTH]
Smyslov, V. and P. Wouters, "The NULL Authentication Method in IKEv2 Protocol", draft-ietf-ipsecme-ikev2-null-auth-07 (work in progress), January 2015.

Authors' Addresses

Yoav Nir
Check Point Software Technologies Ltd.
5 Hasolelim st.
Tel Aviv 6789735
Israel

EMail: ynir.ietf@gmail.com

Valery Smyslov
ELVIS-PLUS
PO Box 81
Moscow (Zelenograd) 124460
Russian Federation

Phone: +7 495 276 0211
EMail: svan@elvis.ru