De-Kodi: Understanding the Kodi Ecosystem

Marc Anthony Warrior
Northwestern University
warrior@u.northwestern.edu

Yunming Xiao
Northwestern University
yunming.xiao@u.northwestern.edu

Matteo Varvello
Brave Software
mvarvello@brave.com

Aleksandar Kuzmanovic
Northwestern University
akuzma@northwestern.edu

ABSTRACT

Free and open source media centers are currently experiencing a boom in popularity for the convenience and flexibility they offer users seeking to remotely consume digital content. This newfound fame is matched by increasing notoriety—for their potential to serve as hubs for illegal content—and a presumably ever-increasing network footprint. It is fair to say that a complex *ecosystem* has developed around Kodi, composed of millions of users, thousands of "add-ons"—Kodi extensions from 3rd-party developers—and content providers. Motivated by these observations, this paper conducts the first analysis of the Kodi ecosystem. Our approach is to build "crawling" software around Kodi which can automatically install an addon, explore its menu, and locate (video) content. This is challenging for many reasons. First, Kodi largely relies on visual information and user input which intrinsically complicates automation. Second, no central aggregators for Kodi addons exist. Third, the potential sheer size of this ecosystem requires a highly scalable crawling solution. We address these challenges with *de-Kodi*, a full fledged crawling system capable of discovering and crawling large cross-sections of Kodi's decentralized ecosystem. With de-Kodi, we discovered and tested over 9,000 distinct Kodi addons. Our results demonstrate de-Kodi, which we make available to the general public, to be an essential asset in studying one of the largest multimedia platforms in the world. Our work further serves as the first ever transparent and repeatable analysis of the Kodi ecosystem at large.

ACM Reference Format:
Marc Anthony Warrior, Yunming Xiao, Matteo Varvello, and Aleksandar Kuzmanovic. 2020. De-Kodi: Understanding the Kodi Ecosystem. In *Proceedings of The Web Conference 2020 (WWW '20), April 20–24, 2020, Taipei, Taiwan.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3366423.3380194

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 [24], 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).

Around Kodi a whole *ecosystem* has been built with several key players: users, addons (plugins), and (content) providers. Kodi users install addons via Kodi's official *repository* (a collection of approved addons) or via third party repositories and sources retrieved on the Web—mostly specialized forums, blogposts, and social media. Installed addons 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.

This work aims to study and quantify the nature of Kodi's ecosystem at large through crawling and analyzing Kodi's *addons*, through which media streaming is facilitated. Although the Kodi platform is designed to be convenient for the typical end user, *crawling* Kodi's addon ecosystem proves extremelly challenging for several key reasons. First, *discovering and locating* Kodi addons is non-trivial, as there exists no global list of Kodi addons. We tackled this challenge by building a scraper which can collect potential Kodi addons from the Web (Github, Reddit, etc.) and quickly reduce them to a unique set of actual addons. Next, we extend Kodi's APIs to allow more informed crawling operations, e.g., by interacting with visual elements while tracking execution path. Finally, we leverage Docker [1] to scale our software while isolating crawler instances from potential malware and/or crashes. The result is 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 thousand of addons can be crawled per day. Further, we show that de-Kodi can effectively explore working addons, and quickly discard erroneous, obsolete (50% of addons in the ecosystem are more than two years old), or otherwise dysfunctional addons which fail to install.

Next, we peform and analyze a full crawl of the Kodi ecosystem from our setup. De-Kodi discovered 9,146 unique Kodi addons (83% more than what is contained in the official Kodi repository) scattered across LazyKodi, a well-known and actively maintained Kodi add-on aggregator, as well as Reddit [12] and GitHub [6], which are known for attracting both Kodi users and developers. Our main findings with respect to the Kodi ecosystem are summarized in the following:

- The Kodi ecosystem largely relies on "free" hosting platforms, such as GitHub and Google CDN.
- The majority of addons do not engage in any "suspicious" activity, e.g., ads and malware injection.
- Very few addons are extremely popular (10x more popular than other addons), and these addons are more likely associated with suspicious activities.
- Lots of content is "stale", *i.e.,* old addons not installing on recent Kodi or unreachable URLs.

2 RELATED WORK

The main contribution of this paper is de-Kodi, a tool facilitating in depth and transparent studies of the Kodi ecosystem. To the best of our knowledge, no previous research paper has investigated this ecosystem yet. Conversely, researchers have directed their attention towards understanding the potential security and privacy threats of the Kodi application [23] as it allows arbitrary code from unknown sources to be executed. The authors show, for instance, how addons 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 addons and comment on the presence of suspicious activity (Section 7.1.1).

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 papers in this area.

Back in 2007–2011, platforms like YouTube and Vimeo were mostly used for redistributing illegal content [18], [16]. Even when legal, the majority of the uploaded content was copied rather than user-generated [17]. 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, etc.

To avoid dealing with copyright detections, "uploaders" directed their attention to *cyberlockers*, or services offering remote file storages, sometimes even for free [22]. In [21], the authors scraped popular cyberlockers, e.g., MegaUpload and RapidShare, and show that 26–79% of the content infringed copyright. More recently, Ibosiola et al. [20] study *streaming cyberlockers*, or illegal websites which distribute pirated content (mostly video). The paper looks at both cyberlockers and the content they serve. Overall, it finds a centralized ecosystem composed of few countries and cyberlockers. Although cyberlockers as a subject are orthogonal to our study, it is worth mentioning that Kodi addons may utilize cyberlockers as sources of content.

An interesting new angle was explored in [19]. In this paper, 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 7.1.1.

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

Addon – An addon is a set of files—code, content, metadata, etc.—which together work to extend the functionality of some Kodi feature, ranging from media access (such as YouTube and Netflix) to Kodi GUI skins and code libraries. An addon's properties, including the set of Kodi features extended, are described by the addon's respective and mandatory `addon.xml` file. In addition to this, many addons 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 addon in question. For convenient distribution, an addon is usually packaged in a zip file; at installation, the zipped addon is extracted into Kodi's local addons directory.

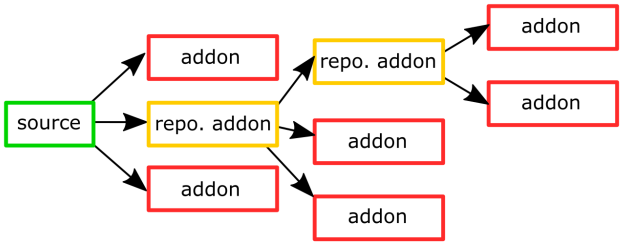


Figure 1: Diagram exemplifying the relationship between source paths, repositories addons (labeled as "repo. addons"), and non-repository addons (labeled as "addons")

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* promotion of a set of add-ons, primarily consisting of add-ons dealing with pirated content [25].

It is worth noting that, as per Kodi's disclaimer,[1] 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 *addons* which can be used to stream both legal (e.g., YouTube and Vimeo) and illegal/pirated (e.g., SportsDevil and Neptune) content.[2]

Repository – A repository is a special type of addon that points to a *collection* of addons such that they can be conveniently installed. Official Kodi is distributed with a single preinstalled repository called "kodi.tv", which only contains addons endorsed by the Kodi team. Anyone can create their own repository to feature the addons of their choice. Some repositories host their content remotely, e.g., on Github or a personal server, as a means to share curated addon 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, etc), many sources act as a means to facilitate addon distribution. Some remotely accessible sources *directly* host addons; 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 1summarizes the relationship between *add-ons*, *repositories*, and *sources*.

Addon manager – The addon manager is an internal Kodi tool which allows users to install addons and repositories. Kodi's addon manager officially provides two approaches to installing addons:

[1] https://kodi.tv/about
[2] http://www.wirelesshack.org/top-best-working-kodi-video-add-ons.html

via repositories (a type of addon pointing to other addons to be conveniently installed) and via sources (direct paths—local or via HTTP—to addon 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 addons (for example, through built-in, Kodi-specific Python modules that make various Kodi features accessible to addon developers). The savvy user may be able to create an addon 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 addon, 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 is trivial—simply respond accordingly to the text in the dialog—this would be devastating to the naïve crawler, as focus is silently and unexpectedly shifted into an unknown state.

Lenient Addon 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 addons to be nontrivial.

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.

Malicious add-ons: Previous work has established the (realized) potential for Kodi add-ons to carry dangerous malware. Kodi addons 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.

4 DE-KODI SYSTEM OVERVIEW

This section presents de-Kodi, the system we have developed to explore the Kodi ecosystem. We first present the detail of de-Kodi's key components, namely the *crawler* and *source finder*. Then, we describe the overall working flow of de-Kodi.

4.1 Crawler

The *crawler* is the core component of de-Kodi. At a high level, its goal is to take an addon 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, etc.

Figure 2a 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 addon 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* [1] (on the 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.

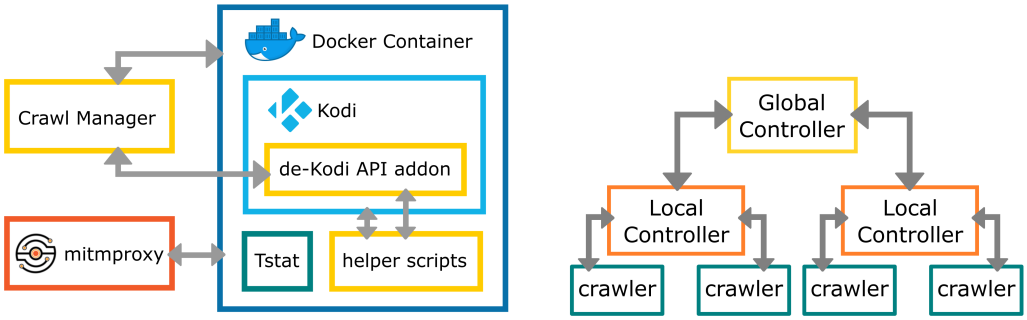
4.1.1 Host. We here describe the crawler's components which run directly on the machine without any OS virtualization (via Docker).

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 addon 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 [10] (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 2a), and 3) the zip file of at least one Kodi addon to be tested. Kodi runs inside the container headlessly via a virtual screen (Xvfb [15]). In the following, we describe in detail de-Kodi's software that runs inside the container.

API addon – 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 API addon is a *service*—meaning it starts automatically when Kodi is launched—that extends Kodi's API to be more automation friendly. The API addon runs an RPC server which receives crawling instructions, e.g., navigate to this menu, from its crawl manager.



(a) Crawler. **(b) Crawling system.**

Figure 2: A visual overview of the de-Kodi system. Figure 2a shows the structure of an individual *crawler*. The *crawlers* in 2b are instances of the *crawler* shown in 2a, but in the case of 2b, we use one instance of *mitmproxy* per machine to capture traffic from all *crawlers*.

We determined, through manual experimentation, how the API addon 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 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 addon content.

TSTAT [14]– 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 addons. 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 addons are available to be tested. This is true for Kodi's official repository, whose addons can easily be installed from any Kodi instance. This assumption does not hold for the larger set of unofficial Kodi addons which are scattered around the Web. This motivates our need to build a *source finder* tool.

Due to the lack of a centralized Kodi addons aggregator, avid Kodi users are forced to socialize to exchange add-ons and sources. We have identified three main places to search for Kodi addons 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 [8], 2) Reddit, a large online social platform with many publicly accessible communities [12], and 3) GitHub, a large online software development platform often used for hosting, maintaining, and distributing open source code [6]. For the remainder of this paper, we refer to these three entities as our "search seeds" or "seeds". In Section 7.1.2, 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 addon 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 addon 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, etc) on both GitHub and Reddit. These links are expected to point either directly to Kodi addons or collections of addons (such collections are often utilized to remotely store the addons pointed to by Kodi repository addons). 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 addons may appear in many repositories. On top of this, outmoded and defunct addon 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" addons by their addon ID and 2) always opting to re-crawl an already crawled addon if a *newer* version is found.

4.3 System 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 2b. First, we start a *global controller* which utilizes previously obtained outputs of a *source finder* to actively discover addons. 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 addon 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 addon 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 addon or a collection of addons. 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 addon id, the list of Kodi features extended by the addon (which we refer to as the addon "type" — note it is possible for an addon to have multiple types), and the addon'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 addon from its required `addon.xml` file as an indication that the downloaded material is not an addon. If no addons are verified from the current link, the local controller repeats this step.

(2) If addons were found in the previous step, the local controller communicates the set of identified addons and their respective data to the global controller. From this set, the global controller removes addons which have been already successfully crawled. The resulting subset of addons is then returned to the local controller which ensures they are packaged 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 addons to be crawled next. If no addons were returned, the local controller returns to the first step.

(3) If addons 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 addon from the set to be crawled. As detailed in subsection 4.1, the crawler then launches a Docker container using the recently created Docker image, attempts to install the addon under test, and finally attempts to find playable media (if appropriate for the addon's determined type). We test discovered media URLs for reachability, geolocalize their respective IPs, and attempt to obtain corresponding video information using ffprobe [4]. Throughout the crawl, the local controller communicates its progress to the global controller, e.g., whether an addon installed successfully or not.

(4) When a crawler completes the crawl of its assigned addon (either from running out of menu items to browse or by reaching a preconfigured timeout capping the amount of time spent on each addon), the local controller closes all of that crawler's active materials (e.g., the Kodi instance, the Docker session, temporary state information, etc). When the number of active crawlers drops below n, the local controller launches new crawlers if there are remaining addons in the current addon 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 addons to its local controller. The local controller then, before closing the crawler, creates a *new* container image so that the newly discovered addons can be crawled. After obtaining permission from the global controller, the local controller then appends these addons (or some subset of these addons, depending on the global controller's response) to its current set to be crawled.

(5) Once the remaining number of addons 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.

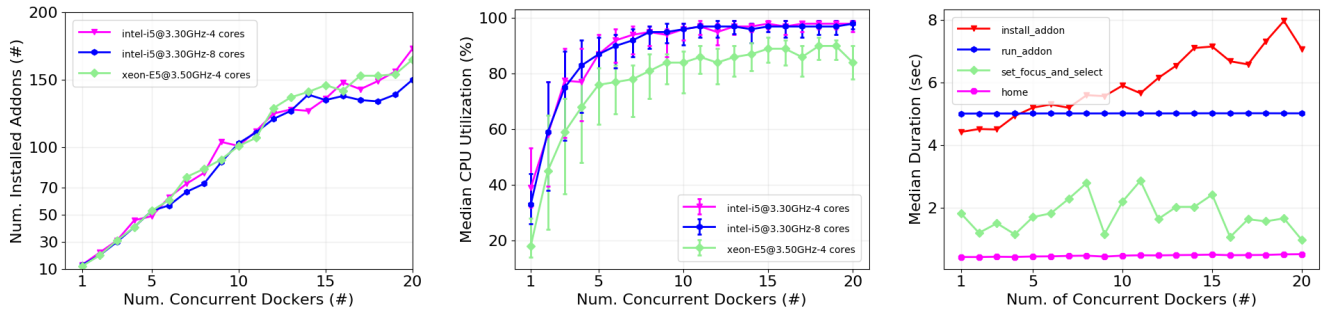
5 DEKODI BENCHMARKING

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 parallelizable, both in terms of Docker instances and number of machines where it can run. We setup three machines[3] 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 addons. Note that in a real crawl, each machine would focus on a different set of addons, 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 addon repo was used for this benchmark.

Figure 3a shows the number of successfully crawled Kodi addons 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 addons grows linearly (between 10 and 100 addons) and no major difference is observable across machines. When $N_{Docker} > 10$, we start observing a sublinear growth in the number of crawled addons 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 term of crawling "speed".

To further understand the previous result, we investigate the CPU utilization during the above benchmarking. Figure 3b 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 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

[3]Two machines mount an Intel i5-4590 (3.30GHz, quadcore); one machine mounts an Intel Xeon E5-1620 (3.50GHz, quadcore). All machines are equipped with 8GB of RAM.



(a) Number of installed addons. **(b) CPU utilization.** **(c) de-Kodi primitives benchmarking.**

Figure 3: de-Kodi benchmarking ; $N_{Docker} = [1:20]$; Crawling-duration: 30 minutes.

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}). The figure shows an overall lower CPU utilization on the (slightly) more powerful machine (xeon-e5) which saturates at 80% versus 90-95% for the other machines.

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 3c 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 addon), 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 500MB ($N_{Docker} = 1$) and a maximum of 4GB ($N_{Docker} = 20$). Based on these empirical results, we set for the crawler a conservative $N_{Docker} = 8$ which should allow us to crawl up to 11,000 addons per day while not overloading the test machines. Note that, in practice, the rate at which *distinct* addons are covered will decline in response to redundant discoveries (if an older addon version is found first), crawl failures (discussed in Section 6.2), and lowered performanced induced by poorly coded addons.

6 DATASET COLLECTION AND VALIDATION

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 a constant 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.

	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	-
Addon zip hosting domains	116	-

Table 1: Crawl summary. Missing fields are "inapplicable".

Table 1 gives, to the best of our knowledge, the first high level overview of today's Kodi ecosystem. The first column shows the total and unique number of discovered entities, e.g., addons 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 addon sources. In addition, we seed de-Kodi with addons from Kodi's official repository: `kodi.tv`. Using this repository and the aforementioned sources, de-Kodi ultimately discovered 9,146 distinct addons, including 1,008 "kodi.tv" addons, as well as 172 "banned" addons, *i.e.,* addons associated with illicit activity and formally denounced by the XBMC Foundation [25]. The crawl discovered 5,435 "pluginsources" addons, *i.e.,* potential media yielding addons, of which only 423 yielded at least one pointer to streamable content. This number is a potential lower bound since navigating Kodi addons is hard and time consuming, and was not the main goal of this crawl. Nevertheless, the crawl yielded 6,117 URLs pointing to audio/video content, spanning 885 fully qualified domain names and 517 (1,147) unique second-level domains (SLDs).

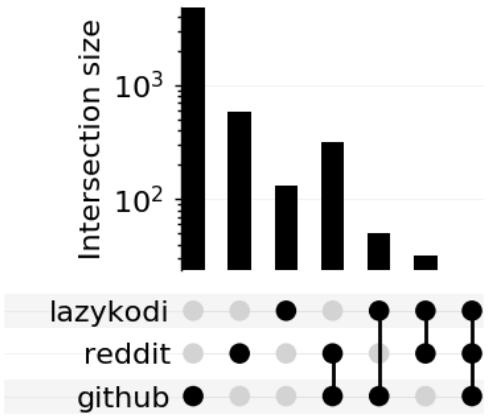


Figure 4: UpSet plot of addons directly found across search seeds. Each bar's respective seeds are marked with a black dot.

6.2 Validation

We can describe the *goals* of de-Kodi's functionality in terms of three chief concerns: addon discovery, addon installation, and media finding. We assess de-Kodi's current abilities with regard to each of these goals below.

6.2.1 Addon Discovery. Any attempt to crawl a large ecosystem such as Kodi leans heavily on the assumption that one has a means 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 addons. Lacking a global view of the set of addons comprising Kodi's ecosystem renders this challenge nontrivial. Therefore, understanding the coverage of our search seeds, introduced in Section 4.2 as de-Kodi's gateway to addon discovery, is essential in assessing de-Kodi's crawling capabilities.

Figure 4 quantifies the number of addons/repos discovered *directly*, *i.e.,* the first layer from the tree in Figure1, via each search seed. We treat addons with matching addon IDs as equivalent discoveries. As depicted in the figure, de-Kodi found 4,887 addons through GitHub—more than half of our total discovered addons[4]—593 through Reddit, and 132 through LazyKodi. Twenty four addons appeared in the search results of all three seeds. The apparent bias in addon discovery towards the GitHub seed highlights the common practice of Kodi users to leverage GitHub as a free hosting service for Kodi repositories. Meanwhile, Figure 4 also draws attention to the dangers of relying on a *single* seed. In the case of our crawl, only 321 of the 593 addons discovered directly through Reddit were also found in GitHub's results, meaning 272 addons may have been missed without seeding Reddit in parallel to GitHub. Despite its small size, Lazikodi still produces a handful of addons 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 extremely marginal. Nevertheless. De-Kodi's design allows for an arbitrarily large set of search seeds to be utilized in future deployments.

[4]The remainder half of discovered addons lie in the successive layers of the tree in Figure1.

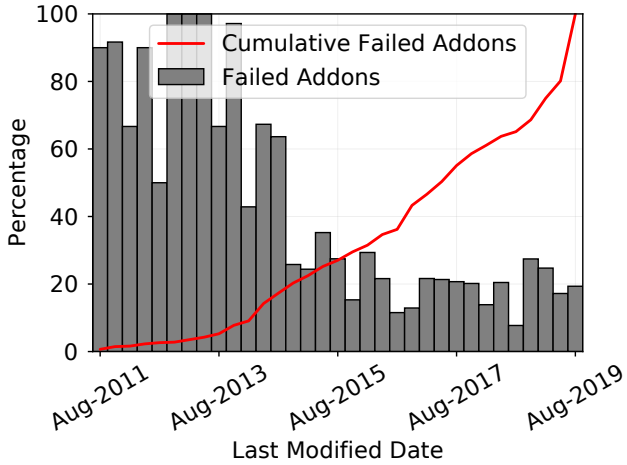


Figure 5: Percentage of failed addons as a function of their staleness (gray barplot) as well as the cumulative percentage of failed addons over time (red curve).

6.2.2 Addon Installation. Table 1 (right column) suggests a non-negligible amount of addons are failing to install: of the 9,146 addons discovered and tested by de-Kodi, 3,881 addons' crawls failed to make it beyond the installation step. The first intuition beyond such "failed-to-install" addons is their *staleness*. As a byproduct of Kodi's own long lifespan, many addons 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 addons suffer from compatibility issues with the latest version of Kodi. To see if any installation failures are attributable to staleness, we obtain, for each addon, the most recently modified date—either the explicit "date modified" value returned in HTTP headers when downloading the addon's zip file, or, in the case of `github.com` hosted addons, the date of the most recent commit to the git repository from which the addon was obtained. In this fashion, we were able to obtain modification dates for 6,261 out of the total 8,485 addons we discovered.

Figure 5 shows the percentage of failed addons as a function of their *staleness* (gray barplot), as well as the cumulative percentage of failed addons over time (red curve). If we focus on addons 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 addons are more prune to fail, likely due to incompatibility issues rather than limitations of de-Kodi. The figure also shows (red curve) that these addons constitute about 50% of total "failed-to-install" addons, *i.e.,* about 1,900 addons which are more than two years old.

To further understand the root causes beyond installation failures, we compare each failed addon's dependencies (obtained via the addon's `addon.xml` file) with the set of addons accessible to the addon's Docker container at the time of installation. We identified 949 addons whose failed installations are attributed directly to missing required dependencies. Next, we leverage the Tesseract OCR engine [13] 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 addons with additional dependency related issues; specifically, Kodi entered a state asking the user for permission to download additional addons in order to install the addon or feature of interest. Although not shown due to space limitations, 70% of these combined 1,052 addons date to no more than two years back. This further strengthens our above incompatibility claim: very stale addons are so disconnected with current Kodi APIs that they even fail using the platform to correctly report errors.

Across these 1,052 addons, 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 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 addon lacks an important dependency.

6.2.3 Media Finding. Once an addon is installed, de-Kodi attempts to find playable media through the addon. An addon typed as "xbmc.python.pluginsource" (according to its `addon.xml` file) can contain music, videos, pictures, or some executable application. Throughout this paper, we refer to such addons as *media addons*. When de-Kodi encounters a media addon, it attempts to browse that addon 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 through the addon. If an addon 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 addon's `addon.xml` data often provides this information. Of the 5,435 media addons discovered by de-Kodi, 4,123 claimed to provide video, 356 claimed to provide music, and 1,046 addons 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 addons require subscriptions to third party downloading and streaming services such as Real-Debrid [11]. 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 addons point to content not available in the region where this experiment was performed. In the context of this paper, all media content identified by de-Kodi was *freely accessible* to de-Kodi, presenting no required sign-in, payment, or otherwise complex barrier.

7 KODI ECOSYSTEM ANALYSIS

This section analyzes the Kodi ecosystem using the data discussed in Section 6. As a reminder, we mostly focus on video addons, *i.e.,* addons which provide an interface to access to remote video content, since it is, by large, today's most popular activity on Kodi.

7.1 Addons

7.1.1 Classification. The previous section has offered a classification of addons as "kodi.tv" or "XBMC banned", with respect to the indication from the Kodi team. This classification only applies

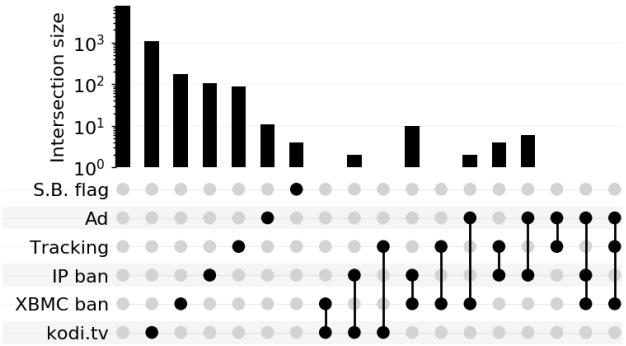


Figure 6: UpSet plot of number of add-ons (y axis) with each tag or combination of tags. Each bar's corresponding tags are marked with a black dot. The bar with no dots corresponds to untagged add-ons.

to 14% of the Kodi ecosystem revealed by de-Kodi and offers no indication on "how" it was obtained. We here "tag" addons based on "suspicious" behaviors (ads injection, tracking, and potential relationship to malware distribution) which are very much rumored in the Web community.

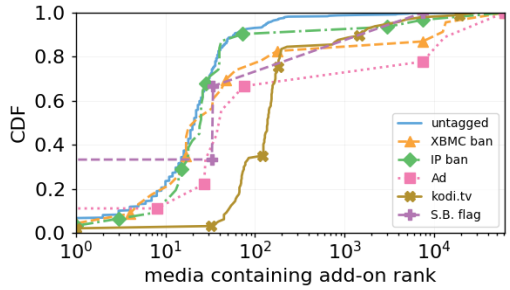
Ads & Tracking – In order to identify tracking and ad traffic, we match our traffic against EasyList and EasyPrivacy (both maintained by [3]), state of the art lists of advertisement and tracking URLs, used by most popular adbockers. We identify 5,247 add-ons that trigger EasyList (advertisement) and 141 add-ons that trigger EasyPrivacy (tracking).

Malware – We investigate potentially malware distributing addons by matching the URLs they contacted against Google Safe Browsing hash, which matches URLs against current known threats and malware [7]. 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 6 as "S.B. flag". To increase coverage, we also compare each observed IP against FireHOL, an automatically updated aggregator of several actively maintained IP banlists [5], labeled in Figure 6 as "flagged IPs". 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 IPs.

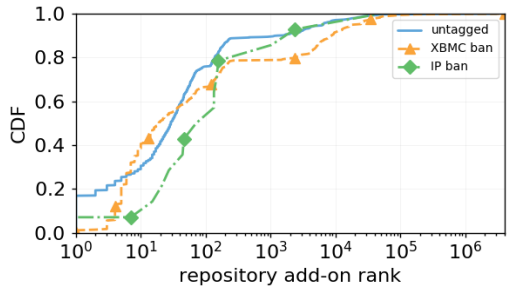
Tag Overlap – It is possible for an individual add-on to meet the criteria for multiple tags. Figure 6, formatted as an UpSet plot, shows the extent of overlap between the add-on sets of each aforementioned tag. All shown tag combinations yielded at least one add-on. From the figure, the stark difference 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 6 — overlap only with two flagged IPs. Conversely, the banned add-on set overlaps with *all four* tags associated with suspicious behavior: tracking, Safe Browsing threats, advertisements, and flagged IPs. This supports Kodi Team's claim that their endorsed add-ons behave in a generally legitimate fashion.

Add-on	Rank	kodi.tv	ban	ads	track	flagged	S.B.
Exodus	60.7K		✓	✓		✓	
Youtube	28,200	✓					
PlaylistLoader	19,200						
Radio_de	19,100	✓					
Phstreams	13,500		✓				
F4mTester	13,100			✓			
ZemTV-shani	12,500		✓				
SportsDevil	7,450		✓	✓	✓	✓	✓
Hdtrailers_net	7,130	✓					
Polishtv.live	5,410						

Table 2: Top 10 content containing add-ons.



(a) Media containing add-ons.



(b) Repository add-ons.

Figure 7: CDFs of popularity rank.

7.1.2 Popularity. The previous analysis suggests that the Kodi ecosystem is mostly composed by "safe" addons, *i.e.,* addons not showing any evident suspicious behavior like tracking or contacting some flagged IPs. However, this does not imply that Kodi users mostly install and use such safe addons. We are thus interested in investigating addons popularity, both as a general research question and to estimate the level of exposure to potentially unsafe addons.

For this measurement, we use Microsoft Azure, which provides a web search API powered by Bing [9]. Bing was selected since no other major search engine provides the same functionality. We estimate addon popularity by counting the number of web search results appearing when searching for an exact match of the addon ID. To reduce potential for false positives, we also require the appearance of either "xbmc" or "kodi" on all web pages contributing to the addon'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 global knowledge of all Kodi users. In our future work, we plan to release a Kodi addon which will help users detect or avoid potentially unsafe addons while opting-in to anonymously report the list of addons they have installed. This approach will help us further corroborate on Kodi add-on popularity.

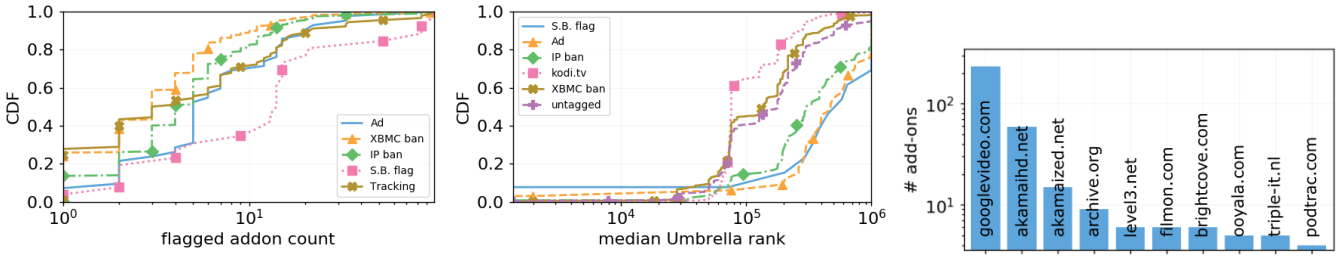
While Kodi offers a range of addon types, we opt to focus our assessment of popularity on *media* addons (video, music, or images) and *repository* addons (collection of addons). Our reasoning for this is threefold. First, most non-media addons only exist to provide support to media addons, e.g., content metadata scraping and cosmetic changes to Kodi's GUI. Second, through manual exploration, we observed that repository addons are often touted an ideal starting place for Kodi users, as they can make the installation of all *other* addons convenient. Lastly, building upon the second point, one's choice of repository addons 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 addons (e.g., addons 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 addons unlikely to be directly searched for by real Kodi users.

Figure 7a shows the Cumulative Distribution Function (CDF) of add-ons *popularity rank* (Bing search ranking) as a function of their classification tag. Overall, the figure shows very skewed distributions with the majority (70-90%) of addons having low popularity ranks ($\sim$ 200), and the remaining addons 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* addons are one to two orders of magnitude more popular than other addons types. A similar trend appears also in Figure 7b, 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 addons. This analysis indicates that a "typical" Kodi user (*i.e.,* relying on a search engine to customize Kodi), has a higher probability to stumble upon a particular repository than a given addon.

The ranks and tags of the top 10 most popular media add-ons—add-ons where audio or video streaming URLs were found—are shown in Table 2. The *highest* ranking media add-on is Exodus, which made headlines in 2017 for being used as a botnet. While it is possible that news reports concerning the Exodus add-on botnet scandal have amplified its popularity rank, that does not necessarily account for the fact that its rank exceeds the next highest ranking media add-on (YouTube) by more than two-fold.

Beyond Exodus, we also see that four of the top 10 media containing add-ons were banned from Kodi's official forum. Two add-ons (PlaylistLoader and f4mTester) although containing content of their own, serve primarily as dependencies for other add-ons, meaning add-ons with suspicious tags make up half of the top video add-ons. Further, three of the remaining six add-ons are served by Kodi's kodi.tv repository, meaning four of the five most popular media containing add-ons not actively endorsed by Kodi are banned by the Kodi Team and, as their tags imply, likely engaging in illicit activity.

Finally, we pause to consider the implications of observed addon popularity with respect to de-Kodi. The long-tailed distribution of addon popularity suggests that, in general, most Kodi users may be turning to the same, small handful of particularly popular addons (the top 1–10% in terms of popularity), which we will loosely refer to as



(a) CDF of addon types per repository. **(b) Resource obscurity.** **(c) Top domains.**

Figure 8

"tier 1" addons. The majority of addons—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 addon potentially missed in de-Kodi's coverage decreases rapidly; the harder the addon is for de-Kodi to find, the less likely, we postulate, it is for a given Kodi user to stumble upon the same addon.

7.1.3 Shared Distribution. Are some addons "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 to quantify the intuition that addons with similar purposes—for example, piracy, malware, etc—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 addons to download or install. In addition, better understanding of how addons 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 addon tags—S.B flags, advertisement, IP ban, and XBMC ban—as *undesirable* flags. For each line in Figure 8a, we plot the number of undesirably flagged addons distributed through a repository where at least one addon of the line's indicated tag was observed. In other words, if we see a repository has an addon tagged for XBMC, we want to know how many undesirably flagged addons in general are provided by that repository. The figure shows that *any* one undesirably flagged addon in a repository has a very low probability of being the *only* addon. In more than 85% of cases, undesirable addon container repositories and sources contained *at least* two such addons. Most notably, the presence of a single SafeBrowsing flag is a strong indicator of 10 or more undesirably flagged addons cohabiting the same repository.

7.2 Content Providers

We here investigate the *providers* behind Kodi content, *i.e.,* the domains where Kodi content (addons, 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 [2]—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 8b plots the *median* Umbrella rank per addon. A clear pattern emerges, dividing our add-on tags into two behavioral groups. We see that, in general, add-ons tagged for flagged IPs, Safe Browsing threats, and advertisements all tend towards 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 8b 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 serving second level domains (ranked by the number of media containing add-ons using at least one media URL from each domain) in Figure 8c. 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.

8 CONCLUSION

This paper introduces the first formal approach to dissecting the behavior of Kodi, today's most popular open source entertainment center. 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 addons 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 for future work.

Using de-Kodi, we discover, install, and test 9,146 unique add-ons within a matter of days, yielding about 6 thousand URLs pointing to video content. We characterize such addons with respect to their potentially suspicious activities, namely tracking, ads and malware injection. We found that most addons are "safe", but the most popular ones tend to engage in suspicious activities.

REFERENCES

[1] 2018. Docker. https://www.docker.com/.
[2] 2019. Cisco Umbrella Top 1 million. https://umbrella.cisco.com/blog/2016/12/14/cisco-umbrella-1-million/.
[3] 2019. EasyList. https://easylist.to/.
[4] 2019. ffprobe Documentation. https://ffmpeg.org/ffprobe.html.
[5] 2019. FireHOL IP Lists. http://iplists.firehol.org/.
[6] 2019. GitHub. https://github.com/.
[7] 2019. Google Safe Browsing. https://safebrowsing.google.com/.
[8] 2019. LazyKodi. http://lazykodi.com/.
[9] 2019. Microsoft Azure. https://azure.microsoft.com/en-us/.
[10] 2019. mitmproxy. https://mitmproxy.org/.
[11] 2019. Real-Debrid. https://real-debrid.com/.
[12] 2019. Reddit. https://www.reddit.com/.
[13] 2019. Tesseract Open Source OCR Engine. https://github.com/tesseract-ocr/tesseract.
[14] 2019. Tstat - TCP STatistic and Analysis Tool. http://tstat.polito.it/.
[15] 2019. XVFB. https://www.x.org/releases/X11R7.6/doc/man/man1/Xvfb.1.xhtml.
[16] Andrew Clay. 2011. Blocking, tracking, and monetizing: YouTube copyright control and the downfall parody. Institute of Network Cultures: Amsterdam.
[17] 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.
[18] Lucas Hilderbrand. 2007. YouTube: Where cultural memory and copyright converge. *FILM QUART* 61, 1 (2007), 48–57.
[19] Luke Hsiao and Hudson Ayers. 2019. The Price of Free Illegal Live Streaming Services. *CoRR* abs/1901.00579 (2019). arXiv:1901.00579 http://arxiv.org/abs/1901.00579
[20] Damilola Ibosiola, Benjamin Steer, Alvaro Garcia-Recuero, Gianluca Stringhini, Steve Uhlig, and Gareth Tyson. 2018. Movie Pirates of the Caribbean: Exploring Illegal Streaming Cyberlockers. In *Proc. INTERNATIONAL AAAI CONFERENCE ON WEB AND SOCIAL MEDIA*.
[21] 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 (RAID 2013)*. Springer-Verlag New York, Inc., New York, NY, USA, 369–389. https://doi.org/10.1007/978-3-642-41284-4_19
[22] Aniket Mahanti, Niklas Carlsson, Martin Arlitt, and Carey Williamson. 2012. Characterizing cyberlocker traffic flows. In *37th Annual IEEE Conference on Local Computer Networks*. IEEE, 410–418.
[23] 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.
[24] Sandvine. 2018. Global Internet Phenomena Spotlight - Kodi. https://www.sandvine.com/hubfs/downloads/archive/2017-global-internet-phenomena-spotlight-kodi.pdf.
[25] XBMC. 2019. Official:Forum rules/Banned add-ons. https://kodi.wiki/view/Official:Forum_rules/Banned_add-ons.