Forensic Science International: Digital Investigation 50 (2024) 301809

Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

journal homepage: www.elsevier.com/locate/fsidi

DFRWS APAC 2024 - Selected Papers from the 4th Annual Digital Forensics Research Conference APAC

MIC: Memory analysis of IndexedDB data on Chromium-based applications

Byeongchan Jeong, Sangjin Lee, Jungheum Park [*]

School of Cybersecurity, Korea University, Seoul, South Korea

ARTICLE INFO

Keywords:
Digital forensics
Memory forensics
Chromium-based applications
Incognito mode
IndexedDB
LevelDB

ABSTRACT

As Chromium-based applications continue to gain popularity, it is necessary for forensic investigators to obtain a comprehensive understanding of how they store and manage browsing artifacts from both filesystem and memory perspectives. In particular, the *incognito* mode developed in the current version of Chromium uses only physical memory to manage data related to active sessions. Therefore, handling physical memory is essential for tracking a user's browsing behaviour in incognito mode. This paper provides an in-depth examination of LevelDB, a lightweight key-value database utilized as Chromium's implementation for IndexedDB. In particular, we delve into the details of how IndexedDB data is managed through LevelDB, taking into account its low-level database file format. Furthermore, we thoroughly explore the possibility of residual data, both complete and incomplete, being retained as applications create and initialize IndexedDB-related data. Based on our research findings, we propose a systematic methodology for inspecting the internal structures of LevelDB-related C++ classes, carving these structures from binary streams, and interpreting the data for forensic analysis. In addition, we develop a proof-of-concept tool based on our approach and demonstrate its performance and effectiveness through case studies.

1. Introduction

The Chromium is widely used as a codebase for various web browsers, including Google Chrome and Microsoft Edge (Google, 2008). In addition, a variety of desktop applications are being developed using frameworks that embed the Chromium to enable the use of web technologies in native application development. Examples of them include Electron (OpenJS Foundation, 2014) and Microsoft Edge WebView2 (Microsoft, 2020).

From a digital forensics perspective, these web browser-related applications are important because they create and manage traces of user's web browsing activities. For example, instant messaging services may generate data such as contacts and chat logs, while cloud storage services may store traces such as lists of uploaded, downloaded and shared files. In terms of forensic analysis on them, various applications built with the Chromium contain artifacts similar to those created by the Google Chrome addressed by existing studies.

More specifically, IndexedDB consists of one or more databases for each domain, and the IndexedDB data is stored in the LevelDB format. An IndexedDB database stores more than one object store containing serialized records. Chromium-based applications also store data in key-value pairs, so it is important to have a deep understanding of the IndexedDB structures and how metadata and records are managed in key-value pairs. In addition, Chromium-based browsers support an incognito mode (Private mode) that stores data in volatile memory without writing it to local disk. Therefore, analyzing the data left in memory dumps becomes a crucial aspect of digital forensic investigations.

This paper aims to identify and extract user data in volatile memory with an understanding of IndexedDB mechanisms. First, we scan the class objects that make up IndexedDB and their structure, and rebuild the IndexedDB data area in memory using the scanned class objects as a starting point. The IndexedDB data consists of a MemTable that is structured as SkipList, which stores key-value paired records. We interpret and classify the key-value pairs stored in the MemTable to produce forensically meaningful results. Based on our research, we experimentally validated our method and developed a proof-of-concept tool, MIC, to help digital forensic investigations.

1.1. Motivation and research questions

With an increased number of online services offering Chromium-based applications, digital forensics community needs to understand how they manage data and respond to challenges posed by the

[*] Corresponding author.
E-mail addresses: naaya@korea.ac.kr (B. Jeong), sangjin@korea.ac.kr (S. Lee), jungheumpark@korea.ac.kr (J. Park).

https://doi.org/10.1016/j.fsidi.2024.301809

Available online 18 October 2024

applications. Furthermore, when Chromium-based browsers are used in incognito mode, they only store data in memory. Many of these applications handle data related to user activity and system data to run the applications. However, existing research focuses on IndexedDB stored on local disks, so there is a relative lack of research on analyzing data remaining in memory.

To solve these challenges, it is important to gain a deep understanding of the structures and mechanisms of IndexedDB and to extract the user data remaining in volatile memory.

In summary, this study led us to the following research questions.

RQ1. How can IndexedDB data be identified and extracted from Chromium-based application effectively?

RQ2. To what degree can data from IndexedDB-related structures be interpreted?

RQ3. To what extent can meaningful forensic data be extracted from the application under incognito mode?

1.2. Contribution

In summary, this research has three major contributions as follows.

- Identifying all the available data from volatile memory by analyzing IndexedDB data manipulation used by Chromium-based application
- Proposing a framework to examine IndexedDB-related artifacts during digital forensic investigation
- Validating our approach through experiments and developing a proof-of-concept tool

The remainder of this paper is structured as follows. Section 2 describes how IndexedDB manages and stores data and its structures. Additionally, we examine research on application-level memory forensics, including artifacts of browser incognito mode within volatile memory, and studies related to IndexedDB in Chromium-based applications. Section 3 outlines our proposed methodology for extracting and analyzing IndexedDB data from memory dumps. Section 4 introduces the MIC tool, a proof-of-concept tool developed based on our proposed methodology. Section 5 presents the setup and results of experiments conducted to evaluate the performance and effectiveness of the MIC tool. Finally, Section 6 summarizes our contributions and offers suggestions for future research directions.

2. Background and related work

2.1. IndexedDB and Chromium-based applications

IndexedDB is a client-side storage solution used in web browsers, designed to overcome the limitations of cookie storage. IndexedDB handles and manages the storage of structured data by supporting transactions to ensure its integrity and consistency. Many web-based services utilize IndexedDB. For example, messaging applications store chat data, attachments, and chat lists, while cloud storage services use IndexedDB to manage file lists, metadata and recent files. It optimizes service performance by reducing server-client communication and leveraging client-side resources. Additionally, IndexedDB enables applications to work offline. Section 2.1 explains the process of how IndexedDB handled user data and provides a detailed description of the data and file formats residing in memory and on disk.

Memory-resident data of IndexedDB handling mechanism IndexedDB has three main components: MemTable, log, and ldb, which are needed to manage and manipulate data in memory. The mechanism of IndexedDB is designed to write and read data effectively, and LevelDB is used to implement IndexedDB. LevelDB, developed by Google, is a fast and lightweight storage library that uses key-value pairs to ensure the performance and stability of IndexedDB (Google, 2011a). MemTable is an in-memory data structure that stores the most recent data. When writing data on IndexedDB, it is recorded in MemTable. As MemTable remains in memory, it enables fast data access. The log file also records the same data. It stores data on a disk, allowing data to be recovered in the event of unexpected events. Once MemTable exceeds the default size (4 MB), LevelDB flushes the data and converts it to an ldb file. The ldb file is stored on disk with aligned key-value pairs, which enables effective data search and ingestion. When MemTable is flushed to disk, IndexedDB allocates a new MemTable to manage the data. When reading data, IndexedDB begins to search for data in the MemTable. If the data is not found, it searches the data from a most recent log file to the oldest. Thus, IndexedDB uses MemTable, log, and ldb to manage data for its consistency and efficiency.

LevelDB file format IndexedDB of Chromium-based applications uses LevelDB as back-end storage to manage data. LevelDB, a key-value paired storage, provides high performance and efficiency for writing and reading data. As mentioned earlier, LevelDB is composed of MemTable, '[0–9]{6}.log', and '[0–9]{6}.ldb' to ensure its performance and database consistency. SkipList-based MemTable is a data structure used for effective data search and insertion (Pugh, 1990). MemTable retains the most recent data in memory and flushes data to ldb files on disk when the MemTable storing data exceeds the default size.

A '[0–9]{6}.log' file records how the data has been changed such as data insertion, modification, and deletion. If an unexpected exception occurs when a transaction run by LevelDB, the database goes back to the state before the transaction occurs to maintain consistency. The exception may cause the data in the MemTable to be lost, it logs data to prevent this. The log file is composed in a sequence of 32 KB blocks, each block has multiple records. A record consists of seven-bytes header (checksum, length, and type) and data of the specified length (Google, 2011c).

When the size of a record exceeds the default block size (32 KB), it uses more blocks in succession. The type value (FULL, FIRST, MIDDLE, and LAST) in header indicates whether the records are consecutive or not. If a single block is used, the type is 'FULL.' If more than one block is used, the type describes the start and end of the blocks using the values FIRST, MIDDLE, and LAST.

A single '[0–9]{6}.ldb' file contains data blocks, meta blocks, meta index block, index block, and footer from the start of the file (Google, 2011b). The ldb file is stored as a sequence of actual key-value pair records, which are separated into a sequence of data blocks of size 4 KB. As the amount of data increases, LevelDB optimizes the space. During this process, LevelDB removes duplicate and deleted records to make efficient use of database space.

2.2. Related work

2.2.1. Identifying significant forensic artifacts for applications stored in memory

Identifying application traces in memory has been addressed for a long time, which has been an ongoing and growing area of research. Previous studies focused on understanding how applications process strings, patterns, or specific features for each operating system. Based on their findings, the authors discovered and extracted forensically important data.

Van Der Horst et al. (2017) and Thomas et al. (2020) examined cryptocurrency client applications on Windows. The authors explored traces of cryptocurrency use by detecting string and binary formatted values from memory dumps.

Wang et al. (2022) studied memory forensics of the V8 JavaScript engine. The authors proposed a method to extract V8 JavaScript engine objects and their descriptors, which can be applied to other applications using the engine.

With recent advances in security technology, the digital forensic community requires volatile memory forensics. While current memory forensics techniques rely on searching for strings, patterns, and specific data structures, they have difficulties in generalizing and applying to a wide variety of situations. To address the challenges, we propose a new methodology to support Chromium Projects with a high market share under the cross-platform environments by making wide use of memory forensic techniques.

2.2.2. Browsers incognito mode and volatile memory

Most web browsers now provide an 'incognito mode', that protects user privacy by not storing browsing history on local devices. From a digital forensic perspective, it is crucial to collect web browser data as it contains important clues to solve criminal cases. When using incognito mode on web browsers, they store user activities in memory temporarily, and these research has been conducted on discovering them.

Satvat et al. (2014) suggested new approach to examine private browsing session of Firefox, Chrome, IE, and Safari. They conducted systematic investigation of local artifacts under the incognito mode. Especially, they confirmed that volatile memory stores visited URLs, password, and cookie.

Mahlous and Mahlous (2020) suggested digital forensic method for Brave browser in a private mode. They studied how to reduce false positives and false negatives when searching for keywords used to scan memory and browsing history. They observed how the amount of data and its type and content differed by taking memory snapshots of incognito and normal mode.

Saputra and Riadi (2020) researched figuring out user data in both normal mode and incognito mode. They discovered local artifacts for Twitter, one of the social network services, after posting text, link, images, videos using Twitter.

Nelson et al. (2020) analyzed forensic artifacts in Chrome, Firefox, and Tor Browser. The authors identified and examined local artifacts under both normal mode and incognito mode. While normal mode stores various local artifacts, incognito mode stores fewer artifacts.

Hariharan et al. (2022) conducted their study on browsing artifacts for portable web browsers including Brave, Tor, Vivaldi, and Maxthon in private mode. After running private mode for each browser, they performed some actions using Facebook, Gmail, Amazon and identified related data in memory.

Zollner et al. (2019) examined web-based bitcoin wallets in browsers such as Chrome, Firefox, IE, Edge, and Tor. They presented a method to discover the artifact related to Bitcoin wallets using regular expressions, file signatures and keywords.

Choi et al. (2023) analyzed source codes of Chromium-based browser to find out classes that associate with the user activity such as creating browser window, adding tab, visiting specific URLs. Based on their findings, they suggested a method to examine web browsing history in memory and developed the tool to automate the processes.

Kim et al. (2024) analyzed IndexedDB, which is used in Gecko-based browsers. Since these browsers use IndexedDB in encrypted SQLite, the authors conducted a study on extracting the encryption key from memory and decrypting IndexedDB.

Despite active research on browser private mode, most studies examine the data using keyword searches or regular expressions with known information, which is limited when data sources are not found in memory. To address this challenge, various memory forensics methodologies have been proposed, such as identifying object layouts associated with a user's web browsing activity and utilizing them to identify data. Developing existing research, we propose an updated memory forensics technique to find out significant data related to user activities stored in the IndexedDB of a Chromium-based application.

2.2.3. IndexedDB of Chromium-based application

As browser-based applications have become more popular, the mechanisms for storing user data have evolved. Web storage enables a reduction in workload and lightens the load on servers, while allowing users to store their data locally. As a result, those common web-based services store data related to user activity as well as system data to provide high-performance services, and a lot of research has been conducted on this topic.

Paligu et al. (2019) proposed a methodology and a developed tool to investigate key forensic artifacts in IndexedDB across five most popular browsers (Chrome, Edge, Fire, Opera, Firefox, and Safari) on popular operating systems (Windows, MacOS, and Ubuntu). They demonstrated that the data stored in IndexedDB can be useful for forensic investigations on fifteen of the most popular websites.

Several studies (Paligu and Varol, 2020, 2022a, 2022b) analyzed Chromium-based applications using IndexedDB. The authors identified user data from the IndexedDB by applying pretest–posttest quasi experiment (Cook and Campbell, 1979) for each case. The study confirmed that the IndexedDB can be useful during digital forensic investigation.

CCL Solutions Group (2020) describes how the IndexedDB data stored on local devices manages LevelDB, and developed an open source tool to interpret the data at the raw level.

Most forensic research and commercial/open source tools related to IndexedDB focus on extracting and analyzing the data in LevelDB within each domain-specific IndexedDB storage that remains on the local device disk. However, there is a lack of research on analyzing how IndexedDB works and manages data. Therefore, we conducted research to gain a deeper understanding of IndexedDB handling mechanisms in volatile memory and to extract data related to user activities.

3. Methodology for memory analysis of IndexedDB data

This study aims to identify Chromium-based applications from memory dumps and extract IndexedDB database of each application. We then select candidates that store IndexedDB through IndexedDB-related data structures. For the valid data structures, we deserialize the serialized records and store them by constructing an integrated schema. Fig. 1 describes an overview of the proposed methodology.

3.1. Extraction of candidate IndexedDB-related objects

In our study, we applied a carving technique based on class object size to extract the IndexedDB-related classes. We determined the InexedDB-related classes analyzing source codes of the Chromium Projects. As the addresses of memory objects are assigned in eight-byte alignment, we set the starting addresses of identified classes as multiples of eight. It starts at the specified address and moves in units of eight multiples until it scans the entire memory corresponding to the size of the object. In addition, it only scans memory areas that the application can read/write to reduce false positives.

As Chromium-based applications are composed of a number of classes to manage and store data. It is necessary to identify class objects to reconstruct IndexedDB, so we analyzed source codes of the Chromium Projects. As a result of our analysis, we found out that DBImpl class is the least unit to extract data from IndexedDB. While the Chromium Projects and IndexedDB class object layouts undergo frequent updates, LevelDB, which is used for back-end storage of IndexedDB, has a less frequent update cycle (Google, 2021). Therefore, we found out DBImpl class to handle the updates of IndexedDB related classes and scan volatile memory using the class. The volatile memory area are scanned by the size of the DBImpl class (0x278) and sent to the validation phase to verify each class field.

3.2. Validation of candidates considering interconnected structures

The IndexedDB structures identified in the extraction phase are validated by matching the data type with the offset where the member fields of the DBImpl class layout should be located in each structure. For example, the objects in Fig. 2 are the target fields for the top-level class, as well as the major fields that make up the class. There is a class named Options inside the class DBImpl that has a size range from hex value 0x30 to 0x68. The Options class include its member fields including

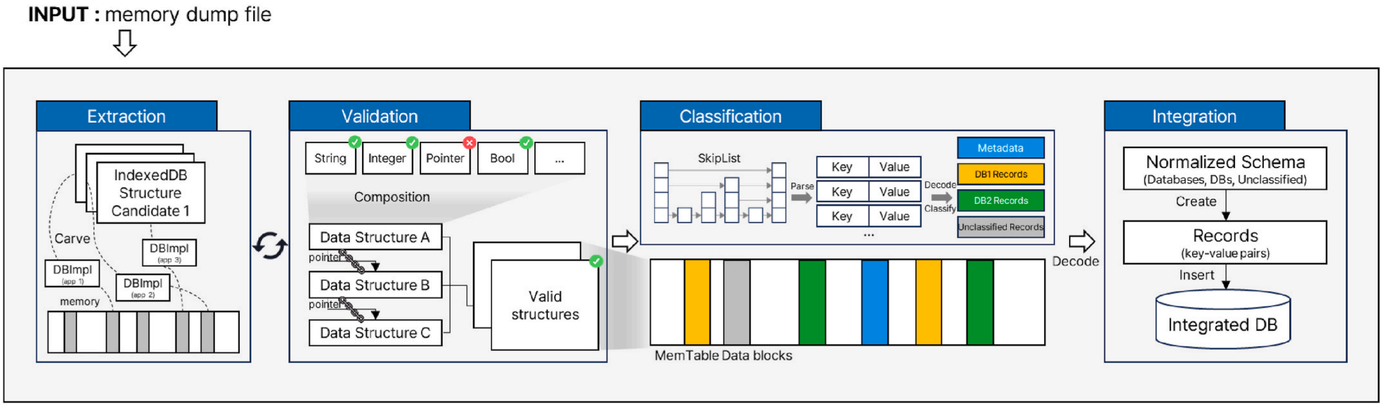
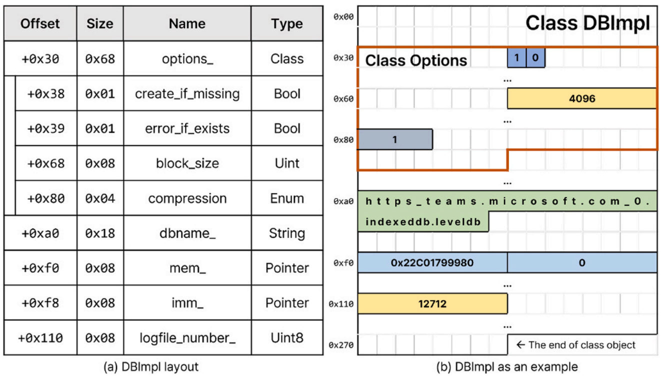


Fig. 1. An overview of the our methodology.



Offset	Size	Name	Type
+0x30	0x68	options_	Class
+0x38	0x01	create_if_missing	Bool
+0x39	0x01	error_if_exists	Bool
+0x68	0x08	block_size	Uint
+0x80	0x04	compression	Enum
+0xa0	0x18	dbname_	String
+0xf0	0x08	mem_	Pointer
+0xf8	0x08	imm_	Pointer
+0x110	0x08	logfile_number_	Uint8

(a) DBImpl layout

(b) DBImpl as an example

Fig. 2. LevelDB’s DBImpl object.

create_if_missing (boolean), error_if_exists (boolean), block_size (unsigned integer), compression (enumeration), and this phase processes its validation by looking over if each field has a type, offset, size in the proper range. When the field matches the defined data type, the structures will be considered as valid, otherwise, they become invalid structures. We set the limited conditions for each data structure and validate the major fields to extract user data from IndexedDB. The IndexedDB-related classes are a set of different data types, and we find candidate objects by adding constraints to validate the class data types.

Fig. 3 demonstrates the classes and the fields that need to be examined to acquire data. If a class has an inherited class, it will store a pointer, which is a common way to represent relationships between classes. To analyze and validate the complex relationships between classes, we classified the candidates of IndexedDB-related structure more accurately. Especially, our study focused on a deep understanding of how the structures and fields are related. During this process, it was important to define and validate appropriate constraints for different data types and complex referential relationships.

This phase validates all the IndexedDB-related structures in volatile memory. If validation fails, our methodology goes back to the extraction phase to scan the memory area.

3.3. *Classification of validated objects based on relevant applications*

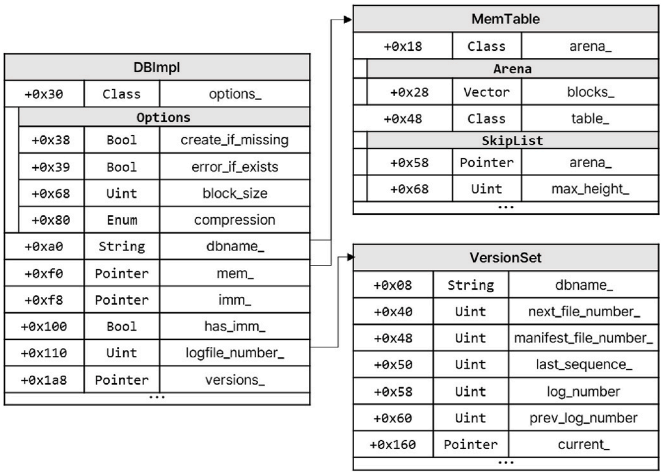
This phase extracts user data stored in IndexedDB by applications from the gathered validated objects. Memory-resident data of IndexedDB is stored in MemTable. The proposed method identifies offset list of storing blocks and block size to reconstruct MemTable in volatile memory. They are stored in a member field ‘blocks_’ of Arena and a member field ‘block_size’ of Options. A member field ‘blocks_’ of Arena objects consists of vector containing offset data list. Options class object stores a member field ‘block_size’, that has information about data block size. With the identified data block and size, we reconstruct it in a sequence of space.

The reconstructed MemTable is structured as a SkipList storing IndexedDB records. Thoroughly parsing the records requires parsing nodes managed by the SkipList. Algorithm 1 illustrates to parse MemTable nodes.

Algorithm 1. Parsing SkipList nodes in MemTable

```
Algorithm 1 Parsing SkipList nodes in MemTable
Input: Memory blocks $blocks$, Block size $block\_size$, Maximum height $max\_height$
Output: Parsed SkipList Nodes
procedure ParseNodes(blocks, block_size, max_height)
    $records \leftarrow$ empty set
    $buf \leftarrow \text{read}(blocks[0], block\_size)$
    for $i \leftarrow 8$ to $max\_height \times 8$ step 8 do
        $ptr \leftarrow$ 8 bytes from $buf[i : i + 8]$
        if $ptr == 0$ then
            continue
        end if
        $ptr \leftarrow \text{read}(ptr, 8)$
        $records.\text{add}(ptr)$
        while is_valid_range($blocks$, $ptr$) do
            $ptr \leftarrow \text{read}(ptr, 8)$
            $records.\text{add}(ptr)$
            $ptr \leftarrow \text{read}(ptr + i, 8)$
        end while
    end for
    return $records$
end procedure
```

While SkipList shares a common feature with a linked list in that it uses ‘pointers’ to represent relationships between nodes, SkipList is composed of multiple levels. To identify the total level, a member field ‘max_height_’ of SkipList should be acquired. The first 8 bytes of the first block are filled with 0, followed by a list of address values. It contains a pointer to the next node of each level from the first node. After skipping the first eight bytes of the first block, the phase reads the address list up to the maximum level. The address list contains pointers to the next node at each level. Then we parse the next node following the pointers for each level. If the value of the pointers exceeds MemTable blocks, it moves to the next pointer. The phase extracts key-value pairs from each node and iterates from the first node to the last node for all levels. The extracted key-value pairs are single IndexedDB record. Fig. 4 illustrates a complete structure of an IndexedDB record.



DBImpl		
+0x30	Class	options_
Options		
+0x38	Bool	create_if_missing
+0x39	Bool	error_if_exists
+0x68	Uint	block_size
+0x80	Enum	compression
+0xa0	String	dbname_
+0xf0	Pointer	mem_
+0xf8	Pointer	imm_
+0x100	Bool	has_imm_
+0x110	Uint	logfile_number_
+0x1a8	Pointer	versions_
...		

MemTable		
+0x18	Class	arena_
Arena		
+0x28	Vector	blocks_
+0x48	Class	table_
SkipList		
+0x58	Pointer	arena_
+0x68	Uint	max_height_
...		

VersionSet		
+0x08	String	dbname_
+0x40	Uint	next_file_number_
+0x48	Uint	manifest_file_number_
+0x50	Uint	last_sequence_
+0x58	Uint	log_number
+0x60	Uint	prev_log_number
+0x160	Pointer	current_
...		

Fig. 3. Diagram of major classes and fields of validation targets.

A record contains ‘key_length’, ‘key’, ‘sequence number’, ‘type’, ‘value_length’, and ‘value’, where ‘key_length’ and ‘value_length’ are in serialized varint format. The ‘key’ is used to identify and access specific data. The ‘sequence number’ helps to track the record history. The higher the ‘sequence number’, the more recently the record was modified. The ‘type’ represents the status of the key, whether it is live or deleted. The ‘value’ represents the V8 object value.

IndexedDB metadata has keys that are commonly composed of KeyPrefix structure and IDBKey. KeyPrefix uses a reserved number for storing each metadata. The prefix has a variable length, and the very first byte indicates the size of the following values including ‘database id’, ‘object store id’, and ‘index id.’ The value stored in the first three bits corresponds to the size of the ‘database id’ minus 1 and the value stored in the next three bits corresponds to the size of the ‘object store id’ minus 1. The value stored in the last two bits corresponds to the size of the ‘index id’ minus 1.

The key for each record is prefixed with <database id, object store id, index id>, and Table 1 shows KeyPrefix for the essential metadata. The KeyPrefix allow identifying the database, object store, and index to which the actual record belongs, as well as the metadata. IDBKey is an key where the first byte represents the data type (Null, Number, Date, String, Binary, Array) followed by a type-specific serialized value.

The next step interprets the records in IndexedDB that user data remains. The actual record stored in the object store has the KeyPrefix <database id, object store id, and value 1 of index id>. It is necessary to determine which database and object store contain the records with the stored KeyPrefix, and then deserialize the value using the V8 object value serializer. Finally, we need to classify the records according to the database and object store.

Interpreting and classifying the IndexedDB data allows for the extraction of user data stored in Chromium-based applications, which may provide important clues for digital forensic investigations.

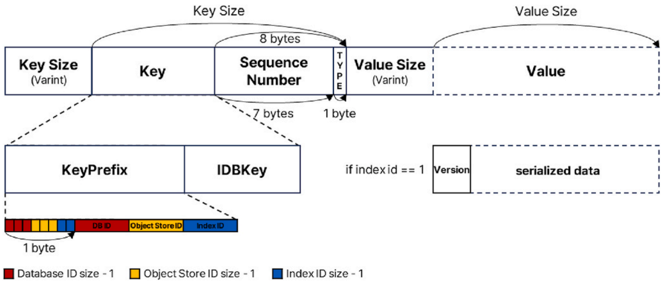


Fig. 4. An IndexedDB record structure.

Table 1
The reserved KeyPrefix for metadata.

KeyPrefix	Description
<0, 0, 0, 0>	backing store version
<0, 0, 0, 1>	maximum allocated database
<0, 0, 0, 201>	origin, database name
<database id, 0, 0, 0>	origin name
<database id, 0, 0, 1>	database name
<database id, 0, 0, 3>	maximum allocated object store id
<database id, 0, 0, 50, object store id, 0>	object store name
<database id, 0, 0, 50, object store id, 5>	maximum allocated index id
<database id, 0, 0, 100, object store id, index id, 0>	index name

3.4. Integration of carved IndexedDB data using a normalized schema

We then reconstruct the records extracted from the memory dump into an organized and structured database. We created three types of tables in a database. Table 2 shows a normalized database schema and their descriptions. The ‘Databases’ table records a list of the extracted databases and object stores. The ‘[DB name 1, 2, ..., N]’ table is named after each name of a database. They are created as many tables as the number of databases. Their schema is structured as Database ID, Database name, Object Store ID, Object Store name, Sequence Number, Key State, Key, Value, and the proposed methodology uses the schema to write the actual record data stored in the object store to which each record belongs. If the records do not have any metadata, they are stored in the ‘Unclassified’ table.

The reconstruction of the raw information extracted from memory dumps into a more structured form plays an important role in data analysis and digital forensic investigations.

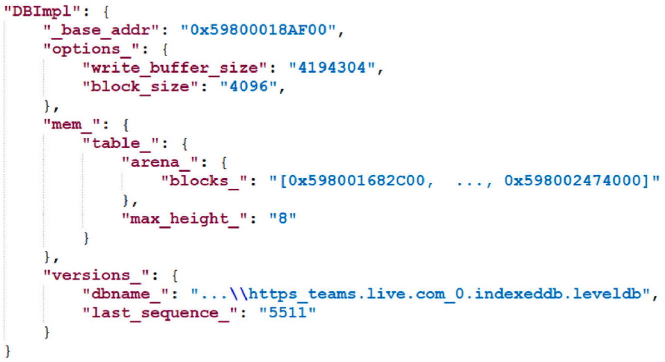
4. Implementation

4.1. Design of MIC

Based on our proposed methodology, we developed a proof-of-concept tool, MIC (Jeong, 2024). We implemented MIC as a proof-of-concept tool consisting of five modules: Validator, Extractor, Deserializer, Classifier and Exporter. The Deserializer module was developed using ccl_chrome_indexeddb (CCL Solutions Group, 2020),

Table 2
A normalized database schema.

Table	Column	Description
Databases	Origin	the source of a app
	Database ID	database id
	Database	database name
	Object Store ID	object store id
	Object Store	object store name
[DBName1, 2, ..., N]	Database ID	database id
	Database	database name
	Object Store ID	object store id
	Object Store	object store name
	Sequence Number	sequence number of record
	Key State	Key state (live or deleted)
	Key	the identifier for a record
	Value	the actual data associated with that key
Unclassfied	Database ID	database id
	Object Store ID	object store id
	Sequence Number	sequence number of record
	Key State	Key state (live or deleted)
	Key	the identifier for a record
	Value	the actual data associated with that key



```
"DBImpl": {
    "_base_addr": "0x59800018AF00",
    "options_": {
        "write_buffer_size": "4194304",
        "block_size": "4096",
    },
    "mem_": {
        "table_": {
            "arena_": {
                "blocks_": "[0x598001682C00,  ..., 0x598002474000]"
            },
            "max_height_": "8"
        }
    },
    "versions_": {
        "dbname_": "...\\https_teams.live.com_0.indexeddb.leveldb",
        "last_sequence_": "5511"
    }
}
```

Fig. 5. The output example of a validated object layout using the MIC tool outputs (JSON).

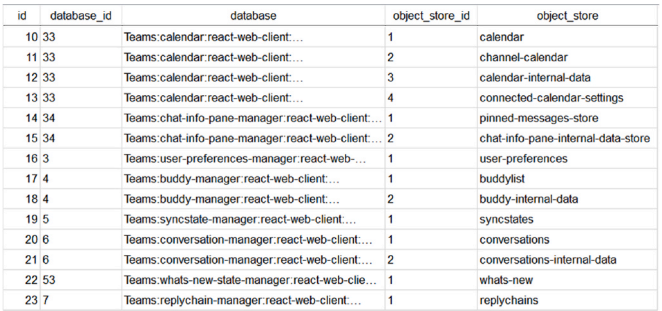
supporting its flexibility and extensibility to be used for the ccl_chrome_indexeddb project as well. We note that the ccl_chrome_indexeddb project only deals with the IndexedDB data stored on local systems, but it now can support analysis on volatile memory using the MIC module.

The Validator module scans the memory dump and validates the candidate class objects. With analysis results of class layout, we verify the identified objects if they match with the expected structure. The Extractor module reconstructs the data blocks to extract the actual records from the candidate set of validated objects, parse and extract all the key-value pairs stored in the SkipList within the reconstructed data blocks. The Deserializer module decodes the key encoded in the structure of IDBKey from the extracted key-value pairs and deserializes the serialized value in the V8 object serialization format. The Classfier module classifies the deserialized key-value pairs into each database and object store, and normalizes the records to be used for the next step. The last module, Exporter, inserts the classified and normalized records into databases.

4.2. Execution and outputs

The MIC tool is run for a given raw memory dump (Windows Minidump format) with traces of the use of Chromium-based applications. The output for this tool has two file formats; JSON and SQLite database. As shown in Fig. 5, the JSON output results include offsets for the validated class objects and values for the class member fields.

Databases, object stores and their metadata, and records are stored in the SQLite database format. Fig. 6 shows the results having SQLite format using the MIC tool.



id	database_id	database	object_store_id	object_store
10	33	Teams:calendar:react-web-client:...	1	calendar
11	33	Teams:calendar:react-web-client:...	2	channel-calendar
12	33	Teams:calendar:react-web-client:...	3	calendar-internal-data
13	33	Teams:calendar:react-web-client:...	4	connected-calendar-settings
14	34	Teams:chat-info-pane-manager:react-web-client:...	1	pinned-messages-store
15	34	Teams:chat-info-pane-manager:react-web-client:...	2	chat-info-pane-internal-data-store
16	3	Teams:user-preferences-manager:react-web-...	1	user-preferences
17	4	Teams:buddy-manager:react-web-client:...	1	buddylist
18	4	Teams:buddy-manager:react-web-client:...	2	buddy-internal-data
19	5	Teams:syncstate-manager:react-web-client:...	1	syncstates
20	6	Teams:conversation-manager:react-web-client:...	1	conversations
21	6	Teams:conversation-manager:react-web-client:...	2	conversations-internal-data
22	53	Teams:whats-new-state-manager:react-web-clie...	1	whats-new
23	7	Teams:replychain-manager:react-web-client:...	1	replychains

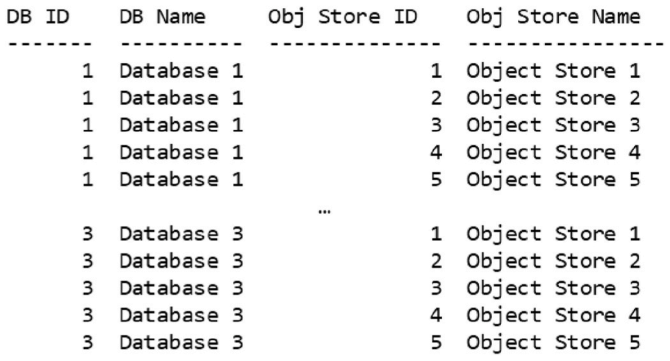
Fig. 6. The output example of an IndexedDB's metadata using the MIC tool (SQLite).

Table 3
Applications used for the experiments.

Type	Application Name	Version
Browser	Google Chrome	125.0.6422.114
	Microsoft Edge	125.0.2535.7931
Desktop Application	Microsoft Teams	24102.2309.2851.4917

Table 4
Defined user activities for Chromium-based services.

Exp #	App (Service Name)	Activities
1	Google Chrome (Custom site)	1. Start normal browsing mode 2. Create 5 Database and 25 Object Stores (1 Database per 5 Object Stores)
2	Google Chrome (Custom site)	1. Insert 100 messages into "Database 5, Object Store 1" 2. Modify sent messages (1–5, 51–55, 86–90) from "Database 5, Object Store 1"
3	Microsoft Edge (Telegram)	1. Start incognito browsing mode 2. Visit and login https://web.telegram.org 3. Browse chats(2), dialogs(5), users(4) in Telegram
4	Desktop App (Microsoft Teams)	1. Run and login Microsoft Teams 2. Send 8 messages to "User"



```
DB ID    DB Name      Obj Store ID    Obj Store Name
-------  ----------   -------------   ----------------
      1  Database 1               1   Object Store 1
      1  Database 1               2   Object Store 2
      1  Database 1               3   Object Store 3
      1  Database 1               4   Object Store 4
      1  Database 1               5   Object Store 5
                       ...
      3  Database 3               1   Object Store 1
      3  Database 3               2   Object Store 2
      3  Database 3               3   Object Store 3
      3  Database 3               4   Object Store 4
      3  Database 3               5   Object Store 5
```

Fig. 7. MIC Result: result of the experiment #1.

5. Experiments and results

5.1. Experimental design

In this work, the experiments were performed with a virtual machine with Windows 11 (VERSION: 23H2, RAM: 16 GB). MIC then performed some activities using browsers and desktop applications built with Chromium-Projects as shown in Table 3. Our tool uses Process Hacker v2.39 (Winsider Seminars & Solutions, Inc, 2016) to collect memory dumps.

In order to validate the proposed approach, we designed the experiments as follows: 1) Extract metadata of IndexedDB such as databases, object stores, and their names. 2) Extract all available records (normal, deleted, and modified) for each IndexedDB 3) Confirm whether all the records are identifiable in incognito mode for Chromium-based browsers 4) Confirm whether the records are identifiable for Chromium-based desktop applications.

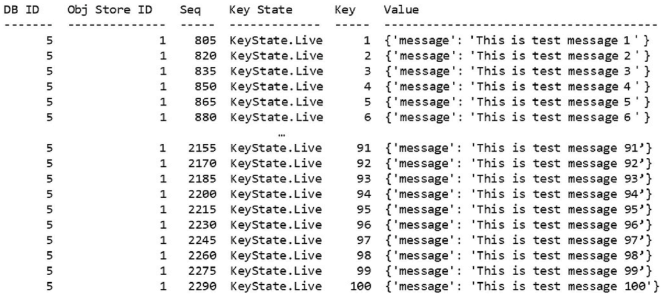
During this study, four experiments were designed as follows. Table 4 illustrates applications, services and defined scenarios used for the experiments.

5.2. Results and findings

In this section, we describe results for our experiments.

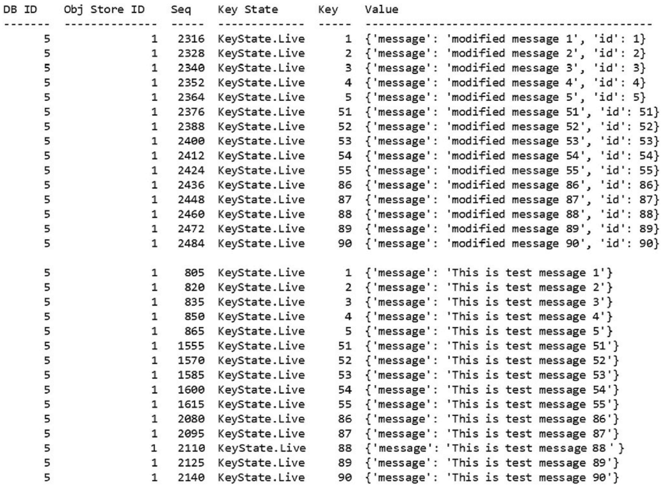
5.2.1. IndexedDB metadata

MemTable stores metadata of IndexedDB in the form of a record. First of all, the method identifies the records storing metadata among the records in MemTable. As metadata has a reserved key prefix for each database and object store. We could recognize the metadata of database



```
DB ID    Obj Store ID    Seq    Key State       Key    Value
-------  -------------  -----  -------------  -----  ---------------------------------------
      5              1    805  KeyState.Live      1  {'message': 'This is test message 1 ' }
      5              1    820  KeyState.Live      2  {'message': 'This is test message 2 ' }
      5              1    835  KeyState.Live      3  {'message': 'This is test message 3 ' }
      5              1    850  KeyState.Live      4  {'message': 'This is test message 4 ' }
      5              1    865  KeyState.Live      5  {'message': 'This is test message 5 ' }
      5              1    880  KeyState.Live      6  {'message': 'This is test message 6 ' }
                                    …
      5              1   2155  KeyState.Live     91  {'message': 'This is test message 91’}
      5              1   2170  KeyState.Live     92  {'message': 'This is test message 92’}
      5              1   2185  KeyState.Live     93  {'message': 'This is test message 93’}
      5              1   2200  KeyState.Live     94  {'message': 'This is test message 94’}
      5              1   2215  KeyState.Live     95  {'message': 'This is test message 95’}
      5              1   2230  KeyState.Live     96  {'message': 'This is test message 96’}
      5              1   2245  KeyState.Live     97  {'message': 'This is test message 97’}
      5              1   2260  KeyState.Live     98  {'message': 'This is test message 98’}
      5              1   2275  KeyState.Live     99  {'message': 'This is test message 99’}
      5              1   2290  KeyState.Live    100  {'message': 'This is test message 100'}
```

Fig. 8. MIC Result: result of the experiment #2.



```
DB ID    Obj Store ID    Seq    Key State       Key    Value
-------  -------------  -----  -------------  -----  ---------------------------------------
      5              1   2316  KeyState.Live      1  {'message': 'modified message 1', 'id': 1}
      5              1   2328  KeyState.Live      2  {'message': 'modified message 2', 'id': 2}
      5              1   2340  KeyState.Live      3  {'message': 'modified message 3', 'id': 3}
      5              1   2352  KeyState.Live      4  {'message': 'modified message 4', 'id': 4}
      5              1   2364  KeyState.Live      5  {'message': 'modified message 5', 'id': 5}
      5              1   2376  KeyState.Live     51  {'message': 'modified message 51', 'id': 51}
      5              1   2388  KeyState.Live     52  {'message': 'modified message 52', 'id': 52}
      5              1   2400  KeyState.Live     53  {'message': 'modified message 53', 'id': 53}
      5              1   2412  KeyState.Live     54  {'message': 'modified message 54', 'id': 54}
      5              1   2424  KeyState.Live     55  {'message': 'modified message 55', 'id': 55}
      5              1   2436  KeyState.Live     86  {'message': 'modified message 86', 'id': 86}
      5              1   2448  KeyState.Live     87  {'message': 'modified message 87', 'id': 87}
      5              1   2460  KeyState.Live     88  {'message': 'modified message 88', 'id': 88}
      5              1   2472  KeyState.Live     89  {'message': 'modified message 89', 'id': 89}
      5              1   2484  KeyState.Live     90  {'message': 'modified message 90', 'id': 90}

      5              1    805  KeyState.Live      1  {'message': 'This is test message 1'}
      5              1    820  KeyState.Live      2  {'message': 'This is test message 2'}
      5              1    835  KeyState.Live      3  {'message': 'This is test message 3'}
      5              1    850  KeyState.Live      4  {'message': 'This is test message 4'}
      5              1    865  KeyState.Live      5  {'message': 'This is test message 5'}
      5              1   1555  KeyState.Live     51  {'message': 'This is test message 51'}
      5              1   1570  KeyState.Live     52  {'message': 'This is test message 52'}
      5              1   1585  KeyState.Live     53  {'message': 'This is test message 53'}
      5              1   1600  KeyState.Live     54  {'message': 'This is test message 54'}
      5              1   1615  KeyState.Live     55  {'message': 'This is test message 55'}
      5              1   2080  KeyState.Live     86  {'message': 'This is test message 86'}
      5              1   2095  KeyState.Live     87  {'message': 'This is test message 87'}
      5              1   2110  KeyState.Live     88  {'message': 'This is test message 88 '}
      5              1   2125  KeyState.Live     89  {'message': 'This is test message 89'}
      5              1   2140  KeyState.Live     90  {'message': 'This is test message 90'}
```

Fig. 9. MIC Result: result of the experiment #3.

and object store with the reserved key prefix. Fig. 7 shows the results coming from MIC for the databases and metadata of object stores created during Experiment 1. A total of five databases were created, each with five object stores. As a result, our tool, MIC, successfully extracted five databases and twenty-five object stores.

The LevelDB records contain their database id and the object store id in their key prefix to which the records belong to. In other words, identifying metadata records indicates a database id and an object store id where they belong to.

5.2.2. IndexedDB object store data

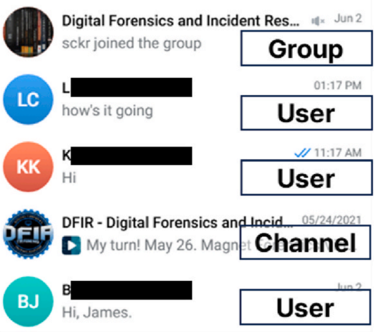
As mentioned above, object stores contain data such as database id and object store id. Therefore, this study classified all the extracted records by database id and object store id. Experiment 2 performed inserting 100 test messages into object store having ‘ID 1’ and database having ‘ID 5.’ Fig. 8 shows the list of records extracted using MIC, one hundred records were extracted from one hundred records. the result was same under the incognito mode.

Then we modified 15 messages (1–5, 51–55, 86–90) out of the 100 messages and collected the memory dump. Fig. 9 shows the results using our developed tool. LevelDB records assume that the record with the highest sequence number is the most recent record when they have the same key. Therefore, we confirmed that both the modified messages and the original messages remain in the volatile memory and MIC extracts them thoroughly.

5.2.3. IndexedDB records in Chromium-based browsers incognito mode

Google Chrome and Microsoft Edge, two of the most popular browsers built on the Chromium, offer incognito mode. This mode does not store data in local disk, but only in volatile memory.

The third experiment analyzed whether IndexedDB-related records are identifiable when using the incognito mode of the Microsoft Edge browser. We logged into Telegram using the incognito mode of the web browser and explored the list of ‘channels’, ‘dialogues’, and ‘contacts.’ Fig. 10 is the user interface accessing Telegram using the browser in incognito mode. The user in this experiment was involved in one channel, one group and had conversations with three other users.



Digital Forensics and Incident Res... Jun 2
sckr joined the group

L[redacted] 01:17 PM
how's it going

K[redacted] 11:17 AM
Hi

DFIR - Digital Forensics and Incid... 05/24/2021
My turn! May 26. Magn

B[redacted] Jun 2
Hi, James.

Fig. 10. Chats tab in web version telegram in Microsoft Edge Browser Incognito mode.

MIC identified and extracted one ‘channel’, two ‘dialogues’, and three ‘users.’ As shown in Fig. 11 (a), MIC extracts user data from the memory dump even when using the service in incognito mode. The approach supports the investigation of user behavior that uses incognito mode to not store the data on local disk.

5.2.4. IndexedDB records in Chromium-based application

Microsoft Teams is one of the well-known desktop applications built with Microsoft Edge WebView2. As it stores user data in IndexedDB, we chose it one of the experimental subject.

We sent a total of 8 messages using Microsoft Teams to evaluate whether MIC successfully extracts user data or not. MIC extracted 7 out of the 8 messages we sent.

Fig. 12 shows the results of MIC extraction of Microsoft Teams messages. The results include message data such as content, display name, from, to, timestamp, and type, which can be useful for the digital forensic investigations. Fig. 11 (b) shows the list of messages extracted using MIC. We confirmed that the data for Microsoft Teams remains in volatile memory, and our approach can be applied to Chromium-based desktop applications.

5.2.5. Other findings

MIC also identifies LevelDB database that are used to maintain and manage Chromium-based applications. Unlike IndexedDB data, LevelDB stores data in key-value pairs, without serialization. Fig. 13 illustrates the partial records that GCM Store of Chrome browser use.

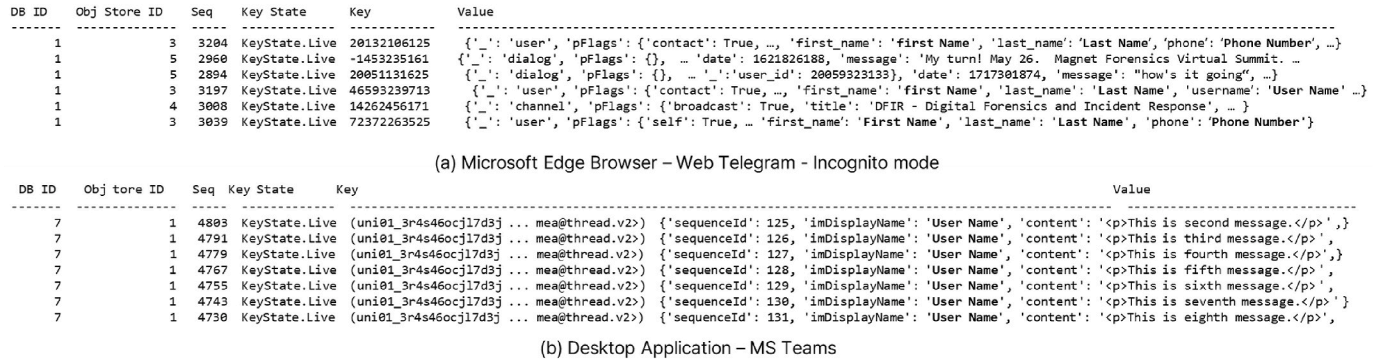
Chromium-based applications store data in LevelDB format using the IndexedDB API. Considering the frequent update cycle of IndexedDB, we noticed that the proposed approach could be outdated in a short time. Therefore, we focused on identifying class objects that are relatively stable rather than the wrapper class having fast updates in LevelDB. This approach allowed us to capture the LevelDB-related data while the applications were running, as well as the IndexedDB data.

5.3. Dataset and performance evaluation

We created test data sets (Jeong, 2024) during the experiments and evaluated the time taken to analyze the IndexedDB and its performance on Intel i9-10900X CPU and 256 GB RAM. Table 5 shows the accuracy of detecting IndexedDB related objects and the time spent for the analysis.

6. Conclusions and future directions

Recently, Chromium-based applications have been widely used and



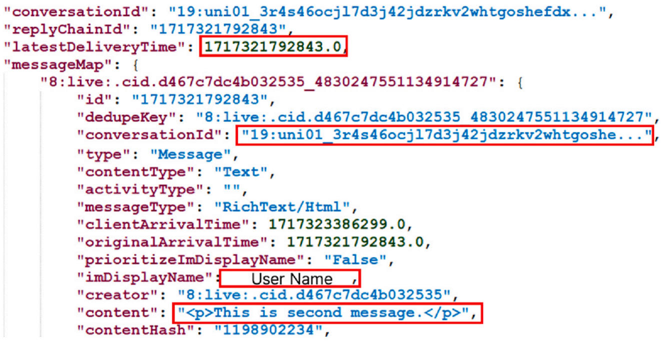
```
DB ID   Obj Store ID   Seq   Key State       Key           Value
-------  -------------- ----- ------------- -----------   ------------------------------------------------------------------------------------------------------------
      1              3  3204  KeyState.Live  20132106125   {'_': 'user', 'pFlags': {'contact': True, …, 'first_name': 'first Name', 'last_name': 'Last Name', 'phone': 'Phone Number', …}
      1              5  2960  KeyState.Live  -1453235161  {'_': 'dialog', 'pFlags': {},  … 'date': 1621826188, 'message': 'My turn! May 26.  Magnet Forensics Virtual Summit. …
      1              5  2894  KeyState.Live  20051131625   {'_': 'dialog', 'pFlags': {},  … '_':'user_id': 20059323133}, 'date': 1717301874, 'message': "how's it going", …}
      1              3  3197  KeyState.Live  46593239713    {'_': 'user', 'pFlags': {'contact': True, …, 'first_name': 'first Name', 'last_name': 'Last Name', 'username': 'User Name' …}
      1              4  3008  KeyState.Live  14262456171   {'_': 'channel', 'pFlags': {'broadcast': True, 'title': 'DFIR - Digital Forensics and Incident Response', … }
      1              3  3039  KeyState.Live  72372263525   {'_': 'user', 'pFlags': {'self': True, … 'first_name': 'First Name', 'last_name': 'Last Name', 'phone': 'Phone Number'}
```

(a) Microsoft Edge Browser – Web Telegram - Incognito mode

```
 DB ID   Obj tore ID    Seq  Key State       Key                                                               Value
-------  -------------- ----- ------------- -----------------------------------------------------------------   ------------------------------
      7              1  4803  KeyState.Live  (uni01_3r4s46ocjl7d3j ... mea@thread.v2>)  {'sequenceId': 125, 'imDisplayName': 'User Name', 'content': '<p>This is second message.</p>' ,}
      7              1  4791  KeyState.Live  (uni01_3r4s46ocjl7d3j ... mea@thread.v2>)  {'sequenceId': 126, 'imDisplayName': 'User Name', 'content': '<p>This is third message.</p>' ,
      7              1  4779  KeyState.Live  (uni01_3r4s46ocjl7d3j ... mea@thread.v2>)  {'sequenceId': 127, 'imDisplayName': 'User Name', 'content': '<p>This is fourth message.</p>',}
      7              1  4767  KeyState.Live  (uni01_3r4s46ocjl7d3j ... mea@thread.v2>)  {'sequenceId': 128, 'imDisplayName': 'User Name', 'content': '<p>This is fifth message.</p>' ,
      7              1  4755  KeyState.Live  (uni01_3r4s46ocjl7d3j ... mea@thread.v2>)  {'sequenceId': 129, 'imDisplayName': 'User Name', 'content': '<p>This is sixth message.</p>' ,
      7              1  4743  KeyState.Live  (uni01_3r4s46ocjl7d3j ... mea@thread.v2>)  {'sequenceId': 130, 'imDisplayName': 'User Name', 'content': '<p>This is seventh message.</p>' }
      7              1  4730  KeyState.Live  (uni01_3r4s46ocjl7d3j ... mea@thread.v2>)  {'sequenceId': 131, 'imDisplayName': 'User Name', 'content': '<p>This is eighth message.</p>',
```

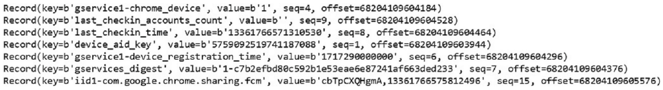
(b) Desktop Application – MS Teams

Fig. 11. MIC Result: (a) Extracting Telegram data using browser in incognito mode (b) Extracting MS Teams data using desktop app.



```
"conversationId": "19:uni01_3r4s46ocjl7d3j42jdzrkv2whtgoshefdx...",
"replyChainId": "1717321792843",
"latestDeliveryTime": 1717321792843.0,
"messageMap": {
    "8:live:.cid.d467c7dc4b032535_4830247551134914727": {
        "id": "1717321792843",
        "dedupeKey": "8:live:.cid.d467c7dc4b032535 4830247551134914727",
        "conversationId": "19:uni01_3r4s46ocjl7d3j42jdzrkv2whtgoshe...",
        "type": "Message",
        "contentType": "Text",
        "activityType": "",
        "messageType": "RichText/Html",
        "clientArrivalTime": 1717323386299.0,
        "originalArrivalTime": 1717321792843.0,
        "prioritizeImDisplayName": "False",
        "imDisplayName":   User Name   ,
        "creator": "8:live:.cid.d467c7dc4b032535",
        "content": "<p>This is second message.</p>",
        "contentHash": "1198902234",
```

Fig. 12. MIC Result: Extracting messages for MS Teams.



```
Record(key=b'gservice1-chrome_device', value=b'1', seq=4, offset=68204109604184)
Record(key=b'last_checkin_accounts_count', value=b'', seq=9, offset=68204109604528)
Record(key=b'last_checkin_time', value=b'13361766571310530', seq=8, offset=68204109604464)
Record(key=b'device_aid_key', value=b'5759092519741187088', seq=1, offset=68204109603944)
Record(key=b'gservice1-device_registration_time', value=b'1717290000000', seq=6, offset=68204109604296)
Record(key=b'gservices_digest', value=b'1-c7b2efbd80c592b1e53eae6e87241af663ded233', seq=7, offset=68204109604376)
Record(key=b'iid1-com.google.chrome.sharing.fcm', value=b'cbTpCXQHgmA,13361766575812496', seq=15, offset=68204109605576)
```

Fig. 13. MIC Result: Other findings.

developed recently for various online services. In particular, these applications utilize the IndexedDB to manage and handle their data. In this paper, we propose a methodology to find forensically relevant data from memory dumps. Chromium-based browsers support incognito mode to enhance privacy. In this mode, they do not store data on the local disk, so investigating volatile memory is essential.

Our methodology focused on understanding the mechanisms by which IndexedDB manipulates data to extract forensically relevant data from memory dumps. With the proposed method, metadata such as databases, object stores and their names from IndexedDB are extracted successfully. The metadata provides important insights into the structures of an IndexedDB. Furthermore, our approach successfully extracts all records stored in IndexedDB, including normal, deleted, and modified records. The extraction allows for an in-depth analysis of the data stored within the IndexedDB database. We extend our investigation to determine if our methodology is also applicable to desktop applications developed using Chromium-based frameworks.

Table 5
Evaluation results using the MIC tool with a high accuracy.

Exp #	Data Type	Accuracy	Elapsed Time
1	Metadata (Database)	5/5	22 s
	Metadata (Object Store)	25/25	
2	Normal records	100/100	22 s
	Modified records	15/15	
3	Chats	1/2	81 s
	Dialogs	2/5	
	Users	3/4	
4	Messages	7/8	26 s

This broad scope of research ensures that our methodology is versatile and compatible with many different kinds of Chromium-based applications. We conducted experiments on the services Google Chrome, Microsoft Edge, and Microsoft Teams, demonstrating that our approach can successfully acquire user data. We expect that our methodology can be effectively used for digital forensic investigations in the future. Based on our findings, we developed a proof-of-concept tool based on our approach and evaluated its performance and effectiveness through several experiments.

However, our methodology has limitations. When running the browser in incognito mode, LevelDB manages partial data in BLOB, but the proposed method does not address this issue. In addition, we have an interest in all the available artifacts related to IndexedDB. Although this study mainly focused on extracting and comprehending data in memory, we plan to expand our research on analyzing and integrating IndexedDB acquired from various digital sources.

Acknowledgements

This work was supported by Police-Lab 2.0 Program(www.kipot.or.kr) funded by the Ministry of Science and ICT (MSIT, Korea) & Korean National Police Agency(KNPA, Korea) [Project Name: Research on Data Acquisition and Analysis for Counter Anti-Forensics/Project Number: 210121M07].

References

CCL Solutions Group, 2020. ccl_chromium_reader. https://github.com/cclgroupltd/ccl_chrome_indexeddb, 2024-06-01.

Choi, G., Bang, J., Lee, S., Park, J., 2023. Chracer: memory analysis of Chromium-based browsers. Forensic Sci. Int.: Digit. Invest. 46, 301613 https://doi.org/10.1016/j.fsidi.2023.301613. URL: https://www.sciencedirect.com/science/article/pii/S2666281723001257.

Cook, T.D., Campbell, D.T., 1979. The design and conduct of true experiments and quasi-experiments in field settings. In: Reproduced in Part in Research in Organizations: Issues and Controversies. Goodyear Publishing Company.

Google, 2008. The chromium projects. https://www.chromium.org/chromium-projects/, 2024-06-01.

Google, 2011a. LevelDB. https://github.com/google/leveldb, 2024-06-03.

Google, 2011b. LevelDB LDB format. https://github.com/google/leveldb/blob/main/doc/table_format.md, 2024-06-03.

Google, 2011c. LevelDB log format. https://github.com/google/leveldb/blob/main/doc/log_format.md, 2024-06-03.

Google, 2021. LevelDB latest version. https://github.com/google/leveldb/releases, 2024-07-14.

Hariharan, M., Thakar, A., Sharma, P., 2022. Forensic analysis of private mode browsing artifacts in portable web browsers using memory forensics. In: 2022 International Conference on Computing, Communication, Security and Intelligent Systems (IC3SIS), pp. 1–5. https://doi.org/10.1109/IC3SIS54991.2022.9885379.

Jeong, B., 2024. Mic. https://github.com/naaya17/MIC, 2024-06-01.

Kim, D., Lee, S., Park, J., 2024. Decrypting IndexedDB in private mode of Gecko-based browsers. Forensic Sci. Int.: Digit. Invest. 49, 301763 https://doi.org/10.1016/j.fsidi.2024.301763. https://www.sciencedirect.com/science/article/pii/S2666281724000829.

Mahlous, A., Mahlous, H., 2020. Private browsing forensic analysis: a case study of privacy preservation in the Brave browser. International Journal of Intelligent

Engineering and Systems 13, 294–306. https://doi.org/10.22266/ijies2020.1231.26.

Microsoft, 2020. Microsoft Edge WebView2. https://learn.microsoft.com/en-us/microsoft-edge/webview2/, 2024-06-03.

Nelson, R., Shukla, A., Smith, C., 2020. Web Browser Forensics in Google Chrome, Mozilla Firefox, and the Tor Browser Bundle. Springer International Publishing, Cham, pp. 219–241. https://doi.org/10.1007/978-3-030-23547-5_12.

OpenJS Foundation, 2014. Electron. https://electronjs.org, 2024-06-03.

Paligu, F., Kumar, A., Cho, H., Varol, C., 2019. BrowStExPlus: a tool to aggregate IndexedDB artifacts for forensic analysis. J. Forensic Sci. 64, 1370–1378. https://doi.org/10.1111/1556-4029.14043. https://onlinelibrary.wiley.com/doi/abs/10.1111/1556-4029.14043.

Paligu, F., Varol, C., 2020. Browser forensic investigations of WhatsApp web utilizing IndexedDB persistent storage. Future Internet 12. https://doi.org/10.3390/fi12110184. https://www.mdpi.com/1999-5903/12/11/184.

Paligu, F., Varol, C., 2022a. Browser forensic investigations of Instagram utilizing IndexedDB persistent storage. Future Internet 14. https://doi.org/10.3390/fi14060188. https://www.mdpi.com/1999-5903/14/6/188.

Paligu, F., Varol, C., 2022b. Microsoft Teams desktop application forensic investigations utilizing IndexedDB storage. J. Forensic Sci. 67, 1513–1533. https://doi.org/10.1111/1556-4029.15014. https://onlinelibrary.wiley.com/doi/abs/10.1111/1556-4029.15014.

Pugh, W., 1990. Skip lists: a probabilistic alternative to balanced trees. Commun. ACM 33, 668–676.

Saputra, R., Riadi, I., 2020. Forensic browser of Twitter based on web services. Int. J. Comput. Appl. 975, 8887.

Satvat, K., Forshaw, M., Hao, F., Toreini, E., 2014. On the privacy of private browsing–a forensic approach. J. Inf. Secur. Appl. 19, 88–100.

Thomas, T., Piscitelli, M., Shavrov, I., Baggili, I., 2020. Memory FORESHADOW: memory forensics of hardware cryptocurrency wallets – a tool and visualization framework. Forensic Sci. Int.: Digit. Invest. 33, 301002 https://doi.org/10.1016/j.fsidi.2020.301002. https://www.sciencedirect.com/science/article/pii/S2666281720302511.

Van Der Horst, L., Choo, K.K.R., Le-Khac, N.A., 2017. Process memory investigation of the Bitcoin clients Electrum and Bitcoin Core. IEEE Access 5, 22385–22398.

Wang, E., Zurowski, S., Duffy, O., Thomas, T., Baggili, I., 2022. Juicing V8: a primary account for the memory forensics of the V8 Javascript engine. Forensic Sci. Int.: Digit. Invest. 42, 301400 https://doi.org/10.1016/j.fsidi.2022.301400. https://www.sciencedirect.com/science/article/pii/S2666281722000816.

Winsider Seminars & Solutions, Inc, 2016. Process hacker. In: https://processhacker.sourceforge.io, 2024-06-01.

Zollner, S., Choo, K.K.R., Le-Khac, N.A., 2019. An automated live forensic and postmortem analysis tool for Bitcoin on Windows systems. IEEE Access 7, 158250–158263. https://doi.org/10.1109/ACCESS.2019.2948774.