Information and Software Technology 117 (2020) 106197

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

The missing link – A semantic web based approach for integrating screencasts with security advisories

Ellis E. Eghan[a,*], Parisa Moslehi[a], Juergen Rilling[a], Bram Adams[b]

[a] Concordia University, Montreal, Canada
[b] Polytechnique Montreal, Montreal, Canada

ARTICLE INFO

Keywords:
Crowd-based documentation
Mining video content
Software security vulnerabilities
Software dependencies
Software traceability
Semantic knowledge modeling
Semantic web

ABSTRACT

Context: Collaborative tools and repositories have been introduced to facilitate open source software development, allowing projects, developers, and users to share their knowledge and expertise through formal and informal channels such as repositories, Q&A websites, blogs and screencasts. While significant progress has been made in mining and cross-linking traditional software repositories, limited work exists in making multimedia content in the form of screencasts or audio recordings an integrated part of software engineering processes.

Objective: The objective of this research is to provide a standardized ontological representation that allows for a seamless knowledge integration of screencasts with other software artifacts across knowledge resource boundaries.

Method: In this paper, we propose a modeling approach that takes advantage of the Semantic Web and its inference services to capture and establish traceability links between knowledge extracted from different resources such as vulnerability information in NVD, project dependency information from Maven Central, and YouTube screencasts.

Results: We performed a case study on 48 videos that illustrate attacks on vulnerable systems and show that our approach can successfully link relevant vulnerabilities and screencasts with an average precision of 98% and an average recall of 54% when vulnerability identifiers (CVE ID) are explicitly mentioned in the metadata (title and description) of videos. When no CVE ID is present, our initial results show that for a reduced search space (for one vulnerability), using only the textual content of the image frames, our approach is still able to link video-vulnerability pairs and rank the correct result within the top two positions of the result set.

Conclusion: Our approach not only establishes bi-directional, direct, and indirect traceability links from screencasts to these other software artifacts; these links can also be used to guide practitioners in comprehending the potential security impact of vulnerable components in their projects.

1. Introduction

Sharing knowledge and information through the Internet has changed the software industry, with open source development becoming a significant part of this industry. In open source software, the development process extends beyond organizational and project boundaries, with software artifacts (e.g., source code and documentation) being crowd-sourced and shared through public portals (e.g., GitHub,[1] SourceForge,[2] and Maven Central[3]). Collaborative tools and software repositories (e.g., version control systems, mailing lists, and bug tracking systems) support not only the crowd-sourced development but also its agile development processes that focus on informal, minimal documentation of a system and its functionalities. As part of this collaborative environment, developers and users often share their product knowledge and expertise through different types of media (i.e., software repositories, Q&A websites, blogs, and multimedia documentation) [1].

One media type which has gained popularity in recent years are screencasts. They are typically created by the crowd and used to document different aspects of a system, such as: explaining how specific system features work, document an observed bug in a system, provide a workaround for a known problem, or demonstrate security issues caused by a known vulnerability. Screencasts deliver their content in the form of

* Corresponding author.
E-mail addresses: e_eghan@encs.concordia.ca (E.E. Eghan), p_mosleh@encs.concordia.ca (P. Moslehi), juergen.rilling@concordia.ca (J. Rilling), bram.adams@polymtl.ca (B. Adams).

[1] https://github.com/.
[2] https://sourceforge.net/.
[3] https://search.maven.org/.

https://doi.org/10.1016/j.infsof.2019.106197
Received 5 March 2019; Received in revised form 26 September 2019; Accepted 30 September 2019
Available online 3 October 2019

audio (through a narrator), video (image frames), and textual metadata (e.g., subtitles, title, description, publish date). For example, screencasts uploaded on crowd-based online video portals or repositories (e.g., YouTube[4] or Vimeo[5]) are used to illustrate how an attacker can exploit a known vulnerability in a system, or how a vulnerable API might introduce security exploits in a client application. Screencasts not only benefit from their ability to deliver dynamic content through images, text, and speech but also that they are mostly created by the crowd who make new content available as a product evolves, or certain features gain popularity among the user base.

At the same time, Information Security (IS) has emerged as an essential part of software engineering best practices [2]. Specialized advisories or **S**ecurity **V**ulnerability **D**ata**B**ases (SVDBs), such as the National Vulnerability Database (NVD),[6] have been introduced in response to the increasing number of known software vulnerabilities. Vulnerabilities are no longer limited to individual projects or computers, but often affect millions of computers and even complete software ecosystems. SVDBs serve in this context as central repositories for tracking software vulnerabilities and potential solutions to resolve them. However, vulnerabilities and their exploits are only described in these repositories in a textual format, lacking hands-on instructions on how to replicate or fix a known vulnerability.

While these global repositories (e.g., video portals, security vulnerability databases) have been widely adopted by industry to support collaborative software development and knowledge sharing, they also introduce new challenges. These repositories often remain information silos – a situation where a repository is typically not directly linked with another one [3]. For example, source code stored in versioning repositories may contain vulnerabilities already reported in SVDBs. However, without having a link between the source code and vulnerability databases, developers must manually search individual repositories for relevant information or artifacts. A major challenge when establishing traceability links among these repositories is the lack of a common, standardized semantics and knowledge representation that is applicable across repository boundaries.

The objective of our research is therefore twofold: 1.) We discuss how a standardized knowledge representation with well-defined semantics can address the integration and linking of these knowledge resources, as well as 2.) allow for the introduction of novel types of software analytics that take advantage of such a unified knowledge base.

In our prior work [4], we introduced our **SE**curity **V**ulnerability **ONT**ology (SEVONT) and **S**oftware **B**uild **S**ystem **On**tology (SBSON) ontologies to model the domain of SVDBs and project dependencies, respectively. We also introduced a Security Vulnerabilities Analysis Framework (SV-AF), which is a semantic modeling approach that establishes traceability links between the NVD security database (modeled by SEVONT), Maven dependency repository (modeled by SBSON), and the source code of projects (modeled by SEON [5]).

The research in this paper is a continuation of our previous work on semantic modeling and tracing of software security vulnerabilities. We extend our existing knowledge base with a video ontology by integrating audio, video (textual cues in image frames) and metadata from screencasts published on YouTube with software dependency and security-related knowledge from our existing SV-AF approach. We also establish bi-directional traceability links from screencasts to NVD security vulnerabilities and infer indirect traceability links between screencasts and Maven project dependencies by taking advantage of our existing traceability links (in SV-AF) between NVD and Maven Central. We argue that these links allow us to enrich existing vulnerability information that can be further used to provide practitioners with a different type of vulnerability analysis services. We also discuss several usage scenarios, where developers watching a screencast could be notified that an API shown in a screencast contains known vulnerabilities or a screencast can be linked to SVDBs to provide additional instructions on how to replicate or fix a vulnerability.

It should be noted that the results which we are presenting are currently not generalizable for all types of videos and vulnerabilities. Given the diversity of screencasts in terms of their content, length, languages/dialects being used and, image quality of the videos, our reported results are not generalizable. Instead, this research presents results from a case study which we conducted as a proof of concept on a set of screencasts related to software vulnerabilities. We illustrate that it is indeed possible to link screencasts, which mention a vulnerability identifier in at least one of its information resources (meta data, image frames, speech) to a vulnerability in a SVDB. We also show that once such direct references to vulnerability identifiers are removed, while more difficult and with lower precision, it is still possible to link vulnerabilities with screencasts.

The main contributions of this work are as follows:

- We introduce our **VID**eo **ONT**ology (VIDONT) to capture the semantics of crowd-based online video repositories (e.g., YouTube).
- We establish bi-directional traceability links between knowledge within our SEVONT and VIDONT ontologies; indirect traceability links are also inferred between VIDONT and SBSON.
- We evaluate the accuracy of these direct traceability links between screencasts and vulnerability information in NVD.
- We perform a case study to illustrate the applicability and flexibility of our modeling approach.

The remainder of this paper is organized as follows: Section 2 motivates our work. Section 3 summarizes background relevant to our research, followed by Section 4, which introduces the methodology we used to create our integrated knowledge model. Section 5 discusses our case study design and findings. Section 6 provides a discussion of potential threats to the validity of our approach. Section 7 compares our work with related work, followed by Section 8, which concludes the paper and discusses future work.

2. Problem statement

2.1. Motivating examples

The following two scenarios motivate our research on establishing a unified representation for integrating screencasts with security advisory databases by establishing semantic traceability links between them. In these scenarios, we take advantage of repositories such as NVD, Maven Central, and YouTube and illustrate the potential benefits of these links. NVD and Maven Central are widely adopted by the software engineering community to capture software vulnerability and project dependency metadata. YouTube, on the other hand, is one of the most popular online video sharing platforms to which 400 h of video, related to a vast variety of subjects, are uploaded per minute [6]. Studies suggest that YouTube provides a crowd-based online platform that allows people who are experts in a domain to share their knowledge with novices [7]. Also, developers create screencasts as an alternative to blogging, since 1) they would prefer learning by videos and 2) they find it easier to express and share their tacit knowledge through screencasts [7].

Scenario #1: A fictional developer, called Bob, performs a YouTube search[7] for a video tutorial on how to use the Commons FileUpload[8] library in his project. From the search results, he selects the highest

[4] https://www.youtube.com/.
[5] https://vimeo.com/.
[6] https://nvd.nist.gov/.

[7] The query "commons-fileupload" using the YouTube search engine was performed in June 2018. Note that given the same key words, YouTube may return different numbers of videos over time due to the addition of new and more relevant videos.
[8] https://commons.apache.org/proper/commons-fileupload/.

Fig. 1. Overview of Scenario #1.

Fig. 2. Overview of Scenario #2.

ranked video,[9] which has close to 60,000 views and 255 likes. Bob finds the video useful and reuses the same API in his project by replicating the steps shown in the screencast. However, Bob (and most likely even the screencast producer) is unaware that the version of the library used in the video contains three known security vulnerabilities (CVE-2014-0050, CVE-2016-3092, and CVE-2016-1,000,031)[10] that are already published in the NVD repository (see Fig. 1). As a result, any developer who includes this library version in their projects based on the recommendation of YouTube tutorials (on how to perform a programming task) are unknowingly risking the security of their project. To identify possible impacts of security vulnerabilities, developers must manually identify and compare the projects mentioned in a video with vulnerable projects found in NVD entries.

Further analysis of the Commons FileUpload library shows that the library is currently directly used by more than 1800 other OSS libraries[11] within the Maven Central repository, further illustrating that not only Bob but also other developers and API users are often unaware of known vulnerabilities.

Scenario #2: Bob is a developer who uses Apache Struts[12] in his project and wants to know how exactly an Apache Struts vulnerability can exploit his system, or how to patch this vulnerability. He uses NVD to check if any vulnerabilities for Apache Struts exist. However, while the vulnerability information exists in NVD,[13] it does not provide enough detailed step-by-step instructions for Bob on how the vulnerability can be exploited (or patched). As a result, he uses different keywords from the NVD vulnerability description to manually search the Internet for such instructions. He finds several screencasts on YouTube,[14] that might demonstrate Apache Struts vulnerabilities. Traditional video searches rely on indexing of available metadata provided by the video creator (e.g., title, comments). The challenge for Bob is now to identify the video(s) that are most relevant to this vulnerability (see Fig. 2).

As illustrated by these two scenarios, developers often resort to a variety of knowledge resources, including informal resources such as screencasts and video tutorials when trying to comprehend and analyze software systems. These resources allow developers to share details such as implementation approaches, practical overview of concepts and theories, and personal development experiences [8]. Though many screencasts and video tutorials exist, they are rarely linked with other software-related knowledge resources, therefore making it difficult to locate the appropriate resources.

Moreover, an adversary can exploit a system vulnerability in different ways, which is also referred to as a system's attack surface [9]. For example, when a vulnerable API can be used in different projects, its attack surface can differ significantly depending on a project context, and different scenarios might exist on how the vulnerability can be exploited. An advantage of screencasts compared to traditional documentation is that screencasts are typically generated independently from each other by the crowd, increasing, therefore, the chances that different screencasts will cover potentially different attack/exploit scenarios for a given vulnerability. Providing such a broader coverage of the attack surface can provide additional insights on vulnerabilities, their impacts and illustrate how to patch/fix these when they occur in different contexts (e.g., environments, usage scenarios).

2.2. Research objective

The objective of this research is to introduce a standardized representation and shared semantics of software and other related artifacts found in heterogeneous knowledge resources. More specifically, we introduce VIDONT, an ontology for capturing the semantics of crowd-based online video repositories and integrate it with our existing SEVONT and SBSON ontologies [4]. This knowledge integration will not only provide developers with direct access to vulnerability information described in a screencast content, but also allow tracing of vulnerability descriptions to relevant screencasts and library dependency information. Furthermore, our approach also allows developers to identify screencasts that demonstrate such attacks and provides developers whose projects might directly or indirectly (e.g., through Maven dependencies) be exposed to vulnerable libraries with insights and direct access to information on how to mitigate such potential vulnerabilities.

[9] https://www.youtube.com/watch?v=4yb16lTxbM8&t=266s.
[10] https://nvd.nist.gov/vuln/search/results?adv_search=true&cpe=cpe%3a%2fa%3aapache%3acommons_fileupload%3a1.3.
[11] https://mvnrepository.com/artifact/commons-fileupload/commons-fileupload/1.3.
[12] https://struts.apache.org/.
[13] https://nvd.nist.gov/vuln/detail/CVE-2017-9805.
[14] https://www.youtube.com/watch?v=Aaglpe4A27A.

Therefore, in this paper, we aim to answer the following research question:

- **RQ**: How accurate are the bi-directional links, between known vulnerabilities and screencasts published on YouTube, provided by our knowledge model?

3. Background

3.1. Crowd-based multimedia documentation

Software users and developers regularly search the Internet for information and documentation that can help them in completing specific tasks. A common characteristic of such online documentation found on the Internet is that it is often created and maintained by many and viewed by many; hence, these are typically called "crowd-based documents" [10]. The motivation behind creating crowd-based documents are manifold, including 1.) for the document creators to gain an online reputation, 2.) to better understand a new subject by creating examples that improve one's expertise by teaching others [7]. Crowd-based documents can be categorized in two formats: textual and multimedia documents. Some examples of crowd-based text documentations are blog posts, emails, Q&A sites, and wikis, whereas screencasts and podcasts are considered multimedia documentation.

Multimedia documents, and more specifically screencasts, differ from traditional textual and formal documents, not only in the format how information is presented to the user, but also how their content is organized [7]. In what follows, we discuss major screencast characteristics that make it difficult to integrate screencasts with other types of software artifacts.

Abstraction level: Screencasts are composed of audio and image frame components. Sometimes, the audio component contains speech created by a narrator who explains how a certain task can be done, while demonstrating it visually through the image frames that capture the screen of the demonstrated application. Therefore, the content of a screencast would be unstructured in comparison to other software artifacts that contain structured or semi-structured textual content. Furthermore, some screencasts demonstrate a certain task or scenario (e.g., an attack using a vulnerability) mostly containing high-level information or descriptions, omitting technical details. Therefore, the content of such screencasts may differ from the one found in NVD in terms of the vocabulary used for describing the type, severity, affected products, and references to advisories or solutions. This could result in potential information ambiguity and inconsistency among the knowledge resources.

Dynamic vs. Static: Screencasts have "time" as a meta component that allows for dynamically showing image frames on the screen with a constant frame rate. In contrast, the content of other textual documents is static (e.g., descriptions of vulnerabilities in NVD).

Information resources: Security vulnerability screencasts are created by security researchers and white hat hackers from around the world. Various information resources exist in the screencasts' content and metadata [7]. The content contains the domain expert's knowledge that is captured by image frames, speech, the sequence of GUI events and user actions (clicking buttons, opening menus, etc.), while the metadata contains captions, publish date, creator, comments, etc. However, the availability and quality of the information resources that come from the screencasts' content can vary significantly among screencasts [1].

3.2. Security vulnerabilities

In the software security domain, a software vulnerability refers to mistakes or facts related to security problems in software, networks, computers, or servers. Such vulnerabilities represent security risks that can be exploited by hackers to gain access to system information or capabilities [11]. Among these systems, reuse of software libraries poses a significant threat, since vulnerabilities in a single component might affect, through their intended reuse, many different systems across the globe.

Advisory databases (e.g., NVD) were introduced to provide a central place for standardizing the reporting of vulnerabilities and to raise developer awareness about the existence of such vulnerabilities. These advisory databases rely on the Common Vulnerabilities and Exposures (CVE),[15] a publicly available dictionary for vulnerabilities that allows for more consistent and concise use of security terminology in the software domain. Once a new vulnerability is revealed and verified by security experts, information about this vulnerability (e.g., unique identifier, source URL, vendor URL, affected resources, and related vulnerabilities information) is added to the CVE database. In addition to the CVE entry, each vulnerability will also be classified using the Common Weakness Enumeration (CWE)[16] database. CWE provides a common language to describe and classify software security vulnerabilities based on their type of weakness. NVD, CVE, and CWE can be considered as being part of a global effort to manage the reporting and classification of known software vulnerabilities.

3.3. Dependency management - Maven

Maven, hosted by the Apache Software Foundation, is an open-source build automation tool used primarily for Java projects. In Maven, a software project defines its dependence on any of its artifacts as part of its XML configuration file (also called the POM file), which is stored in the central repository. Upon the build of a project, Maven dynamically downloads the requested versions of all required Java libraries and Maven plug-ins from the Maven Central repository into a local cache for use by the project. The Maven Central repository provides open source organizations with an easy, free, and secure way to publish their components for access by millions of developers. The repository is updated with new projects and new versions of existing projects that can depend (in)directly on different versions of the same dependency.

Transitive dependencies: One of the core dependency management features provided by Maven are transitive dependencies. If project-A depends on project-B, which in turn depends on project-C, then project-C is considered a transitive dependent of project-A. Part of Maven's appeal is that it can manage these transitive dependencies and shield developers from having to keep track of all build dependencies required to compile and run an application [12]. As a result, one can now just include a Java library (e.g., Spring Framework) without having to specify the dependencies of that library oneself.

3.4. Ontologies and semantic web

The Semantic Web has been defined by Berners-Lee et al. as "an extension of the Web, in which information is given well-defined meaning, better enabling computers and people to work in cooperation" [13]. It forms a Web from documents to data, where data should be accessed using the general Web architecture (e.g., URIs). Using this Semantic Web infrastructure allows data to be linked, just as documents (or portions of documents) are already, allowing data to be shared and reused across application, enterprise, and community boundaries. In a Semantic Web, data can be processed by computers as well as by humans, including inferring new relationships among pieces of data. HTML, for instance, can present information in terms on how information is displayed by machines to the user, but it lacks the necessary semantics to allow for further machine interpretation of the displayed facts in terms of their meanings. The Semantic Web overcomes this limitation by adding semantics to the information, making information machine processable and linkable. For example, the YouTube video "File Upload in Java Servlet" used in our motivating example (Scenario #1) has been created

[15] https://cve.mitre.org/.
[16] https://cwe.mitre.org/.

by the YouTube contributor "Telusko". Analyzing the HTML source of the web page would allow us to identify that a text string "Telusko" exists, but in contrast to the Semantic Web, HTML does not allow us to reason or associate that "Telusko" corresponds to the author of the video.

For machines to understand and reason about knowledge, this knowledge needs to be represented in a well-defined, machine-readable language. Ontologies provide a formal and explicit way to specify concepts and relationships in a domain of discourse. The Semantic Web uses the Resource Description Framework (RDF) as its underlying data model to formalize the meta-data of real-world resources as subject-predicate-object triples, which are stored in triple-stores. A resource in Semantic Web can be anything: a person, project, software, a security bug, etc. Triple-stores are Database Management Systems (DBMS) for data modeled using RDF. Unlike Relational Database Management Systems (RDBMS), which store data in relations (or tables) and are queried using SQL, triple-stores store RDF triples and are queried using SPARQL [13]. The RDF data-model is domain independent, and users define ontologies using an ontology definition language.

The Web Ontology Language (OWL) [14] is an example of such a definition language and has been standardized by the W3C.[17] It supports the creation of machine-understandable information to enable Web resources to be automatically processed and integrated. The OWL-DL sublanguage, is based on Description Logics (DLs) [15]. DL is a logic-based formalism using predicate calculus to define facts that can formally describe a domain. Therefore, DLs are a set of axioms called a TBox (e.g., *Doctor*$\sqsubseteq$*Person*) and set of facts called ABox (e.g., {Parent(John), hasChild(John, Mary)}). Both TBox and ABox form a Knowledge Base (KB) and are often written $K = \prec T, A \succ$. The RDF data-model forms a graph where nodes (subject, object) are connected through edges (predicates). The SPARQL query language [16] is used to retrieve information from RDF data-model graphs.

Ontologies vs. Models. A model is "an abstraction that represents some view on reality, necessarily omitting details, and for a specific purpose" [17]. In SE, ontologies and models try to address the same problems (representing the software complexity abstractly) but from very different perspectives. The differences between ontologies and models often result in different artifacts, uses, and possibilities. For example, modern SE practices advise developers to look for components that already exist when implementing functionality, since reuse can avoid rework, save money, and improve the overall system quality [18]. In this example, ontologies can provide clear advantages over models in integrating information that normally resides isolated in several separate component descriptions. Furthermore, models (e.g., UML) rely on the closed world assumption, while ontologies (e.g., OWL) support open-world semantics. OWL, an example ontology language, is a "computational logic-based language" that supports full algorithmic decidability in its OWL-DL (description logic) variant. It is not possible to use algorithms supported by OWL (e.g., subsumption) for modeling languages due to their different semantics. Additional differences between ontologies and models are reported and discussed elsewhere [19].

3.5. SV-AF: security vulnerability analysis framework

It is generally accepted that inadvertent programming mistakes can lead to software security vulnerabilities and attacks [11]. Mitigating such vulnerabilities can become a major challenge for developers, since not only their own source code might contain exploitable code, but also the code of third-party APIs or external components used by their system. In our previous work [4], we introduced SV-AF to guide developers in identifying the potential impact of vulnerabilities at both the system and global level.

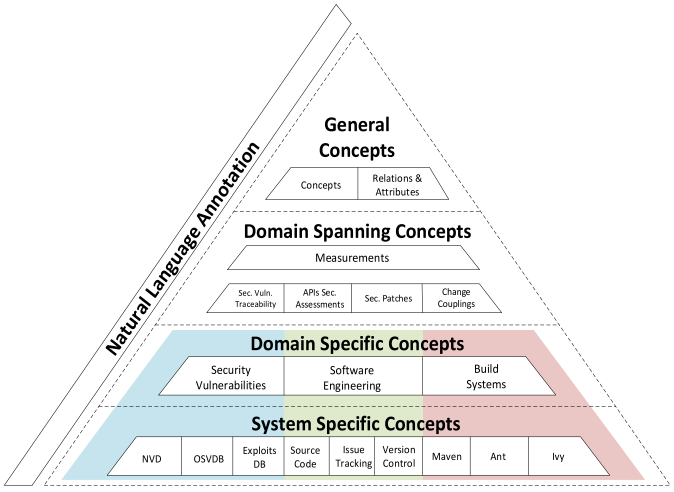


Fig. 3. The SV-AF Ontologies Abstraction Hierarchies.

SV-AF uses a bottom-up modeling approach where system-specific concepts are first extracted, followed by an iterative process of abstracting shared concepts into upper ontologies. To minimize any potential abstraction error, three Ph.D. students from our lab[18] performed a cross-validation of the abstracted ontology layers, reaching an average inter-rater agreement of 95%. The disagreements were resolved through further discussion. The resulting four-layer modeling hierarchy (Fig. 3) is based on a metadata modeling approach introduced by the Object Management Group (OMG),[19] with each layer providing a different level of abstraction in terms of its purpose and design rationale.

General Concepts: Classes in the top layer represent omnipresent general concepts found in the software evolution and security domain.

Domain-Spanning Concepts: This layer captures concepts that span across several subdomains (e.g., security databases, video repositories, and source code).

Domain-Specific Concepts: Concepts in this layer are common across resources in a domain. At the core of the domain-specific layer, we have several domain ontologies: (1) Software sEcurity Vulnerability ONTologies (SEVONT), (2) Software Evolution ONtologies (SEON) [5] and (3) Software Build System (Dependencies) ONtologies (SBSON).

System-Specific Concepts: Concepts in this layer extend the knowledge from the upper layers to specific vulnerability databases, tool implementations, or programming languages. For example, the Maven system-specific ontology will contain concepts found only in the Maven tool.

SV-AF uses the Probabilistic Soft Logic (PSL) framework [20] to establish weighted links between ontological models of vulnerability databases (SEVONT) and software dependency repositories (SBSON). These traceability links are created based on semantically identical or similar concepts within the different knowledge sources. In this case, similarity among SEVONT-SBSON instance pairs are determined based on the extracted literal information such as name, version and vendor. Using manually defined rules, the PSL framework computes similarity weights between all possible instance pairs in the knowledge base (total of |SEVONT| x |SBSON| instance pairs). These computed similarity weights, based on a given similarity threshold, are used to infer owl:sameAs relations between similar instances found in the two ontologies. The owl:sameAs construct is a built-in OWL predicate used to align two concepts from different ontologies. More details on the ontologies, ontology alignment process, and evaluation of the SEVONT-SBSON alignment can be found elsewhere [4].

[17] https://www.w3.org/.

[18] http://aseg.encs.concordia.ca.

[19] http://www.omg.org/.

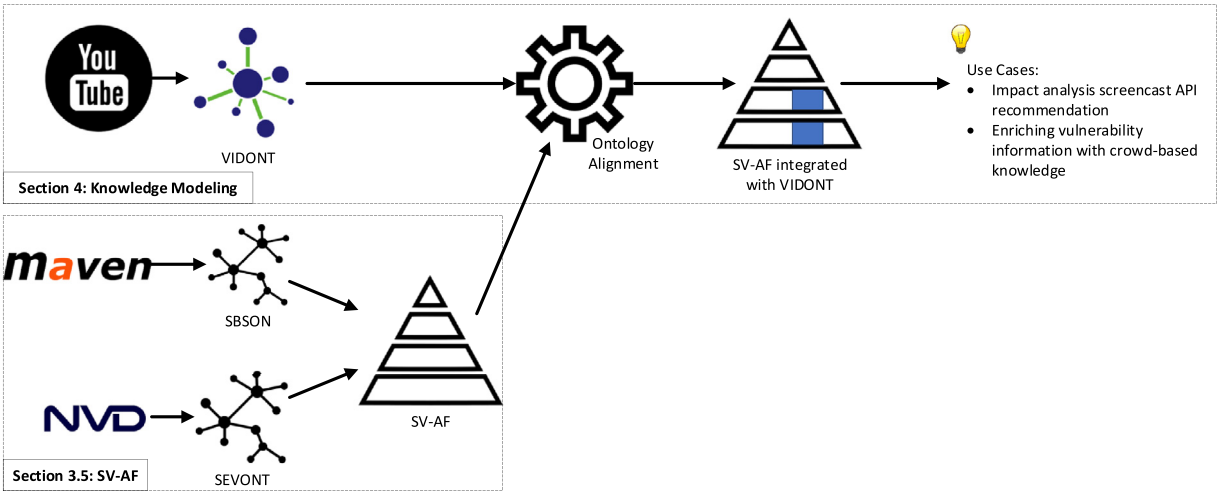


Fig. 4. Overview of the overall methodology. SV-AF is extended with ontologies for the domain of Screencast repositories.

4. Methodology

4.1. Overview

The knowledge modelling approach proposed in this paper establishes traceability links between screencasts (e.g., YouTube) and existing software vulnerability and dependency knowledge found in SV-AF. In what follows, we describe how we extend SV-AF with knowledge from screencasts. Fig. 4 illustrates our overall research methodology and its major steps.

4.2. Fact extraction process

4.2.1. Fact extraction from video artifacts

Video portals such as YouTube contain videos that are created by the crowd to document different aspects of software systems. These videos and screencasts contain different kinds of information, that can be mined and analyzed to support the linking between video content and other software artifacts. In what follows, we discuss in more detail the information being extracted and the linking opportunities provided by our approach.

- **Speech:** If a video has a narrator that describes the content of a how-to video, the speech content may share similar words with the image frames shown in the screencast, as well as the relevant entity in NVD.
  - *Information extraction*: In case closed captioning is enabled for a video, we extract the screencast's subtitles using existing tools (e.g., youtube-dl[20]). If closed captioning is not available, we automatically transcribe the spoken text.
  - *Linking opportunities*: If the transcription of a video mentions the CVE ID of a vulnerability, one can use this CVE ID to link the video to the relevant NVD entry. However, automatic speech to text tools will not always transcribe such information accurately. To improve the accuracy of traceability links, we also take advantage of Information Retrieval (IR) (see Section 4.3.1) to locate similarities between artifacts such as the transcribed speech, NVD entries, and image frames (GUI text).
- **Image frames:** Screencasts capture the interactions with the GUI of a software application or with a terminal window. As a result, the textual information of a typical window screen (i.e., a screencast image frame) will often contain corresponding string literals found in the vulnerability dataset. Fig. 5 illustrates matching string literals between speech, image frame text, and vulnerability description in NVD. Image frames also capture graphical cues such as icons, frames, and colors that can be used to identify different parts of the software application or the environment in which the user is working. Extracting such graphical cues requires image processing techniques that will be explored in future work.
  - *Information extraction*: First, we extract image frames of the screencasts using FFmpeg[21] at a rate of 1 frame per second, to avoid having many repetitive image frames. We then perform Optical Character Recognition (OCR) [21] on each image frame, using Google Vision API[22] to recognize the text shown in the image frame. This text, which is enclosed within neighboring pixels on the images, is returned as a sentence or word together with the coordinates where the words are located on the image frames.
  - *Linking Opportunities*: If an image frame contains a CVE ID, OCR will be able to extract this CVE ID with high accuracy. Such an extracted CVE ID can then again be linked to the NVD repository. In addition, similar to spoken text, we can use the text on the GUI as an information source to link a video with NVD entries. With each image frame, we also have its position (timestamp) within the video, which allows us to directly link external knowledge resources to the relevant part of the video.
- **Metadata:** Videos published on YouTube typically include metadata that can be extracted and analyzed.
  - *Information extraction:* Among the metadata to be extracted are video title, description, published date, comments, number of likes and dislikes, number of views, closed captioning (if enabled) and other information related to the screencast [1].
  - *Linking opportunities:* This metadata often contains important information that can be added to the video documents to support the linking of screencast content to NVD vulnerabilities. For example, the publication date can be used to help identify vulnerabilities discussed in a video.

It should be noted that when preprocessing text extracted from the transcribed speech, image frames, and metadata, we do not remove tokens that combine characters and numbers since CVE IDs are composed of these characters (Fig. 5 contains a sample CVE ID). We, however, apply tokenization, removal of stop words and punctuation, and perform stemming to clean up the data before populating the ontology in 4.3.

[20] https://youtube-dl.org.

[21] http://ffmpeg.org/.

[22] https://cloud.google.com/vision/.

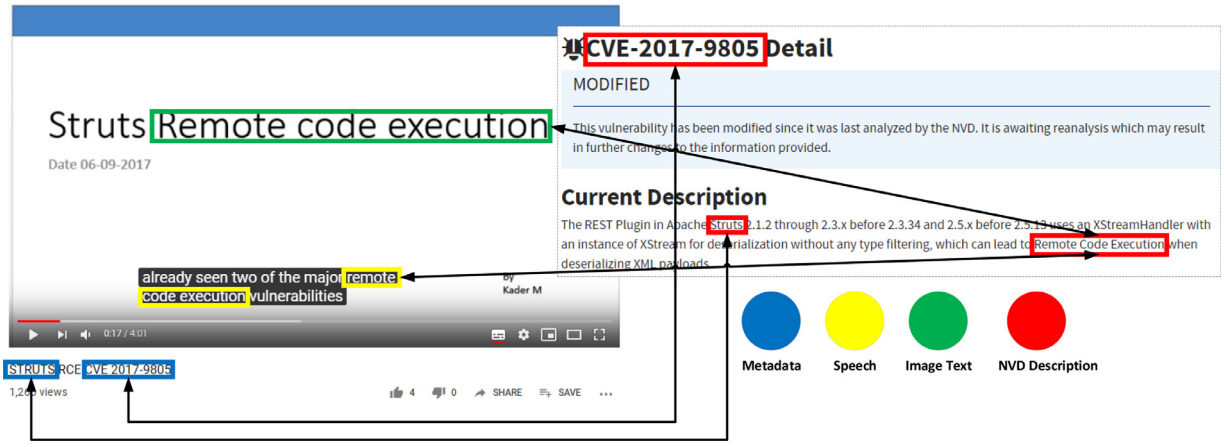


Fig. 5. Similar string literals in speech, image text, and NVD description.

4.2.2. Fact extraction from vulnerability artifacts

Security databases (e.g., NVD) provide a central place for reporting vulnerabilities affecting existing software applications and systems. Extracting facts from NVD consists of downloading and parsing its vulnerability XML feeds.

- Information extraction: In NVD, a vulnerability is identified by its unique CVE ID. NVD also captures additional vulnerability details, such as: vulnerability disclosure date, severity score, vulnerability summary, and sources of information that demonstrate the vulnerability. Each vulnerability has a list of affected products associated, described by its Common Platform Enumeration (CPE)[23] a standard machine-readable format for encoding names of IT products and platforms.
- Linking opportunities: Screencasts may contain some of the information in published vulnerabilities such as the CVE ID and vulnerability summary. The CVE ID may be found in the video's title, description, speech, or image content. Also, the same words that are used in the description of a vulnerability in NVD may be used in the screencast data (i.e., speech, images, metadata). Therefore, we extract this information from the vulnerabilities to be further used in our linking approach.

4.2.3. Fact extraction from Maven artifacts

Maven artifacts describing the dependencies used by a project are stored in the Maven Central Repository. Extracting Maven facts consists of transforming the Maven Central repository index into a list of GAV (groupId, artifactId, and version) coordinates - the three required elements to describe every project. The groupId is a unique name amongst an organization, and the artifactId is generally the name by which the project is known. Several projects developed by the same organization or under the same parent project will have the same groupId. The POM file for each GAV entry is parsed and populated into our knowledge base.

- Information extraction: As stated above, each project release in Maven Central is identified by its unique GAV coordinate. The POM file for each GAV entry captures various project details, such as the project name, dependencies on other project releases, organization, developers, release date, and links to its repositories (e.g., issue-trackers and CVS).
- Linking opportunities: The details captured within each project's POM file provide several linking opportunities to existing knowledge bases. For example, project identification details such as the GAV, project name, or URL can be used to identify project entities in the textual descriptions of videos based on string matching. They can also be used to identify products affected by vulnerabilities within the NVD dataset (see [22]). Furthermore, project dependency information (using the <dependency> tags) can be used to establish links to the text extracted from source code samples in videos.

[23] Common Platform Enumeration – http://cpe.mitre.org.

4.3. Knowledge modeling – extending SV-AF with knowledge from screencast repositories

In this section, we introduce VIDONT, our ontology to capture the semantics of crowd-based online video repositories (e.g., YouTube). As part of our knowledge modeling, we then integrate VIDONT with our existing SV-AF ontologies at the domain level. The integration of VIDONT and SV-AF allows not only for knowledge sharing and reuse across repository boundaries by eliminating traditional information silos these resources have remained, but also allows for novel types of vulnerability analysis and documentation approaches.

Fig. 6 provides an overview of the main classes and object properties across all layers of our knowledge model that are used for linking screencasts to project vulnerabilities. To improve the readability of this paper, we denote OWL classes in *italic* and properties are underlined. The core concepts used in our model are *Vulnerabilities, Videos*, and *APIs*. A project version that is released to the public or customer is referred to as a *BuildRelease* (a *BuildRelease* can dependOn *APIs* from other *BuildReleases*). Different project metadata are captured using the hasName, hasDescription, hasURL, and hasVersionNumber properties. Whenever such a project is identified to be affectedBy a *Vulnerability*, a description of a vulnerability is publicly disclosed in repositories such as NVD. Each publicly disclosed vulnerability is issued a unique ID, captured by the hasVulnerabilityID property.

Details of existing vulnerabilities, their exploits, and how such exploits can be mitigated are describedBy *Videos* provided by *Publishers* using video repositories such as YouTube. A publisher isA type of project *Stakeholder*. In our model, published videos have information encoded as *Speech* and *Image* frames. Videos have metadata associated such as title, description, and published date. Keywords and tags for a video are captured through the label property. Typical for crowd-based videos is that they allow for *Comments* and discussions from other users and viewers. The link between videos and vulnerabilities can be established using a *SimilarityMeasure* that measures the relevance of a given video in terms of covering a known vulnerability. For a complete description of our ontologies, we refer the reader to our earlier work [4,22].

4.3.1. Ontology alignment and knowledge inferencing

To further improve the knowledge integration between the VIDONT, SEVONT and SBSON ontologies, we establish semantic traceability links through ontology alignment. During ontology alignment, identical or

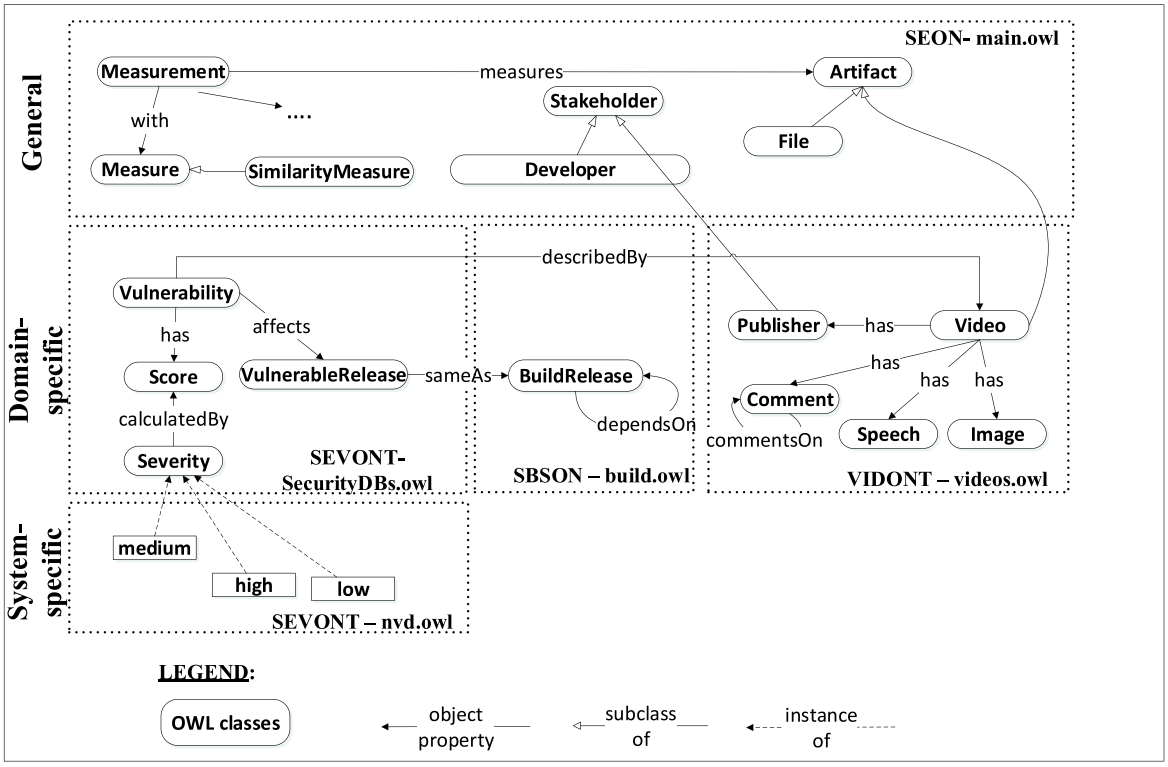


Fig. 6. An overview of concepts and properties in our integrated knowledge model.

Table 1
Ontology Namespaces.

Namespace	URL
RDF	http://www.w3.org/1999/02/22-rdf-syntax-ns#
OWL	http://www.w3.org/2002/07/owl#
SBSON	http://aseg.cs.concordia.ca/segps/ontologies/domain-spanning/2015/02/build.owl#
SEON	http://se-on.org/ontologies/general/2012/02/main.owl#
SEVONT	http://aseg.cs.concordia.ca/segps/ontologies/domain-spanning/2015/02/vulnerabilities.owl#
VIDONT	http://aseg.cs.concordia.ca/segps/ontologies/domain-spanning/2018/01/video.owl#
MEASURE	http://se-on.org/ontologies/general/2012/02/measurement.owl#

equivalent concepts, properties or facts in multiple ontologies are identified by analyzing the data captured as part of ontology descriptions (e.g., labels, comments, attributes and types, relations with other entities) and by using logical reasoning to infer correspondences. By establishing these links, we can reduce the semantic gap between the individual ontologies, which is an essential prerequisite for providing a unified knowledge model.

In what follows, we discuss in more detail how we use Semantic Web inference techniques to establish these additional semantic traceability links between our ontologies. The reasoning services allow us to discover additional relations to other knowledge and facts captured in our knowledge base. It should be noted that we omitted the ontology namespace prefixes (summarized in Table 1) from our illustrative queries and rules shown in this section to improve their readability.

SEVONT and VIDONT Ontology Alignment. Screencasts and other tutorial videos often contain references and keywords related to a vulnerability, such as the CVE ID in the video title, description, speech, or text encoded in image frames. Our alignment process links instances in the two ontologies based on the presence of such references and keywords. These vulnerability references are used as tags during the video data extraction process. For example, the triple <https://youtu.be/Wewl5fAhnXA> *rdfs:label "CVE-2015-0096"* represents a video resource tagged with a CVE ID that was found in its title. This knowledge can now be used to perform terminology matching, by aligning instances from the vulnerability and video ontologies. For the alignment, we use the Semantic Web Rule Language (SWRL)[24] to create the rule in Listing 1, which can now infer links between vulnerability and video instances.

However, the alignment rule in Listing 1 only applies when a vulnerability's CVE ID is explicitly mentioned in a video. To support cases where no CVE ID is explicitly mentioned, we complement our alignment approach with the BM25 probabilistic relevance model [23]. BM25 is a popular model used in Information Retrieval (IR) to rank a set of documents based on their relevance to words in a given query. It is based mainly on the term and document frequency measures. Given a query, Q, containing keywords $q_1,\ldots,q_n$, the BM25 score of a document, D, that measures the similarity between Q and D is calculated as:

$$score(D,Q) = \sum_{i=1}^{n}\left(\frac{N}{df(q_i)} * \frac{tf(q_i,D)*(k_1+1)}{tf(q_i,D)+k_1\left(1-b+b*\frac{|D|}{avgdl}\right)}\right), \quad (1)$$

where $tf(q_i, D)$ is $q_i's$ term frequency in D, $df(q_i)$ is the number of documents containing q_i, |D| is the length of D in words, avgdl is the average length of all documents used in the relevance scoring, N is the total num-

[24] https://www.w3.org/Submission/SWRL/.

```
Video(?video), label(?video,?label), Vulnerability(?vuln), hasVulnerabilityID(?vuln,?label)
→ describedBy(?vuln,?video)
```

Listing 1. SWRL rules for aligning CVE facts with the video ontology.

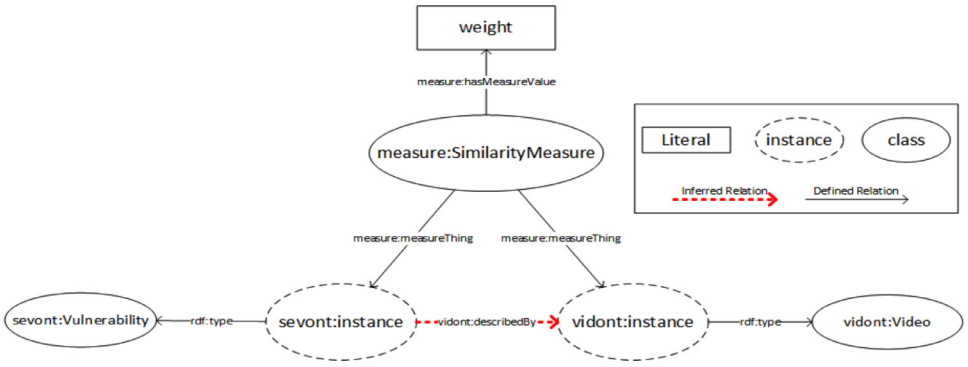


Fig. 7. Ontology alignment based on BM25 relevance scores.

ber of documents indexed, and k_1 and b are free parameters used to scale the document term frequency and document length respectively.

BM25 in Ontology Alignment: Using BM25 (Eq. 1), we can derive the relevance score for each vulnerability-video pair. These derived scores become confidence measures for the alignment links between a vulnerability-video pair, and are later materialized into our knowledge base using the *SimilarityMeasure* class and the measure:measuresThing and measure:hasMeasureValue properties (see Fig. 7). The measure:measuresThing property identifies the vulnerability and video facts that are being compared, while the measure:hasMeasureValue property stores the numeric similarity value. With this new knowledge, the SWRL rule in Listing 2 can be executed to establish vidont:describedBy relations (similar to those inferred in Listing 1) between vulnerability and video instances captured by the SEVONT and VIDONT ontologies.

Given our populated ontologies, it is now possible to infer implicit knowledge using the describedBy link between a vulnerability and a screencast. These links are instances where either a CVE ID is explicitly mentioned in a video (Fig. 8a) or the link is inferred when the similarity measure between the vulnerability index and video text queries is within the specified threshold (Fig. 8b).

SBSON and VIDONT Ontology Alignment. Having the SEVONT-VIDONT alignment and existing SEVONT-SBSON alignment (see Section 3.5), we are now able to infer indirect traceability links between VIDONT and SBSON by taking advantage of Semantic Web inferencing services such as owl:sameAs and *owl:TransitiveProperty*.

Same-As Inference: The same-as inference is commonly used to align two semantically equivalent concepts or individuals. For example, in our prior work [4,22], we used the owl:sameAs property to align vulnerable project releases from the SEVONT ontology to their corresponding instances in SBSON ontology based on a similarity threshold. Using the SPARQL query shown in Listing 3, we can now take advantage of this knowledge to establish indirect links from videos to project releases in the SBSON ontology (e.g., retrieve metadata of projects affected by a vulnerability described in a video).

Transitive closure inference: A relation R is said to be transitive if R(a,b) and R(b,c) implies R(a,c); this can be expressed in OWL through the *owl:TransitiveProperty* construct. In SBSON, we define a dependency between projects using the bi-directional transitive seon :dependsOn property to allow us to retrieve a list of all releases that have a direct or transitive dependency on a specified project release, and vice versa. Using the SPARQL query in Listing 4, we can now establish indirect links from videos to this project dependency knowledge in the SBSON ontology (e.g., to verify if a project transitively uses another project affected by a vulnerability described in a video).

```
Video(?video), Vulnerability(?vuln), measuresThing(?measure, ?video), measuresThing(?measure, ?vuln),
hasMeasuresValue(?measure, ?relevanceScore), swrlb:greaterThan(?relevanceScore, thresholdValue)
→ describedBy(?vuln,?video)
```

Listing 2. SWRL rules for aligning CVE facts with the video ontology based on their relevance score.

```
SELECT ?video ?release ?name ?desc ?version ?url
WHERE {
   ?vulnerability vidont:describedBy ?video.
   ?release a sbson:BuildRelease.

   #using the same-as links between SEVONT and SBSON release instances
   ?vulnerability sevont:affectsRelease ?release.
   ?release seon:hasName ?name ; sbson:hasVersionNumber ?version ;
        seon:hasURL ?url ; seon:hasDescription ?desc.
}
```

Listing 3. SPARQL query returning videos and related project details.

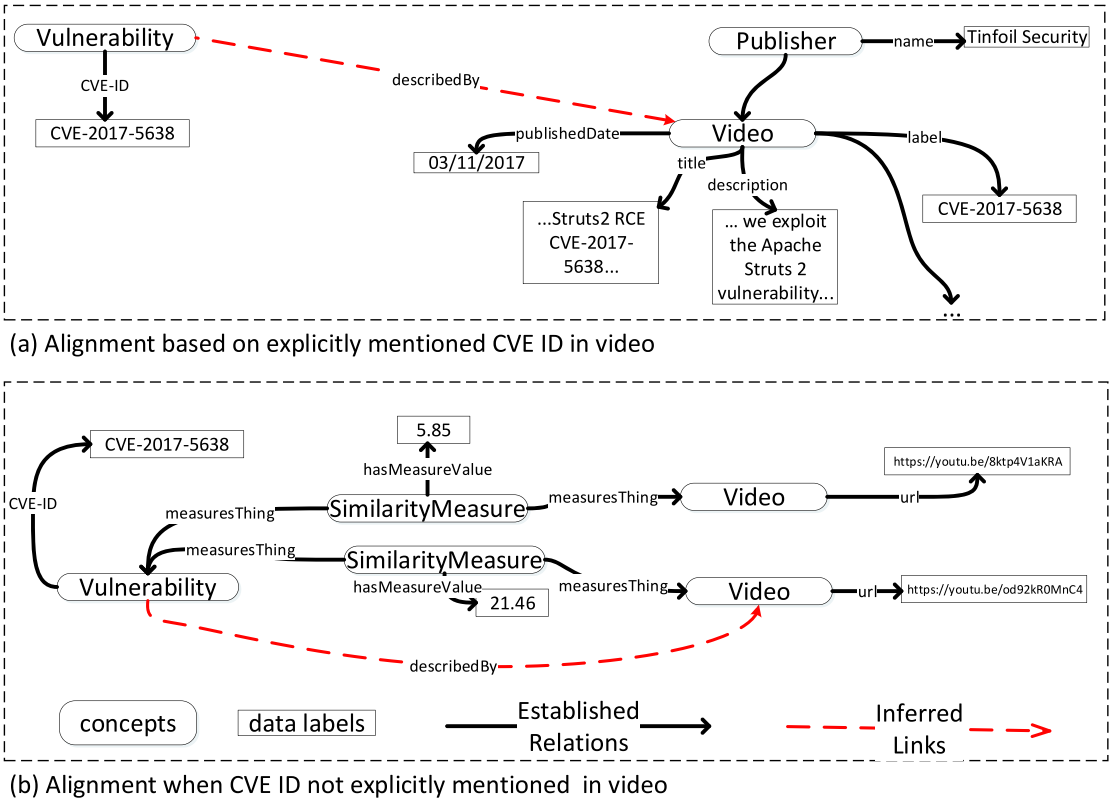


Fig. 8. Inferring the describedBy link between vulnerability and screencast instances when a CVE ID is (a) explicitly mentioned in a video, and (b) not mentioned.

```
SELECT ?video ?release ?dependent
WHERE {
  # using the transitive inference to detect dependencies on a vulnerable release with a video description
  ?dependent sbson:hasBuildDependencyOn ?release option (transitive).
  ?vulnerability vidont:describedBy ?video.
  ?release a sbson:BuildRelease.
  ?vulnerability sevont:affectsRelease ?release.
}
```

Listing 4. SPARQL query returning videos, related projects, and their dependencies.

5. Case study: CVE-2017-5638

In what follows, we report on results from a case study that we conducted to illustrate the applicability of our approach and to answer our research question from Section 2.2.

5.1. Case study setup

5.1.1. Dataset

We use a publicly disclosed vulnerability, CVE-2017-5638,[25] which has been reported in the NVD repository as a vulnerability affecting several Apache Struts[26] releases. Our dataset for this case study includes data from NVD, Maven Central, and YouTube.

Our vulnerability dataset consists of all NVD vulnerability XML feeds published since 1990. The dataset includes 74,402 unique vulnerabilities that affect 186,212 projects. Our Maven Central dataset consists of 178,763 unique projects with 1849,756 releases.[27] For our video dataset, we downloaded 48 YouTube videos related to the CVE-2017-5638 vulnerability.

The videos were selected using search queries developers would use when manually searching YouTube for videos related to this specific vulnerability: "CVE-2017-5638″, "CVE-2017-5638 Apache Struts", "input validation vulnerability Apache Struts", and "input validation vulnerability exploitation", with input validation corresponding to the CWE vulnerability category. From these search results, we selected 48 videos that also matched additional selection criteria such as video length and video resolution. We only selected screencasts with a length between 20 s and 12 min to ensure that they contain sufficient data in terms of speech and video content and where the screencast was recorded as High Definition (HD) to allow for a more accurate information extraction from the image processing step.

Of the selected 48 videos, 39 videos explicitly mention the CVE ID in either their title, description or video content. The 9 videos that do not mention explicitly the CVE ID were manually evaluated to ensure they were related to the CVE-2017-5638 vulnerability. Among the videos only 13 videos have an English narration. Table 2 provides an overview how many of the videos in our dataset capture the CVE-ID in the different video artifacts (speech part, image frames, video title, and video description), with (√) indicating that the CVE-ID was explicitly mentioned and (✗) indicating the CVE-ID was not mentioned at all.

[25] https://nvd.nist.gov/vuln/detail/CVE-2017-5638.
[26] https://struts.apache.org/.
[27] Dataset last updated 2017-10-23

Table 2
Classification of the video dataset based on the presence of CVE ID.

Presence of CVE ID				
title	description	image frames	speech	# of videos
✗	✗	✗	✗	9
√	✗	✗	✗	5
✗	√	✗	✗	3
✗	✗	√	✗	2
√	√	✗	✗	2
√	✗	√	✗	11
✗	√	√	✗	5
√	√	√	✗	9
√	√	√	√	2

5.1.2. Applying our methodology

For all screencasts, we downloaded the audio (speech part) and automatically transcribed the speech using IBM Watson's Speech-To-Text (STT) service. In our previous work [10], we have conducted a comparison of different STTs by manually transcribing videos and comparing the output of the IBM Watson STT tool against those of 4 other tools in terms of precision and recall values. In this comparison, IBM Watson obtained the highest precision and recall values (0.75 and 0.88, respectively). We used FFmpeg to extract image frames from the downloaded videos at a rate of 1 frame per second (to reduce the number of continuous frames with duplicate content). Next, we manually checked and removed image frames at the beginning and end of the screencasts which contain non-relevant information (e.g., greetings, introducing the YouTube channel or video creator, inviting people to like/subscribe, etc.) to reduce the amount of noise in our screencast data set. It should be noted that this could be also be automated (e.g., by removing the first and last 10 s of each video. Then we use Google Vision API's text recognition service to perform Optical Character Recognition (OCR) and automatically extract all text from the remaining image frames.

We also automatically extracted the title, description, publication date, and publisher information from the metadata provided with each video (if applicable) using the youtube-dl tool. Using a regular expression, we then searched for the CVE ID in the speech, image text, and metadata to label each video with a CVE ID and populate our knowledge base.

As part of the next processing step, we create an inverted index using the extracted text from the vulnerability dataset as our document corpus. More specifically, for this index, we treat each vulnerability as an individual document, with its own document id and its textual content being a bag of words. We then apply a simple preprocessing step consisting of case-folding, stop word removal, and stemming before indexing the document. For our case study, we use the extracted text from the metadata, images, and speech of our videos as query terms and populate the *SimilarityMeasure* instances (using the BM25 equation discussed in Section 4.2.3) for the Top 10 ranked vulnerabilities returned by each search query. Applying the semantic rules introduced earlier (Listings 1 and Listing 2), we can now automatically infer bidirectional traceability links from the screencast instances to their related vulnerability instances.

5.1.3. Evaluation measures

For our case study, we evaluate the linking accuracy of our approach. The objective of this evaluation is to validate whether our modeling approach is indeed capable of inferring hidden and indirect links between videos and other software repositories.

As part of our evaluation, we compared the retrieved ranking results against our video dataset oracle (baseline) that was created using manually selected and verified videos covering the CVE-2017-5638 vulnerability. Since in most cases we only have one relevant result (vulnerability) that matches a query (video), we used the position of the first true positive in the search result ranking as our assessment criteria. For the evaluation, we used the Reciprocal Rank (RR) measure. RR considers the position of the first true positive in the search results [24] and is calculated using the following formula:

$$RR = \frac{1}{rank\ of\ the\ 1st\ TP} \quad (2)$$

For the second part of the study, we used BM25 [24] to evaluate the ability of our approach to rank documents that do not contain an explicit mentioning of the CVE ID in neither their metadata nor screencast content.

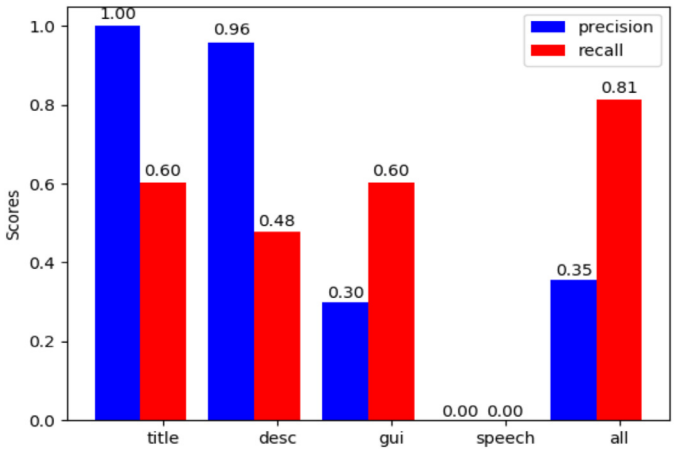


Fig. 9. Precision and recall of our CVE ID alignment, using only video title, description, image frames ("gui"), speech, or all the above ("all").

5.2. Case study results

RQ. How accurate are the bi-directional links, between known vulnerabilities and screencasts published on YouTube, provided by our knowledge model?

In what follows, we evaluate the linking accuracy of our approach, by comparing it with our baseline (our initial labeled video dataset) of 48 vulnerability-screencast pairs. The objective of this section is to illustrate that our modeling approach is indeed capable of inferring hidden and indirect links between video and other software repositories with sufficient accuracy.

For our evaluation, we use precision and recall measures, with true positives being the number of vulnerability-screencast pairs correctly matched, while false positives correspond to the number of vulnerability-screencast pairs incorrectly matched. For recall, false negatives correspond to the number of correct vulnerability-screencast pairs that were not identified by our approach.

Evaluation of CVE ID Alignment: Fig. 9 shows the results of our approach when using our 39 videos that explicitly mention a CVE ID in their title, description or content. The results show that, our approach achieves, as expected, high precision values of 1.0 and 0.96 when a video explicitly mentions a CVE ID in its title or description. During our analysis, we also observed that the text extracted from the image frames explicitly mentioned several other CVE IDs (usually related to the CVE-2017-5638 vulnerability) which resulted in a low precision of 0.30, which also affects the precision of our approach when all information sources are used together.

A manual inspection of the inferred links revealed that the lower recall (Fig. 9) for the individual information resources in a screencast is due to inaccuracies found in the transcription, text of the image frames, or the metadata of the screencasts. Also, depending on the accuracy of the speech to text transcriptions, numbers mentioned within the CVE ID in the speech part of a video are often erroneously transcribed into another word(s), which no longer allow us to link these CVE IDs with

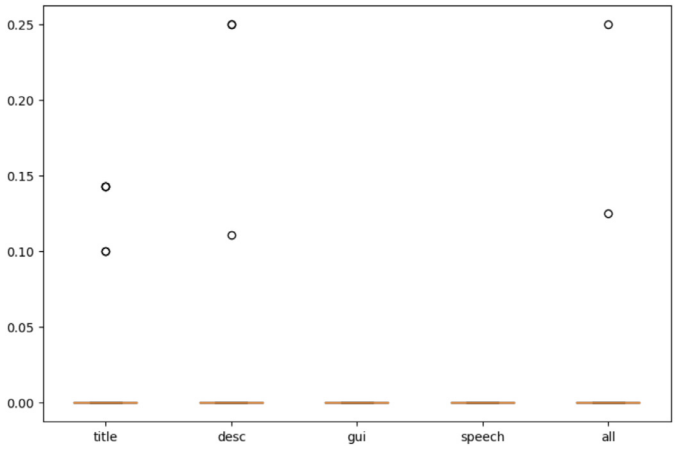


Fig. 10. Reciprocal Rank results for BM25 Alignment evaluation.

their NVD counterparts. However, if all information resources are combined, we were able to achieve a higher recall of 0.81. The evaluation also highlights the impact of different information resources and their trade-off on precision and recall. Using only available title and description information, our approach can achieve high precision, but recall will be quite low. In contrast, by taking advantage of all available information resources, our linking approach will achieve a much higher recall but also a significant lower precision, due to many false positives.

Evaluation of BM25 Relevance Alignment: As our previous evaluation showed, our approach can link in most cases successfully NVD and Video content if a CVE ID is present. In this part of our evaluation, we focus on the ability to rank documents that do not contain any explicit mentioning of the CVE ID in either their metadata or the screencast content with the description found for the vulnerability in the NVD repository. As part of this evaluation process, we are interested in the ranking of relevant (true positive) results in the top hits of the result set. For the evaluation, we first manually searched and removed all instances of CVE IDs found in the 48 videos of our video dataset and then re-applied our linking approach on this data set. For the evaluation, we compared the linking results, with the results from our initial (labeled) video dataset. Fig. 10 shows that the median RR for our dataset without any explicit mentioning of the CVE ID is 0.01, which means that our top 10 results rarely contain any true positive. The main reason for the poor performance of our approach is that the vulnerability related text in the analyzed videos is too generic and inclusive, to allow for a relevant matching between the video content and a specific CVE ID (and its description).

To further improve the ranking results of our approach, we narrowed the vulnerability search space by taking advantage of available semantic information such as affected projects and vulnerability category keywords found in the video text. We used DBPedia Spotlight,[28] a tool which automatically annotates mentions of DBPedia[29] resources in a given text, to identify any reference to a software product (which are represented as entities of DBpedia's Software class) in our video and vulnerability datasets. DBpedia is a linked-data knowledge base that provides a rich source of RDF descriptions of million entities such as companies and products. Using this approach, we can reduce our vulnerability search space (from 74,402 to 73 vulnerabilities), by including only those vulnerabilities which were annotated with the same software products shown in the videos of our dataset. As the RR results in Fig. 11(a) show, our semantic alignment process shows a significant improvement for RR values when all our information sources are used on the reduced number of vulnerabilities. Using only the text from the image frames (i.e., gui text) resulted in the highest RR value of 0.16. This can be interpreted as follows: 100% of the time, the first true positive (matched vulnerability) was ranked in the 6th position in the result set. The average median RR value over all boxplots increased from 0.01 to 0.11, representing a 9th position rank for the first true positive, 50% of the time. A further manual analysis of the results showed that 3 videos are ranked in the 1st position when we used only video descriptions, and 1 video is ranked in the 1st position when we used either the video titles only or all the combined information sources as our query terms during the alignment.

We further reduced the vulnerability search space (from 73 to 14 vulnerabilities) by performing a hierarchical search, first by comparing video text with vulnerability category (CWE) descriptions and then comparing videos with vulnerabilities within the top 10 ranked category results. Using this hierarchical search approach, we now only include vulnerabilities of the same category. Fig. 11(b) shows the RR results after applying this filtering by "product and CWE category". The average median RR value of all the boxplots improved to 0.27, indicating that the 1st true positive is now ranked 50% of the time at the 4th position in the results set. Also, the highest median RR value increased to 0.5, indicating that the 1st true positive is now 100% of the time ranked in the 2nd position text when only text from image frames (i.e., gui text) are used. However, we did not see any significant improvements in the number of true positives ranked in the first position (p-value = 0.81). These initial results look promising and provide avenues for future enhancements to our approach.

A key finding from our assessment is that when a CVE ID is not explicitly mentioned, text from video image frames is the most informative (relevant) resource to be used for linking videos to vulnerabilities.

We also believe that similar to how traceability links are generated between unannotated commits and issues, techniques such as comparing video and vulnerability publication dates and identifying the presence of the vulnerable source code elements (class or method names) within the content of videos can be used to improve the accuracy of our proposed approach. Also, creators of vulnerability related videos should follow more rigorously existing screencast best practices (e.g., [7]) and provide more precise and concise vulnerability information in their videos to allow for easier integration (linking) of video content with other software artifacts. For example, CVE IDs, name of the affected project being demonstrated, and keywords from the vulnerability classification should be explicitly included in the video.

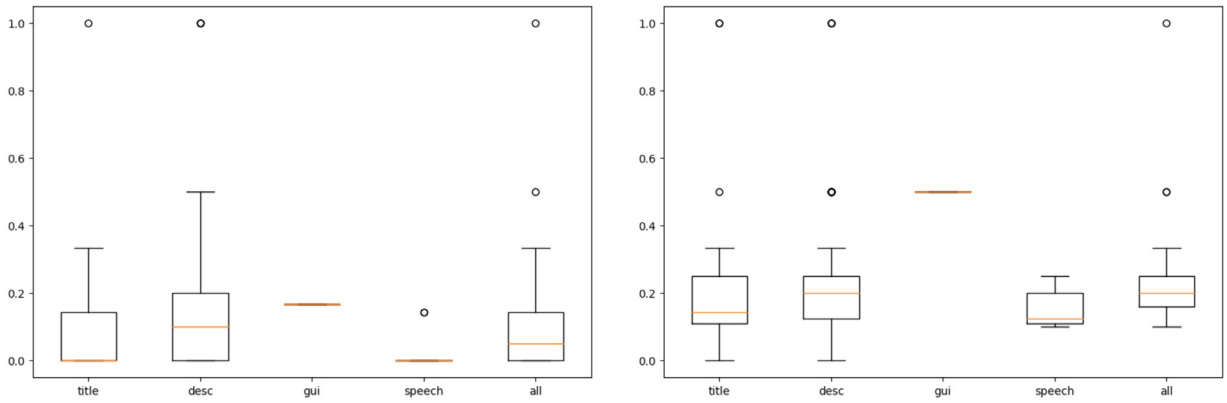
Another finding from our case study is that both short and concise information sources such as video titles and description, and verbose sources such as video speech and text displayed in image frames are important for our linking approach. For example, Fig. 11(a) and 11(b) above show that the results obtained by using only text extracted from image frames as queries are ranked higher than the other information sources on average, probably because they contain more content than the short descriptions/titles.

5.3. Discussion

As our evaluation shows, although videos and NVD contain different types of information describing known vulnerabilities, similarities and semantics captured by these artifacts can be used to allow for the linking of software knowledge resources (NVD) with video content describing such vulnerabilities. As the case study further illustrates, our modeling approach can indeed allow for the integration of these heterogeneous knowledge resources by transforming these traditional information silos into information hubs, where knowledge can be seamlessly shared and reused across resource borders. In addition, our ontology-based knowledge model provides a machine-human readable representation that supports incremental knowledge population based on the Open World Assumption.

Furthermore, our previously established bi-directional traceability links from Maven Central to NVD security vulnerabilities [4,22] enable us to infer indirect traceability links between screencasts and Maven

[28] https://www.dbpedia-spotlight.org/.
[29] https://wiki.dbpedia.org.



(a) RR results after "product" filtering of search space

(b) RR results after "product and CWE category" filtering of search space

Fig. 11. Reciprocal Rank results after narrowing the vulnerability search space.

Table 3
Examples of vulnerabilities and their associated YouTube videos.

Vulnerability	*Related Video*	*Vulnerable Projects*	*# (Maven Central) Dependencies on Vulnerable Project*
CVE-2017-5638	https://youtu.be/od92kR0MnC4	Apache Struts 2.3.16.3	5927
		Apache Struts 2.3.16.1	5340
		Apache Struts 2.3.8	1173
		Apache Struts 2.3.16	585

project dependencies. The Maven repository includes many vulnerable projects/components that are commonly reused within existing projects in the Maven ecosystem. More specifically, while a project might not have any direct vulnerability reported in the NVD database, it can still be potentially affected indirectly through its dependency on other (external) vulnerable libraries and components. Providing traceability links between screencasts describing vulnerabilities found in projects (or their direct/indirect dependent components) can provide developers with additional insights in the potential threats their project is exposed to and help them to mitigate the security issues.

To illustrate how our knowledge model can be used to infer new knowledge, we revisit our motivating example (scenario #1) from Section 2, where Bob is following the instructions shown in a screencast implementing his project without being aware that the component which is explained in the screencast is using (dependent) a vulnerable third-party API. Given our knowledge model, we are now able to first identify vulnerable APIs or components Bob's project might directly or indirectly depend on and then recommend him a screencast that illustrates these known vulnerabilities. For example, Table 3 shows the four most commonly used Apache Struts releases affected by the vulnerability CVE-2017-5638 and the number of projects dependent on them, based on Listing 4 (Section 4.3.1). As shown in the table, 5927 Maven Central projects declare a dependency on Apache Struts 2.3.16.3. Any project that uses, either directly or indirectly, one of these Apache Struts releases could benefit from the YouTube video listed in the table. The video not only illustrates how the vulnerability CVE-2017-5638 can be exploited but also shows how to mitigate it.

It should be noted that the accuracy of these indirect links is dependent on the accuracy of the SBSON-SEVONT alignment (provided by our previous SV-AF approach [4]) and the SEVONT-VIDONT alignment introduced in this paper. In our previous work, we reported a precision of 0.87 and a recall of 0.64 for linking our SBSON-SEVONT ontologies.

6. Threats to validity

Our research is introducing a methodology for integrating relevant content from screencast tutorials, which are created by the crowd, with other software security knowledge resources. However, some threats to validity exist that might affect our reported results and the applicability of our approach.

Construct Validity: We identify three threats that relate to the tools and mechanisms used to obtain our results. The first threat is that our case study relies on our ability to mine facts from both YouTube and the NVD repository to populate our ontologies. A common problem when mining software repositories is that these repositories often contain noise in their data, due to data ambiguity, inconsistencies, or incompleteness. Although studies (e.g., [25]) have shown the presence of incorrect NVD information, this threat is partly mitigated since vulnerabilities published in NVD are curated by security experts. Similarly, the Maven tool ensures that defined project dependencies are fully specified and available in the Maven Central repository, limiting not only ambiguities and inconsistencies at the project build but also at the complete dataset level.

Regarding the knowledge extracted from YouTube, not all vulnerability related videos explicitly mention a vulnerability CVE in their title, description, or content (i.e., image frames text or speech). As our case study has shown, this can significantly reduce the accuracy of our approach to automatically link such videos to the related vulnerabilities. To address this potential limitation, we use the BM25 information retrieval approach [23] to identify NVD vulnerabilities that are most similar to a given video to further improve our alignment between these two resources, since BM25 performs well on shorter textual descriptions. While we are not able to mitigate this threat completely, different approaches and techniques (such as reducing the search space) can be used to improve the linking results.

Another potential threat is the automatic transcription of videos, a process that is prone to errors and potentially can cause CVE IDs not being correctly transcribed. Using a specialized gazetteer list (e.g., WordNet [26]) to detect and correct erroneously transcribed CVE IDs can be considered as future work to improve the accuracy of our approach in terms of identifying these CVE IDs.

The final threat to construct validity is related to the extraction of project dependency information. The work in this paper does not consider dependency scopes, configurations, and exclusions during project

dependency resolution; this can introduce false positives when identifying potentially vulnerable projects based on transitive project dependencies. For future work, we plan to extend our dependency analysis to cover such cases.

Internal validity: One internal threat that potentially can affect our results is the quality of the established links from vulnerabilities to release builds which we used to answer our RQ. SV-AF approach establishes these traceability links with a precision of 0.87 and a recall of 0.64. In addition, we compared SV-AF against a publicly available [27] and a proprietary tool[30] (now open source) [28]; SV-AF's accuracy compared favorably against the free tool and just below the proprietary tool.

External validity: In terms of external threats to validity, a potential threat is that the presented experiments are not generalizable to non-YouTube videos and non-NVD vulnerabilities. This threat can be mitigated by our modeling approach with its different abstraction layers. Our domain-specific ontologies (e.g., SEVONT and VIDONT) contain the core shared concepts and relations common to that particular domain. These domain ontologies can be extended and instantiated to include system level ontologies that support new vulnerability and video repositories. Also, our dataset can be considered incomplete, covering only a limited number of existing videos and vulnerabilities, limiting the generalizability of our results. To mitigate this problem, we plan to extend the datasets as part of our future work to include a larger number of videos and vulnerabilities in our analysis.

7. Related work

Given the diversity in software development processes and their distributed nature, there is a need for knowledge integration and sharing among software artifacts, to improve knowledge reuse and allow for new types of analyses across resource boundaries. Several semantic-web based approaches have been proposed that use ontologies to establish taxonomies in the software engineering domain (e.g., [29,30]). These ontologies describe and capture domain knowledge of developers, source code, and other software artifacts. Also, other approaches have been proposed to address the issue of seamless integration of these knowledge resources [5,31]. While all these approaches aim to promote the inference and integration of new knowledge in an existing knowledge base, they have not considered crowd-based (multimedia) documents as part of their solution space.

Crowd-based documents have become an increasingly popular reference for learning software development/maintenance related skills. This has motivated researchers to embark on research in different areas of extracting information from crowd-based documents and linking these documents to their associated artifacts [32–34]. More specifically, recent work has focused on analyzing tutorial screencasts [1,7,8,10,35–39], since screencasts contain tacit knowledge shared by developers and are being frequently produced and used for learning purposes [7].

In what follows, we discuss the work the closest related to ours, on vulnerability dependency analysis, semantic-web enabled software analysis research, as well as mining and linking of crowd-based documents and screencasts.

7.1. Vulnerability analysis in software dependencies

Several static vulnerability analysis and detection approaches (tools) exist (e.g. [28,40–43]) that identify vulnerability dependencies in the source code. Common to these approaches is that they identify and track security vulnerabilities and their dependencies at the project level. In contrast, our approach also includes a global dependency analysis of vulnerabilities across project boundaries. Our approach therefore not only allows us to integrate different information resources as part of the analysis, but also provides us with the ability to take advantage of semantic reasoning services to infer implicit facts about the vulnerable code usages within the system, to support bi-directional dependency analysis – including both impacts to external dependencies and vice versa.

Among the existing research most closely related to ours are Cadariu et al. [44], Plate et al. [28], Ponta et al. [45], Decan et al. [46], and Pashchenko et al. [47]. Cadariu et al. [44] introduce in their Vulnerability Alert Service (VAS) an approach that notifies users if a vulnerability is reported for software systems. VAS depends on the OWASP Dependency-Check tool [27]. Plate et al. [28] proposed a technique that supports the impact analysis of vulnerability based on the dynamic analysis of code changes introduced by security fixes. Their work was extended by Ponta et al. [45] to include static analysis of code changes and provide a novel combination of static and dynamic analysis. Among the other related work, Decan et al. [46] perform an empirical study of the evolution of vulnerabilities within the npm ecosystem and Pashchenko et al. [47] propose an approach for the reliable measurement of vulnerable dependencies in OSS libraries.

Several studies have shown that projects are becoming increasingly susceptible to security vulnerabilities due to the rate at which software libraries are reused within projects. Alqahtani et al. [48] show 750 Maven projects (0.062% of all Maven projects) contain known security vulnerabilities that have been reported in the NVD database. A study by Kula et al. [49] on 4600 GitHub projects showed that 81.5% of them do not update their direct dependencies on vulnerable libraries. A similar study by Eghan et al. [50] on the dependencies of four popular vulnerable projects showed that 36.7% of these projects' dependents updated their dependency to more vulnerable versions. Our approach of providing traceability links between vulnerabilities, project dependencies, and online screencasts addresses the lack of awareness about security vulnerabilities.

7.2. Semantic-Web enabled software analysis

Hyland-Wood et al. [51] proposed an OWL ontology of software engineering concepts, including classes, tests, metrics, and requirements. Bertoa et al. [52] focused on software measurement. Witte et al. [18] used text mining and static code analysis to map documentation to source code in RDF for software maintenance purposes. Yu et al. [29] also model static source code information using an ontology and take advantage of SWRL rules to infer common bugs in the source code. Happel et al. [30] proposed KOntoR, which conceptualizes knowledge about software artifacts, such as the programming language used or licensing models. Dietrich et al. [53] developed a tool that scans the abstract syntax tree of Java programs and detects design patterns, described in terms of OWL ontologies, for documentation purposes.

Several researchers (e.g., [31,54]) have modeled software evolution knowledge found in software repositories as ontologies. Their approaches integrate different artifacts to facilitate common repository mining activities. Tappolet [55] presents a roadmap towards integrating semantics of different software project repositories in three main steps: 1) data representation using RDF/OWL ontologies, 2) intra-project repository integration, and finally 3) inter-project repository integration. Based on these ideas, Kiefer et al. [54] presented EvoOnt, which introduces and integrates the source code, bug and versioning system ontologies. EvoOnt also takes advantage of Semantic Web reasoning services to detect bad code smells, calculate metrics, and to extract data for visualizing changes in code over time. Iqbal et al. [56] presented their Linked Data Driven Software Development (LD2SD) methodology to provide a uniform and centralized RDF-based access to JIRA bug trackers, Subversion, developer blogs, project mailing lists. Wursch et al. [5] presented SEON, a family of ontologies that describe many different facets of a software's life-cycle. SEON is unique in that it comprises multiple abstraction layers. Our core ontologies build upon the SEON knowledge model, which we further extend to support additional soft-

[30] The tool in [28] was proprietary at the time the evaluation was performed in our previous work.

ware artifacts (e.g., build systems ontology, Video ontology, vulnerability ontologies) and additional reasoning.

Given that these approaches are all based on RDF as a standardized knowledge representation format, we can envision interesting interactions between our knowledge models and the ontologies presented by the other authors. Such extensions could lead to a completely new family of software analysis services or at least simplify the implementation of existing ones.

7.3. Linking crowd-based documents

Jiau and Yang [32], proposed an approach that recovers traceability links between API classes and question and answers on StackOverflow to improve document coverage. Their goal is to reduce the inequality of crowdsourced API documents in StackOverflow since a larger proportion of existing discussions and "question and answers" address a smaller portion of topics. Barzilay et al. in [33], developed Example Overflow, which is a code search tool on top of StackOverflow to extract high-quality code examples. As part of their empirical analysis, they studied the type of questions posted on StackOverflow and to what extent these questions could be answered by their approach. Subramanian et al. [34] proposed a method for linking code examples on StackOverflow to API documentation. Based on the proposed method, they implemented a tool, Baker, that links code snippets to Java classes and methods or JavaScript functions, with an observed precision of 97%. Bao et al. [35, 36] proposed a method of tracking user activities and developed a tool, ActivitySpace, to support inter-application information needs of software developers. The tool reduces the efforts of developers while locating documents and recalling their history activities in daily work. In contrast to this existing research, our work focuses on analyzing multimedia crowd-based documentation [1].

7.4. Analyzing software engineering screencasts

The first study in the area of using crowd-based screencasts to share and document developer knowledge was conducted by MacLeod et al. [7]. They investigated the goals and techniques of developers in creating screencasts and the benefits and challenges of this type of knowledge sharing. As part of their work, they analyzed 20 tutorial screencasts and interviewed 10 developers/YouTubers. They found that by creating screencasts, developers demonstrate and share how to customize a program, the challenges they encountered and their development experiences, solutions to problems, how to apply design patterns, and their programming language knowledge. They also observed that developers are creating these screencasts to promote themselves and gain reputation by helping others. An extension of this study [39] compared how Ruby on Rails screencasts are hosted and shared on YouTube (a free platform) to how they are shared on a formal screencast site like RailsCasts,[31] which is a paid platform. Finally, they extracted guidelines for screencast creators to produce clear and understandable screencasts.

In earlier work [10], we used the speech component of screencasts to provide relevant information to various software engineering tasks. For example, by leveraging information extraction techniques (e.g., LDA) on the spoken text, we were able to extract steps of use-case scenarios from the videos. In an extension of the previous work, we proposed a feature location approach [1] that links software application features demonstrated in how-to-tutorials to their corresponding source code implementation.

Ponzanelli et al. [8,37] developed an approach to extract relevant fragments of software development tutorial videos and link them to relevant StackOverflow discussions by mining the (captioned) speech and GUI content of the video tutorials. In line with the aforementioned works on software development tutorial videos, Yadid et al. [38] developed an approach to extract code from programming video tutorials to enable deep indexing of them. They attempted to consolidate code across multiple image frames of the videos and used statistical language models to make corrections on the extracted code.

8. Conclusion and future work

Developers often resort to a variety of knowledge resources, including informal resources such as screencasts and video tutorials when comprehending and analyzing software systems. Though many screencasts and video tutorials exist, these resources are rarely integrated with other software-related knowledge resources. The objective of our research is to provide a standardized ontological representation that allows for seamless knowledge integration at different abstraction levels across knowledge resource boundaries. Having this knowledge integration not only provides developers with direct access to vulnerability information described in a screencast content, but also allows us to link vulnerability descriptions to relevant screencasts and dependency information. In addition, our approach also allows developers to identify screencasts that demonstrate such attacks and provides developers who are indirectly using vulnerable libraries in their project (e.g., through Maven dependencies) with insights on how to reduce the potential impact of being directly or indirectly exposed to a vulnerability.

We evaluate the flexibility and applicability of our approach to 1) provide bi-directional links between known vulnerabilities and screencasts published on YouTube, and 2) link relevant NVD entries, screencasts, and build dependencies to provide developers and maintainers with valuable insights during vulnerability management and impact (ripple effect) analysis. We performed a case study on 48 videos that describe the exploits of known vulnerabilities and how these vulnerabilities can be fixed. Our evaluation shows our approach can successfully link relevant vulnerabilities and screencasts with an average precision of 98% and an average recall of 54% when vulnerability identifiers (CVE ID) are explicitly mentioned in the videos. When no direct reference to a CVE ID exists in the screencast, our approach was still able to link video-vulnerability descriptions, with up to 100% of the time relevant links being ranked in the 2nd position of our results set.

As part of our future work, we plan to make the filtering process (introduced in the case study) to reduce the search space an integrated part of our linking approach by extending our current knowledge model to include a CWE ontology and link to DBpedia. Having such a unified knowledge representation, will allow us to infer additional knowledge and further restrict our search space during the linking process of videos which do not contain a CVE ID. We also plan to conduct another empirical study to improve the generalizability of our approach by covering different vulnerability and video repositories. In addition, having a larger dataset would also allow us to classify vulnerability related videos based on their popularity and content. We also consider extending our modeling approach to integrate videos and their content with other software artifacts such as blogs and Q/A forums (e.g., StackOverflow) to derive new application scenarios for our modeling approach.

Declaration of Competing Interest

None.

References

[1] P. Moslehi, B. Adams, J. Rilling, Feature Location using Crowd-based Screencasts, in: Proceedings of the 15th IEEE Working Conference on Mining Software Repositories (MSR), 2018.
[2] P.T. Devanbu, S. Stubblebine, Software engineering for security: a roadmap, in: Proceedings of the Conference on the Future of Software Engineering, 2000, pp. 227–239.
[3] A.E. Hassan, The road ahead for Mining Software Repositories, in: 2008 Frontiers of Software Maintenance, 2008, pp. 48–57.
[4] S.S. Alqahtani, E.E. Eghan, J. Rilling, SV-AF - A Security Vulnerability Analysis Framework, in: IEEE 27th International Symposium on Software Reliability Engineering (ISSRE), 2016, pp. 219–229.

[31] http://railscasts.com.

[5] M. Würsch, G. Ghezzi, M. Hert, G. Reif, H.C. Gall, SEON: a pyramid of ontologies for software evolution and its applications, Computing 94 (11) (Nov. 2012) 857–885.
[6] I. Duncan, L. Yarwood-Ross, C. Haigh, YouTube as a source of clinical skills education, Nurse Educ. Today 33 (12 (December)) (2013) 1576–1580.
[7] L. MacLeod, M.-A. Storey, A. Bergen, Code, Camera, Action: How Software Developers Document and Share Program Knowledge Using YouTube, 2015 IEEE 23rd Int. Conf. Progr. Compr., 2015.
[8] L. Ponzanelli, et al., Too long; didn't watch!, in: Proceedings of the 38th International Conference on Software Engineering - ICSE '16, 2016, pp. 261–272.
[9] P.K. Manadhata, J.M. Wing, An attack surface metric, IEEE Trans. Softw. Eng. 37 (3) (2011) 371–386.
[10] P. Moslehi, B. Adams, J. Rilling, On mining crowd-based speech documentation, in: Proceedings - 13th Working Conference on Mining Software Repositories, MSR 2016, 2016.
[11] J. Williams, A. Dabirsiaghi, The unfortunate reality of insecure libraries, Asp. Secur. Inc (2012) 1–26.
[12] I. Sonatype, Maven: The Definitive Guide, O'Reilly, 2008.
[13] T. Berners-Lee, J. Hendler, O. Lassila, The Semantic Web, Sci. Am. 284 (5 (May)) (2001) 34–43.
[14] D.L. McGuinness, F. Van Harmelen, Owl web ontology language overview, W3C Recomm. 10.2004-03 2004 (February) (2004) 1–12.
[15] C.J.H. Mann, The Description Logic Handbook – Theory, Implementation and Applications, Kybernetes 32 (9/10) (2003) 2003.06732iae.006, Dec.
[16] B. DuCharme, Learning SPARQL, 2n Edition, O'Reilly Media, 2011.
[17] B. Henderson-Sellers, Bridging metamodels and ontologies in software engineering, J. Syst. Softw. 84 (2 (February)) (2011) 301–313.
[18] R. Witte, Y. Zhang, J. Rilling, Empowering software maintainers with semantic web technologies, in: Eur. Conf. Semant. Web Res. Appl., 2007, pp. 37–52.
[19] C. Atkinson, M. Gutheil, K. Kiko, On the relationship of ontologies and models, in: Proc. 2nd Work. MetaModelling Ontol. WoMM06 LNI P96 Gesellschaft fur Inform. Bonn, 2006, pp. 47–60.
[20] A. Kimmig, S. Bach, M. Broecheler, B. Huang, L. Getoor, A short introduction to probabilistic soft logic, in: Proceedings of the NIPS Workshop on Probabilistic Programming: Foundations and Applications, 2012, pp. 1–4.
[21] M. Cheriet, N. Kharma, C. Liu, C. Suen, Character Recognition Systems: A Guide For Students and Practitioners, Wiley-Interscience, 2007.
[22] S.S. Alqahtani, E.E. Eghan, J. Rilling, Recovering Semantic Traceability Links between APIs and Security Vulnerabilities: An Ontological Modeling Approach, in: Proceedings - 10th IEEE International Conference on Software Testing, Verification and Validation, ICST 2017, 2017, pp. 80–91.
[23] C.D. Manning, P. Raghavan, H. Schütze, An Introduction to Information Retrieval, Cambridge University Press, 2009.
[24] I. Keivanloo, Source Code Similarity and Clone Search, Concordia University, 2013.
[25] V.H. Nguyen, F. Massacci, The (un)reliability of NVD vulnerable versions data, in: Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security - ASIA CCS '13, 2013, pp. 493–498.
[26] S. Deerwester, S.T. Dumais, G.W. Furnas, T.K. Landauer, R. Harshman, Indexing by latent semantic analysis, J. Am. Soc. Inf. Sci. 41 (6) (1990) 391–407.
[27] J. Long, S. Springett, and W. Stranathan, "OWASP Dependency Check," 2015. [Online]. Available: https://www.owasp.org/index.php/OWASP_Dependency_Check. [Accessed: 30-Dec-2018].
[28] H. Plate, S.E. Ponta, A. Sabetta, Impact assessment for vulnerabilities in open-source software libraries, in: 2015 IEEE 31st Int. Conf. Softw. Maint. Evol. ICSME 2015 - Proc., 2015, pp. 411–420.
[29] L. Yu, J. Zhou, Y. Yi, P. Li, Q. Wang, Ontology model-based static analysis on java programs, in: 2008 32nd Annual IEEE International Computer Software and Applications Conference, 2008, pp. 92–99.
[30] H.-J. Happel, A. Korthaus, S. Seedorf, P. Tomczyk, KOntoR: an ontology-enabled approach to software reuse, in: Proc. Of The 18Th Int. Conf. On Software Engineering And Knowledge Engineering, 2006.
[31] J. Tappolet, C. Kiefer, A. Bernstein, Semantic web enabled software analysis, Web Semant. Sci. Serv. Agents World Wide Web 8 (2–3 (July)) (2010) 225–240.
[32] H.C. Jiau, F.-P. Yang, Facing up to the inequality of crowdsourced API documentation, ACM SIGSOFT Softw. Eng. Notes 37 (1 (January)) (2012) 1.
[33] O. Barzilay, C. Treude, A. Zagalsky, Facilitating Crowd Sourced Software Engineering via Stack Overflow, in: Finding Source Code on the Web for …, New York, Springer New York, 2013, pp. 1–19.
[34] S. Subramanian, L. Inozemtseva, R. Holmes, Live API documentation, in: Proceedings of the 36th International Conference on Software Engineering - ICSE 2014, 2014, pp. 643–652.
[35] L. Bao, Z. Xing, X. Wang, B. Zhou, Tracking and analyzing cross-cutting activities in developers' daily work, in: 30th IEEE/ACM International Conference on Automated Software Engineering (ASE 2015), 2015, pp. 277–282.
[36] L. Bao, J. Li, Z. Xing, X. Wang, X. Xia, B. Zhou, Extracting and analyzing time-series HCI data from screen-captured task videos, Empir. Softw. Eng. 22 (1) (Feb. 2017) 134–174.
[37] L. Ponzanelli, et al., Automatic Identification and Classification of Software Development Video Tutorial Fragments, IEEE Trans. Softw. Eng. (2017) 1.
[38] S. Yadid, E. Yahav, Extracting code from programming tutorial videos, in: Proceedings of the 2016 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software - Onward! 2016, 2016, pp. 98–111.
[39] L. MacLeod, A. Bergen, M.-A. Storey, Documenting and sharing software knowledge using screencasts, Empir. Softw. Eng. 22 (3 (June)) (2017) 1478–1507.
[40] M. Hirzel, D. Von Dincklage, A. Diwan, M. Hind, Fast online pointer analysis, ACM Trans. Program. Lang. Syst. 29 (2 (April)) (2007) 11–66.
[41] S. Mancoridis, B.S. Mitchell, Y. Chen, E.R. Gansner, Bunch: A clustering tool for the recovery and maintenance of software system structures, in: Software Maintenance, 1999.(ICSM'99) Proceedings. IEEE International Conference on, 1999, pp. 50–59.
[42] J.-D. Choi, M. Burke, P. Carini, Efficient flow-sensitive interprocedural computation of pointer-induced aliases and side effects, in: Proceedings of the 20th ACM SIGPLAN-SIGACT symposium on Principles of programming languages - POPL '93, 1993, pp. 232–245.
[43] N. Rutar, C.B. Almazan, J.S. Foster, A Comparison of Bug Finding Tools for Java, in: 15th International Symposium on Software Reliability Engineering, 2004, pp. 245–256.
[44] M. Cadariu, E. Bouwers, J. Visser, A. Van Deursen, Tracking known security vulnerabilities in proprietary software systems, in: 2015 IEEE 22nd Int. Conf. Softw. Anal. Evol. Reengineering, SANER 2015 - Proc., 2015, pp. 516–519.
[45] S.E. Ponta, H. Plate, A. Sabetta, Beyond metadata: code-centric and usage-based analysis of known vulnerabilities in open-source software, in: 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2018, pp. 449–460.
[46] A. Decan, T. Mens, E. Constantinou, On the impact of security vulnerabilities in the npm package dependency network, in: Proceedings of the 15th International Conference on Mining Software Repositories - MSR '18, 2018, pp. 181–191.
[47] I. Pashchenko, H. Plate, S.E. Ponta, A. Sabetta, F. Massacci, Vulnerable open source dependencies, in: Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM '18, 2018, pp. 1–10.
[48] S.S. Alqahtani, E.E. Eghan, J. Rilling, Tracing known security vulnerabilities in software repositories - a semantic web enabled modeling approach, Sci. Comput. Program. 121 (Feb. 2016) 153–175.
[49] R.G. Kula, D.M. German, A. Ouni, T. Ishio, K. Inoue, Do developers update their library dependencies? Empir. Softw. Eng. 23 (1 (February)) (2018) 384–417.
[50] E.E. Eghan, S.S. Alqahtani, C. Forbes, J. Rilling, API trustworthiness: an ontological approach for software library adoption, Softw. Qual. J. (2019) 1–46.
[51] D. Hyland-Wood, D. Carrington, S. Kaplan, Toward a Software Maintenance Methodology using Semantic Web Techniques, in: 2006 Second International IEEE Workshop on Software Evolvability (SE'06), 2006, pp. 23–30.
[52] M.F. Bertoa, A. Vallecillo, F. García, An Ontology for Software Measurement, in: Ontologies for Software Engineering and Software Technology, Berlin Heidelberg, Springer, 2006, pp. 175–196.
[53] J. Dietrich, C. Elgar, A Formal Description of Design Patterns Using OWL, in: Australian Software Engineering Conference, 2005, pp. 243–250.
[54] C. Kiefer, A. Bernstein, J. Tappolet, Mining Software Repositories with iSPAROL and a Software Evolution Ontology, Fourth International Workshop on Mining Software Repositories (MSR'07:ICSE Workshops 2007), 2007 10–10.
[55] J. Tappolet, Semantics-aware software project repositories, in: Proceedings of the European Semantic Web Conference, 8, Ph.D. Symposium, 2008, p. 8.
[56] A. Iqbal, G. Tummarello, M. Hausenblas, O.-E. Ureche, LD2SD: linked data driven software development, International Conference on Software Engineering & Knowledge Engineering, 2009.