Wanted Hacked or Patched: Bug Bounties for Third Party Open-Source Software Components

Chujiao Ma†
Comcast Cable
Philadelphia, PA, USA
Chujiao_ma@comcast.com

Matthew Bosack
Comcast Cable
Philadelphia, PA, USA
matthew_bosack@comcast.com

Wendy Rothschell
Comcast Cable
Philadelphia, PA, USA
wendy_rothschell@comcast.com

Noopur Davis
Comcast Cable
Philadelphia, PA, USA
noopur_davis@comcast.com

Vaibhav Garg
Comcast Cable
Philadelphia, PA, USA
vaibhav_garg@comcast.com

Introductions

In this article, we propose a targeted open-source bug bounty initiative that offers OSC users a proactive approach towards investigating the security of relevant components by crowdsourcing the discovery of security vulnerabilities to external security researchers. All without breaking the bank. We illustrate the process with a case study of bug bounty for JavaScript OSCs used at Comcast. Overall, we conclude that these bounty programs are a cost-effective and low effort solution to the hidden security risk of OSCs.

1. Overview

In 2022 Synopsys found that over 90% of codebases contain open-source software components (OSCs) [1]. OSCs are crucial to commercial software developments. However, they also contain hidden security risks [5]. The security risks of using OSCs were highlighted by the discovery of the Heartbleed bug in OpenSSL in 2014, which allowed attackers remote access to private keys and passwords. Approximately 24-55% of popular websites using TLS, as well as more than one billion Android devices, were exposed to Heartbleed attacks [2]. Despite the pervasiveness of OpenSSL, the project had no full-time developers, received just $2000 a year, and had no policies for handling vulnerabilities [2].

Heartbleed highlighted some of the unique challenges of securing open-source projects. Maintainers have neither the necessary resources nor the expertise for security analysis or incident response. The security of open-source projects is then contingent on ah-hoc security analysis by third party researchers, who may focus on more popular projects. Thus, a small number of vulnerabilities for a project may simply indicate a lack of security analysis rather than a lack of vulnerabilities.

To address the hidden security risk from OSCs that are used across the cyber ecosystem, the European Union (EU) launched a bug bounty initiative as part of the Free and Open-Source Software Audit (FOSSA) project [3]. In 2018, companies located in EU invested approximately one billion euros in open-source projects [4]. This makes evident the dependency of EU companies on open-source software projects and the importance of their security guarantees. However, EU's effort only addresses OSCs that are broadly popular across the ecosystem. Individual companies may have additional OSCs used in key products that are not addressed by FOSSA.

There is thus a need to address the security of OSCs that are popular within an individual company, but perhaps not across the ecosystem. With an average of 528 OSCs per codebase as of 2020 [1], an exhaustive security analysis of all OSCs used by a company may be fiscally challenging and arguably imprudent. The alternative is to identify areas of concentrated risk, e.g. OSCs with greater number of dependents within the company. Thus, scope the analysis to a smaller set of key components [5]. This smaller set of components can be effectively examined as part of a bug bounty program for third party OSCs. A well designed bug bounty program may expose the targeted components to the broader cybersecurity community leveraging their wide range of expertise. This can provide a more comprehensive examination than internal teams who may have different focus and competing priorities.

However, as these components are neither owned nor maintained by the bounty issuer, the process for setting up a bug bounty program raises additional questions. For example, how should the bounty issuers inform or collaborate with the project owners? Who should be responsible for building a patch? How does the open-source license inform the bug bounty requirements? In this article,

we answer these questions by outlining a process for setting up a bug bounty program for third party OSCs. We illustrate how this process can be operationalized with a case study from Comcast.

2. Background

2.1. Securing Open Source

The security of OSCs can vary widely based on its popularity and security expertise of the maintainers. Also, the same OSC may be implemented in different ways in different commercial applications. These challenges make remediation and assessing security of the OSCs quite difficult for the general users.

One example is Log4j, an open source logging software used by a wide range of applications from Minecraft to Apple iCloud and AWS. The vulnerability in Log4j allows the attacker to remotely control the targeted server, posing a severe risk to millions of consumer products, enterprise software and web applications [5].. However, due to its diverse use, there is no one-size-fits-all solution to patching it. The fix could require a wholesale system update, software update or manual removal of vulnerable code [5].

The different approach for fixes based on deployment highlights one of the differences between securing open-source third party code and commercial applications. Since the company does not own the code, it can be difficult to identify the impact and severity with limited time and cost. Even when a fix is provided by the open source community, the companies will need to identify how the vulnerable OSCs are incorporated into the application to see if the fix can be applied to them and how.

One approach to reduce such issues is to assess the security of an open source project before it is used in a commercial application. The standard method is by examining the associated CVEs. The presence of CVE means that a vulnerability has been disclosed. However, OSCs that haven't been through a security analysis or are not popular enough to be targeted may not have any CVEs despite lack of security. Another way to assess the health of the OSC is by using metrics based on the characteristics of the OSC. Open Source Security Foundation (OSSF) scorecard project scores 1 million most critical open source projects based on security status and code characteristics [6]. OSSF also have a criticality score that, given a project located on Github, will generate a score based on the activity level [7]. The 'npm' package repository also has its own score for all open-source projects in it based on quality, maintenance and popularity [8]. The different scoring systems are all based on different combination of risk indicators or are platform-specific. For companies with a large repertoire of OSCs in different languages, the risk indicator information that can be collected may differ by platform or due to manual entry. A security analysis is still necessary to ensure there are no hidden risks. Since open-source projects used may not fall under anyone's responsibility, a good way to incentivize security analysis is through a bug bounty program.

2.2. Open Source Bug Bounties

Bug bounty programs allow companies to enhance their security by engaging a wider array of security researchers with diverse expertise [9].

Bug bounty programs are also cost effective. The bounty issuer only pays for verified exploits in a bug bounty. The issuer can also set the scope for analysis of different OSCs to focus on the top security concerns. In fact, the average cost of operating a bug bounty program for a year may be less than the cost of hiring two additional software engineers as of 2019 [10]. Research found that contributors are largely motivated by non-monetary factors, so a company is still able to derive utility from bug bounties even if they have a limited budget [11].

Traditional bug bounty programs address first party code. The first security initiative that includes third party code is Google's Project Zero, founded in 2014. The focus of the project are zero-day vulnerabilities in hardware and software systems, both from and outside of Google [12]. The first security initiative specifically for third party open-source code is the software bug bounty by the European Commission from January of 2019. The European Commission started paying out bug bounties for any vulnerabilities found in 14 open-source projects used in European infrastructure [3]. At the time of writing, it had more than 300 vulnerabilities reported during first 2 months and over 90k Euro paid. [13]. The bug bounty has been crucial in discovering a host of bugs within open-source projects, including a 20 year old bug in PuTTY that has been fixed [14]. The latest industry-wide effort was Internet Bug Bounty program in September of 2021 [15]. It focuses on open source projects within the software supply chain that are commonly used across the internet, from rails, Django, Nginx to OpenSSL [16].

In terms of other open-source bug bounties, typically companies will host them for open-source projects written by their own developers. However, in 2020, Google collaborated with Cloud-Native Computing Foundation (CNCF) to launch the Kubernetes bug bounty program, scoped to 'core' Kubernetes. [17]. Google also launched Vulnerability Reward Program (VRP) for third party open source projects with payment upfront to encourage the maintainers to prioritize security work [18]. The latest iteration includes open source bug bounty Google-released open source software (Google OSS). The focus are on security flaws that would significantly impact software supply chain, including vulnerabilities in Google OSS's third-party dependencies [19]. The open source community can also take the matter into their own hands using one of the open source bug bounty platform such as IssueHunt [20], PlugBounty [21], Huntr [22], and BountySource [23].

These open source bug bounties and security initiatives focus on OSCs that are most popularly used across industry, or OSCs that are popular with the open source community. With the large amount of OSCs used within a company, there are many that may be unpopular externally but widely used internally that warrant a security analysis.

3. Open Source Bug Bounty Design

The existing open source initiatives have been successful so far by both providing financial resources to the open source community and identifying hidden vulnerabilities that could have been exploited. Yet, few companies have set up bug bounty programs for third party OSCs. This is in part driven by the lack of guidance on how to set up such a program. If companies would like to secure third party OSCs used in their applications, they face the challenge of how to identify OSCs that are high risk to them while collaborating with the open source community. In this section, we present a four step approach to the bug bounty process for third party OSCs that addresses the unique challenges of handling third party open-source code.

3.1. Pre-Bug Bounty: Tracking

Before planning a bug bounty program, the first task is to take inventory of the OSCs used within the company. An audit of 1,500 commercial codebases found that 85% of them contained open-source dependencies more than four years out of date [1]. This means that while updates and security patches are available, they are not being applied downstream by consumers. This is driven, in part, by the difficulty of tracking the use of open-source software components.

Tracking of OSC usage in first party software can be done with Software Component analysis (SCA) tools or manual inputs. Individual OSCs may also have upstream open-source dependencies.

Short of reverse engineering, customers usually do not have visibility into the open-source dependencies of third party proprietary software offerings. Over 70% of vulnerabilities are found in indirect dependencies. However, 60% of companies surveyed do not have a good view into the full dependency trees of their software, so it's difficult to identify if a newly discovered vulnerability in OSC affects their code or not [24]. One proposed solution is to leverage Software Bill of Materials, which provides an itemized list of all OSCs included in a commercial software product as well as associated information such as version number [25].

Thus, key considerations for the company when tracking OSCs are:

- How to take inventory of the OSCs in an automated fashion with tools to reduce manual entry?
- How far back in the dependency chain should the inventory focus on tracking?

These decisions should be made and integrated into security practices within the company regardless of whether it chooses to approach the security internally or externally. By having a good idea of the inventory of OSCs used within the company, we will have a better idea of the focus for the bug bounty initiative.

3.2. Scoping

The second step of bug bounty planning is to set the scope. A company may use thousands of OSCs in different languages. However, an analysis of OSCs used by Comcast found that about a quarter to a third of the risk for each language are concentrated in the top 100 OSCs [26]. The amount of coverage by the top 100 will vary depending on the set of OSCs used by each company. However, by plotting the dependencies for each component it is possible to determine the area of concentrated risk.

In some cases, it is not feasible to analyze all OSCs used by the company. In order to ensure the best trade-off between risk covered and resources spent, it is possible to narrow down the list of targets by the following criteria:

- **Popularity** – popular OSCs are more likely to attract the attention of the open source community for security analysis while the less popular OSCs are often left unexamined and potentially vulnerable. Thus, the targets should be concentrated on OSCs that are widely used internally but unpopular externally. This can be determined by a relative popularity risk ranking [26].
- **Lines of code** – larger code bases are more likely to contain vulnerabilities since they are often more complex and requires more effort to analyze or fix.
- **Last update** – if the OSC was updated recently, it may be actively maintained while the one updated years ago may be neglected in terms of security.
- **License** – depending on the company policy, some may prefer to focus on OSCs with licenses that don't require open sourcing the downstream products.
- **Usage** – OSCs used by applications that are in production or for critical operations can have widespread impact and should be of high importance.
- **Owner/maintainer support** – if the owner or maintainer is willing to support the bug bounty with patches or updates, it will greatly help with the remediation process.
- **Existing initiatives** – if the OSC has been through another security initiative already, it would be of a lower priority than one that has never been analyzed.

3.3. Set Up the Bug Bounty Program

Setting up a bug bounty program for third party OSCs would require setting up a bug bounty process internally as well as setting up a parallel process for the maintainer.

For the internal process of the company, the components of the bug bounty program include:

- Setting the scope or list of targets of the program.
- Defining the scope of the vulnerabilities accepted. A company may choose to focus on server side vulnerabilities rather than client side vulnerabilities, on code vulnerabilities rather than usage, or on a predetermined class of attacks, e.g. OWASP Top 10.
- Deciding on a bug bounty platform. The company can utilize one of the established platforms for hosting security analysis such as HackerOne [27], Intigriti [28], Bugcrowd [29], and OpenBugBounty [30]. There are also new platforms in recent years focusing specifically on open source bug bounties such as Huntr [22], BountySource [23] and PlugBounty [21].
- Setting the budget and bounty amount. This may differ depending on the cost of the platform, number of targets chosen, and severity level or type of vulnerabilities accepted. To attract the most amount of researchers, it is important for the bounty amount to be competitive.
- Checking if there are any legal or compliance issues with license or contribution of open source code.
- Drafting up a bounty brief. The bounty brief outlines the company's expectation and program details such as the targets, goals, scope, rewards and timeline. It ensures all stakeholders are on the same page before the program details are posted and bug bounty started.

The support from and collaboration with the maintainers is important since disclosure and contribution of the patch may be needed. The outreach to the maintainers should include:

- Gaining support from the maintainers to launch the bug bounty program. The maintainers should be kept in communication throughout the process from initial report, validation, and severity rank to remediation.
- Checking if the maintainers are willing to provide patching support. It is important to collaborate with the owner or maintainers to make sure they agree with the remediation plan, as they may need to come up with their own fixes or incorporating submitted fixes back to the main branch.
- Asking the maintainers to lock the OSC at current version with a security policy posted on it at the start of the bug bounty.

Once the logistics of the bug bounty program has been worked out, the program is ready to be opened to security researchers.

3.4. Verifying Vulnerabilities

Once the bug bounty program is live and vulnerability reports submitted, the next step is to verify the relevance of the vulnerability. Unlike commercial applications, the same OSC can be used in multiple commercial applications for multiple purposes. The issuers of the bug bounty can work with the contributors to ensure the validity of the test cases for verification. This will allow the internal teams to better understand the impact of the vulnerability within the company since the configuration and set up may be different from the testing environment. Once verified, the issuer and contributor can sometimes negotiate a reasonable window of time to allow for remediation before disclosing the vulnerabilities.

Disclosure doesn't always occur, and the disclosure deadlines vary among companies, researchers and organizations. One common recommendation is a 90 days deadline before going public, with a 45 days deadline for vulnerabilities reported to CERT Coordination Center and a 7 day requirement for critical security issues [31]. Due to the public nature of the code, attackers may take advantage of newly discovered vulnerabilities as soon as they are disclosed. To ensure a quick and smooth remediation process, the companies should have a plan in place that allows them to:

- Identify impact such as whether it is affecting sensitive applications and how widespread is the usage of the OSC.
- Identify the severity such as by referencing CVE scores or how difficult is it to exploit the vulnerability based on how the OSCs are used within different applications.

This knowledge allows them to prioritize which assets to secure first if the OSCs are widespread among their applications, and what remediation strategy to pursue.

3.5. Remediation

How to remediate the risk depends on the nature of the exploit and the type of license used for the OSCs. If the new exploit is leveraging an existing vulnerability where the patch might already be available, updating the OSCs used in the application may be sufficient. The vulnerability may also be remediated by a security measure or compensation control already in place, such as tightening the access control.

If the exploit is based on a new vulnerability, companies can check for initiatives in creating a patch within the open-source community. While disclosure of vulnerabilities in open-source projects gave attackers opportunities to exploit them, it also allows other members of open source community to contribute to a patch. When using patches from open-source community, internal teams should perform an analysis on their specific application before implementation. This prevents the update from introducing new vulnerabilities or negatively impact the operations of the application.

If a patch isn't available, then the company will need to create a patch with the internal team in accordance with the company policy and open source licenses. The licenses typically fall into 2 categories: copyleft and permissive. If the developer uses a OSC with copyleft license, then they need to make their product open

source, too. Permissive licenses, however, do not require downstream products to be open source [32]. Since a patch for copyleft license will be open sourced, the internal team need to make sure there's nothing proprietary in the patch. If the patch is for permissive license, then the internal team can choose to use proprietary code and not open sourcing or disclose the patch.

4. Open Source Bug Bounty Case Study

Section 3 described the process of setting up a bug bounty program for third party OSCs. Based on this process, we set up a bug bounty for JavaScript OSCs on the Bugcrowd platform [29].

1. **Set the scope** – the list of top 100 OSCs was ranked based on the relative popularity first. Then, OSCs with less than 100 lines of code and have been updated within the last year were eliminated. Lastly, the internal team reached out to the maintainers to ask for their collaboration. The OSC will not be chosen as a target if the owner or maintainer does not want to be involved. Based on these factors, the list of targets was narrowed down to 4 OSCs for the pilot program.
2. **Set up the bounty** – once the plans and logistics are in place, we then worked with Bugcrowd on the set up for the bug bounty on the 4 OSCs [33]. The vulnerabilities submitted will be categorized and paid according to the level of severity: critical, high, medium, and low. The bug bounty was launched in two phases. It was first launched privately on the platform with invites sent to 200 researchers. After three months, it was opened to the public.
3. **Verification** – to validate submissions, the internal team checked against known open issues for duplicates. The owners or maintainers had the final say on whether or not a submission would be classified as a vulnerability, and should be accepted, rejected, or deemed "won't fix".
4. **Remediation** – remediation could be done by the contributor, internal teams or the maintainer. If none of the options were chosen, then we would investigate the potential of notifying a subset of the community for a potential fix. There was valid concern over fix acceptance testing and making merges to otherwise stable repositories. Fixes needed to be validated to ensure that they were not introducing new vulnerabilities into the code. We continued to work with the OSC owners or maintainers on a solution and over final discretion on whether or not to merge any fixes.

All the submissions for this bug bounty program were made in the first 6 months, and the analysis of the 4 OSCs covers 1.1% of risk in total for JavaScript OSCs used within Comcast. Aside from the cost of the platform, $1,500 was paid out for the 3 verified submissions.

5. Conclusion

Using third party open source projects in commercial application can reduce the total cost of ownership and vendor lock-in [4]. However, it also puts the responsibility of securing them onto the users. By taking a proactive approach to security analysis of OSCs, the company can mitigate the hidden risks before they are realized. Thus minimize the risk from zero day attacks due to undisclosed vulnerabilities. A bug bounty program is a cost effective way to leverage the security research community when individual product teams or maintainers may not have the bandwidth or expertise.

In this article, we provided the bug bounty process for third party OSCs and an example case study on a selected group of JavaScript OSCs. Our results indicated that the open source community is generally very responsive. The list of top OSCs used by the company is unlikely to change drastically from year to year, so the scope of hidden risk will decrease as more bug bounties and security analysis are conducted. Thus, open source bug bounty is an effective way to uncover hidden risk and to give back to the open source community.

REFERENCES

[1] "2021 Open Source Security and Risk Analysis Report," Synopsys, 2021.

[2] J. Walden, "The impact of a major security event on an open source project: The case of OpenSSL," in *Proceedings of the 17th International Conference on Mining Software Repositories*, 2020.

[3] "Europe to Fund Open Source Software Bug Bounty Programme," TechMonitor, 2019.

[4] K. Blind, S. Pätsch, S. Muto, et al., "The Impact of Open Source Software and Hardware on Technological Independence, Competitiveness and Innovation in the EU Economy," European Commission, 2021.

[5] S. Torres-Arias, "What is Log4j? A cybersecurity expert explains the latest internet vulnerability, how bad it is and what's at stake," The Conversation, 22 December 2021. [Online]. Available: https://theconversation.com/what-is-log4j-a-cybersecurity-expert-explains-the-latest-internet-vulnerability-how-bad-it-is-and-whats-at-stake-173896.

[6] "Security Scorecards - Security health metrics for Open Source," OSSF, 2022. [Online]. Available: https://github.com/ossf/scorecard.

[7] "Criticality Score," OSSF, 2021. [Online]. Available: https://github.com/ossf/criticality_score.

[8] "npms - About," [Online]. Available: https://npms.io/about .

[9] A. Sivagnanam, S. Atefi, A. Ayman, J. Grossklags, and A. Laszka, "On the Benefits of Bug Bounty Programs: A Study of Chromium Vulnerabilities," in *Workshop on the Economics of Information Security (WEIS)*, 2021.

[10] T. Walshe and A. Simpson, "An Empirical Study of Bug Bounty Programs," in *IEEE 2nd International Workshop on Intelligent Bug Fixing* , 2020.

[11] K. Sridhar and M. Ng, "Hacking for good: Leveraging HackerOne data to develop an economic model of Bug Bounties," *Journal of Cybersecurity*, vol. 7, no. 1, 2021.

[12] "Project Zero," Google, 2014. [Online]. Available: https://googleprojectzero.blogspot.com/p/about-project-zero.html.

[13] "EU-FOSSA Bug Bounties in Full Force," European Commission, 2019. [Online]. Available: https://ec.europa.eu/info/news/eu-fossa-bug-bounties-full-force-2019-apr-05_en.

[14] "20-year-old open source bug found and fixed under the EU-FOSSA 2 project," European Commission, 2019. [Online]. Available: https://ec.europa.eu/info/news/20-year-old-open-source-bug-found-and-fixed-under-eu-fossa-2-project-2019-dec-11_en.

[15] "The Internet Bug Bounty," HackerOne, 2021. [Online]. Available: https://www.hackerone.com/internet-bug-bounty.

[16] "Internet Bug Bounty Program," HackerOne, [Online]. Available: https://hackerone.com/ibb?type=team.

[17] "Securing open-source: how Google supports the new Kubernetes bug bounty," Google Security Blog, 2020. [Online]. Available: https://security.googleblog.com/2020/01/securing-open-source-how-google.html.

[18] J. Keller, "Announcing updates to our Patch Rewards program in 2020," Google Security Blog, 2019. [Online]. Available: https://security.googleblog.com/2019/12/announcing-updates-to-our-patch-rewards.html.

[19] S. Gatlan, "Announcing Google’s Open Source Software Vulnerability Rewards Program," 30 August 2022. [Online]. Available: https://security.googleblog.com/2023/08/Announcing-Googles-Open-Source-Software-Vulnerability-Rewards-Program%20.html.

[20] "IssueHunt," [Online]. Available: https://issuehunt.io/.

[21] "Plugbounty," [Online]. Available: https://www.plugbounty.com/.

[22] "Huntr," [Online]. Available: https://huntr.dev/.

[23] "BountySource," [Online]. Available: https://bountysource.com/.

[24] Snyk, "State of Open Source Security Report," 2020.

[25] "Software Bill of Materials," [Online]. Available: https://www.ntia.gov/SBOM.

[26] C. Ma and V. Garg, "Hidden Risk of Unpopularity in Open Source," in *Society of Cable Telecommunications Engineers*, 2021.

[27] "HackerOne," [Online]. Available: https://www.hackerone.com/.

[28] "Intigriti," [Online]. Available: https://www.intigriti.com/.

[29] "Bugcrowd," [Online]. Available: https://www.bugcrowd.com/.

[30] "Open Bug Bounty," [Online]. Available: https://www.openbugbounty.org/.

[31] K. T. Hanna, "Vulnerability Disclosure," 2021. [Online]. Available: https://www.techtarget.com/searchsecurity/definition/vulnerability-disclosure.

[32] A. Goldstein, "Open Source Licenses Explained," WhiteSource, 2021. [Online]. Available: https://www.whitesourcesoftware.com/resources/blog/open-source-licenses-explained/.

[33] "Xfinity Opensource," Comcast, [Online]. Available: https://bugcrowd.com/xfinity-opensource.