Global Information Assurance Certification Paper

Google Chrome Notification Analysis in Depth

GIAC (GCIH) Gold Certification

Author: Vincent Lo, lylc.symhonica@gmail.com
Advisor: Dr. Johannes Ullrich
Accepted: June 7th 2021

Abstract

Google Chrome contains a notification feature to allow websites to show meeting reminders, email notifications or message notifications to users in the system tray. It has been discovered its “Push” API has been abused to deliver spam, erotic images or malicious content to the victims. Unfortunately, the notification history is not recorded in Chrome's browsing history. The notification detail appears to be stored in a separate file, which is not in the format that can be easily parsed by the known open source tools or freeware. This situation impacts incident response's triaging exercise. This paper will look into the Google Chrome's notification feature and its storage structure in order to reveal its content for the incident response purpose.

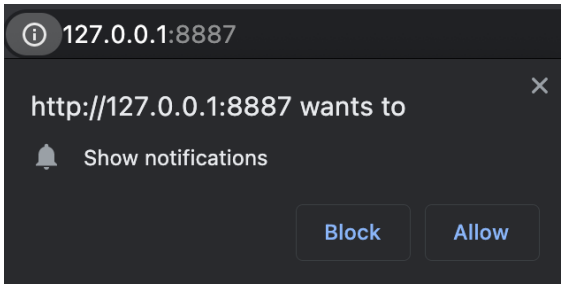
1. Introduction

Google Chrome browser's notification function allows the websites to show email notifications or meeting reminders to users in the system tray, shown as below. It also allows the websites to deliver various messages to the users. Since then, it has been discovered this feature is abused to deliver spam or malicious content to the victims.

During the investigation, the detail of the spam notifications would help incident responders understand where they come from, which can be considered in the containment strategy. However, at the time of writing this paper, the information around how to extract the notification history from Chrome and parse its content is not publicly available on internet. This paper will look into Chrome notification function and its storage mechanism to reveal how to perform this task.

2. Web Notifications

Some websites would like to enrich the user experience by adding the notifications to provide the extra content to the user. If a website wants to use the notification feature, it will need to send a request to the browser for the user's approval. Once the request is accepted, the notification will start to appear in the system tray. This preference will be recorded in the browser setting, which can be modified anytime.



Vincent Lo, lylc.symphonica@gmail.com

2.1. Web Notifications Technologies

Web notifications can be created by two technologies, Push API and Notification API. Since version 42, released on 4 April 2015, Google Chrome browser supports Push API and Notification API (Gaunt, 2020).

2.1.1. Notification API

Email notifications or meeting reminders tend to use Notification API. This API allows them to be displayed in the system tray event if the user switch to a different browser tab or a different program (MDN Contributors, 2020).

2.1.2. Push API

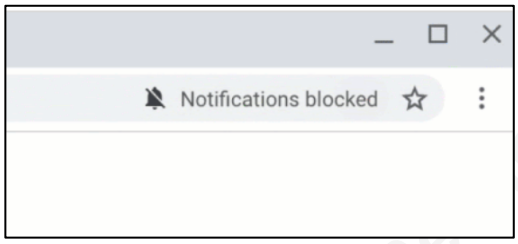
Push API is similar to Notification API. However, it allows the servers to push the messages to the web applications. Even when the websites are not loaded in the browser, the messages can still be displayed in the system tray (MDN Contributors, 2020).

3. Google Chrome Notifications

3.1. Notifications Settings

Chrome's Notifications settings can be found through "Settings" → "Privacy and security" → "Site settings", shown as below. By default, websites can request to send notifications to Chrome. If requests are accepted, the websites are allowed to show the notifications in the system tray. The Notifications setting lists those websites in the "Allow" section. If the requests are rejected, those websites will be listed in the "Block" section.

After the notification feature starts being abused to distribute spam or malicious content, Google has been trying to address it. In Chrome 80, a quieter notification UI is introduced to silently block the notification requests. This feature can be enabled under one of the followings conditions (McLachlan, 2020).



1. **Manual enrollment**

   Users can enable the setting manually by checking the "Use quieter messaging" checkbox or disable it.

2. **User behavior**

   If users frequently deny notification requests, the quieter notifications UI will be enabled automatically.

3. **Notification request's low acceptance rate**

   If the acceptance rate of a website's notification request is low, the quieter notification UI for that website will be enabled automatically.

In Chrome 84, released on 14 July 2020, the quieter notifications UI will be automatically enabled for websites with abusive permission requests or abusive notifications. In Chrome 86, 6 October 2020, the quieter notification UI will be automatically enabled for the abusive notification content (McLachlan, 2020).

3.2. Chrome Notification Storage

Chrome doesn't store the notification in the browser history. It stores the notifications, created by the Push API, in a LevelDB database instead, which can be found in the following folders.

Operating System	Path
Windows	`%LOCALAPPDATA%\Google\Chrome\User Data\Default\Platform Notifications`
Mac OS X	`~/Library/Application Support/Google/Chrome/Default/Platform Notifications`
Linux	`~/.config/google-chrome/Default/Platform Notifications`

4. LevelDB

In order to extract the notifications from a LevelDB database, it is important to know how it stores the data. LevelDB is an open-source key-value database program developed by Google. Keys and values are stored in arbitrary byte arrays. Unlike the traditional relational databases, LevelDB databases don't support indexes nor SQL queries. (Ghemawat & Dean, 2020).

4.1. LevelDB File Structure

A LevelDB database is supported by multiple files (Google, 2020). This section will go through them.

- **Log Files**

When a notification is created through the Push API, its information and its updates are recorded in a log (*.log) file. A copy of this log file is also stored in the memory with the data structure named MemTable. Once this log file reaches a predefined size (around 4 MB by default) , it will be saved in a sorted table file and a new log file will be created to store the new data (Google, 2020). The experiment conducted with Chrome 88 shows the predefined log size in Chrome remains 4 MB. The structure of the log file will be explained in the following section.

- **Sorted Table Files**

A sorted table file contains a list of sorted key-value pairs. The keys and values are in arbitrary byte arrays. Initially, its file extension is .sst. However, in Microsoft Windows operating system, a file with the .sst file extension is used to store Microsoft serialised certificate, which is monitored by System Restore in Windows Vista or later (Microsoft, 2018). From LevelDB 1.14, the file extension is changed from .sst to .ldb (Grogan, 2013). Its file format will be discussed later in this paper.

- **MANIFEST**

A MANIFEST file stores the metadata of the database. Every time when the database is opened, a new manifest will be created with a new number in the filename.

- **Current**

CURRENT is the file that contains the latest Manifest file name.

- **Information Logs**

LOG and LOG.old store the database messages. An example is shown below.

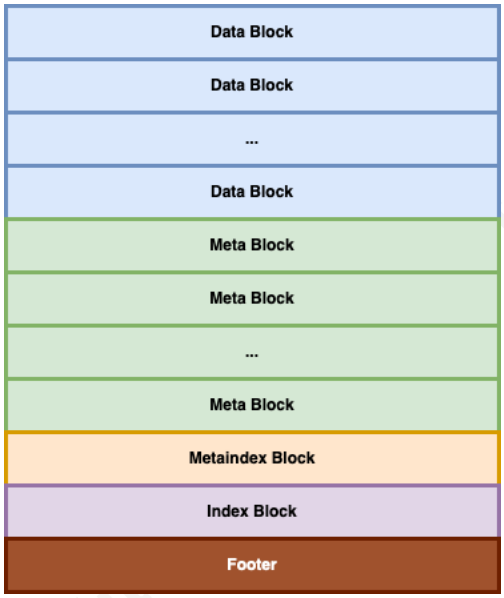
```
2021/01/07-09:35:17.350 6703 Reusing MANIFEST /home/[user name]/.config/google-chrome/Default/Platform Notifications/MANIFEST-000001
2021/01/07-09:35:17.351 6703 Recovering log #3
2021/01/07-09:35:17.351 6703 Reusing old log /home/[user name]/.config/google-chrome/Default/Platform Notifications/000003.log
```

- **Others**

Other miscellaneous files, such as LOCK, may be found in the folder.

5. LevelDB .ldb/.sst File Format

The .ldb/.sst file contains the file structure, shown in the diagram below. The data blocks store the key-value pairs. The following meta blocks store filter policy and stats. The Metaindex block stores entries or those meta blocks. The index block stores the entries of the data blocks (Google, 2017).



The footer has a fixed length, which is 48 bytes. It contains the block handle for metaindex, block handle for index, padding and file signature, 0xdb4775248b80fb57 (Google, 2017).

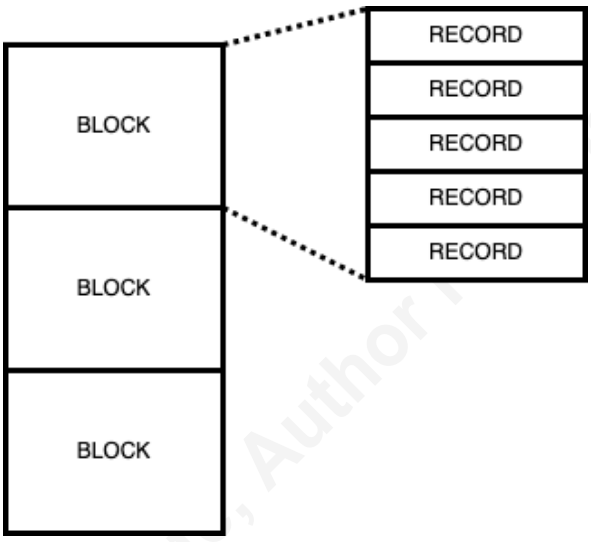
Currently, many open source tools are available to read the .ldb/.sst file. However, the CURRENT file and MANIFEST file may need to be present in the same folder to allow those tools parse the files correctly.

If a .ldb file is found in Chrome's Platform Notifications folder, this database contains the notifications that are copied from the log file once the log file exceeds 4MB. However, the value is the notification data or its update serialised with Protocol Buffer encoding mechanism, which will be discussed in the session seven.

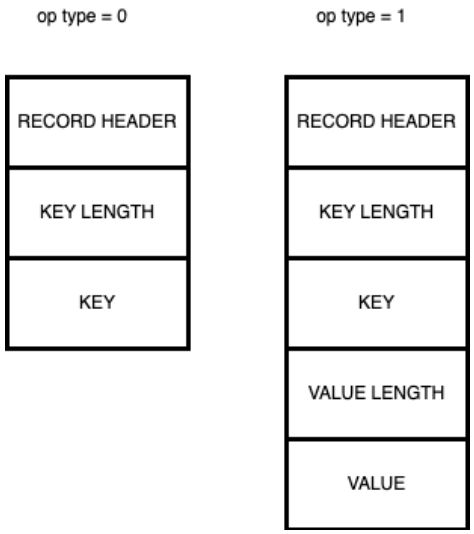
6. LevelDB Log Parsing

6.1. LevelDB Log format

LevelDB log (*.log) stores the notifications created by Push API and their updates. The log file is divided in 32 KB blocks. Every block stores multiple records.



Depending on the op type, some records may contain a record header, key length, key. Some records may contain a record header, key length, key, value length and value. Key length and value length are encoded in varint format. The diagram below shows the difference between the op types.



6.1.1. Record Header

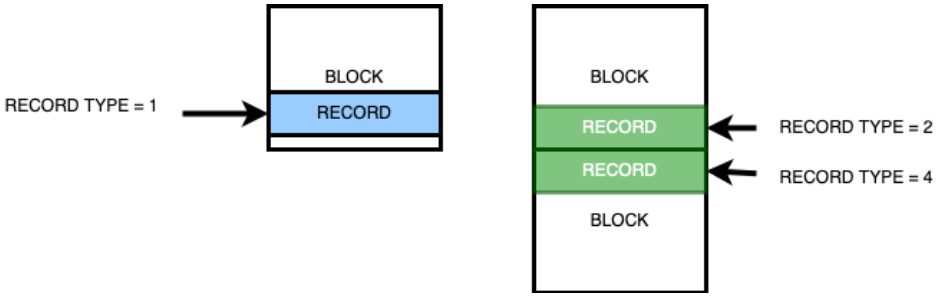
Each record header consists of the following fields. CRC field contains the CRC32C checksum of the record type and the following data in the record (Mumfold, 2017).

Byte Offset	Field Length	Field Name
0x00	4	CRC
0x04	2	Record Length
0x06	1	Record Type
0x07	8	Sequence Number
0x15	4	Count
0x19	1	Op Type

Record Length stores the length of the record in the little endian format. Record type shows whether the record is complete in the block.

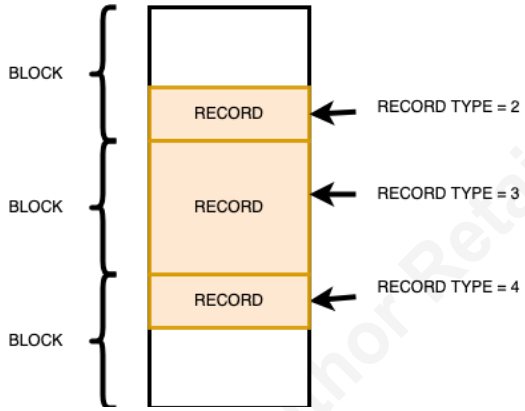
Record Type	Description
1	The whole record is stored in this block.
2	The data is stored across multiple blocks. This record contains the first part of the data.
3	The data is stored across multiple blocks. This record contains the middle part of the data.
4	The data is stored across multiple blocks. This record contains the final part of the data.

The diagrams illustrates when the whole record is stored in the block, the record type would be 1. However, when the data is stored across two blocks, the data will be stored in two records. The first record's record type will be 2 and the next one's record type will be 4.



However, if the data is stored across more than two blocks, shown in the diagram below, data will be stored in more than two records. The first record's record type will be 2. The middle record's record type will be 3. The last one's record type will be 4. Please note if the record type is 2, 3 or 4, the Sequence Number field, the Count field and the Op Type field won't be present in the record.

Vincent Lo, lylc.symphonica@gmail.com



The Sequence Number field stores the sequence number of the first key-value pair in the record. The Count field in the record header shows how many key-value pairs are stored in the record. The Op Type field shows whether the key pair is added or deleted. If it is 1, the key-value pair will be added. If it is 0, the key will be deleted. Please note if Op Type is 0, the value won't be present.

6.1.2. Key-Value Pair

In the record, the value is encoded with the protocol buffer format. The definition of this particular protocol buffer format is stored in "notification_database_data.proto", which can be found in Chromium's source code repository. The link can be found in References and Appendix A.

7. Protocol Buffers

Protocol Buffers (Protobuf) is an encoding mechanism that can serialise the structured data with different encoding algorithms. Google starts developing it in early 2001. (Google, 2020) It continues to evolve since then.

Protobuf stores the encoding structure in a definition (*.proto) file, which can allow the users to declare the field, type and the structure. An example, a snippet of "notification_database_data.proto", is shown as below. A full copy can be found in Appendix A.

```
optional string title = 1;
optional Direction direction = 2;
optional string lang = 3;
optional string body = 4;
optional string tag = 5;
optional string image = 15;
optional string icon = 6;
optional string badge = 14;
repeated int32 vibration_pattern = 9 [packed=true];
optional int64 timestamp = 12;
optional bool renotify = 13;
optional bool silent = 7;
optional bool require_interaction = 11;
optional bytes data = 8;
repeated NotificationAction actions = 10;
// Stored as offset from the windows epoch in microseconds.
optional int64 show_trigger_timestamp = 16;
```

7.1. Encoding

In the Protobuf definition file, the data is defined in the format below. For instance, in the snippet above, the first line is "optional string title = 1". It means the type is "string". The field name is "title". The field number is "1".

```
[type] [field name] = [field number]
```

When Protobuf follows the definition file to encode the value, the encoded data will be represented in the following binary format.

```
[field & type] [data]
```

7.1.1. Fields & Types

Protobuf puts the field and type in one byte. The first five bits stores the field number. The following three bits store type number. The following is an example.

```
00001000
```

The first five bits are 00001, which means the field number is 1. The following three bits are 000, which means the type number is 0. It suggests this field's type is Varint, which is used for int32, int64, uint32, uint64, sint32, sint64, bool and num. The list below shows the types and their meanings (Google, 2020).

Type	Meaning	Used For
0	Varint	int32, int64, uint32, uint64, sint32, sint64, bool, enum
1	64-bit	fixed64, sfixed64, double
2	Length-delimited	string, bytes, embedded messages, packed repeated fields
3	Start group	groups (deprecated)
4	End group	groups (deprecated)
5	32-bit	fixed32, sfixed32, float

7.1.2. Varint

Varint is an important encoding method used in many areas. Its flexibility can allow it to represent a large integer.

This encoding method uses the first bit to determine whether the next byte is part of value. Hence, it can help us determine how many bytes are used for this integer. The following is an example.

```
0000 0100
```

The first bit is 0. That means the next byte is not part of the value. In other words, this value only uses this byte. It uses two's complement to store the value, which is 4. The following is another example.

```
1000 0000 0000 0100 0000 1101
```

This first bit of the first byte is 1. It means the next byte is also part of the value. The first bit of the second byte is 0. It means the following byte is not part of the value. In other words, only the first two bytes, 1000 0000 and 0000 0100, are used to store the value. In order to decode these two bytes, it is important to know how Varint stores the value.

Since the first bit is used to indicate whether the following byte is part the value, it is not used to part of the value representation. Hence, the first bits of those two bytes can be removed during the decoding process.

```
1000 0000 0000 0100
```

Varint stores the least significant group first. The order of these bytes needs to be reversed, shown as below. Through two's complement, the final value is 512.

```
000 0100 000 0000 = 0000 0010 0000 0000  = 512
```

7.1.3. Length-Limited

Protobuf uses the length-limited type to store strings, bytes, and Protobuf structure data. This type is composed of two parts, data length in varint and data shown as below.

```
[data length in Varint] [data]
```

The following hexadecimal numerals is an example. The first hexadecimal numeral is 04, which means the length of the data is 4. The following four numbers is data, which is “test” represented in hexadecimal.

```
04 74 65 73 74
```

7.1.4. Signed Integer

Protobuf uses “zig-zags” encoding for signed integers. This encoding algorithm is also used in the V8 serialization mechanism, which will be covered in the following section.

The following is a short list of the “zig-zags” encoding examples. It interleaves positive numbers and negative numbers to corresponding numbers.

Signed Integer	Encoded Code
0	0
-1	1
1	2
-2	3
2	4
-3	5
3	6
-4	7
4	8

This encoding concept can be translated into the following encoding algorithm and decoding algorithm.

Zig-Zag	Algorithm
Encoding	([signed integer] >> [length of bits] - 1) ^ ([signed integer] << 1)
Decoding	([encoded code] >> 1) ^ -([encoded code] & 1)

7.1.5. Fixed Integer

Protobuf also supports fixed integer types, such as float (32-bit) and double (64-bit). When those types are used, a fixed amount of bytes will be reserved for the value, which will be stored in the little endian format (Google, 2020).

8. V8 Serialization

V8 is a JavaScript and WebAssemby engine, used in Chrome & Node.js (V8, n.d.). V8 contains an encoding mechanism that can serialise the data. It has been observed Chrome's notification log file that may contain the data serialised with V8 serialization mechanism.

V8 serialization mechanism is similar to Protobuf. However, V8 serialization uses the "serialization tag" to define the data type. The following is an example.

```
22 04 74 65 73 74
```

In Protobuf, the field number and type are defined in the first byte. In V8 serialization, the field number is not included. The first byte defines the type of the data. In V8's source code, it is called "serialization tag." A list of the serialization tags is included in Appendix B. In the example, the first byte is `22`. That tag means "One Byte String" data type. The following byte, `04`, is the length of data. That means the following four bytes are part of value which is "test" represented in hexadecimal.

The field name and JavaScript object syntax can also be included in V8's serialised data. The following is an example.

```
6f 22 04 74 65 73 74 49 02 7b 01
```

The first tag, `6f`, means the beginning of the JavaScript object. The following five bytes, `22 04 74 65 73 74`, as explained in the previous paragraph, means "test". The following tag, `49`, means the int32 data type with "zig-zags" encoding algorithm. Hence, the following byte, `02`, means 1. The following tag, `7b`, means the end of the JavaScript object. The following byte, `01`, means only one element in this object. To put the result in the JavaScript object syntax, the following is what it looks like.

```
{ test: 1 }
```

The following is another example. The same principle applies.

```
6f 22 04 74 65 73 74 49 02 6f 22 03 61 62 63 49 06 7b 01 7b 02
```

The following is the decoded result represented in the JavaScript object syntax.

```
{ test: 1, { abc: 3 } }
```

Vincent Lo, lylc.symphonica@gmail.com

9. Conclusion

Chrome stores notification data created by Push API in a LevelDB database. The relevant files of this database are placed in the “Platform Notifications” folder. The experiment conducted with Chrome 88 reveals when the internet history is cleared, the data in that folder is not deleted.

In order to parse Chrome’s notification data, the relevant values in the .ldb file or .log file need to be extracted. Depending on the tools, some supporting files, such as CURRENT and MANIFEST, may be required to perform this task.

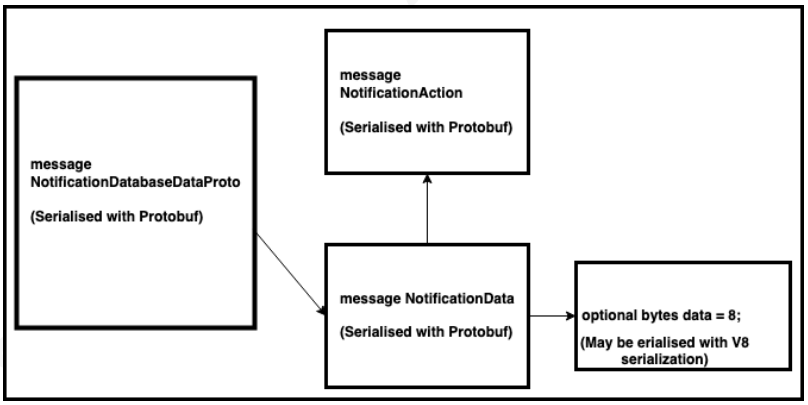


Figure 1: notification_database_data.proto

Those values are serialized data with Protobuf mechanism. Once the values are extracted, they need to be deserialized/decoded with the serialization definition stored in “notification_database_data.proto” so the data can be interpreted correctly.

```
00000000: 546b 7897 1e01 0101 0000 0000 0000 0001  Tkx.............
00000010: 0000 0001 3844 4154 413a 6874 7470 5f31  ....8DATA:http_1
00000020: 3237 2e30 2e30 2e31 5f38 3838 3700 7023  27.0.0.1_8887.p#
00000030: 6874 7470 3a2f 2f31 3237 2e30 2e30 2e31  http://127.0.0.1
00000040: 3a38 3838 372f 2330 3130 3238 32d6 0112  :8887/#010282...
00000050: 1668 7474 703a 2f2f 3132 372e 302e 302e  .http://127.0.0.
00000060: 313a 3838 3837 2f18 3622 8501 0a0c 5075  1:8887/.6"....Pu
00000070: 7368 2043 6f64 656c 6162 1002 1a00 220f  sh Codelab....".
00000080: 4743 4948 2047 6f6c 6420 5061 7065 722a  GCIH Gold Paper*
00000090: 0032 2568 7474 703a 2f2f 3132 372e 302e  .2%http://127.0.
000000a0: 302e 313a 3838 3837 2f69 6d61 6765 732f  0.1:8887/images/
000000b0: 6963 6f6e 2e70 6e67 3800 5800 60bf c0d1  icon.png8.X.`...
000000c0: b1d8 a9c6 1768 0072 2668 7474 703a 2f2f  .....h.r&http://
000000d0: 3132 372e 302e 302e 313a 3838 3837 2f69  127.0.0.1:8887/i
000000e0: 6d61 6765 732f 6261 6467 652e 706e 677a  mages/badge.pngz
000000f0: 002a 1f70 2368 7474 703a 2f2f 3132 372e  .*.p#http://127.
00000100: 302e 302e 313a 3838 3837 2f23 3031 3032  0.0.1:8887/#0102
00000110: 3832 3000 3800 4000 48e0 a1d2 b1d8 a9c6  820.8.@.H.......
00000120: 1768 0270 00                             .h.p.
```

Figure 2: 000003.log

```
key = DATA:http_127.0.0.1_8887p#http://127.0.0.1:8887/#010282
ORIGIN = http://127.0.0.1:8887/
SERVICE_WORKER_REGISTRATION_ID = 54
TITLE = Push Codelab
DIRECTION = 2
BODY = GCIH Gold Paper
ICON = http://127.0.0.1:8887/images/icon.png
SILENT = 0
REQUIRE_INTERACTION = 0
TIMESTAMP = 13257144660156479 (2021-02-07 04:11:00z)
RENOTIFY = 0
BADGE = http://127.0.0.1:8887/images/badge.png
NOTIFICATION_ID = p#http://127.0.0.1:8887/#010282
REPLACED_EXISTING_NOTIFICATION = 0
NUM_CLICKS = 0
NUM_ACTION_BUTTON_CLICKS = 0
CREATION_TIME_MILLIS = 13257144660168928 (2021-02-07 04:11:00z)
CLOSED_REASON = 2
HAS_TRIGGERED = 0
```

Figure 3: Deserialised Content

Vincent Lo, lylc.symphonica@gmail.com

The notification data contains a lot of information. The example above shows the data contains the notification message, its origin, its URL for the icon and its creation date. Those details can be very useful for the incident response purpose. The notification data also records whether the user clicks the notification. The full list of the content can be found in "notification_database_data.proto." It is recommended to read though it if the notification-related incident occurs. The notification data may provide the critical information for the investigation.

Vincent Lo, lylc.symphonica@gmail.com

References

MDN Contributors. (2020, December 18). *Notification API*. MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/API/Notifications_API

MDN Contributors. (2021, January 3). *Using the Notifications API*. MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/API/Notifications_API/Using_the_Notifications_API

MDN Contributors. (2020, December 21). *Push API*. MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/API/Push_API

Mineer, A. (2015, April 14). *Stable Channel Update*. Chrome Releases. https://chromereleases.googleblog.com/2015/04/stable-channel-update_14.html

Bommana, P. (2020, October 6). *Stable Channel Update for Desktop*. Chrome Releases. https://chromereleases.googleblog.com/2020/10/stable-channel-update-for-desktop.html

Google. (n.d.). *Turn notifications on or off*. Google Chrome Help. Retrieved January 17, 2021, from https://support.google.com/chrome/answer/3220216?co=GENIE.Platform%3DDesktop&hl=en

Medley, J. (2019, February 12). *Web Push Notifications: Timely, Relevant, and Precise*. Web Fundamentals. https://developers.google.com/web/fundamentals/push-notifications

McLachlan, P. (2020, January 7). *Introducing quieter permission UI for notifications*. Chromium Blog. https://blog.chromium.org/2020/01/introducing-quieter-permission-ui-for.html

McLachlan, P. (2020, May 28). *Protecting Chrome users from abusive notifications*. Chromium Blog. https://blog.chromium.org/2020/05/protecting-chrome-users-from-abusive.html

McLachlan, P. (2020, October 21). *Reducing abusive notification content*. Chromium Blog. https://blog.chromium.org/2020/10/reducing-abusive-notification-content.html

Vincent Lo, lylc.symphonica@gmail.com

Gaunt, M. (2020, July 24). *Push Notifications on the Open Web*. Web Updates. https://developers.google.com/web/updates/2015/03/push-notifications-on-the-open-web

Google. (n.d.). *User Data Directory*. Chromium Docs. Retrieved January 17, 2021, from https://chromium.googlesource.com/chromium/src/+/master/docs/user_data_dir.md

Ghemawat, S., & Dean, J. (2020, December 17). *README.md*. leveldb. https://github.com/google/leveldb

Google. (2020, December 17). *impl.md*. leveldb. https://github.com/google/leveldb/blob/master/doc/impl.md

Google. (2017, March 1). *leveldb File format*. leveldb. https://github.com/google/leveldb/blob/master/doc/table_format.md

Grogan, David. (2013, September 20). *Release LevelDB 1.14*. leveldb. https://github.com/google/leveldb/commit/0b9a89f40efdd143fa1426e7d5cd997f67ba6361

Grigorik, Ilya. (2012, February 06). *SSTable and Log Structured Storage: LevelDB*. igvita.com. https://www.igvita.com/2012/02/06/sstable-and-log-structured-storage-leveldb/

Microsoft. (2018, May 31). *Monitored File Name Extensions*. Microsoft Docs. https://docs.microsoft.com/en-us/windows/win32/sr/monitored-file-extensions?redirectedfrom=MSDN

Grogan, David. (2013, September 17). *Changing file extension from .sst to .ldb*. Google Groups. https://groups.google.com/g/leveldb/c/u9izbG-pDis

Mumford, Chris. (2017, March 1). *leveldb Log format*. LevelDB. https://github.com/google/leveldb/blob/master/doc/log_format.md

Google. (2020, Oct 27). *notification_database_data.proto*. Chromium. https://github.com/chromium/chromium/blob/master/content/browser/notifications/notification_database_data.proto

Google. (2020, June 23). *Frequently Asked Questions*. Protocol Buffers. https://developers.google.com/protocol-buffers/docs/faq

Vincent Lo, lylc.symphonica@gmail.com

Google. (2020, December 21). *Encoding*. Protocol Buffers. https://developers.google.com/protocol-buffers/docs/encoding

V8. (n.d.). *What is V8?*. Home. Retrieved January 25, 2021, from https://v8.dev/

Vincent Lo, lylc.symphonica@gmail.com

Appendix A
notification_database_data.proto

A copy of "notification_database_data.proto" from https://github.com/chromium/chromium/blob/master/content/browser/notifications/notification_database_data.proto is provided as below.

```
// Copyright 2015 The Chromium Authors. All rights reserved.
// Use of this source code is governed by a BSD-style license that can be
// found in the LICENSE file.

syntax = "proto2";

option optimize_for = LITE_RUNTIME;

package content;

// Stores information about a Web Notification. This message is the protocol
// buffer meant to serialize the content::NotificationDatabaseData structure.
//
// Next tag: 15
message NotificationDatabaseDataProto {
  enum ClosedReason {
    USER = 0;
    DEVELOPER = 1;
    UNKNOWN = 2;
  }

  // DEPRECATED: Use |notification_id| instead.
  optional int64 persistent_notification_id = 1;

  optional string notification_id = 5;

  optional string origin = 2;
  optional int64 service_worker_registration_id = 3;
  optional bool replaced_existing_notification = 6;
  optional int32 num_clicks = 7;
  optional int32 num_action_button_clicks = 8;
  optional int64 creation_time_millis = 9;
  optional int64 time_until_first_click_millis = 10;
  optional int64 time_until_last_click_millis = 11;
  optional int64 time_until_close_millis = 12;
  optional ClosedReason closed_reason = 13;

  // A notification action, corresponds to blink::PlatformNotificationAction.
  //
  // Next tag: 6
  message NotificationAction {
    // Corresponds to blink::PlatformNotificationActionType.
    enum Type {
      BUTTON = 0;
      TEXT = 1;
    }

    optional string action = 1;
    optional string title = 2;
    optional string icon = 3;
```

Vincent Lo, lylc.symphonica@gmail.com

```
  optional Type type = 4;
  optional string placeholder = 5;
 }

 // Actual data payload of the notification. This message is the protocol
 // buffer meant to serialize the blink::PlatformNotificationData structure.
 //
 // Next tag: 17
 message NotificationData {
  enum Direction {
   LEFT_TO_RIGHT = 0;
   RIGHT_TO_LEFT = 1;
   AUTO = 2;
  }

  optional string title = 1;
  optional Direction direction = 2;
  optional string lang = 3;
  optional string body = 4;
  optional string tag = 5;
  optional string image = 15;
  optional string icon = 6;
  optional string badge = 14;
  repeated int32 vibration_pattern = 9 [packed=true];
  optional int64 timestamp = 12;
  optional bool renotify = 13;
  optional bool silent = 7;
  optional bool require_interaction = 11;
  optional bytes data = 8;
  repeated NotificationAction actions = 10;
  // Stored as offset from the windows epoch in microseconds.
  optional int64 show_trigger_timestamp = 16;
 }

 optional NotificationData notification_data = 4;

 // Keeps track if a notification with a |show_trigger_timestamp| has been
 // displayed already.
 optional bool has_triggered = 14;
}
```

Appendix B
V8's Serialization Tags

A list of serialization tags from https://github.com/v8/v8/blob/master/src/objects/value-serializer.cc is provided as below.

```
enum class SerializationTag : uint8_t {
  // version:uint32_t (if at beginning of data, sets version > 0)
  kVersion = 0xFF,
  // ignore
  kPadding = '\0',
  // refTableSize:uint32_t (previously used for sanity checks; safe to ignore)
  kVerifyObjectCount = '?',
  // Oddballs (no data).
  kTheHole = '-',
  kUndefined = '_',
  kNull = '0',
  kTrue = 'T',
  kFalse = 'F',
  // Number represented as 32-bit integer, ZigZag-encoded
  // (like sint32 in protobuf)
  kInt32 = 'I',
  // Number represented as 32-bit unsigned integer, varint-encoded
  // (like uint32 in protobuf)
  kUint32 = 'U',
  // Number represented as a 64-bit double.
  // Host byte order is used (N.B. this makes the format non-portable).
  kDouble = 'N',
  // BigInt. Bitfield:uint32_t, then raw digits storage.
  kBigInt = 'Z',
  // byteLength:uint32_t, then raw data
  kUtf8String = 'S',
  kOneByteString = '"',
  kTwoByteString = 'c',
  // Reference to a serialized object. objectID:uint32_t
  kObjectReference = '^',
  // Beginning of a JS object.
  kBeginJSObject = 'o',
  // End of a JS object. numProperties:uint32_t
  kEndJSObject = '{',
  // Beginning of a sparse JS array. length:uint32_t
  // Elements and properties are written as key/value pairs, like objects.
  kBeginSparseJSArray = 'a',
  // End of a sparse JS array. numProperties:uint32_t length:uint32_t
  kEndSparseJSArray = '@',
  // Beginning of a dense JS array. length:uint32_t
  // |length| elements, followed by properties as key/value pairs
  kBeginDenseJSArray = 'A',
  // End of a dense JS array. numProperties:uint32_t length:uint32_t
```

Vincent Lo, lylc.symphonica@gmail.com

```
kEndDenseJSArray = '$',
// Date. millisSinceEpoch:double
kDate = 'D',
// Boolean object. No data.
kTrueObject = 'y',
kFalseObject = 'x',
// Number object. value:double
kNumberObject = 'n',
// BigInt object. Bitfield:uint32_t, then raw digits storage.
kBigIntObject = 'z',
// String object, UTF-8 encoding. byteLength:uint32_t, then raw data.
kStringObject = 's',
// Regular expression, UTF-8 encoding. byteLength:uint32_t, raw data,
// flags:uint32_t.
kRegExp = 'R',
// Beginning of a JS map.
kBeginJSMap = ';',
// End of a JS map. length:uint32_t.
kEndJSMap = ':',
// Beginning of a JS set.
kBeginJSSet = '\'',
// End of a JS set. length:uint32_t.
kEndJSSet = ',',
// Array buffer. byteLength:uint32_t, then raw data.
kArrayBuffer = 'B',
// Array buffer (transferred). transferID:uint32_t
kArrayBufferTransfer = 't',
// View into an array buffer.
// subtag:ArrayBufferViewTag, byteOffset:uint32_t, byteLength:uint32_t
// For typed arrays, byteOffset and byteLength must be divisible by the size
// of the element.
// Note: kArrayBufferView is special, and should have an ArrayBuffer (or an
// ObjectReference to one) serialized just before it. This is a quirk arising
// from the previous stack-based implementation.
kArrayBufferView = 'V',
// Shared array buffer. transferID:uint32_t
kSharedArrayBuffer = 'u',
// A wasm module object transfer. next value is its index.
kWasmModuleTransfer = 'w',
// The delegate is responsible for processing all following data.
// This "escapes" to whatever wire format the delegate chooses.
kHostObject = '\\',
// A transferred WebAssembly.Memory object. maximumPages:int32_t, then by
// SharedArrayBuffer tag and its data.
kWasmMemoryTransfer = 'm',
// A list of (subtag: ErrorTag, [subtag dependent data]). See ErrorTag for
// details.
kError = 'r',

// The following tags are reserved because they were in use in Chromium before
// the kHostObject tag was introduced in format version 13, at
//   v8          refs/heads/master@{#43466}
```

```
  //   chromium/src refs/heads/master@{#453568}
  //
  // They must not be reused without a version check to prevent old values from
  // starting to deserialize incorrectly. For simplicity, it's recommended to
  // avoid them altogether.
  //
  // This is the set of tags that existed in SerializationTag.h at that time and
  // still exist at the time of this writing (i.e., excluding those that were
  // removed on the Chromium side because there should be no real user data
  // containing them).
  //
  // It might be possible to also free up other tags which were never persisted
  // (e.g. because they were used only for transfer) in the future.
  kLegacyReservedMessagePort = 'M',
  kLegacyReservedBlob = 'b',
  kLegacyReservedBlobIndex = 'i',
  kLegacyReservedFile = 'f',
  kLegacyReservedFileIndex = 'e',
  kLegacyReservedDOMFileSystem = 'd',
  kLegacyReservedFileList = 'l',
  kLegacyReservedFileListIndex = 'L',
  kLegacyReservedImageData = '#',
  kLegacyReservedImageBitmap = 'g',
  kLegacyReservedImageBitmapTransfer = 'G',
  kLegacyReservedOffscreenCanvas = 'H',
  kLegacyReservedCryptoKey = 'K',
  kLegacyReservedRTCCertificate = 'k',
};
```