**System software and applications not updated**

**SEVERITY: MODERATE HIGH**

**VULN. CODE:** SSAP-NU

**AUTH. REQUIRED:**

**VULNERABILITY DESCRIPTION**

Keeping the system and application software up-to date is a fundamental practice in the management of the systems and the applications in order to keep a high standard of corporate security. The missing updates or the missing security patches and/or workarounds, grants the exposure for a long period of time of the systems and the applications to public domains vulnerabilities and it highly reduces the effort that a generic threat agent has to make to carry out a successful attack. During the course of the analysis activities, we found out that some system components and/or application ones are not updated to the latest version made available by the vendor. We highlight that the following vulnerabilities could be exploited from a threat agent in order to carry out a successful attack with the aim to either access the information kept in the system, either to do a denial of service or to compromise the systems.

The following outdated and vulnerable software applications were detected:

* nginx   
  Version: 1.14.2  
  Host: 10.10.10.197  
  Severity: HIGH ( CVE-2019-9516 )  
    
  CVE: CVE-2019-20372, CVE-2019-9516, CVE-2019-9513, CVE-2019-9511, CVE-2018-16843, CVE-2018-16844, CVE-2018-16845  
    
  The next Box shows the evidence of the software component version in the response header.

|  |
| --- |
| Request to 10.10.10.197:80 HTTP/1.1 301 Moved Permanently Server: nginx/1.14.2 Date: Mon, 23 Nov 2020 18 Content-Type: text/html Content-Length: 185 Connection: keep-alive Location: http |

**Box 1 – Leak software version: not updated**

* nginx   
  Version: 1.14.2  
  Host: 10.10.10.197  
  Severity: HIGH ( CVE-2019-9516 )  
    
  CVE: CVE-2019-20372, CVE-2019-9516, CVE-2019-9513, CVE-2019-9511, CVE-2018-16843, CVE-2018-16844, CVE-2018-16845  
    
  The next Box shows the evidence of the software component version in the response header.

|  |
| --- |
| Request to 10.10.10.197:80 HTTP/1.1 301 Moved Permanently Server: nginx/1.14.2 Date: Mon, 23 Nov 2020 18 Content-Type: text/html Content-Length: 185 Connection: keep-alive Location: http |

**Box 1 – Leak software version: not updated**

**SUGGESTED SOLUTIONS**

It is suggested to first adopt countermeasures on the perimeter defense systems in order to mitigate the exploitation of the vulnerabilities carried by a generic threat agent by blocking the requests specifically made to exploit the vulnerabilities or by neutralizing the values supplied to the vulnerable application components. After applying the suggested workaround and after checking out the release notes from the vendors, it is suggested to update the system software and the application software that provide the services to the latest version available.

**REFERENCES**

More information regarding the vulnerability and its possible solutions can be found on the following addresses:

**REF 1 – https://www.cvedetails.com/**

**Transport layer without the use of encryption**

**SEVERITY:** **MODERATE**

**VULN. CODE:** NOCRYPTT

**AUTH. REQUIRED:** NO

**VULNERABILITY DESCRIPTION**

Insufficient protection of the transport layer exposes communications to an attacker, allowing an attack vector to compromise a web application and/or extract sensitive information. In particular, unencrypted connections allow interception, injection and redirection (also known as man in the middle/MiTM attacks). An attacker may passively intercept communication to obtain information about the systems they are communicating and may be able to actively inject/remove content from the communication to forge or omit information, insert malicious scripting, or redirect the client to untrusted content. Similarly, the attacker will be able to redirect the communication so that the web application and the client are no longer able to exchange information directly, but through the attacker, and believing they are communicating in a trusted way.

Responses coming from HTTP service without using any TLS/SSL protocol have been logged.

**SUGGESTED SOLUTIONS**

It is recommended to implement encryption mechanisms to protect all communications between the client and the server after the authentication process. If encryption mechanisms cannot be enabled to protect all communications after authentication, it is recommended to protect at least communications related to sensitive processes. Communications to be protected include all communications related to the authentication process and its related functionalities, functionalities where sensitive data is manipulated for the application or which allow the execution of operations of an administrative nature. In particular, it is recommended, first of all, to disable the data transfer mode via HTTP, and secondly to consider the use of an encrypted exchange protocol, such as HTTPS through the correct configuration of SSL certificates.

**REFERENCES**

More information regarding the vulnerability and its possible solutions can be found on the following addresses:

**REF 20 – www.owasp.org/index.php/OWASP\_Application\_Security\_FAQ**

**REF 21 – cwe.mitre.org/data/definitions/311.html**

**REF 22 – cwe.mitre.org/data/definitions/523.html**

**REF 23 – cwe.mitre.org/data/definitions/319.html**

**REF 24 – projects.webappsec.org/w/page/13246945/Insufficient Transport Layer Protection**

**HTTP Response Headers**

**SEVERITY:** **MODERATE**

**VULN. CODE:** RES-H

**AUTH. REQUIRED:**

**VULNERABILITY DESCRIPTION**

Modern web applications require modern defense and reinforcement systems. With the progress of the sophistication of web based applications, user browsers have also become a major component in mitigating client-side prosecution attacks. The HTTP protocol (Hyper Text Transfer Protocol) is a protocol without nature of the state control. Currently, with the development of web technologies, the HTTP protocol, which has become the backbone of the entire World Wide Web, has evolved to the point of being able to manage and maintain the state of the sessions of multiple contemporary users who use the same web application. This is made possible mainly through the implementation of session controls via cookies. The multitude of attacks that can be prosecuted via HTTP can lead to exploit the exchange of important information between the user's browser and the web server on every single request and response. Therefore, the setting and strengthening of HTTP headers, on the web server side, is a good basis for preventing and mitigating client-side attacks (i.e. aimed at end users' browsers).

During the analysis, it was detected that the web application responds to client HTTP requests without implementing useful controls within the protocol header. In particular, within the HTTP RESPONSE coming from the analyzed web application, the following HTTP headers are not configured:

Strict-Transport-Security: max-age=31536000; includeSubDomains

The Strict-Transport-Security HTTP header (HSTS) strengthens secure SSL / TLS connections. This setting, on the web server side, reduces the impact that any bugs present in the applications make possible the loss of session information, through cookies and external links, eliminating any possibility of passing in clear on the network of such information. The transport in cleartext on the network can facilitate Man-in-the-middle attacks, giving rise to attacks aimed at the theft of the session and the interception of the contents. The HSTS header is also set to inhibit the ability of the user-browser side to ignore the warnings arising during the negotiation phase of the SSL / TLS certificates, thus rejecting the connection in case of errors or inconsistencies on the certificates. The parameter max-age is the time in seconds for which the enforcement policy on the user's browser will be valid.

X-XSS-Protection: 1; mode=block

This HTTP header, set on the web server side, forces all Cross-Site Scripting (XSS) filters to be enabled on modern web browsers, even if the user had on purpose disabled these controls.

X-Content-Type-Options: nosniff

This header is particularly important to instruct modern browsers in order to prevent the interception of MIME contents. This web server-side setting reduces exposure to attacks carried by requests for downloading and uploading content that dynamic HTML pages and executables can pursue without the knowledge of the end user.

Content-Security-Policy: script-src 'self'

The Content-Security-Policy header extends the Same Origin Policy rule, introducing a Whitelist concept allowing to specify precise rules on the domains and Cross-scripting sites that are accepted. With this system you can individually specify accepted urls for iframes, for image css scripts and much more. So the header Content-Security-Policy must be associated with a parameter through which to indicate the URL of the CDN (Content Delivery Network) authorized to make available its libraries. Through the proposed header the only accepted scripts will be those available through the current (self) domain. Furthermore, the directive can be specified for:

* script-src: JavaScript code;
* connect-src: XMLHttpRequest, WebSockets e EventSource;
* font-src: fonts;
* frame-src: frame;
* img-src: images;
* media-src: audio/video
* object-src: multimedia plugins like Adobe Flash;
* style-src: CSS sheets.

|  |
| --- |
| Request to 10.10.10.197:80 HTTP/1.1 301 Moved Permanently Server: nginx/1.14.2 Date: Mon, 23 Nov 2020 18 Content-Type: text/html Content-Length: 185 Connection: keep-alive Location: http |

**Box 1 – Missing HTTP Response Headers**

**SUGGESTED SOLUTIONS**

We recommend, where possible, to implement these settings on the web server side, in order to prevent a large class of attacks directed at the user client, that is targeted attacks on end users who use the web application at a given moment. It is advisable to remove the "X-Powered-By:" information header from which information about the software components in question can be easily obtained. Alternatively, it is possible to specify a fictitious value that has no specific reference and that can not external threat to installed software components.

After the modification and restart of the WEB server, all the headers with the set values ​​will be contained within all the responses (RESPONSE HTTP) by the server. These headers will be interpreted by the end user's browser that will automatically prevent the user's browser from operations that could be automatically recalled without the user's knowledge from pages properly constructed for fraud and / or other malicious uses.

|  |
| --- |
| X-Content-Type-Options: nosniff  Content-Security-Policy: default-src 'self'; script-src 'self'  X-Frame-Options: DENY  X-XSS-Protection: 1; mode=block  Strict-Transport-Security: max-age=31536000; includeSubDomains |

**REFERENCES**

More information regarding the vulnerability and its possible solutions can be found on the following addresses:

**REF 1 – www.owasp.org/index.php/List\_of\_useful\_HTTP\_headers**

**REF 2 – securityheaders.com/**

**REF 3 – blogs.msdn.com/b/ie/archive/2008/09/02/ie8-security-part-vi-beta-2-update.aspx**

**REF 4 – blogs.msdn.com/b/ie/archive/2008/07/02/ie8-security-part-iv-the-xss-filter.aspx**

**REF 5 – tools.ietf.org/html/rfc6797**

**REF 6 – hackertarget.com/http-header-security-analysis/**

**REF 7 – www.html5rocks.com/en/tutorials/security/content-security-policy/**

**Information Leak**

**SEVERITY: LOW**

**VULN. CODE:** IN-LEAK

**AUTH. REQUIRED:**

**VULNERABILITY DESCRIPTION**

During the course of the activities, we found out that there are some sources of information leak. In the production servers, there should not be any disclosure of sensible information. Particularly, as suggested by best practices, custom pages and information about the usernames or passwords should be omitted. This type of vulnerability could allow a threat agent to obtain information on the technologies and logic being used by the application and the system by observing the verbose error messages.

During the analysis phase, header fields were detected that show the name of the software used and the relative versions.

|  |
| --- |
| Request to 10.10.10.197:80 HTTP/1.1 301 Moved Permanently Server: nginx/1.14.2 Date: Mon, 23 Nov 2020 18 Content-Type: text/html Content-Length: 185 Connection: keep-alive Location: http |

**SUGGESTED SOLUTIONS**

It is advisable to remove any source of information leaks, limiting the verbosity of information, or set up access controls on web paths and applications, limiting the disclosure of information to unauthorized personnel.

The fields in the response headers that often represent the subject of information leaks are “X-Powered-By”, “X-AspNet-Version”, “Server”, etc. They are not necessary for the proper functioning of the application, nor are they useful for users who connect to them correctly, but they are an excellent source of information for a threat agent regarding the composition of the system. For this reason, they should be disabled in any case.

**REFERENCES**

More information regarding the vulnerability and its possible solutions can be found on the following addresses:

**REF 37 – projects.webappsec.org/Information-Leakage**

**REF 38 – cwe.mitre.org/data/definitions/200.html**