Opsgenie
Opsgenie is a modern incident management platform for operating always-on services, empowering Dev & Ops teams to plan for service disruptions and stay in control during incidents.

Bugcrowd Ongoing program results

Report created on July 09, 2020
Report date range: April 01, 2020 - June 30, 2020

bugcrowd

Prepared by

bmarriott@atlassian.com
# Table of contents

1 Executive summary 3

2 Reporting and methodology 4
   Background 4

3 Targets and scope 5
   Scope 5

4 Findings summary 6
   Findings by severity 6
   Risk and priority key 7

5 Appendix 8
   Submissions over time 8
   Submissions signal 8
   Bug types overview 9

6 Closing statement 10
**Executive summary**

**Opsgenie** engaged Bugcrowd, Inc. to perform an Ongoing Bounty Program, commonly known as a crowd-sourced penetration test.

An Ongoing Bounty Program is a cutting-edge approach to an application assessment or penetration test. Traditional penetration tests use only one or two personnel to test an entire scope of work, while an Ongoing Bounty leverages a crowd of security researchers. This increases the probability of discovering esoteric issues that automated testing cannot find and that traditional vulnerability assessments may miss in the same testing period.

The purpose of this engagement was to identify security vulnerabilities in the targets listed in the targets and scope section. Once identified, each vulnerability was rated for technical impact defined in the findings summary section of the report.

This report shows testing for **Opsgenie's** targets during the period of: **04/01/2020 – 06/30/2020**.

For this Ongoing Program, submissions were received from **68** unique researchers.

The continuation of this document summarizes the findings, analysis, and recommendations from the Ongoing Bounty Program performed by Bugcrowd for **Opsgenie**.
Reporting and methodology

Background

The strength of crowdsourced testing lies in multiple researchers, the pay-for-results model, and the varied methodologies that the researchers implement. To this end, researchers are encouraged to use their own individual methodologies on Bugcrowd Ongoing programs.

The workflow of every penetration test can be divided into the following four phases:

01 Reconnaissance
Gathering information before the attack

02 Enumeration
Finding attack vectors

03 Exploitation
Verifying security weaknesses

04 Documentation
Collecting results

Bugcrowd researchers who perform web application testing and vulnerability assessment usually subscribe to a variety of methodologies following the highlighted workflow, including the following:
Targets and scope

Scope

Prior to the Ongoing program launching, Bugcrowd worked with Opsgenie to define the Rules of Engagement, commonly known as the program brief, which includes the scope of work. The following targets were considered explicitly in scope for testing:

- Opsgenie (Android)
- Opsgenie (IoS)
- *.opsgeni.us
- mobileapp.opsgeni.us
- app.opsgeni.us

All details of the program scope and full program brief can be reviewed in the Program Brief.
Findings summary

Findings by severity

The following chart shows all valid assessment findings from the program by technical severity.
**Risk and priority key**

The following key is used to explain how Bugcrowd rates valid vulnerability submissions and their technical severity. As a trusted advisor Bugcrowd also provides common "next steps" for program owners per severity category.

<table>
<thead>
<tr>
<th>TECHNICAL SEVERITY</th>
<th>EXAMPLE VULNERABILITY TYPES</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Critical</strong></td>
<td>- Remote Code Execution</td>
</tr>
<tr>
<td></td>
<td>- Vertical Authentication Bypass</td>
</tr>
<tr>
<td></td>
<td>- XML External Entities Injection</td>
</tr>
<tr>
<td></td>
<td>- SQL Injection</td>
</tr>
<tr>
<td></td>
<td>- Insecure Direct Object Reference for a critical function</td>
</tr>
<tr>
<td><strong>High</strong></td>
<td>- Lateral authentication bypass</td>
</tr>
<tr>
<td></td>
<td>- Stored Cross-Site Scripting</td>
</tr>
<tr>
<td></td>
<td>- Cross-Site Request Forgery for a critical function</td>
</tr>
<tr>
<td></td>
<td>- Insecure Direct Object Reference for an important function</td>
</tr>
<tr>
<td></td>
<td>- Internal Server-Side Request Forgery</td>
</tr>
<tr>
<td><strong>Medium</strong></td>
<td>- Reflected Cross-Site Scripting with limited impact</td>
</tr>
<tr>
<td></td>
<td>- Cross-Site Request Forgery for an important function</td>
</tr>
<tr>
<td></td>
<td>- Insecure Direct Object Reference for an unimportant function</td>
</tr>
<tr>
<td><strong>Low</strong></td>
<td>- Cross-Site Scripting with limited impact</td>
</tr>
<tr>
<td></td>
<td>- Cross-Site Request Forgery for an unimportant function</td>
</tr>
<tr>
<td></td>
<td>- External Server-Side Request Forgery</td>
</tr>
<tr>
<td><strong>Informational</strong></td>
<td>- Lack of code obfuscation</td>
</tr>
<tr>
<td></td>
<td>- Autocomplete enabled</td>
</tr>
<tr>
<td></td>
<td>- Non-exploitable SSL issues</td>
</tr>
</tbody>
</table>

**Bugcrowd’s Vulnerability Rating Taxonomy**

More detailed information regarding our vulnerability classification can be found at: https://bugcrowd.com/vrt
Appendix

Included in this appendix are auxiliary metrics and insights into the Ongoing program. This includes information regarding submissions over time, payouts and prevalent issue types.

Submissions over time

The timeline below shows submissions received and validated by the Bugcrowd team:

![Submissions Over Time Graph]

Submissions signal

A total of 83 submissions were received, with 10 unique valid issues discovered. Bugcrowd identified 32 duplicate submissions, removed 39 invalid submissions, and is processing 2 submissions. The ratio of unique valid submissions to noise was 12%.

<table>
<thead>
<tr>
<th>Submission Outcome</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid</td>
<td>10</td>
</tr>
<tr>
<td>Invalid</td>
<td>39</td>
</tr>
<tr>
<td>Duplicate</td>
<td>32</td>
</tr>
<tr>
<td>Processing</td>
<td>2</td>
</tr>
<tr>
<td>Total</td>
<td>83</td>
</tr>
</tbody>
</table>

![Ratio of Unique Valid Submissions to Noise]

Ongoing Program Results | Opsgenie
Bug types overview

This distribution across bug types for the Ongoing program only includes unique and valid submissions.
Introduction

This report shows testing of Opsgenie between the dates of 04/01/2020 - 06/30/2020. During this time, 68 researchers from Bugcrowd submitted a total of 83 vulnerability submissions against Opsgenie's targets. The purpose of this assessment was to identify security issues that could adversely affect the integrity of Opsgenie. Testing focused on the following:

1. Opsgenie (Android)
2. Opsgenie (iOS)
3. *.opsgeni.us
4. mobileapp.opsgeni.us
5. app.opsgeni.us

The assessment was performed under the guidelines provided in the statement of work between Opsgenie and Bugcrowd. This letter provides a high-level overview of the testing performed, and the result of that testing.

Ongoing Program Overview

An Ongoing Program is a novel approach to a penetration test. Traditional penetration tests use only one or two researchers to test an entire scope of work, while an Ongoing Program leverages a crowd of security researchers. This increases the probability of discovering esoteric issues that automated testing cannot find and that traditional vulnerability assessments may miss, in the same testing period.

It is important to note that this document represents a point-in-time evaluation of security posture. Security threats and attacker techniques evolve rapidly, and the results of this assessment are not intended to represent an endorsement of the adequacy of current security measures against future threats. This document contains information in summary form and is therefore intended for general guidance only; it is not intended as a substitute for detailed research or the exercise of professional judgment. The information presented here should not be construed as professional advice or service.

Testing Methods

This security assessment leveraged researchers that used a combination of proprietary, public, automated, and manual test techniques throughout the assessment. Commonly tested vulnerabilities include code injection, cross-site request forgery, cross-site scripting, insecure storage of sensitive data, authorization/authentication vulnerabilities, business logic vulnerabilities, and more.
Summary of Findings

During the engagement, Bugcrowd discovered the following:

<table>
<thead>
<tr>
<th>Count</th>
<th>Technical Severity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Critical vulnerability</td>
</tr>
<tr>
<td>1</td>
<td>High vulnerability</td>
</tr>
<tr>
<td>4</td>
<td>Medium vulnerabilities</td>
</tr>
<tr>
<td>3</td>
<td>Low vulnerabilities</td>
</tr>
<tr>
<td>1</td>
<td>Informational finding</td>
</tr>
</tbody>
</table>