



Decoding the Kodi Ecosystem

YUNMING XIAO, Northwestern University

MATTEO VARVELLO, Nokia Bell Labs

MARC WARRIOR and ALEKSANDAR KUZMANOVIC, Northwestern University

2

Free and open-source media centers are experiencing a boom in popularity for the convenience they offer users seeking to remotely consume digital content. Kodi is today's most popular home media center, with millions of users worldwide. Kodi's popularity derives from its ability to centralize the sheer amount of media content available on the Web, both *free* and *copyrighted*. Researchers have been hinting at potential security concerns around Kodi, due to *add-ons* injecting unwanted content as well as user settings linked with security holes. Motivated by these observations, this article conducts the first comprehensive analysis of the Kodi ecosystem: 15,000 Kodi users from 104 countries, 11,000 unique add-ons, and data collected over 9 months.

Our work makes three important contributions. Our first contribution is that we build "crawling" software (de-Kodi) which can automatically install a Kodi add-on, explore its menu, and locate (video) content. This is challenging for two main reasons. First, Kodi largely relies on visual information and user input which intrinsically complicates automation. Second, the potential sheer size of this ecosystem (i.e., the number of available add-ons) requires a highly scalable crawling solution. Our second contribution is that we develop a solution to discover Kodi add-ons. Our solution combines Web crawling of popular websites where Kodi add-ons are published (LazyKodi and GitHub) and SafeKodi, a Kodi add-on we have developed which leverages the help of Kodi users to learn which add-ons are used in the wild and, in return, offers information about how *safe* these add-ons are, e.g., do they track user activity or contact sketchy URLs/IP addresses. Our third contribution is a classifier to passively detect Kodi traffic and add-on usage in the wild.

Our analysis of the Kodi ecosystem reveals the following findings. We find that most installed add-ons are *unofficial* but *safe* to use. Still, 78% of the users have installed at least one *unsafe* add-on, and even worse, such add-ons are among the most popular. In response to the information offered by SafeKodi, one-third of the users reacted by disabling some of their add-ons. However, the majority of users ignored our warnings for several months attracted by the content such unsafe add-ons have to offer. Last but not least, we show that Kodi's auto-update, a feature active for 97.6% of SafeKodi users, makes Kodi users easily identifiable by their ISPs. While passively identifying which Kodi add-on is in use is, as expected, much harder, we also find that many unofficial add-ons do not use HTTPS yet, making their passive detection straightforward.¹

CCS Concepts: • **Networks** → *Network measurement*; **Network measurement**; **Network security**; **Network privacy and anonymity**; • **Theory of computation** → *Theory and algorithms for application domains*; • **Mathematics of computing** → *Mathematical optimization*; • **Computing methodologies**

¹This article is a major extension of our previous article "De-Kodi: Understanding the Kodi Ecosystem" which was published at the Web conference (WWW'20).

Authors' addresses: Y. Xiao, M. Warrior, and A. Kuzmanovic, Northwestern University, Computer Science Department, 2233 Tech Drive, Seeley Mudd – 3rd Floor, Evanston, IL 60201; emails: {yunming.xiao, warrior}@u.northwestern.edu, akuzma@northwestern.edu; M. Varvello, Nokia Bell Labs, 600 Mountain Ave, New Providence, NJ 07974; email: matteo.varvello@nokia.com.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2023 Association for Computing Machinery.

1559-1131/2023/01-ART2 \$15.00

<https://doi.org/10.1145/3563700>

→ **Modeling methodologies; Model verification and validation; • Software and its engineering** → **Software libraries and repositories; • Information systems** → **Traffic analysis; Crowdsourcing; Evaluation of retrieval results; • Social and professional topics** → **User characteristics; • Security and privacy** → **Web application security;**

Additional Key Words and Phrases: Kodi, crawling, crowdsourcing, measurement

ACM Reference format:

Yunming Xiao, Matteo Varvello, Marc Warrior, and Aleksandar Kuzmanovic. 2023. Decoding the Kodi Ecosystem. *ACM Trans. Web* 17, 1, Article 2 (January 2023), 36 pages.

<https://doi.org/10.1145/3563700>

1 INTRODUCTION

Kodi is an open-source entertainment center that allows users to stream both local and remote media content (videos, music, and pictures) on a range of consumer devices, from PCs and set-top boxes to smartphones. Kodi has recently received lots of attention from both content providers, network operators, and the media. This is due both to its growing popularity—according to Sandvine [64], 9% of North American households host at least one Kodi box—as well as its increasing notoriety as the perfect vehicle for illegal content distribution (mostly video) [6, 12]. Despite ongoing warnings and negative press [7–9, 12, 13, 15], tens of millions of households around the world have turned to Kodi as a means of consuming video, audio, and other forms of digital media. The conveniences offered by the apparently free Kodi platform have come at some intangible cost to the public; that cost has been, until now, difficult to measure. Researchers and content owners are in the dark regarding the impact of Kodi’s ecosystem, and Kodi users are at risk of exploitation.

Around Kodi, a whole *ecosystem* has been built with several key players: add-ons (plugins), (content) providers, and users. Kodi users install add-ons via Kodi’s official *repository* (a collection of approved add-ons) or via third-party repositories and sources retrieved on the Web—mostly specialized forums, blogposts, and social media. Installed add-ons provide extra functionalities, such as easy access to remote video libraries from which their desired content can be streamed. This large ecosystem, consisting of millions of users and countless user-developed add-ons, presents a uniquely wide, cross-sectional view of the modern video streaming and various methods of media distribution and consumption.

We aim at studying and quantifying the nature of Kodi’s ecosystem at large through crawling and analyzing Kodi’s *add-ons*, through which media streaming is facilitated. Although the Kodi platform is designed to be convenient for the typical end user, *crawling* Kodi’s add-on ecosystem proves extremely challenging for two key reasons. First, Kodi largely relies on visual information and user input which intrinsically complicates automation. Second, the potential sheer size of this ecosystem requires a highly scalable crawling solution. We tackle the first issue of automation complicated by visual inputs by extending Kodi’s APIs to allow more informed crawling operations, e.g., by interacting with visual elements while tracking the execution path. We tackle the second issue of scalability by leveraging Docker [25] to scale our software while isolating crawler instances from potential malware and/or crashes.

While finishing the crawling software that is capable of automating the tests of Kodi add-ons at a large scale, we come to find that *discovering and locating* Kodi add-ons is even harder, as there exists no global list of Kodi add-ons. We combine two approaches to tackle this challenge. First, we build a scraper that can collect potential Kodi add-ons from the Web (Github, Reddit, and so on) and quickly reduce them to a unique set of actual add-ons. Second, we build *SafeKodi*, a Kodi add-on we have developed that leverages the help of Kodi users to learn which add-ons are used in the wild and, in return, offers information about how *safe* these add-ons are. In particular,

SafeKodi informs the user of suspicious add-ons in their machines, further enhanced with a quick link to disable such add-ons. Participating users agree to anonymously share with our servers the list of add-ons installed on their machine, so as to (1) extend our knowledge of *which* Kodi add-ons currently exist, and (2) shed light on the prevalence of various add-ons. SafeKodi users can also *opt-in* further by sharing more information about their device settings, such as Kodi version, screen resolution, and so on. The above results in a full-fledged crawling system, *De-Kodi*, capable of “*decoding*” the Kodi ecosystem.

We start by *validating* both the performance and the accuracy of De-Kodi. We show that De-Kodi scales linearly with the available underlying hardware resources (three machines located at a North American campus network, in our setup), and that tens of thousands of add-ons can be crawled per day. Further, we show that De-Kodi can effectively explore working add-ons, and quickly discard erroneous, obsolete (50% of add-ons in the ecosystem are more than two years old), or otherwise dysfunctional add-ons which fail to install.

Next, we perform and analyze the Kodi ecosystem solely from the eyes of our Web crawl. De-Kodi successfully tested 5,265 out of 9,146 unique Kodi add-ons (83% more than what is contained in the official Kodi repository) which we discovered via LazyKodi, a well-known and actively maintained Kodi add-on aggregator, as well as Reddit [37] and GitHub [30], which are known for attracting both Kodi users and developers. We then complement the above results with the new set of add-ons discovered via SafeKodi. We launched SafeKodi on February 24, 2020. Coined as Kodi’s first “antivirus” by several news outlets [1, 5], SafeKodi has been installed 15,768 times across 104 countries worldwide. Coupling add-ons crowdsourced by SafeKodi with Web crawls, we discovered and tested 11,112 unique add-ons over 7 months.

Finally, we leverage this wealth of information collected to answer the following question: *is it possible to detect Kodi traffic in the wild?* This question is important for Kodi users who aim at protecting their privacy, but also for law enforcement and content providers who very much aim at deploying solutions to mitigate Kodi traffic [23, 36]. We build a classifier based on the traffic characteristics of Kodi itself and actively test 10 out of its most popular add-ons.

In the following, we summarize our main findings from the analysis of the Kodi data we have collected.

The Kodi ecosystem largely relies on “free” hosting platforms, and lots of content is “stale”—Only 10% of 11,112 unique add-ons we have discovered belong to the Kodi’s official repository; the remainder 90% are not approved by Kodi and mostly hosted on Github and LazyKodi. In addition, we find that 50% of the add-ons were last updated more than two years ago, indicating the staleness of these add-ons.

Most Kodi add-ons are safe, but most users are exposed to at least one unsafe add-on—Out of 2.3 million add-ons reported by SafeKodi users, 10,705 were unique, and only 255 are tagged unsafe, which means that these add-ons tried, for example, to contact at least one malicious IP address as indicated by Figure 10(b). However, around 80% of SafeKodi users run at least one unsafe add-on on their devices.

Kodi is not only about illegal content—*YouTube* is the most popular add-on installed by SafeKodi users, followed by *Exodus* and *SportsDevil*. YouTube is clearly an official and safe Kodi add-on, while both Exodus and SportsDevil are banned by Kodi (copyright violation) and are marked as “ipban” by SafeKodi, due to the fact that they communicate with at least one malicious IP address.

SafeKodi allows Kodi users to make informative decisions on which add-ons to keep or remove—Over one-third of SafeKodi users disabled at least one add-on using our quick disable function. Add-ons tagged as *unsafe* are the most commonly disabled, but 468 users also disabled

perfectly *safe* add-ons, which they considered as unwanted. Still, the average SafeKodi user runs 3 *unsafe* add-ons even after being informed of their danger. This suggests that the content served, likely and mostly illegal, is enough of an incentive for Kodi users to take the risk.

Kodi traffic is far from private and easy to identify—Kodi itself still heavily relies on HTTP, coupled with a specific User-Agent reporting on the Kodi version run. Due to the *auto-update* feature—used by 97.6% of SafeKodi users—an ISP can easily detect the presence of Kodi users in its network. The detection of the specific add-on run by a user is, as expected, harder. However, unofficial add-ons are less careful with respect to their user privacy, i.e., utilize HTTPS less frequently and resort to Kodi’s default User-Agent. This combined with a more unique (and sketchy) set of URLs/IP addresses they contact, makes them overall easier to identify than official add-ons.

1.1 Ethical Consideration

Since SafeKodi involves human subjects, we followed the best community practices when conducting our work. Accordingly, SafeKodi does not collect any information while running in the background, and it prompts a window (see Figure 4) during installation to (1) disclose the minimum data collected, (2) ask for the user permission to collect extra data.

SafeKodi’s data collection is anonymous. We combine the SHA256 hash value of the concatenate of the user ID and the users’ MAC addresses as user identifiers. The concatenated string has variable length and thus it cannot be indexed. With respect to a user’s IP address, we use the GeoIP package [29] to obtain coarse-grained geo-location information and then discard it. Given these measures, we cannot identify the individuals from the collected data. While checking the IRB at our institution, it was determined that our work is not considered human research because we used non-identifiable private information about living individuals, and the data collected does not contain any accompanying information by which we could identify such individuals.

2 RELATED WORK

One of the main contributions of this article is De-Kodi, a tool facilitating in-depth and transparent studies of the Kodi ecosystem. To the best of our knowledge, no previous research article has investigated this ecosystem yet. Conversely, researchers have directed their attention toward understanding the potential security and privacy threats of the Kodi application [61] as it allows arbitrary code from unknown sources to be executed. The authors show, for instance, how add-ons and video subtitles can be used as backdoors to gain control on the client device. In this work, we investigate the network traffic generated by a plethora of Kodi’s add-ons and comment on the presence of suspicious activity (Section 6.3.1). Last, we also move to build a traffic classifier to study the privacy issues of the use of Kodi.

More related work can be found in the area of copyrighted video distribution, a well-explored topic over the last 10 years. Since our work also comments on the legality of content distributed over Kodi, we here summarize the main research articles in this area.

Back in 2007–2011, platforms like YouTube and Vimeo were mostly used for redistributing illegal content [44, 47]. Even when legal, the majority of the uploaded content was copied rather than user-generated [46]. Video platforms implemented several technical solutions to prevent copyrighted materials, which in turn triggered ingenious evasion techniques such as reversing of the video (particular used in sports), covering of TV logos, and so on.

To avoid dealing with copyright detections, “uploaders” directed their attention to *cyberlockers*, or services offering remote file storages, sometimes even for free [57]. In [54], the authors scraped popular cyberlockers, e.g., MegaUpload and RapidShare, and showed that 26–79% of the content infringed copyright. More recently, Ibosiola et al. [49] study *streaming cyberlockers*, or illegal

websites that distribute pirated content (mostly video). The article looks at both cyberlockers and the content they serve. Overall, it finds a centralized ecosystem composed of a few countries and cyberlockers. Although cyberlockers as a subject are orthogonal to our study, it is worth mentioning that Kodi add-ons may utilize cyberlockers as sources of content.

An interesting new angle was explored in [48]. In this article, the authors investigate a very intuitive question: why are illegal streaming services free? They focus on illegal sports streaming and show a huge extent of user tracking—much more than what was done in legitimate streaming services. We also investigate the Kodi ecosystem for signs of tracking in Section 6.3.1.

More work on the traffic classification is related to our Kodi traffic classifier. Traffic classification is a popular networking problem, with many products built upon, e.g., firewalls and traffic engineering. The starting approach was to map applications to transport ports. This approach is nowadays ineffective as applications use either dynamic ports [51, 65] or rely on HTTP (port 80 and 443). Before the widespread of encryption (HTTPS), deep packet inspection was commonly used for traffic classification [45, 52, 59, 65]. To deal with encryption, recent work proposes to classify applications by transport layer characteristics [51], e.g., packet sizes and inter-arrival times, which are fed to machine learning methods [53, 55, 60, 67]. Furthermore, domain names extracted from DNS traffic or the **Server Name Indication (SNI)** in TLS can still help in traffic classification [58, 68]. Our method to identify the Kodi traffic involves payload inspection and domain classification, when available.

3 BACKGROUND

This section summarizes Kodi's main components and usage model, to provide the reader with the context driving the design of De-Kodi. Following this, we discuss the key challenges in crawling Kodi.

3.1 Terminology

Add-on—An add-on is a set of files—code, content, metadata, and so on.—which together work to extend the functionality of some Kodi features, ranging from media access (such as YouTube and Netflix) to Kodi GUI skins and code libraries. An add-on's properties, including the set of Kodi features extended, are described by the add-on's respective and mandatory `addon.xml` file. In addition to this, many add-ons contain special, Kodi supported Python code to be triggered deliberately or automatically by events in Kodi, such as Kodi starting or the user clicking a menu button belonging to the add-on in question. For convenient distribution, an add-on is usually packaged in a zip file; at installation, the zipped add-on is extracted into Kodi's local add-ons directory.

Many Kodi add-ons are not made by official Kodi affiliates, but by third-party developers leveraging the convenience of the Kodi platform. It has been well established that a number of these third-party add-ons engage in piracy. Kodi's official wiki site *bans* the promotion of a set of add-ons, primarily consisting of add-ons dealing with pirated content [70].

It is worth noting that, as per Kodi's disclaimer,² Kodi does not provide content. Rather, Kodi is software that facilitates media content consumption, in the same way, a browser allows for browsing the Web. Third-party developers can build Kodi's *add-ons* which can be used to stream both legal (e.g., YouTube and Vimeo) and illegal/pirated (e.g., SportsDevil and Neptune) content.³

Repository—A repository is a special type of add-on that points to a *collection* of add-ons such that they can be conveniently installed. Official Kodi is distributed with a single preinstalled

²<https://kodi.tv/about>.

³<http://www.wirelesshack.org/top-best-working-kodi-video-add-ons.html>.

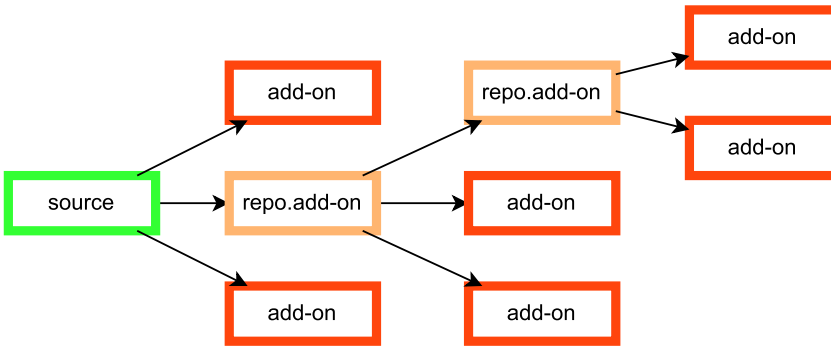


Fig. 1. Diagram exemplifying the relationship between source paths, repositories add-ons (labeled as “repo.add-ons”), and non-repository add-ons (labeled as “add-ons”).

repository called “kodi.tv”, which only contains add-ons endorsed by the Kodi team. Anyone can create their own repository to feature the add-ons of their choice. Some repositories host their content remotely, e.g., on Github or a personal server, as a means to share curated add-on lists while actively maintaining and updating their contents.

Sources—Sources are simply paths—local or remote—that point to files to be used by Kodi. While some sources directly provide consumable media (music, video, and so on), many sources act as a means to facilitate add-on distribution. Some remotely accessible sources *directly* host add-ons; others serve as *hubs*, providing a “one-stop shop” by elaborately pointing to the contents of a collection of other sources via HTTP redirection techniques. Figure 1 summarizes the relationship between *add-ons*, *repositories*, and *sources*.

Add-on manager—The add-on manager is an internal Kodi tool, which allows users to install add-ons and repositories. Kodi’s add-on manager officially provides two approaches to installing add-ons: via repositories (a type of add-on pointing to other add-ons to be conveniently installed) and via sources (direct paths—local or via HTTP—to add-on zip files).

Kodi API—Kodi offers a built-in, JSON-RPC API for generalized operations, such as navigating a menu or exposing the contents of Kodi’s built-in databases. In parallel to this, Kodi also exposes many controls exclusive to add-ons (for example, through built-in, Kodi-specific Python modules that make various Kodi features accessible to add-on developers). The savvy user may be able to create an add-on to, essentially, extend the set of Kodi operations at their disposal beyond the set provided by the outward-facing API. With De-Kodi’s API add-on, discussed in Section 4.1, we leverage *both* of these API hooks to maximize De-Kodi’s ability to control Kodi.

3.2 Challenges

Visual dependent interaction: Although Kodi’s API allows some automation, Kodi largely relies on visual information and user input to operate. This complicates crawling operations since: (1) some visual data is inaccessible to the software—menu text and on screen notifications are often not exposed through any built-in Kodi API hooks—and (2) even when this data is accessible, it can be hard for automated software to understand and react accordingly. For clarity, consider the following example. Suppose an add-on currently being crawled raises a pop-up dialog in response to the first time it is launched. This pop-up may appear at some random or inconvenient time; perhaps while the crawler is in the midst of navigating a menu. While, to a human, this

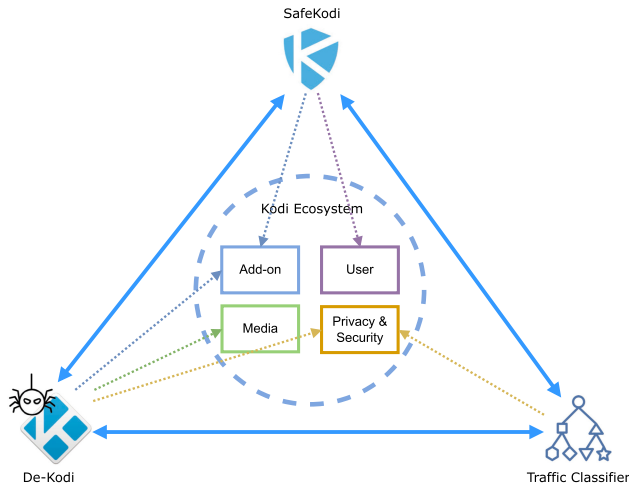


Fig. 2. System overview.

is trivial—simply respond accordingly to the text in the dialog—this would be devastating to the naïve crawler, as the focus is silently and unexpectedly shifted into an unknown state.

Lenient Add-on Implementation Requirements: Kodi does offer some guidelines for add-on structure, implementation, and metadata, but adherence to many of these guidelines is arbitrary and generally unenforced. This renders automated attempts to install, navigate, or analyze add-ons to be nontrivial.

Malicious add-ons: Previous work has established the (realized) potential for Kodi add-ons to carry dangerous malware. Kodi add-ons are generally unrestricted from accessing content located “outside” of Kodi’s explicit jurisdiction (i.e., scripts are not isolated from arbitrary local or remote files). It follows that we need to ensure any potential threats are sufficiently *isolated* to protect both our lab resources as well as the correctness of our crawl from harm.

Decentralized nature: Although there are many community maintained repositories, there is no single “app-store-like” database from which one can reliably obtain a comprehensive list of all Kodi add-ons. Therefore, crawling the Kodi ecosystem implies first *discovering* it. Further, the size of Kodi’s ecosystem is unknown, and any attempt to explore it must take into account the potentially large size of the space.

4 SYSTEM OVERVIEW

This section presents the system we have developed to explore the Kodi ecosystem. As shown in Figure 2, our system consists of three modules: De-Kodi, SafeKodi, and a Kodi traffic classifier. These modules help each other in exploring the Kodi ecosystem at 360 degrees, i.e., from discovering new add-ons to inform Kodi users about potential privacy and security threats of the add-ons they have installed. We start by describing De-Kodi which consists of a *source finder*, to discover a large corpus of add-ons in the Web, and a *crawler* which tests each add-on for their functionalities as well as potential privacy and security threats. Next, we present SafeKodi, a Kodi add-on we have developed to inform users of potential threats of the add-ons they have installed while *crowdsourcing* information about add-on popularity. Lastly, we present an algorithm to passively identify and classify Kodi traffic.

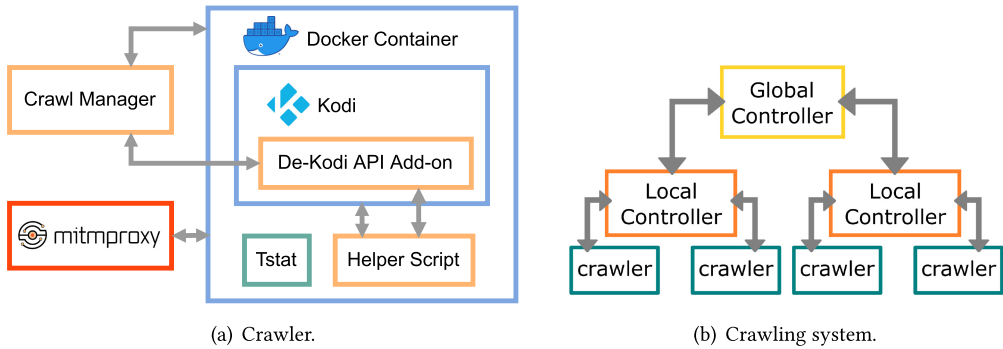


Fig. 3. A visual overview of the De-Kodi system. Figure 3(a) shows the structure of an individual *crawler*. The *crawlers* in 3(b) are instances of the *crawler* shown in 3(a), but in the case of 3(b), we use one instance of *mitmproxy* per machine to capture traffic from all *crawlers*.

4.1 Crawler

The *crawler* is the core component of De-Kodi. At a high level, its goal is to take an add-on as an input and crawl it, i.e., install it on a Kodi instance and navigate through its functionalities while recording things like its structure, network traffic, and so on.

Figure 3(a) shows a high-level overview of De-Kodi’s crawler. A key observation is that the crawler relies on Docker’s technology. The reasons behind this choice are twofold. First, it is a convenient tool to isolate Kodi instances without allowing debris, e.g., code/libraries from previous add-on installations or potential malware. Second, it allows De-Kodi to inherit Docker’s scalability property.

Observe, in the aforementioned figure, that portions of De-Kodi’s crawler run directly on the *host* machine, borrowing Docker terminology, while others operate from inside a Docker *container* [25] (on the figure’s right-hand side). In the following, we explain each sub-component of De-Kodi’s crawler in detail distinguishing between its host and container component.

4.1.1 Host. We here describe the crawler’s components which run directly on the machine without any OS virtualization (via Docker).

Crawl Manager—This a Python script whose goal is to “manage” a crawl. At a high level, this implies (1) launching a Docker container equipped with Kodi and additional software; (2) launching Kodi and necessary support software, such as TSTAT, (3) managing high-level crawl operations, such as add-on installations, and (4) collecting both state data and experiment results from the crawl.

Mitmproxy. Much of Kodi’s traffic is encrypted, so we use the Mitmproxy [35] (a “man in the middle” proxy server) to expose the contents of such traffic. We run the Mitmproxy at the host, instead of one instance per Docker container, since it minimizes the waste of resources (CPU and memory) and, by definition, the host has full visibility into the traffic originated by each container. Note that this requires installing Mitmproxy’s root/CA Certificate in our containers to ensure proper functioning of Kodi. While this approach does not work in presence of pinned certificates, we found no evidence of this technology currently being used in the Kodi ecosystem.

4.1.2 Container. The crawler’s container runs a Docker image derived from an Ubuntu 16.04 image, primed with (1) Kodi (vrs 18.0), (2) De-Kodi’s software that runs inside the container (see the right-hand side of Figure 3(a)), and (3) the zip file of at least one Kodi add-on to be tested. Kodi

runs inside the container headlessly via a virtual screen (Xvfb [19]). In the following, we describe in detail De-Kodi’s software that runs inside the container.

Add-on—Although Kodi provides several API hooks, many “advanced” operations (e.g., adding a new data source and navigating and interpreting complex menus) require a human user, actively looking at the screen for visual feedback as they make decisions. De-Kodi’s add-on is a *service*—meaning it starts automatically when Kodi is launched—that extends Kodi’s API to be more automation friendly. The add-on runs an RPC server that receives crawling instructions, e.g., navigate to this menu, from its crawl manager.

We determined, through manual experimentation, how the add-on can strategically react to the diverse scenarios that vary unpredictably with each add-on (see Section 3.2), often acting with drastically incomplete information at its disposal. Accordingly, the API add-on is tasked to monitor Kodi’s state and notice when it deviates from its expected path, e.g., clicking a menu entry should open either a new menu or some playable items like a video. In a case such deviation is detected, the API add-on attempts an intelligent “guess” at how to return to the expected path, e.g., close a dialog by accepting a potential warning. This is often a guess as many dialogs contain text that is only visually accessible—for our crawler, lacking eyes, such context is out of reach.

Helper Scripts. We refer to Python scripts, bash scripts, and other Docker environment altering files we have placed within the Docker image, but outside of Kodi, as “helper scripts”. Helper scripts serve to enhance De-Kodi’s visibility into Kodi’s interactions with its environment. The specific purposes of each helper script vary greatly, ranging from restarting Kodi upon getting stuck to retrieving the URLs of playable add-on content.

TSTAT [18]—TSTAT is a tool providing detailed, per-flow, statistical analysis of TCP traffic. We chose to use TSTAT to gain high level insights into the nature of traffic generated by Kodi add-ons. De-Kodi’s copy of TSTAT is configured to log all DNS queries/answers, and HTTP requests/responses (this often includes a domain name and file name if unencrypted), and general connection statistics for all observed TCP and UDP traffic.

4.2 Source Finder

The underlying assumption for De-Kodi’s crawler is that add-ons are available to be tested. This is true for Kodi’s official repository, whose add-ons can easily be installed from any Kodi instance. This assumption does not hold for the larger set of unofficial Kodi add-ons which are scattered around the Web. This motivates our need to build a *source finder* tool.

Due to the lack of a centralized Kodi add-ons aggregator, avid Kodi users are forced to socialize to exchange add-ons and sources. We have identified three main places to search for Kodi add-ons on the Web: (1) LazyKodi, a well-known and actively maintained Kodi add-on source which aggregates collections of add-on repositories and add-ons into a convenient, single location [34], (2) Reddit, a large online social platform with many publicly accessible communities [37], and (3) GitHub, a large online software development platform often used for hosting, maintaining, and distributing open-source code [30]. For the remainder of this article, we refer to these three entities as our “search seeds” or “seeds”. In Section 6.3.2, we leverage the apparent distribution of popularity across add-ons to assess the effectiveness of our seeds in terms of coverage.

As LazyKodi is itself designed to be a Kodi source, pointing directly to remotely stored add-on zip files, De-Kodi’s source finder browses LazyKodi using a special *crawler* instance, acting as its crawl manager and guiding the crawl across a source menu (corresponding to LazyKodi) as opposed to an add-on menu. Note that it is also possible to crawl LazyKodi using an ordinary web crawler, given that a Kodi User-Agent is used.

For our other seeds, we built a simple Web crawler which looks for Kodi-related terms (e.g., Kodi, XBMC, and so on) on both GitHub and Reddit. These links are expected to point either directly to Kodi add-ons or collections of add-ons (such collections are often utilized to remotely store the add-ons pointed to by Kodi repository add-ons). The source finder attempts to filter GitHub and Reddit results to exclude false positive links—specifically, URLs that point to non-Kodi content.

It is also worth noting that discovering *redundant* copies of an add-on is common and difficult to avoid: popular add-ons may appear in many repositories. On top of this, outmoded and defunct add-on versions can persist online, often remaining retrievable despite the release of newer versions. We mitigate the impact of this redundancy by (1) identifying “already crawled” add-ons by their add-on ID and (2) always opting to re-crawl an already crawled add-on if a *newer* version is found.

4.3 Crawling Workflow

In this subsection, we document the relationships between the aforementioned components of De-Kodi and describe De-Kodi’s overarching control flow and structure, which is depicted in Figure 3(b). First, we start a *global controller* which utilizes previously obtained outputs of a *source finder* to actively discover add-ons. Next, we start some number of *local controllers* which run several instances of the *crawler*. The global controller serves as a centralized point of contact for all local controllers, which periodically query the global controller for overall crawl state information (for example, “Does this add-on need to be crawled, or has it already been crawled?”) and to provide the global controller with updates concerning an ongoing or recently completed crawl (for example, “this add-on was successfully installed, but no playable content was identified”). The following procedure then occurs repeatedly:

(1) A *local controller* queries the global controller which replies with a link, or a URL obtained by the source finder via a seed. The local controller then downloads the resources pointed to by the provided link and formally verifies that they contain either an add-on or a collection of add-ons. Specifically, the local controller looks for `addon.xml` files and inspects them to ensure that they are formatted correctly. From a properly formatted `addon.xml` file, a local controller extracts, at a minimum, the add-on id, the list of Kodi features extended by the add-on (which we refer to as the add-on “type”—note it is possible for an add-on to have multiple types), and the add-on’s version number. Often additional details are provided, which the local controller will also capture when present. The local controller treats failure to capture any of the required pieces of information about an add-on from its required `addon.xml` file as an indication that the downloaded material is not an add-on. If no add-ons are verified from the current link, the local controller repeats this step.

(2) If add-ons were found in the previous step, the local controller communicates the set of identified add-ons and their respective data to the global controller. From this set, the global controller removes add-ons that have been already successfully crawled. The resulting subset of add-ons is then returned to the local controller which packages them in zip archives. Next, it creates a Docker image which contains, in addition to default De-Kodi’s container code, the zip files of the add-ons to be crawled next. If no add-ons were returned, the local controller returns to the first step.

(3) If add-ons to crawl were obtained in the previous step, the local controller launches up to n crawlers, where n is the maximum number of crawlers the local controller has been configured to allow in a crawl session. Each crawler is assigned exactly one add-on from the set to be crawled. As detailed in Section 4.1, the crawler then launches a Docker container using the recently created Docker image, attempts to install the add-on under test, and finally attempts to find playable media (if appropriate for the add-on’s determined type). We test discovered media URLs for reachability, geolocalize their respective IP addresses, and attempt to obtain corresponding video information

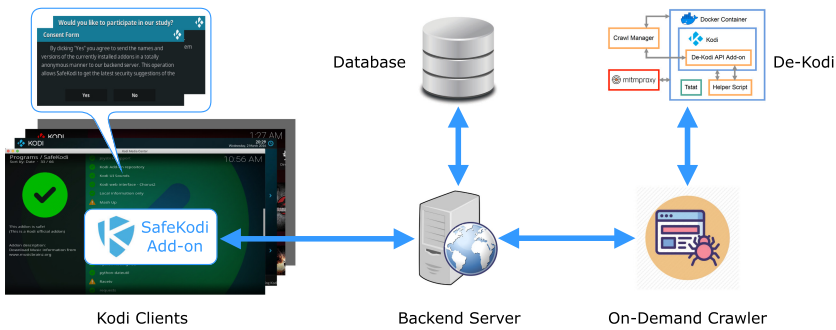


Fig. 4. SafeKodi system overview.

using `ffprobe` [11]. Throughout the crawl, the local controller communicates its progress to the global controller, e.g., whether an add-on installed successfully or not.

(4) When a crawler completes the crawl of its assigned add-on (either from running out of menu items to browse or by reaching a pre-configured timeout capping the amount of time spent on each add-on), the local controller closes all of that crawler’s active materials (e.g., the Kodi instance, the Docker session, temporary state information). When the number of active crawlers drops below n , the local controller launches new crawlers if there are remaining add-ons in the current add-on set to be crawled.

In some cases, an installed add-on may itself be a repository, pointing to many *other* potentially new add-ons to test. In such a scenario, the crawler communicates newly discovered add-ons to its local controller. The local controller then, before closing the crawler, creates a *new* container image so that the newly discovered add-ons can be crawled. After obtaining permission from the global controller, the local controller then appends these add-ons (or some subset of these add-ons, depending on the global controller’s response) to its current set to be crawled.

(5) Once the remaining number of add-ons to be crawled in its current set drops to zero, the local controller repeats the cycle, querying the global controller again to obtain a new link.

We have open-sourced De-Kodi [4] hoping it can be helpful to the community as a starting point for other studies of Kodi, e.g., focusing on different content apart from video.

4.4 SafeKodi

Figure 4 shows a visual representation of SafeKodi’s key components: Kodi add-on, backend server, and on-demand crawler. The workflow is as follows. De-Kodi runs at our premises to monitor the Kodi ecosystem, e.g., with a weekly frequency. The data collected is stored in a PostgreSQL database and contains, among other information, the labeling of each discovered add-on, e.g., *safe* or potentially *malicious*. This data is made available to Kodi users via restful APIs implemented in the SafeKodi add-on. In turn, the add-on exposes to our backend servers the list of the user’s installed add-ons to be crawled *on demand*, if not yet available in our database. In the following, we detail the design of each SafeKodi’s component.

4.4.1 On-demand Crawler. We extend the system to enable on-demand crawling in two steps. First, we add a database where to store *crawl* data in a scalable manner. That is, we reduce the raw crawl data from hundreds of GBytes (pcap traces and container state) to a handful of GBytes per crawl. This allows us to run De-Kodi continuously with realistic data storage settings. Second, and more importantly, we introduce *on-demand* crawling. This feature allows a user to request an add-on to be crawled by *keyword*, a feature required to enable SafeKodi’s workflow described

above. Under the hood, the source finder will search Github using the provided keyword paired with “Kodi” and “add-on”. Next, we inspect the top search results verifying potential Kodi add-ons, i.e., they contain a properly formatted *addon.xml* file required by Kodi. In case a valid match was found, the new add-on is recorded and its testing initiated. Note that while the search can be extended to other websites easily, we only use Github as the add-on repository since our results show it to be the most reliable source (see Section 6.2).

4.4.2 Add-on. The SafeKodi add-on provides a user-friendly interface to inform Kodi users about potential security issues of their installed add-ons. As shown in the left of Figure 4, SafeKodi shows a list of installed add-ons labeled as *safe* (green tick), *tracking* (yellow exclamation point), *malicious* (red cross), and *unavailable* (gray question mark). Each add-on in the list can be selected to offer further information on the function claimed by an add-on coupled with an explanation on the labeling provided by SafeKodi. For example, if an add-on is marked unsafe the actual motivation behind such labeling is provided. Last but not least, SafeKodi also offers a shortcut to *disable* a selected add-on.

The SafeKodi add-on can be downloaded from our website [38] and installed as any other Kodi add-on. Our code is open-source and we welcome an investigation to verify our privacy claims. When the installation completes, the add-on informs the user about the undergoing research project with a clarification of what information will be collected. At a minimum, the add-on will share with SafeKodi’s backend server the list of Kodi add-ons installed along with the user’s IP address—which is inherently available to us when the add-on contacts our backend server. The installation will further ask the Kodi user if (s)he is willing to share extra information from the Kodi settings, e.g., Kodi version, screen resolution, bandwidth limit, and some security-related settings.

Under the hood, SafeKodi add-on leverages Kodi’s JSON RPC calls [33]. When launched, the list of installed add-ons is retrieved using *Addons.GetAddons* RPC. If the user previously agreed on sharing extra data with SafeKodi, local Kodi settings are also retrieved using *Settings.SetSettingValue* RPC. This data is then sent to our backend server via an HTTPS POST message. In response, the add-on will receive the information needed to populate SafeKodi’s GUI as per the above description. The SafeKodi add-on only reports data to our backend server when manually launched by the user. However, a background service checks every 12 hrs whether a database update is available, and if so it informs the user via a popup message. If a user requests to disable an add-on, this is achieved via the *Addons.SetAddonEnabled* RPC call.

4.4.3 Backend Server. SafeKodi backend server currently runs on an AWS EC2 instance [21], which has demonstrated to be enough to handle our user-base (over 15,000 users, as we will discuss in the upcoming section). Software-wise, it consists of a PostgreSQL database and a Web application written in Python. The database has two tables: one for storing information on the add-ons crawled by De-Kodi, and one for storing the data (add-ons lists and potentially some Kodi settings) collected by the SafeKodi add-on.

The Python-based Web application exposes and manages a set of restful APIs implemented using CherryPy.⁴ This task translates into: (1) handling GET requests for add-on information updates, and POST requests for add-on lists sharing, (2) anonymizing the requests received by the SafeKodi add-on, (3) database management, i.e., querying for add-on tags and storing the data POSTed by SafeKodi users, (4) on-demand crawling management, i.e., sharing with De-Kodi any add-on which was not found in the database, and adding any newly crawled add-on into the database.

⁴<https://cherrypy.org/>.

4.5 KODI TRAFFIC IDENTIFICATION

Whether Kodi traffic can be detected or not is an important research question with many practical implications. From an institutional perspective, network administrators can use such findings to detect and potentially block Kodi traffic. From an ISP perspective, content providers and law enforcement are pressured to deploy solutions to mitigate Kodi traffic [22, 24, 41, 43], as recently observed in the UK [23, 36]. Last but not least, Kodi users should be aware of potential privacy risks.

Motivated by the above, we leverage De-Kodi’s data to build a Kodi detection algorithm assuming transparent traffic interception, i.e., without the assumption of breaking TLS (encrypted) communication. We first discuss several observations about Kodi traffic, that we then generalize to conceive an algorithm for Kodi detection. Finally, we evaluate the accuracy of our algorithm in detecting usage of Kodi along with the top 10 most popular add-ons, assuming transparent traffic interception at both LAN and WAN level.

Methodology—Kodi itself is a home media center, which does not carry much traffic. Most of Kodi’s traffic is instead generated by add-ons to retrieve media content available on the Web. We are able to generalize a method to identify the traffic generated from Kodi and its add-ons based on the following observations:

- (1) When Kodi boots, it registers two services, “xbmc-jsonrpc” and “xbmc-events,” through broadcasting **multicast DNS (mDNS)** messages within the LAN (home network, for instance). Kodi keeps these two services alive by regularly broadcasting these messages. This traffic is likely what is broadly defined as “heartbeat traffic” in the Sandvine report [6].
- (2) We find 12 URLs and their corresponding IP addresses that are very much Kodi specific. For example, `mirror.kodi.tv` is the website that provides access to all official Kodi add-ons. If *auto update*—a Kodi feature to check if a new version of an add-on is available—is enabled, Kodi will access this URL every time it is booted. According to Section 6.5, 97.6% of SafeKodi users have auto update enabled on their Kodi.
- (3) Legacy Kodi uses a custom User-Agent in the HTTP header which contains the keyword “Kodi” along with a version number. Despite today’s traffic is mostly encrypted [32], thus hiding such information to a third-party observer, we find that many add-ons (33 out of the top 40, see Table 1) still rely on HTTP for a portion of their traffic, e.g., to retrieve the icon to display on Kodi’s main menu, as well as for some content streaming, or for remote control. While most of these add-ons (29 out of 33, see Appendix A) modify the User-Agent to disguise themselves as regular Web browsers, Table 1 shows that nearly half of the add-ons—mostly unofficial add-ons—also fallback to Kodi’s default User-Agent when using some Kodi APIs. For example, *Exodus Redux* uses a custom HTTP header when accessing the content lists. However, when user selects a movie to watch, it fetches the image of the movie with Kodi default User-Agent in the HTTP header.
- (4) Our crawls continuously maintain up-to-date lists of URLs/IP addresses that each Kodi add-on access. We refer to these as *candidate* URLs/IP addresses, each of which can reversely map to one or more add-ons. We match collected traffic against such a list to identify which add-ons might be in use.

We leverage these observations to build the Kodi detection algorithm described in Algorithm 1. We make the algorithm generic such that it can be run both from within a LAN, e.g., at a home router, or a WAN, e.g., at one of the many middleboxes deployed by ISPs [50, 66]. The detection algorithm iterates through PCAP traces—network and application traffic (when available, i.e., no HTTPS)—looking for matches against the above observations. The output of the algorithm are

Table 1. HTTP Usage in the Top 20 Official and Unofficial Media add-ons

Official Add-ons					Unofficial Add-ons				
✔	⚠	Ⓚ	✔	⚠	Ⓚ	Ⓚ	Ⓚ	⚠	Ⓚ
✔	⚠	✔	⚠	⚠	Ⓚ	⚠	Ⓚ	Ⓚ	Ⓚ
✔	⚠	⚠	⚠	✔	Ⓚ	Ⓚ	⚠	⚠	Ⓚ
⚠	⚠	⚠	✔	⚠	Ⓚ	Ⓚ	Ⓚ	Ⓚ	Ⓚ

Add-ons are labeled as follows: (1) solely rely on HTTPS (✔), (2) some HTTP requests are found but all with custom User-Agent (⚠), (3) some HTTP requests are found which expose Kodi usage in the User-Agent field (Ⓚ). Full details can be found in Appendix A.

ALGORITHM 1: Kodi Traffic Identification

Input : Captured Packet Set P ; Kodi Sessions $S_k = \{\}$;
 Maps from URL/IP address to potential add-on (s) M ;
 Set of URLs/IP addresses that only serve Kodi content U_k ;
 Candidate Windows T .

```

1 for  $p \in P$  do
2    $addonList \leftarrow \{\}$  // potentially belongs to which add-ons
3    $isKodi \leftarrow False$ 
4   if ( $p$  contains  $mDNS$ ) && ( $mDNS$  contains 'xbmc') then
5     |  $isKodi \leftarrow True$ 
6     |  $addonList.add("kodi")$ 
7   end
8   if ( $p$  contains  $HTTP$ ) && ( $HTTP$  Head contains 'kodi') then
9     |  $isKodi \leftarrow True$ 
10  end
11  if  $p.dst \in U_k$  then
12    |  $isKodi \leftarrow True$ 
13  end
14  if  $p.dst \in M.keys$  then
15    |  $addonList.add(M[p.dst])$  // Which add-ons access this destination
16  end
17  for  $s \in S_k$  do
18    | if ( $p.src = s.src$ ) && ( $p$  happens within  $s \pm T$ ) && ( $s.addons \cap addonList \neq \emptyset$ ) then
19      | |  $p$  matches  $s$ 
20      | |  $s.addons \leftarrow s.addons \cap addonList$ 
21      | | break
22    | end
23  end
24  if ( $p$  is not matched) && ( $isKodi = True$ ) then
25    |  $s \leftarrow New\ Session(p.src, p.timestamp, addonList)$ 
26    |  $S_k.add(s)$ 
27  end
28 end
Output : Kodi Sessions  $S_k$ .

```

Kodi *sessions*, defined as the set of packets generated by a Kodi action such as simply booting Kodi or running an add-on.

Lines 4 (L4) to L15 in Algorithm 1 have two main roles: (1) identify if a packet is Kodi, and (2) potentially return a list of matching add-ons (L15). To do so, each `if` statement implements one of the above four observations we derived from Kodi traffic. L4 looks for Kodi-specific mDNS traffic, while L8 verifies if the “User-Agent” contains the distinctive keyword “Kodi”. L11 matches Kodi-specific URLs/IP addresses, while L14 matches add-on-specific URLs/IP addresses. All matches on URLs are realized as (1) full URL match in case of HTTP, (2) SNI match (domain name) in case of HTTPS, (3) domain name match in case of unencrypted DNS. When none of these are available, e.g., the user uses HTTPS with encrypted SNI, we resort to IP addresses. The IP addresses in our dataset were collected by De-Kodi running at our campus over a period of 12 months (from the first Web crawl in October 2019 to the end of the data collection with SafeKodi in September 2020). This does not guarantee that these IP addresses are used by an add-on when running from another network location, or even at the same location after a long period of time. This is because of the dynamicity of IP addresses (much higher than URLs) due to load balancing and geolocation, e.g., as used by **Content Delivery Networks (CDNs)**.

L17 iterates through all Kodi sessions detected so far, and L18 decides if the current packet matches an existing Kodi session using three conditions: (1) it originates from the same IP address as any previous Kodi session, (2) it happens within a *detection window* defined as the time from the last packet received in a session plus $T = 60$ seconds, and (3) the intersection between previously matched add-ons for this session and new add-ons matched by this packet (L15) is not empty. Note that if there are no common add-ons, then most likely this packet was originated either from other software on the same machine or from other machines under the same IP address (e.g., when collecting traffic in a WAN).

A *matching* packet is assigned to the corresponding Kodi session (L19), and the set of matching add-ons for a session is restricted to its intersection with newly matched add-ons (L20). The rationale of this restriction is to, as the pcap traces progresses, improve the add-on detection. For example, *SportsDevil* first contacts `www.filmon.com`, which maps to 10 add-ons including *SportsDevil*, *FileOn*, and *FTV*. Then, it contacts `m.liveonlinetv.org`, which maps to two add-ons: *SportsDevil* and *All In One Video*. The intersection of these two sets result in one add-on: *SportsDevil*. An *unmatched* Kodi-related packet (L24) triggers the creation of a new session (L25, L26) containing the source IP address (`p.src`), a timestamp (`p.timestamp`), and the set of matched add-ons (`addonList`).

Limitations. Within a LAN, multiple concurrent Kodi users are differentiated via their source IP address (`p.src`). The same does not hold from a WAN where the source IP address gets aggregated at the home router level. This does not impact the traffic detection question, but rather the ability to distinguish how many Kodi users are within a household. This is a generic problem of transparent traffic analysis which is not specific to Kodi.

For add-on traffic destination matching (L11 and L14), in absence of HTTP we rely on SNI match—in case of HTTPS—and domain name match—in case of DNS. The SNI field is currently in the clear even with TLS 1.3 [62] but it will eventually disappear when the proposal to encrypt *all* TLS extensions will be adopted [63]. Similarly, DNS traffic is shifting more and more to DNS-over-HTTPS [42, 56] which implies such a field will eventually disappear as well.

5 DE-KODI BENCHMARKING

De-Kodi aims at being sufficiently lightweight for use on commodity hardware and readily scalable for arbitrarily large snapshots of the Kodi ecosystem. To this end, De-Kodi was designed to be easily

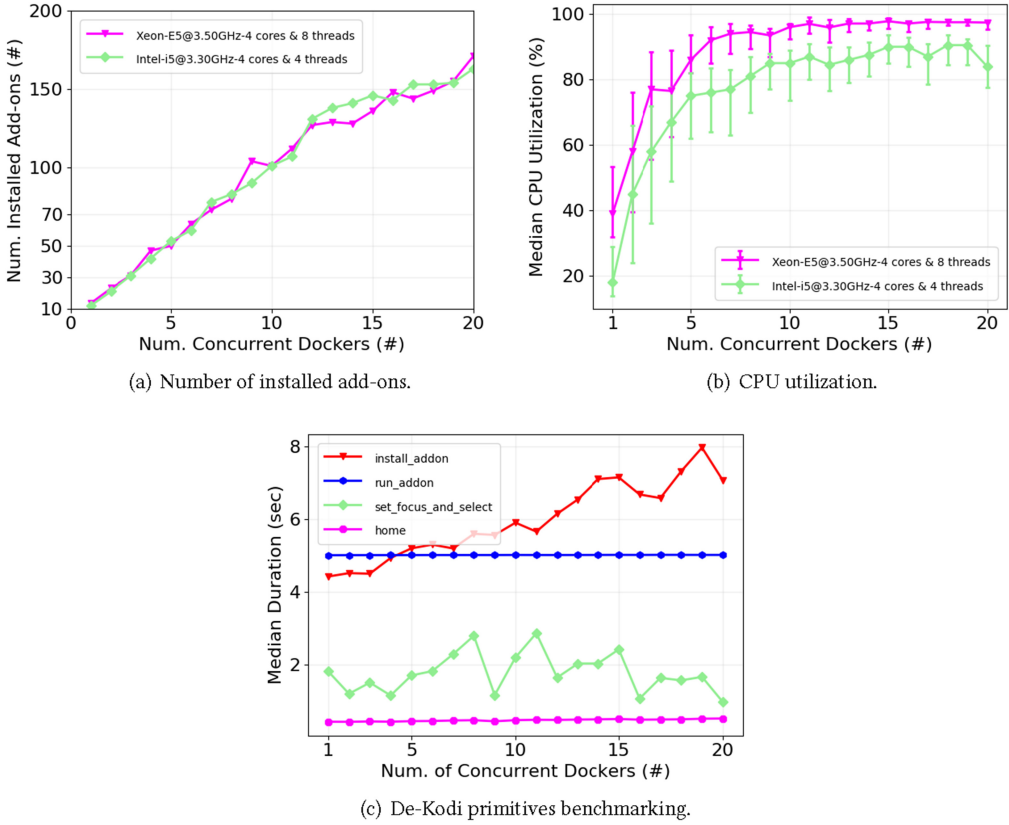


Fig. 5. De-Kodi benchmarking; $N_{Docker} = [1 : 20]$; Crawling-duration: 30 minutes.

parallelizable, both in terms of Docker instances and the number of machines where it can run. We setup two machines⁵ at a university campus connected to the Internet via a shared Gigabit connection (both in download and upload). Next, we instrument each machine to run De-Kodi for 30 minutes while crawling the same set of add-ons. Note that in a real crawl, each machine would focus on a different set of add-ons, but the goal here is to compare their performance while operating on the same workload. We repeat each crawl 20 times while increasing the number of Docker instances (N_{Docker}) used per machine from 1 to 20. Kodi’s default add-on repo was used for this benchmark.

Figure 5(a) shows the number of successfully crawled Kodi add-ons as a function of the number of Docker instances used and the machine where the crawler ran. When $N_{Docker} \leq 10$, the number of crawled add-ons grows linearly (between 10 and 100 add-ons) and no major difference is observable across machines. When $N_{Docker} > 10$, we start observing a sublinear growth in the number of crawled add-ons and more “noise” in the results. This suggests that, eventually, the overhead of running more Docker instances on a single machine does not pay off in a term of crawling “speed”.

To further understand the previous result, we investigate the CPU utilization during the above benchmarking. Figure 5(b) shows the median CPU utilization as a function of the number of Docker instances and machine used. Error-bars relate to 25th and 75th percentiles. Note that the CPU

⁵One machine mounts an Intel i5-4590 (3.30 GHz, quadcore, 4 threads); the other machine mounts an Intel Xeon E5-1620 (3.50 GHz, quadcore, 8 threads). Both machines are equipped with 8 GB of RAM.

utilization is indeed a distribution since we sample it every 5 seconds during the benchmarking. The trend mimics the one observed above, i.e., linear increase followed by a saturation as we approach exhaustion of available CPU. It can be observed how the distance between percentiles becomes more tight as N_{Docker} increases. This implies that the machines are under higher CPU utilization for a longer time as the overall load increases (higher N_{Docker}).

To understand the latter results, we benchmark low-level De-Kodi “primitives”, i.e., functions like “install_addon” or “run_addon” which are composed together to enable crawling. Figure 5(c) shows the average duration of key De-Kodi primitives as a function of N_{Docker} . These results refer to one of the machines, but they are representative of all machines. The figure shows how most De-Kodi primitives are not impacted by N_{Docker} , i.e., their durations are limited by Kodi’s implementation-induced constraints rather than the machine resources. The primitive “install_addon” is the only one impacted by the machine resources. This happens because this primitive is more complex and requires network operations (to pull the add-on), and CPU usage (to perform its installation). However, this operation only constitutes a small fraction of De-Kodi operations which are instead dominated by atomic or constant time operations.

No significant difference was instead reported in term of memory consumption. Across the machines, De-Kodi requires a minimum of 500 MB ($N_{\text{Docker}} = 1$) and a maximum of 4 GB ($N_{\text{Docker}} = 20$). Based on these empirical results, we set for the crawler a conservative $N_{\text{Docker}} = 8$ which should allow us to crawl up to 11,000 add-ons per day while not overloading the test machines. Note that, in practice, the rate at which *distinct* add-ons are covered will decline in response to redundant discoveries (if an older add-on version is found first), crawl failures (discussed in Section 6.2), and lowered performance induced by poorly coded add-ons.

6 KODI ECOSYSTEM ANALYSIS

This Section performs an in-depth analysis of the Kodi ecosystem focusing on add-ons discovered via a Web crawl, and crowdsourced via SafeKodi. First, we summarize the Web crawling and SafeKodi user-base and their contribution in terms of discovered add-ons. Then, we focus on *which* add-ons are popular, their potential danger, and whether Kodi users are concerned by their presence. Finally, we conclude with an overview of Kodi settings chosen by its users, and their potential security implications.

6.1 Dataset Collection

We deploy De-Kodi across the three machines used for benchmarking, enabling up to eight concurrent Docker instances per machine, which offers high utilization of the available resources without significant overload. The more powerful machine is instrumented to act both as a crawl manager and a source finder (see Section 4). We then crawl Kodi over the course of 5 days in October 2019.

Table 2 gives the high-level overview of Kodi’s ecosystem from one Web crawl. The first column shows the total and unique number of discovered entities, e.g., add-ons and media pointing URLs. The second column, when applicable, shows the subset of entities that properly installed on the most recent version of Kodi running alongside De-Kodi. Remember that the source finder is instrumented with the three source seeds introduced in Section 4.2: LaziKodi, Reddit, and GitHub. Together, the search seeds yielded 1,769 links to potential Kodi add-on sources. In addition, we seed De-Kodi with add-ons from Kodi’s official repository: `kodi.tv`. Using this repository and the aforementioned sources, De-Kodi ultimately discovered 9,146 distinct add-ons, including 1,008 “kodi.tv” add-ons, as well as 172 “banned” add-ons, i.e., add-ons associated with illicit activity and formally denounced by the XBMC Foundation [70]. The crawl discovered 5,435 “pluginsources” add-ons, i.e., potential media yielding add-ons, of which only 423 yielded at least one pointer to streamable content. This number is a potential lower bound since navigating Kodi add-ons is hard

Table 2. Web Crawl Summary

	Total Distinct	Installed
Search seed links	1,769	-
Total add-ons	9,146	5,265
Media add-ons	5,435	3,191
Repository add-ons	1,212	779
kodi.tv add-ons	1,008	988
XBMC banned add-ons	172	109
SafeBrowsing flagged add-ons	4	4
Ad containing add-ons	11	11
IP banned add-ons	105	105
Add-ons with media URLs discovered	423	423
Media URLs	6,117	-
Media domains	885	-
Media second-level domains	517	-
ip hosting domains	116	-

Missing fields are “inapplicable”.

and time consuming. Nevertheless, the crawl yielded 6,117 URLs pointing to audio/video content, spanning 885 fully qualified domain names and 517 unique **second-level domains (SLDs)**. Out of the 9,146 add-ons discovered, 5,265 add-ons were successfully installed and tested by De-Kodi.

Later, we have launched SafeKodi on February 24th, 2020. Until September 2020, SafeKodi has attracted 15,768 unique users from 104 countries. These users have run SafeKodi at least once, whereas 32% stick with SafeKodi and run it, on average, every 4 days. SafeKodi users report, on average, about 147 installed add-ons each which sum up to a staggering 2,322,843 add-ons reported over seven months. Overall, 10,705 crowdsourced add-ons are *unique*; the majority of these add-ons (67.4%) were unknown to De-Kodi and triggered *on-demand* crawling. 55.6% of these unknown add-ons were successfully crawled. When combined with our *periodic* crawling, this sums up to 11,112 working add-ons, 2x the number of add-ons that were discovered the Web crawl.

We acknowledge that our dataset encompasses a relatively small sample of the Kodi user-base (15,000 out of millions of users worldwide); yet, it is the first and most comprehensive analysis so far of the Kodi ecosystem from the users’ perspective.

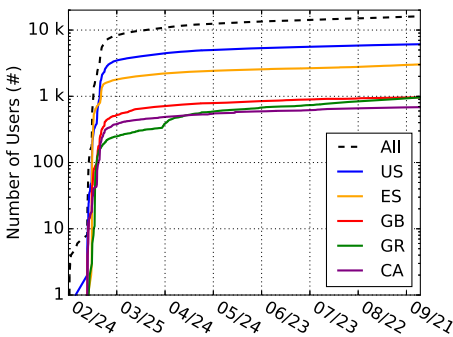
Figure 6(a) shows SafeKodi’s user-base growth overall (black dashed line, labeled *All*) and in the top five countries as determined via IP geo-location: **United States (US)**, **Spain (ES)**, **Great Britain (GB)**, **Greece (GR)**, and **Canada (CA)**. Note that we have excluded the 274 users which reported to be accessing Kodi through a proxy.⁶ The figure shows that many of SafeKodi users joined in the first half of March. We tracked this down to several media articles [1, 5] and YouTube videos [2, 3] which covered SafeKodi as *the first antivirus for Kodi*. Despite the logarithm used for the *y*-axis suggests a flat growth, during the coming months SafeKodi kept adding around 500 users per week, on average, irrespective of the country. The impact of media coverage is quite noticeable for Spain, where the article published in [1] triggered a thousand new users in less than a week, making it the second most popular country in SafeKodi’s user-base. Note that GeoIP [29] only failed to geo-locate 1,240 IP addresses out of 15,768.

Next, we focus on the Kodi add-ons discovered by SafeKodi along with on-demand crawling via De-Kodi. Figure 6 (blue curve) shows the evolution over time of the number of unique Kodi

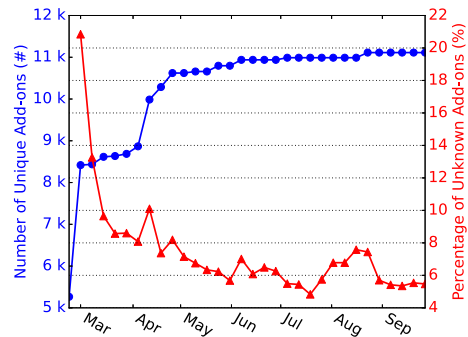
⁶When a proxy is involved, Spain becomes the most popular country of access with 110 users, followed by Greece and US, as shown in Table 3.

Table 3. Summary of SafeKodi’s Data Collection

Duration	7 months
Number of requests	27,670
Number of unique users	15,768
Returned user	4,979
Returned avg interval	4 days and 7h
Users behind a proxy	274, 40% in Spain
Add-ons reported	2,322,843
Unique add-ons reported	10,705
Failed add-ons	3,940
Total add-ons	18,933
Total installed add-ons	11,112
Users opt in sharing the settings	6,015
Users set bandwidth limit	108, 50% set <1 Mbps
Users enabled remote control	771
Users disabled auto update	146
Kodi versions	[15.3, 21.1], 87% are 18.6 to 18.8
Screen resolution	[360p, 4K], 60% are 1080p



(a) Evolution over time of SafeKodi’s user growth (February-Sep. 2020).



(b) Time evolution of Kodi add-ons installed (blue line) and percentage of unknown add-ons (red line) at the time of reporting (weekly average).

Fig. 6. Characterization of Kodi add-ons.

add-ons that were correctly tested and tagged by De-Kodi. The curve starts from October 2019 when we ran the first Web crawl based on the seeds we have identified using De-Kodi and discovered 5,265 add-ons. Next, the curve shows how many new add-ons were discovered as crowdsourced add-ons are offered by SafeKodi users. The red line shows the percentage of crowdsourced add-ons which were *unknown*, i.e., not previously known at a point in time. The red curve starts from March 2nd, i.e., one week after SafeKodi’s launch, at which point 21% of the add-ons discovered were not previously crawled. Over time, this percentage dropped down to roughly 5% (as of 09/29/2020), showing the effectiveness of Web crawl plus SafeKodi to offer a high coverage of the Kodi ecosystem. Among the newly discovered add-ons, 161 were developed after the De-Kodi crawl on October 2019.

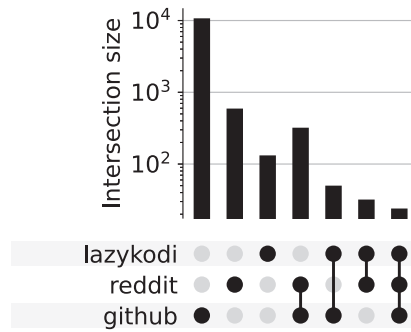


Fig. 7. UpSet plot of add-ons directly found across search seeds. Each bar’s respective seeds are marked with a black dot.

6.2 Dataset Validation

De-Kodi has three main tasks: add-on discovery (along with SafeKodi), add-on installation, and media finding. Below we benchmark, to the best of our abilities, each of the above tasks.

6.2.1 Add-on Discovery. Any attempt to crawl a large ecosystem such as Kodi leans heavily on the assumption that one has a way of traversing the space to be crawled. In the case of Kodi, the primary space to cover is Kodi’s large and decentralized library of add-ons, which is discovered via a Web crawl (see Section 4.2). Lacking a ground truth view of the set of add-ons comprising Kodi’s ecosystem, in the following we only comment on the amount of unique add-ons offered by each seed: GitHub, Reddit, and Lazikodi.

Figure 7 quantifies the number of add-ons/repos discovered *directly*, i.e., the first layer from the tree in Figure 1, via each search seed. We treat add-ons with matching add-on IDs as equivalent discoveries. The figure shows that the code of 10,387 add-ons is found through GitHub, 593 through Reddit, and 132 through LazyKodi. The code of 24 add-ons was found across all three seeds. The apparent bias in add-on discovery toward the GitHub seed highlights the common practice of Kodi users to leverage GitHub as a free hosting service for Kodi repositories. Meanwhile, Figure 7 also draws attention to the dangers of relying on a *single* seed: only 321 of the 593 add-ons discovered via Reddit were also found on GitHub, meaning 272 add-ons may have been missed without seeding Reddit in parallel to GitHub. Despite its small size, Lazikodi still produces a handful of add-ons, which are not found elsewhere. Because of the popularity of the three above services, we expect potential additional search seeds to still provide some benefit but are extremely marginal. Nevertheless, De-Kodi’s design allows for an arbitrarily large set of search seeds to be utilized in future deployments.

6.2.2 Add-on Installation. Tables 2 and 3 (right column) suggest a non-negligible amount of add-ons are failing to install: of the 18,933 add-ons discovered and tested by De-Kodi, 7,821 add-ons’ crawls failed to make it beyond the installation step. The first intuition beyond such “failed-to-install” add-ons is their *staleness*. As a byproduct of Kodi’s own long lifespan, many add-ons are quite old and have multiple release versions from different points in time. Kodi itself is now on version 18 as of the time of this writing. Therefore, it is very possible that old add-ons suffer from compatibility issues with the latest version of Kodi. To see if any installation failures are attributable to staleness, we obtain, for each add-on, the most recently modified date—either the explicit “date modified” value returned in HTTP headers when downloading the add-on’s zip file, or, in the case of github.com hosted add-ons, the date of the most recent commit to the git repository

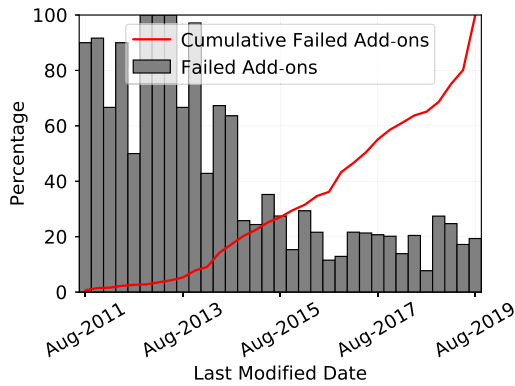


Fig. 8. Percentage of failed add-ons as a function of their staleness (gray barplot) as well as the cumulative percentage of failed add-ons over time (red curve).

from which the add-on was obtained. In this fashion, we were able to obtain modification dates for 16,709 out of the total 18,933 add-ons we discovered.

Figure 8 shows the percentage of failed add-ons as a function of their *staleness* (gray barplot), as well as the cumulative percentage of failed add-ons over time (red curve). If we focus on add-ons last modified before 2014, the figure shows failure rates between 40 and 90%. This ratio drops to 20%, on average, when we focus to the last couple of years. This result confirms our intuition that older add-ons are more prone to fail, likely due to incompatibility issues rather than limitations of De-Kodi. The figure also shows (red curve) that these add-ons constitute about 50% of total “failed-to-install” add-ons, i.e., about 3,900 add-ons which are more than two years old.

To further understand the root causes beyond installation failures, we compare each failed add-on’s dependencies (obtained via the add-on’s `addon.xml` file) with the set of add-ons accessible to the add-on’s Docker container at the time of installation. We identified 949 add-ons whose failed installations are attributed directly to missing required dependencies. Next, we leverage the Tesseract OCR engine [17] to extract text from screenshots of Kodi taken by De-Kodi near the time of each installation’s failure. Our OCR analysis revealed an extra 103 add-ons with additional dependency related issues; specifically, Kodi entered a state asking the user for permission to download additional add-ons in order to install the add-on or feature of interest. Although not shown due to space limitations, 70% of these combined 1,052 add-ons date to no more than two years back. This further strengthens our above incompatibility claim: very stale add-ons are so disconnected with current Kodi APIs that they even fail using the platform to correctly report errors.

Across these 1,052 add-ons, the unique set of missing dependencies was only 334, and only 35 of said dependencies remained undiscovered by De-Kodi by the end of our crawl. It is thus possible to improve De-Kodi by retroactively addressing missing dependencies upon discovery. This has the potential to increase the scope of De-Kodi’s overall coverage, e.g., when failing to respond to a dialog asking permission to install some dependencies, as well as to offer a useful service to Kodi users, e.g., when an add-on lacks an important dependency.

6.2.3 Media Finding. Once an add-on is installed, De-Kodi attempts to find playable media through the add-on. An add-on typed as “`xbmc.python.pluginsource`” (according to its `addon.xml` file) can contain music, videos, pictures, or some executable application. Throughout this article, we refer to such add-ons as *media add-ons*. When De-Kodi encounters a media add-on, it attempts to browse that add-on until it finds videos or music. It is first worth noting that the current version of De-Kodi is not capable of identifying when pictures or executables are opened

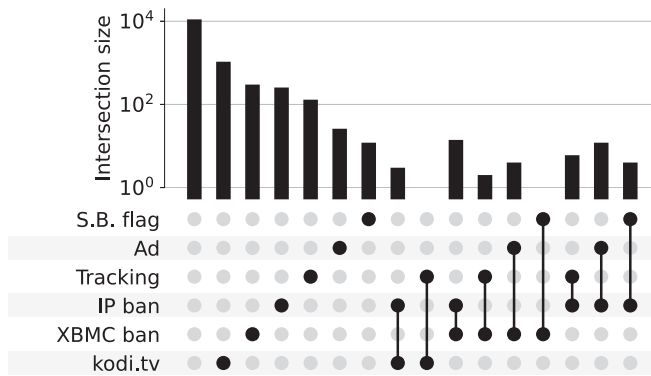


Fig. 9. UpSet plot of a number of add-ons (y-axis) with each tag or combination of tags. Each bar’s corresponding tags are marked with a black dot. The bar with no dots corresponds to untagged add-ons.

through the add-on. If an add-on does not provide music or video, it would appear that De-Kodi failed to find content that should not be expected to exist. Additional metadata provided by an add-on’s `addon.xml` data often provides this information. Of the 5,435 media add-ons discovered by De-Kodi, 4,123 claimed to provide video, 356 claimed to provide music, and 1,046 add-ons made no claims regarding video or music. Note that the sets providing music and video are overlapping.

Beyond this, the possibilities regarding media exploration failures are broad. From manual inspection, we observed that many add-ons require subscriptions to third-party downloading and streaming services such as Real-Debrid [16]. Others may be attempting to dynamically pull content lists from defunct web resources. While we do not attempt to provide or test for an exhaustive list of such scenarios, we have designed De-Kodi to be easily extendable to handle new interaction requirements. Some add-ons point to content not available in the region where this experiment was performed. In the context of this article, all media content identified by De-Kodi was *freely accessible* to De-Kodi, presenting no required sign-in, payment, or otherwise complex barrier.

6.3 Add-ons

6.3.1 Classification. We here clarify and formally “tag” add-ons based on three systems: (a) the type/function of the add-on, (b) the approval of the Kodi official, (c) the safety of the add-ons.

Add-on Types/Function—As per Kodi officials, Kodi add-ons can be classified as follows [20]. A **repository (repo)** add-on is an add-on manager that contains one or more add-ons allowing users to download, install, and update add-ons. A **plugin** add-on is a script or module that adds to Kodi’s functionalities, which offers media content like audio and video. A **script** add-on is a **runnable** program that mainly serves as a support for other add-ons (referred to as a module) or as a standby program that starts at either login or startup (referred to as a service). A **skin** add-on provides a customization of Kodi’s user interface. Finally, a **resource** add-on provides additional files such as languages, fonts, and so on.

Kodi Official Approval—Kodi official maintains a repository in which all the add-ons are manually checked by the Kodi official team. The checking usually includes the purpose of the add-on and if there are any copyrighted content provided. We tag these add-ons as *kodi.tv*. The Kodi official also maintains another list of banned add-ons mainly because of copyright violation. We tag these add-ons as *XBMC banned*. The add-ons that are in neither repository/list are tagged as *unofficial*.

Add-on Safety—We tag the add-ons based on the “suspicious” behaviors (ads injection, tracking, and potential relationship to malware distribution) which are very much rumored in the Web community. We specify the suspicious behaviors as follows:

- **Ads & Tracking**—In order to identify tracking and ad traffic, we match our traffic against EasyList and EasyPrivacy (both maintained by [26]), state-of-the-art lists of advertisement and tracking URLs, used by the most popular adblockers. We identify 5,247 add-ons that trigger EasyList (advertisement) and 141 add-ons that trigger EasyPrivacy (tracking).
- **Malware**—We investigate potentially malware distributing add-ons by matching the URLs they contacted against the Google Safe Browsing hash, which matches URLs against current known threats and malware [31]. Note that Google’s Safe Browsing hashes malicious URLs generally encountered via web browsing and may not necessarily address threats that operate outside of that space, such as botnets. The number of add-ons triggering the Safe Browsing hash is plotted in Figure 9 as “S.B. flag”. To increase coverage, we also compare each observed IP address against FireHOL, an automatically updated aggregator of several actively maintained IP address banlists [28], labeled in Figure 9 as “IP ban”. We found 13 add-ons to serve URLs, which Google SafeBrowsing labels as “social engineering” threats, and 131 add-ons to access domains resolving to potentially malicious IP addresses.

If an add-on is associated with any of the malware tags, we consider it *unsafe*. Otherwise it is considered to be *safe*. In addition, if an add-on is not successfully installed or its source code is not found, we tag it *unknown*.

Tag Overlap—Each add-on will be tagged by all three tagging systems. For the type/function and Kodi official approval systems, the tags are mutually exclusive. However, for the safety tagging system, it is possible for an individual add-on to meet the criteria for multiple tags. Figure 9, formatted as an UpSet plot, shows the extent of overlap between the add-on sets of each aforementioned tag. All shown tag combinations yielded at least one add-on. From the figure, the stark difference between the behavior of add-ons banned by the Kodi Team and the add-ons *endorsed* by the Kodi Team becomes apparent. The add-ons available through the repository distributed with Kodi—labeled “kodi.tv” in Figure 9—overlap only with three add-ons labeled “IP ban”. Conversely, the banned add-on set overlaps with *all four* tags associated with suspicious behavior: tracking, Safe Browsing threats, advertisements, and banned IP addresses. This supports Kodi Team’s claim that their endorsed add-ons behave in a generally legitimate fashion.

In the following paragraphs, we analyze which kind (repository, script, and so on.) of add-on Kodi users install, and how many add-ons actually serve content. We then investigate which add-on category (safe, banned, and so on) is most prominent among SafeKodi users, which the previous section shows being representative of the Kodi ecosystem. We further report on SafeKodi users’ “reaction” to the information learned via SafeKodi, i.e., whether they consider removing add-ons or not and shed some light on *why*.

Figure 10(a) shows the **Cumulative Distribution Functions (CDFs)** of the numbers of different types of add-ons installed by SafeKodi users. The most frequent type is *script* (median of 70 installs per user, mostly modules) which is twice as popular as actually content-serving add-ons: *plugins*. Among the plugins, 85.5% provide video content, while the rest consist of audio (6%) and programs (8.5%). With respect to *repository* add-ons, the median users have installed 13 of them. While this number might seem low, it has to be noted that repositories come equipped with a potentially large number of add-ons. Finally, skins and resource add-ons are overall unpopular as the median installation is only three per user, including two official skins at default. That is to say, nearly half of the users do not seek to custom the interface, while the others may try several skins.

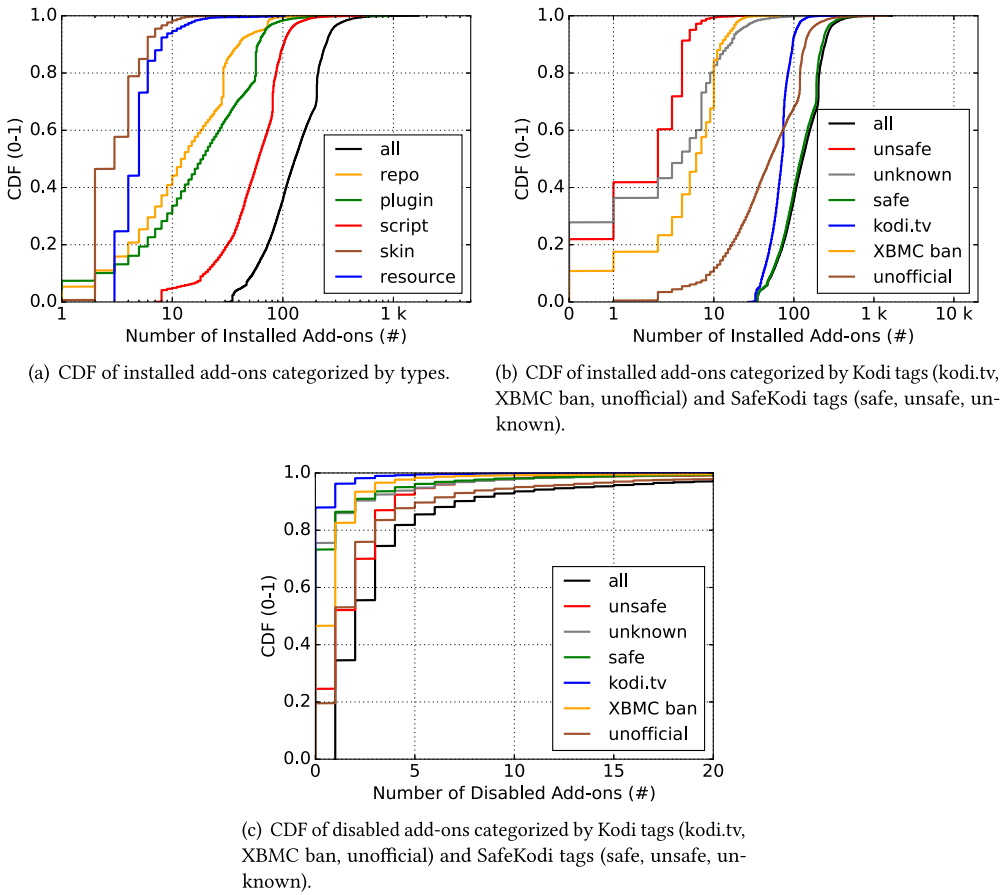


Fig. 10. Characterization of Kodi add-ons.

Figure 10(b) shows the CDF of the number of installed add-ons across SafeKodi users partitioning add-ons based on Kodi-provided tags (Kodi approved, banned, and unofficial, i.e., neither approved nor banned) and De-Kodi-provided tags (safe, unsafe, e.g., tracking or contacting black-listed IP addresses, and unknown). Add-ons tagging is realized considering the final data-set as of Oct 2020; this implies that unknown add-ons are the ones that our on-demand crawling either fail at locating on GitHub or at crawling, e.g., due to some manual interaction required.

The good news from Figure 10(b) is that Kodi users mostly run *safe* add-ons, as shown by the similarity between “safe” and “all” curves. The “median” user install about 140 safe add-ons and only 3–4 *unsafe* add-ons. This is expected given our estimate that only 2% of Kodi add-ons are unsafe, but still concerning given that even a single unsafe add-on is potentially dangerous and only 15% of SafeKodi users only install safe add-ons. The figure also shows the importance of third party add-ons which are not in the default Kodi-approved repository. Indeed, only 47% of the add-ons installed by a user are Kodi-approved, and in 40% of the cases users have more *unofficial* than *official* Kodi add-ons.

Finally, we conclude this subsection commenting on *which* add-ons are disabled as a reaction to the tagging offered by SafeKodi (see Section 4). Overall, one-third of SafeKodi users used our disabling function at least once. Figure 10(c) shows the CDF of the number of disabled add-ons

according to Kodi and De-Kodi tags. The median user only disables 2 add-ons, but 20% of the users disable between 4 and 20 add-ons. Unsafe, unofficial, and XBMC banned add-ons are the ones that are disabled the most, showing the benefit of SafeKodi’s visual tags. Among unsafe add-ons, the majority of disabled add-ons are the one marked to communicate with malicious IP addresses. This happens even for very popular add-ons; for example, among the add-ons disabled by most users we find, in order, *SportsDevil*, *Just4Him* (Adult content), *Exodus* and *IberiKa* (IPTV resources), which each have more than 1,000 installations. Given that, even after multiple months of running SafeKodi, users still have installed 3–4 unsafe add-ons on average, it is fair to say that many Kodi users are willing to take such risk to gain access to the content offered by such add-ons.

6.3.2 Popularity. The previous analysis suggests that the Kodi ecosystem is mostly composed of “safe” add-ons, i.e., add-ons not showing any evident suspicious behavior like tracking or contacting some banned IP addresses. However, this does not imply that Kodi users mostly install and use such safe add-ons. We are thus interested in investigating add-ons popularity, both as a general research question and to estimate the level of exposure to potentially unsafe add-ons.


































For this measurement, we use two metrics: one is search rank with Microsoft Azure, which provides a web search API powered by Bing [14], and the other is the statistics from SafeKodi. For the search rank, Bing was selected since no other major search engine provides the same functionality. We estimate add-on popularity by counting the number of web search results appearing when searching for an exact match of the add-on ID. To reduce potential for false positives, we also require the appearance of either “xbmc” or “kodi” on all web pages contributing to the add-on’s result tally. It has to be noted that this approach is an approximation of add-ons popularity, whose ground truth can only be collected with the global knowledge of all Kodi users. For the statistics from SafeKodi, it helps users detect or avoid potentially unsafe add-ons while opting-in to anonymously report the list of add-ons they have installed. This approach helps us further corroborate on Kodi add-on popularity.







Table 4 shows the top 10 media add-ons with respect to both the number of installs and their search rank in Bing—this combination boils down to 17 unique add-ons. The table is ordered based on the number of installs since search rank is indeed a more arbitrary measure. We mark in blue the new add-ons discovered via SafeKodi, and with an asterisk the *utility* add-ons, i.e., add-ons which mostly offer functionalities used by other add-ons. For instance, *F4mTester* allows decoding F4M files and strip out video streams to be played on Kodi, and as such it is used by many live TV streaming add-ons. Note that *F4MProxy* is a pre-requisite proxy module for *F4mTester* and it has also around 11,000 installations. We have combined both add-ons in the list.

Table 4 contains only three Kodi-approved add-ons: *YouTube*, *Radio_de* (international radio broadcasts) and *HD-Trailers.net* (movie trailers). With the exception of *PlaylistLoader* and *Live Polish Tv*, the rest of the add-ons are banned by Kodi officials. These add-ons provide access to audio and video content with potential copyright infringements. Among these, the two most popular add-ons are *Exodus* (on-demand access to movies) and *SportsDevil* (live streams of most of the major sporting events). *Exodus* has been abandoned by its original developers, but was inherited by other independent developers, who carried out the newer add-on *Exodus Redux*. We have combined both add-ons in the list.

Table 4 shows that search rank is not always accurate when reporting on add-ons popularity. For example, *YouTube* is the most popular add-on among SafeKodi users despite having one third of the search rank of *Exodus*, when associated with the “Kodi” keyword. This is intuitive since *Exodus* sparks lots of discussions online both on how to use it and copyright concern. Similarly, many new add-ons discovered by SafeKodi have high installation numbers (e.g., 6/5,000 for *The Crew* and *The Magic Dragon*) but very low search ranks.

Table 4. Most Popular Media Kodi add-ons given their Bing Rank and Number of Installations Across SafeKodi Users

Add-on	Num. of Installs	Search Rank	Tag
Youtube	13,588	22,000	 
F4MTester*	10,700	8,790	 
Exodus (Redux)	7,820	63,900	 
SportsDevil	6,810	14,100	  
The Crew	6,715	28	 
The Magic Dragon	5,472	41	 
Venom	4,990	50	 
Seren	4,526	150	 
Rising Tides	4,483	90	 
cCloud TV	4,394	7,710	 
PlaylistLoader*	3,122	24,000	 
Radio_de	1,178	9,080	 
Phstreams	86	2,640	 
HD-Trailers.net	76	5,450	 
ZemTV-shani	62	42,600	 
Live Polish TV	22	5,150	 

In blue, add-ons newly discovered via SafeKodi. The “*” refers to *utility* add-ons, i.e., add-ons which offer functionalities to other add-ons. SafeKodi tags include  safe,  tracking, and  ipban; Kodi tags include  kodi.tv,  XBMC banned, and  unofficial.

With respect to the *legality* of Kodi consumed content, Table 4 shows a mixed result. The top add-on, *YouTube*, is both a safe and Kodi-approved add-on, suggesting that Kodi users value their experience with Kodi and not only the potentially illegal content they can consume on it. Nevertheless, *Exodus* and *SportsDevil* are also in the top 5 add-ons– top 3 if we ignore utility add-ons– and they offer access to copyrighted movies and live sport [27, 40]. Both add-ons are banned by Kodi and tagged as “ipban”, meaning that they communicate with malicious IP addresses (see Section 2). *SportsDevil* is also tagged as “tracking” since it triggers a rule in the EasyList/EasyPrivacy list [26]. These two add-ons alone are installed by 50% of SafeKodi users, showing evidence of a significant privacy/security threat.

We proceed to look at the whole picture of all the identified add-ons. While Kodi offers a range of add-on types, we opt to focus our assessment of popularity on *media* add-ons (video, music, or images) and *repository* add-ons (collection of add-ons). Our reasoning for this is threefold. First, most non-media add-ons only exist to provide support to media add-ons, e.g., content metadata scraping and cosmetic changes to Kodi’s GUI. Second, through manual exploration, we observed that repository add-ons are often touted an ideal starting place for Kodi users, as they can make the installation of all *other* add-ons convenient. Lastly, building upon the second point, one’s choice of repository add-ons is likely illustrative of one’s intended use of Kodi. XBMC banned repositories, for example, often earn their “banned” status for referencing other known illicit add-ons (e.g., add-ons engaging in piracy and other nefarious activity). In general, our choice to narrow the scope of our popularity measurement serves to avoid potential noise added by add-ons unlikely to be directly searched for by real Kodi users.

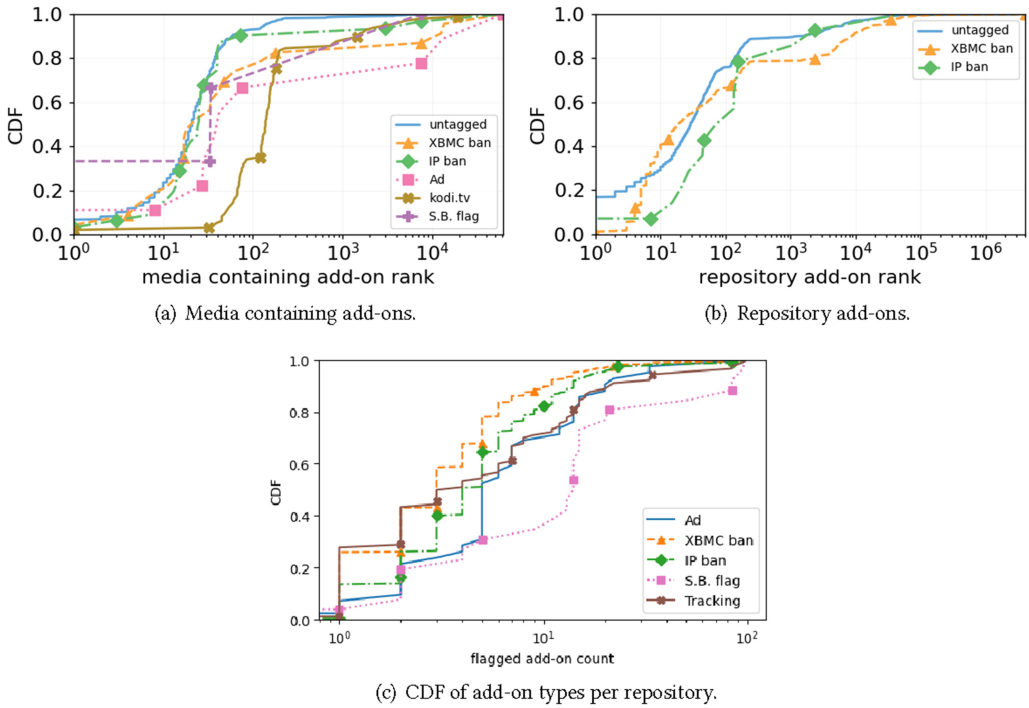


Fig. 11. CDFs of popularity rank.

Figure 11(a) shows the **Cumulative Distribution Function (CDF)** of add-ons *popularity rank* as a function of their classification tag. Overall, the figure shows very skewed distributions with the majority (70–90%) of add-ons having low popularity ranks (~ 200), and the remaining add-ons having ranks up to two orders of magnitude higher. When focusing on the tail of the distributions, we further observe that *XBMC banned* and *tracking* add-ons are one to two orders of magnitude more popular than other add-ons types. A similar trend appears also in Figure 11(b), which shows the CDF of repositories popularity. The figure further shows that the most popular repositories rank two orders of magnitude higher than the most popular media add-ons. This analysis indicates that a “typical” Kodi user has a higher probability to stumble upon a particular repository than a given add-on.

Finally, we pause to consider the implications of observed add-on popularity with respect to De-Kodi. The long-tailed distribution of add-on popularity suggests that, in general, most Kodi users may be turning to the same, small handful of particularly popular add-ons (the top 1%–10% in terms of popularity), which we will loosely refer to as “tier 1” add-ons. The majority of add-ons—most of which are orders of magnitude less popular than tier 1—likely each individually either serve a small minority of users or otherwise are obtained *through* tier 1 repositories. By this logic, it is fair to reason that the marginal significance of each additional add-on potentially missed in De-Kodi’s coverage decreases rapidly; the harder the add-on is for De-Kodi to find, the less likely, we postulate, it is for a given Kodi user to stumble upon the same add-on.

6.3.3 Shared Distribution. Are some add-ons “guilty by association”? While slanderous to label the Kodi community at large as nefarious actors, what may be more productive is an investigation of smaller ecosystems and distribution channels that exist within factions of the Kodi community. Specifically, in this subsection, we aim at quantifying the intuition that add-ons with similar

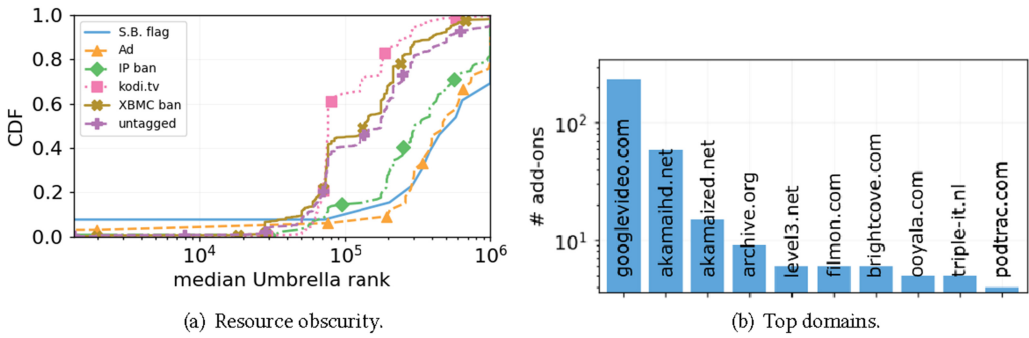


Fig. 12. Content providers.

purposes—for example, piracy, malware, and so on—will naturally congregate together. This serves two vital functions. First, it provides the unsavvy Kodi user with empirically backed reasoning to help them in deciding what add-ons to download or install. In addition, better understanding of how add-ons pool together according to their nature may advance future work concerning threat detection and online content analysis.

For this analysis, we refer to five of our add-on tags—S.B flags, advertisement, IP ban, and XBMC ban—as *undesirable* flags. For each line in Figure 11(c), we plot the number of undesirably flagged add-ons distributed through a repository where at least one add-on of the line’s indicated tag was observed. In other words, if we see a repository has an add-on tagged for XBMC, we want to know how many undesirably flagged add-ons in general, are provided by that repository. The figure shows that *any* one undesirably flagged add-on in a repository has a very low probability of being the *only* add-on. In more than 85% of cases, undesirable add-on container repositories and sources contained *at least* two such add-ons. Most notably, the presence of a single SafeBrowsing flag is a strong indicator of 10 or more undesirably flagged add-ons cohabiting the same repository.

6.4 Content Providers

We here investigate the *providers* behind Kodi content, i.e., the domains where Kodi content (add-ons, repositories, and media) is hosted. We obtain, for each add-on, the set of domains it accesses. Next, we use Cisco’s Umbrella top 1 million [10]—which ranks domain names by the frequency with which the Cisco Umbrella global network receives queries for each name—to *rank* the domains contacted by Kodi. Figure 12(a) plots the *median* Umbrella rank per add-on. A clear pattern emerges, dividing our add-on tags into two behavioral groups. We see that, in general, add-ons tagged for banned IP addresses, Safe Browsing threats, and advertisements all tend toward using very unpopular domain names. Much of the fourth quartile (beyond the 75th percentile) of these add-ons use domains so obscure that their medians fall *beyond* the least popular domains ranked by the Umbrella top 1 million (i.e., their Umbrella rank falls outside of 1 million). We postulate that the providers of the content consumed by these add-ons place high priority on deliberate obscurity (to avoid detection of nefarious activity) and low costs (as opposed to using potentially more expensive, well-known infrastructure platforms).

Conversely, Figure 12(a) shows that only a small fraction of add-ons with other tags (kodi.tv, XBMC banned, and untagged) have median domains less popular than the top 1 million. More than 40% of the add-ons in this latter group of tags have median domain ranks that fall within the top 10,000. Surprisingly, along this dimension, we see banned add-ons behaving similarly to kodi.tv add-ons, suggesting they may have comparable or even overlapping infrastructure. To investigate this, we plot the number of media containing add-ons using each of the top 10 media

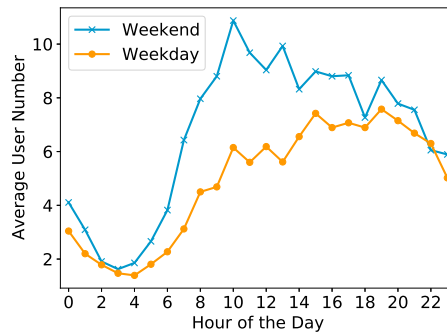


Fig. 13. Local accessing times of the users.

servicing second-level domains (ranked by the number of media containing add-ons using at least one media URL from each domain) in Figure 12(b). Add-ons tend to have little overlap in the set of domains hosting their content, as seen in the figure. However, we see that over 200 add-ons employ “googlevideo.com”, an alias utilized by YouTube for streaming-related network requests.

6.5 User Characterization

To capture Kodi usage in the wild, we have asked SafeKodi users to share some of their local settings via extra permission (see Figure 4). Out of 15,768 SafeKodi users, 6,015 (or about 38%) have opted-in, offering a statistically valuable sample. Most users report Kodi version 18.0+, with a peak (1,083 users) on Kodi 18.6 which is the one adopted by De-Kodi. The latest stable version is 18.8, which is installed by 585 users, followed by 18.7 with 583 users. A handful of users run quite old versions (e.g., 15.x and 16.x from 2015) as well as versions under current development (19.0 and 21.1).

We use the screen resolution reported to conjecture on the potential device used. Half of the SafeKodi users report a screen resolution of 1080p, typical of HD TVs or computer desktops. About 15% of users have a higher resolution, mainly 2160p or 4K, suggesting running Kodi along with an Ultra HD TV unit. Lower resolutions are also quite popular (5% at 480p, and 20% at 720p) suggesting potential mobile devices.

Next, we find that 771 out of 6,015 users have enabled remote control of their Kodi box, i.e., they allow Kodi to spin a simple webserver on port 8,080, unless changed by the user. According to [61], this setting enables full control of the Kodi application. An adversary can achieve any function on Kodi through the RESTful interface, including reading and writing arbitrary files from/to the machine Kodi is running on. It has to be noted that we did not collect the port information, neither keep their IP addresses to verify on public repositories (like Shodan [39]) if these Kodi instances are indeed accessible via the Internet.

Finally, Figure 13 shows *when* during a day users access SafeKodi, which implies they are actively running Kodi. It has to be noted that this results in an approximation of Kodi usage, given that Kodi users do not run SafeKodi each time they use Kodi. However, this is the best we can do to guarantee user privacy, and it still offers an interesting insight with respect to the usage trend. The figure shows that Kodi is more used on weekends than during the weekdays – as typical of generic residential traffic [69]. The usage pattern between weekday and weekend is also quite different. While during the week, usage starts increasing from early morning (4–5 am) until the peak at around 7 PM. During weekends, we instead notice a peak around 10 AM, after which Kodi usage decreases as time goes by, until the next morning.

7 TRAFFIC IDENTIFICATION

In this section, we evaluate the Kodi traffic classification algorithm as proposed in Section 4.5.

7.1 Evaluation Settings

We consider two scenarios for traffic collection and identification: LAN and WAN. As discussed above, the LAN is a less challenging environment given that: (1) we have access to mDNS traffic, (2) each machine is uniquely identified by its local IP address. Accordingly, traffic from multiple machines have no impact on the detection algorithm when running from within a LAN.

We emulate LAN and WAN using a single machine. We build a LAN hosting on a single virtual machine, and then leverage the *host* machine as a node in the WAN. The VM NAT blocks any internal traffic (e.g., mDNS and ARP) to the host. Traffic is collected using WireShark inside the VM (to emulate traffic collection inside the LAN) and the host (to emulate traffic collection inside the WAN).

At the host, we run multiple software including music and video streaming, chat, and Web browsing. In the VM, we only run Kodi which is representative of, for instances, TV boxes equipped only with Kodi. An alternative scenario consists of a user running Kodi on a laptop alongside other software. We ignore this scenario since it represents a simplified version of the WAN scenario, thanks to the availability of mDNS traffic. We instrument Kodi to launch and perform specific *actions*, e.g., install or run an add-on. To illustrate the effectiveness of our algorithm, we focus on the top 10 most popular media add-ons which are used by 91% of SafeKodi users.

We run Algorithm 1 on the packets captured both inside the VM and the host, approximating LAN and WAN respectively. We compute the *accuracy* of the algorithm as the ratio of the correctly identified packets (true positives) over the true total Kodi traffic. For this, we rely on manual labeling. We compute the *recall* as the ratio of the true positive packets over the positive (identified) packets. Finally, we use *F1 score*⁷ as a performance metric of our algorithm. We consider the F1 score both in terms of number of packets and traffic volume. Finally, we also compute the *lag* of the algorithm, i.e., the difference between the timestamps of the first identified packet and the first true Kodi packet, as well as that between the last identified packet and the last true Kodi packet.

7.2 Results

Table 5 summarizes the results of our experiments. The first row focuses on three simple actions. First, we verify that in absence of Kodi traffic no detection is indeed triggered. Next, we evaluate the detection of simply running Kodi. This actions show a perfect F1 score from the LAN, regardless of whether we focus on the packet or traffic count. Accordingly, no lag is detected (both at start and end of the action) since all packets are correctly detected. This detection is purely based on mDNS, and for this reason it is not possible from the WAN. When Kodi is updating its add-ons, we also achieve perfect detection both from the LAN and from the WAN.

We now focus on the *install* operation. Overall, Table 5 indicates that the algorithm accurately identifies (F1 score > 0.9) most packets when installing different add-ons, either from the official repository (e.g., *YouTube*) or from third-parties (e.g., *F4MTester*). Without extra traffic (LAN), the algorithm identifies all packets/traffic with no lag, the only exceptions being *Rising Tides* and *cCloudTV* due to some destination URLs not being recorded in our database. This suggest that by increasing our crawling frequency and depth, i.e., by budgeting more time to crawl an add-on, this penalty can be mitigated.

We now focus on the *run* operations. From within the LAN, the algorithm accurately identifies all (*YouTube*, *Rising Tides* and *Seren*) or most (*Exodus Redux*, *Magic Dragon*, and *cCloud TV*) of the

⁷F1 Score = $\frac{2 \cdot \text{accuracy} \cdot \text{recall}}{\text{accuracy} + \text{recall}}$.

Table 5. Performance Evaluation of Kodi Detection Algorithm Assuming Passive Traffic Collection at a LAN and WAN

Add-on	Tag	Operation	Classified Action	F1 of Pkt Num (LAN)	F1 of Pkt Vol (LAN)	Lag (LAN)	F1 of Pkt Num (WAN)	F1 of Pkt Vol (WAN)	Lag (WAN)
Do not run Kodi	-	-	-	-	-	-	-	-	-
Run Kodi only	✓	✓	1.000	1.000	(0, 0)	-	-	-	-
Update add-ons	✓	✓	1.000	1.000	(0, 0)	1.000	1.000	(0, 0)	-
YouTube*	❖ ✓	Install Run	✓ ≥	1.000 1.000	1.000 1.000	(0, 0) (0, 0)	1.000 0.659	1.000 0.621	(0, 0) (0, 23)
F4MTester	⊗ ✓	Install Run	✓ ✓	1.000 0.716	1.000 0.770	(0, 0) (0, 0)	1.000 0.671	1.000 0.603	(0, 0) (0, 6)
Exodus Redux	⊗ Ⓜ	Install Run	✓ ≥	1.000 0.989	1.000 0.994	(0, 0) (0, 0)	1.000 0.868	1.000 0.814	(0, 0) (0, -18)
SportsDevil	⊗ Ⓜ ▲	Install Run	✓ ✓	1.000 0.832	1.000 0.730	(0, 0) (0, 0)	1.000 0.814	1.000 0.736	(0, 0) (0, 0)
The Crew*	⊗ ✓	Install Run	✓ ✓	1.000 0.801	1.000 0.356	(0, 0) (0, -1)	1.000 0.799	1.000 0.392	(0, 0) (0, -11)
Magic Dragon	⊗ ✓	Install Run	✓ ≥	1.000 0.937	1.000 0.961	(0, 0) (0, 0)	1.000 0.937	1.000 0.961	(0, 0) (0, 0)
Venom	⊗ ✓	Install Run	✓ ≥	1.000 0.846	1.000 0.956	(0, 0) (0, 0)	1.000 0.846	1.000 0.956	(0, 0) (0, 0)
Seren*	⊗ ✓	Install Run	✓ ✓	1.000 1.000	1.000 1.000	(0, 0) (0, 0)	1.000 0.978	1.000 0.990	(0, 0) (0, 3)
Rising Tides	⊗ ✓	Install Run	✓ ✓	0.800 1.000	0.820 1.000	(0, 0) (0, 0)	0.772 0.324	0.800 0.909	(0, 0) (0, -1)
cCloud TV	⊗ ✓	Install Run	✓ ≥	0.988 0.993	0.993 0.994	(0, 0) (0, 0)	0.714 0.902	0.935 0.980	(0, 0) (0, 6)

The lags are expressed as two integers in round brackets, corresponding to the lag from the start and the end of an action, respectively. For the add-ons with ^{***}, a *hint* that Kodi was running, e.g., Kodi was recently launched, is used to help with detection in the WAN. SafeKodi tags include ✓ safe, ▲ tracking, and Ⓜ ipban. Kodi tags include ❖ Kodi official, ⊗ Kodi banned, and ☹ unofficial.

packets/traffic. An exception is *The Crew*, for which we measure an F1 score of 0.801 and 0.356 for packets and traffic, respectively. As above this is due to missing some destination URLs/IP addresses which carry large traffic volume. When including such missing URLs/IP addresses in our database, we achieve F1 score of 1 and 0.997 for packets and traffic, respectively. For *YouTube*, the table shows a large discrepancy between F1 score in LAN (1.0) and WAN (0.6). This happens because many add-ons access YouTube content, which results in YouTube URLs being mapped to tens if not hundreds of add-ons. In this case, when the background “noise” is effectively mapped to some add-ons as well, more misclassification is originated which detracts from YouTube detection. This is also the reason of the long (23 seconds) lag for the end of the detection. For most of the add-ons, we have similar results with a relative lower F1 score to the LAN. The reason also comes from the impact of lower recalls.

Some add-ons in Table 5 are associated with a “*”. These add-ons only use HTTPS (🔒: *Youtube*) or a custom User-Agent (🚗: *The Crew* and *Seren*) in HTTP—thus making them indistinguishable from a browser, for instance. Accordingly, their detection in the WAN is only possible if paired with a *hint*, e.g., a previous indication of an ongoing Kodi session. While in the LAN this hint naturally comes from the frequent mDNS messages, in the WAN it needs to be replaced by events like updates, add-on installations, and so on.

Finally, five out of the top ten add-ons are classified as multiple add-ons, as indicated by “⊃” in Table 5. This happens for add-ons which are more careful with respect to the user privacy (e.g., using HTTPS or a less unique User-Agent) and which serve popular content which is accessed by many add-ons. This is the case of *YouTube*, whose content is accessed by 201 other add-ons, thus causing few false positives with respect to the only classification possible (domain name either via DNS or SNI).

8 CONCLUSIONS

Kodi is a convenient and widely popular media center which mostly aggregates audio/video content on the Web in a simple and extensible interface. Kodi is also the perfect vehicle for copyrighted content, traffic manipulation (ads injection), and even malware execution.

This article introduces the first formal approach to dissecting the behavior of Kodi. By leveraging features of the Kodi platform itself, we were able to build De-Kodi, a full-fledge crawling system for the Kodi ecosystem, spanning thousand of add-ons and content providers. We demonstrate tool’s effectively tunable levels of crawl depth, breadth, and speed, with scalability at the heart of our design, and we make tool publically available to other researchers.

We exploit an overlap of interest between researchers aiming at studying Kodi, and its users looking for a safer approach to Kodi. In February 2020, we have launched *SafeKodi*, a Kodi add-on which warns users about potential threats associated with their installed add-ons. SafeKodi users share with our servers which add-ons they have installed along with local settings which might expose them to security threats. In the background, our servers extensively test these add-ons to verify their behavior, e.g., interaction with blacklisted IP addresses.

Coined as Kodi’s first “antivirus” by several news outlets, SafeKodi currently counts a user-base of more than 15,000 users across 104 countries. Over 7 months, SafeKodi users have reported 2.3 million add-ons which led to the testing, identification, and labeling (safe versus unsafe) of, in joint with an active Web crawl based on the seeds we have identified, about 11,000 unique add-ons. We find that while most add-ons are *safe*, most users have installed at least one *unsafe* add-on. Exposing this information to Kodi users is important, and in fact, one-third of SafeKodi users reacted by disabling unsafe add-ons. Still, many users opted to keep 3–4 unsafe add-ons on their devices, suggesting they value more the offered content than their “safety”.

We further explore the add-on popularity, content provider, and user characterization. We find *YouTube* to be the most popular add-ons installed by SafeKodi users followed by several add-ons that are banned from Kodi official due to copyright infringement. Besides *YouTube*, over 200 add-ons access the YouTube resources, making Google to be the top content provider for Kodi users. This shows that Kodi is not all about illegal content.

Last but not least, we show that Kodi traffic is far from private and easily identifiable in the wild. For example, given that 97.6% of SafeKodi users have enabled Kodi's *auto-update*, they can be passively identified by a third party like an ISP. Detecting the usage of a specific add-on is harder, especially without assuming LAN access, e.g., detection running at a home gateway. Still we found that, especially unofficial add-ons, rely less on HTTPS and often use Kodi's default User-Agent. In combination with the sketchy URLs/IP addresses they tend to contact, this makes them much easier to identify by a snoopy ISP. In our mission to "rescue" Kodi users, we plan to extend and further automate such methodology and offer, via SafeKodi, a new *privacy* label next to their installed add-ons.

APPENDIX

A APPENDIX

The HTTP usage in top 20 official media add-ons and top 20 unofficial (including banned) media add-ons. For each add-on, we investigate if it is HTTP dependent (depend.). If it is, we check if it has customized the HTTP header that does not contain Kodi information. If it does, we further look for if the customization covers all the requests versus falling back to the default Kodi HTTP header at some time.

Table 6. Top 20 Official Media add-ons

Add-on	Num. of Installs	HTTP Depend.	Custom Header	Cover All Requests
YouTube	13,588	×	-	-
NFL	2,517	√	√	√
Radio_de	1,178	√	√	×
Fox News	786	×	-	-
Xumo TV	755	√	√	√
Crackle	651	×	-	-
NBC Live	597	√	√	√
IPlayer WWW	529	×	-	-
Popcorn Film	524	√	√	√
Pluto TV	472	√	√	√
Sound Cloud	414	×	-	-
Daily Motion	390	√	√	√
Live Stream	349	√	√	√
Earth Cam	330	√	√	√
DAZN	275	×	-	-
Snag Films	270	√	√	√
TED Talks	245	√	√	√
iTunes Podcast	224	√	√	√
RedBull TV	191	×	-	-
Classic Cinema	186	√	√	√

Table 7. Top 20 Unofficial Media add-ons

Add-on	Num. of Installs	HTTP Depend.	Custom Header	Cover All Requestss
F4MTester	10,700	√	√	×
Exodus Redux	7,820	√	√	×
SportsDevil	6,810	√	×	-
The Crew	6,715	√	√	√
Magic Dragon	5,472	√	×	-
Venom	4,990	√	√	×
Seren	4,526	√	√	√
Rising Tides	4,483	√	√	×
cCloud TV	4,394	√	√	×
NuMb3r5	4,366	√	√	×
MP3 Streams	4,298	√	√	×
Limitless	4,175	√	√	×
Spotiwa TV	4,166	√	√	√
Video Devil	4,042	√	√	√
UWC	3,927	√	×	-
7 of 9	3,633	√	√	×
Goto	3,473	√	√	×
Deja Vu	3,457	√	×	-
Wolf Pack	3,441	√	√	×
YouTube Music	3,372	√	√	×

REFERENCES

- [1] Crean el primer antivirus para Kodi: protégete de addons con malware. [n. d.]. Retrieved June 2021 from <https://www.adszone.net/noticias/seguridad/safekodi-primer-antivirus-kodi>.
- [2] Is Your Kodi Virus Free? How to Scan With SafeKodi - TROYPOINT Vids. [n. d.]. Retrieved June 2021 from https://www.youtube.com/watch?v=xCW_2v1vkWM.
- [3] Kodi2020 - Novedad para kodi - El antivirus! - tutvboxaldia kodiAndroid. [n. d.]. Retrieved June 2021 from <https://www.youtube.com/watch?v=tLxmJLcaZq4>.
- [4] mwarrior/dekodi. [n. d.]. Retrieved June 2021 from <https://github.com/mwarrior92/dekodi>.
- [5] Safekodi, el addon definitivo si quieres utilizar Kodi de forma segura. [n. d.]. Retrieved June 2021 from <https://www.hobbyconsolas.com/noticias/safekodi-addon-definitivo-quieres-utilizar-kodi-forma-segura-599759>.
- [6] Sandvine 2017. [n. d.]. Spotlight: The “Fully Loaded” Kodi Ecosystem. Available at Retrieved June 2021 from <https://www.sandvine.com/hubfs/downloads/archive/2017-global-internet-phenomena-spotlight-kodi.pdf>.
- [7] Warning - Be Aware What Additional Add-ons You Install. 2016. Retrieved June 2021 from <https://kodi.tv/article/warning-be-aware-what-additional-add-ons-you-install/>.
- [8] Kodi Add-ons Launch Cryptomining Campaign. 2018. Retrieved June 2021 from <https://www.welivesecurity.com/2018/09/13/kodi-add-ons-launch-cryptomining-campaign/>.
- [9] Rampant Kodi Malware? It’s Time to Either Put Up or Shut Up. 2018. Retrieved June 2021 from <https://torrentfreak.com/rampant-kodi-malware-its-time-to-either-put-up-or-shut-up-190610/>.
- [10] Cisco Umbrella Top 1 Million. 2019. Retrieved June 2021 from <https://umbrella.cisco.com/blog/2016/12/14/cisco-umbrella-1-million/>.
- [11] ffmpeg Documentation. 2019. Retrieved June 2021 from <https://ffmpeg.org/ffmpeg.html>.
- [12] Fishing in the Piracy Stream: How the Dark Web of Entertainment is Exposing Consumers to Harm. 2019. Retrieved June 2021 from https://www.digitalcitizensalliance.org/clientuploads/directory/Reports/DCA_Fishing_in_the_Piracy_Stream_v6.pdf.
- [13] Kodi Add-On Developer Arrested On Same Day as Popular Repo Goes Down. 2019. Retrieved June 2021 from <https://torrentfreak.com/kodi-add-on-developer-arrested-on-same-day-as-popular-repo-goes-down-190619/>.
- [14] Microsoft Azure. 2019. Retrieved June 2021 from <https://azure.microsoft.com/en-us/>.

- [15] Popular Kodi Addon ‘Exodus’ Turned Users into a DDoS “Botnet”. 2019. Retrieved June 2021 from <https://torrentfreak.com/popular-kodi-addon-exodus-turned-users-into-a-ddos-botnet-170203/>.
- [16] Real-Debrid. 2019. Retrieved June 2021 from <https://real-debrid.com/>.
- [17] Tesseract Open Source OCR Engine. 2019. Retrieved June 2021 from <https://github.com/tesseract-ocr/tesseract>.
- [18] Tstat - TCP STatistic and Analysis Tool. 2019. Retrieved June 2021 from <http://tstat.polito.it/>.
- [19] XVFB. 2019. Retrieved June 2021 from <https://www.x.org/releases/X11R7.6/doc/man/man1/Xvfb.1.xhtml>.
- [20] Add-on Structure. 2020. Retrieved June 2021 from https://kodi.wiki/view/Add-on_structure#Directory_Name.
- [21] AWS EC2. 2020. Retrieved June 2021 from <https://aws.amazon.com/ec2/>.
- [22] Canadian ISPs Continue Quest To Bankrupt TVAddons, Site That Hosted Tons Of Legal Kodi Addons. 2020. Retrieved June 2021 from <https://www.techdirt.com/articles/20190924/17181743063/canadian-isps-continue-quest-to-bankrupt-tvaddons-site-that-hosted-tons-legal-kodi-addons.shtml>.
- [23] CBlocked Kodi Streams by UK Service Providers: Access More Streams! 2020. Retrieved June 2021 from <https://koditips.com/blocked-kodi-streams-uk/>.
- [24] Online: 18 Million Brits Fall Victim To Counterfeit Electrical Goods Online. 2020. Retrieved June 2021 from <https://www.electricalsafetyfirst.org.uk/media-centre/press-releases/2018/06/online-18-million-brits-fall-victim-to-counterfeit-electrical-goods-online/>.
- [25] Docker. 2020. Retrieved June 2021 from <https://www.docker.com/>.
- [26] EasyList. 2020. Retrieved June 2021 from <https://easylist.to/>.
- [27] Exodus Redux. 2020. Retrieved June 2021 from <https://github.com/I-A-C/I-A-C.github.io/>.
- [28] FireHol IP Lists. 2020. Retrieved June 2021 from <https://iplists.firehol.org/>.
- [29] GeoLite2. 2020. Retrieved June 2021 from <https://dev.maxmind.com/geoip/geoip2/geolite2/>.
- [30] Github. 2020. Retrieved June 2021 from <https://github.com>.
- [31] Google Safe Browsing. 2020. Retrieved June 2021 from <https://safebrowsing.google.com/>.
- [32] HTTPS Encryption on the Web. 2020. Retrieved June 2021 from <https://transparencyreport.google.com/https/overview>.
- [33] Kodi’s JSON-RPC. 2020. Retrieved from https://kodi.wiki/view/JSON-RPC_API/v8.
- [34] LazyKodi. 2020. Retrieved from <https://lazykodi.com>.
- [35] Mitmproxy. 2020. Retrieved June 2021 from <https://mitmproxy.org/>.
- [36] Pirate TV Box Seller Sentenced to 16 Months in Jail. 2020. Retrieved June 2021 from <https://torrentfreak.com/pirate-tv-box-seller-sentenced-to-16-months-in-jail-180820/>.
- [37] Reddit. 2020. Retrieved June 2021 from <https://reddit.com>.
- [38] SafeKodi. 2020. Retrieved June 2021 from <https://safekodi.com/>.
- [39] Shodan. 2020. Retrieved June 2021 from <https://www.shodan.io/>.
- [40] SportsDevil. 2020. Retrieved June 2021 from <https://github.com/AsvpArchy/plugin.video.SportsDevil/>.
- [41] TVAddons Returns, But in Ugly War With Canadian Telcos Over Kodi Addons. 2020. Retrieved from <https://torrentfreak.com/tvaddons-returns-ugly-war-canadian-telcos-kodi-addons-170801/>.
- [42] Timm Böttger, Félix Cuadrado, Gianni Antichi, Eder Leão Fernandes, Gareth Tyson, Ignacio Castro, and Steve Uhlig. 2019. An empirical study of the cost of DNS-over-HTTPS. In *Proceedings of the Internet Measurement Conference*. 15–21. DOI : <https://doi.org/10.1145/3355369.3355575>
- [43] Cisco. 2020. A New Paradigm for Dealing with Illegal Redistribution of Content. 2020. Retrieved June 2021 from <https://blogs.cisco.com/sp/a-new-paradigm-for-dealing-with-illegal-redistribution-of-content>.
- [44] Andrew Clay. 2011. *Blocking, Tracking, and Monetizing: YouTube Copyright Control and the Downfall Parody*. Institute of Network Cultures: Amsterdam.
- [45] Christian Dewes, Arne Wichmann, and Anja Feldmann. 2003. An analysis of Internet chat systems. In *Proceedings of the 3rd ACM SIGCOMM Internet Measurement Conference*. 51–64. DOI : <https://doi.org/10.1145/948205.948214>
- [46] Yuan Ding, Yuan Du, Yingkai Hu, Zhengye Liu, Luqin Wang, Keith Ross, and Anindya Ghose. 2011. Broadcast yourself: Understanding YouTube uploaders. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*. ACM, 361–370.
- [47] Lucas Hilderbrand. 2007. YouTube: Where cultural memory and copyright converge. *FILM QUART* 61, 1 (2007), 48–57.
- [48] Luke Hsiao and Hudson Ayers. 2019. The price of free illegal live streaming services. arXiv:1901.00579. Retrieved from <http://arxiv.org/abs/1901.00579>.
- [49] Damilola Iboisola, Benjamin Steer, Alvaro Garcia-Recuero, Gianluca Stringhini, Steve Uhlig, and Gareth Tyson. 2018. Movie pirates of the caribbean: Exploring illegal streaming cyberlockers. In *Proceedings of the International AAAI Conference in Web and Social Media*.
- [50] Dilip Antony Joseph, Arsalan Tavakoli, and Ion Stoica. 2008. A policy-aware switching layer for data centers. In *Proceedings of the ACM SIGCOMM 2008 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. 51–62. DOI : <https://doi.org/10.1145/1402958.1402966>

- [51] Thomas Karagiannis, Andre Broido, Michalis Faloutsos, and Kimberly C. Claffy. 2004. Transport layer identification of P2P traffic. In *Proceedings of the 4th ACM SIGCOMM Internet Measurement Conference*. 121–134. DOI : <https://doi.org/10.1145/1028788.1028804>
- [52] Thomas Karagiannis, Konstantina Papagiannaki, and Michalis Faloutsos. 2005. BLINC: Multilevel traffic classification in the dark. In *Proceedings of the ACM SIGCOMM 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. 229–240. DOI : <https://doi.org/10.1145/1080091.1080119>
- [53] Hyunchul Kim, Kimberly C. Claffy, Marina Fomenkov, Dhiman Barman, Michalis Faloutsos, and KiYoung Lee. 2008. Internet traffic classification demystified: Myths, caveats, and the best practices. In *Proceedings of the 2008 ACM Conference on Emerging Network Experiment and Technology*. 11. DOI : <https://doi.org/10.1145/1544012.1544023>
- [54] Tobias Lauinger, Kaan Onarlioglu, Abdelberi Chaabane, Engin Kirda, William Robertson, and Mohamed Ali Kaafar. 2013. Holiday pictures or blockbuster movies? Insights into copyright infringement in user uploads to one-click file hosters. In *Proceedings of the 16th International Symposium on Research in Attacks, Intrusions, and Defenses - Volume 8145*. Springer-Verlag New York, Inc., New York, NY, 369–389. DOI : https://doi.org/10.1007/978-3-642-41284-4_19
- [55] Wei Li and Andrew W. Moore. 2007. A machine learning approach for efficient traffic classification. In *Proceedings of the 15th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*. 310–317. DOI : <https://doi.org/10.1109/MASCOTS.2007.2>
- [56] Chaoyi Lu, Baojun Liu, Zhou Li, Shuang Hao, Hai-Xin Duan, Mingming Zhang, Chunying Leng, Ying Liu, Zaifeng Zhang, and Jianping Wu. 2019. An end-to-end, large-scale measurement of DNS-over-encryption: How far have we come?. In *Proceedings of the Internet Measurement Conference*. 22–35. DOI : <https://doi.org/10.1145/3355369.3355580>
- [57] Aniket Mahanti, Niklas Carlsson, Martin Arlitt, and Carey Williamson. 2012. Characterizing cyberlocker traffic flows. In *Proceedings of the 37th Annual IEEE Conference on Local Computer Networks*. IEEE, 410–418.
- [58] Srdjan Matic, Costas Iordanou, Georgios Smaragdakis, and Nikolaos Laoutaris. Identifying sensitive URLs at web-scale. In *Proceedings of the 20th ACM SIGCOMM Internet Measurement Conference*.
- [59] Andrew W. Moore and Konstantina Papagiannaki. 2005. Toward the accurate identification of network applications. In *Proceedings of the International Workshop on Passive and Active Network Measurement*. 41–54. DOI : https://doi.org/10.1007/978-3-540-31966-5_4
- [60] Andrew W. Moore and Denis Zuev. 2005. Internet traffic classification using bayesian analysis techniques. In *Proceedings of the International Conference on Measurements and Modeling of Computer Systems*. 50–60. DOI : <https://doi.org/10.1145/1064212.1064220>
- [61] Alexios Nikas, Efthimios Alepis, and Constantinos Patsakis. 2018. I know what you streamed last night: On the security and privacy of streaming. *Digital Investigation* 25 (2018), 78–89. DOI : <https://doi.org/10.1016/j.diin.2018.03.004>
- [62] E. Rescorla. 2020. The Transport Layer Security (TLS) Protocol Version 1.3. 2020. Retrieved June 2021 from <https://tools.ietf.org/html/rfc8446>.
- [63] E. Rescorla, K. Oku, and N. Sullivan and. 2020. TLS Encrypted Client Hello Draft-ietf-tls-esni-07. 2020. Retrieved June 2021 from <https://tools.ietf.org/html/draft-ietf-tls-esni-07>.
- [64] Sandvine. 2018. Global Internet Phenomena Spotlight - Kodi. 2018. Retrieved June 2021 from <https://www.sandvine.com/hubfs/downloads/archive/2017-global-internet-phenomena-spotlight-kodi.pdf>.
- [65] Subhabrata Sen, Oliver Spatscheck, and Dongmei Wang. 2004. Accurate, scalable in-network identification of p2p traffic using application signatures. In *Proceedings of the 13th international conference on World Wide Web*. 512–521. DOI : <https://doi.org/10.1145/988672.988742>
- [66] Justine Sherry, Shadi Hasan, Colin Scott, Arvind Krishnamurthy, Sylvia Ratnasamy, and Vyas Sekar. 2012. Making middleboxes someone else’s problem: Network processing as a cloud service. *ACM SIGCOMM Computer Communication Review* 42, 4 (2012), 13–24.
- [67] Haifeng Sun, Yunming Xiao, Jing Wang, Jingyu Wang, Qi Qi, Jianxin Liao, and Xiulei Liu. 2019. Common knowledge based and one-shot learning enabled multi-task traffic classification. *IEEE Access* 7 (2019), 39485–39495. DOI : <https://doi.org/10.1109/ACCESS.2019.2904039>
- [68] Pelayo Vallina, Victor Le Pochat, Álvaro Feal, Marius Paraschiv, Julien Gamba, Tim Burke, Oliver Hohlfeld, Juan Tapiador, and Narseo Vallina-Rodriguez. Mis-shapes, mistakes, misfits: An analysis of domain classification services. In *Proceedings of the 20th ACM SIGCOMM Internet Measurement Conference*.
- [69] Florian Wamser, Rastin Pries, Dirk Staehle, Klaus Heck, and Phuoc Tran-Gia. 2011. Traffic characterization of a residential wireless Internet access. *Telecommunication Systems* 48, 1–2 (2011), 5–17.
- [70] XBMC. 2019. Official:Forum Rules/Banned Add-ons. 2019. Retrieved from https://kodi.wiki/view/Official:Forum_rules/Banned_add-ons.

Received 9 June 2021; revised 3 June 2022; accepted 11 September 2022