SSD ADVISORY – IVANTI AVALANCHE DIRECTORY TRAVERSAL

May 11, 2021 SSD Secure Disclosure technical team
Vulnerability publication

TL;DR

Find out how a directory traversal vulnerability in Ivanti Avalanche allows remote unauthenticated user to access files that reside outside the 'image' folder.

Vulnerability Summary

Ivanti Avalanche powered by "Wavelink is a mobile device management system. Network security features allow you to manage wireless settings (including encryption and authentication), and apply those settings on a schedule throughout the network. Software distribution features allow you to schedule software distribution to specific devices from the Avalanche Console. Avalanche also provides tools for managing alerts and reports".

A vulnerability in Ivanti Avalanche allows remote unauthenticated users to request files that reside outside the 'image' folder.

CVE

CVE-2021-30497

Credit

An independent security researcher, Ahmed Y. Elmogy, has reported this vulnerability to the SSD Secure Disclosure program.

Affected Versions

Avalanche Premise 6.3.2 for Windows v6.3.2.3490

The vendor has promptly – in less than 7 days – fixed the vulnerability and has released a patch to address the issue.

Vulnerability Analysis

The `imageFilePath` parameter received from the user is used to allow users to retrieve locally stored images on the server. This is normally used to access images such as ‘icons’ and ‘logos’, due to lack of security checking of the parameter, a remote user can request other files to be retrieved that are neither an image or reside in the images folder that is normally accessed.

```java
String paramImageFilePath = request.getParameter("imageFilePath"); // vulnerable GET parameter
boolean cacheImage = true;
String parameterIcon = request.getParameter("icon");
if (paramImageFilePath != null) {
  File imageFile = new File(paramImageFilePath); // reading from user-input path
  byte[] icon = FileUtils.readFileToByteArray(imageFile);
  String queryString = request.getQueryString();
  if (icon != null && icon.length > 0) {
    handleIcon(response, icon, queryString, false); // outputting the contents
  } else {
    logger.warn(String.format("ImageServlet::missing icon for device(%s)", new Object[] {
      queryString
    }));
  }
...
private void handleIcon(HttpServletResponse response, byte[] icon, String imageSource, boolean cacheImage) throws IOException {
    response.setContentLength(icon.length);
    if (cacheImage) {
      HttpUtils.expiresOneWeek(response);
    } else {
      HttpUtils.expiresNow(response);
    }
    ImageInputStream inputStream = ImageIO.createImageInputStream(new ByteArrayInputStream(icon));
    try {
      Iterator < ImageReader > imageReaders = ImageIO.getImageReaders(inputStream);
      if (imageReaders.hasNext()) {
        ImageReader reader = imageReaders.next();
        String formatName = reader.getFormatName();
        response.setContentType(String.format("image/%s", new Object[] {
          formatName
        }));
      } else {
        logger.warn(String.format("ImageServlet::unknown image format for (%s)", new Object[] {
          imageSource
        }));
      }
    } finally {
      try {
        inputStream.close();
      } catch (IOException iOException) {}
    }
    ServletOutputStream outputStream = response.getOutputStream();
    outputStream.write(icon); // outputting the contents of the file
  }
```

As can be seen above, the file is accessed without regard to where it is stored, allowing a remote attacker to provide a full path to files that reside elsewhere and retrieve their content.

Exploit

Exploitation is straight forward, by accessing any of the following URL (examples), you can observe that the content return is the file stored on the remote server.

Examples:

```
ı.sav
ɛeFilePath=C:/sysprep/sysprep.inf
```

Get in touch

Any questions? Interested in our services?
We'd love to hear from you

CONTACT US