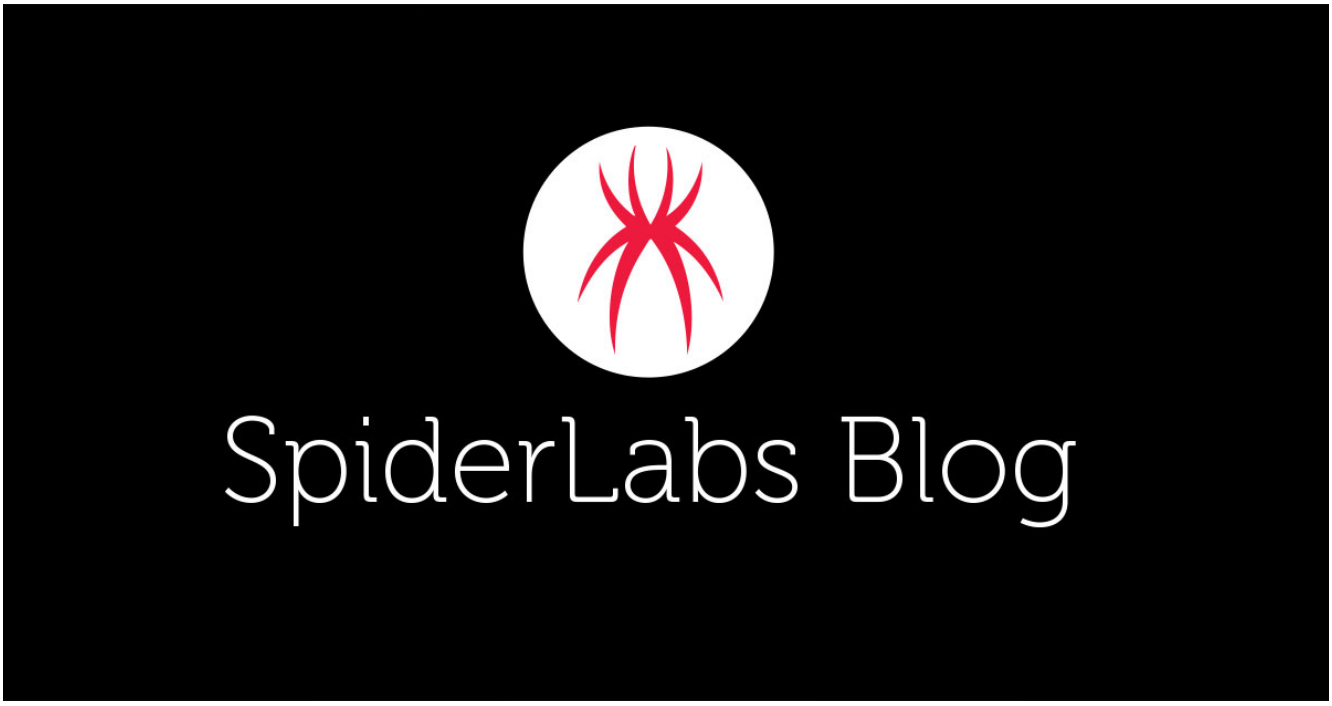
RATs and Spam: The Node.JS QRAT

trustwave.com/en-us/resources/blogs/spiderlabs-blog/rats-and-spam-the-nodejs-qrat/



The Qua or Quaverse Remote Access Trojan (QRAT) is a Java-based RAT that can be used to gain complete control over a system. Introduced in 2015, QRAT was marketed as an undetectable Java RAT and is offered under the software-as-a-service model. Just after its original debut, we blogged about QRATs being spammed. As shown in Figure 1, the functionality of the spammed QRATs can be extended through the plugins offered by Quaverse.

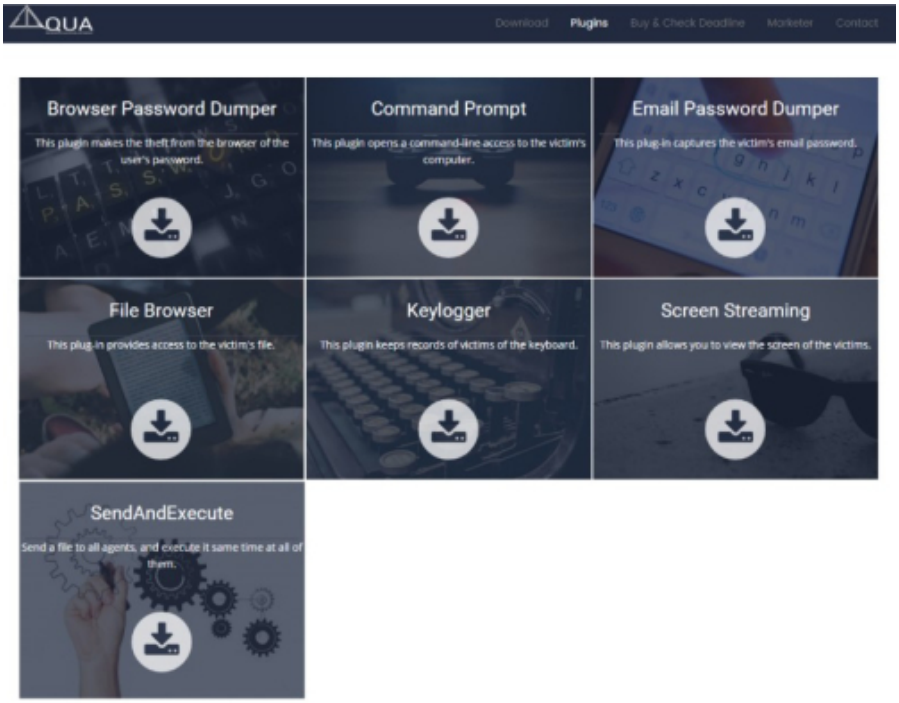


Figure 1: The website of Quaverse, captured from the 2015 blog, offering plugins for QRATs

Recently, we have encountered more spam campaigns that attempt to spread QRATs. The initial malware contains a subscription account for QHub (*user:<digits>@qhub-subscription[.]store[.]qua[.]one)*, a service that offers a single interface to control remote machines. Its domain *qua.one* contains the same logo as to Quaverse's website we have seen way back in 2015.

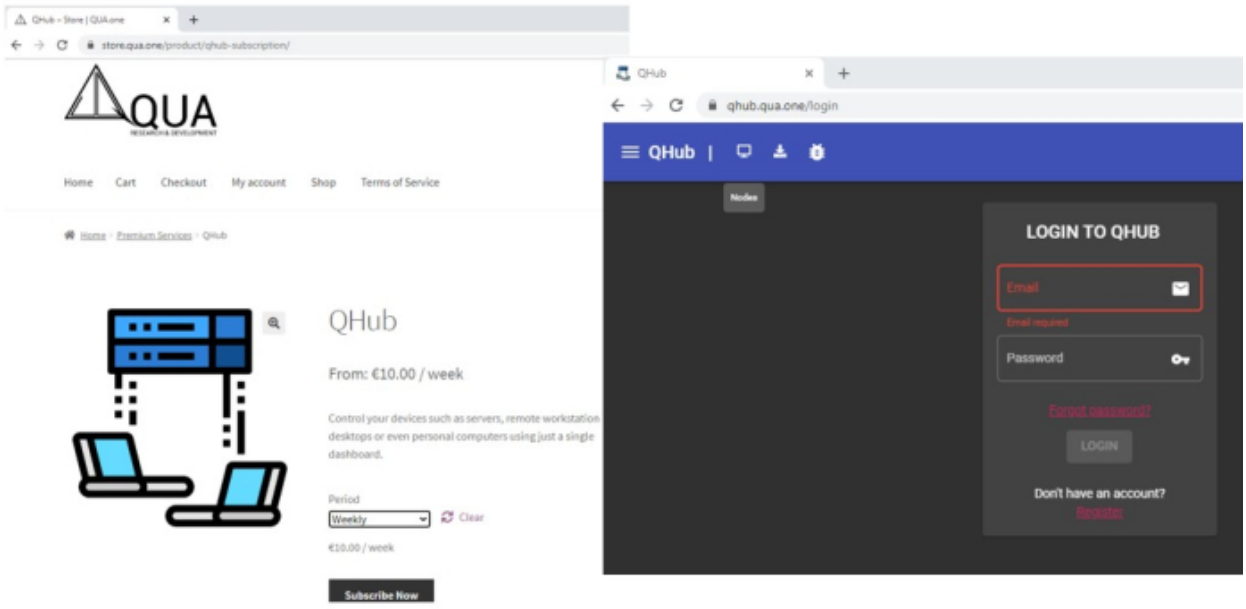


Figure 2: The QHub service is offered as a Premium Service

Malware Analysis

Figure 3: The spam campaign flow

The JAR Downloader

This spam campaign using QRAT malware has multi-stage downloaders. The first one is a JAR file that may arrive as an email attachment or can be downloaded from a link contained in a spam message. All the JAR files we have collected related to this campaign are obfuscated using the Allatori Obfuscator – the class names all have the same name and length but have different case.

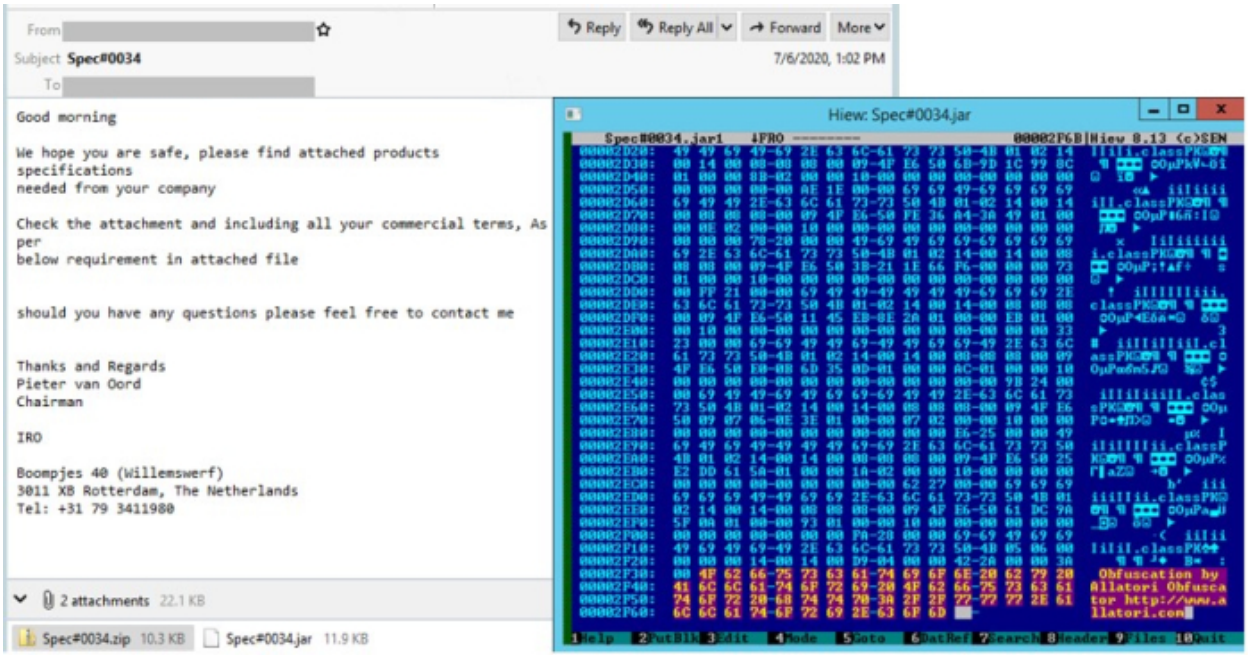


Figure 4: The JAR attachment Spec#0034.jar is obfuscated with Allatori

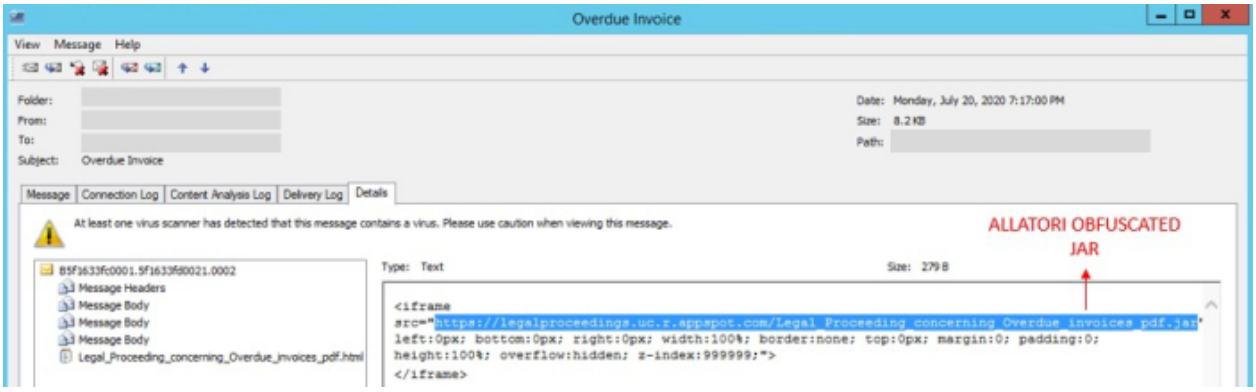


Figure 5: The HTML downloader attached to the malspam has link to the first stage downloader hosted in a cloud service platform

The first downloader has 2 major functions. These are setting up the Node.Js platform onto the system, and then downloading and executing the second-stage downloader.

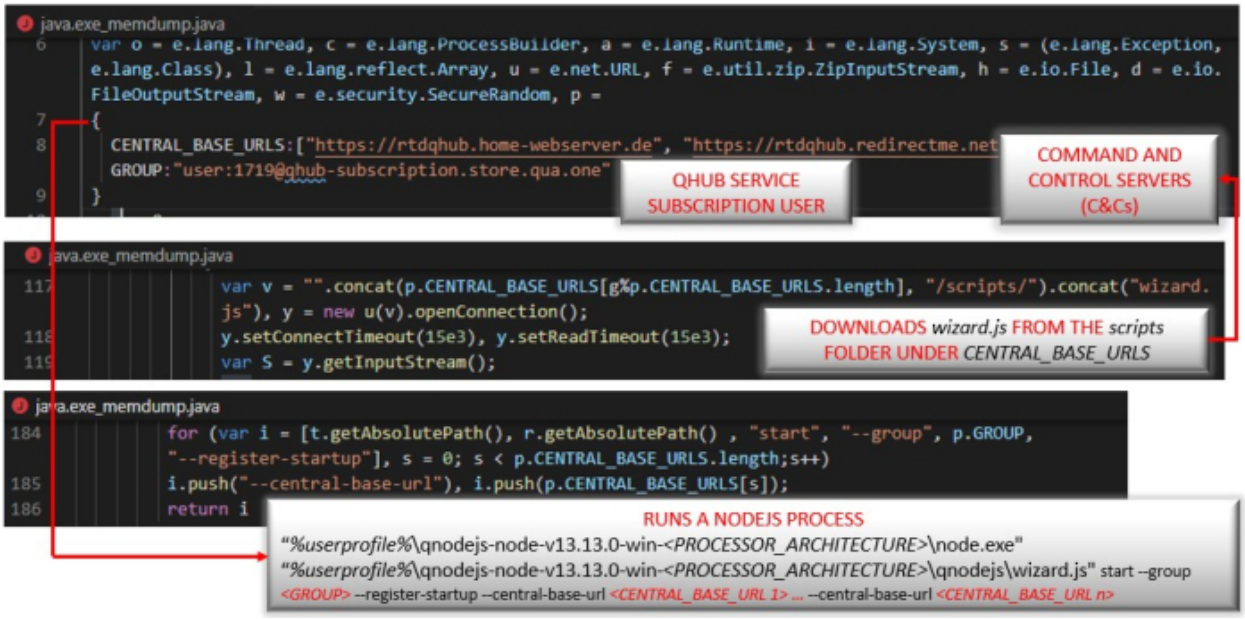


Figure 6: The code snippet of java.exe's memory dump when the attachment Spec#0034.jar from Figure 4 was executed

Firstly, upon the execution of the JAR file, the process architecture of the system will be checked, and that information will be used in downloading the appropriate Node.js for the machine. The JAR files we observed downloaded Node.Js version 13.13.0 from *https://nodejs.org/dist/v13.13.0/node-v13.13.0-win-<x86|x64>.zip* and extracted its content at *%userprofile%/qnode-node-v13.13.0-win-<x86|x64>.zip.* The JAR files were designed to run in Windows environments only.

Secondly, the JAR file Spec*#0034.jar* downloaded *wizard.js* from its command and control servers (C&Cs) then saved it under the *qnodejs* folder located inside the Node.js installation path. Then, the JAR file executed *wizard.js* with the C&Cs and the QHub service subscription user as arguments, as shown in Figure 6.

The Node.Js Downloader

The downloaded file *wizard.js* is the second stage downloader written in Node.Js. This script file is responsible for setting the persistence of this threat, and the downloading and execution of the payload. Just like the JAR file, this supports the Windows platform only. Without the arguments from the JAR file, this script will not work.

We were able to obtain the file *wizard.js*, from *hxxps://environment[.]theworkpc[.]com/scripts/wizard[.]js,* through the JAR file from Figure 5. The file *wizard.js* is encrypted using Base64. Looking through its decrypted code, this script has its own defined modules.

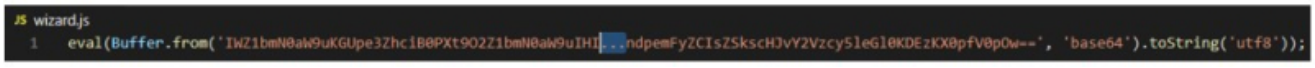


Figure 7: The code snippet of the downloaded second stage downloader

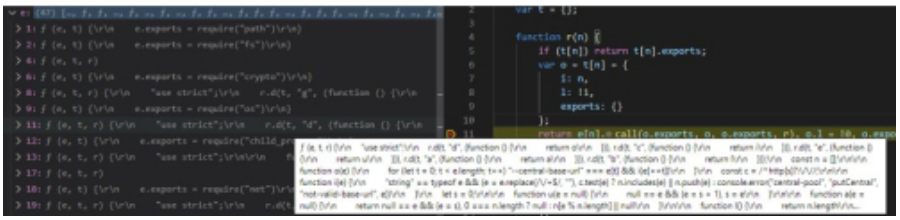


Figure 8: The modules of wizard.js

The first main function of *wizard.js* is to set its persistence. The file *qnode-<8 hex>*.cmd will serve as the autorun file and written in it are the same arguments that the JAR file downloader set on *wizard.js* (see Figure 6) appended with a "*--delegate"* command.

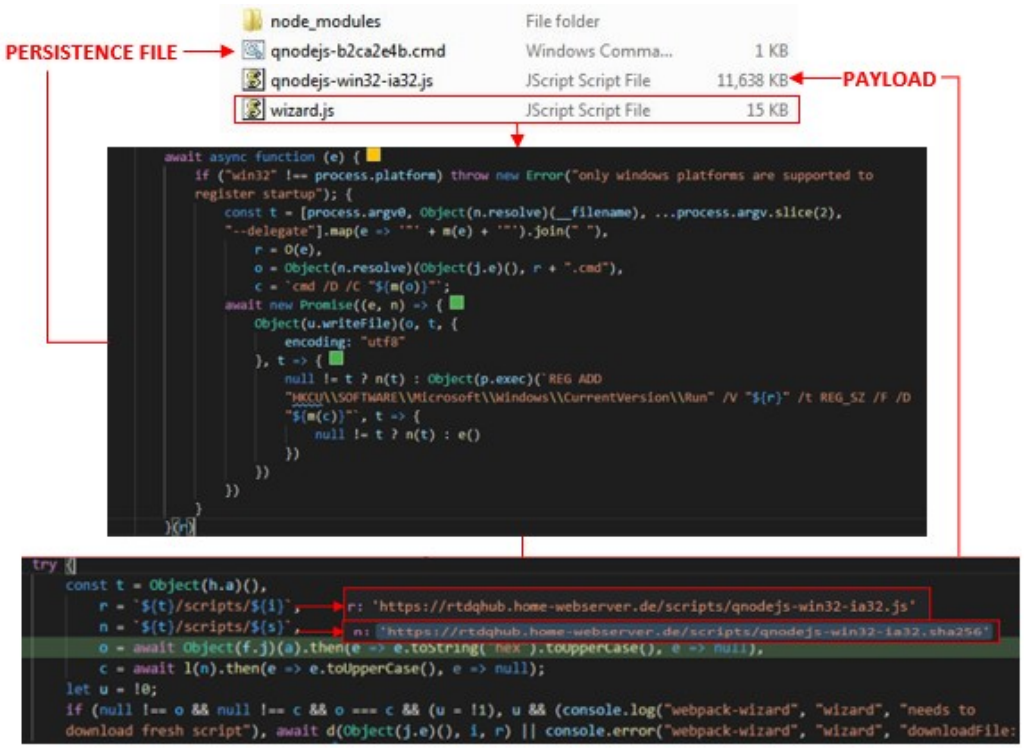


Figure 9: The two main functions of wizard.js

Then, based on the platform and architecture of the system its running on, *wizard.js* will download the main malware. Using the C&Cs from the JAR file in Figure 4, we were able to download *qnodejs-win32-ia32.js* on 24-July-2020.

Before downloading, as shown in Figure 9, the script *wizard.js* verified the sha1 of the main malware *qnodejs-win32-ia32.js* through the file *qnodejs-win32-ia32.sha256*. Lastly, the main malware was executed using the same arguments supplied to *wizard.js* in Figure 6 plus a *"serve"* command after the path of *qnodejs-win32-ia32.js.*

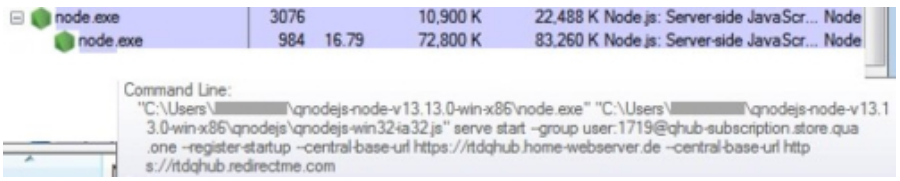


Figure 10: The Node.Js process which executes the payload qnodejs-win32-ia32.js

The Payload – Node.Js QRAT

Just like the downloader *wizard.js*, the main payload *qnode-win32-ia32.js* is written in Node.Js, its code is encrypted with Base64, and it has its own written modules. It contains an encrypted Node.Js packages folder *node_modules* hence its size is almost 12KB.

The Node.Js script uses the string “qnode-service” as the node command name and requires the arguments --*central-base-url* and --*group* when executed.

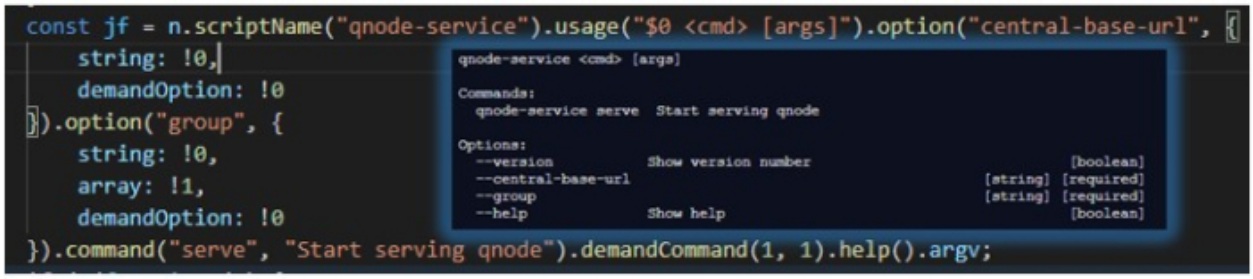


Figure 11: The help menu of the qnode-win32-ia32.js

The QRAT *qnode-win32-ia32.js* has the following functionalities:

- obtain system information
- perform file operations
- acquire credentials of certain applications

Some of the information, like the machine’s UUID, tags, and labels generated by the malware, will be written in the config file *%userprofile%/<hex>-config.json*. Meanwhile, the malware will use the service *hxxps://wtfismyip[.]com* to obtain network information.

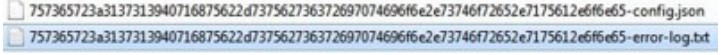


Figure 12: The filename of the config and error logs were prepended with the hex representation of the QHub subscription account shown in Figure 6

Figure 13: Code snippet of the data retrieved from the service hxxps://wtfismyip[.]com/json

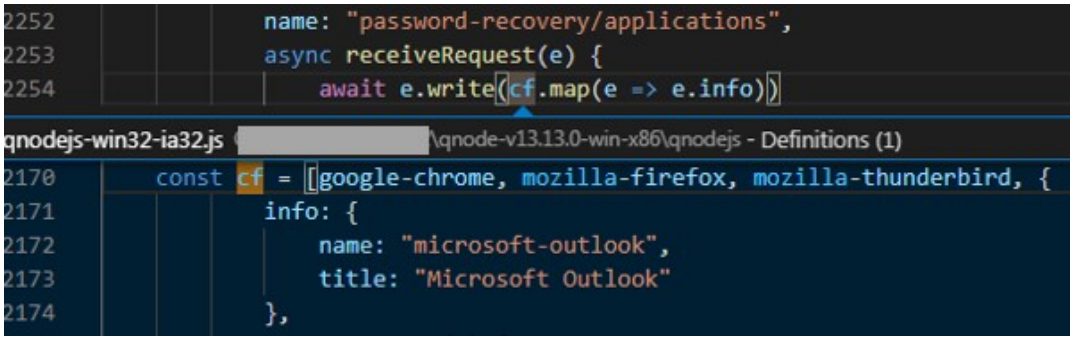


Figure 14: The applications chrome, firefox, thunderbird, and outlook are supported in this QRAT's password-recovery functionality

This threat will communicate to its C&Cs through the Websocket connection protocol and below is the list of commands related to the 3 functions of the QRAT *qnode-win32-ia32.js* file mentioned above.

info/get-machine-uuid	get the machine UUID generated by the malware
info/get-label	get the labels generated by the set-label command
info/set-label	set label
info/get-os-name	get the plaform and the architecture of the infected machine
info/get-ip-address	get the IPAddress obtained from hxxps://wtfismyip[.]com
info/get-location	get the all network information obtained from the service hxxps://wtfismyip[.]com/
info/get-user-home	get the path of the current user's home directory
info/get-tags	get tags
info/add-tag	add tags
info/remove-tag	remove tags
file-manager/absolute	retrieve the aboslute path of a file
file-manager/execute	execute a file together with a platform dependent command
file-manager/delete	delete a file
file-manager/mkdirs	add a directory
file-manager/list	list files in a directory
file-manager/forward-access	generate a forwarding URL
file-manager/write	write a file onto the infected system
password-recovery/applications	list the supported applications in the system: google-chrome, mozilla-firefox, mozilla-thunderbird, microsoft-outlook
password-recovery/recover	password recovery utility for the supported applications

Figure 15: List of commands

Summary

Remote access trojans are one of the commodity malware nowadays. With services like QHub, RATs can be a more attractive instrument to the threat actors as the machines infected by the RATs can be easily monitored in an already available environment they offer. The QRAT and its downloader are currently supporting the windows platform for now. Since they leveraged Node.Js which is a cross-platform, there is a possibility that this threat will be enhanced to support other platforms in the future.

In terms of mitigation, we recommend blocking inbound emails with Java files outright at the email gateway. We have also added protection for this threat to the Trustwave Secure Email Gateway for our customers.

IOCs

Spec#0034.jar (12139 bytes) SHA1: 36DA7F23828283B6EA323A46806811F8312DD468
Legal_Proceeding_concerning_Overdue_invoices_pdf.jar (12,241 bytes) SHA1: 42d843c74e304d91297e21e748f4b528df422316
wizard.js (14433 bytes) SHA1: D6B1D3317C0D938C8AF21F1C22FD1B338A06B1C2
qnodejs-win32-ia32.js (11916833 bytes) SHA1: 31F541074C73D02218584DF6C8292B80E6C1FF7D

hxxps://legalproceedings[.]uc[.]r[.]appspot[.]com/Legal_Proceeding_concerning_Overdue_invoices_pdf[.]jar
hxxps://rtdqhub[.]home-webserver[.]de/
hxxps://rtdqhub[.]redirectme[.]net
hxxps://environment[.]theworkpc[.]com
hxxps://environment[.]spdns[.]org