Extracting your AWS Access Keys through a PDF file

MicroStrategy SSRF through PDF Generator (CVE-2020-24815)

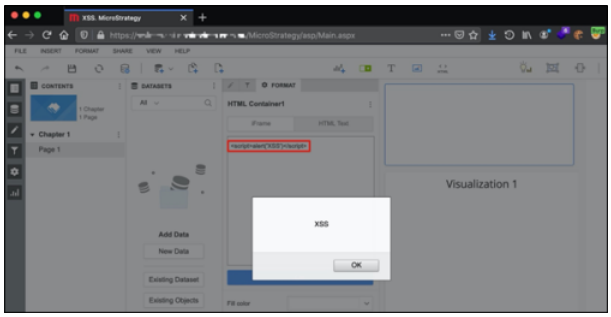
The Portable Document Format (PDF) was first developed in 1993 and since then, PDF files have gained widespread adoption, becoming the de facto standard for document publishing. It was only natural that developers followed the trending popularity and nowadays it is common for applications to allow you to export documents to a PDF file. At first glance, this functionality may seem harmless; something which you might not spare too much thought. This mentality often leads to security flaws in applications with hackers focusing their attention on overlooked components and functionality.

In this post, we will show how a seemingly innocuous "Export to PDF" function can be an open door to your organisation's internal network and how it can be used to extract sensitive information, including access keys to your organisation's cloud infrastructure. We will present the results from a recent penetration test performed by Triskele Labs, which lead to the identification of multiple vulnerabilities in a popular business analytics software. One of these vulnerabilities was first identified by Triskele Labs and was assigned CVE-2020-24815.

Cross-Site Scripting (XSS)

In a recent penetration test, Triskele Labs was tasked with testing a popular business analytics application created by MicroStrategy. MicroStrategy is a NASDAQ listed company offering a range of analytics software since 1989 for organisations such as Facebook and Commonwealth Bank of Australia (CBA).

During testing the team observed the application allowed the insertion of arbitrary HTML code when editing dashboards (also known as dossiers). This was not a flaw but indeed a feature of the application to allow the customisation of dashboards. HTML code editors often lack strict validation of user input as this is a non-trivial task. As such, these editors usually allow the insertion of arbitrary code including JavaScript. This often leads to Cross-Site Scripting (XSS) vulnerabilities. In fact, this part of the application was affected by a known stored XSS vulnerability [1].



XSS Execution

Taking it further: Blind XSS

At this point, we had a straight-forward stored XSS vulnerability in the application. Some might have stopped here and reported the issue, but we wanted to explore the impact further. We started to analyse other parts of the application and one feature that caught our attention was the "Export to PDF" function which allowed users to export a dashboard as a PDF file.

To load and convert a web page to a PDF file, developers often make use of headless browsers such as PhantomJS. Information leakage in metadata is often overlooked and as such, a reliable source of information. By analysing the metadata of an exported PDF file, we discovered that the application was created by Chromium using the Skia/PDF m65 graphics library. This indicated that the application was utilising Headless Chrome version 65 to render and convert the dashboard pages, potentially allowing the execution of JavaScript code.

PDF Metadata

The next step was to attempt to trigger the execution of XSS payloads within the headless browser used by the PDF generation service. Since the conversion occurs on the server, we had no visibility of the immediate execution of the payloads, and therefore had to use a blind XSS payload which would alert the team when triggered. Soon after injecting the payload into a dashboard and exporting it as a PDF file, we received a notification that our payload executed successfully. This demonstrated that we could execute JavaScript code under the context of the headless browser located in the server.

Blind XSS Execution

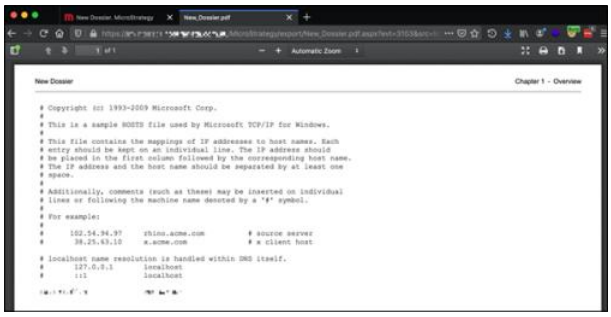
Taking it even further: XSS to SSRF

In essence, the successful execution of our XSS payload implied that we had some control over a browser located inside the organisation's internal network. As such, we could manipulate the browser to request arbitrary restricted resources which would be unreachable by outside users. This is known as a Server-Side Request Forgery (SSRF) attack.

SSRF vulnerabilities occur when an attacker can manipulate an application to make requests to arbitrary domains of the attacker's choosing. By itself, this does not represent a vulnerability as there are legitimate cases for allowing a user to define a resource to be fetched by an application, like loading a profile picture from a given URL. However, SSRF becomes a problem when an attacker can exploit this behaviour to access restricted web-based services. Although SSRFs issues have been known for a while, the vulnerability has gained considerable attention in the past few years. A recent case involved Facebook paying a $30,000 bounty for an SSRF vulnerability [2] [3], also affecting a component of a MicroStrategy application.

In our case, as the application was allowing the injection of arbitrary HTML code, the first step was to attempt loading restricted resources by simply using an iFrame tag. In fact, by injecting an iFrame tag in a dashboard and exporting this into a PDF document, it was possible to extract the contents of local files residing on the server. For example, the following payload was used to read the contents of the host's file.

```
<iframe src=file:///C:\WINDOWS\System32\drivers\etc\hosts>
```



Hosts File from Local System

Taking it much further: the Keys to the Cloud Infrastructure

At this point, we successfully demonstrated that we could access restricted data by using the PDF generation function. The next step was to gain access to even more sensitive information. As the application was hosted in AWS, an idea formed that the instance metadata could be accessible.

The metadata service provides information that aids the configuration and management of running instances. One of the solutions provided by the service is that it eliminates the need for hardcoding or distribution of sensitive credentials. The metadata is attached locally to every instance through a local IP address (169.254.169.254) which means that only software

running on the instance can access it [4], much like our PDF generation service.

Once again, we inserted an iframe tag into the dashboard, but this time pointing to the instance metadata located at *http://169.254.169.254/latest/meta-data/*. After exporting the dashboard to a PDF document, we found that the application successfully loaded and rendered the metadata information within the exported file. More surprisingly, the team discovered access keys for an AWS account by further analysing the information stored within the metadata. This was a similar scenario to what occurred in the CapitalOne breach [5].

Access Keys from Instance Metadata

Responsible Disclosure & Remediation

Upon discovering the issue, Triskele Labs immediately reported the vulnerability to the MicroStrategy team. MicroStrategy promptly replied that they were aware of the issue and that it had been resolved in recent versions of the platform. MicroStrategy stated that in the newest versions, "a new whitelisting facility was developed to mitigate the SSRF vulnerability" [6]. As such, the official MicroStrategy recommendation is for customers to upgrade to the latest version of the MicroStrategy platform. MicroStrategy also provided a reward under their bug bounty program, since the vulnerability affected supported versions of the application.

Conclusion

In summary, be careful to underestimate inconspicuous functions in your applications, as hackers will certainly focus their attention on the most overlooked components. As these functions communicate with other services within an organisation's infrastructure, they can often be affected by severe vulnerabilities like SSRF, which can become an open door to your organisation's internal resources and data. Therefore, ensure your applications are frequently pentested by a reliable team. At Triskele Labs we incorporate novel techniques within our test framework, so you can rest assured that your applications are being tested for the latest vulnerabilities.

References

1. https://packetstormsecurity.com/files/157068/MicroStrategy-Intelligence-Server-And-Web-10.4-XSS-Disclosure-SSRF-Code-Execution.html
2. https://medium.com/@win3zz/how-i-made-31500-by-submitting-a-bug-to-facebook-d31bb046e204
3. https://portswigger.net/daily-swig/facebook-security-researcher-scoops-31k-bug-bounty-for-flagging-ssrf-vulnerabilities
4. https://aws.amazon.com/blogs/security/defense-in-depth-open-firewalls-reverse-proxies-ssrf-vulnerabilities-ec2-instance-metadata-service/
5. https://blog.appsecco.com/an-ssrf-privileged-aws-keys-and-the-capital-one-breach-4c3c2cded3af
6. https://community.microstrategy.com/s/article/Securing-PDF-and-Excel-Export-with-Whitelists?language=en_US

Latest from the blog

Wdigest: Extracting Passwords in Cleartext

Read more

Why moving to multi-factor authentication is crucial for security in 2020

Read more

Responding to a cyber-attack is a game of minutes

Read more