Computest Security

>>

State of Application Security 2024

Technical Write Up of Key Findings

Table of Contents >>

State of Application Security 2024 >>

Background

Computest Security performs several hundred security tests every year on all types of applications. Our specialists have in-depth information and unique insights into the security of applications in the Netherlands and abroad. By analyzing this information and sharing it in our report 'The State of Application Security 2024' we want to create awareness and demonstrate the urgency to actively contribute to strengthening societal security by means of enhancing applications security.

Our research methodology in brief

The basis for this research is the slightly more than 300 application security tests that we carried out during 2023 and in the first quarter of 2024. All these tests have been conducted for various organizations that have requested this for various reasons. For example, as part of their test cycle, because stakeholders or customers required this, or based on (security) standards or regulations. Although these tests took place on different types of applications, they were tested making use of the same method, based on comparable test elements. This means that the different tests can be compared with each other. The findings that were included for purposes of data analytics, were used on anonymized basis. Below we provide insight into the most important results.

Objective and focus of this write up

This document was drawn up as a background to our management summary which contains more generic findings and recommendations, and which is also accessible for a less technically inclined audience. This write up contains some more detailed insights for a more tech oriented crowd and puts more in depth focus on some of the key findings. More specifically we provide some additional statistics and analytics for three of the most prevalent and potentially dangerous issues, being:

1. Unpatched, outdated or unsupported components (69%)
2. Authentication issues (34%)
3. Cross-site scripting risks (32%)

In addition, we also provide a comparison of our findings to the industry acclaimed Top 10 vulnerabilities periodically provided by OWASP.

For the conclusions and recommendations stemming from our State of the Application Security 2024, we recommend to give the management summary a close read. This document only contains relevant backgrounds related to three major vulnerabilities and a comparison with OWASP's top ten findings.

While these statistics might be interesting to the average person, a more tech-savvy reader would likely love to dive deeper into the reasons behind common vulnerabilities. Luckily, we are those people. That is why, for three very common and dangerous problems: outdated software (69.17% of websites), unsupported software (39.13%) and cross-site scripting (32.41%), we dove into the nitty gritty details and related impact to uncover any associated mysteries.

#1 Unpatched, outdated or unsupported components >>

From a quantitative perspective, this is the most frequently observed vulnerability. Nearly 70% of the applications we subjected to our testing activities, ultimately had issues related to unpatched, outdated or unsupported components. Beneath we shed light on some relevant statistics and analytics.

Infrastructure

The security tests included in this research were focused on application(s) and related infrastructure. This infrastructure, which commonly consists of services that are exposed to the internet – in most cases a web server – makes use of outdated software in 15% of the cases. Much of the outdated software that was observed, contains publicly known vulnerabilities. Of these cases, nearly 50% of these vulnerabilities could potentially result in full server takeover, by for example using remote code execution. Taken from this research, the Top 5 of outdated software on infrastructure level is:

1. PHP
2. NGINX and Apache Tomcat
3. Apache HTTP Server
4. OpenSSH
5. IIS

Web-applications

Taken from our findings concerning (web)applications, the outdated software that was observed, mostly concerns client-side JavaScript libraries. We found that over half of the tested applications (54%) make use of outdated libraries. One of the popular JavaScript libraries is jQuery, which in turn amounts to 49% of the outdated libraries found.

Taken from all outdated JavaScript libraries in general, 63% are potentially vulnerable to the risk of cross-site scripting ('XSS'). In these cases, a publicly known cross-site scripting issue has been observed. This does of course not imply that this vulnerability

can actually be exploited within the web application. In some cases, the application has to make use of specific problematic functionality or implement the functionality in a certain way to be directly vulnerable to XSS. However, this finding re-stresses the (basic) importance of actively keeping this software up-to-date. The Top 5 of outdated software on web application level is as follows:

1. jQuery
2. Moment.js
3. jQuery UI (including plugins)
4. Bootstrap
5. CKEditor

End-of-life

The previous section dissected results with respect to outdated and vulnerable software for infrastructure and web applications. This section specifically highlights the risk of end-of-life software. Of the tested applications where we found one or more pieces of unsupported software 39% contained software that was end-of-life. This concerns for example AngularJS, PHP 7, jQuery 1/2/3, Bootstrap 3 and older versions of Nginx and Apache.

> "During our tests, jQuery is the most found End-of-life software"

While 58% of this unsupported software was discontinued over 3 years ago, 45% was even abandoned over six years ago, before 2018. It was even observed that Adobe Flex or YUI 2 platforms were still used which were discontinued over 12 years ago. These outcomes quite clearly underline the necessity of using up to date software to limit obvious security flaws. The Top 5 of unsupported software taken from this research can be listed as follows:

1. jQuery
2. Bootstrap
3. AngularJS
4. Angular
5. PHP

#2 Authentication issues >>

With authentication being second on the list of biggest issues it is something worth mentioning. Authentication vulnerabilities have been encountered in 34% of the application security tests, indicating their significance. The causes vary and can range from simply applications not applying authentication constraints to certain endpoints or pages or as an effect from a bug discovered by a third-party vendor. One of the top issues found is the lack of multi-factor authentication (MFA) for back-end portals as well as front-end portals. Out of all authentication issues found, more than half (56%) do not enforce MFA. Especially for portals that feature a separation between low and high-privileged users this is an important security mitigation to add to web applications.

A common contender where authentication incorrectly is applied is using a URL with a (predictable) token or identifier. Since the URL can get stored locally or on intermediate systems such as a proxy, relying solely on a URL for access is insufficient and should be combined with validating one's session for granting access.

The information that can be extracted by poorly implemented authentication controls differ from application to application, although the primary concern seems to be Personally Identifiable Information (PII). The examples that were seen during tests include social security numbers (BSN), phone number, email and passport scans. In other portals financial information can be seen such as invoice payments or tax information. Finally, it was possible to gain access to some service-specific files such as accessible configuration files from the system or images uploaded to a portal.

> "A multi-factor authentication implementation was missing in 19% of the tests performed"

Causes for common authentication issues

All “Missing authentication” items have been examined for their different causes. As can be observed, these are quite diverse. Provided the importance of understanding the causes of common authentication issues, we have listed those that have been observed:

- Of all projects, 34% have the item that matches “Missing authentication”
- Missing MFA in either the front- or backend login functionality (56% of all missing authentication items, 19% of the total)
- Insufficient authentication (i.e. missing MFA) in the backend login functionality (39% of all missing authentication items)
- No authentication required at all for a specific page or API endpoint (items provide limited information as to why, likely not implemented in the code)
- MFA applied but no rate limiting applied on the OTP validation functionality
- GraphQL does not apply authentication or allows a GraphQL-specific vulnerability: introspection (allows retrieving more data than a user should be allowed to)
- Authentication is implemented with a random token/id/GUID in combination with the path to the file in the URL, but is considered sensitive information in URLs
- JWT authentication token is not validated
- Specific header required to access the resource but can be guessed (e.g. Referer header including the company domain)
- Middleware issues (e.g. authentication is applied but caching results in a resource being accessible)
- Bug in service of third-party vendor software
- Authentication applied but can be obtained in another way (e.g. source code of the page already contains information)

Types of data that could be accessed

As can be derived from the table below, authentication issues may result in exposing confidential information. Apart from more generic information related to organizational matters – that require discretion – it is clear that exploiting authentication issues will in most cases serve for a data privacy breach.

The following are examples of information that could be obtained:

- Passport information
- Tax information
- Email
- PII (e.g. NAW) information
- Personal signature
- Invoice payments
- Phone number
- Server-specific files such as configuration files or images uploaded to a portal
- BSN/social security information

#3 Cross-site scripting risks >>

While modern web frameworks provide building blocks to pre-empt Cross Site Scripting (XSS) vulnerabilities, it can be observed that this vulnerability still prevails in many contemporary applications. This is either caused by the fact that existing mitigations have not been applied properly, or resulting from issues that arise when no framework or custom code is developed as an extension to the framework.

Cross-site scripting was encountered in 32% of the total performed security tests in 2023 and Q1 2024. 15% of these cases can be considered structural, where the platform is littered with vulnerabilities and examining and cleaning the codebase would be the only to resolve these issues. In 59% of the cross-site scripting cases, an attacker could exploit the vulnerability without an existing user account. This obviously reduces the amount of effort required for an attacker to target users and gain access to sensitive data.

> "Cross-site scripting (XSS) is possible in 1 in 3 (web) applications"

Possible cross-site scripting vulnerabilities are diverse. Based upon our research, XSS can be typically divided in three subcategories, that may be at play in parallel (non exclusive):

DOM-based (9,76%) — **Reflective** (39,02%) — **Stored** (75,61%)

Provided the serious risks entailed with XSS we will provide more insight in the manifestation of the three different types below.

DOM-based XSS

One of the issues that can be observed with DOM-based cross-site scripting is the use of unsafe methods within JavaScript that are mis-used for DOM manipulation. User input is the elephant in the room and has to be seen as untrusted. If it is seen as trusted, it could otherwise mistakenly be directly injected into the DOM. For example, using `innerHTML` will directly inject content into the DOM where HTML elements will be directly interpreted by the browser. Instead, it would be better to make use of safe *assignment properties* such as `innerText`, `textContent` or by modifying the *value* of the element.

Frameworks such as AngularJS and React escape HTML by default when used in templates, except when using purposely built-in functions that bypass the protection. For example, React offers dangerouslySetInnerHTML which allows unescaped HTML content.

Reflective XSS

While the web application development landscape has partially moved to client-side frameworks such as React or AngularJS, It appears that a lot of web applications are still using PHP. Even though there is a myriad of frameworks readily available, it can be suggested that PHP web applications have been inherited but and not replaced over the years. Of the total cross-site scripting vulnerabilities that were found, 26% stem from applications built in PHP. The primary issue that arises is that language or framework-specific helper functions that prevent cross-site scripting are not implemented. For example, in PHP, the function `htmlspecialchars()` can be used to escape user input for use in HTML. When using Django, `is_safe` can be used in templates to allow HTML elements to be unescaped. While there are valid use cases for allowing unescaped HTML, one has to know the source of the content. In most cases where a user has influence over the content, this can introduce vulnerabilities.

Stored XSS

Stored is similar to reflective cross-site scripting in a few ways. The user-influenced content can be reflected using JavaScript but can also use the same stack such as the framework of the application to view data on a page, albeit without storing the data. It is of importance that one has to know where data comes from and how it flows through the application. Data that can be inserted as part of a URL is already seen as dangerous,

but data that is stored by other means such as localStorage is now always seen as a threat and trusted by the client-side application.

A few notable examples can be seen observed regarding stored cross-site scripting that are not encountered as compared to the other XSS-subcategories. Of the encountered cross-site scripting vulnerabilities, another 26% are related to file uploads. This happens in an instance where upload functionality is present and a file or document other than intended can be uploaded and subsequentially be retrieved to execute the vulnerability. Aside from restricting file uploads to a certain type by not only the extension but also the MIME type, one also has to enforce the download of the file when accessing it. This can be done, for example, by configuring the `Content-Disposition` header to `attachment`. This will prevent loading potentially malicious content inline the web browser, which would execute the code the web browser's context. While a vector graphic element (SVG) can be frequently used to embed JavaScript code in the XML document, there have also been instances where modified PDF files or spreadsheet documents embedding HTML elements can be used for this type of attack.

Another important aspect that must be noted is that cross-site scripting can be caused by the incorrect usage of the Content-Type header. Many web applications make use of an API which serves data in JSON. In some cases, the Content-Type header of these API responses is not set to `application/json` but to, for example, `text/html`. If content is reflected in the JSON response it is not an issue per se. However, when the request can be visited in a browser and the JSON contains HTML elements, it is interpreted resulting in cross-site scripting.

Taken from the observed total XSS cases, this latter – Stored XSS-variant – is most prevalent (75% of the cases). We've observed the Reflective XSS variant in 39% of the cases and DOM based XSS in 10% of the cases. The reason these numbers do not add up to 100% is because platforms deemed vulnerable to more than one type at the same time.

OWASP Top 10 Comparison >>

As you will probably know, there is a global initiative called the 'Open Web Application Security Project' (OWASP). This is a vast network of specialists involved in promoting application security. The 'Top 10' of vulnerabilities that OWASP periodically publishes is a household name and serves as a point of reference for many involved in application security. The OWASP list is created by researching incidents that have occurred, examining the results of pen tests carried out, and sending a questionnaire to the partners and specialists involved. For that reason, we also use OWASP as an important professional reference for the tests we perform for our clients.

Based on the research results, we are able to make a comparison between 'The State of Application Security' and the 'Top 10' published by OWASP (this is published every four years). The ranking of the vulnerabilities in the OWASP list is based, among other things, on the frequency, impact, and how easily the vulnerability can be exploited. It is important to consider that in our research we looked at frequency and impact (the latter based on the CVSS scoring).

You will find the consolidated results of this comparison in a table projected on the next page.

#	OWASP TOP 10 2021	COMPUTEST SECURITY TOP 10 2024	Difference
1	A01: Broken Access Control	A05: Security Misconfiguration	+ 4
2	A02: Cryptographic Failures	A07: Identification and Authentication Failures	+ 5
3	A03: Injection	A02: Cryptographic Failures	- 1
4	A04: Insecure Design	A06: Vulnerable and Outdated Components	+ 2
5	A05: Security Misconfiguration	A08: Software and Data Integrity Failures	+ 3
6	A06: Vulnerable and Outdated Components	A04: Insecure Design	- 2
7	A07: Identification and Authentication Failures	A01: Broken Access Control	- 6
8	A08: Software and Data Integrity Failures	A09: Security Logging and Monitoring Failures	+ 1
9	A09: Security Logging and Monitoring Failures	A03: Injection	- 6
10	A10: Server-Side Request Forgery	A10: Server-Side Request Forgery	=

It is interesting to determine how the results of the Computest Security specialists differ from those of OWASP. The ‘Top 10’ of Computest Security has a different ranking than the OWASP top 10, except for vulnerability number 10. Security Misconfiguration ranks our Top 10. This vulnerability consists of misconfigurations in either the system services (such as a web server) or the application itself (e.g. incorrectly configured security headers). This vulnerability is followed by identification and authentication (as was described extensively above). The cryptographic failures can lie within the fact that an older protocol is used (TLS 1.0, TLS 1.1 are still seen often) in combination with weak encryption algorithms such as 3DES. As was highlighted above as well, we also observe vulnerabilities related to vulnerable and outdated components being on the increase. Since we perceive XSS as a vulnerability that overarches multiple categories – not only ‘Injection’ – we make no further reference to XSS here.

Computest Security

Any questions about the State of Application Security 2024?

Contact us:

info@computest.nl

+31(0)88 733 1337

Independent. Security. Partner. >>