Osterman Research

# WHITE PAPER

**White Paper** by Osterman Research
Published **August 2021**
Sponsored by **GrammaTech, Inc.**

# Uncovering the Presence of Vulnerable Open-Source Components in Commercial Software

# Executive Summary

Commercial off-the-shelf software often includes open-source software components, but vendors frequently do not disclose details of the presence of such components. Many open-source components contain a range of known vulnerabilities that can be used as egress points for cyberattack. This lack of awareness of open-source components used by organizations in commercial off-the-shelf software increases the security risk, attack surface, and potential for compromise by cybercriminals.

In this white paper, we present the findings of an investigation into the use of open-source components in commercial off-the-shelf software—many of which have a list of known vulnerabilities—across five common software categories. The base data was generated by GrammaTech using its CodeSentry software supply chain security product. CodeSentry uses multiple methods of identifying open-source components used in commercial off-the-shelf software that is delivered in binary form. CodeSentry does not need access to the vendor's source code to complete its analysis of included open-source components.

***Commercial off-the-shelf software often includes open-source software components, but vendors frequently do not disclose details of the presence of such components.***

## KEY TAKEAWAYS

Key takeaways of this research are:

- **The Meetings and Email Client Categories Are the Most Vulnerable**
  Applications in the online meetings and email client categories contained the highest average weighting of vulnerabilities. Given the widespread usage of these tools, organizations should better understand their security attack surface and the potential for compromise.
- **Open-Source Components Widely Used**
  The applications analyzed in this white paper all contained open-source components. On average, 30% of all open-source components contained at least one vulnerability or security flaw that has been assigned a CVE (Common Vulnerabilities and Exposures) identifier.
- **Components with Critical Vulnerabilities Commonly Used**
  All but three of the applications in this study included at least one critical vulnerability with a 10.0 CVSS (Common Vulnerability Scoring System) score—the highest possible. The near-ubiquitous usage of such vulnerable components renders comparisons between applications on this basis meaningless as all applications analyzed are seen as vulnerable.
- **Newer Versions of Components Aren't Always More Secure**
  Several components presented with multiple versions across the analyzed applications, but newer versions of components were not always more secure, either as measured by the number of vulnerable components used or the weighted score of vulnerabilities in each component.
- **Multiple Approaches for Organizations to Use CodeSentry**
  Organizations assessing the security standing of new commercial off-the-shelf software should use CodeSentry to address enterprise risk before choosing and implementing applications. CodeSentry can help organizations rapidly identify open-source components, ascertain the presence of security vulnerabilities, and fast-track penetration testing if necessary. These activities can result in organizations working with vendors to mitigate found issues or rejecting software based on security and risk policies.

### ABOUT THIS WHITE PAPER

This research study was conducted on behalf of GrammaTech. Findings are based on the output of the vulnerability reports produced by GrammaTech CodeSentry. Information about GrammaTech can be found at the end of this white paper.

# Methodology

GrammaTech used its CodeSentry product to look for the presence of open-source components in the binary packaging of widely used software applications. The output reports for each application were supplied in PDF format to Osterman Research. The applications were grouped in five categories which were used in the analysis presented in this white paper:

- Web browsers
- Email clients
- File sharing clients, e.g., for cloud-based storage and file sync
- Online meetings clients
- Messaging clients

Osterman Research collated and correlated the findings and analysis presented for use in this study. This included:

- Transcribing the executive risk scores and security ratings from each output report from CodeSentry into a spreadsheet
- Analyzing the executive risk scores and security ratings to create one of the overall security scores presented in this white paper
- Tabulating the use of all identified open-source components across the analyzed products, including drilling into details on the types of vulnerabilities included in each component
- Calculating averages by category, best and worst scoring applications by category, and other measures to give comparative security scores and enable the creation of the charts shown in this white paper

***GrammaTech used CodeSentry to analyze widely used software applications. Osterman Research studied the output of these reports in five application categories.***

### EXCLUDED APPLICATIONS

Some of the software applications analyzed by CodeSentry were excluded from this study. There were two reasons for doing so:

- One software application in the web browsers category contained no open-source components. The web browser in question was developed by a well-known vendor using proprietary code only, and while the browser does include a long list of publicly disclosed code vulnerabilities, the absence of open-source components in the code base rendered it beyond the scope of this study.
- Other software applications did not fit into any of the five categories above. Each of these software applications belonged to its own category, and since the intent of this white paper was to present category-level findings, they were excluded.

# Assessing Security by Category

We examined reports from CodeSentry for each of the analyzed software applications in five categories. We calculated four security scores across the five categories. In this section, we look at each of these in turn.

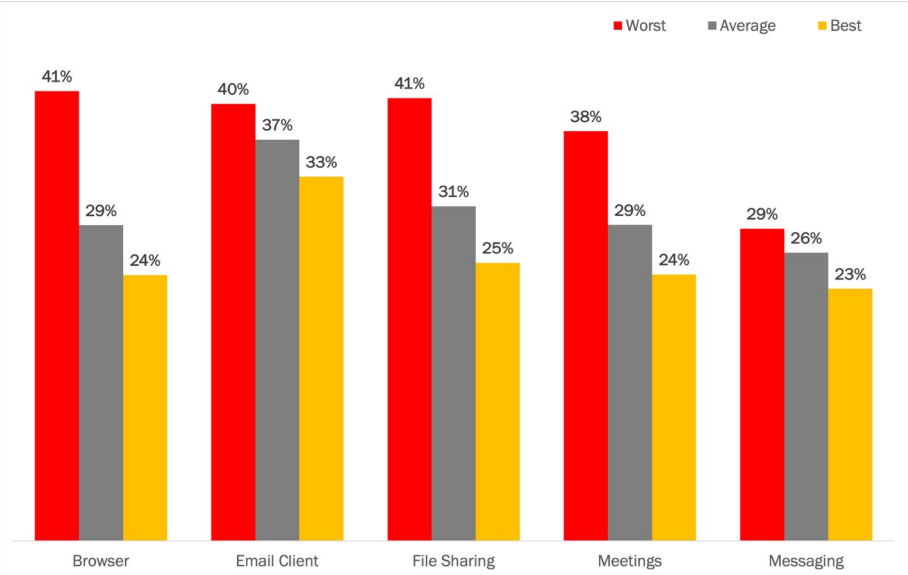
## SCORE 1. BASE SECURITY SCORE

The base security score for each category is the number of open-source components in the software application that include at least one N-Day vulnerability as a percentage of total components. Three values can be calculated, and are shown in Figure 1:

- **Worst Security Score**
  The security score for the worst-performing product in the category. For example, one application in the web browsers category was found to include 41% of open-source components with at least one N-Day vulnerability.
- **Average Security Score**
  The average number of vulnerable components as a percentage of total components for all applications in the category. Browsers, for example, have on average 29% open-source components that include at least one N-Day vulnerability.
- **Best Security Score**
  The security score for the best performing application in the category, or the application with the lowest number of vulnerable components. In the web browsers category, the best performing application uses 24% of components with at least one N-Day vulnerability. The category with the lowest best score is messaging.

***The base security score is the percentage of vulnerable components in the software application.***

**Figure 1**
**Base Security Score of Five Product Categories**
Percentage of vulnerable open-source components




*Source: Osterman Research (2021)*

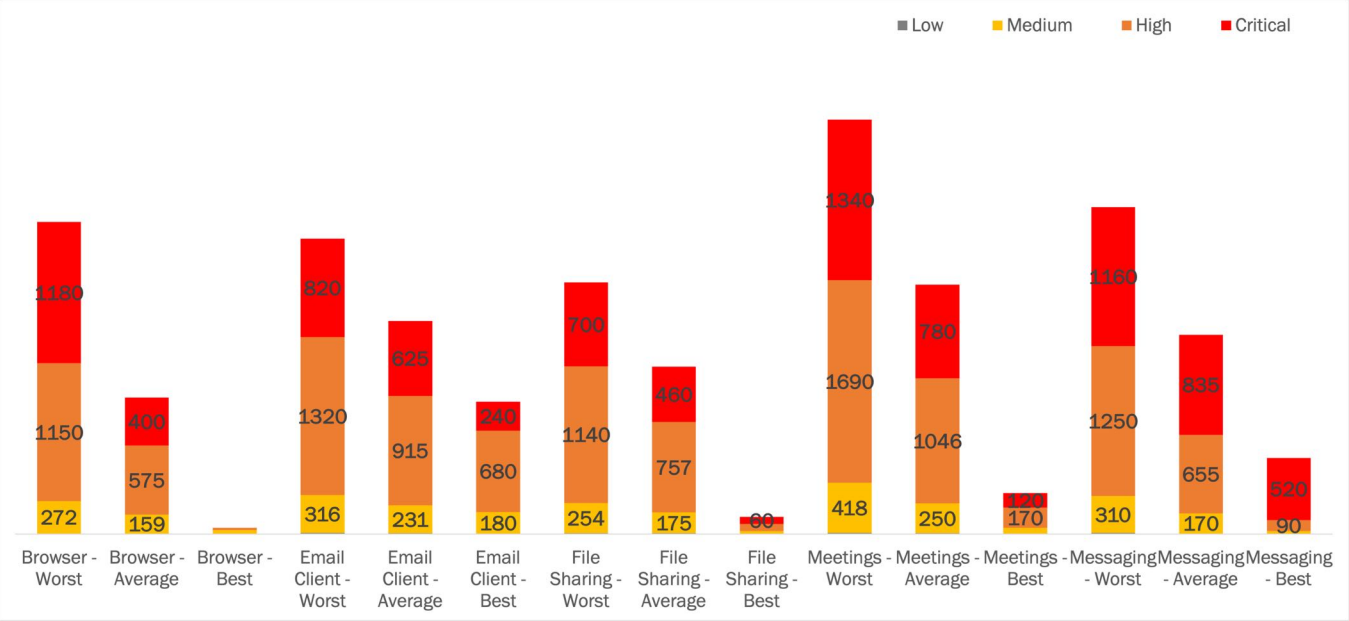
## SCORE 2. WEIGHTED SECURITY SCORE

The number of components that include N-Day vulnerabilities provides only a rough sense of the security of a category (and each application within the category). There is no sense of magnitude portrayed by the base security score because it focuses simply on the count of vulnerable components instead of the composition of vulnerabilities in each component. The weighted security score, by contrast, enumerates the count of vulnerabilities in each component by a weighting to derive a single overall assessment of the severity of the vulnerabilities included in the application. The enumeration is based on:

- The count of vulnerabilities with a **Low CVE** is multiplied by 1.
- The count of vulnerabilities with a **Medium CVE** is multiplied by 2.
- The count of vulnerabilities with a **High CVE** is multiplied by 10.
- The count of vulnerabilities with a **Critical CVE** is multiplied by 20.

The total weighted security score is the sum of these values. Highlighting the critical vulnerabilities is important as the likelihood of attack is much higher with less complexity than can impact confidentiality, availability, and integrity. See Figure 2.

**Figure 2**
**Weighted Security Score of Vulnerabilities in Identified Components**
Weighted value of vulnerabilities (low, medium, high, and critical)




*Source: Osterman Research (2021)*

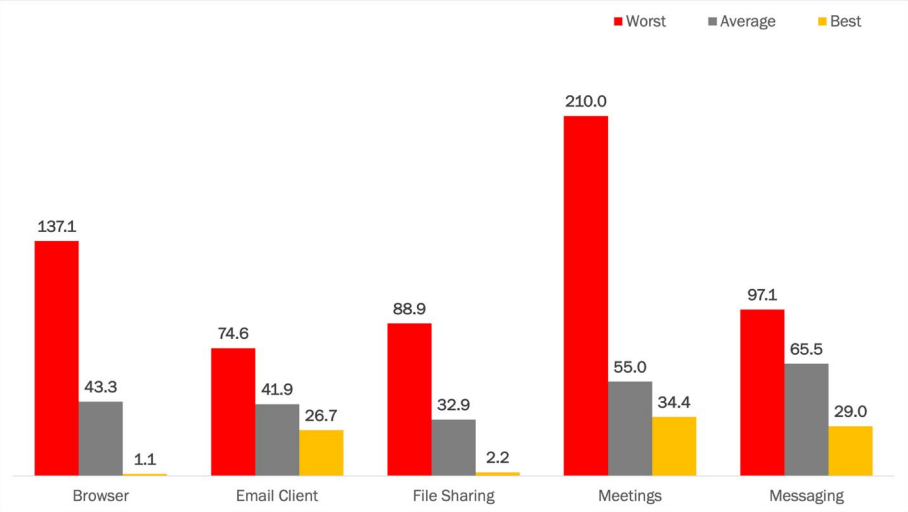
Figure 2 shows that although the total average weighted security score for the meetings category is 2,082 (the sum of the values of the four levels of severity—780 for critical plus 1046 for high plus 250 for medium plus 6 for low), the worst-performing application in the meetings category has a total weighted security score of 3,458. This single application contains a weighted vulnerability that is almost 10 times higher than the best performing application in the same category. The highest variation between the worst and best application in a single category is in the web browsers category, where the worst is 49 times higher than the best.

## SCORE 3. NORMALIZED WEIGHTED SECURITY SCORE

The weighted security score above provides a gross weighted measure of vulnerabilities across all vulnerable open-source components in an application. This raises the assessment problem that applications with differential component compositions can have an equivalent security score. For example, an application with a high number of vulnerabilities in a single component could have the same score as an application with a high number of components, each with a single vulnerability. Similar scores are unhelpful in this context because the driver of insecurity—and by implication the mitigation strategy required—are at polar extremes.

To enable a better basis of comparison between categories, the weighted security score can be normalized to account for the complexity of the application, which is proxied as the total number of identified components in the application. By including the weightings for the high and critical CVEs in each component only, we get the results shown in Figure 3.

**Figure 3**
**Normalized Weighted Security Score**
Weighted value of vulnerabilities (medium/high) per component in the application




*Source: Osterman Research (2021)*

***The weighted security score can be normalized to account for the complexity of the application, which is proxied as the total number of identified components in the application.***

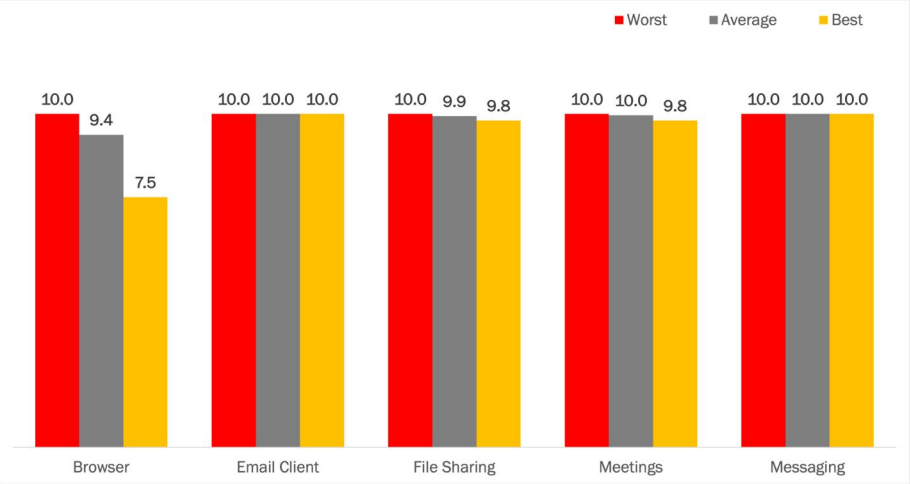
We make the following observations:

- The messaging category has the highest average across the five, and the meetings category the single application with the highest weighted value of vulnerabilities. The messaging category increases in threat level once the data is normalized, in comparison to the previous gross weighted measure.
- The rank order placement of applications within some categories has shifted too. In the email client category, for example, the worst-performing application under the weighted security scoring approach (total value 2465) is the second worst under the normalized approach (normalized score of 48.6). The second worst under the weighted approach is also the worst under the normalized one. This was due to the application with the highest weighted security score using almost twice as many components as the other application.

## SCORE 4. HIGHEST-RATED VULNERABILITY IN A COMPONENT

The fourth security measure we explored—but ultimately discarded—involved looking at the security score of the highest-rated vulnerability (i.e., CVE) across all open-source components in each application. This provides an assessment of the worst-case vulnerability to address when introducing a new application into an organization. Using the CVSS (Common Vulnerability Scoring System) number, we get the results as shown in Figure 4.

**Figure 4**
**Vulnerability Severity per Category**
Component with the highest CVSS score in a product




*Source: Osterman Research (2021)*

In looking at these results, we make the following observations:

- **Almost All Applications Use Highly Vulnerable Components**
  Each of the five categories in this study includes applications that use open-source components with critical vulnerabilities. All applications in the Email Client and Messaging categories use components with at least one critical vulnerability with a CVSS score of 10.0.
- **Minimal Variation**
  There is minimal to no variation across applications and categories. Of all the individual applications in the five categories, only three products were found not to use an open-source component containing a critical vulnerability with a CVSS rating of 10.0.
- **Highest-Rated Vulnerability is a Useless Measure for Comparison**
  The lack of variation across applications and the near-ubiquitous usage of open-source components that contain a critical vulnerability make this measure largely meaningless. Since every application is basically the same in its presentation in this security score, its use as a comparative measure is unnecessary. This does not change the fact, however, that all applications analyzed present serious risk to an organization due to the widespread presence of critical vulnerabilities.

***Highest-rated vulnerability in a component used by an application is a meaningless comparative score due to near-ubiquitous use of open-source components with critical vulnerabilities.***

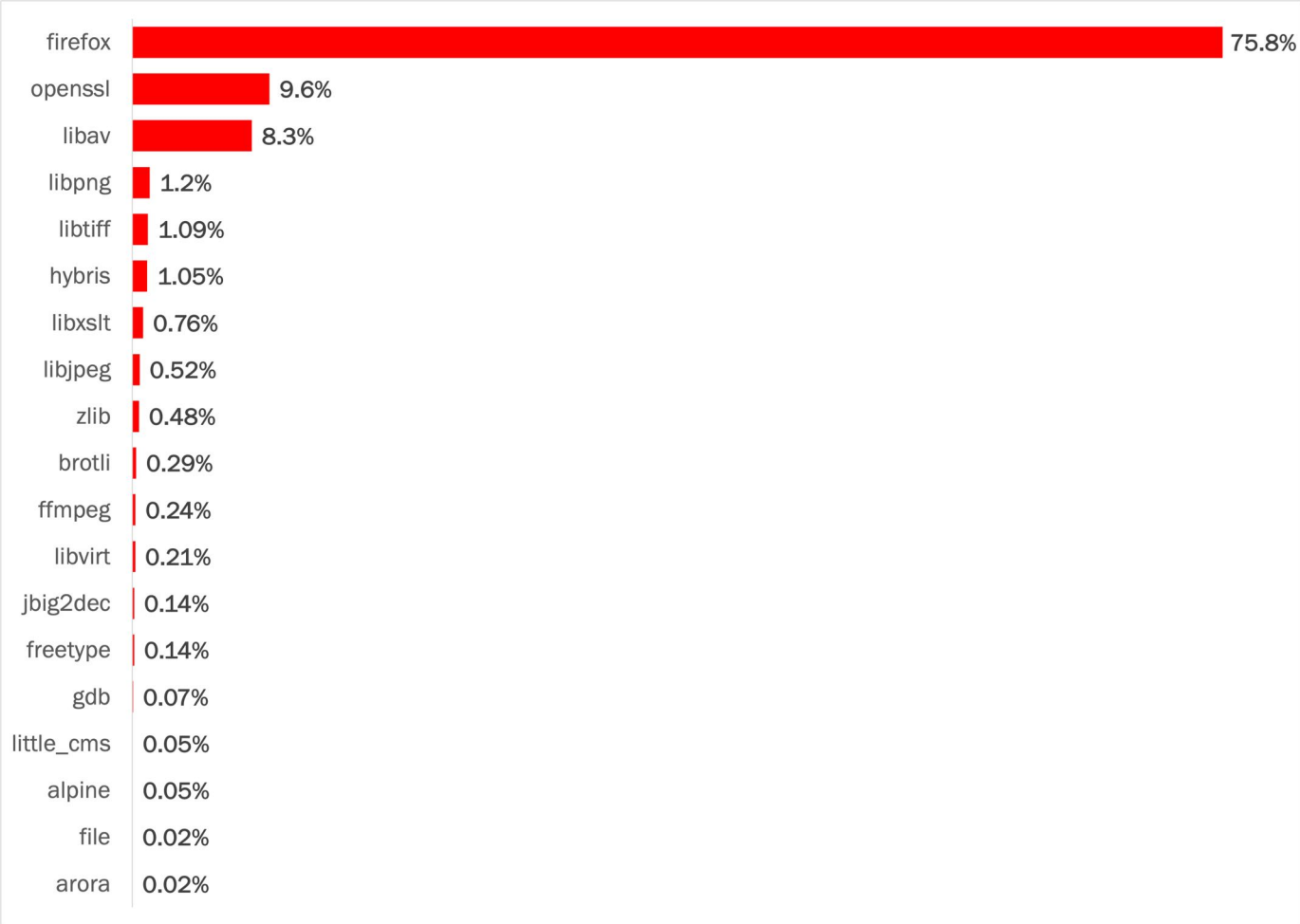
# Common Vulnerable Components

In this section, we look at the distribution of vulnerable open-source components across the applications included in this study, as well as how the pattern of vulnerabilities changes over time when newer versions of a component are released.

## COMPONENTS WITH EXTREME VULNERABILITIES

Of the components identified across the applications analyzed by CodeSentry and used in this study, two versions of the **firefox** open-source component (not the browser itself) contributed 75.8% of the CVEs. In second place, 16 versions of **openssl** had a combined 9.6% of the CVEs, and two versions of **libav** were 8.3% of the CVEs. These numbers are derived by counting the number of vulnerabilities (e.g., CVEs) in each component when a component is used in an application. Multiple instances of the same component in a single application are only counted once. See Figure 5.

**Figure 5**
**Frequency Weighting of Vulnerable Components Identified in Applications**
Count of CVEs in vulnerable components used by the applications in this study

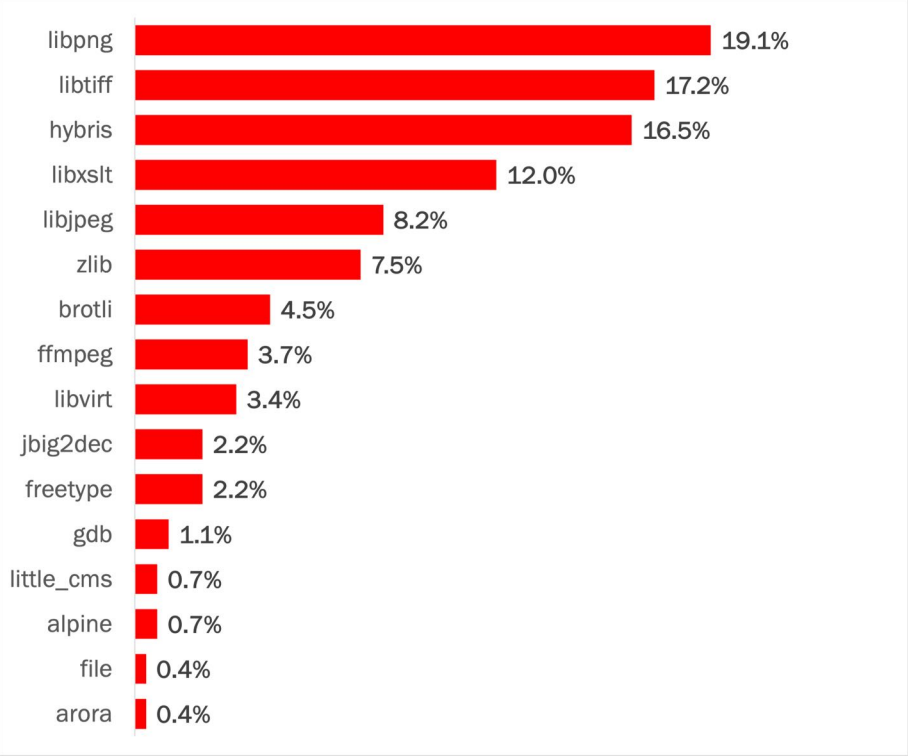



*Source: Osterman Research (2021)*

Based on Figure 5, the immediate conclusion is that urgently addressing the use of versions of the **firefox**, **openssl**, and **libav** open-source components with vulnerabilities would make a significant contribution to decreasing the security risks of using open-source software across the five product categories examined for this report.

However, there is a second and equally valid conclusion: any open-source component that includes a high or critical vulnerability should not be ignored and must be dealt with urgently to reduce risk. The **firefox** open-source component features a disproportionate but concentrated share of the vulnerabilities with high and critical ratings as represented in this study (as do **openssl** and **libav**, but to a lesser extent), while the other components feature a broad and distributed set of risks. Assuming the risks of the top three open-source components are fully addressed—as per the first conclusion above—the risk profile as shown in Figure 6 remains to be addressed. Organizations should urgently address the use of all open-source components that include high and critical vulnerabilities.

**Figure 6**
**Frequency Weighting of Vulnerable Components Identified in Applications—Excluding the Top Three Open-Source Components**
Count of CVEs in vulnerable components used by the applications in this study



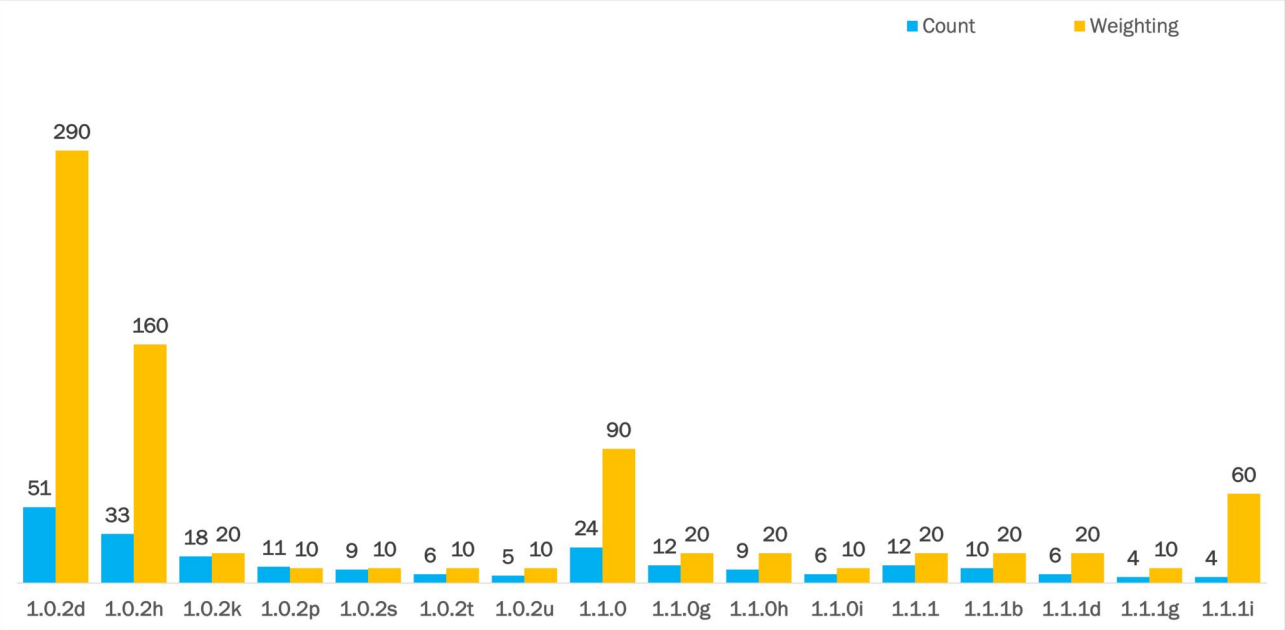

*Source: Osterman Research (2021)*

***Urgently address the use of versions of the firefox, openssl, and libav open-source components with vulnerabilities … but don't stop there.***

## NEWER VERSIONS DO NOT ALWAYS MEAN FEWER VULNERABILITIES

The **openssl** open-source component presented with vulnerabilities in 16 different versions. Version 1.0.2d was the earliest version (51 vulnerabilities), and 1.1.1i was the latest (4 vulnerabilities). While the drop from 51 to 4 vulnerabilities is commendable, new major versions (e.g., 1.1.0 and 1.1.1) have presented with a higher number of vulnerabilities than the stepwise previous version. There has not been a straight-line decrease in newer versions in the number of vulnerabilities nor the weighted value of high and critical vulnerabilities. See Figure 7.

**Figure 7**
**Vulnerabilities in OpenSSL by Version**
Count of vulnerabilities and weighting of high/critical vulnerabilities by version




*Source: Osterman Research (2021)*

Other open-source components also presented with vulnerabilities across multiple versions, and generally there was a decline in the number of vulnerabilities. For example:

- **Firefox** (2 versions). From 254 vulnerabilities in 66.0.3 to 130 vulnerabilities in 75.0.
- **libav** (2 versions). From 45 vulnerabilities in 0.8.1 to 16 in 0.8.17.
- **libxslt** (4 versions). From 14 vulnerabilities in 1.1.26 to 2 vulnerabilities in 1.1.32 and 4 vulnerabilities in 1.1.34.

Open-source software is unlikely to disappear from third-party software, and a straight-line decrease in the number of vulnerabilities is not evidenced from the data collected for this study. We make the following recommendations:

- **Continual optics required**
  Organizations need the ability to continually assess evolving open-source component usage and the fluctuation of vulnerabilities across newer and emerging versions.

***Newer versions of open-source components do not necessarily contain fewer vulnerabilities.***

- **Address the fear of breaking the existing code base**
  Although open-source components are frequently being updated, developers are not always using the most current version, either by release date or the latest version with the fewest vulnerabilities (or the lowest weighted security score for high and critical vulnerabilities). Driving developers to release code on an ever-faster cycle results in the fear of breaking existing code and hence the preference to keep using known but vulnerable components which only extends the presence of vulnerabilities across time.

# Approaches for Using CodeSentry

We see several approaches for organizations to use CodeSentry.

- **Pre-Purchase/Deployment Security Vulnerability Assessment**
  The code composition of commercial off-the-shelf software has traditionally been an unknown quantity to organizations, with the mix and composition of proprietary code and open-source components often not disclosed by the vendor. Organizations can use CodeSentry in the pre-purchase or pre-deployment phase of software evaluation to carry out internal due diligence to ascertain the presence of open-source components in compiled software or the software supply chain. CodeSentry's ability to produce a Software Bill of Materials (SBOM) supporting the CycloneDX export standard to match identified components with vulnerability listings drawn from multiple data sets, including the National Vulnerability Database (NVD), streamlines the classification of applications by vulnerability level.
- **Legacy Software Code Analysis**
  Applications which have been in use for years may contain previously undisclosed open-source components in which may be hidden critical vulnerabilities. With CodeSentry, you can quickly analyze these applications to determine what security risks must be addressed and prioritize a mitigation strategy.
- **Fast-Track Penetration Testing Activities**
  Security professionals carrying out penetration testing activities on commercial off-the-shelf software can fast-track their activities by using CodeSentry to give an initial assessment of embedded vulnerabilities and a roadmap for where to focus efforts. Organizations can therefore optimize where pen testers spend their time.
- **Policy-Based Rejection of Vulnerable Applications**
  By integrating CodeSentry's findings with an organization's security policy engine, vulnerable software applications can be rejected by policy. For example, if an employee attempts to download an email client that contains too many high and critical vulnerabilities, the download session can be halted automatically. As an extension, an organization could then suggest less vulnerable alternatives from the same product line or an alternative product from the same category.

Security teams face the challenge of protecting the organization from the most egregious security threats while equipping employees with the best applications to support secure productivity and collaboration. CodeSentry contributes to this vision, enabling security teams to move from being the team that always says "no" towards a team that is better equipped to give an informed "yes" on a faster cadence.

***CodeSentry gives organizations the ability to rapidly execute security assessments on new applications and fast-track pen testing activities, thus strengthening security posture.***

# Summary and Next Actions

Buying commercial off-the-shelf software applications is not a risk-free proposition. For multiple reasons, vendors surreptitiously use open-source components in their applications, including components that contain significant levels of vulnerability. CodeSentry provides a simple method for organizations to assess source code for the presence of open-source components and identify the number and severity of vulnerabilities in each component. The use of CodeSentry puts security assessments of commercial software on the fast track, enabling organizations to be intentional about managing their security risks and sizing their attack surface as they bring on new applications for the benefit of the business.

# Sponsored by GrammaTech

GrammaTech is a leading global provider of application security testing (AST) solutions used by the world's most security-conscious organizations to detect, measure, analyze and resolve vulnerabilities for software they develop or use.

GrammaTech's solutions include:

- **CodeSentry**
  Quickly perform binary analysis on software applications to identify third-party and open-source components, generate a comprehensive SBOM, detect 0-Day and N-Day vulnerabilities, and get an overall risk score.
- **CodeSonar**
  Seamlessly integrate static application security testing into the DevSecOps process to analyze source and binary code, address security issues early, improve code quality throughout the software development life cycle, and accelerate projects.

GrammaTech is also a trusted cybersecurity and artificial intelligence research partner for the nation's civil, defense, and intelligence agencies. GrammaTech has corporate headquarters in Bethesda, MD, a Research and Development Center in Ithaca, NY, and publishes Shift Left Academy, an educational resource for software developers.

Visit us at www.grammatech.com and follow us on LinkedIn and Twitter.

GRAMMATECH
www.grammatech.com
@grammatech
+1 301 530 2900
sales@grammatech.com