Identifying, Prioritizing and Evaluating Vulnerabilities in Third Party Code

Cobleigh, Alexander; Hell, Martin; Karlsson, Linus; Reimer, Oscar; Sönnerup, Jonathan; Wisenhoff, Daniel

Published in:
IEEE 22nd International Enterprise Distributed Object Computing Workshop

DOI:
10.1109/EDOCW.2018.00038

2018

Document Version:
Peer reviewed version (aka post-print)

Identifying, Prioritizing and Evaluating Vulnerabilities in Third Party Code

Alexander Cobleigh
debricked AB
Malmö, Sweden
alexander.cobleigh@debricked.com

Martin Hell
Dept. of Electrical and Information Technology, Lund University
Lund, Sweden
martin.hell@eit.lth.se

Linus Karlsson
Dept. of Electrical and Information Technology, Lund University
Lund, Sweden
linus.karlsson@eit.lth.se

Oscar Reimer
Dept. of Electrical and Information Technology, Lund University
Lund, Sweden
oscar.reimer@eit.lth.se

Jonathan Sönnerup
Dept. of Electrical and Information Technology, Lund University
Lund, Sweden
jonathan.sonnerup@eit.lth.se

Daniel Wisenhoff
debricked AB
Malmö, Sweden
daniel.wisenhoff@debricked.com

***Abstract*—We demonstrate a tool for identifying, prioritizing and evaluating vulnerabilities in software. The tool aims to improve security in products by making maintenance more efficient and robust. Software components and release versions are matched with vulnerability information from open resources. The results are visualized on several different levels, ranging from product portfolio and individual products, to specific releases and vulnerabilities. The tool keeps track of how security evolves over time in deployed releases, and also how the maintenance organization progresses in evaluating new vulnerabilities. This will result in more efficient, accurate, and robust security analysis and awareness within the organization, and the anticipated long term effect is more secure products.**

***Index Terms*—Security, Vulnerabilities, Demonstration, Recommender system, Software maintenance**

I. Introduction and Background

This paper presents the design of, and motivation behind, a tool that provides a set of dashboards and information about developed and/or maintained products. It can track and present the current security for a product portfolio, a single product, and a single firmware version, and it also provides details on specific vulnerabilities. The overall motivation has been the recent interest in Internet of Things (IoT) devices. The inherent connectivity of IoT devices increases the possibility for attacks, and cybersecurity has become increasingly important. This is recognized by the industry and it is difficult to keep up with the rapid technology development and companies have different ways of organizing the identification and evaluation of vulnerabilities [6]. More specifically, the motivation for the demonstrated tool is based on four main observations. First, there are many new vulnerabilities disclosed every year. The NVD database [5] currently contains over 100'000 vulnerabilities, with information such as vulnerable software and versions (in a standardized format), links to exploits and advisories, and a CVSS severity score [4], [8]. The number of new CVEs has for many years been relatively stable, ranging between approximately 4200 to 7900 between 2005-2016. In 2017 the number increased to about 14700 and halfway through 2018 there were about 10000 so far. This amounts to an average of more than 50 new vulnerabilities each day.

Second, the increasing focus on connected devices, e.g., IoT devices, has resulted in many new manufacturers and integrators of connected products and systems. Also traditional manufacturers, such as car and truck manufacturers, the industry for home appliances (refrigerators, washing machines, etc), and the automation industry is turning towards connectivity. If successful, this paradigm shift will allow more efficient use of resources, sustainable cities and transportation systems, and high quality services at reduced cost [7]. The market competition results in a strive towards shorter time-to-market for new functionality and the development organization typically build the devices and platforms on open source software. This allows for well-tested and maintained code at a very low cost. Additional functionality is then added in-house on top of the open source software.

Third, when new vulnerabilities are discovered, either in configurations (e.g., weak default passwords or cryptographic primitives), in third-party software or in in-house software, the software is typically patched. For third-party software, new software versions can be retrieved from the source (assuming that the code is maintained), be added to the build system, and new firmware or software updates can be made available to customers. In a recent survey [10] based on 2205 users, it was reported that only 14% have ever updated their router's firmware. Some devices or software can be automatically updated, but this is far from always possible. Apart from the fact that this functionality is still lacking in many devices, high availability systems can not afford a reboot or devices may

This work was financially supported in part by Swedish Governmental Agency for Innovation Systems (Vinnova), grant 2016-00603, and in part by the Swedish Foundation for Strategic Research, grant RIT17-0035.

not be connected at all times. Privacy issues may also prevent certain customers from allowing frequent communication with the manufacturer's servers or cloud services. Thus, even though there is new firmware, many customers still use old versions. Since the manufacturer's reputation is at stake when new vulnerabilities are disclosed, there is a need to keep track of all released firmware and understand its exposure to attacks. A related aspect is that the developer's efforts to minimize the number of vulnerabilities can be tracked through different firmware releases. High security awareness can be used as selling points for manufacturers and developers and can be seen as a competitive advantage.

Finally, the large number of new vulnerabilities will require significant effort from the product maintenance organization. Identified potential vulnerabilities must be evaluated and some vulnerabilities will be left with no action while others will require immediate actions. Prioritizing the order in which to evaluate vulnerabilities can reduce the time of exposure, make the evaluation more cost efficient, and improve the accuracy and robustness of the evaluations. The most straight-forward way of prioritizing is the vulnerability severity. The most widely used and recognized severity metric is the CVSS score, provided by NVD. This metric takes into account various aspects of exploitability and impact of the vulnerability, but the metric is designed to be repeatable and not tied to specific products, operating environments or organizations. The metric does have support for stakeholders manually adjusting it to environmental circumstances, but this is done as part of the evaluation and does not help when prioritizing which vulnerabilities to evaluate first.

II. System Overview

The main focus has been to design a tool that can help improve working with vulnerabilities and patching of IoT devices. This is done by presenting products, firmware versions, and vulnerabilities on a set of configurable dashboards. The tool is not limited to specific types of devices, but can be used on standalone applications as well. Features and interface have been designed in close cooperation with companies, providing feedback already at early stages and throughout the development of the tool. It takes two types of inputs, information on vulnerabilities and information about devices and software components. This information is used to present the current security status of devices and to help users and organizations to more efficiently work with vulnerabilities. Output information is provided through a series of views, tables and graphs, focusing on devices, software components and vulnerabilities.

A. Vulnerability Information

Information related to vulnerabilities is primarily taken from the NVD database, which is the most well known and widely used public vulnerability database. The database is provided both in the form of web pages and as XML and JSON feeds. The feeds are very easy to use, but they unfortunately do not include all information. Some information can only be retrieved from scraping the web pages, so this is used as a complement as well. From NVD, we collect e.g.,

- The summary text for the vulnerability.
- The CVSS score, together with the different subscores.
- Relevant links to more information. These links can be used to determine if there are available exploits.
- Which type of weakness that the vulnerability represents, i.e., the CWE [3]. This could be e.g., Buffer Errors (CWE-119), Cross-Site Scripting (CWE-79) or Input Validation (CWE-20).
- The vulnerable software components. These are given as CPEs (Common Platform Enumeration), which is a standardized format for representing vendor, name and version of a software component (or e.g., operating systems).

Other sources of information that are used include e.g., more detailed CWE information, allowing an analyst to better grasp the problem, and information collected from mailing lists discussing software security.

B. Device Information

One main feature is to support tracking of a large number of devices and releases. One manufacturer might have dozens, or even hundreds, of different devices, and each device might receive firmware updates quarterly or monthly. This results in a large number of datapoints as the software modules included, and also the version of the module, will differ between both devices and firmware releases. Vulnerabilities affecting one device will differ between firmware versions since patched components will decrease and newly added components might increase the number of vulnerabilities. The information collected is simply the device, the firmware/release version, and a list of the components that are included in the firmware. This information can be added and maintained either using the graphical interface in the tool, or through an API. Thus, it is possible to track the software with very high level of granularity, e.g., specific builds.

C. Views and Dashboards

The tool supports a variety of views. The views are divided into four main levels. Though there are default information on each view, they are highly customizable on organizational or even user-based level.

Product Portfolio. This presents an overview of the registered products together with summary information on the number of releases and vulnerabilities.

Product Releases. All releases for one specific product are summarized together with the number of vulnerabilities. Note that several releases are typically deployed at the same time since deployed devices might not be updated for each new release. Two different vulnerability metrics are relevant here. First, the number of vulnerabilities at the time of release, and second, the current number of vulnerabilities affecting that release. The first metric will allow organizations to measure their security awareness since this number can to a large extent be controlled by the organization by using up-to-date

software. The second metric will measure the current security of a product. This is more difficult to control, but e.g., hardened devices, device configuration and proper separation of privileges, will affect to which extent vulnerabilities are exploitable. An example of the first metric is given in Fig. 1. Five releases for a product is shown, together with the number of vulnerabilities that have been identified for the respective releases.

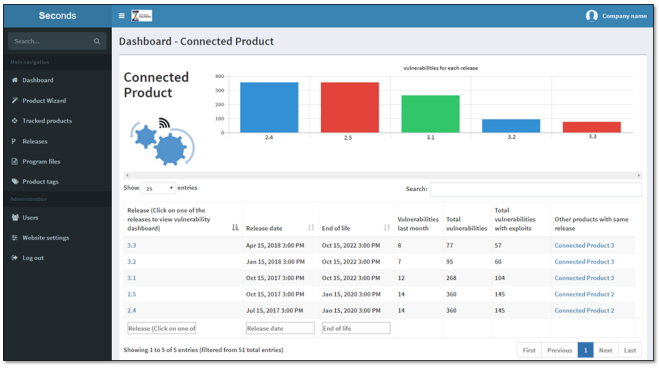

Fig. 1. An example of an overview of the tracked releases for a chosen product.

Release Details. This view will give a summary of all vulnerabilities that affects a given release. The view can show a vulnerability listing , with one vulnerability per row, and a software component listing, with one component on each row together with the number of vulnerabilities.

Vulnerability Details By choosing a vulnerability, details on this vulnerability will be displayed. The information here will largely be based on the information given in NVD, but will also use information from other sources.

D. Evaluation Support

By default, all vulnerabilities that have a matching CPE in the NVD database will be listed, shown as a vulnerability that affects the product/release. However, after evaluation of each vulnerability, it is often concluded that no action is needed since the vulnerability does not affect that particular configuration, or it is not exploitable due to the particular usage of the component. Thus, the tool has support for recording evaluations and labeling vulnerabilities as evaluated, together with the decision made. This will reduce the number of vulnerabilities listed as affecting a device or a release, and can be used both for tracking the security of releases and for the security work within the organization.

E. Summary Reports

The tool is able to generate summary reports on several different levels. Both overview reports for the complete product portfolio, for individual products, and for individual releases. The main purpose of these reports is to generate overview results that can be used in strategic decisions or even marketing. Examples of data generated by the reports include number of vulnerabilities, number of vulnerabilities for products and releases, their severity, how many of them have been evaluated and how many of them have an exploit. This data is also presented based on time of appearance. Additionally, the tool supports creation of detailed reports for individual vulnerabilities.

F. Integration Support

The actual usage of a tool like this will depend on how well integrated it is into already existing processes in an organization. Due to this, it is important to support integration with a wide variety of other third-party tools that are typically used in development or product maintenance organizations. The demonstrated tool supports integration with the BitBake build tool and Jenkins, GitLab CI, and Travis continuous integration tools for automatically detecting which software is used. It also supports Jira, GitLab and GitHub for managing tickets and integrating with project management.

G. API

In addition to using the web based interface for interacting with the tool, it is possible to use an API, both to deliver input, and to extract output. This will allow third parties to integrate parts of the data into their own customized tools and environments, if the provided integration support is not sufficient..

III. Innovative Concepts

Recently, several other organizations have shown interest in integrating vulnerability information into development environments. One notable example is GitHub, which tracks vulnerabilities in dependencies that are found in Ruby gems and NPM packages [2]. Also the NPM package manager [9] has recently added support for a security review of the dependency tree. The demonstrated tool is in contrast not limited to certain languages, such as Ruby and JavaScript, since it will track all software included in a release at build time. Neither is it limited to only matching software with reported vulnerabilities, but instead attempts to help the software maintainer (or security analyst) in prioritizing which vulnerabilities to evaluate first. This is accomplished using ideas similar to recommender systems for e.g., products at e-commerce sites or movies and music recommender systems. Another innovative concept is the tracking of vulnerabilities between different products and product releases. These two concepts are described in more detail below.

A. Recommender System

Recommender systems can be divided into three main variants. *Knowledge-based recommender system* are based on explicit knowledge about users and items. This is saved in a user profile that can be defined and tweaked. Item features are then extracted and matched with the user profile. Another variant is a *content-based recommender system*, in which user preferences are learned by analyzing past user behaviour. Previously viewed items can be used to build a profile for a user. These recommender systems can capture preferred features that the user might not be explicitly aware of. On

the other hand, they suffer from a cold start problem in that it will take some time to build reliable profiles for new users. The third main variant is the *collaborative filtering recommender system*. In this case, items are recommended based on considerations of what other similar users have viewed. This also has a cold-start problem since new items will not have many views by users, so they are not as often recommended. All variants have their pros and cons, and it is possible to combine them into hybrid systems. A recommender can be seen as a scoring system, since each item will receive a certain score depending on how well it matches the user's profile. High scoring items are then recommended. See e.g., [1] for a more thorough overview. In the demonstrated tool, vulnerabilities are recommended. Instead of using only CVSS score, which is designed to be repeatable and not tied to a specific organization or product, our tool uses a wider set of features. It includes e.g., vulnerability type (CWE), date of disclosure, Twitter activity, if the component handles sensitive data, and the number of products that are affected. It also considers availability of exploits, together with the exploit origin. The existence of Metasploit modules should be seen as more severe than proof of concept code. The recommender system used is a hybrid recommender based on knowledge-based and content-based filtering. Due to privacy aspects, collaborative filtering is not used, but might be considered in future work. The knowledge based part takes advantage of the fact that the user (product owner, integrator or manufacturer) has certain product knowledge that a third-party analyst might not have. As an example, the number of affected products (that have e.g., been sold) is only known to the user, and the importance of this metric is probably very user dependent. It can also handle new users very well. The content-based part takes advantage of not requiring users to fine-tune all preferences completely, and can also capture certain user preferences that are difficult to explicitly determine. In addition to knowledge-based and content-based data, the recommender also takes domain knowledge as input, i.e., some portion of the score will be user independent. This guarantee that important vulnerabilities are not missed just because the profile or user history do not correlate well with the vulnerability. The weighting between knowledge-based, content-based, and domain knowledge influence will vary over time. Content-based influence will increase as the amount of knowledge based on user history increases.

B. Release Tracking

The tracking of all products and their releases is an important part of the tool. It is motivated partly by the fact that customers do not immediately apply updates. Particularly in a B2B business model, it is still important to understand the security of all deployed devices, and the threats customers are exposed to. When identifying new critical vulnerabilities in unpatched components, the manufacturer might want to explicitly inform customers of the risks related to not patching. Another motivating aspect of release tracking is to allow the maintenance organization to explicitly report and measure the security work and provide an overview of the product portfolio and its exposure to attacks. This can be used within the organization to improve security work, but it can also be used in marketing situations and in communication with different stakeholders. A third aspect is the manufacturer's support organization, where first and second line support can more easily answer vulnerability related questions, and customers can be given a dedicated information channel for security, and update recommendations, related to their purchased products.

IV. Conclusion

A tool for identifying, prioritizing and evaluating vulnerabilities in software has been outlined and demonstrated. The demonstration consists of showing the different views and how they can help organizations to more efficiently work with security vulnerabilities. It is also demonstrated how the recommender system will help prioritizing between vulnerabilities and how reports can be generated. Since it has been designed with feedback from industry throughout all phases, it has potential to significantly increase the security for developed IoT devices, and to improve how companies throughout the value chain understand and work with security. The tool has recently been commercialized by *debricked AB*. The main reasons for commercialization are that the industry has already shown high interest in the tool, and they have expressed a need for receiving more detailed information and evaluations of vulnerabilities.

It is anticipated that the proposed tool will provide higher reputation for manufacturing and development organizations, as well as increased trust throughout the value chain. It will also lead to more competitive companies and increased innovation speed through more efficient use of open source software.

References

[1] Charu C Aggarwal. *Recommender Systems, The Textbook*. Springer, 2016.

[2] The GitHub Blog. Introducing security alerts on github, 2017-11-16. Available at: https://blog.github.com/2017-11-16-introducing-security-alerts-on-github/.

[3] The Mitre Corporation. Common weakness enumeration (cwe). Online, Last accessed 2018-06-03.

[4] First. Common vulnerability scoring system v3.0: Specification document. Online, Last accessed 2018-06-03.

[5] National Institute for Standards and Technology (NIST). National vulnerability database. Online, Last accessed 2018-06-03.

[6] Martin Höst, Jonathan Sönnerup, Martin Hell, and Thomas Olsson. Industrial practices in security vulnerability management for iot systems – an interview study. In *Proceedings of Software Engineering Research and Practice (SERP)*, pages 61–67, 2018.

[7] Mckinsey Global Institute. The internet of things: Mapping the value beyond the hype, June 2015. Report.

[8] Peter Mell, Karen Scarfone, and Sasha Romanosky. A complete guide to the common vulnerability scoring system version 2.0. In *Published by FIRST-Forum of Incident Response and Security Teams*, volume 1, page 23, 2007.

[9] The npm Blog. 'npm audit': identify and fix insecure dependencies, 2018-05-08. https://blog.npmjs.org/post/173719309445/npm-audit-identify-and-fix-insecure.

[10] Matt Powell. Wi-fi router security knowledge gap putting devices and private data at risk in uk homes. Technical report, 2018. Available at https://www.broadbandgenie.co.uk/blog/20180409-wifi-router-security-survey.