HPACK Security Considerations

Some Attacks Against Compression

CRIME-like

Memory exhaustion

Huffman considerations

CPU consumption

CRIME and CRIME-like attacks

Probe a compression context to verify hypothesis about a secret.

Examples:

Begins with ‘A’

Contains ‘B’

CRIME and CRIME-like attacks

Obtain data by examining the length of the compressed output:

Information is obtained when compression output changes length because the hypothesis interacts with secret within the compression context.

CRIME and CRIME-like attacks

Work when amount of data inferred is nonzero for hypotheses that don’t match the entire secret.

In other words, works when you can probe the compression context effectively.

CRIME and CRIME-like attacks

Don’t work when the entire secret must be matched against the hypothesis

You need a brute-force attack when you can’t effectively probe the compression context!

How does HPACK work?

For each key, value:

Does it completely match another key-value?

If so, backreference.

If not, add the new one to the state table

Optionally huffman-encode the new one

Pad to next byte boundary if necessary.

How does HPACK work?

Also includes delta-coding:

- Only differences from one set of headers to the next are sent.

HPACK thwarts CRIME

Since HPACK requires a full key+value match, no information is learned by probing the compression context until the entire secret is guessed in its entirety.

HPACK and memory usage

HPACK has a strict bound on the amount of memory consumed.

Overhead of keeping entries is included in memory limit.

HPACK Decoder Memory Consumed

Default size currently 4Kb.

HTTP2 has provisions for mechanisms to allow a receiver to safely request a decoder state-size change, both upwards and downwards.

HPACK Decoder Memory Consumed

Under memory pressure?

Request a zero compressor state size.

Kill connections which don’t respond quickly.

HPACK Decoder Memory Consumed

The decoding algorithm allows for all state to be within a fully contiguous memory region.

HPACK implementations can be robust against to heap-fragmentation attacks.

HPACK Encoder Memory Consumed

Max encoder memory consumption is always a function of the encoder's willingness to use memory.

An encoder may always use zero memory if it desires.

HPACK uses Huffman Coding

HPACK achieves a ~30% cumulative reduction in size by using huffman coding.

The huffman table is static, and does not react to changes in letter frequency.

Huffman encoding and security

Huffman reduces the number of bits on the wire.

Does this leak information?

Huffman: Does it leak information?

Probably not enough to matter, but should be watched!

Huffman: Does it leak information?

One can observe the number of bits sent, thus one knows that one must make 2^num_bits guesses.

If huffman typically reduces the size of output by 30%, then instead of 2^n_bits, it takes 2^(.7*n_bits) guesses.

Huffman: Does it leak information?

The static table is based on frequency analysis of real data.

… So is this a real reduction in the common case?

Huffman: Does it leak information?

If one knows the length of the plaintext and one observes the length of the huffman-coded output, the search state-space can be reduced.

e.g. A one-letter long secret encoded in one byte is one of the codes requiring 8 bits or less.

Huffman: Does it leak information?

This attack becomes less useful in general as the size of the secret increases.

This attack is also somewhat less useful when the size of the secret is padded to the next byte boundary, as occurs in HPACK.

CPU consumption

A malicious actor can attack a compressor by sending an unending bytestream which adds/removes elements from the working set.

Since it isn’t possible to distinguish between malicious and stupid, heuristics are required to protect against these kinds of attacks.

CPU consumption

In cases where the compressor is used with data which is unlikely to be detected as malicious (e.g. slowloris), HPACK uses ~1/3rd the CPU or less of gzip.

Thanks!

fenix@google.com