g / pulse-host-checker-rce

Watch 1 Star 0 Fork 0

Files Issues 0 Pull Requests 0 Wiki

MITM RCE Host Checker for Pulse Secure

3 Commits 1 Branches 0 Releases

Branch: master pulse-host-c... HTTP SSH https://git.lsd.cat/g/pulse-h

Giulio	855d6def8d	Updated with Cves and Juniper Advistory	2 years ago
Readme.md	855d6def8d	Updated with Cves and Juniper Advistory	2 years ago
tncc.jar	8ba4c651fb	Writeup draft	2 years ago

Readme.md

Juniper Host Checker Linux MITM RCE

CVEs

- No certificate Validation - CVE-2020-11580
- Command Injection - CVE-2020-11581
- DNS Rebindig - CVE-2020-11582

Link to Juniper official advisory SA44426

Intro

The Host Checker is a client side component that the Pulse Connect Secure appliance may require in order to connect to the VPN. The Host Checker requests a policy from the server and perform basic checks on the client accordingly. Checks may include MAC Addresses, running process (ie: checking for an antivirus) and some others. While on Windows the plugin is an ActiveX component, in Linux, Solaris and OSX it is a Java Applet. Of course client checks can always be bypassed, and an open source (yet not well documented) implementation do exist.

Sumamry

Probably in order to still work with misconfigured instances, the Host Cheker does not check neither the validity of the server certificate nor its hostname. The server can set a malicious cookie (or it can be done via DNS Rebinding), which can be used to exploit a command injection when the user is found not compliant. Note that a malicious server can force a user to be non compliant.

Code

The client implement a custom protocol in order to talk to the server. For further reference, the open source client has reverse engineered and implemented the same protocol. The file `tncc.jar` is not obfuscated in any way and the originalk source code can be obtained with almost any Java decompiler.

Certificate

Below are some extracts of code from the classes that handle the connection with the Pulse Connect Secure appliance. In `net.juniper.tnc.client.HttpNAR.HttpNAR` :

```
    private void trustAllCerts() throws Exception {
        final TrustManager[] tm = { new X509TrustManager() {
                @Override
                public X509Certificate[] getAcceptedIssuers() {
                    return null;
                }

                @Override
                public void checkClientTrusted(final X509Certificate[] array, final String s) {
                }

                @Override
                public void checkServerTrusted(final X509Certificate[] array, final String s) {
                }
            } };
        final SSLContext instance = SSLContext.getInstance(NARUtil.getSSLProtocol());
        instance.init(null, tm, new SecureRandom());
        HttpsURLConnection.setDefaultSSLSocketFactory(instance.getSocketFactory());
    }

    private void allowHostnameMismatch() {
        HttpsURLConnection.setDefaultHostnameVerifier(new HostnameVerifier() {
            @Override
            public boolean verify(final String s, final SSLSession sslSession) {
```

```
                return true;
            }
        });
    }
```

Both function gets executed when initializing the connection to a server. From the same class as above:

```
    public void initialize(final String[] array) throws Exception {
            [..parametr parsing..]
        if (this.mHomeDir.length() == 0) {
            this.mHomeDir = System.getProperty("user.home");
        }
        if (HttpNAR.gLoggingEnabled) {
            [..omitted log instructions..]
        }
        if (!this.isPlatformSupported()) {
            NARUtil.logError("HttpNAR: unsupported operating system " + NARUtil.getOSName() + "; stopping...");
            throw new Exception("Unsupported operating system");
        }
        this.mAppSupportDir = this.getPlatformSupportDir(this.mHomeDir);
        [..proxy usage..]

        (this.mTncClient = new TNCClient()).initialize(this);
        this.loadbundledIMC();
        this.mHandshakeRequestor.start();
        this.trustAllCerts();
        this.allowHostnameMismatch();
    }
```

Cookie

In order for the Host Checker to work two cookies are needed, `DSPREAUTH` and `DSSIGNIN`. They can be either set by the server or from sending commands to a socket listening to all interfaces (but accepting connections only from localhost). The following code updates the DSPREAUTH cookie when sending periodic updates to the server. Periodic updates may or may not be required depending on the policy configuration. From `net.juniper.tnc.client.HttpNAR.HttpConnection`:

```
    public int sendUpdate(final byte[] array, final ByteArrayOutputStream byteArrayOutputStream, final boolean b) throws
        NARUtil.logInfo("TNCoHTTP: sending update = ");
        NARUtil.logData(array);
        final String byteArrayToBase64 = Base64.byteArrayToBase64(array);
        final String string = "https://" + this.mIveHost + "/dana-na/hc/tnchcupdate.cgi";
        NARUtil.logInfo("TNCoHTTP: opening connection to " + string);
        final HttpsURLConnection httpsURLConnection = (HttpsURLConnection)new URL(string).openConnection();
        httpsURLConnection.setDoOutput(true);
        httpsURLConnection.setDoInput(true);
        httpsURLConnection.setRequestProperty("Cookie", "DSPREAUTH=" + this.mPreauthCookie + ";DSSIGNIN=" + this.mSignIn
        if (this.mUserAgent.length() > 0) {
            httpsURLConnection.setRequestProperty("User-Agent", this.mUserAgent);
        }
        NARUtil.setConnectTimeout(httpsURLConnection, 2000);
        httpsURLConnection.connect();
        final PrintWriter printWriter = new PrintWriter(httpsURLConnection.getOutputStream());
        [..adding parameters..]
        if (b) {
            printWriter.print("firsttime=1;");
        }
        printWriter.close();
        String headerFieldKey;
        for (int n = 1; (headerFieldKey = httpsURLConnection.getHeaderFieldKey(n)) != null; ++n) {
            final String headerField = httpsURLConnection.getHeaderField(n);
            if (headerFieldKey.equalsIgnoreCase("Set-Cookie") && headerField.startsWith("DSPREAUTH=")) {
                final int index = headerField.indexOf(59);
                this.mPreauthCookie = headerField.substring(10, index);
                this.mPreauthOpts = headerField.substring(index);
                NARUtil.logInfo("TNCoHTTP: received response DSPREAUTH = " + this.mPreauthCookie + this.mPreauthOpts);
            }
        }
```

Command injection

When a client is found to be non compliant, remediation instructions have to be shown to the user in order to give him a chance to fix his problems. In `net.juniper.tnc.client.HttpNAR.TNCHandshake`:

```
    public void doCustomRemediateInstructions() {
        final StringBuffer sb = new StringBuffer();
        sb.append("https://").append(this.mIcURL);
        sb.append("/dana-na/auth/rdpreauth.cgi?DSPREAUTH=" + this.mCookie);
        final String string = sb.toString();
        NARUtil.logInfo("TncHandshake: launching browser for " + string);
        NARUtil.execCommand((NARUtil.isMacOSX() ? "open -a Safari " : ((NARUtil.isLinux() || NARUtil.isSolaris()) ? "fir
        NARUtil.logInfo("TncHandshake: browser launched");
    }
```

From `net.juniper.tnc.client.HttpNAR.NARUtil`:

```
    public static String execCommand(final String s) {
        String execCommand = null;
        if (NARUtil.PlatformUtil != null) {
            execCommand = NARUtil.PlatformUtil.execCommand(s);
        }
```

```
        return execCommand;
    }
```

From `net.juniper.tnc.client.NARPlatform.linux.LinuxNARPlatform` :

```
    @Override
    public String execCommand(final String s) {
        String output = null;
        try {
            final Process exec = Runtime.getRuntime().exec(s);
            final CommandOutputThread commandOutputThread = new CommandOutputThread(exec.getInputStream());
            final CommandOutputThread commandOutputThread2 = new CommandOutputThread(exec.getErrorStream());
            commandOutputThread.start();
            commandOutputThread2.start();
            final int wait = exec.waitFor();
            if (wait != 0) {
                this.logError("Command " + s + " failed; return = " + wait + "; error output = " + commandOutputThread2.
            }
            output = commandOutputThread.getOutput();
        }
        catch (Exception ex) {
            this.logException(ex);
        }
        return output;
    }
```

As we can see, the `NARUtil.execCommand()` function is just a wrapper around `Runtime.getRuntime().exec()` .

Full Chain

An attacker who is in a position where he can perform a Man in the Middle attack may spoof the server and send a malicious cookie along with an impossible to comply policy. An example cookie could be any method of command injection on linux (ex: `; sleep 20;` , `$(sleep 20)` , `\nsleep 20` , etc.). The client will then fail to comply with the policy and execute the command with the appended value when trying to show remediation instructions.

DNS Rebinding (Bonus)

The Host Checker is controlled using a command socket. By default, when the Host Checker is started, it opens a socket using `ServerSocket(0)` which will automatically choose a port to listen on all interfaces. The selected port will then be writte to `~/.pulse_secure/narport.txt` . The code prevents sending commands froma non local host but apart from that doesn't have any other authentication mechanism. An attacker may brute force the port using JavaScript or if in the same network directly ports can since it is listening on all interfaces. Once the ports is known, a DNS Rebinding attack can be done and commands can be sent to the socket. While this does not imply a command execution per se, one of the supported commands is `setcookie` which sets the cookie used for the command injection descripted in the paragraph above. Note: the command socket expect a command to start at the beginnig of the first line but will try to parse up to 25 invalid commands before exiting, so a GET or a POST requests should work.