**Vulnerability Assessment Report**

**For**



**Example Company**

**Date March 30, 2021**

**Document Security Level:** Confidential

**Document Version:** 1.0

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Edit Report** | **Editor** |
| 1.0 | October 13, 2021 | Creation | INET Managed Services |

|  |  |  |  |
| --- | --- | --- | --- |
| Example Company Limited. | | INET Managed Services CO., LTD. | |
| Name |  | **Name** |  |
| Position |  | **Position** |  |
| Tel |  | **Tel** |  |
| Signature |  | **Signature** |  |

**Table of Contents**

[1. Restrictions on disclosure and use of information 4](#_heading=h.gjdgxs)

[2. Operation Method 5](#_heading=h.30j0zll)

[3. Project Scope 6](#_heading=h.1fob9te)

[3.1 Infrastructure Vulnerability Assessment 6](#_heading=h.3znysh7)

[3.2 Web Application Vulnerability Assessment 6](#_heading=h.tyjcwt)

[4. Testing Tools 7](#_heading=h.3dy6vkm)

[5. Infrastructure Vulnerability Assessment 7](#_heading=h.4d34og8)

[5.1 Target Information 8](#_heading=h.2s8eyo1)

[5.2 Executive summary 9](#_heading=h.3rdcrjn)

[5.2.1 Summary Vulnerability by Severity 9](#_heading=h.26in1rg)

[5.2.2 Vulnerability by Target 9](#_heading=h.35nkun2)

[5.3 Infrastructure Vulnerability Detail 10](#_heading=h.1ksv4uv)

[6. Web Application Vulnerability Assessment 14](#_heading=h.44sinio)

[6.1 Target Information 15](#_heading=h.2jxsxqh)

[6.2 Executive summary 16](#_heading=h.z337ya)

[6.2.1 Summary Vulnerability by Severity 16](#_heading=h.3j2qqm3)

[6.2.2 Vulnerability by Target 16](#_heading=h.1y810tw)

[6.3 Web Application Vulnerability Detail 17](#_heading=h.4i7ojhp)

[7. Port Discovery 20](#_heading=h.2xcytpi)

[8. Appendix 21](#_heading=h.1ci93xb)

[8.1 About Nessus 21](#_heading=h.3whwml4)

[8.1.1 Nessus vulnerabilities 21](#_heading=h.2bn6wsx)

[8.1.2 Nessus risk score 21](#_heading=h.qsh70q)

[8.2 About Acunetix 23](#_heading=h.3as4poj)

[8.2.1 Acunetix web vulnerabilities 23](#_heading=h.1pxezwc)

[8.2.2 Acunetix risk score 23](#_heading=h.49x2ik5)

# Restrictions on disclosure and use of information

Restriction on Disclosure and Use of Confidential Information. The Executive understands and agrees that the Confidential Information constitutes an asset of the Company and its affiliated entities and may not be converted to the Executive's own use. Accordingly, the Executive hereby agrees that the Executive shall not, directly, or indirectly, at any time, reveal, divulge, or disclose to any Person not expressly authorized by the Company any Confidential Information, and the Executive shall not, directly, or indirectly, use or make use of any Confidential Information in connection with any business activity other than that of the Company. The parties acknowledge and agree that this Agreement is not intended to, and does not, alter either the Company's rights or the Executive's obligations under any state or federal statutory or common law regarding trade secrets and unfair trade.

# Operation Method

* 1. Posture Review
  2. Information Gathering
  3. Enumeration
  4. Vulnerability Assessment
  5. Analyze & Evaluate Risk Value
  6. Report



Figure 1: Operation Method

# Project Scope

## **3.1 Infrastructure Vulnerability Assessment**

**Target / IP Address:**

| **No.** | **Domain / Server Name** | **Public IP Address** | **Private IP Address** | **OS/Model** | **Functions** | **Public Assessment** | **Private Assessment** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | DATABASE01 | - | 172.16.69.13 | Ubuntu 18 | Database Server 01 |  | ✓ |
| 2 | WEB01 | 123.123.123.123 | 172.16.69.14 | Ubuntu 20 | Web Server |  | ✓ |
| 3 | TERM | - | 172.16.69.52 | Windows Server 2016 | Terminal Server |  | ✓ |
| 4 | SMB01 | 12.12.12.12 | 172.16.69.53 | Windows Server 2019 | SMB Server |  | ✓ |
| 5 | DATABASE02 | - | 172.16.69.54 | Ubuntu 18 | Database Server 02 |  | ✓ |

## **3.2 Web Application Vulnerability Assessment**

**Target / IP Address:**

| **No.** | **Domain / Server Name** | **Public IP Address** | **Private IP Address** | **OS/Model** | **Functions** | **Public Assessment** | **Private Assessment** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | https://example.com/ | 123.123.123.123 | 172.16.69.14 | Ubuntu 20 | เว็บไซต์ขายของ | ✓ |  |

# Testing Tools

|  |  |
| --- | --- |
| **Tool Name** | **Testing Type** |
| Nmap | Host and Service Discovery |
| Nessus | Infrastructure Vulnerability Assessment |
| Acunetix | Web Application Vulnerability Assessment |

# Infrastructure Vulnerability Assessment

**Vulnerability Assessment from Public Access (for public target)**

**Testing date:** March 30, 2021

**Tester IP Address:** 203.150.110.29

Diagram

Description automatically generated

Figure 2: Vulnerability Assessment from Public Access

**Vulnerability Assessment from Private Access (for private or restricted access target)**

**Testing date:** March 30, 2021

**Tester IP Address:** Private IP from VPN access

A picture containing diagram

Description automatically generated

Figure 3: Vulnerability Assessment from Private Access

## **5.1 Target Information**

| **No.** |  | **IP Address** | **OS/Model** | **Port** |
| --- | --- | --- | --- | --- |

## **5.2 Executive summary**

The purpose of this activity is to find the vulnerability on the target infrastructure.

### **5.2.1 Summary Vulnerability by Severity**

Figure 4: Summary by Severity of Infrastructure Vulnerability Assessment

### **5.2.2 Vulnerability by Target**

| **No.** | **Domain/Server Name** | **IP Address** | **Critical** | **High** | **Medium** | **Low** | **Total** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | DATABASE01 | 172.16.69.13 | 0 | 1 | 7 | 2 | 10 |
| 2 | WEB01 | 172.16.69.14 | 0 | 1 | 7 | 2 | 10 |
| 3 | TERM | 172.16.69.52 | 0 | 1 | 8 | 2 | 11 |
| 4 | SMB01 | 172.16.69.53 | 0 | 1 | 7 | 1 | 9 |
| 5 | DATABASE02 | 172.16.69.54 | 0 | 1 | 7 | 1 | 9 |
| **Total** | | | **0** | **5** | **36** | **8** | **49** |

## **5.3 Infrastructure Vulnerability Detail**



# Web Application Vulnerability Assessment

**Vulnerability Assessment from Public Access (for public target)**

**Testing date:** March 30, 2021

**Tester IP Address:** 203.150.79.252

Diagram

Description automatically generated

Figure 5: Vulnerability Assessment from Public Access

## **6.1 Target Information**

| **No.** | **Domain / Server Name** | **IP Address** | **OS/Model** | **Port** |
| --- | --- | --- | --- | --- |
| 1 | https://example.com | 123.123.123.123 | Ubuntu 20 | TCP 22, 53, 80, 113, 123, 443, 2000, 4118, 4119, 4120, 4121, 4122, 4444, 5000, 5060, 8008, 8082 |

## **6.2 Executive summary**

The purpose of this activity is to find the vulnerability on the target web application.

### **6.2.1 Summary Vulnerability by Severity**

Figure 6: Summary by Severity of Web Application Vulnerability Assessment

### **6.2.2 Vulnerability by Target**

| **No.** | **Domain/Server Name** | **IP Address** | **Critical** | **High** | **Medium** | **Low** | **Total** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | https://example.com | 123.123.123.123 | 0 | 0 | 1 | 3 | 4 |
| **Total** | | | **0** | **0** | **1** | **3** | **4** |

## **6.3 Web Application Vulnerability Detail**

|  |  |  |  |
| --- | --- | --- | --- |
| **ID.** | 1 | **Finding** | Cross-site scripting (DOM-based) |
| **Severity** | **High** | **Port** | 443 |
| **Target** | https://demo.testfire.net/high\_yield\_investments.htm https://demo.testfire.net/index.jsp | | |
| **Detail** | DOM-based vulnerabilities arise when a client-side script reads data from a controllable part of the DOM (for example, the URL) and processes this data in an unsafe way. DOM-based cross-site scripting arises when a script writes controllable data into the HTML document in an unsafe way. An attacker may be able to use the vulnerability to construct a URL that, if visited by another application user, will cause JavaScript code supplied by the attacker to execute within the user's browser in the context of that user's session with the application. The attacker-supplied code can perform a wide variety of actions, such as stealing the victim's session token or login credentials, performing arbitrary actions on the victim's behalf, and logging their keystrokes. Users can be induced to visit the attacker's crafted URL in various ways, similar to the usual attack delivery vectors for reflected cross-site scripting vulnerabilities.  Burp Suite automatically identifies this issue using static code analysis, which may lead to false positives that are not actually exploitable. The relevant code and execution paths should be reviewed to determine whether this vulnerability is indeed present, or whether mitigations are in place that would prevent exploitation. | | |
| **Solution** | The most effective way to avoid DOM-based cross-site scripting vulnerabilities is not to dynamically write data from any untrusted source into the HTML document. If the desired functionality of the application means that this behavior is unavoidable, then defenses must be implemented within the client-side code to prevent malicious data from introducing script code into the document. In many cases, the relevant data can be validated on a whitelist basis, to allow only content that is known to be safe. In other cases, it will be necessary to sanitize or encode the data. This can be a complex task, and depending on the context that the data is to be inserted may need to involve a combination of JavaScript escaping, HTML encoding, and URL encoding, in the appropriate sequence. | | |
| **Remark** | https://portswigger.net/web-security/cross-site-scripting https://portswigger.net/web-security/cross-site-scripting/dom-based | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **ID.** | 2 | **Finding** | Cross-site scripting (reflected) |
| **Severity** | **High** | **Port** | 443 |
| **Target** | https://demo.testfire.net/index.jsp [content parameter] https://demo.testfire.net/sendFeedback [name parameter] https://demo.testfire.net/sendFeedback [email\_addr parameter] | | |
| **Detail** | Reflected cross-site scripting vulnerabilities arise when data is copied from a request and echoed into the application's immediate response in an unsafe way. An attacker can use the vulnerability to construct a request that, if issued by another application user, will cause JavaScript code supplied by the attacker to execute within the user's browser in the context of that user's session with the application. The attacker-supplied code can perform a wide variety of actions, such as stealing the victim's session token or login credentials, performing arbitrary actions on the victim's behalf, and logging their keystrokes. Users can be induced to issue the attacker's crafted request in various ways. For example, the attacker can send a victim a link containing a malicious URL in an email or instant message. They can submit the link to popular web sites that allow content authoring, for example in blog comments. And they can create an innocuous looking web site that causes anyone viewing it to make arbitrary cross-domain requests to the vulnerable application (using either the GET or the POST method). The security impact of cross-site scripting vulnerabilities is dependent upon the nature of the vulnerable application, the kinds of data and functionality that it contains, and the other applications that belong to the same domain and organization. If the application is used only to display non-sensitive public content, with no authentication or access control functionality, then a cross-site scripting flaw may be considered low risk. However, if the same application resides on a domain that can access cookies for other more security-critical applications, then the vulnerability could be used to attack those other applications, and so may be considered high risk. Similarly, if the organization that owns the application is a likely target for phishing attacks, then the vulnerability could be leveraged to lend credibility to such attacks, by injecting Trojan functionality into the vulnerable application and exploiting users' trust in the organization in order to capture credentials for other applications that it owns. In many kinds of application, such as those providing online banking functionality, cross-site scripting should always be considered high risk. | | |
| **Solution** | In most situations where user-controllable data is copied into application responses, cross-site scripting   attacks can be prevented using two layers of defenses:   Input should be validated as strictly as possible on arrival, given the kind of content that  it is expected to contain. For example, personal names should consist of alphabetical  and a small range of typographical characters, and be relatively short; a year of birth  should consist of exactly four numerals; email addresses should match a well-defined  regular expression. Input which fails the validation should be rejected, not sanitized. User input should be HTML-encoded at any point where it is copied into  application responses. All HTML metacharacters, including < > " ' and =, should be  replaced with the corresponding HTML entities (&lt; &gt; etc). In cases where the application's functionality allows users to author content using   a restricted subset of HTML tags and attributes (for example, blog comments which   allow limited formatting and linking), it is necessary to parse the supplied HTML to   validate that it does not use any dangerous syntax; this is a non-trivial task. | | |
| **Remark** | https://portswigger.net/web-security/cross-site-scripting https://portswigger.net/web-security/cross-site-scripting/reflected https://support.portswigger.net/customer/portal/articles/1965737-Methodology\_XSS.html | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **ID.** | 3 | **Finding** | File path manipulation |
| **Severity** | **High** | **Port** | 443 |
| **Target** | https://demo.testfire.net/index.jsp [content parameter] | | |
| **Detail** | File path manipulation vulnerabilities arise when user-controllable data is placed into a file or URL path that is used on the server to access local resources, which may be within or outside the web root. If vulnerable, an attacker can modify the file path to access different resources, which may contain sensitive information. Even where an attack is constrained within the web root, it is often possible to retrieve items that are normally protected from direct access, such as application configuration files, the source code for server-executable scripts, or files with extensions that the web server is not configured to serve directly. | | |
| **Solution** | Ideally, application functionality should be designed in such a way that user-controllable data does not need to be placed into file or URL paths in order to access local resources on the server. This can normally be achieved by referencing known files via an index number rather than their name. If it is considered unavoidable to place user data into file or URL paths, the data should be strictly validated against a whitelist of accepted values. Note that when accessing resources within the web root, simply blocking input containing file path traversal sequences (such as dot-dot-slash) is not always sufficient to prevent retrieval of sensitive information, because some protected items may be accessible at the original path without using any traversal sequences. | | |
| **Remark** | https://portswigger.net/web-security/file-path-traversal | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **ID.** | 4 | **Finding** | TLS certificate |
| **Severity** | **Medium** | **Port** | 443 |
| **Target** | https://demo.testfire.net/ | | |
| **Detail** | TLS (or SSL) helps to protect the confidentiality and integrity of information in transit between the browser and server, and to provide authentication of the server's identity. To serve this purpose, the server must present an TLS certificate that is valid for the server's hostname, is issued by a trusted authority and is valid for the current date. If any one of these requirements is not met, TLS connections to the server will not provide the full protection for which TLS is designed. It should be noted that various attacks exist against TLS in general, and in the context of HTTPS web connections in particular. It may be possible for a determined and suitably-positioned attacker to compromise TLS connections without user detection even when a valid TLS certificate is used. | | |
| **Solution** |  | | |
| **Remark** | https://wiki.mozilla.org/Security/Server\_Side\_TLS | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **ID.** | 5 | **Finding** | Strict transport security not enforced |
| **Severity** | **Low** | **Port** | 443 |
| **Target** | https://demo.testfire.net/ | | |
| **Detail** | The application fails to prevent users from connecting to it over unencrypted connections. An attacker able to modify a legitimate user's network traffic could bypass the application's use of SSL/TLS encryption, and use the application as a platform for attacks against its users. This attack is performed by rewriting HTTPS links as HTTP, so that if a targeted user follows a link to the site from an HTTP page, their browser never attempts to use an encrypted connection. The sslstrip tool automates this process.   To exploit this vulnerability, an attacker must be suitably positioned to intercept and modify the victim's network traffic.This scenario typically occurs when a client communicates with the server over an insecure connection such as public Wi-Fi, or a corporate or home network that is shared with a compromised computer. Common defenses such as switched networks are not sufficient to prevent this. An attacker situated in the user's ISP or the application's hosting infrastructure could also perform this attack. Note that an advanced adversary could potentially target any connection made over the Internet's core infrastructure. | | |
| **Solution** | The application should instruct web browsers to only access the application using HTTPS. To do this, enable HTTP Strict Transport Security (HSTS) by adding a response header with the name 'Strict-Transport-Security' and the value 'max-age=expireTime', where expireTime is the time in seconds that browsers should remember that the site should only be accessed using HTTPS. Consider adding the 'includeSubDomains' flag if appropriate. Note that because HSTS is a "trust on first use" (TOFU) protocol, a user who has never accessed the application will never have seen the HSTS header, and will therefore still be vulnerable to SSL stripping attacks. To mitigate this risk, you can optionally add the 'preload' flag to the HSTS header, and submit the domain for review by browser vendors. | | |
| **Remark** | https://developer.mozilla.org/en-US/docs/Web/Security/HTTP\_strict\_transport\_security https://github.com/moxie0/sslstrip https://hstspreload.appspot.com/ | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **ID.** | 6 | **Finding** | Link manipulation (DOM-based) |
| **Severity** | **Low** | **Port** | 443 |
| **Target** | https://demo.testfire.net/disclaimer.htm https://demo.testfire.net/disclaimer.htm | | |
| **Detail** | DOM-based vulnerabilities arise when a client-side script reads data from a controllable part of the DOM (for example, the URL) and processes this data in an unsafe way. DOM-based link manipulation arises when a script writes controllable data to a navigation target within the current page, such as a clickable link or the submission URL of a form. An attacker may be able to use the vulnerability to construct a URL that, if visited by another application user, will modify the target of links within the response. An attacker may be able to leverage this to perform various attacks, including:  Causing the user to redirect to an arbitrary external URL, to facilitate a phishing attack.Causing the user to submit sensitive form data to a server controlled by the attacker.Causing the user to perform an unintended action within the application, by changing the file or query string associated with a link.Bypassing browser anti-XSS defenses by injecting on-site links containing XSS exploits, since browser anti-XSS defenses typically do not operate on on-site links. Burp Suite automatically identifies this issue using static code analysis, which may lead to false positives that are not actually exploitable. The relevant code and execution paths should be reviewed to determine whether this vulnerability is indeed present, or whether mitigations are in place that would prevent exploitation. | | |
| **Solution** | The most effective way to avoid DOM-based link manipulation vulnerabilities is not to dynamically set the target URLs of links or forms using data that originated from any untrusted source. If the desired functionality of the application means that this behavior is unavoidable, then defenses must be implemented within the client-side code to prevent malicious data from introducing an arbitrary URL as a link target. In general, this is best achieved by using a whitelist of URLs that are permitted link targets, and strictly validating the target against this list before setting the link target. | | |
| **Remark** | https://portswigger.net/web-security/dom-based/link-manipulation | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **ID.** | 7 | **Finding** | Open redirection (DOM-based) |
| **Severity** | **Low** | **Port** | 443 |
| **Target** | https://demo.testfire.net/disclaimer.htm https://demo.testfire.net/disclaimer.htm https://demo.testfire.net/disclaimer.htm | | |
| **Detail** | DOM-based vulnerabilities arise when a client-side script reads data from a controllable part of the DOM (for example, the URL) and processes this data in an unsafe way.  DOM-based open redirection arises when a script writes controllable data into the target of a redirection in an unsafe way. An attacker may be able to use the vulnerability to construct a URL that, if visited by another application user, will cause a redirection to an arbitrary external domain. This behavior can be leveraged to facilitate phishing attacks against users of the application. The ability to use an authentic application URL, targeting the correct domain and with a valid SSL certificate (if SSL is used), lends credibility to the phishing attack because many users, even if they verify these features, will not notice the subsequent redirection to a different domain. Note: If an attacker is able to control the start of the string that is passed to the redirection API, then it may be possible to escalate this vulnerability into a JavaScript injection attack, by using a URL with the javascript: pseudo-protocol to execute arbitrary script code when the URL is processed by the browser.  Burp Suite automatically identifies this issue using static code analysis, which may lead to false positives that are not actually exploitable. The relevant code and execution paths should be reviewed to determine whether this vulnerability is indeed present, or whether mitigations are in place that would prevent exploitation. | | |
| **Solution** | The most effective way to avoid DOM-based open redirection vulnerabilities is not to dynamically set redirection targets using data that originated from any untrusted source. If the desired functionality of the application means that this behavior is unavoidable, then defenses must be implemented within the client-side code to prevent malicious data from introducing an arbitrary URL as a redirection target. In general, this is best achieved by using a whitelist of URLs that are permitted redirection targets, and strictly validating the target against this list before performing the redirection. | | |
| **Remark** | https://portswigger.net/web-security/dom-based/open-redirection | | |



# Port Discovery

| **Port** | **Protocol** | **Service** |
| --- | --- | --- |
| 22 | TCP | ssh |
| 80 | TCP | http |
| 110 | TCP | pop3 |
| 143 | TCP | imap-proxy |
| 443 | TCP | https |
| 465 | TCP | ssl/smtp |
| 587 | TCP | smtp |
| 993 | TCP | ssl/imap-proxy |
| 995 | TCP | ssl/pop3 |
| 8443 | TCP | https-alt? |
| 9071 | TCP | ssl/http |

# Appendix

## **8.1 About Nessus**

Nessus is a proprietary vulnerability scanner developed by Tenable, Inc. Nessus is trusted by more than 30,000 organizations worldwide as one of the most widely deployed security technologies on the planet - and the gold standard for vulnerability assessment.

Reference: https://www.tenable.com/products/nessus

### **8.1.1 Nessus vulnerabilities**

As information about new vulnerabilities are discovered and released into the public domain, Tenable, Inc. research staff designs programs to enable Nessus to detect them. These programs are named plugins, and are written in the Nessus proprietary scripting language, called Nessus Attack Scripting Language (NASL). Plugins contain vulnerability information, a generic set of remediation actions, and the algorithm to test for the presence of the security issue.

Reference: https://www.tenable.com/plugins

### **8.1.2 Nessus risk score**

There are four risk levels in this document: Critical, High, Medium, and Low. There are methods for determining the risk level. Based on the Common Vulnerability Scoring System (CVSS), a standard for assessing the severity of vulnerabilities in computer systems. Regarded by the NIAC (National Infrastructure Advisory Council), expert assessments are measured in a range of 0 – 10

| **Severity** | **Description** | **Score** |
| --- | --- | --- |
| Critical | Vulnerabilities that score in the critical range usually have most of the following characteristics:   * Exploitation of the vulnerability likely results in root-level compromise of servers or infrastructure devices. * Exploitation is usually straightforward, in the sense that the attacker does not need any special authentication credentials or knowledge about individual victims, and does not need to persuade a target user, for example via social engineering, into performing any special functions.   For critical vulnerabilities, is advised that you patch or upgrade as soon as possible, unless you have other mitigating measures in place. For example, a mitigating factor could be if your installation is not accessible from the Internet. | 9.0 – 10.0 |
| High | Vulnerabilities that score in the high range usually have some of the following characteristics:   * The vulnerability is difficult to exploit. * Exploitation could result in elevated privileges. * Exploitation could result in a significant data loss or downtime. | 7.0 – 8.9 |
| Medium | Vulnerabilities that score in the medium range usually have some of the following characteristics:   * Vulnerabilities that require the attacker to manipulate individual victims via social engineering tactics. * Denial of service vulnerabilities that are difficult to set up. * Exploits that require an attacker to reside on the same local network as the victim. * Vulnerabilities where exploitation provides only very limited access. * Vulnerabilities that require user privileges for successful exploitation. | 4.0 – 6.9 |
| Low | Vulnerabilities in the low range typically have very little impact on an organization's business. Exploitation of such vulnerabilities usually requires local or physical system access. | 0.1 – 3.9 |

## **8.2 About Acunetix**

Acunetix by Invicti Security is an application security testing tool built to help small & mid-size organizations around the world take control of their web security. Acunetix is built to evolve and stay ahead of cybersecurity changes. Acunetix industry-leading dynamic and interactive application security testing (DAST and IAST) technology automates vulnerability management and empowers security teams to uncover more vulnerabilities, reduce false positives, increase productivity, and simplify remediation efforts.

Reference: https://www.acunetix.com/product/, https://www.acunetix.com/about/

### **8.2.1 Acunetix web vulnerabilities**

The following reference link is a list of known web application vulnerabilities that can be automatically detected by Acunetix.

Reference: https://www.acunetix.com/vulnerabilities/web/

### **8.2.2 Acunetix risk score**

Severity is a metric for classifying the level of risk which a security vulnerability poses. The severity level of a vulnerability is assigned based on the security risk posed to an organization should the vulnerability be exploited, as well as the degree of difficulty involved in exploiting it. The result of a successful attack by exploiting a vulnerability could vary from denial of service and information disclosure to a complete compromise of applications or systems. The following provides a description of what the results in this analysis consider to be the impact of each vulnerability severity level.

| **Severity** | **Description** |
| --- | --- |
| High | An attacker can **fully** compromise the confidentiality, integrity, or availability, of a target system without specialized access, user interaction or circumstances that are beyond the attacker’s control. Very likely to allow lateral movement and escalation of attack to other systems on the internal network of the vulnerable application. |
| Medium | An attacker can **partially** compromise the confidentiality, integrity, or availability, of a target system. Specialized access, user interaction, or circumstances that are beyond the attacker’s control may be required for an attack to succeed. Very likely to be used in conjunction with other vulnerabilities to escalate an attack. |
| Low | An attacker can **limitedly** compromise the confidentiality, integrity, or availability, of a target system. Specialized access, user interaction, or circumstances that are beyond the attacker’s control is required for an attack to succeed. Needs to be used in conjunction with other vulnerabilities to escalate an attack. |