Individual submission
Internet-Draft
Intended status: Informational
Expires: May 1, 2020

W. Handte
Facebook, Inc.

October 29, 2019

Security Considerations Regarding Compression Dictionaries
draft-handte-httpbis-dict-sec-00

Abstract

Dictionary-based compression enables better performance, but brings state into the process of compression, with all the complexities that follow. This document explores the security implications of this technique in the context of internet protocols and enumerates known risks and mitigations.

Table of Contents

1. Introduction

General-purpose data compression algorithms are designed to achieve good performance on many different kinds of data. However, that general-purpose nature makes them, to a certain extent, jacks of all trades and masters of none: a compressor that has been tuned for a specific use case can always perform better than a generic equivalent.

In response, a number of modern compression algorithms (including DEFLATE [DEFLATE], Brotli [BROTLI], and Zstandard [ZSTD]) have developed a generic capability to specialize themselves. In addition to the actual message to be processed, these compressors allow users to provide additional context information, which the compressor and decompressor can use to tailor their internal states to that particular use case. To the extent that this auxiliary data matches the nature of the message being compressed, the compressor can use it to produce a smaller compressed representation of the message. This auxiliary data can include various things, but it has come to be known as a "dictionary."

As dictionary-based compression has been adopted, it has been found that its use can present security challenges. This document is a collection of those challenges. As future use cases for dictionaries are contemplated, this document can be used as a checklist to ensure that the protocols, their specifications, and their implementations have been appropriately evaluated against these concerns.

2. Basis

2.1. Compression Environments

The security of any use of compression depends greatly on the environment in which it is deployed, and the threats it is subjected to. This document analyzes dictionary-based compression as it might be used by a generic internet protocol, in which:

- Agents exchange messages over possibly-trusted, possibly-authenticated, possibly-encrypted channels, which are vulnerable to some combination of traffic analysis, eavesdropping, and manipulation.

- Agents exchange messages with parties they may not trust.

- Agents may take protocol actions (generating, sending, receiving, and interpreting messages) in response to triggers other than user action. Some examples include:

  * Replying automatically to received messages.

  * Relaying or forwarding received messages to other agents (e.g., an SMTP relay).

  * Exchanging messages at the behest of trusted or untrusted code (e.g., trusted: a website codebase generating responses to HTTP requests, untrusted: a website's JavaScript code running in a browser.

This document aims to enumerate all security risks raised when using dictionary-based compression in this baseline environment. In addition to attempting an exhaustive list of possible security risks, this document will identify desirable properties of the protocol stack and environment in which the compression is used and other methods with which individual concerns can be obviated or mitigated.

2.2. Security Properties

[TODO]

2.3. Threat Model

[TODO]

2.4. Existing Attacks

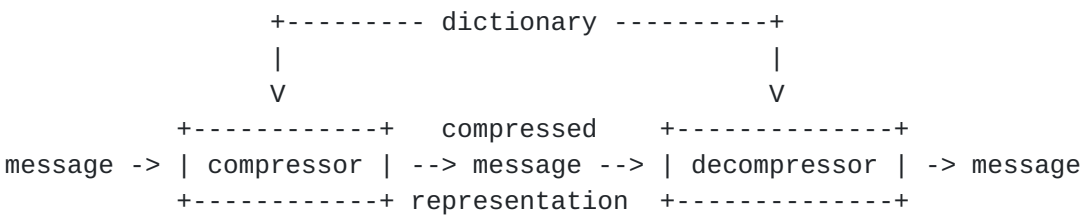
This document excludes from its analysis security risks that are already present without the use of dictionary compression.

In particular, compression as it broadly used today--without dictionaries--is known to introduce vulnerabilities. The most well known series of these attacks ([CRIME] et al.) recovers message content of inaccessible or encrypted traffic by observing message sizes while manipulating other parts of the message or traffic stream.

3. Dictionaries

3.1. Dictionary Compression

Classically, compression algorithms operate as stateless, pure functions. In that mode, their output depends solely on the input message and the algorithm's implementation details. Dictionaries break that paradigm, introducing an additional input to the compression and decompression operations. Compressors may then leverage the contents of that additional input--the dictionary--to produce more compact representations of their inputs.



In introducing this other element, the interpretation of the compressed message becomes dependent on the content of the dictionary, and therefore same dictionary that was used to compress a message must be presented at decompression time. In this way, dictionaries are in effect an out-of-band communication or pre-shared key between the compressor and decompressor.

3.2. Dictionary Contents

In principle, the contents of a dictionary are solely the concern of the compressor and decompressor, and implementations should be free to treat them as opaque blobs. However, when analyzing their security characteristics, it's useful to understand the data that is actually present in a dictionaries.

Dictionaries take two broad forms.

3.2.1. Unstructured Dictionaries

Some compressors (e.g., DEFLATE [DEFLATE] and Zstandard [ZSTD]) accept arbitrary, unstructured bytestreams as dictionaries. In these cases, the dictionary is used purely as a buffer in which LZ77-style content matches can be found [LZ77]. That is, when the dictionary contains some sequence of bytes that is also present in the message, the compressor can choose to represent those bytes by referencing them in the dictionary, rather than by representing them literally.

3.2.2. Structured Dictionaries

Some compressors (e.g., Brotli [BROTLI] and Zstandard [ZSTD]) accept dictionaries that conform to a specific and defined format. In these cases, the dictionary data can consist of multiple components, each of which is used in different ways.

metadata: The dictionary may contain metadata that identifies the dictionary. For example, Zstandard dictionaries include a 32-bit integer ID field.

statistics: The dictionary may contain frequency distributions of various kinds of symbols, which the compressor can use to more efficiently encode the corresponding streams instead of using a default frequency distribution.

initial values: For example, Zstandard allows the dictionary to initialize certain parts of the compressor's internal state (in particular, the initial values of Repeated_Offset1, Repeated_Offset2, and Repeated_Offset3) [ZSTD].

instructions: The dictionary may describe preprocessing or transformation steps to be taken on the input. [TODO: expand]

corpus content:

untokenized: For LZ77-style compressors [LZ77], the structured dictionaries may still contain unstructured content for the compressor to make matches against.

tokenized: Alternatively, for LZ78-style compressors [LZ78], the match content is tokenized (i.e., it consists of a collection of independent strings, serialized in some form).

3.3. Using Dictionaries

In order to use compression dictionaries in a system, it is not only the internals and integration points of the compressor and decompressor whose behavior must change. Dictionaries make compression stateful, and applications that use dictionaries must therefore participate in the whole lifecycle of state management.

3.3.1. Generating Dictionaries

As noted in Section 3.2.1, some compression algorithms can accept arbitrary, unstructured inputs as dictionaries. These unstructured dictionaries do not require an explicit generation step; users can simply repurpose existing messages as dictionaries. This potentially

avoids the need to perform additional coordination and communication to distribute purpose-built dictionaries. See for example the Compression Dictionaries for HTTP/2 proposal [I-D.vkrasnov-h2-compression-dictionaries].

Alternatively, the dictionary may be a separate object, purpose-built for the task. Generating such a dictionary may be desirable for a number of reasons, including:

- Building a dictionary is necessary to produce the structure in a structured dictionary.
- Trained dictionaries generally perform better than using raw content. The training process selects the parts of the sample corpus that are useful for compression and discard the parts that are not, producing a more compact and more effective dictionary.
- The training process is an opportunity to sanitize the content that ends up being used as a dictionary, potentially enhancing security and privacy (see Section 4.8).

In general, an algorithm is run over a corpus of sample messages (such as the COVER algorithm [COVER] in Zstandard), which selects commonly occurring substrings and bundles them together.

Any structured metadata (e.g., symbol distribution statistics) can then be calculated. For example, Zstandard then compresses some of the sample messages it was given with the dictionary and aggregates the statistics resulting from those compressions and writes them into the dictionary's header.

3.3.2. Identifying Dictionaries

If freedom exists in a system as to which dictionary is to be used for a given message, there must be some way to distinguish which dictionary to use, so that decompressors can use the same one. In practice, this means associating each dictionary with an identifier.

Popular methods to do this include:

Identity ID: The "identifier" for the dictionary is the dictionary itself. This is not really very popular, since information theory strongly suggests that a compressed message without a dictionary will always be smaller than a message compressed with a dictionary plus the dictionary.

Arbitrary IDs: The scheme associates an arbitrary identifier (e.g., a number or string) with this dictionary. This can have the

advantage of being the most compact, but has the disadvantage that it neither describes the content of the dictionary nor where to get it.

Content-Derived IDs: Identifiers that are deterministically derived from the content they identify (such as hashes), when designed well, have the benefit that they can validate the associated dictionary without requiring trusting the dictionary source. (Though they are of course vulnerable to collision attacks.) They have the disadvantage that they do not describe where to source the dictionary. In order to be secure, they may also have to be relatively verbose.

Location-Based IDs: Identifiers of this form (notably, URLs) do not identify the content directly, but rather describe where to get it. They are suitable insofar as that source can be trusted to reliably serve the same content to different participants.

3.3.2.1. Existing Systems

Existing compression schemes have selected the following identification systems:

DEFLATE: DEFLATE writes an Adler32 checksum of the dictionary into its compressed message header and checks it at decompression-time.

Brotli: Brotli always implicitly uses a single static dictionary. As such, no identifier is needed or provided [BROTLI].

Shared Brotli: Shared Brotli uses either a 256-bit Highwayhash digest of the dictionary or a direct pointer to the dictionary when it is included in the same compressed stream [I-D.vandevenne-shared-brotli-format].

Zstandard: Zstandard uses 32-bit integers to identify dictionaries [ZSTD].

SDCH: SDCH uses a URL to describe how to fetch a dictionary and then a hash (a 96-bit prefix of the SHA-256 digest of the dictionary) in negotiations [I-D.lee-sdch-spec].

CDH2: Compression Dictionaries for HTTP/2 uses an 8-bit integer [I-D.vkrasnov-h2-compression-dictionaries].

3.3.3. Distributing Dictionaries

Dictionaries must themselves be made accessible to participants. There are several possible approaches to doing this:

static: The protocol defines the set of dictionaries. Protocol implementations can statically include or independently generate these dictionaries. No further distribution mechanism is required.

local: When dictionaries are not specified by the protocol, but are derived locally or provided by the user, no dictionary distribution mechanism is required, although a negotiation mechanism might be.

centralized: The set of dictionaries in use by the system changes over time, coordinated by and available from a central authority.

distributed: The set of dictionaries in use by the system changes over time. Some or all participants can generate and publish dictionaries.

3.3.4. Selecting Dictionaries

Related to the above, because the same dictionary must be used to compress and decompress a particular message, it is necessary for the compressor and decompressor to come to an understanding as to which dictionary they will use for a given message, presumably based on selecting which dictionary of those available to both the sender and receiver is most suitable. This selection process can take multiple forms:

implicit: In situations where the compressor or protocol specifies a single dictionary that is always used (e.g., Brotli [BROTLI]), no particular selection process is required. Use of the compression scheme at all (which may or may not itself be negotiated) is sufficient to identify the dictionary to use.

unilateral: When the set of dictionaries available to the decompressing agent is known to the compressing agent, the compressing agent may unilaterally select a dictionary to use, and include an identification of that dictionary in either the compressed message itself (e.g., Zstandard's Dictionary_ID field in the frame header) or in protocol metadata (e.g., an HTTP response header). This mechanism can be applied in simple situations, such as when the set of dictionaries used by the protocol is fixed and guaranteed to be immediately available to all participants (such as by being included in the

implementation's installation). It can also be applied to a more loose definition of availability, if the decompressing agent is known to be capable of retrieving the dictionary based on the provided identifier, even if it doesn't have the dictionary at present.

bilateral: When the set of dictionaries available to each party is not known to the other, additional messages may be required in order for the compressing agent to select a dictionary available to both both parties. In particular, while other negotiation patterns only require a flow of information from the compressor to the decompressor, which matches the flow of the compressed message itself, this mechanism requires communication in both directions.

3.3.5. Using Dictionaries

Having selected and retrieved a dictionary, it remains to actually present the dictionary to the compressor or decompressor and perform the compression operation.

Dictionaries, whether structured or not, are flat byte streams. In order to be used (especially in compression), most implementations require that a preparation step be performed on the content of the dictionary, populating the compressor's internal datastructures.

This materialization of the dictionary can sometimes be performed transparently as part of the compression or decompression operation. Alternatively, some compressors allow this materialization step to be performed separately / explicitly. When this capability is used, the work of processing the serialized dictionary into the compressor's internal datastructures only needs to be performed once, even when this materialized dictionary object is used for many compressions or decompressions. This can lead to significant efficiencies.

3.3.6. Deleting Dictionaries

Dictionary compression inherently entangles the lifetimes of different pieces of data. When a dictionary is generated, it collects and incorporates information about the data it was trained on (whether that be diffuse statistical information, small common substrings or tokens, or significant contiguous excerpts of the training data). When that dictionary is used to compress a set of messages, it must be retained by the system for as long as the system desires to be able to decompress any of those messages. The lifetime of information derived from individual messages is thus tied to the lifetime of many messages, or even the whole system. This introduces complexities for systems that wish to minimize or bound the lifetime of individual pieces of data.

4. Risks

These subsections each describe a class of security issues that have been raised concerning dictionary-based compression and the surrounding protocol mechanisms. Where possible and known, mitigations are described.

4.1. Revealing Message Content

This section discusses attacks that use dictionary compression to recover content in the message.

4.1.1. By Observing Which Dictionary is Used

Because dictionaries' effectiveness improves the more that they target a specific type of data, a protocol may want to use multiple dictionaries, each targeting a subclass of the system's traffic. Alternatively, a participant may always avoid using a dictionary in certain scenarios, such as when reporting an error. When this is the case, the use of a particular dictionary or not for a message implies that the message belongs to the corresponding subclass of traffic.

The metadata identifying which dictionary was used to compress a message should therefore be protected to the same extent that the message content is protected. (Similarly, the choice of dictionary and any data exchanged in that selection process may reveal other information about the sender and receiver, independent of the content of the specific message being handled, which is discussed in Section 4.5.1.)

This information may itself be inferred from other signals, and therefore serve as a stepping stone connecting those signals to conclusions about message content.

Message Size: Observations of message sizes, especially headers or connection negotiations (also discussed in Section 4.1.2), can indicate whether a dictionary was used, or even perhaps which dictionary was used.

Timing: Compression with and without a dictionary may take observably different amounts of time. This is also discussed in Section 4.1.3.

Dictionary Retrieval: When dictionaries are retrieved dynamically, another vector for learning this information is simply observing whether a message triggers a fetch for a dictionary, and if so, which dictionary. (This is also discussed in Section 4.5.2.) Protocols should consider decoupling retrieving dictionaries

(especially when doing so is easily observable) from using them. For example, SDCH advertises and retrieves dictionaries independently of using them [I-D.lee-sdch-spec].

4.1.2. By Observing Message Size

By manipulating a portion of the message and observing the overall size of the compressed message, the attacker can recover information about the portions of the message not under its control [BREACH] [CRIME] [HEIST]. Given that dictionary-based compression is an extension of dictionary-less compression, it is certainly also vulnerable to this attack.

In particular, the dictionary itself can be used in this sort of attack, to the extent that its contents are attacker-controlled. Note that the ability to control which dictionary is used may indirectly give an attacker the effective ability to modify the contents of the dictionary.

Protocol designers should therefore prevent parties that will not have access to the message content from being able to influence the dictionary used to compress the message.

In settings where the dictionary that is used is derived from previous traffic, especially if previous traffic is directly used as a dictionary, the problem of ensuring that private data and attacker-controlled data grows in complexity. In such a scheme, the attacker may also be able to exercise more control over the content of the dictionary if they can influence the order in which messages are exchanged. Protocols of this sort may wish to place strong controls on the kinds of messages that can be included in the dictionary. See for example [I-D.vkrasnov-h2-compression-dictionaries].

The remaining question is whether the dictionary constitutes a third class of data (fixed, known data), with distinct security properties. That is, even if the dictionary is neither under attacker control nor does it contain private information, can its use still reveal information about the contents of the message under compression.

4.1.2.1. Mitigating with Padding

One possible mitigation of the compressed message size oracle is to add padding to messages, either at the compression level or at the transport layer (e.g., [I-D.pironti-tls-length-hiding]). Even simple padding schemes can significantly inflate the cost of mounting such an attack, if not mitigate it completely.

4.1.2.2. Mitigating by Separating Content

Another possible strategy to mitigate this attack is to avoid letting attacker-controlled data be matched against private data. This can be accomplished by avoiding compressing one or the other, or by compressing them independently of each other. See, e.g., [CLOUDFLARE-NO-COMPRESS].

4.1.2.3. Mitigating by Avoiding Repeated Compressions

A crucial feature of these attacks is that they require the message under attack to be re-compressed many times (proportional to the amount of information being extracted). The attack can therefore be mitigated either by limiting the number of times the same message can be compressed (rate-limiting), or by making sure that it is not the same message that is compressed every time.

That is to say, these attacks are most effective when the attacker-controlled data is the only thing that is changing between compressions. Changing or randomizing content (ideally, including the secrets in question) in the message on each compression can make it much harder to extract information.

4.1.3. By Observing Timing

Timing is another classic side-channel through which information can leak. An attacker could potentially observe the time taken during compression or decompression, and draw conclusions about the contents of a message. As discussed in Section 4.7.1.3.1, it is possible that a dictionary could affect the efficiency of compression and decompression.

In addition, timing can act as a vector for extracting information from another side-channel. As described in the HEIST attack [HEIST], compression ratio information can be leaked by counting round-trip latencies.

Alternatively, while compression and decompression are usually relatively fast and fairly content-insensitive operations, retrieving and initializing a dictionary might be a high-latency operation, and therefore may be identifiable by observing timing. Timing is therefore another potential avenue to observe which dictionary is used, which may in turn reveal information about the message being processed (Section 4.1.1).

4.2. Revealing Dictionary Content

This section investigates the ability to leverage dictionary-based compression to reveal data other than the message content being compressed (i.e., revealing content used as the dictionary). Note that this is only of interest when there are secrets in the dictionary, which violates the common model that is mostly analyzed in this document, in which the dictionary is assumed to be a shared, public resource.

In systems with multiple privacy domains, the ability to nominate arbitrary resources in that system as dictionaries poses a risk.

Protocol designers and implementors should ensure that compressing and decompressing agents cannot use as dictionaries resources from privacy domains that either agent does not have access to.

A corollary is that a transport system that mixes resources from multiple privacy domains into the same compression context through dictionary-based compression should not reveal the compressed representation of messages (or information derived from the compressed representation, such as its size) to other components of the system that are only trusted in a particular privacy domain.

4.2.1. By Observing Message Size

Analogously to Section 4.1.2, an attacker can exploit knowledge about the contents of a message and its compressed size to draw conclusions about the contents of the dictionary.

4.2.2. In Compression

If an attacker can inspect the compressed representation of a message, they may be able to draw conclusions about the contents of the dictionary that was used to compress it. This is especially the case if the attacker knows the original message that was compressed (i.e., a known-plaintext attack) or if the attacker can supply the message to be compressed (i.e., a chosen-plaintext attack), and is helped if the attacker can cause the message to be compressed multiple times while varying some aspect of the compression.

4.2.3. In Decompression

In compression schemes that support the use of dictionaries, and especially unstructured dictionaries, it is possible to craft compressed messages independent of a dictionary in such a way that, when decompressed with a provided dictionary, the decompressed message that is produced will reveal information about the contents

of the dictionary that was not known by the compressor (possibly trivially, by directly reproducing some or all of the dictionary's contents).

Consider a protocol that allows a compressing agent to freely identify any other resource in the system as the dictionary for a message. The compressing agent could select as a dictionary some resource to which the decompressing agent has access, but to which it does not. Without access to that resource, it could nonetheless generate a compressed message the effect of which would be to reproduce that resource in part or in its entirety. This message, decompressed by the target, would cause a resource in the compressing agent's trust domain to appear to have the contents of a resource it does not itself have access to.

This could cause the decompressing agent to take some action that the compressing agent would not otherwise have had the authority to initiate. Alternatively, with some additional mechanism, the compressing agent could then cause the decompressing agent to reveal the uncompressed message (i.e., the selected third-party resource) back to the compressing agent.

4.3. Manipulating Message Content

When the decompressing agent uses a different dictionary to decompress a message than was used to compress the message (which is possible due to confusion on the part of either the compressing or decompressing agent), the reconstituted message produced by decompression may differ from the original message the compressing agent intended.

An attacker that can induce this situation can therefore use dictionary compression to manipulate the perceived content of messages, even when they cannot directly manipulate the contents of the messages themselves.

A particular implication of this is that a compressed message may have multiple interpretations. In one context (with one dictionary), the message can be constructed so as to appear benign or to pass a validation or authentication step when decompressed. Later, if a different component or agent can be induced to decompress the same message with a different dictionary, the reconstructed message may be completely different.

A general mitigation against this attack is to specify mechanisms to validate the integrity of the message. In particular, it may be desirable to validate the ultimate, uncompressed message, rather than validating the various components that the decompressing agent relies

on to reconstitute the uncompressed message--the compressed message, the metadata identifying the dictionary, the associated dictionary contents, etc. (However, this has its own problems [ENCRYPT-THEN-AUTHENTICATE].)

4.3.1. By Manipulating Message Content

The degenerate version of this attack is to manipulate the uncompressed message by directly manipulating the compressed representation of the message. In such a scenario, the presence or absence of a dictionary is irrelevant. In most cases, this attack is defended against by some scheme that protects the integrity of the compressed message.

However, it is useful to point this attack out, as the other attacks in this space aim to achieve the same result indirectly, and may do so by exploiting protocols which protect the integrity of the compressed message, but perhaps not its metadata describing which dictionary to use nor the contents of that dictionary, such as might arise particularly if dictionary-based compression is an extension to an existing protocol.

4.3.2. By Manipulating Dictionary Content

One possible avenue for this kind of attack is to cause the compressing agent and decompressing agent to have differing views of the same dictionary (whether by manipulating a participant's local copy or by causing a fetch to return different results for different users or otherwise).

Protocol designers should therefore take care to protect the integrity of dictionaries. Two broad strategies exist to do so.

4.3.2.1. Mitigating by Validating Dictionary Contents

In the first, the identifier for the dictionary may itself be used to validate the contents that are retrieved, if the identifier scheme includes a cryptographically secure digest of the identified dictionary's contents (see Section 3.3.2). Alternatively, even if the identifier itself does not provide for , designers should specify other mechanisms to ensure the integrity and correctness of dictionaries (signatures, checksums, etc.). See for example schemes like Subresource Integrity [SRI].

4.3.2.2. Mitigating by Validating Dictionary Sources

Alternatively, participants can rely on a secure chain of custody from a trusted source. ... [TODO]

In practice, it is probably advisable to implement both mitigations in some form.

4.3.3. By Manipulating Dictionary Identifiers

Another similar attack is to cause the different agents to have differing views of which dictionary to use. That is, even if the integrities of compressed messages and dictionary contents are protected, if the association between one and the other can be manipulated, the same effect can be achieved.

4.4. Obfuscating Message Content

This section discusses attacks that obfuscate a malicious response's content through the use of dictionary-based compression.

4.4.1. From Intermediaries

Various internet protocols exchange messages through intermediaries which inspect or modify the traffic as it passes by (proxies, caches, firewalls, etc.), sometimes for reasons that include security. If the compressing and decompressing agents on a connection use a dictionary to compress the messages they exchange, and the intermediaries between them are not themselves capable of processing messages compressed this way, the intermediaries may be prevented from being able to inspect the traffic, which may harm their ability to detect and filter malicious traffic.

In practice, the relevance of this concern is questionable. Intermediaries of this form [PERVASIVE-MONITORING] can be more harmful than they are beneficial to the security of participants and their traffic. Many protocols are moving towards end-to-end encrypted models that preclude intermediaries from interacting with messages in this way.

Nonetheless, designers of protocols that involve intermediaries that might not support dictionary based compression should give those intermediaries the ability to downgrade the message exchange to not use dictionaries. Intermediaries which inspect messages in the course of their business should either implement the dictionary based compression scheme in question or downgrade the message exchange to avoid its use.

4.4.2. Multiple Representations

Although the majority (if not the entirety) of compression schemes do not guarantee determinism in compression, many implementations are deterministic in practice (under fixed parameters). Experience has demonstrated that this state of affairs sometimes entices implementors into confusing equality-of-message comparison with equality-of-representation comparison. Representing the same message in a new way can therefore violate assumptions and potentially be used as a vector for exploitation. Dictionaries potentially contribute to this issue, by introducing a new vector for non-determinacy in the compressed representation of a message.

Users of compression should therefore avoid assumptions that a message will always be transformed into the same compressed representation.

4.5. Tracking Users

This section discusses attacks that identify users through their negotiation and use of dictionaries.

Like any other protocol extension or option, the use or advertisement of dictionaries, may allow observers to distinguish participants that do and do not support the feature.

4.5.1. Through Dictionary Negotiation

In systems which distribute dictionaries dynamically, a participant or observer may be able to learn about the past actions of other participants by observing the dictionaries they advertise or select.

For example, if a user exchanged messages with some site (www.mybank.com), and in doing so acquired dictionaries published by that operator, and then sometime later negotiated a connection with some other site (www.curiousaboutyou.com), in which the user advertised the dictionaries in their possession, the second operator could reasonably conclude that the user had a bank account at MyBank.

Designers of protocols that use dynamically distributed and negotiated dictionaries should therefore take care that dictionaries distributed in one privacy domain are not advertised or used in others without reason.

4.5.2. Through Dictionary Retrieval

The distributor of a dictionary may also be able track the propagation of traffic amongst participants as it receives requests for a particular dictionary, especially if it can collude with the party that generated that message to use a unique dictionary identifier.

Dictionaries that are dynamically fetched should therefore be fetched from the same privacy domain they are used in.

4.6. Denial of Service

Because dictionary-based compression introduces additional dependencies to the processes of generating and interpreting messages, an attacker that cause those dependencies to be unavailable can potentially cause participants to fail to process messages.

Protocols that use dictionary-based compression, especially when the dictionaries are retrieved in ways that could fail, should be prepared to gracefully degrade when those fetches fail. Designers may consider whether messages should only be compressed with dictionaries known to already be in the possession of the recipients.

4.7. Resource Exhaustion

This section discusses attacks that use dictionaries and dictionary-based compression to induce failures through the exhaustion of various resources.

Aside from more specific concerns and corresponding protections discussed in the following sections, implementors should take care to apply at least the same resource usage constraints to dictionaries that they do to the other traffic they handle. Stronger constraints may be warranted, in fact, since the goal of dictionaries is to lower total resource consumption.

4.7.1. Resources

4.7.1.1. Bandwidth

Attacks of this form cause the target to consume their network resources, resulting in expense and degradation of service.

4.7.1.1.1. Messages

If dictionaries can be used to make the compressed representation of messages artificially large, it may be possible to cause normal traffic to consume disproportionately large bandwidth. With existing dictionary schemes, this is unlikely.

The reverse is also potentially dangerous, though. Systems that are accustomed to using dictionary-based compression (and whose resources are allocated according to the efficiencies achieved thereby) may be vulnerable to resource exhaustion when subjected to downgrade attacks. If an attacker can force the system to fall back to not using dictionaries, or to using bad dictionaries, or to not using compression at all, the system may exceed its allocated network resources.

4.7.1.1.2. Dictionaries

In protocols in which dictionaries are distributed dynamically, it may be possible to cause a target to repeatedly attempt to fetch dictionaries, whether by causing dictionary fetches to fail, triggering retries, or by causing the target to use many new dictionaries that it must then load.

Since dictionaries can be quite large relative to the messages they are used to compress, this could potentially be an effective amplification attack.

4.7.1.2. Storage

Attacks of this form target the storage resources of a participant (any of main memory, cache, disk space, etc.).

4.7.1.2.1. Message Size

The same concerns apply here as in Section 4.7.1.1.1.

Additionally, if dictionaries can be used to make the compressed representation of a message extremely small relative to the its uncompressed size, they may play a role in enabling a "zip bomb" type attack, in which a specially crafted, small (and therefore cheap to send) message causes the recipient to consume a huge amount of storage space after decompression.

Implementors should therefore apply storage quotas to messages based on the size of the representation in which they will actually be stored. Implementors may also wish to consider rejecting messages

whose compressed representation is significantly larger than the message represented.

4.7.1.2.2. Message Duplication

Obviously, flooding a target with messages is an easy way exhaust that participant's resources. Using a dictionary does not natively affect that brute force strategy. However, simple mitigations to this sort of attack sometimes leave chinks in systems' armor, which dictionaries might play a role in exploiting.

For example, if an attacker can cause a participant to receive and store a single logical message more than once, with different metadata (such as the dictionary used) or with a different compressed representation (as a result of using a different dictionary), the participant may not be able or willing to deduplicate the message. For example, an HTTP Cache may be forced to store the same resource multiple times, compressed with different dictionaries, if the choice of dictionary is part of the cache's secondary key [HTTP-CACHING].

4.7.1.2.3. Dictionaries

Another possible avenue of attack would be to cause a participant to consume space by storing the dictionaries themselves. The effectiveness of attacks of this form are driven by the product of (1) the number of dictionaries stored, (2) their size, and (3) how long they are retained.

Dictionaries may themselves be fairly large. But one thing to note in particular is that, when in use, the space consumed by a dictionary may be significantly greater than its raw size. In order to be used in compression or decompression (but particularly in compression), the dictionary contents must be loaded into the compressor's internal datastructures. This can be done at compression-time, for every compression, using the datastructures already allocated for that compression.

Alternatively, some compression algorithms allow the user to do this preparation step separately, producing a materialized representation of the dictionary in memory that can be reused across a number of compression operations (e.g., a ZSTD_CDict). While this avoids duplicated work (processing the dictionary for each compression), applications which cache these materialized dictionaries can accidentally consume a lot of memory. In addition to the factors mentioned above that control the total size of stored dictionaries, the expansion factor as those dictionaries are materialized is controlled by the compression settings (and potentially instructions in the dictionary).

Applications that allow other participants to influence the contents, number, size, retention period, or compression settings of dictionaries should take care to constrain the total at rest and in-memory footprints of those dictionaries.

4.7.1.3. Computation

Attacks of this form target the computational resources (and by extension, the time and energy) of a participant in the protocol.

4.7.1.3.1. Using a Dictionary

For existing compressors that support dictionaries, compression and decompression with a dictionary is usually faster than without one.

However, as the kinds of information captured in dictionaries grows, as described in Section 3.2.2, dictionaries may come to include instructions that significantly influence the speed of the compressor. For example, dictionaries might specify a particularly laborious transformation to be performed on the input. Or they might specify internal compression parameters, which might instruct the compressor to do huge amounts of work during compression.

If dictionary-based compressions systems evolve to include these sorts of features, care should be taken to avoid allowing dictionaries from untrusted sources to influence compression behavior or parameters. Note: this is not a concern for existing dictionaries.

Analogously, care should be taken to avoid allowing dictionaries to influence decompression performance.

4.7.1.3.2. Generating Dictionaries

Training a dictionary, depending on the methodology, can be a very expensive computation (building an optimal dictionary is NP-hard). Designers of protocols that involve creating new dictionaries on the fly should constrain either or both of (1) who can cause a participant to train a new dictionary and (2) the computational cost of training a new dictionary (by selecting a fast algorithm or by limiting the amount of data over which the algorithm is run).

4.7.2. Targets

In addition to the immediate compressing and decompressing agents, the mechanisms surrounding dictionary-based compression may allow for the targeting of other agents.

4.7.2.1. An Intermediary

Insofar as intermediaries in internet protocols are often responsible for handling a much higher volume of traffic in a much lighter-weight way than protocol endpoints, any additional per-message or per-connection burden has the potential to significantly increase the workload of the intermediary. Retrieving, caching, and processing dictionaries, especially when the set of dictionaries is unbounded, is potentially untenable for intermediaries of that type.

4.7.2.2. A Third Party

The mechanisms surrounding dictionary-based compression potentially also enable attacks against third parties, including parties with whom the attacker cannot exchange messages directly.

If a recipient can be induced to relay messages to a third-party, or to generate new messages directed at a third-party, a third party can become the effective recipient of dictionary-compressed traffic. If the dictionaries used to compress these messages are hard or slow to load (or even non-existent), the work of handling these messages could be significant. This is especially a risk when decompression of the message is required before it can be evaluated against an access-control policy or otherwise distinguished from legitimate traffic.

Protocol designers should therefore consider carefully the risks of using dictionary-based compression on (the parts of) messages that are used for authentication.

Another possible attack, when dictionaries are distributed dynamically, arises from the ability for compressed messages to trigger the retrieval of a dictionary from a third party. This is especially a risk when the source for a dictionary can be arbitrarily specified (as, for example, a URL).

These approaches potentially allow an attacker to amplify their efforts and turn their attack into a distributed one.

Protocol designers should consider how the source for the retrieval of a dictionary is derived, who can influence that derivation, and whether it should be constrained to preclude nominating a third party.

Protocol designers and implementors who relay messages should also consider whether those messages should be relayed compressed with the same dictionary, or whether dictionary selection and negotiation should occur for each hop in the path of a message.

4.8. Generating Dictionaries

This section discusses the potential for inadvertent leakage of private information in the creation of dictionaries.

As described in Section 3.3.1, dictionaries are commonly generated by an algorithm run over a corpus sampled from the application's traffic. For systems which wish to publish dictionaries publicly (or, at any rate, with less strict access controls than the traffic on which they are trained), it is important to prevent the leakage of private information in the creation of dictionaries.

The output of this training process, the dictionary, as described in Section 3.2, may be composed of several different kinds of data. Some of these pieces, like statistical summaries around symbol frequencies, are unlikely to represent vectors for leaking useful information about the corpus they were trained on. Other components, however, directly represent substrings found in the input corpus.

Protocol designers, implementors, and participants that construct their own dictionaries should take care to do so in a way that does not reproduce private data in the produced dictionaries' contents.

4.8.1. Handling Samples

Since dictionaries are generally produced from a collection of sample data, implementing a dictionary training capability may require storing or otherwise handling message traffic in ways it would otherwise not. This in itself can create an attack surface, for example if secrets that would normally exist only in transit or in memory are persisted or passed to other systems.

Care should be taken by implementors to protect the security of messages that are selected as samples for future use in dictionary training. Protections should be implemented both at rest and in transit, including retention limits, so as to limit the window of compromise.

4.8.2. Tagging Mitigations

One strategy for ensuring that private data does not appear in dictionaries is to avoid presenting private data to the training algorithm at all. This sanitization of the training samples can be accomplished either by removing just the specific parts of samples that are private or by entirely removing samples that contain any private data in them.

This discrimination of private and public content can rely on being able to identify private information on sight (e.g., [CLOUDFLARE-NO-COMPRESS]).

Alternatively, the trainer can rely on explicit signals, provided alongside the messages, to perform that discrimination.

4.8.3. Probabilistic Mitigations

Another strategy relies on a statistical approach for the identification and removal of private information.

In building the dictionary's contents, the goal of the dictionary training algorithm is to collect the set of strings that most effectively improve the compression ratio of messages in the corpus. This goal is best served by including strings that appear frequently in the sample corpus and rejecting strings that appear rarely.

In a loose way, it is reasonable to expect that commonly occurring substrings are less private, and rarely occurring substrings may be more private. So the dictionary trainer's interests are broadly aligned with this goal of not including private information in the dictionary.

While existing public dictionary training algorithms largely do not include specific protections or offer hard guarantees to prevent the inclusion of private data in their output, there is ongoing research in this area. Future algorithms may be able to provide confidence that private data (that is not somehow overrepresented in the training corpus) will be filtered out of the produced dictionary.

4.9. Complexity

Complexity is ever the enemy of security. It is unavoidably the case that dictionary-based compression is more complicated than stateless compression.

5. Conclusions

This document attempts to analyze risks and responses at the intersection of several widely varying factors--the protocol, the environment, the threat model--and its conclusions are necessarily situational.

From that space of configurations, some broad conclusions can nonetheless be drawn. Much of the complexity and risk in implementing dictionary-based compression comes from its surrounding apparatus: creating dictionaries, handling them, distributing them,

storing them, identifying them, and so on. A significant distinction can therefore be drawn between systems that have to grapple with those challenges versus those that don't.

[TODO]

6. IANA Considerations

This document includes no actions for IANA.

[RFC Editor: Please remove this section before publication.]

7. Security Considerations

This document enumerates known security considerations about a space that is under development. The list of issues discussed above may not be exhaustive, but it is hopefully complete enough to aid in the design and implementation of future systems and protocols.

8. References

8.1. Normative References

[BROTLI] Alakuijala, J. and Z. Szabadka, "Brotli Compressed Data Format", RFC 7932, DOI 10.17487/RFC7932, July 2016, <https://www.rfc-editor.org/info/rfc7932>.

[DEFLATE] Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3", RFC 1951, DOI 10.17487/RFC1951, May 1996, <https://www.rfc-editor.org/info/rfc1951>.

[ZSTD] Collet, Y. and M. Kucherawy, Ed., "Zstandard Compression and the application/zstd Media Type", RFC 8478, DOI 10.17487/RFC8478, October 2018, <https://www.rfc-editor.org/info/rfc8478>.

8.2. Other Examples of Dictionary-Like Compression

[HPACK] Peon, R. and H. Ruellan, "HPACK: Header Compression for HTTP/2", RFC 7541, DOI 10.17487/RFC7541, May 2015, <https://www.rfc-editor.org/info/rfc7541>.

[HTTP-DELTA-ENCODING] Mogul, J., Krishnamurthy, B., Douglis, F., Feldmann, A., Goland, Y., van Hoff, A., and D. Hellerstein, "Delta encoding in HTTP", RFC 3229, DOI 10.17487/RFC3229, January 2002, <https://www.rfc-editor.org/info/rfc3229>.

[I-D.ietf-quic-qpack]
Krasic, C., Bishop, M., and A. Frindell, Ed., "QPACK: Header Compression for HTTP/3", draft-ietf-quic-qpack-10 (work in progress), 2019.

[I-D.lee-sdch-spec]
Butler, J., Lee, W., McQuade, B., and K. Mixter, "A Proposal for Shared Dictionary Compression over HTTP", draft-lee-sdch-spec-00 (work in progress), October 2016.

[I-D.reschke-http-oob-encoding]
Reschke, J. and S. Loreto, "'Out-Of-Band' Content Coding for HTTP", draft-reschke-http-oob-encoding-12 (work in progress), 2017.

[I-D.vandevenne-shared-brotli-format]
Alakuijala, J., Duong, T., Kliuchnikov, E., Obryk, R., Szabadka, Z., and L. Vandevenne, Ed., "Shared Brotli Compressed Data Format", draft-vandevenne-shared-brotli-format-04 (work in progress), August 2019.

[I-D.vkrasnov-h2-compression-dictionaries]
Krasnov, V. and Y. Weiss, "Compression Dictionaries for HTTP/2", draft-vkrasnov-h2-compression-dictionaries-03 (work in progress), 2018.

8.3. Informative References

[BREACH] Prado, A., Harris, N., and Y. Gluck, "BREACH: SSL, Gone in 30 Seconds", 2013, <https://breachattack.com/>.

[CLOUDFLARE-NO-COMPRESS]
Loring, B., "A Solution to Compression Oracles on the Web", March 2018, <https://blog.cloudflare.com/a-solution-to-compression-oracles-on-the-web/>.

[COOKIES] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <https://www.rfc-editor.org/info/rfc6265>.

[COVER] Liao, K., Petri, M., Moffat, A., and A. Wirth, "Effective Construction of Relative Lempel-Ziv Dictionaries", DOI 10.1145/2872427.2883042, 2016, <https://doi.org/10.1145/2872427.2883042>.

[CRIME] Rizzo, J. and T. Duong, "Compression Ratio Info-leak Made Easy", 2012, <https://www.ekoparty.org/archive/2012/CRIME_ekoparty2012.pdf>.

[ENCRYPT-THEN-AUTHENTICATE]
Krawczyk, H., "The Order of Encryption and Authentication for Protecting Communications (Or: How Secure is SSL?)", 2001, <https://iacr.org/archive/crypto2001/21390309.pdf>.

[HEIST] Vanhoef, M. and T. Van Goethem, "HEIST: HTTP Encrypted Information can be Stolen through TCP-windows", 2016, <https://tom.vg/papers/heist_blackhat2016.pdf>.

[HTTP-CACHING]
Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, DOI 10.17487/RFC7234, June 2014, <https://www.rfc-editor.org/info/rfc7234>.

[I-D.arkko-arch-internet-threat-model]
Arkko, J., "Changes in the Internet Threat Model", draft-arkko-arch-internet-threat-model-01 (work in progress), July 2019.

[I-D.draft-farrell-etm]
Farrell, S., "We're gonna need a bigger threat model", draft-farrell-etm-03 (work in progress), July 2019.

[I-D.draft-kucherawy-httpbis-dict-sec]
Kucherawy, M., "Security Considerations Regarding Compression Dictionaries", draft-kucherawy-httpbis-dict-sec-00 (work in progress), November 2018.

[I-D.pironti-tls-length-hiding]
Pironti, A. and N. Mavrogiannopoulos, "Length Hiding Padding for the Transport Layer Security Protocol", draft-pironti-tls-length-hiding-02 (work in progress), September 2013.

[LZ77] Ziv, J. and A. Lempel, "A Universal Algorithm for Sequential Data Compression", DOI 10.1109/TIT.1977.1055714, May 1977, <https://ieeexplore.ieee.org/document/1055714>.

[LZ78] Ziv, J. and A. Lempel, "Compression of individual sequences via variable-rate coding", DOI 10.1109/TIT.1978.1055934, September 1978, <https://ieeexplore.ieee.org/document/1055934>.

[PERVASIVE-MONITORING]
Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <https://www.rfc-editor.org/info/rfc7258>.

[PRIVACY] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <https://www.rfc-editor.org/info/rfc6973>.

[RFC2360] Scott, G., "Guide for Internet Standards Writers", BCP 22, RFC 2360, DOI 10.17487/RFC2360, June 1998, <https://www.rfc-editor.org/info/rfc2360>.

[SECURITY-GUIDELINES]
Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <https://www.rfc-editor.org/info/rfc3552>.

[SRI] Akhawe, D., Braun, F., Marier, F., and J. Weinberger, "Subresource Integrity", March 2014, <https://www.w3.org/TR/SRI/>.

[ZSTD-DICTS]
Collet, Y., Handte, W., and N. Terrell, "5 ways Facebook improved compression at scale with Zstandard", December 2018, <https://code.fb.com/core-data/zstandard/>.

Appendix A. Acknowledgements

The author wishes to acknowledge the following for their help in writing and improving this document: Murray Kucherawy, Yann Collet, Nick Terrell, ... [TODO]

Author's Address

W. Felix P. Handte
Facebook, Inc.
770 Broadway
New York, NY 10003
US

EMail: felixh@fb.com