Analyzing "Not-a-Virus" Bundled Adware: The Wajam Case*

Xavier de Carné de Carnavalet and Mohammad Mannan
Concordia University, Montreal, QC, Canada
{x_decarn, mmannan}@ciise.concordia.ca

***Abstract*—Case studies on malicious code mostly focus on botnets and worms (recently revived with IoT devices), prominent pieces of malware or Advanced Persistent Threats, exploit kits, ransomware, yet very little has been done on adware. Previous studies on "unwanted" applications, including adware, favored breadth of analysis, uncovering ties between different actors and distribution methods. We investigate the evolution over nearly six years of a particularly successful and active adware business: *Wajam*. As of 2016, revealed by the Office of the Privacy Commissioner of Canada, Wajam had "hundreds of millions of installations" and collected 400TB of private information from users.**

We gather 52 samples of Wajam, released between 2013 to 2018, and analyze the technical evolution from a simple browser add-on to full-fledged obfuscated malware including rootkit, browser process injection, and antivirus evasion capabilities. We uncover its strategy to ensure a low detection rate, which heavily relies on numerous layers of encryption, and more recently on steganography. Furthermore, Wajam leaks the browsing histories of four major browsers, along with the keywords searched by users on highly popular websites. It is also vulnerable to arbitrary content injection on HTTPS webpages, and likely to remote code execution. We show evidence that Wajam is a widespread threat, actively maintained with daily obfuscated samples that are poorly detected by antivirus engines. More worrisome, we found the same evasion techniques in another piece of adware, suggesting that they could be provided by a third-party, and reused in other cases. Finally, we conclude that the adware problem has been overlooked for too long, which can reach (or even surplus) the complexity of advanced malware, and pose both privacy and security risks to users, more so than many well-known and thoroughly-analyzed malware families.

I. Introduction

The business of generating revenue through ads is becoming very intrusive for end users. Popular application download websites are known to bundle adware with their custom installers [24], [21]. Users can also be misled to install Potentially Unwanted Programs (PUPs) that provide limited or deceptive services (e.g., toolbars, cleanup utilities) along with invasive ads [49], [62]. The prevalence of adware is also increasing. Recent studies show that Google Safe Browsing triggers 60 million warnings per week for pay-per-install bundled installers, twice the rate of malware-related warnings [31], [62].

However, adware applications are generally not considered as much of a threat as malware, as apparent from some antivirus labels, e.g., "not-a-virus", "Riskware", "Unwanted-Program", "PUP.Optional", which may not even trigger an alert [20], [28]. After all, displaying ads is not considered a *malicious* activity. Consequently, adware has received less scrutiny from the malware research community. Indeed, studies on PUPs focus only on the revenues, distribution and relationships between actors [31], [62]. One exception is Kotzias et al. [32] who looked into the abuse of code signing and found that PUPs leverage digital signatures more often than malware, enabling PUPs to appear more legitimate.

Malware analysis has a long history in the academia, see e.g., the Morris Worm report from 1989 [56]. Past malware case studies focused on regular botnets [58], IoT botnets [7], prominent malware [50], [11], web exploit kits [27], [36], Advanced Persistent Threats [59], [38], and ransomware [29]. However, while several malware authors have been identified and arrested [23], [14], resulting in some reduction of exploit kits (at least temporarily, see, e.g., [57]), adware campaigns remain unscathed. Previous cases of ad-related products received media attention as they severely downgrade HTTPS security [1], [2], but they generally do not adopt techniques from malware (e.g., obfuscation and evasion). Consequently, security companies may prioritize their effort on malware, while academic researchers may consider adware as a non-problem, or simply a technically uninteresting one, enabling adware to survive and thrive for long periods of time. Important questions remain unexplored about adware, including: 1) Are they all simply displaying advertisements? 2) Are all strains limited in complexity and reliably detected by AVs?

We explore the case of *Wajam*, a seven-year old *social search engine* that progressively turned into sophisticated adware, originally developed by a Canadian company. We initially observed TLS certificates from some user machines with seemingly random issuer names, e.g., `b02669b9042c6a8f`. Some of those indicated an email address that led us to Wajam. We then proceeded with the gathering of 52 samples dated from 2013 to 2018. Historical samples are challenging to obtain, since Wajam is often dynamically downloaded by other software installers, and relies either on generic or randomized filenames and root certificates, limiting the number of searchable fingerprints.

Wajam probably would not subsist for seven years if it did not impact many users, and in turn generate enough revenue. To this end, we tracked 248 domain names that were ever used by Wajam, as found in code signing certificates, hardcoded URLs in samples, ad injection rules we downloaded, other domains that were hosted simultaneously from the same IP address, and those declared in legal documents of the company.

* Version: December 12, 2018.

We found some of these domains with a rank as high as #117,351 in Alexa top-1M and #93,915 in Umbrella top-1M, as of June 2018.[1] Wajam's domains are queried when ads are injected into webpages, suggesting that a substantial number of users are infected. Indeed, during an investigation by the Office of the Privacy Commissioner (OPC) of Canada in 2016 [42], the company behind Wajam reported to OPC that it had made "hundreds of millions of installations" and collected "approximately 400 terabytes" of user personal information.

We study the technical evolution of content injection and identify four major generations, including browser add-on, proxy settings changer, browser process injector, and system-wide traffic interceptor. Browser process injection involves hooking into a browser to modify the traffic after it is decrypted and before it is rendered, enabling man-in-the-browser (MITB) attacks. This technique is previously unreported in the adware realm and is known to be last used by the Zeus malware for stealing banking information [5], [25].

Across generations, Wajam increasingly makes use of several anti-analysis and evasion techniques including: a) nested installers, b) steganography, c) string and library call obfuscation, d) encrypted strings and files, e) deep and diversified dead code, f) polymorphic resources, g) valid digital signatures, h) randomized filenames and root certificate Common Names, i) encrypted updates, and j) daily release of polymorphic variants. It also implements anti-detection features ranging from disabling Windows Malicious Software Removal Tool (MRT), self-excluding its installation paths from Windows Defender, and sometimes leveraging rootkit capabilities to hide its installation folder from users. We detail these techniques, which are still effective as of Nov. 2018 to lower its detection rate, reaching as low as 3/68 AV engines on VirusTotal for a 19-min old variant and only a few AVs after a day.

More importantly, we discovered a separate piece of adware, *OtherSearch*, that reuses the same model and some of the same techniques as Wajam, sometimes more advanced. This may hint at a common third-party who provides an obfuscation framework to both businesses and perhaps others as well. Our analysis here is focused on Wajam only due to the abundance of samples we could collect.

We also found security flaws that have exposed (possibly) millions of users for the last four years to potential arbitrary content injection, man-in-the-middle (MITM) attacks, and remote code execution (RCE). As the third generation of Wajam leverages browser process injection, the injected content is present in the webpage *without* its HTTPS certificate being changed, preventing even a mindful user from detecting the tampering. In addition, Wajam systematically downgrades the security of a number of websites by removing their Content Security Policy (CSP), e.g., *facebook.com*, and other security-related HTTP headers from the server's response. Further, Wajam sends—*in plaintext*—the browsing histories from four major browsers (if installed), and the list of installed programs, to Wajam's operators. Finally, search keywords input on 100 groups of domains spanning millions of websites are also sent out. Hence, Wajam remains as a major privacy and security threat to millions of users.

While the existence of traffic-injecting malware is known, and TLS flaws are reminiscent of Superfish and Privdog [1], [2], Wajam is significantly more sophisticated. As anti-analysis techniques get more advanced and innovative over time, they become increasingly difficult to study. We even failed to fully reverse-engineer all layers of few latest samples within reasonable time. Considering Wajam's complexity and automation of evasion techniques, we argue that the adware problem has long been overlooked, and mandates more serious analysis effort.

Contributions.

1) We collect and reverse-engineer 52 unique samples of Wajam spanning across six years and identify four content injection techniques, one of which was previously used in a well-known banking trojan. This analysis is a significant reverse-engineering effort to characterize the technical and design evolution of a successful ad injector. This study is the first to investigate the chronological *evolution* for such an application over six years, shedding light on the practices, history and techniques used by such software.
2) We uncover the serious level of complexity used in Wajam across generations. These 52 samples used various combinations of effective anti-analysis and rootkit-like features, which are even rarely found in a single piece of advanced malware. Such adware samples are generally much less analyzed than malware by academic and industrial researchers. Our revelations on Wajam call for more concentrated reverse engineering efforts towards adware.
3) We also highlight serious privacy (e.g., private information leakage) and security risks (e.g., enabling MITM and possibly RCE attacks) to users affected by Wajam. As new Wajam variants remain largely undetected by malware engines during the first days, users with up-to-date AV software and OS remain vulnerable.

II. Wajam's history

Wajam Internet Technologies Inc. was originally headquartered in Montreal, Canada [45]. Their product (Wajam) aimed at enhancing the search results of a number of websites (e.g., Google, Yahoo, Ask.com, Expedia, Wikipedia, Youtube) with content extracted from a user's social media connections (e.g., Twitter, Facebook, Google+, LinkedIn). Wajam was first released in October 2011, rebranded as Social2Search in May 2016 [42], then as SearchAwesome in August 2017 (as we found). We use the name Wajam interchangeably throughout the paper to refer to the company or the software they developed. To gain revenue, Wajam injects ads into browser traffic [54]. The company progressively lost its connection with social media and became purely adware in 2017.

The Office of the Privacy Commissioner (OPC) of Canada investigated the company between October 2016 and July 2017 [42], based on documents provided by the company, their

[1] Cf. *mcgill.ca* ranks #94,148, and *craigslist.ca* ranks #89,229 in Umbrella (Canadian domains used for the sake of comparison).

own testing of the software, and a two-day visit of the office. OPC found numerous violations of Canadian Personal Information Protection and Electronic Documents Act (PIPEDA), relative to the egregious collection and preservation of personal data ("*approximately 400 terabytes*" by the company's own admission), and problematic user consent/EULA, installation/uninstallation methods. OPC issued a list of 14 recommendations to correct the situation. Instead, Wajam organized the sale of its activities to a newly-created company based in Hong-Kong, called Iron Mountain Technology Limited (IMTL), and therefore declared itself unaccountable to Canadian regulations. IMTL seems to have continued the operations of Wajam uninterrupted since then and continued to develop its capabilities towards ads injection and antivirus evasion.

III. Related work

Previous studies on worms and botnets mostly focused on the network aspect of such threats, instead of particular software complexity or advanced obfuscation techniques; see e.g., Conficker [50], Torpig [58] and Mirai [7]. While the largest known botnet reached up to an estimated 50 million users [61], it is still an order of magnitude less than the total distribution of Wajam ("*hundreds of millions*").

The Mirai botnet was studied from a software standpoint across a thousand samples [7]. Authors tracked forks of the original malware and their newly added features, including e.g., self-deleting binary, more hardcoded passwords to infect devices; however, such changes are minor and not difficult to understand. Moreover, while the number of samples is larger than in our study, the Mirai malware's source code was leaked and readily available (in contrast to Wajam). In our case, the reverse-engineering of Wajam was done from scratch to understand the full extent of its capabilities, and bridge significant gaps across generations and major updates, including dealing with e.g., steganography-based installers, custom packers and multiple layers of encryption.

The Zeus banking malware [25], a prominent strain reaching 3.6 million infections, shares similar traits with Wajam, including encrypted code sections (albeit done differently), dynamic library loading, encrypted payloads (for configuration files only) with XOR or RC4 hardcoded keys. Zeus also achieved MITM by injecting a DLL in browser processes, similar to the third generation of Wajam. However, similar to Mirai later in 2016, Zeus source code went public, which helped understand its behavior. Also, forks of Zeus are still active [9]; yet, they seem to no longer inject DLLs but simply filter network connections to banks. Wajam's browser process injection is still well maintained and effective as of July 2018.

Advanced Persistent Threats (APTs) are known for the extent of their operations, both in time and in terms of their complexity, e.g. [59], [38]. In contrast, our focus is an *adware* application, which is not expected to make use of APT-related techniques e.g., 0-day vulnerabilities. Nevertheless, we found that Wajam leverages effective antivirus evasion techniques, and significantly hinders reverse-engineering, over a long period of time. These behaviors are rare in regular malware.

Similar to adware, but more aggressive, ransomware is also heavily motivated by monetary gains. Previous studies about the inner workings of generations of ransomware focused on their *features* and did not report significant challenges in analyzing them. For example, Kharraz et al. [29] focused on 1,359 samples and report insights into the encryption modules, file replacement and deletion mechanisms, ways to recover or permanently delete data.

Web exploit kits have also been analyzed [27], [36], including PHP and JavaScript components. Similarly, their level of sophistication was limited.

Wajam has been cited in broad analyses covering the distribution models of pay-per-install PUPs [31], [62]; however, only little information about Wajam itself is revealed, including an estimated user base (in the order of hundreds of thousands, significantly less than the figure reported by its operators [42]), and general features (e.g., reporting that Wajam is a browser-addon—incorrect since the end of 2014).

The NetFilter/ProtocolFilters SDKs [51] were used in PrivDog [2], which was vulnerable to MITM attacks as it did not use the certificate validation capabilities of the SDK. Böck [13] extracted the hardcoded private keys from ProtocolFilters found in AdGuard and PrivDog, and listed PUPs that may rely on this library (did not include Wajam). While PrivDog received significant attention, only one version of the product was vulnerable, affecting 57k users [2]. In contrast, Wajam has exposed millions of users to similar MITM attacks for about four years. Compared to SuperFish that was installed by default on certain Lenovo laptops, Wajam is not bound to a specific hardware vendor nor geographic region.

Various obfuscation techniques have been documented and can be found in other malware, e.g., encrypted code section [63], junk code [48], polymorphic icons used by Winwebsec, SecurityShield and zbot [40], inflated executable file size as done by the XXMM toolkit [26], rootkit as found in the Komodia traffic interception SDK [17], the use of NSIS installers with decryption DLLs in Cerber, Gamarue, Kovter and ZCrypt [16], hiding encrypted payload in BMP [10] and PNG files [39]. Wajam combines all these techniques from the malware realm, layers them, and improves on them. Notably, Wajam's junk code introduces thousands of seemingly purposeful functions interconnected in a dense call graph where the real program functions are hidden. Also, the use of steganography is diversified to various file formats, and is combined with layers of obfuscated encryption and compression in samples from 2018.

IV. Sample collection and categorization

We detail below our collection of 52 samples, and provide an overview of their capabilities by grouping them according to their four main content injection techniques. A summary of their notable features (e.g., the use of code-signing, auto/stealthy installation), is provided in Table I. Hashes of the samples are available in Table XI (Appendix).

TABLE I: Samples summary (N/A means not applicable, e.g., expired downloader samples do not install an application)

ID	Installer/downloader/patch filename	Signature	Date (UTC)	Authenticode CN	Installed name	Autoinstall	Opens webpage	Stealthy	Rootkit	Origin
A1	wajam_install.exe	✓	2013-01-03	Wajam	Wajam		✓			Hybrid Analysis
A2	wajam_setup.exe	✓	2014-01-09	Wajam Internet Technologies Inc	Wajam					Hybrid Analysis
A3	wajam_download.exe	✓	2014-05-21	Insta-Download.com	*N/A*	*N/A*	*N/A*	*N/A*		Malekal MalwareDB
A4	wajam_download_v2.exe	✓	2014-07-11	Insta-Download.com	*N/A*	*N/A*	*N/A*	*N/A*		Malekal MalwareDB
B1	WIE_2.15.2.5.exe	✓	2014-09-25	FastFreeInstall.com	Wajam		✓			Malekal MalwareDB
B2	WIE_2.16.1.90.exe	✓	2014-10-03	FastFreeInstall.com	Wajam		✓			Malekal MalwareDB
C1	WWE_1.1.0.48.exe	✓	2014-10-21	AutoDownload.net	Wajam		✓			VirusShare
C2	WWE_1.1.0.51.exe	✓	2014-11-05	AutoDownload.net	Wajam		✓			VirusShare
C3	WWE_1.2.0.31.exe	✓	2014-12-03	AutoDownload.net	Wajam		✓			VirusShare
B3	wajam_setup.exe	✓	2014-12-09	Wajam Internet Technologies Inc	Wajam		✓			Archive.org
C4	WWE_1.2.0.53.exe	✓	2015-01-21	AutoDownload.net	Wajam		✓			VirusShare
C5	wwe_1.43.5.6.exe	✓	2015-04-13	installation-sur-iphone.com	Wajam		✓			Hybrid Analysis
C6	WWE_1.52.5.3.exe	✓	2015-09-17	chabaneltechnology.com	Wajam	✓	✓			Hybrid Analysis
C7	WWE_1.53.5.19.exe	✓	2015-10-16	trudeautechnology.com	Wajam	✓	✓			Hybrid Analysis
B4	WIE_2.38.2.13.exe		2015-10-27	*N/A*	Wajam		✓			Malekal MalwareDB
B5	wie_2.39.2.11.exe		2015-11-05	*N/A*	Wajam		✓			Malekal MalwareDB
C8	wajam_install.exe	✓	2015-11-13	preverttechnology.com	Wajam	✓	✓			Malekal MalwareDB
C9	WWE_1.55.1.20.exe	✓	2015-11-16	preverttechnology.com	Wajam	✓	✓			Hybrid Analysis
C10	WWE_1.58.101.25.exe	✓	2016-01-04	yvonlheureuxtechnology.com	Wajam	✓	✓			Hybrid Analysis
B6	WIE_2.40.10.5.exe		2016-01-19	*N/A*	Wajam	✓		✓		Hybrid Analysis
C11	WWE_1.61.80.6.exe	✓	2016-02-23	saintdominiquetechnology.com	*(nothing)*	✓	✓		✓	Hybrid Analysis
C12	WWE_1.61.80.8.exe	✓	2016-02-24	saintdominiquetechnology.com	Wajam	✓	✓			Hybrid Analysis
C13	WWE_1.63.101.27.exe	✓	2016-03-25	carmenbienvenuetechnology.com	Wajam	✓	✓			Hybrid Analysis
C14	WWE_1.64.105.3.exe	✓	2016-04-07	Telecharger-Installer.com	Wajam	✓	✓			Hybrid Analysis
D1	WBE_0.1.156.12.exe	✓	2016-04-11	technologieadrienprovencher.com	Wajam	✓	✓			VirusShare
C15	WWE_1.65.101.8.exe	✓	2016-04-14	sirwilfridlauriertechnology.com	Wajam	✓	✓			VirusShare
D2	wbe_0.1.156.16.exe	✓	2016-04-21	technologieadrienprovencher.com	Wajam	✓	✓			VirusShare
C16	WWE_1.65.101.21.exe	✓	2016-04-21	sirwilfridlauriertechnology.com	Wajam	✓	✓			VirusShare
D3	WBE_3.5.101.4.exe	✓	2016-04-28	technologieadrienprovencher.com	Wajam	✓	✓			Hybrid Analysis
C17	wwe_9.66.101.9.exe	✓	2016-05-09	sirwilfridlauriertechnology.com	Social2Search	✓	✓		✓	VirusShare
D4	WBE_11.8.1.26.exe	✓	2016-08-29	technologieferronnerie.com	Social2Search	✓	✓			Hybrid Analysis
C18	patch_1.68.15.18.zip	✓	2016-10-18	beaubourgtechnology.com	*N/A*	*N/A*	*N/A*	*N/A*	✓	wajam-download.com
D5	WBE_crypted_bundle_11.12.1.100.release.exe	✓	2016-11-22	emersontechnology.com	Social2Search	✓	✓			Hybrid Analysis
D6	WBE_crypted_bundle_11.12.1.301.release.exe	✓	2017-01-30	wottontechnology.com	Social2Search	✓	✓			Malekal MalwareDB
D7	WBE_crypted_bundle_11.12.1.310.release.exe	✓	2017-02-03	piddingtontechnology.com	Social2Search	✓	✓			Hybrid Analysis
D8	WBE_crypted_bundle_11.12.1.334.release.exe	✓	2017-02-10	quaintontechnology.com	Social2Search	✓	✓			Hybrid Analysis
D9	WBE_crypted_bundle_11.13.1.52.release.exe	✓	2017-03-21	wendleburytechnology.com	Social2Search	✓	✓			Hybrid Analysis
C19	patch_1.77.10.1.zip		2017-04-01	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*		wajam-download.com
D10	WBE_crypted_bundle_11.13.1.88.release.exe	✓	2017-04-13	technologieflagstick.com	Social2Search	✓	✓			Hybrid Analysis
D11	Setup.exe	✓	2017-07-11	terussetechnology.com	Social2Search	✓				Hybrid Analysis
D12	Setup.exe	✓	2017-08-25	vanoisetechnology.com	SearchAwesome	✓				Hybrid Analysis
D13	Setup.exe	✓	2017-09-18	technologievanoise.com	SearchAwesome	✓				Hybrid Analysis
D14	s2s_install.exe	✓	2017-11-27	boisseleautechnology.com	SearchAwesome	✓				Hybrid Analysis
D15	update.exe	✓	2017-12-25	barachoistechnology.com	SearchAwesome	✓				Hybrid Analysis
D16	Setup.exe	✓	2018-01-02	technologienouaillac.com	SearchAwesome	✓				Hybrid Analysis
D17	Setup.exe	✓	2018-02-12	pillactechnology.com	SearchAwesome	✓				Hybrid Analysis
D18	Setup.exe	✓	2018-02-19	pillactechnology.com	SearchAwesome	✓				Hybrid Analysis
D19	Setup.exe	✓	2018-03-05	technologiepillac.com	SearchAwesome	✓				mileendsoft.com
D20	Setup.exe	✓	2018-04-18	monestiertechnology.com	SearchAwesome	✓				technologiesnowdon.com
D21	Setup.exe	✓	2018-05-30	bombarderietechnology.com	SearchAwesome	✓				technologiesnowdon.com
D22	Setup.exe	✓	2018-06-12	technologiebombarderie.com	SearchAwesome	✓				technologiesnowdon.com
D23	Setup.exe	✓	2018-07-16	technologievouillon.com	SearchAwesome	✓				technologiesnowdon.com

Legend: The "Filename" is the most descriptive name we found from either the source where we found the sample, HA [18] or VirusTotal. "Signed component" indicates whether the installer or a component it installs is authenticode-signed, in which case the Date column refers to the authenticode signature date, otherwise it shows the latest file timestamp among all installed files. "Authenticode CN" reflects the corresponding Common Name on the signing certificate. "Installed name" refers to the name of the application that appears in the list of installed programs on Windows. "Autoinstall" reflects the ability of the installer to automatically proceed with the installation without user interaction (beyond launching the executable and agreeing to the UAC prompt), i.e., it does not require clicking a button first or giving consent. "Open webpage" indicates whether a Wajam website is opened at the end of the installation (typically to congratulate the user). "Stealthy" indicates whether the installation process is totally transparent to the user. It requires Autoinstall and not opening a webpage by the end of the setup, and also not showing any setup window. "Rootkit" indicates the ability to hide the installed application folder from the user. Finally, "Origin" indicates the provenance of the sample.

A. *Sample collection*

We obtained our first sample with a known URL to *wajam.com* through the Internet Archive as it is no longer available on the official website. This sample dates back from December 2014, and appears to be a relatively early version of the product. We obtained 10 more samples from an old malware database [37] by searching for the application name, two of which were only components of the whole application (DLLs), which we discarded. After we analyzed a few of these samples, we learned about URLs fetched by the application, which allowed us to query keywords from another malware database [18]. We also learned the URLs that serve variants of the installer and downloaded one per month in 2018. At the end of this iterative process, we collected 48 standalone installers, two online installers, and two update packages.

The variants we fetched directly from Wajam's servers are named `Setup.exe`; however, when submitting these samples on VirusTotal, they are sometimes already known by other filenames, e.g., `update.exe`. We could not find obvious paths that include such filenames on known Wajam servers. This could suggest that Wajam is also hosted elsewhere, or downloaded through different vectors.

As most of the samples are digitally signed and timestamped or install a signed component, we could trace the history of Wajam over five and a half years, from Jan. 2013 to July 2018.

B. Categories

We identified four injection techniques that were used mostly chronologically. Hence, we refer to each group as a *generation*. We provide the distribution of samples among generations in Table II. In the rest of the paper, we refer to a given sample by its generation letter followed by its chronological index within its generation, e.g., C18. We keep a numerical reference when referring to an entire generation, e.g., third generation.

Generation A: Browser add-on. The two oldest samples (Jan. 2013 and 2014) install add-ons to Chrome, Firefox and IE. There was a Safari add-on as well according to the "Uninstall" page on *wajam.com*. A Chrome add-on is still available as of Nov. 2018. These add-ons were used to directly modify the content of selected websites to insert social-related results in search pages, and ads. In samples A1–2, the injection engine, *Priam*, receives search queries and bookmark events.

Generation B: FiddlerCore. Samples from Sept. 2014 to Jan. 2016 have their own interception component and leverage the FiddlerCore library [44] to proxy browser traffic. Each detected browser has its proxy settings set to localhost with a port on which Wajam is listening. HTTPS traffic is broken at the proxy, which certifies the connection by a certificate issued on-the-fly and signed by a root certificate inserted into the Windows and Firefox trust stores. Only selected domains are intercepted. The application is installed in the Program Files folder with a meaningful name; however, core files have long random names. Since no component strictly requires a signature by the OS, some samples do not bear any signature. We rely either on a signature on the installer (as seen prior to 2015), or the timestamp of the latest modified file installed (from 2015) to establish a release date for those samples.

Generation C: Browser process injection. Installers dated between Oct. 2014 to May 2016 and two update packages up to Mar. 2017 inject a DLL into IE, Firefox and Chrome, and hooks specific functions to modify page contents after they are fetched from the network (and decrypted in the case of HTTPS traffic), but before they are rendered. As a consequence, the injected traffic in encrypted pages is displayed while the browser shows the original server certificate, making this generation more stealthy (for similar MITB attacks, see e.g., Zeus [25], SpyEye [33], Citadel [53]). We tested the latest

TABLE II: Distribution of samples among generations

Gen.	Period covered	# samples	Injection technique
A	2013-01 – 2014-07	4	Browser add-on
B	2014-09 – 2016-01	6	FiddlerCore
C	2014-10 – 2017-03	19	Browser process injection
D	2016-01 – 2018-07	23	NetFilter+ProtocolFilters

versions of IE/Firefox/Chrome on an up-to-date Windows 7 32-bit and confirmed that the injection method is still fully functional. We later found that browser hooking parameters are actively maintained and kept updated hourly (Section VIII).

Generation D: NetFilter SDK+ProtocolFilters. Starting from Apr. 2016, a fourth generation implements yet another injection technique, based on NetFilter. Installers dated after May 2016 install a program called Social2Search instead of Wajam. Furthermore, samples dated from Aug. 2017 (i.e., few months after the company was sold to IMTL) are again rebranded as SearchAwesome. The NetFilter SDK enables traffic interception, which is combined with ProtocolFilters that provides APIs for tampering with the traffic at the application layer. Instead of explicitly configuring browsers' proxy settings, NetFilter installs a network driver that intercepts all the network traffic irrespective of the application. In this generation, all HTTPS traffic is intercepted and all TLS connections are broken at the proxy, except for the traffic originating from blacklisted process names.

V. Methodology and Results Summary

A. Test environment and sample execution

We leverage VMware Workstation (WS) and an up-to-date installation of Windows 7 Pro 32-bit with Internet Explorer (IE) 11 and Firefox 61 to capture the installation process. For each sample, we instrument WS to start from a fresh VM snapshot, transfer the sample on the guest's desktop, start Process Monitor[2] to capture a trace of I/O activities, and start Wireshark on the host OS to record the network traffic. We also take a snapshot of the filesystem and registry before and after the sample is installed to detect modifications made on the system.

We run the sample with UAC disabled to avoid answering the prompt, and complete the installation, which usually requires clicking only one button. It could be possible to instrument the UI to fully automate the process; however, we wanted to verify whether the sample installs without asking for user consent, opens a webpage at the end of the setup, or if the process is completely stealthy. We note that the UAC prompt is not a significant barrier for Wajam, as it is found bundled (statically or downloaded at runtime) in other installers for which users already gave admin privileges.

We could have used existing malware analysis sandboxes; however, a local deployment would have been required as we needed control over certain registry keys (e.g., Machine GUID[3]) for our analysis. Furthermore, for consistency and

[2]https://docs.microsoft.com/en-us/sysinternals/downloads/procmon

[3]The Machine GUID used in our experiment is `81cba4ed-36b4-4d66-9b6a-6a4a508dc394`

ease of debugging, we used the same environment to capture runtime behaviors and selectively debug samples.

We also verify for selected samples whether they remain fully functional on a Windows 8.1 Pro 64-bit installation. We noticed that some samples lead to a denial of service for certain websites in this case. We also conduct a more thorough analysis on selected samples from each generation as needed to fully understand functionalities, by debugging the application and conducting MITM attacks.

B. Dynamic and static analysis methodology

Anti-analysis techniques are not reflected by the changes made to the system; hence, we look more closely at each step of installation and execution of the payloads.

Studying NSIS installers. Wajam is always based on Nullsoft Scriptable Install System (NSIS [60]), a popular open-source generator of Windows installers [55]. NSIS uses LZMA as a preferred compression algorithm and as such, 7-Zip offers the ability to extract packed files in NSIS-generated installers, provided they are not using a modified version [41]. We used 7-Zip for unpacking when possible. NSIS also compiles an installer based on a configurable installation script written in its own language. Several NSIS-specific decompilers used to reconstruct the script from installers but trivial modifications in the source code could thwart such automated tools. 7-Zip stopped supporting the decompilation of installer scripts in version 15.06 (Aug. 2015) [4]. We use version 15.05 and successfully decompile these scripts.

Debugging. We leverage IDA Pro and x64dbg [65] to debug all binaries to understand some of their anti-analysis techniques. Due to the extensive use of junk code, identifying meaningful instructions is challenging. In particular, when reverse-engineering encrypted payloads, we first set breakpoints on relevant Windows API calls to load files (e.g., `CreateFile`, `ReadFile`, `WriteFile`, `LoadLibrary`), then follow modifications and copies of buffers of interests by setting memory breakpoints on them. We also rely on interesting network I/O events as seen in Process Monitor to identify relevant functions from the call stack at that time.

To understand the high-level behavior of decryption routines, we combine static analysis and step-by-step debugging. We also leverage Hex-Rays Decompiler to study the decompiled code when possible; however, obfuscation sometimes fails Hex-Rays. Static analysis is also often made difficult due to many dynamic calls that resolved only at runtime.

C. Results summary

Wajam is composed of several modules, some of them are specific to a generation. We briefly outline their organization, then provide a timeline with evolution milestones regarding anti-analysis and evasion techniques, privacy leaks, and new prominent features. Then, we demonstrate the efficiency of such techniques by the AV detection rates on samples fetched from Aug. to Nov. 2018.

Wajam modules and evolution of anti-analysis techniques. Wajam's installer is the first executable an AV gets to analyze, justifying a certain level of obfuscation that constantly increased over time. It calls a payload (`brh.dll`, called BRH hereafter) to retrieve information about the system and browsers, e.g., browsing histories, which is then leaked. The installed binaries comprise the main application, an updater, and a browser hooker called "goblin" in the 3rd generation, and a persistence module. Several features and new anti-analysis techniques were introduced over the years; see Fig. 2.

Antivirus detection rates. We collected and submitted samples to VirusTotal that we obtain directly from one of Wajam's servers. We pool a known URL to retrieve daily samples as soon as possible after they are released to observe early detection rates. We show in Fig. 1 the detection rates from VirusTotal of 36 samples collected between Aug.—Nov. 2018. The rates are given relative to the release time as indicated by the "Last-Modified" HTTP header provided by the server. We trigger a rescan on VirusTotal approximately every hour after the first submission to observe the evolution for up to two weeks.

The figure illustrates the averaged rates for these 36 samples, along with the overall lowest and highest rate during each hour. The rates converge around 37 detections out of about 69 AV engines at the end of the two-week period, and include most popular AVs with few exceptions. Importantly, we notice that the rates start arguably low during the first hours. The lowest detection ratio of 3/68 is found on Aug. 8th's sample, 19min after its release. The average rate during the first hour is only about 9.

Wajam is rarely labeled as is by AVs. Rather, they often output generic names[4] or mislabel samples.[5] Certain AVs label Wajam as PUP/not-a-virus/Riskware/Optional;[6] however, we note that depending on the configuration of such AVs, no alert or action may be triggered upon detection, or the alert may show differently than for regular malware [24], [21].

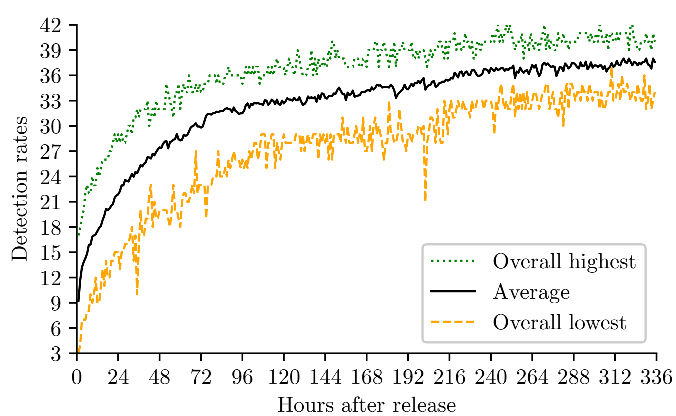

Fig. 1: VirusTotal detection rates of 36 samples starting from their release time

[4] "*Win32.Adware-gen*", "*heuristic*", "*Trojan.Gen.2*", "*Unsafe*"

[5] "*Adware.Zdengo*", "*Gen:Variant.Nemesis.430*"

[6] "*Generic PUA PC (PUA)*", "*PUP/Win32.Agent.C2840632*", "*not-a-virus:HEUR:AdWare.Win32.Agent.gen*", "*PUA:Win32/Wajam*", "*Pua.Wajam*", "*Riskware.NSISmod!*", "*Riskware*", "*PUP.Optional.Wajam*"

Domains. We tracked 248 domains used by Wajam for fetching updates, injecting ads, distributing or signing installers, domains declared by the company in its legal documents [45], as well as other domains that were hosted simultaneously from the same IP address.[7] We provide the list in Table X. Those domains do not change over time, and mostly follow similar patterns (technologie*.com or *technology.com). During our study, they were hosted in France (OVH) and the US (Secured Servers). Some served browser-trusted certificates issued by RapidSSL until Mar. 2018, then switched to Let's Encrypt.

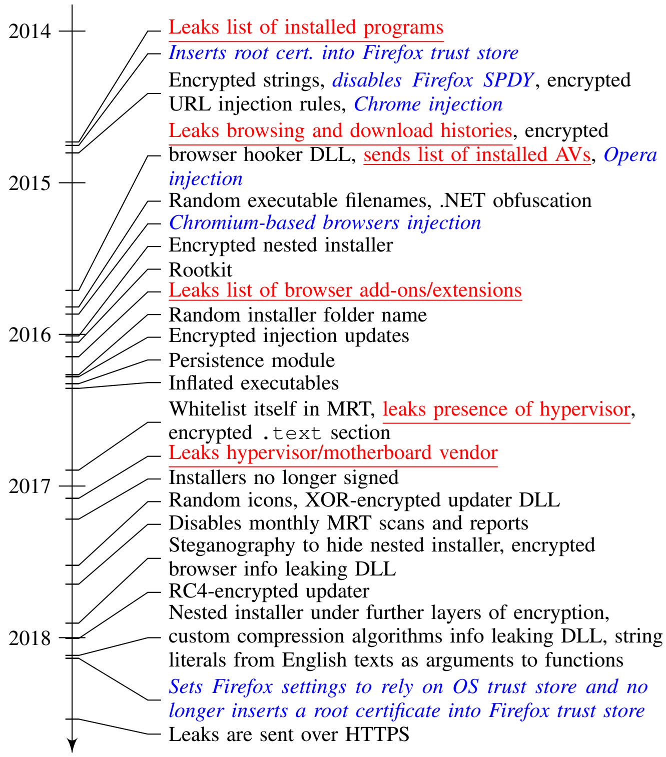

Fig. 2: Timeline of first appearance of key features according to collected samples (colors: black → anti-analysis/evasion improvements, *blue* → new features, red → information leaks)

VI. Anti-Analysis and Evasion Techniques

Wajam's installer makes use of several techniques to hide its payload and thwart static analysis. Some techniques are also shared with the installed binaries. We detail those numerous techniques below.

A. Nested installers and steganography

To study the installed application, we could simply unpack the files from the NSIS installer. In general, however, unpacked files could be called by the installer with specific parameters, or be further altered to initiate a second layer of unpacking. Indeed, starting from C10, only a single encrypted file is packed in the installer. The decryption of this module is detailed in Section VI-D.

[7] We leverage historical DNS data from *DnsTrails.com*.

Algorithm 1 Double XOR operations in samples 46 and above

Input: ciphertext c, first key key_1, second key key_2
Output: plaintext p

$p \leftarrow []$
for i from 0 to len$(c) - 1$ **do**
  $p[i] \leftarrow c[i] \oplus key_1[i \mod \text{len}(key_1)]$
  $key_1[i] \leftarrow p[i]$
  $p[i] \leftarrow p[i] \oplus key_2[i \mod \text{len}(key_2)]$
  $key_2[i] \leftarrow p[i]$
end for

Starting from D14, Wajam's installer unpacks a handful of small DLL files along with a large picture or audio file, including MP3, WAV, BMP, GIF, PNG. At first, this media file appears to contain only random audio noise or colors and could play the role of a *confounder* only useful to arbitrarily inflate the installer's size (cf. [26]). In reality, the installer script instructs NSIS to call a function from one of the DLLs, which in turn reads and reassembles a payload hidden in the media file. For instance, in D14, an MP3 file is composed of MPEG frames starting with a four-byte header and followed by 622 bytes of data. We found that the DLL extracts and concatenates the data section from each frame to reconstruct a GZip file, which in turn reveals a second NSIS installer. This nested installer contains the actual files to install and is guided by a several-thousand lines obfuscated NSIS script.

With time, the techniques become more sophisticated. In particular, the payload may start at a random offset of a data section to thwart effort that may fingerprint known formats in such areas. Furthermore, we found several layers of encryption and a non-standard compression technique to replace GZip. See Table III for examples of techniques we uncovered. The double XOR operation mentioned in this table is detailed in Algorithm 1.

B. Obfuscation

The code responsible for decoding the payload in media files or enabling some other features (detailed in Section VII), is itself heavily obfuscated using a large amount of junk code. including numerous added layers of functions that 1) perform string manipulation on large random strings, 2) perform inter-register operations, 3) call Windows library functions that only swap or return some fixed values, 4) test the result of such dummy functions, or 5) large dead conditional branches. Junk code can usually be detected because its output is not used elsewhere in the program. In Wajam, the junk code often checks and changes global variables (BSS section), resulting in non-deterministic behavior that may prevent junk code removal techniques from recognizing these functions.

Moreover, calls to such functions pass as argument unique strings of random characters, or brief extracts from public texts, e.g., we found the Polish version of *Romeo and Juliet*. likely in an effort to avoid function prototype fingerprinting and clone detection, Useful functions are thus difficult to identify. Similar to the installer, the main binary and the updater

TABLE III: Steganographic techniques to hide a nested installer in samples from end-2017 to 2018

ID	Hidden in	Payload reconstruction	Encryption/Compression	Encryption keys
D14–15	MP3	Concatenated MPEG frame data	plaintext (GZip)	*Not applicable*
D16	MP3	Concatenated MPEG frame data	custom encryption	*Not applicable*
D17	GIF	In section after LSD + custom offset	2 XORs + custom compression	`2njZEYFf`, `qsjmoRZ7FM`
D18	BMP	BitmapLine section + custom offset	custom encryption + 2 XORs + custom compression	`ldXTyqwQ`, `ckXKI19jmC` (XOR keys only)
D19	WAV	First DataChunk samples + custom offset	2 XORs + custom compression	`47txnKuG`, `eyimwKIOBG`

are filled with junk code and added string arguments to dummy functions. However, this time the strings are either random, or taken from *The Art of War* by Sun Tzu starting in D17.

Furthermore, external library calls are made dynamically by calling the `LoadLibrary` API function provided with a DLL name as argument, generated at runtime using complex string manipulations. In D17, the DLLs that read and decode media files contain more than 2000 and 400 junk functions, respectively, that can be called up to a dozen times each. The resulting call graph is unhelpful.

C. Digital signatures

Early samples were digitally signed by COMODO or thawte, which could help the installer appear legitimate to users when prompted for administrative rights (when distributed as a standalone app), and lower detection by AVs [32]. The signature disappeared from sample D9, shortly after Wajam was sold to IMTL. This move could be explained as signatures could help antiviruses to fingerprint the installer, or simply because Wajam already inherits from admin privileges from the software installer that runs it. We note that installed system drivers are still signed as a signature is required by Windows. They are signed by certificates issued by DigiCert or GlobalSign, to domain names that belong to Wajam.

D. Encryption

Various encryption layers are leveraged at different stages of the installation and runtime, which evolve across variants.

1) Installer: Installers from 2016 to mid-2017 (C10–C16, D1–D10) contain a file, often named `wie.dat`, that is decrypted by the primary NSIS installer, although not by using its installation script and external DLLs. Each file footer contains a 64-byte random key, used to decrypt the rest of the file using RC4, which reveals the nested NSIS installer. Table VI lists the keys we recovered for such samples.

Samples that rely on steganography add a layer of decryption after the payload is recovered from the media files, starting from D16. In certain media formats, it is possible to observe 18 or more printable characters towards the beginning of the payload. This string is split into two keys that are combined with XOR operations to decrypt the rest of the payload.

2) DLL/side executables: Steganography-based samples D14–18 further protect the BRH, by XORing it with a random string found in a stub DLL. Due to the challenges in understanding the decryption routine to find the key, we found that it is easier to brute-force the decryption with all printable strings from that stub DLL until an executable format is decrypted. Alternatively, since parts of the PE headers are predictable, it is possible to recover this key using a known-plaintext attack. However, since D17, this attack is no longer possible as the plaintext is further compressed using a custom method for which there is no known fixed values. Table IV lists the keys we recovered for the BRH.

TABLE IV: Decryption keys for the DLL used to retrieve information about the system and browsers (`brh.dll`) found encrypted in samples from end-2017 to 2018

ID	XOR key	Output
D14	`NAF6TDWRR8H0E3`	plaintext
D15	`K3H20MKNH5UZKO`	plaintext
D16	`AVBZALVDGSAQ2MXF1WHE3XU`	plaintext
D17	`0BYRGU14TWHBNTQ0P`	custom compression
D18	`RR5TQZ88AL6E7Z4NS8`	custom compression
D19-23	*(not fully RE'd)*	*(not fully RE'd)*

Similarly, the goblin DLL is compressed and encrypted starting from C6 using RC4 and a hardcoded 16-byte key. The key is located in the main executable and can be found by extracting all strings and trying them to decrypt the DLL until a valid GZip header appears. The DLL is sometimes decrypted at runtime and written back to disk to be injected directly into browser processes. It is inflated by appending 10MiB of apparently random data, and by changing its name and the name of the exported function to random ones. Table V lists the keys we recovered for the goblin module.

TABLE V: Decryption keys for the "goblin" DLL injected into browsers in samples from the third generation

ID	Key	Type	DLL name
C6	`Q7P6ZTLWMLK6HTU3`	RC4	wajam_goblin.dll
C7	`TKOHVURJCWAXXINA`	RC4	wajam_goblin.dll
C8	`CPAU7VKQRI7U8PEK`	RC4	`md5(GUID+'wajam_goblin.dll')`
C9	`NT0DRJ1RJKIWSSA7`	RC4	`md5(GUID+'wajam_goblin.dll')`
C10	`3ZHLH3HJ4NOW1FVK`	RC4	`md5(GUID+'wajam_goblin.dll')`
C11	`KVFB47HIYXRVNT4T`	RC4	`md5(GUID+'wajam_goblin.dll')`
C12	`BQS1MUAW64ENNRF3`	RC4	`md5(GUID+'wajam_goblin.dll')`
C13	`HBS57M2BD1OHHK6S`	RC4	`md5(GUID+'wajam_goblin.dll')`
C14	`5682VXAM34MFB5TK`	RC4	`md5(GUID+'wajam_goblin.dll')`
C15	`56B38AXWW2YAAMMH`	RC4	`md5(GUID+'wajam_goblin.dll')`
C16	`1M7O6L9LU4C2KMIK`	RC4	`md5(GUID+'wajam_goblin.dll')`
C17	`T0R00V9B64TR7RKK`	RC4	`md5(GUID+'wajam_goblin.dll')`

Finally, a separate updater runs a Windows service that relies on an encrypted payload called `service.dat`. In D11–15, the encryption also simply relies on a 16-byte XORed pattern; however it is not found as plaintext in the main or updater file. Instead, by XORing a known pattern from the PE header, we can recover the key. To fix this weakness, samples starting from D16 switched to RC4, forcing the search of the key obfuscated in one of the executables.

TABLE VI: Nested installer's decryption keys for samples from 2016 to mid-2017

ID	Nested installer filename	RC4 key (64 bytes)
C10	wie.dat	`AXOD3MTRAXX9ISMKLRE401YOJCJOZZL7NOBDTBJ2033UWCNO9QA6JJFOMROLD5KI`
C11	wie.dat	`88D03624GQWEZUBFUJZ1PJHWB1UYU5COP8UU3FW4NV1ID85Q8M57PFNFTL4C3YMR`
C12	wie.dat	`RT93UX0MIDZQVMXT2QBZFV5358F477KPLGX1ZCXV4UWPC0ZXZSOR7YF1MGJVLZOY`
C13	wie.dat	`6OE985384DJTMR44UD2P77BDEHMX03Q603KZT5H7KMTI18A76P6NOBEWGQ92CIED`
C14	wie.dat	`NIFSDC8UDA9I1QZGVXA446WGWI7YC0RZTBYRX50SY57SI3W21U9LZHW3BNN2CZTF`
D1	wie.dat	`R2SFHDEPTV3WGO8ZJUMJ4DW6PXEWDFXZYTX7FA6BA8EKFQVO7FC5X2GCEVKN3H0R`
C15	wie.dat	`A56GE1T9P8EK6O8VFFR4RM6NNX4I1NWYT82EC39WLDBBDS6QMWVYZWTMK3D1NBQ4`
D2	wie.dat	`K0BEB3V1JY0AA5HLWZKTTX95CTWZPM2N0KIWIB8XVZXQ9EM38EG27TOJXACPCGGX`
C16	wie.dat	`5CFRULZVADR6C05MOFL4IJH6V8UBJ81CID5AQNRS2XVDP2LI03PQ0GQG0HUI7ZTP`
D3	wie.dat	`ZNSNBB99R8EQWIL7VB7NWC0S02ALWLB40RW1C9JDW346IGI81KMYESFMOA89YDO3`
C17	wie.dat	`HA8K2LHUO8D08EQXQJ0IGL0XBBGWFNM0ROGQPNIB3J5WNKYS4TLOAJIBIPXEPYS6`
D4	wie.dat	`SP347G50FVI0O32ESQIKYUDH94GTWI1VX0W56W858VKDFQROEOVN8ZDALVQRAT95`
D5	wie.dat	`D54PD8AE7ZRCBG9HSEZW3IJ38OUNLKQTSGHU8OCL56L8CC6C0G0VA5P5IPN6Z5Y5`
D6	chvfcNyhg	`AK7VBN4JF7LAGY4PO1VFZVV2TUKTOQWOEVOHKSWJB7KSV47WK452RWVDOKWE418P`
D7	XUUw8ETr58EQQBUXE2W	`3UI7IX2F3L5RKMQUSN5XSDZUOY7WMRWIH1XT3H0U1N4YLYXWRRV9QF87ED4682CW`
D8	WrTQxzGTW	`090C0U9PUC287RXILDXJ7Y8J0ZMTBMLOB9WJ3E3XV5OLOYF00N1S1RP97OMYGPG5`
D9	sGC6_x	`HNSOWVL1V9SO9W6JA7BTEUOGYR7YPPL3HC5D5SZF51GN90A5OHTCFDT1F5F82EW0`
D10	g044B2e	`9VMW325WML3F0JBTKIW492R8IQVVYF8THXKPRLGZAFHG5BDSV1GBGQZM1T6ZE0HH`

3) Main application: Samples from the beginning of the 3rd generation come with most of their strings encrypted, which are decrypted at runtime by a simple XOR operation. There are two flavors depending on the encoding of the plaintext string: a single-byte key for ASCII strings, and a two-byte key for Unicode strings. Each string is associated with one decryption function. Encrypted strings are loaded on the stack either 4 or 16 bytes at a time (through a 32-bit or XMM register), probably due to different compiler optimizations. The lowest stack address and the key are then passed to the in-place decryption function.

The main executable's code section is encrypted in D5–10 with a custom algorithm based on several byte-wise XOR and subtraction operations. Chunks of 456KiB are decoded with the same logic, while each chunk is decoded differently. Such samples correlate with installers whose name is prefixed with "WBE_crypted_bundle_", suggesting that the encryption layer was added after compilation.

The encrypted executable sections are often large and difficult to parse by a disassembler. In our case, IDA Pro hangs for more than two hours on sample D9, which contains 4MiB of the byte `B9`, followed by another 3MiB of encrypted instructions. It is possible to zeroize the constant part of the code section to accelerate the parsing by IDA; however, since most of the program's code is encrypted, we study dumps of the running process, once its code section is fully decrypted.

E. Inflated size

Some malware scanners are known to discard large file [15], [35], hence an obvious anti-analysis technique is to inflate the size of the executable. Seven samples rely on enlarged `.rdata` (C17, D4) or code sections (D6–10), resulting in binaries ranging from 9 to 26MiB in size. The first type consists of a large `.rdata` section that contains strings duplicated hundreds of times. However, this section contains actual strings used in the unobfuscated application. Given that such strings are meant to be decrypted at runtime, it is unclear why the developers left plaintext strings in the binary, or if large `.rdata` sections are at all meant for evasion. Large code sections tend to slow IDA Pro's analysis, possibly due to gibberish instructions parsed.

F. Unique file and folder names

Before B4, the installer unpacks most of the files in a folder under Program Files. The names of the files and folders are static and are well connected to Wajam. Executable filenames appear random in later samples. The installation folder itself becomes randomized from C14 and D3. The names are actually derived from the original name combined with the Machine GUID obtained from registry, and hashed using MD5.[8] The Machine GUID is considered to be a unique identifier bound to a given installation of Windows and stays the same until it is reset, which can be triggered by removing the registry key. As a result, the installed files would always be named identically on a given instance of Windows for samples with compatible naming scheme, while the names are different across machines.

G. Antivirus scanning evasion

Starting from D5, Wajam adds itself to the exclusion list of Windows Defender in an effort to avoid detection once Defender could detect Wajam due to a definition update.[9] Starting from D12, Wajam also modifies the settings of Windows Malicious Software Removal Tool (MRT). MRT is served each month as an update through Windows Update and performs a malware scan of the system. Wajam disables this monthly scan. It also disables the reporting of infections to Microsoft. Figure 3 shows the NSIS script responsible for changing MRT's settings, with support for 32-bit and 64-bit systems.

[8]For instance, `C:\Program Files\WaNetworkEn\wajam.exe` becomes `C:\Program Files\686d944556d5de03afc6aa639bff9c7\06ca8c13762fca02c5dae8e502fd91c9.exe`, with the folder name corresponding to `md5(MachineGUID+'WaNetworkEn')` and the filename taken from `md5(MachineGUID+'wajam.exe')`.

[9]Wajam inserts the paths of its main components under `HKLM\Software\Microsoft\Windows Defender\Exclusions\Paths`

```
Function func_3030
  SetRegView 64
  WriteRegDWORD HKLM SOFTWARE\Policies\Microsoft\MRT
      DontReportInfectionInformation 0x00000001
  WriteRegDWORD HKLM SOFTWARE\Policies\Microsoft\MRT
      DontOfferThroughWUAU 0x00000001
  SetRegView 32
  WriteRegDWORD HKLM SOFTWARE\Policies\Microsoft\MRT
      DontReportInfectionInformation 0x00000001
  WriteRegDWORD HKLM SOFTWARE\Policies\Microsoft\MRT
      DontOfferThroughWUAU 0x00000001
FunctionEnd
```

Fig. 3: NSIS script to modify Microsoft MRT settings

H. Rootkit capabilities and persistence

We found two samples, C11 and C17, that rely on a kernel-mode driver to hide the installation folder from the user space, effectively turning Wajam into a rootkit. Sample C11 remains even more stealthy as it does not register itself as an installed program and hence does not appear in the list for users to uninstall it. The file system driver responsible for hiding Wajam's files is called Lacuna and is either named `pcwtata.sys` or similar, and is signed by DigiCert. We also found an update package (C18) that also comes with this driver.

Wajam establishes persistence through executables or scripts that are left in the `C:\Windows` folder and not removed by uninstalling the product. While executables could be detected by antiviruses, Wajam leverages (obfuscated) Powershell scripts in samples C17, D3 and D12–13. A scheduled task is left on the system to trigger the persistence module at user logon. From D14 onward, the persistence module is a regular executable, inheriting some anti-analysis techniques previously mentioned, and set up as a Windows service that starts at boot-time. The module checks for the presence of the installation directory and main executable. If they do not exist, the module follows the process of updating the application by querying a hardcoded URL to download a fresh variant. This behavior is mostly intended for reinstalling the application after it has been uninstalled, or removed by an antivirus. However, we found that the hardcoded URL is not updated throughout the lifetime of the module on the system, and could be inaccessible when necessary.

I. .NET and Powershell obfuscation

In the FiddlerCore generation, the Windows service is responsible for adjusting the browser proxy settings and launching the FiddlerCore-based network proxy written in C#. Samples from 2014 are not obfuscated and the C#/.NET components are decompilable. Starting from sample B4, the method and variable names of C# components are randomized. The deobfuscator de4dot [3] detects that Dotfuscator [43] was used to obfuscate the program; however, only generic method and variable names were reconstructed. Also, de4dot does not remove obvious dead code. Indeed, useful lines of code are interleaved with string declarations made of concatenated random strings. Since such strings are never used, except possibly in the declaration of other such strings, they are easy to remove automatically.

Fig. 4: Icon polymorphism with slight pixel alteration

Fig. 5: Icons used in the Wajam's installers we collected

The Powershell persistence module consists of a long *encrypted standard string*, using a user-specific key. As the script runs with SYSTEM privileges, only this account can successfully decrypt the string, revealing another Powershell script that is then invoked. Since decrypting such strings is not directly allowed, the script converts the standard string to a *SecureString*, creates a `PSCredential` object, and sets the SecureString as the password. Then, it obtains the plaintext password from this object.

J. Polymorphic icon

Early versions of Wajam shared the same icon on their installers. The icon is later changed between variants at few random pixel locations. The color of these pixels is slightly altered to give a new icon while remaining visibly identical, see Figure 4. As a result, the hash of the resource section varies, preventing easy resource fingerprinting. Starting from D11, Wajam pick random icons from third party icon libraries for both the installer and installed binaries. An illustration is given in Figure 5.

K. Summary: Integration of the techniques

Wajam leverages several techniques and layers to hinder static analysis, fingerprinting, reverse engineering, and antivirus detection. A typical sample from 2018 is an NSIS installer with a random icon that unpacks DLLs that find, deobfuscate, decrypt and uncompress a second-stage installer from a media file. In turn, this second installer executes a long obfuscated NSIS script that first calls an unpacked DLL to decrypt and load its BRH companion to perform a number of leaks (see Section VII). Then, it installs the main obfuscated Wajam files under Program Files with names following the `md5(GUID+'filename')` pattern. It also adds a persistence module in the Windows directory along with the generated TLS certificate in an 'SSL' subdirectory, and a signed network driver in the `System32\drivers` folder. The installer creates three Windows services: 1) the network driver, 2) the main application, 3) the persistence module; and a scheduled task to start the second service at boot time if not already started. It disables MRT monthly scanning and reporting. The main application starts by reading the encrypted updater module, decrypting and executing it. In turn, the module reads the encrypted injection rules, updates them and fetches program updates.

VII. Leaks

Beyond installing the files onto the system, the installer also performs other core tasks, including the generation of unique IDs, leaking browsing and download histories, and the presence of AVs.

A. Unique IDs

Two unique identifiers are generated during installation, and written in the Windows registry. All requests made to Wajam's servers include these identifiers. The first one, called `unique_id` or `uid` is generated as the uppercased MD5 hash of the combination of: 1) the MAC address of the main network adapter, 2) the path for the temporary folder for applications (which contains the user account's name), and 3) the corresponding disk's serial number. The calculation of second identifier, `machine_id` or `mid`, appears to intend including the Machine GUID; however, a programming error fails to achieve this goal, and instead includes some artifact of the string operations performed on the MAC address. In our case, the `mid` was simply the MAC address prepended by a "1". This issue was never fixed. These identifiers are used for ad tracking, and to detect repeated installations to identify pay-per-install frauds by Wajam distributors (i.e., a distributor faking numerous installations to increase its revenue from Wajam) [42].

B. Leaking personal and browser info

In A2, the installer sends a verbose installation log over plain HTTP to a script named `client_send_debug_info.php` on *wajam.com*. The POST request contains full paths including the user's home directory, along with the network adapter's MAC address, the drive's serial number, and the unique IDs mentioned above. This behavior occurred only in this sample. Given the name of the target script and the single occurrence of such installer, the sample could be a version intended for debugging purposes only.

Starting from B1, the installer leaks the list of installed programs as found in the registry, minus Microsoft-specific updates in some cases. The leak may happen several times during the installation, possibly due to multiple components leaking this information. The OS version and the date of the installation obtained from Wajam's own timestamping service, are also sent in each queries.

From C6, the browsing history of IE, Firefox and Chrome is sent in plaintext to Wajam's servers, and the history of Opera from D6. Only the newest sample we analyzed, dated from July 2018, sends this information over HTTPS. This leak is the most privacy-sensitive. For users who do not configure an expiration of their history, the leak could span over several months worth of private data. In Chrome, the local history expires after three months [6], mitigating the extent of the leak; however, other browsers do not expire their history, which could last for years. In parallel, the download history, i.e., the URLs of downloaded files, is also sent in plaintext except in the latest sample. Functions exported by the DLL in charge of collecting this data have explicit names, e.g., `SendAllBrowsersDownloadHistory`.

After the installation, Wajam continues to send the list of browser addons/extensions and installed programs whenever it fetches updates from the server.

Later samples dated after the end of 2016 (from D5) also check the 31st bit of the feature flags stored in ECX as returned by the CPUID instruction, indicating whether the program is running in a virtual machine environment. The result is appended to all HTTP(S) queries made by the installer. The installer also invokes Windows Management Instrumentation (WMI) to obtain the BIOS manufacturer name and appends it to the queries. We are unsure about the consequences of reporting that Wajam is installed in a hypervisor. We still observed fully functional and apparently complete updates, and could observe injected ads.

C. Antivirus detection

In every sample since C6, Wajam looks for the presence of a series of 22 major antiviruses and other endpoint security software, then attaches the list of detected products to almost every query. Notably, not all of products belong to home products. For instance, AhnLab and McAfee Endpoint only offer products to businesses, raising concerns that Wajam also targets enterprises specifically. The list of security product and/or vendors that Wajam searches for are listed in Table VII.

TABLE VII: Security solutions checked by Wajam in registry

AVAST Software	Microsoft Antimalware
AVG	Norman Data Defense Systems
AhnLab	Norton
Avira	Panda Security
BitDefender	Safer Networking Limited
BullGuard Ltd.	SUPERAntiSpyware.com
ESET	TrendMicro
KasperskyLab	UnThreat
Malwarebytes Anti-Malware	VIPRE Internet Security
McAfee Endpoint	WRData
McAfee MSC	Zone Labs

VIII. Updates

We discuss in this section the auto-update mechanism that allows Wajam to update the whole application, the injections rules, or the browser hooking configuration. Updates are fetched upon first launch, then Wajam waits for a duration indicated in the last update (from 50 to 80 minutes in our tests), before it updates again.

A. Encryption

While early samples fetched plaintext update files, all recent samples and the whole 4th generation are downloading encrypted files. The decryption is handled in an encrypted DLL loaded at runtime. We found that Wajam uses the MCrypt library to decrypt updates with a hardcoded key and IV using the Rijndael-256 cipher (256-bit block, key and IV) in CFB-8 mode. The key and IV are the same across all versions.

B. Program update

Wajam starts by querying hardcoded URLs for an `update` or `manifest` file, generally located at `/webenhancer/update`, `/browserenhancer/update` or `/proxy/manifest` on the remote server. Several parameters are passed, including Wajam's version, supposedly to provide relevant updates. The list of detected security solutions is leaked at this point. If an update is available, the URL where to fetch a ZIP package is provided. The ZIP file is uncompressed into the installation directory. Wajam is known to have been updated frequently to avoid antivirus detection [42]; however, the software update manifest we fetched did not specify any available update package to download. We did find two update packages from a malware database, samples C18–19, showing the possibility of updating the application. These two samples contain the main executables and DLLs for 32 and 64-bit systems, with a clear naming convention, e.g., `wajam.exe`, `wajam_goblin_64.dll`. A `patcher.cfg` file contains various parameters including the base name of the executables, i.e., wajam, the service name it should be installed with, e.g., `WajIEn Monitor`, and the type and name of the driver file. In C18, the driver type is explicitly called `HIDING_DRIVER` and refers to the filesystem driver that hides the installation folder. In C19, such a driver is not included.

C. Traffic injection rules

Except the first generation, others also fetch an `injections` or `mapping` file (located at `/addon/mapping` or `/webenhancer/injections`) containing a list of domains and instructions to inject scripts.

The injections/mapping file is a JSON structure that contains "supported websites". For each website, a list of regular expressions are provided to match URLs of interest, often specifically about search or item pages, along with a specific JavaScript and CSS URLs to be injected from one of Wajam's several possible domains. The rules also include HTTP headers or tags to be added or removed. Since the content injection relies on loading a remote third-party script, browsers may refuse to load the content due to mixed-content policies or the Content Security Policy (CSP) configured by the website. Mixed-content is addressed by loading the script over the same protocol as the current website. For websites that specify a CSP HTTP header or HTML tag, Wajam is able to remove this CSP from the server's response before the browser sees it, to ensure their script is properly loaded. Figure 6 shows an example of an injection rule (after formatting) where the CSP header is to be dropped from `facebook.com`.

The injection rules fetched between Feb. to July 2018 always include 100 regular expressions to match the domains of major websites, with only one change during this period. The injected domains include popular search engines, social networks, blogging platforms, and various other localized businesses in North America, Western Europe, Russia, and Asia. The list contains notable websites, e.g., Google, Yahoo,

```
[facebook]
  [domains]
    [0] => facebook
  [patterns]
    [0] =>
     ^https?:\/\/(www\.)?facebook.com(?!(\/xti\.php))
  [js]
    [0] =>
     se_js.php?se=facebook&integration=searchenginev2
  [css]
  [headers]
    [remove]
      [response]
        [0] => content-security-policy
```

Fig. 6: Example of traffic injection rule for `facebook.com` that matches all pages except `xti.php`

Bing, TripAdvisor, eBay, BestBuy, Ask, YouTube, Twitter, Facebook, Kijiji, Reddit, as well as country-specific websites, e.g., *rakuten.co.jp*, *alibaba.com*, *baidu.com*, *leboncoin.fr*, *willhaben.at*, *mail.ru*. The total number of websites that are subject to content injection is not easy to quantify due to the nature of some URL matching rules, e.g., in the case of the blogging platform Wordpress, blogs are hosted as a subdomain of *wordpress.com* and Wajam's rules match *any* subdomain, which could be several millions [64].

Bootstrap and cache. The first update is fetched from a hardcoded URL. Later updates are made based on the "update_url" parameter found in the previously fetched file. Once the injection rules are downloaded, they are stored in the program's folder in plaintext in a file named `WJManifest` for early samples (i.e., B2 and earlier), or encrypted as is in a file named `waaaghs` or its obfuscated version (see Section VI-F).

D. Browser injection rules

The third generation specifically retrieves a `config` file (`/webenhancer/config`) with offsets to functions to be hooked in a number of browsers and versions. An example of hooking parameters is given in Figure 7. Unlike the traffic injection rules, the browser injection rules are preloaded in the installer. Hence, it is possible to study their evolution in time. These rules are cached in a similar way as the traffic injection rules, under a file named `snotlings` or its obfuscated version.

History. The earliest third generation sample (Nov. 2014, C1) only includes addresses of functions to be hooked for 47 versions of Chrome, from version 18 to 39. The file also lists supported versions of IE and Firefox, although old and without specific function addresses. In Sept. 2015 (C6), Wajam introduces the support for seven versions of the Opera browser. Two months later, five other Chromium-based browsers are introduced, of which four are adware, i.e., BrowserAir, BoBrowser, CrossBrowser, MyBrowser; and one is a legitimate browser intended for the Vietnamese population, i.e., Coc Coc. By Jan. 2016 (C10), 200 versions of Chrome are supported, up to version 49.0.2610.0 with finer granularity for intermediate versions.

```
[hooks]
  [chrome]
    [...]
    [66_0_3353_2]
      [32bits]
        [PR_Close] => 0x0181C296
        [PR_Write_App] => 0x01824532
        [SSL_read_impl] => 0x01817684
      [64bits]
        [PR_Close] => 0x02318A7C
        [PR_Read] => 0x02312A0C
        [PR_Write] => 0x0232307C
        [PR_Write_App] => 0x0232307C
        [SSL_read_impl] => 0x02312A0C
```

Fig. 7: Browser injection rule for Chrome 66.0.3353.2

Although we did not capture any new sample from the third generation dated past Jan. 2016, we noticed that the browser injection rules are kept up-to-date, suggesting that this generation is still actively maintained and distributed. In an update from July 2018, we count 1176 supported Chrome versions including the latest Canary build, and additional Chromium-based browsers, e.g., Torch, UC Browser, and Amigo Browser. Versions of Opera are outdated by more than a year. Other Chromium-based browsers only have entries for a limited number of selected versions.

Injection methods. The third generation of Wajam injects a DLL into browser processes, which further hooks a number of functions to manipulate the traffic. While the offsets of the functions are available in the hourly update for Chromium-based browsers, IE and Firefox do not require additional information since the functions to be hooked are readily exported by `wininet.dll` (in the case of IE) and `nss3.dll` (for Firefox), and hence can be found easily at runtime. Given the names corresponding to the addresses found in this update file, e.g., `PR_Write`, `SSL_read_impl`, Wajam seems to follow the same function hooking strategy to inject content in the network traffic as the Citadel malware [53].

Wajam avoids intercepting non-browser applications as evident from a blacklistlist of process names in the update file, e.g., dropbox.exe, skype.exe, bittorrent.exe. Additionally, a whitelist is also present, including the name of supported browser processes; however, it appears not to be used.

Furthermore, Wajam seems to have had difficulties handling certain protocols and compression algorithms in the past. It disables SPDY in Firefox. Before Chrome version 46, Wajam also modifies the value located at a given offset that represents whether SPDY is enabled to disable this feature. Similarly, the SDCH compression algorithm is disabled. The number of functions to be hooked evolves from one version of the browser to another, with a different set for 32 and 64-bit versions, sometimes including only `PR_`(`Read`, `Write`, `Write_App`, `SetError`, `Close`), or also `SSL_read_impl`.

E. Injected content

On selected pages (detailed in Section VIII), Wajam injects a JavaScript and CSS right before the `</head>` tag. The scripts were either self-contained in early samples, or inserting remote scripts with parameters including Wajam's and the OS versions/architecture (32 or 64 bits), the two unique IDs discussed in Section VII, an advertiser ID, and the installation timestamp. See Figure 8 for an example of injected content. The remote JavaScript URL to insert in the page is dependent on which website is visited. In particular, two categories of websites are distinguished: search engines, and shopping websites. We give below an example for each case.

Search engines. There are three possible behaviors that we observed when visiting a search engine website. For instance, when searching on *google.com*, Wajam can change the action on the first few results' links returned by Google. In effect, when a user clicks on these results, the original link opens in a new browser tab while the original tab loads a series of ad trackers (including Yahoo and Bing) provided with the keywords searched by the user, and eventually lands on an undesirable page, e.g., a search result page from *informationvine.com* about foreign exchange. Alternatively, the script may just redirect the user to *searchpage.com*, a domain that belongs to Wajam, which in turn redirects to a Yahoo search result page about the user's original search keywords. A user may not notice that her original search on Google is eventually served by Yahoo. In the meantime, her keyword searches were sent to Wajam's server. Also, the Yahoo result URL contains parameters that may indicate an affiliation with Wajam, i.e., `hspart=wajam` and `type=wjsearchpage_ya_3673_fja6rh1`. Finally, Wajam may simply insert several search results that it fetched from its servers, as the top results. Wajam performs a seamless integration of those results in the page, breaching the trust that a user has in the search engine results. This behavior is part of a patent owned by Wajam Internet Technologies Inc [8].

Shopping websites. When searching on *ebay.com*, Wajam loads a 180KiB JavaScript file (more than 7700 SLOC) that contains the Priam engine intended to retrieve search keywords, fetch related ads, and integrate them in the page. The capabilities of this engine seem extensive, explaining its size. Inserted ads are shown at the top of the result list in a large format, also seamlessly integrated, thanks to injected CSS. When the user clicks one of the ads, she is redirected to a third party website that sells a product related to her search.

In both cases, the unique ID generated by Wajam's installer (the `uid`) accompanies each URL pointing to Wajam's domains. In the end, both Wajam and the advertisers can build a profile of the user based on her searches.

IX. Other security issues caused by Wajam

We describe below other security vulnerabilities in Wajam, and security issues it introduces on user machines.

A. Downgraded TLS security

The second and fourth generations leverage a TLS proxy to intercept HTTPS traffic; we briefly investigate it against common security issues, cf. [19].

```
<script data-type="injected" src="//technologietravassac.com/addon/script/google?
    integration=searchenginev2&har=2&v=n11.14.1.86&os_mj=6&os_mn=1&os_bitness=32&
    mid=b8230ac083f9fb5067a66e03b4882491&uid=B77FCD732C2E5337FF907BFAA44758D1&aid=3673&aid2=none&
    ts=1531782569&ts2="></script>
<link rel="stylesheet" type="text/css" href="//main-social2search.netdna-ssl.com/css/cdn/
    min_search_engine_v2.css?wv=1.00434"/>
```

Fig. 8: Example of injected content on google.com

1) Private key generation: The second generation relies on FiddlerCore, which inserts a new root certificate during installation, generated with Microsoft's *MakeCert* utility, with a randomly-generated RSA-1024 key. The certificate's CN varies among samples and includes "Wajam_root_cer", "WNetEnhancer_root_cer", and "WaNetworkEnhancer_root_cer", and previously has been reported as an indication of Wajam infection [54], [12].

In the fourth generation, Wajam leverages NetFilter to intercept connections and ProtocolFilters as a TLS proxy. ProtocolFilters relies on OpenSSL; however, IDA FLIRT was unable to identify any OpenSSL-related functions, even with added signatures from [30]. Considering Wajam's large obfuscated binary file, this becomes an issue to locate the code of the root certificate generation.

NetFilter is statically linked with OpenSSL, as indicated by hardcoded strings (e.g., "RSA part of OpenSSL 1.0.2h 3 May 2016"), which helps us determine the exact version of OpenSSL used, and easily label a number of essential OpenSSL functions that call `ERR_put_error()`. Indeed, such calls specify the source file path and line number where an error is thrown, which uniquely identifies a function. By investigating the use of several such functions, we could identify the part responsible for generating the root certificate.

The function of interest can either generate a RSA-2048 private key or use a default hardcoded one. We found that it uses the default one for the generation of the root certificate. We successfully matched this private key to the root certificate that was installed in the Windows trust store. We performed an MITM attack on our test system designed to verify whether the TLS proxy logic accepts its own root certificate as a valid issuer for site certificates, and the sample accepted our certificate for a test domain. Consequently, all an attacker needs to impersonate any HTTPS websites to a machine running Wajam's fourth generation, is to know the root certificate's CN to properly chain generated certificates. However, in this generation, the CN is generated based on the Machine GUID, similar to installed file names.

2) Common Name generation: We investigated the generation of the CN across samples and report our findings in Table VIII. The name is the 16 first hexadecimal characters of the MD5 hash of the Machine GUID concatenated with various aliases of Wajam depending on the samples (e.g., SrcAAAesom), resulting in e.g., `3fd59c0fada5d9ad`. Recovering this algorithm is not straightforward as several intermediate functions separate the CN generation from the certificate generation. We first identify the function in charge of retrieving the Machine GUID from the registry, and label

TABLE VIII: TLS root certificates in generations 2 and 4

ID	Root certificate's Common Name
B1–B3	Wajam_root_cer
B4–B5	WNetEnhancer_root_cer
B6	WaNetworkEnhancer_root_cer
D1–D2	`md5(GUID+'WajaInterEn')[0:16]`
D3	`md5(GUID+'WNEn')[0:16]`
D4	`md5(GUID+'Social2Se')[0:16]`
D5–D8	`md5(GUID+'Socia2Sear')[0:16]`
D9	`md5(GUID+'Socia2Se')[0:16]`
D10	`md5(GUID+'Socia2S')[0:16]`
D11	`md5(GUID+'Soci2Sear')[0:16]+' 2'`
D12–D21	`md5(GUID+'SrcAAAesom')[0:16]+' 2'`
D22–D23	`base64(md5(GUID+'SrcAAAesom')[0:12])+' 2'`

the parent responsible for concatenating a given string to it and applying the MD5 hash. Then, we identify the function that writes the certificate to a file named after the CN, and trace the origin of the filename to a function that calls the previously labeled function. The argument passed in the call corresponds to the concatenated string. After observing in a few samples that the concatenated string matches the registry key of the installed application, we simply proceed to try this key to match the generated certificates in other samples. The various application names can be found in Table VIII. In the last two samples (D22–23), the process is similar; however, only the 12 first hexadecimal characters of the MD5 hash are taken into account, which are further encoded using base64 giving e.g., `ZmJiYmRiODYxNTZi`. We also found that samples branded as SearchAwesome install a certificate with a CN appended with the digit "2", corresponding to a new feature in ProtocolFilters that appeared in May 2015 [52].

Since the Machine GUID is unpredictable and generally unknown to an attacker, and since the resulting CN carries at least 48 bits of entropy (starting from D22, 64 bits in prior samples), crafting certificates signed by a target Wajam's root certificate is generally impractical. Indeed, an attacker would need to serve an expected number of 2^{47} certificates to a victim before one is accepted. We note that environments with cloned Windows installations across hosts could be more vulnerable if the Machine GUID is not properly regenerated on each host, as it is possible to obtain it from a single host with few privileges.

3) Certificate validation: FiddlerCore-based samples (2nd generation) properly validate server certificates. To verify the fourth generation's server-side certificate validation, we faked a DNS response for *google.com* to point the domain to our own server and served test certificates. We found that Wajam performs certificate validation and rejects self-signed certificates. However, it does not perform hostname validation, and even replaces the CN in the server certificate with the domain requested by the client. As a consequence, a valid certificate for *example.com* is accepted by Wajam for *any* other domain. The certificates seen by the client does not reflect *example.com* but rather the domain that was requested. The browser accepts the certificate since it trusts Wajam's root certificate.

Swapping the CN with the requested one is mitigated since 1) CAs include a Subject Alternate Name (SAN) extension in their certificates, which is copied from the original certificate by ProtocolFilters, and 2) browsers ignore the CN field in a certificate if a SAN extension is present. In particular, Chrome rejects certificates that do not contain a SAN [47]. Consequently, if an attacker obtains a valid certificate for any domain and without a SAN extension, they are still able to perform a MITM attack against IE and Firefox when Wajam is installed.

Despite the deprecation of CN as a way of binding a certificate to a domain [46] in 2000, Kumar et al. [34] recently showed that one of the most common error in certificate issuance by public trusted CAs is the lack of a SAN extension. For the sake of our experiment, we inserted our own root certificate in the Windows trust store and issued a certificate without SAN for *evil.com*. It was successfully accepted by Wajam when visiting *google.com*, and the certificate it generated as a result was accepted by IE.

B. Downgraded website security

Wajam removes CSP headers from the server's response on *mail.ru*, Yandex, Facebook, *flipkart.com*, Yahoo Search, and the `X-Frame-Options` from *rambler.ru*. Such behaviors not only allow injected scripts to be successfully loaded, but also effectively downgrade website security (e.g., XSS vulnerabilities may become exploitable).

The traffic injection rules also support header modifications in the HTTP request, e.g., removal the `Referer` and `Origin` headers. Other response headers are also removed in some cases, including `Access-Control-Allow-Origin`, which would allow the given website's resources to be fetched from different origins than those explicitly allowed by the website, or `X-Frame-Options`, enabling the website to be loaded in a frame.

C. Hijacking updates with persistence

Traffic injection rules are always fetched over plain HTTP. Before mid-Feb. 2018 (D18), software updates are also fetched over HTTP. Although updates are encrypted, an attacker who understands the encryption algorithm and extracts the hardcoded key/IV from any Wajam sample in the last few years, is able to forge updates and serve them to a victim through a simple MITM attack.

As a proof-of-concept, we targeted a prominent banking website, which we will refer as *bank.com*. We suppose that this bank offers online banking and that its login interface is served on https://login.bank.com. We craft an update file that instructs Wajam to insert a JavaScript file of our choice, hosted on our own server, and encrypt it using the key that we recovered in Section VIII-A. The plaintext traffic injection rule is provided in Figure 9. Once the update is fetched by

```
{"version":"1",
 "update_interval":60,
 "base_url":"\/\/attacker.evil\/",
 "supported_sites":
   {"bank":
     {"domains":["bank"],
      "patterns":["^https?:\\\/\\\/login\\.bank\\.
          com"],
      "js":["bank.js"],
      "css":[],"version":"1"}},
 "process_blacklist":[],
 "process_whitelist":[],
 "update_url":"https:\/\/attacker.evil\/mapping",
 "css_base_url":"\/\/attacker.evil\/css\/",
 "url_filtering":[],
 "bi_events":[],
 "url_tracking":[],
 "protocols_support":
   {"quic_udp_block":1}}
```

Fig. 9: Plaintext traffic injection rule to insert a malicious script on login.bank.com located at //attacker.evil/bank.js, and redirect future update queries to https://attacker.evil/mapping

Wajam (i.e., after around an hour, or at boot time), and upon visiting the bank's login page, our malicious script is loaded on the bank's page and is able to manipulate the page's objects, including listening to keystroke events in the username and password fields. No default cross-origin policy would prevent our attack. If the bank's website implemented a CSP, it could be removed by specifying the HTTP header to drop from the server's HTTP response.

Moreover, updates systematically contain the URL of the next update to fetch. Once Wajam downloads an update and caches it to disk, it does not use its hardcoded URL anymore. Hence, the effect of a compromised update is persistent. Our malicious update also instructs Wajam to fetch further updates from our own server, which alleviates the need to repeatedly perform MITM attacks.

A similar attack could be possible against the software update before D18, which would enable an attacker to inject its own binary that will be run with SYSTEM privileges; however, we have not tested this attack. Starting from D18, software updates are fetched over HTTPS and it appears that Wajam properly validates the server certificate, mitigating this attack.

X. Directions for better detection

Security solutions overall fail to statically analyze Wajam's installers and binaries. Unless such binaries are submitted for analysis, possibly because they look suspicious and endpoint solutions may decide to upload them to the antivirus cloud, Wajam can still be installed on most user systems due to its daily polymorphic installer. We identified simple fingerprints that could hint at an infection, either from the host or network activities. First, Wajam registers an installed product on the system using either a known registry key or known names (e.g., SearchAwesome), which could be blacklisted. Then, it tries to add its installation folder and network driver as exceptions for Windows Defender, which could help locate Wajam's binaries. Moreover, Wajam uses a long but bounded list of domains so far. A simple domain blacklist would

TABLE IX: Fingerprints for Wajam-issued leaf certificates (SQL regular expression syntax)

Operator	**Issuer Distinguished Name**
=	emailAddress=info@wajam.com, OU=Created by http://www.wajam.com, O=WajamInternetEnhancer, CN=Wajam_root_cer
=	emailAddress=info@technologiesainturbain.com, OU=Created by http://www.technologiesainturbain.com, O=WajamInternetEnhancer, CN=WNetEnhancer_root_cer
=	emailAddress=info@technologievanhorne.com, OU=Created by http://www.technologievanhorne.com, O=WajamInternetEnhancer, CN=WaNetworkEnhancer_root_cer
REGEXP	ˆemailAddress=info@technologie.+\.com, C=EN, CN=[0-9a-f]{16}$
REGEXP	ˆC=EN, CN=[0-9a-f]{16} 2$
REGEXP	ˆC=EN, CN=(([YZMNO][WTmj2zGD][FEJINMRQVUZYBAdchglk][h-mw-z0-5]){4}) 2$

prevent Wajam to communicate with its servers and leak private information. Samples communicating in plaintext can further be fingerprinted due to the URL patterns and type of data sent, i.e., list of installed programs. Later samples that leverage HTTPS at install-time and later to fetch updates could still be fingerprinted due to known domains present in the TLS SNI extension, or simply by blacklisting corresponding IP addresses. Since daily variants of Wajam are served from known domains at known locations, it is possible for security solutions to constantly monitor these servers for new samples and create corresponding signatures earlier. When a new system driver is installed, additional verifications could quickly find out Wajam's network driver as it is signed with a certificate for one of the known domains.

Finally, we were able to build fingerprints for Wajam-issued certificates, shown in Table IX. It is possible to match a leaf certificate's distinguished name (DN) with our patterns to confirm whether it has been issued by Wajam. They may be particularly relevant if integrated into browsers to warn users. Chrome already detects well-known software performing MITM to alert users of possible misconfigurations or unwanted interceptions [22].

The use of ProtocolFilters can also be fingerprinted by the files and folder structure it sets up. Online searches for *malware "2.cer"* and *"SSL" "cert.db" "*.cer"* yield several forum discussions about infections, e.g., Win.Dropper.Mikey, iTranslator, ContentProtector, SearchProtectToolbar, GSafe, OtherSearch, and even an Indian security solution (Protegent Total Security). Most of these applications likely use ProtocolFilters' default key, as we could verify for Protegent, and hence make end users vulnerable to MITM attacks, in addition to being a nuisance. More work is needed to understand the extent of the use of this interception SDK.

XI. Future work and conclusion

We recently found that the OtherSearch adware (also known as FlowSurf/CleverAdds) shares very similar obfuscation, evasion and steganography techniques with Wajam, sometimes in a more or less advanced way. For instance, it installs a rootkit to hide itself, but does not leak the browser histories. We could not find an organizational connection between Wajam and OtherSearch, thus suggesting that both may leverage a common third-party obfuscation framework. We plan to fingerprint Wajam/OtherSearch's obfuscation framework and leverage malware databases to discover other related samples.

Compared to previous studies on adware, we provide an in-depth look into a wide-spread strain in particular, and provide insights into the business and technical evolutions. We uncovered advanced anti-analysis and antivirus evasion techniques. We also identified important security risks and privacy leakages. Considering the huge amount of private data collected by its operators, and the number of installations it made, it is surprising that nobody looked at it more closely. Perhaps, "adware" applications may not seem much attractive. However, we hope that the security community will recognize the need for better scrutiny of such applications, and more generally PUPs, as they tend to survive and evolve into more robust variants that should be interesting to malware researchers.

References

[1] "Lenovo PCs ship with man-in-the-middle adware that breaks HTTPS connections," news article (Feb. 19, 2015). http://arstechnica.com/security/2015/02/lenovo-pcs-ship-with-man-in-the-middle-adware-that-breaks-https-connections/.

[2] "PrivDog SSL compromise potentially worse than Superfish," news article (Apr. 24, 2015). http://www.computerweekly.com/news/2240241126/PrivDog-SSL-compromise-potentially-worse-than-Superfish.

[3] 0xd4d, "de4dot," https://github.com/0xd4d/de4dot.

[4] D. Alto, "7-zip 15.10 no longer decompiles NSIS script," reply to forum post (Dec. 7, 2015). https://sourceforge.net/p/sevenzip/discussion/45797/thread/5d10a376/#6e1d/3fa3/6840/fe9c.

[5] D. Andriesse, C. Rossow, B. Stone-Gross, D. Plohmann, and H. Bos, "Highly resilient peer-to-peer botnets are here: An analysis of gameover Zeus," in *MALWARE'13*, Fajardo, PR, USA, Oct. 2013.

[6] Anonymous, "Keeping history saved for longer than 3 months," 2015, chrome issue 500239. https://bugs.chromium.org/p/chromium/issues/detail?id=500239.

[7] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the Mirai botnet," in *USENIX Security Symposium*, Vancouver, BC, Canada, Aug. 2017.

[8] M.-L. Archambault, S. Giroux, and A.-P. Paquet, "Method and system for aggregating searchable web content from a plurality of social networks and presenting search results," July 2013, US Patent 2013/0179427 A1.

[9] BankInfoSecurity.com, "Zeus banking trojan spawn: Alive and kicking," 2017, news article (Nov. 24, 2017). https://www.bankinfosecurity.com/zeus-banking-trojan-spawn-alive-kicking-a-10471.

[10] D. Bestuzhev, "Steganography or encryption in bankers?" Kaspersky Labs blog article (Nov. 10, 2011). https://securelist.com/steganography-or-encryption-in-bankers-11/31650/.

[11] H. Binsalleeh, T. Ormerod, A. Boukhtouta, P. Sinha, A. M. Youssef, M. Debbabi, and L. Wang, "On the analysis of the Zeus botnet crimeware toolkit," in *PST'10*, Ottawa, ON, Canada, Dec. 2010.

[12] BleepingComputer.com, "Wajam and WNetEnhance Removal Guide," 2015, tech article (Apr. 1, 2015). https://www.bleepingcomputer.com/virus-removal/remove-wajam-ads-wnetenhance.

[13] H. Böck, "More TLS Man-in-the-Middle failures - Adguard, Privdog again and ProtocolFilters.dll," blog article (Aug. 13, 2015). https://blog.hboeck.de/archives/874-More-TLS-Man-in-the-Middle-failures-Adguard,-Privdog-again-and-ProtocolFilters.dll.html.

[14] C. Brook, "Mirai IoT botnet co-authors plead guilty," 2017, news article (Dec. 14, 2017). https://digitalguardian.com/blog/mirai-iot-botnet-co-authors-plead-guilty.

[15] BullGuard, "Antivirus settings," https://www.bullguard.com/support/product-guides/internet-security/guides-for-current-version/main/antivirus-settings.aspx.

[16] S. Chimakurthi, “Malware hides in installer to avoid detection,” mcAfee blug article (Aug. 25, 2016). https://blogs.mcafee.com/mcafee-labs/malware-hides-in-installer-to-avoid-detection/.

[17] Z. Clark, “Komodia rootkit findings,” 2015, https://gist.github.com/Wack0/f865ef369eb8c23ee028.

[18] CrowdStrike, “Hybrid Analysis,” https://www.hybrid-analysis.com/.

[19] X. de Carné de Carnavalet and M. Mannan, “Killed by proxy: Analyzing client-end TLS interception software,” in *NDSS’16*, San Diego, CA, USA, Feb. 2016.

[20] ESET, “What is a potentially unwanted application or potentially unwanted content?” 2018, ESET Knowledge Base ID: KB2629. https://support.eset.com/kb2629/.

[21] B. N. Giri, P. P. Ramagopal, and V. Thomas, “Alerting the presence of bundled software during an installation,” Nov. 2016, US Patent 2016/0328223 A1.

[22] Google, “SSL error assistant,” chromium source code. https://cs.chromium.org/chromium/src/chrome/browser/resources/ssl/ssl_error_assistant/ssl_error_assistant.asciipb.

[23] G. M. Graff, “Inside the hunt for Russia’s most notorious hacker,” 2017, news article (Mar. 21, 2017). https://www.wired.com/2017/03/russian-hacker-spy-botnet/.

[24] HowToGeek.com, “Here’s what happens when you install the top 10 Download.com apps,” 2017, tech. article (Apr. 3, 2017. https://www.howtogeek.com/198622/heres-what-happens-when-you-install-the-top-10-download.com-apps/).

[25] IOActive, “Reversal and analysis of zeus and spyeye banking trojans,” 2012, technical White Paper. https://ioactive.com/pdfs/ZeusSpyEyeBankingTrojanAnalysis.pdf.

[26] S. Ishimaru, “Old malware tricks to bypass detection in the age of big data,” Kaspersky Labs blog article (Apr. 13, 2017). https://securelist.com/old-malware-tricks-to-bypass-detection-in-the-age-of-big-data/78010/.

[27] J. Jones, “The state of web exploit kits,” in *BlackHat’12*, Las Vegas, NV, USA, July 2012.

[28] Kaspersky, “Not-a-virus: What is it?” 2017, blog article (Aug. 21, 2017). https://www.kaspersky.com/blog/not-a-virus/18015/.

[29] A. Kharraz, W. K. Robertson, D. Balzarotti, L. Bilge, and E. Kirda, “Cutting the gordian knot: A look under the hood of ransomware attacks,” in *DIMVA’15*, Milan, Italy, July 2015.

[30] M. Kiros, “FLIRT Signature File Database,” https://github.com/michaelkiros/FLIRTDB.

[31] P. Kotzias, L. Bilge, and J. Caballero, “Measuring PUP prevalence and PUP distribution through pay-per-install services,” in *USENIX Security Symposium*, Austin, TX, USA, Aug. 2016.

[32] P. Kotzias, S. Matic, R. Rivera, and J. Caballero, “Certified PUP: abuse in authenticode code signing,” in *CCS’15*, Denver, CO, USA, Oct. 2015.

[33] B. Krebs, “SpyEye Targets Opera, Google Chrome Users,” Apr. 2011, blog article (Apr. 26 2011). https://krebsonsecurity.com/2011/04/spyeye-targets-opera-google-chrome-users/.

[34] D. Kumar, M. Bailey, Z. Wang, M. Hyder, J. Dickinson, G. Beck, D. Adrian, J. Mason, Z. Durumeric, and J. A. Halderman, “Tracking certificate misissuance in the wild,” in *IEEE S&P*, San Francisco, CA, US, May 2018.

[35] Linux man page, “clamd.conf(5).”

[36] G. D. Maio, A. Kapravelos, Y. Shoshitaishvili, C. Kruegel, and G. Vigna, “Pexy: The other side of exploit kits,” in *DIMVA’14*, Egham, UK, July 2014.

[37] Malekal, “Liste Malware,” http://malwaredb.malekal.com/index.php?malware=wajam.

[38] Mandiant, “APT1 – Exposing one of China’s cyber espionage units,” 2013, https://www.fireeye.com/content/dam/fireeye-www/services/pdfs/mandiant-apt1-report.pdf.

[39] T. Marques, “PNG embedded - malicious payload hidden in a PNG file,” Kaspersky Labs blog article (Mar. 24, 2016). https://securelist.com/png-embedded-malicious-payload-hidden-in-a-png-file/74297/.

[40] A. Nappa, M. Z. Rafique, and J. Caballero, “Driving in the cloud: An analysis of drive-by download operations and abuse reporting,” in *DIMVA’2013*, Berlin, Germany, July 2013.

[41] NSIS Wiki, “Can I decompile an existing installer?” http://nsis.sourceforge.net/Can_I_decompile_an_existing_installer.

[42] Office of the Privacy Commissioner of Canada, “Canadian adware developer Wajam Internet Technologies Inc. breaches multiple provisions of PIPEDA,” Tech. Rep. #2017-002, Aug. 2017, https://www.priv.gc.ca/en/opc-actions-and-decisions/investigations/investigations-into-businesses/2017/pipeda-2017-002/.

[43] PreEmptive Solutions, “Dotfuscator — .NET Obfuscator & much more,” https://www.preemptive.com/products/dotfuscator/overview.

[44] Progress Software, “What is Telerik FiddlerCore?” https://www.telerik.com/fiddler/fiddlercore.

[45] Quebec Government, “Registraire des entreprises,” http://www.registreentreprises.gouv.qc.ca.

[46] E. Rescorla and RTFM, Inc., “RFC 2818: HTTP Over TLS,” 2000, RFC 2818 (Informational Track).

[47] E. Roman, “Chrome no longer accepts certificates that fallback to common name,” 2017, chromium issue 700595 (Mar. 11, 2017). https://bugs.chromium.org/p/chromium/issues/detail?id=700595&desc=2.

[48] M. Schiffman, “A brief history of malware obfuscation: Part 2 of 2,” Cisco blog article (Fev. 22, 2010). https://blogs.cisco.com/security/a_brief_history_of_malware_obfuscation_part_2_of_2.

[49] S. Shah and D. Cole, “Spyware/Adware – The quest for consumer desktops & how it went wrong,” in *BlackHat’05 Japan*, Tokyo, Japan, Oct. 2015.

[50] S. Shin and G. Gu, “Conficker and beyond: a large-scale empirical study,” in *ACSAC’10*, Austin, TX, USA, Dec. 2010.

[51] V. Sidorov, “Network filtering toolkit,” http://netfiltersdk.com/.

[52] ——, “ProtocolFilters history,” http://netfiltersdk.com/protocolfilters_history.html.

[53] A. K. Sood and R. Bansal, “Prosecting the Citadel botnet - revealing the dominance of the zeus descendent,” 2014, white paper (Sep. 8 2014). https://www.virusbulletin.com/uploads/pdf/magazine/2014/vb201409-Citadel.pdf.

[54] P. Soucy, “Wajam,” 2015, blog post (Aug. 21, 2015). http://dev-smart.com/wajam/.

[55] SourceForge.net, “NSIS download statistics,” https://sourceforge.net/projects/nsis/files/NSIS%203/stats/timeline.

[56] E. H. Spafford, “The Internet worm program: An analysis,” *SIGCOMM Comput. Commun. Rev.*, vol. 19, no. 1, pp. 17–57, Jan. 1989.

[57] T. Spring, “Where have all the exploit kits gone?” 2017, news article (Mar. 15, 2017). https://threatpost.com/where-have-all-the-exploit-kits-gone/124241/.

[58] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. A. Kemmerer, C. Kruegel, and G. Vigna, “Your botnet is my botnet: analysis of a botnet takeover,” in *CCS’09*, Chicago, IL, USA, Nov. 2009.

[59] Symantec, “W32.Stuxnet Dossier,” 2011, white paper (Feb. 2011). https://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf.

[60] A. Szekely, “NSIS (Nullsoft Scriptable Install System),” http://nsis.sourceforge.net/Main_Page.

[61] TheGuardian.com, “In millions of Windows, the perfect Storm is gathering,” news article (Oct. 21, 2007). https://www.theguardian.com/business/2007/oct/21/1.

[62] K. Thomas, J. A. E. Crespo, R. Rasti, J.-M. Picod, C. Phillips, M.-A. Decoste, C. Sharp, F. Tirelo, A. Tofigh, M.-A. Courteau, L. Ballard, R. Shield, N. Jagpal, M. A. Rajab, P. Mavrommatis, N. Provos, E. Bursztein, and D. McCoy, “Investigating commercial pay-per-install and the distribution of unwanted software,” in *USENIX Security Symposium*, Austin, TX, USA, Aug. 2016.

[63] W. Wong and M. Stamp, “Hunting for metamorphic engines,” *Journal in Computer Virology*, vol. 2, no. 3, pp. 211–229, 2006.

[64] WordPress, “A live look at activity across WordPress.com,” https://wordpress.com/activity/.

[65] x64dbg, “An open-source x64/x32 debugger for windows,” https://x64dbg.com/.

APPENDICES

TABLE X: List of 248 domains that appear to belong or have belonged to Wajam

adrienprovenchertechnology.com
armandlamoureuxtechnology.com
autodownload.net
autotelechargement.net
barachoistechnology.com
beaubourgtechnology.com
bellechassetechnology.com
bernardtechnology.com
berritechnology.com
boisseleautechnology.com
boissytechnology.com
bombarderietechnology.com
bouloitechnology.com
bourassatechnology.com
boussactechnology.com
calmonttechnology.com
carmenbienvenuetechnology.com
cartiertechnology.com
chabaneltechnology.com
chabottechnology.com
chamoilletechnology.com
champlaintechnology.com
charlevoixtechnology.com
chaumonttechnology.com
chavanactechnology.com
cherriertechnology.com
chestertontechnology.com
clairavauxtechnology.com
colonialetechnology.com
coolappinstaler.com
cremazietechnology.com
datawestsoftware.com
dateandtimesync.com
dkbsoftware.com
download-flv.com
download-install.com
downloadmngr.com
downlowd.com
downlowd.org
drapeautechnology.com
emersontechnology.com
fastappinstall.com
fastfreeinstall.com
fastnfreeinstall.com
ferronnerietechnology.com
file-extract.com
fileextractor.net
fileopens.com
flvplayer-hd.com
freeappdownloader.com
freeusip.mobi
fullumtechnology.com
fulmartechnology.com
garfieldtechnology.com
garniertechnology.com
glencoetechnology.com
grendontechnology.com
henaulttechnology.com
hutchisontechnology.com
insta-download.com
install-apps.com
installateurdappscool.com
installationdappgratuite.com
installationrapideetgratuite.com
installationrapidegratuite.com
installsofttech.com
ios-vpn.com
jarbontechnology.com
jeanlesagetechnology.com
jolicoeurtechnology.com
kingswoodtechnology.com
labroyetechnology.com
laubeyrietechnology.com
launtontechnology.com
laurendeautechnology.com
lauriertechnology.com
main-social2search.netdna-ssl.com
mandartechnology.com
manillertechnology.com
mansactechnology.com
media-c9hg3zwqygdshhtrps.stackpathdns.com
mercilletechnology.com
mertontechnology.com
mileendsoft.com
monestiertechnology.com
monroetechnology.com
montorgueiltechnology.com
montroziertechnology.com
mounactechnology.com
nouaillactechnology.com
pagerecherche.com
papineautechnology.com
payennetechnology.com
pelletiertechnology.com
piddingtontechnology.com
pillactechnology.com
plateau-technologies.com
preverttechnology.com
quaintontechnology.com
racheltechnology.com
rambuteautechnology.com
rivolettechnology.com
sagardtechnology.com
saintdominiquetechnology.com
saintjosephtechnology.com
sainturbaintechnology.com
searchawesome.net
searchpage.com
sentiertechnology.com
shermantechnology.com
sirwilfridlauriertechnology.com
snowdontechnology.com
socialwebsearch.co
sommerytechnology.com
superdownloads.com
supertelechargements.com
technologieadrienprovencher.com
technologiearmandlamoureux.com
technologiebarachois.com
technologiebeaubourg.com
technologiebellechasse.com
technologiebeloeil.com
technologiebernard.com
technologieberri.com
technologieboisseleau.com
technologieboissy.com
technologiebombarderie.com
technologiebouloi.com
technologiebourassa.com
technologieboussac.com
technologiecalmont.com
technologiecarmenbienvenue.com
technologiecartier.com
technologiechabanel.com
technologiechabot.com
technologiechamoille.com
technologiechamplain.com
technologiecharlevoix.com
technologiechaumont.com
technologiechavanac.com
technologiecherrier.com
technologiechesterton.com
technologieclairavaux.com
technologiecoloniale.com
technologiecremazie.com
technologiedrapeau.com
technologieemerson.com
technologieferronnerie.com
technologieflagstick.com
technologiefullum.com
technologiefulmar.com
technologiegarfield.com
technologiegarnier.com
technologieglencoe.com
technologiegoyer.com
technologiegrendon.com
technologiehenault.com
technologiehutchison.com
technologiejarbon.com
technologiejeanlesage.com
technologiejolicoeur.com
technologiekingswood.com
technologielabroye.com
technologielangelier.com
technologielaubeyrie.com
technologielaunton.com
technologielaurendeau.com
technologielaurier.com
technologiemandar.com
technologiemaniller.com
technologiemansac.com
technologiemercille.com
technologiemerton.com
technologiemonestier.com
technologiemonroe.com
technologiemontorgueil.com
technologiemontroyal.com
technologiemontrozier.com
technologiemounac.com
technologienouaillac.com
technologiepapineau.com
technologiepayenne.com
technologiepelletier.com
technologiepiddington.com
technologiepillac.com
technologieprevert.com
technologiequainton.com
technologierachel.com
technologierambuteau.com
technologierivolet.com
technologierutherford.com
technologiesagard.com
technologiesaintdenis.com
technologiesaintdominique.com
technologiesaintjoseph.com
technologiesaintlaurent.com
technologiesainturbain.com
technologiesentier.com
technologiesherman.com
technologiesirwilfridlaurier.com
technologiesnowdon.com
technologiesommery.com
technologiestdenis.com
technologiestlaurent.com
technologieterusse.com
technologiethorel.com
technologietoleto.com
technologietravassac.com
technologietreeland.com
technologietrudeau.com
technologieturenne.com
technologievanhorne.com
technologievanoise.com
technologievassy.com
technologieviau.com
technologievimy.com
technologievouillon.com
technologiewendlebury.com
technologiewilson.com
technologiewoodham.com
technologiewoodstream.com
technologiewotton.com
technologieyvonlheureux.com
technologyflagstick.com
technologyrutherford.com
technologytreeland.com
technologywilson.com
technologywoodstream.com
terussetechnology.com
thoreltechnology.com
toletotechnology.com
travassactechnology.com
trudeautechnology.com
turennetechnology.com
vanhornetechnology.com
vanoisetechnology.com
vassytechnology.com
viautechnology.com
videos-conversion.com
vouillontechnology.com
vpn-free.mobi
wajam-download.com
wajam.com
wendleburytechnology.com
woodhamtechnology.com
wottontechnology.com
youcansearch.net
yvonlheureuxtechnology.com

TABLE XI: Hashes of the 52 samples we collected

ID	MD5	SHA1	SHA256
A1	225ccdcfe5625795647043679cb77112	3bd8f8845df04ac40b78da0fb9ecd7205514db62	96fafff2e4076a0a0fe2c9d151f37441507bf3c0dc4b761c66f65cbbc94c823c
A2	20e274902bd0249c68f756694d43e8eb	d77aa518dbfb56782ca8efc030e09767a3c39fcd	9a3c8fdc8cd34be72d24b1d3f7f52078469c0f5e26ae373df18a871fd020fb08
A3	5d2b2eb701b38066318dcb254f2400a1	a2853d27c2378b9065deb3c69c5cf608f7c2ee1d	84aaf3531cde8a4ab67ca5d971039a12bc3010d59729f922e816eca5b12c28c1
A4	f314d12cbd75002f6249d2f50cdd2ce2	9876e0dfd6348285c99f2593e9cdaca7b91e3590	c5b2ad40c663f603e10ee53281bdf611704db441efbcec507dd46727bd245c6a
B1	c80db840ac2597b988e1c88b5d7015f2	343f9ab838ed64e862bbf8ff0ce723222ca97f90	ce755f50d228d92aca01a54b81bd534f188a93e74c73160e008e7cc81480bba0
B2	572b59e1225fc16a1478e7ff27919278	1de9bb908915f24730153ef5bbddd1e5467a034b	2673b0bb4d705a8cf29aeb86079485c51bab0aedaae8c960afe0c38efa7b151b
C1	2a791c466a3fe634b642ac636c31ae75	c291d5bae79149a2361daa69a39c29c23c564092	358633ea6e06f81de0af1c8ba2a774439c39073de012a0a50be28823a6d0f951
C2	68079e4133596ab3f4894353b572a476	5ee5373a55c4fdcaa4e1f2d62121da38aea6a8a5	12baaf1dc8d4abd03270d942e7498b7588480dc70305ee9a3e82870b6df4978f
C3	28709615566405e17290e59990850635	019da3fdb927bb47635df65f0d108b29d735eac4	023f680d7475da1fa0d0f2125c88db25d046720986a84e7075eef12734b37b95
B3	97e8f6b46de9e1e3e312de78ed90e17f	86e6c43ce0811930e7ea760546b1b1a933fee637	183d4b92d6b048bbdc871df240bb5de8d3343e50fc93dc363ef6fbfc892f107d
C4	49a6c8adb892e0423e27396ceb4171aa	6964f4c2ded64135728b160b19a1e6491bf8ae6e	4913597301b2f87401e12b33a5be3a8d07c07b0152c3769d327478e3ea89a416
C5	fec0f4a9a37069cd1bd8b32b9b05bd7d	56b6b6b8172ab418cfe1b3316f67bdbc71e25db8	68bcb81fd0bf65694c624224eb33e93c7ac6816469edb91ea61de2218734df39
C6	cb9a30d0aaef0335b4f8b4363bfb68a2	d4081867928b00a4d81d36b33c16185e16030684	b5e0dd43c424eb7e982b3c89e5b191864496464fb15482e3986d2198c0db5910
C7	1bc90276e8686b8ce545b22a1e8b19d3	9e64d510e3b624a1c13586053e9c59b6f66c30f4	45bc45bf74fa39e9f1c5a511a1858c984cec8c2c4b6a83521e918c08c68413cb
B4	45b1d58c23f15c841318abe1a786fbf8	779fdddbaa916eb54dea1ab9e51649f891a00d78	b62ac7510f58751d51a28077b81981c99aee512f4976e04c39f4e7f9efeacd09
B5	775367aad190fdb847f2628a47584c1b	251e2a0530b3eadc1543548ffd829bb38ce2f6b3	11b7ef63d462424ebd04b41258f75df3d936ecfac91cccbb1c930d63ead3573a
C8	c1d044237977df5bb779152a5c7bb941	6182744b23d1900ece5e3ffdfeb2aaeda3451722	78f1888d1d918b1924f02ead3fe00d9546aaf8b4db17892807fbbdd6df80ccbe
C9	ec0b9463a4564c63bab76985aff98f06	c46075248e528ed418b3a595a77bc40298464b08	936f27d444bac46c4012cb4fcf6093b34952e314cb68d780de1cdd510ecac697
C10	21624ae93359e523f6d69f52109e69bb	f99e444107a822f22c0fdcd7b3ff0f57fc211507	ac3b6bab836a308dc68584364ef5cca3b747ce46e34de3fb1d235d45761e877f
B6	e4272e1164e458b400e39de26484d5c5	73538adff00910979b3cce6434891e182e36b942	6dd2651f5d62e4787354a0c04fac98d2d2a586439f623ad8b64e4ec6e7c97f82
C11	b67716e043f53e1f4af4cf318b4c5a03	a0d951243aa36511f7327c59d1ce2e098622d814	aeae711f64f921eb7b86864a1c09a00cc93cc1afed0f346150c0ada995cf93bd
C12	c276a93d4faa48ba9305ef43b4724200	7913ae29483a18dd22c451d28901f1fa9401a130	5b2e66ba34b66e12ea4668006bac9a6556b4073b4366747254ce3965627a60bd
C13	f4dc7103bdb0752b8d030b090fdfa475	8fc5553247ff7e47533ebdd73760d2a134fea257	f2f5b2608d1a0a68dd21e67055bc80dd5e214abefdb4c37ec866d12c503b44c4
C14	da4370da224a43456960bcff2c4b44ad	d9d0fe82e40cc5528da1d4669325991a65a9c4a1	1013634048a79f181b191995276230a05532a3ea5fba8d638cd265f44c464bd4
D1	f7d2c05eccd522764167632b3a8dc122	0df370fdd35cee653b14418c858dba039a141479	c539963dd900a7771d33844cc48730d47cc510bdf1a7dea429e08c3bf060d393
C15	d31b23e32013385b1554c59cc02bf3d3	626b486c53abebc8fc412ea7af05f7d8bc0e92de	c15ed12cc339b736ec73e6b710e8f7b646bc58914f888026558bd763f2feba1f
D2	8aefc1f5ad40155241aa87db4ff8eb6a	086d740ee131afc4c4408dcecbfab29712a9b8e0	7f17ac7ea17f54a1961fb3aabd3ca16a8180815641319d155a649c9c6e8a2128
C16	a1e84cf06ed6b583103120f36b53fcd1	0b6294dba9cfe4e2a79c6117b20d0475fb787d94	f51e192eb397c6df169713f5ec327d92bbfe9436ab169fed0e07f6865049f59f
D3	80e5c2fc7f0637df90f39204eebac932	f071e2f5a25a79d48c8cf8232d7a7775ecf016d6	89760cc89415efcf270090b2469afe0f6cb64bc3476936a1521b5055a4b71400
C17	f472a9dad90df05492a01135307bb2cf	58f14d93aa16f7fd1ee3930323ad39537eb974d5	ce590c460a34f946944228abb2e964505f75eedee8680998aa532ee93044b09a
D4	990ce267cff603a00e081a473b234f2a	a1dd7e9d896214da1db85e98b83b1546d4f1c1e7	3c76908cf7dd9a8072724674ddcc64dd81f94e47a3bb38a669ecb26b5f95203a
C18	34fd34fad31242a57fda9284b4cde461	df6f20e89c06e8fd1e2cca3a06ef6da265e104c6	e42b339c4b12abd413ddef0051fb22bdd2ec0a0b6969a35e20c68c7e353d7f94
D5	0bc19d446ab54343afed0f8493cbadf9	e3930757733213461cbbc58e4fc45dc2b87529fa	9cdba6d9dc5cc505425217e0b4990ea60ef80de2c7f774f5dc760b3a4efe504b
D6	9936ddbcdc9828df9dc132508022231a	49f90582088d49f02ea2ae93f40fa12d4396c679	dc9fff7dff59a10a8717188655e7b8e39e05d522363c7d1522be215d2548bf67
D7	759ad7685285f7d6dbf4e29a0f44d6aa	855167f21cdd40af0917385975680f5b7948c6d6	0c2c5a9b31b6ecbae20ccdc89971db3ab9910f165605c5969440a24fe718ea65
D8	f3c774ee3a87b6c1b628b1f28e3e130a	e91c12ec77fc2c11369014b9deb8055e7d51a320	b77f48c56903890e73c4348263beb22970978d6106188e6859599249c8dc70e7
D9	855ce542c7fc7da18f2696784ed7a181	0674e654524a7862247ebc75f3c786a17927d6a6	df00d50b957c684da709de2704252ff03c118f6dfe385c4e708a7d95187269b5
C19	9d64162edf85cd58e177aa7c444c297b	728cfab0d37503b1c1156929c2a8106a5b663328	248ecf25fdc624a3bb4eb27aa60d07a541c4e462e94b84ed86d006cd03450f60
D10	8961634b77b478bb85429b86780cfe28	90f536d7631548d980898d2473c5c46b93131022	e637d1c86ec77036d8ca43f69296543e51175e8294bc26ed4acfbec87beb8c76
D11	ad1425976256881e037bd9b26524f1c8	0b8a401e904b310c17315cb9bce49eebc5c69ede	0d0bfecb1d5e72773532398319e8eefc7bc778a88e2806ebb005864f04096aa8
D12	0d3a7053f911d368f80990062e82d20a	6677af3eed2b512d297d973f65d4c8db98c4df1d	97ef8100f215d2d2603d5cb780f37ef715a486e7847613427b7c0b481e9de194
D13	f5c0e283062bbd50799dca72b025d228	63f1af45a4975cf62d0173dd3f0ddaa7103bc471	45bc3bf77b741b508bb480a4fb7c49d4d04e0a5fd8f28c93f27f013a087789f1
D14	7460f80448708135e62afab652076e25	f37d15a89a6ea4d30be52668d50dc76f64a59e6e	dfb6ba9cb3a53357e13de282ba2d3f001ed4f0b54ccb582d951afcc33f3fe303
D15	d7238809f6c14e663526fc0a32a14413	c902582e8971edc2f721cbf0b54cfb0c12d19c39	379e3863e863431ff9b51e8f0966416cc18d55c4e54f2fc1a7d885b9fb8dfac6
D16	8240e2fb284d278846c814008ab88546	bf0e5e0feb5db9fd940a6b75ee5cd2e841a67cbc	aaf3a711ca3fdd538f51cb970f98d8b8a6414b5c9a1b3806c2a6c6ad43a8268c
D17	019df633f66910063a5ae8db6cb20ce9	e53b2d1742f46467ad45c9aac14d30c98574b57b	1b9975d97c9b4f622362e58acaf11b906a7a13d23f2ca058be0dfa8f464e3dbc
D18	fd9628d2187a886cb2c47348db012a5a	b2d8f09d23ef79ac3e390ef493b70d4a7f632dd7	20528bac0f54eb5c6f2ba6cb75401697251fd8624dc36375205d949524f8f77e
D19	6751963e0862ac6bb94d66ff5f501977	074fad99fce9babe7925f144be4885e511eff5bc	8396d0099ce9c6c132ffc924f7cfa5fd29200a383ab46e2ed92d386104de63cd
D20	e4b85ab5cb039fc24d56946728b4213e	11152f9fe3090c54f3a5c223bcb01da02766605c	e13363896eef7fae729b3766fcae20354bd9f787227b893ce3a4e7feab83836b
D21	f11b060a7092cedb7251509d4ecf0f14	b88e804aa94139d0aa628c9b141cc72e6128ce6e	787c4b5284c7d55d9510f519a2af6a5f085ff75b75fc273ab6134715a8d2633f
D22	ae3c32975f4ec3d1e2dd0dcd7f4636d4	0a63e46309dbfa2b13d47b488753b7c64b2896e2	c75009997a38afb450d73388cf42782ec4074c64f7acec3b2eb5dff89b265498
D23	aaf7ea90dfe3b22ad42b35727bf8bd20	e4f855747da8352969c8bf217657f7bb78332fe6	c4d96ff7de37715911b165711c034fc1159940fc2d110696bb481cd39d60a2f9