ThreatHawk Vulnerability Assessment Team

Vulnerability Report

Report Version: 1.0

Report ID: fd62a1fa-ddb0-426f-b895-be5c8a56e590

Date:[Date]

CONFIDENTIAL
# Table of contents

## Scope

## Statistics and Risk

### Issues (8)

- A7: Cross-Site Scripting (XSS)  High
- A6: Security Misconfiguration  High
- A3: Sensitive Data Exposure  Medium
- A2: Broken Authentication  Medium

- [T1156] .bash_profile and .bashrc  Info
- (CAPEC-558) Accessibility Features  Info

- 1.1.4 Requirements for a firewall at each Internet connection and between any demilitarized zone (DM...  Info
- 3.2 Do not store sensitive authentication data after authorization (even if encrypted). If sensitive...  Info

## Report authors
Scope

<table>
<thead>
<tr>
<th>IP</th>
<th>hostname</th>
<th>role</th>
<th>comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>127.0.0.1</td>
<td>localhost.localdomain</td>
<td>PROD</td>
<td>client asked to test this one with care</td>
</tr>
<tr>
<td>255.255.255.255</td>
<td>N/A</td>
<td>DMZ</td>
<td>test you can go do whatever you want on it</td>
</tr>
<tr>
<td>192.88.371</td>
<td>domain.com</td>
<td>WEB</td>
<td>OWASP</td>
</tr>
</tbody>
</table>

Methodology and Standards:

- OSTTMM (Open Source Security Testing Methodology Manual)
- OWASP (Open Web Application Security Project)
- ISSAF (Information Systems Security Assessment Framework)
- WASC-TC (Web Application Security Consortium Threat Classification)
- PTF (Penetration Testing Framework)
- OISSG (Information Systems Security Assessment Framework)
- NIST SP800-115 (Technical Guide to Information Security Testing and Assessment)

Statistics and Risk

<table>
<thead>
<tr>
<th>Severity</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Critical</td>
<td>0</td>
</tr>
<tr>
<td>High</td>
<td>2</td>
</tr>
<tr>
<td>Medium</td>
<td>2</td>
</tr>
<tr>
<td>Low</td>
<td>0</td>
</tr>
<tr>
<td>Info</td>
<td>4</td>
</tr>
</tbody>
</table>

The risk of application security vulnerabilities discovered during an assessment will be rated according to a custom-tailored version of the OWASP Risk Rating Methodology. Risk severity is determined based on the estimated technical and business impact of the vulnerability, and on the estimated likelihood of the vulnerability being exploited:

Overall Risk Severity

<table>
<thead>
<tr>
<th>Impact</th>
<th>HIGH</th>
<th>Medium</th>
<th>High</th>
<th>Critical</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MEDIUM</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td></td>
<td>LOW</td>
<td>Info</td>
<td>Low</td>
<td>Medium</td>
</tr>
<tr>
<td></td>
<td>LOW</td>
<td>MEDIUM</td>
<td>HIGH</td>
<td></td>
</tr>
</tbody>
</table>

Our Risk rating is based on this calculation: Risk = Likelihood * Impact.
Issues (8)

High  A7: Cross-Site Scripting (XSS)

Issue status:
Open (Waiting for review)

Vulnerability description
XSS is the second most prevalent issue in the OWASP Top 10, and is found in around two thirds of all applications. Automated tools can find some XSS problems automatically, particularly in mature technologies such as PHP, J2EE / JSP, and ASP.NET. There are three forms of XSS, usually targeting users' browsers: * Reflected XSS: The application or API includes unvalidated and unescaped user input as part of HTML output. A successful attack can allow the attacker to execute arbitrary HTML and JavaScript in the victim’s browser. Typically the user will need to interact with some malicious link that points to an attacker-controlled page, such as malicious watering hole websites, advertisements, or similar. * Stored XSS: The application or API stores unsanitized user input that is viewed at a later time by another user or an administrator. Stored XSS is often considered a high or critical risk. * DOM XSS: JavaScript frameworks, single-page applications, and APIs that dynamically include attacker-controllable data to a page are vulnerable to DOM XSS. Ideally, the application would not send attacker-controllable data to unsafe JavaScript APIs. Typical XSS attacks include session stealing, account takeover, MFA bypass, DOM node replacement or defacement (such as trojan login panels), attacks against the user’s browser such as malicious software downloads, key logging, and other client-side attacks.

References
https://owasp.org/www-project-top-ten/2017/A7_2017-Cross-Site_Scripting_(XSS)
https://owasp.org/www-project-top-ten/

High  A6: Security Misconfiguration

Issue status:
Open (Waiting for review)

Vulnerability description
Security misconfiguration can happen at any level of an application stack, including the network services, platform, web server, application server, database, frameworks, custom code, and pre-installed virtual machines, containers, or storage. Automated scanners are useful for detecting misconfigurations, use of default accounts or configurations, unnecessary services, legacy options, etc. The application might be vulnerable if the application is: * Missing appropriate security hardening across any part of the application stack, or improperly configured permissions on cloud services. * Unnecessary features are enabled or installed (e.g. unnecessary ports, services, pages, accounts, or privileges). * Default accounts and their passwords still enabled and unchanged. * Error handling reveals stack traces or other overly informative error messages to users. * For upgraded systems, latest security features are disabled or not configured securely. * The security settings in the application servers, application frameworks (e.g. Struts, Spring, ASP.NET), libraries, databases, etc. not set to secure values. * The server does not send security headers or directives or they are not set to secure values. * The software is out of date or vulnerable (see A9:2017-Using Components with Known Vulnerabilities). Without a concerted, repeatable application security configuration process, systems are at a higher risk.

References
https://owasp.org/www-project-top-ten/

Medium  A3: Sensitive Data Exposure

Issue status:
Open (Waiting for review)

Vulnerability description
Over the last few years, this has been the most common impactful attack. The most common flaw is simply not encrypting sensitive data. When crypto is employed, weak key generation and management, and weak algorithm, protocol and cipher usage is common, particularly for weak password hashing storage techniques. For data in transit, server-side weaknesses are mainly easy to detect, but hard for data at rest. The first thing is to determine the protection needs of data in transit and at rest. For example, passwords, credit card numbers, health records, personal information and business secrets require extra protection, particularly if that data falls under privacy laws, e.g. EU’s General Data Protection Regulation (GDPR), or regulations, e.g. financial data protection such as PCI Data Security Standard (PCI DSS). For all such data: * Is any data transmitted in clear text? This concerns protocols such as HTTP, SMTP, and FTP. External internet traffic is especially dangerous. Verify all internal traffic e.g. between load balancers, web servers, or back-end systems. * Are any old or weak cryptographic algorithms used either by default or in older code? * Are default crypto keys in use, weak crypto keys generated or re-used, or is proper key management or rotation missing? * Is encryption not enforced, e.g. are any user agent (browser) security directives or headers missing? * Does the user agent (e.g. app, mail client) not verify if the received server certificate is valid?

References
https://owasp.org/www-project-top-ten/2017/A3_2017-Sensitive_Data_Exposure
Medium A2: Broken Authentication

Issue status:
Open (Waiting for review)

Vulnerability description
The prevalence of broken authentication is widespread due to the design and implementation of most identity and access controls. Session management is the bedrock of authentication and access controls, and is present in all stateful applications. Attackers can detect broken authentication using manual means and exploit them using automated tools with password lists and dictionary attacks. Confirmation of the user’s identity, authentication, and session management are critical to protect against authentication-related attacks. There may be authentication weaknesses if the application: * Permits automated attacks such as credential stuffing, where the attacker has a list of valid usernames and passwords. * Permits brute force or other automated attacks. * Permits default, weak, or well-known passwords, such as “Password1” or “admin/admin”. * Uses weak or ineffective credential recovery and forgot-password processes, such as “knowledge-based answers”, which cannot be made safe. * Uses plain text, encrypted, or weakly hashed passwords (see A3:2017-Sensitive Data Exposure). * Has missing or ineffective multi-factor authentication. * Exposes Session IDs in the URL (e.g., URL rewriting). * Does not rotate Session IDs after successful login. * Does not properly invalidate Session IDs. User sessions or authentication tokens (particularly single sign-on (SSO) tokens) aren’t properly invalidated during logout or a period of inactivity.

References
https://owasp.org/www-project-top-ten/2017/A2_2017-Broken_Authentication
https://owasp.org/www-project-top-ten/

Info [T1156] .bash_profile and .bashrc

Issue status:
Open (Waiting for review)

Vulnerability description

References
https://attack.mitre.org/techniques/T1156

Info [CAPEC-558] Accessibility Features

Issue status:
Open (Waiting for review)

Vulnerability description

References
https://attack.mitre.org/techniques/T1015
https://capec.mitre.org/data/definitions/558.html
https://www.slideshare.net/DennisMaldonado5/sticky-keys-to-the-kingdom
http://blog.crowdstrike.com/registry-analysis-with-crowdresponse/

Info 1.1.4 Requirements for a firewall at each Internet connection and between any demilitarized zone (DM... 

Issue status:
Open (Waiting for review)

Vulnerability description
1.1.4 Requirements for a firewall at each Internet connection and between any demilitarized zone (DMZ) and the internal network zone

Proof of Concept
Testing:

1.1.4.a Examine the firewall configuration standards and verify that they include requirements for a firewall at each Internet connection and between
1.1.4.b Verify that the current network diagram is consistent with the firewall configuration standards

1.1.4.c Observe network configurations to verify that a firewall is in place at each Internet connection and between any demilitarized zone (DMZ) and the internal network zone, per the documented configuration standards and network diagrams.

Guidance:

Using a firewall on every Internet connection coming into (and out of) the network, and between any DMZ and the internal network, allows the organization to monitor and control access and minimizes the chances of a malicious individual obtaining access to the internal network via an unprotected connection.

References
https://www.pcisecuritystandards.org/
https://www.pcisecuritystandards.org/documents/PCI_DSS_v3-2-1.pdf

Info 3.2 Do not store sensitive authentication data after authorization (even if encrypted). If sensitive...

Issue status:
Open (Waiting for review)

Vulnerability description
3.2 Do not store sensitive authentication data after authorization (even if encrypted). If sensitive authentication data is received, render all data unrecoverable upon completion of the authorization process. It is permissible for issuers and companies that support issuing services to store sensitive authentication data if:• There is a documented business justification for the storage of sensitive authentication data.

Proof of Concept
Testing:

3.2.a For issuers and/or companies that support issuing services and store sensitive authentication data, review policies and interview personnel to verify there is a documented business justification for the storage of sensitive authentication data.

3.2.b For issuers and/or companies that support issuing services and store sensitive authentication data, examine data stores and system configurations to verify that the sensitive authentication data is secured.

3.2.c For all other entities, if sensitive authentication data is received, review policies and procedures, and examine system configurations to verify the data is not retained after authorization.

3.2.d For all other entities, if sensitive authentication data is received, review procedures and examine the processes for securely deleting the data to verify that the data is unrecoverable.

Guidance:

Sensitive authentication data consists of full track data, card validation code or value, and PIN data. Storage of sensitive authentication data after authorization is prohibited! This data is very valuable to malicious individuals as it allows them to generate counterfeit payment cards and create fraudulent transactions. Entities that issue payment cards or that perform or support issuing services will often create and control sensitive authentication data as part of the issuing function. It is allowable for companies that perform, facilitate, or support issuing services to store sensitive authentication data only if they have a legitimate business need to store such data. It should be noted that all PCI DSS requirements, apply to issuers, and the only exclusion for issuers and issuer processors is that sensitive authentication data may be retained if there is a legitimate reason to do so. A legitimate reason is one that is necessary for the performance of the function being provided for the issuer and not one of convenience. Any such data must be stored securely and in accordance with all PCI DSS and specific payment brand requirements. For non-issuing entities, retaining sensitive authentication data post-authorization is not permitted.

References
https://www.pcisecuritystandards.org/
https://www.pcisecuritystandards.org/documents/PCI_DSS_v3-2-1.pdf
<table>
<thead>
<tr>
<th>Name</th>
<th>E-mail</th>
<th>Social links</th>
<th>Website</th>
</tr>
</thead>
<tbody>
<tr>
<td>ThreatHawk, LLC</td>
<td><a href="mailto:team@threathawk.io">team@threathawk.io</a></td>
<td></td>
<td><a href="https://threathawk.io">https://threathawk.io</a></td>
</tr>
</tbody>
</table>