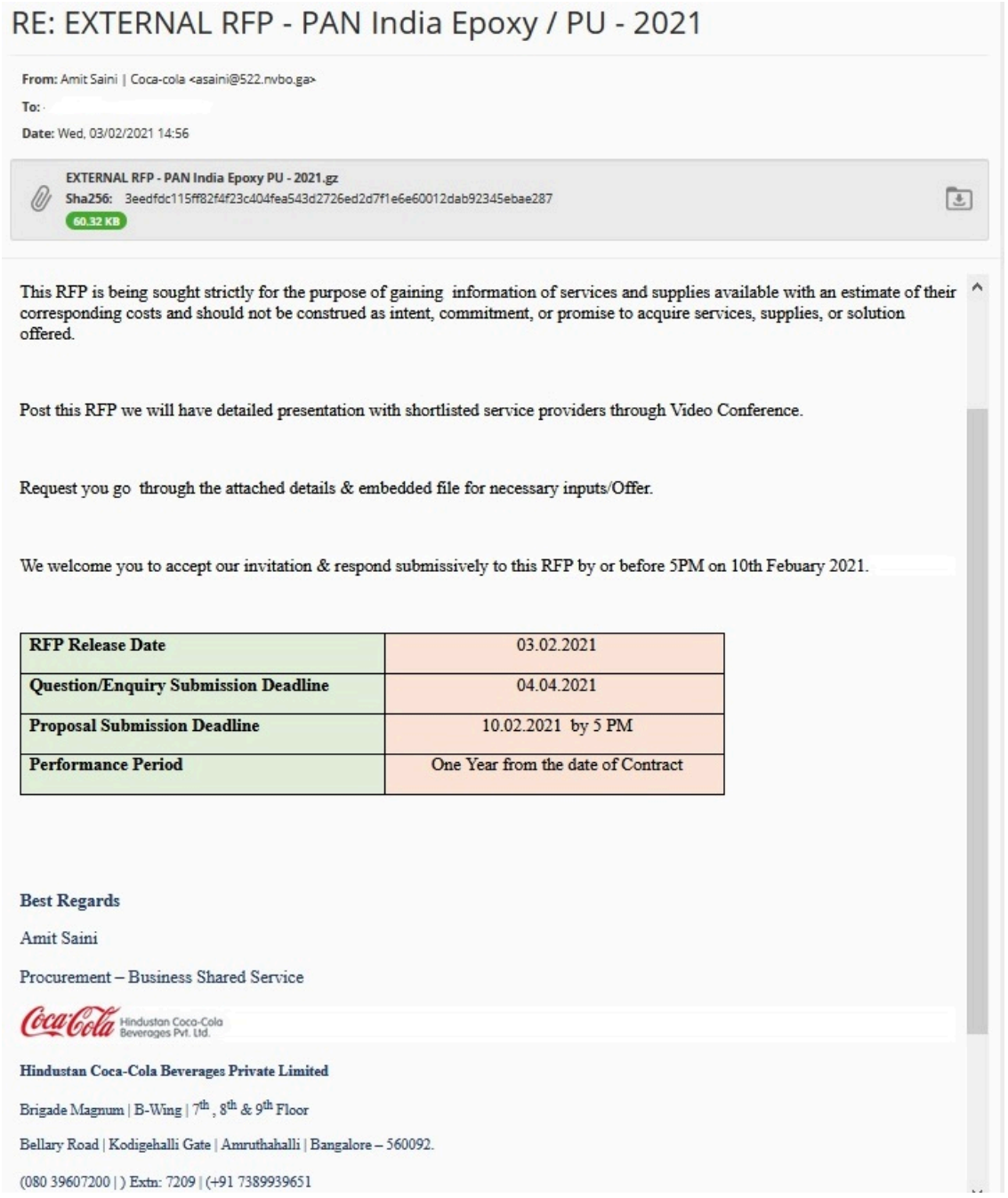
GuLoader Snowballs via MalSpam Campaigns

Published: 2021-02-17 · Archived: 2026-04-02 12:45:48 UTC

GuLoader is one of the well-known **downloader malware** of 2020, as its prevalence was very high during the first half of the year. Its common payloads were **FormBook**, **Agent Tesla**, **LokiBot**, **Remcos RAT**, just to name a few, which were delivered by abusing storage services like **OneDrive**, **Google Drive** etc. Our 1st encounter with the GuLoader binary was in March 2020 when it was delivering FormBook in a **spam campaign**. Later, Check Point revealed their findings about the similarities between GuLoader and **CloudEye**, a protector for binaries.

Recently, we got our hands on the latest GuLoader binary which was submitted to bazzar[.]abuse[.]ch by JAMESWT (@JAMESWT_MHT). It came as an email attachment as depicted in Figure 1. The email seemed interesting because the sender's name was Amit Saini claiming to be **from Coca-Cola, Bangalore, India**.



RE: EXTERNAL RFP - PAN India Epoxy / PU - 2021

From: Amit Saini | Coca-cola <asaini@522.nvbo.ga>

To:

Date: Wed, 03/02/2021 14:56

EXTERNAL RFP - PAN India Epoxy PU - 2021.gz
Sha256: 3eedfdc115ff82f4f23c404fea543d2726ed2d7f1e6e60012dab92345ebae287
60.32 KB

This RFP is being sought strictly for the purpose of gaining information of services and supplies available with an estimate of their corresponding costs and should not be construed as intent, commitment, or promise to acquire services, supplies, or solution offered.

Post this RFP we will have detailed presentation with shortlisted service providers through Video Conference.

Request you go through the attached details & embedded file for necessary inputs/Offer.

We welcome you to accept our invitation & respond submissively to this RFP by or before 5PM on 10th Febuary 2021.

RFP Release Date	03.02.2021
Question/Enquiry Submission Deadline	04.04.2021
Proposal Submission Deadline	10.02.2021 by 5 PM
Performance Period	One Year from the date of Contract

Best Regards

Amit Saini

Procurement – Business Shared Service

Hindustan Coca-Cola Beverages Pvt. Ltd.

Hindustan Coca-Cola Beverages Private Limited

Brigade Magnum | B-Wing | 7th , 8th & 9th Floor

Bellary Road | Kodigehalli Gate | Amruthahalli | Bangalore – 560092.

(080 39607200 |) Extn: 7209 | (+91 7389939651

Figure 1: Email with Attachment (courtesy of **@JAMESWT_MHT**)

The infection vector hasn't changed yet but we at **K7 Labs** still keep track of GuLoader because of the efforts taken by them to keep improving their code for detecting the Virtual/Debug environment. Although some of the tricks are old, they still get the job done. In this blog, we'll see the improvements that have been made to the code over time.

Anti-Analysis & Anti-VM/Debug Techniques

GuLoader in March 2020

1. Debugger Anti-Attach technique – using **ntdll.ZwSetInformationThread()** with parameter **0x11**
2. Patching **ntdll.DbgBreakPoint()** and **ntdll.DbgUiRemoteBreakin()**

3. Patching User mode hooks – patching the 1st 5 bytes of unconditional jump (0xe9 ????????) set by some AV & sandboxes

GuLoader after July 2020

In addition to previous techniques mentioned above, there were some more tricks found in the binary which was received after the end of June and they are

1. **ZwQueryVirtualMemory()** – to detect execution with in virtual machine
2. Check breakpoints
3. Enumerating the active windows using **EnumWindows()** API
4. Checking for **qemu-ga.exe** and **qga.exe** under Program Files.

While all these were documented tricks there are 2 tricks in particular which were quite interesting to us.

- RDTSC and CPUID instruction combination as depicted in Figure 2.

It uses RDTSC instruction to get the elapsed time in EAX:EDX and performs OR operation between EAX & EDX and saves it in ESI. Then it calls CPUID instruction with EAX=1 and checks if the 31st bit (0x1f) is set (by default it is 0 & if run under virtual machine it will be set) and then exits execution by displaying a popup message stating **“The program cannot be run under virtual Environment or debugging software!”.**

Again it calls RDTSC instruction and performs the OR operation between EDX and EAX and subtract the new result with the previous result stored in ESI. In normal execution, the difference between 2 RDTSC instructions will never be 0, but the code checks if the difference is less than or equal to 0 which results in an endless loop.

```
003C3FC7   0F AE E8          lfence
003C3FCA   0F 31             rdtsc                  Read Timestamp counter
003C3FCC   0F AE E8          lfence
003C3FCF   C1 E2 20          shl edx,20
003C3FD2   09 C2             or edx,eax
003C3FD4   89 D6             mov esi,edx
003C3FD6   60                pushal
003C3FD7   B8 01 00 00 00    mov eax,1
003C3FDC   0F A2             cpuid                  cpuid when eax=1, returns feature flag in ecx,edx
003C3FDE   0F BA E1 1F       bt ecx,1F              Checks if 31st bit is Set (feature flag)
003C3FE2   0F 82 D6 3C 00 00 jb 3C7CBE              31st bit is always zero, if it is set then debugger/vm present
003C3FE8   61                popal
003C3FE9   0F AE E8          lfence
003C3FEC   0F 31             rdtsc                  Read Timestamp counter
003C3FEE   0F AE E8          lfence
003C3FF1   C1 E2 20          shl edx,20
003C3FF4   09 C2             or edx,eax
003C3FF6   29 F2             sub edx,esi            Time difference between 2 rdtsc instruction
003C3FF8   83 FA 00          cmp edx,0              diff between 2 rdtsc will not be less than or equal to zero
003C3FFB   7E CA             jle 3C3FC7
003C3FFD   C3                ret
```

Figure 2: RDTSC and CPUID Instructions

- Apart from the infinite loop mentioned above, it also uses one more loop which executes for **0x186a0** times (that is 100000 times). The value 0x186a0 is stored in ECX and performs addition between EDI (EDI=0 initially) and the result received after the difference between two RTDSC instructions (mentioned above). This loop is executed till ECX becomes 0 and if the value in EDI after the loop ends is greater than **0x68e7780** it again returns to the start of the check where it again sets ECX to 0x186a0.

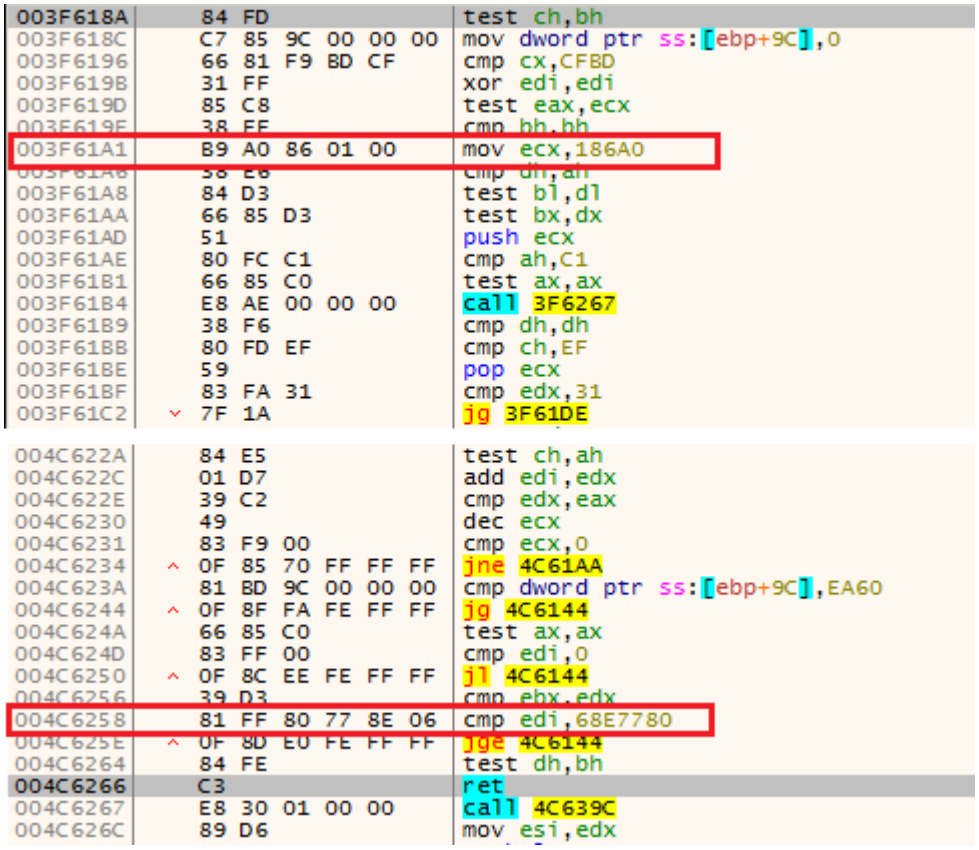


Figure 3: RDTSC loop

- Retrieves the name of the active window and creates a hash with it and matches it with the predefined hash stored in the code as depicted in Figure 4.

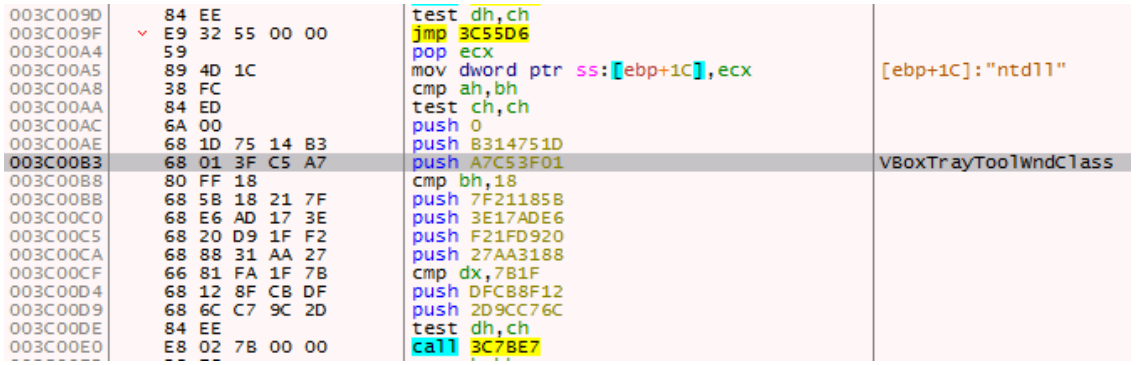


Figure 4: Hash Comparison of the Active Window

GuLoader 2021

The GuLoader sample which was analyzed recently had almost every check mentioned above except for the active window hash comparison. Instead they have a different hash comparison technique.

- Using **MsiEnumProductsA()** and **MsiGetProductInfo()** function

First it calls MsiEnumProductsA() function with iPoductIndex as 0 and increments it by 1 for subsequent calls. It returns a product code which is a 38 character GUID with a null terminating character making it 39 character long.

This GUID is given as input to MsiGetProductInfo() function to retrieve the product name installed and this loop is executed for 0xff times as depicted in Figure 5.

```
003E0A0C    57                  push edi
003E0A0D    66 51               push cx
003E0A0F    66 B9 C5 BF         mov cx,BFC5
003E0A13    66 59               pop cx
003E0A15    56                  push esi
003E0A16    84 EF               test bh,ch
003E0A18    FF 95 00 01 00 00   call dword ptr ss:[ebp+100]   [ebp+100]:MsiEnumProductsA
003E0A1E    83 F8 00            cmp eax,0
003E0A21    0F 85 51 01 00 00   jne 3E0B78
003E0A27    90                  nop
003E0A28    EB 00               jmp 3E0A2A                   jmp $0
003E0A2A    E8 0C 00 00 00      call 3E0A3B
003E0A2F    50                  push eax

004C0AA9    81 EF FF 00 00 00   sub edi,FF                   edi:"{4A03706F-666A-4037-7777-5F2748764D10}"
004C0AAF    57                  push edi                     edi:"{4A03706F-666A-4037-7777-5F2748764D10}"
004C0AB0    38 C0               cmp al,al
004C0AB2    FF 95 04 01 00 00   call dword ptr ss:[ebp+104]  [ebp+104]:MsiGetProductInfoA
004C0AB8    66 39 C8            cmp ax,cx
004C0ABB    56                  push esi
004C0ABC    38 F6               cmp dh,dh
004C0ABE    81 C7 FF 00 00 00   add edi,FF
004C0AC4    57                  push edi
004C0AC5    F6 C1 08            test cl,8
004C0AC8    E8 B0 51 00 00      call 4C5C7D                  Hash calculation
004C0ACD    5E                  pop esi
004C0ACE    81 FA 68 AC BB 73   cmp edx,73BBAC68
004C0AD4    3D FD A9 8A 7C      cmp eax,7C8AA9FD
004C0AD9    0F 84 3A 5D 00 00   je 4C6819
004C0ADF    66 81 FE 5B 19      cmp si,195B
004C0AE4    3D 51 FB 8F 9B      cmp eax,9B8FFB51
004C0AE9    0F 84 2A 5D 00 00   je 4C6819
004C0AEF    84 FE               test dh,bh
004C0AF1    3D 91 16 5E 55      cmp eax,555E1691
004C0AF6    0F 84 1D 5D 00 00   je 4C6819
004C0AFC    84 D1               test cl,dl
004C0AFE    3D 5D C8 81 CE      cmp eax,CE81C85D
004C0B03    0F 84 10 5D 00 00   je 4C6819
004C0B09    66 85 D2            test dx,dx
004C0B0C    F6 C1 49            test cl,49
004C0B0F    46                  inc esi
004C0B10    39 D0               cmp eax,edx
004C0B12    38 EC               cmp ah,ch
004C0B14    81 FE FF 00 00 00   cmp esi,FF
004C0B1A    0F 85 DF FE FF FF   jne 4C09FF
004C0B20    EB 48               jmp 4C0B6A
```

Figure 5: MsiEnumProductsA() and MsiGetProductInfo() loop

The result received after a call to MsiGetProductInfo() is the name of the product in strings which needs to be converted to a hash for comparison. This eliminates performance overhead since comparing each character sequentially takes time. The hashing function used here is djb2 as depicted in Figure 6 which is quite simple and lightweight.

```
int FUN_00005c7d(byte *Product_Info)

{
  int iVar1;                          Hashing logic

  iVar1 = 0x1505;
  do {
    if (0xa3 < *Product_Info) {
      return 0;
    }
    iVar1 = iVar1 * 0x21 + (uint)*Product_Info;
    Product_Info = Product_Info + 1;
  } while (*Product_Info != 0);
  return iVar1;
}
```

djb2

this algorithm (k=33) was first reported by dan bernstein many years ago in comp.lang.c. another version of this algorithm (now favored by bernstein) uses xor: `hash(i) = hash(i - 1) * 33 ^ str[i];` the magic of number 33 (why it works better than many other constants, prime or not) has never been adequately explained.

```
unsigned long
hash(unsigned char *str)
{
    unsigned long hash = 5381;
    int c;

    while (c = *str++)
        hash = ((hash << 5) + hash) + c; /* hash * 33 + c */

    return hash;
}
```

Figure 6: Hashing Logic

The hashes used in the code (like **0x7c8aa9fd**, **0x9b8ffb51**) are unknown to us at this point in time but anyone can guess that it must be mostly related to check if AV, sandboxes or debuggers are installed.

- Use of **NtQueryInformationProcess()** with processInformationClass parameter as 0x07 (process debug port) as depicted in Figure 7. This API usage is well documented and is an old trick to detect if the process is being debugged.

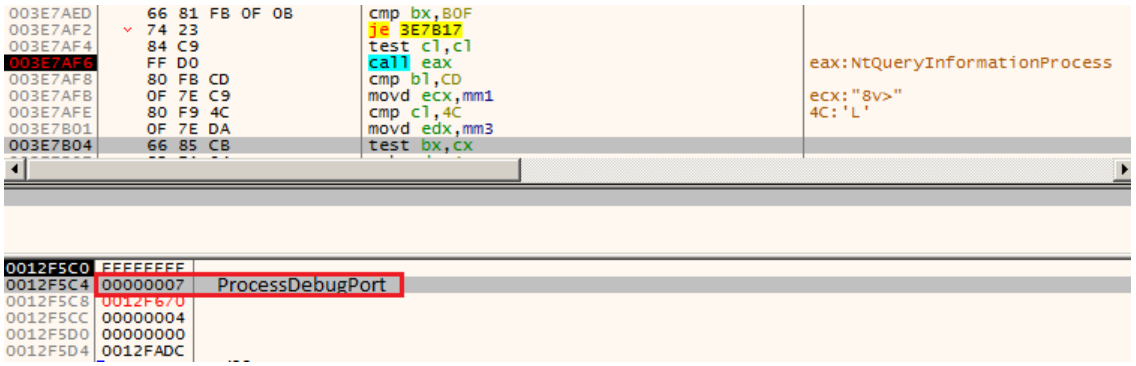


Figure 7: NtQueryInformationProcess() function

- Code implementation changes – to make the process of reversing/debugging a little harder they have implemented **spaghetti code** which is a code having a lot of jumps and calls.

Once all these Anti-VM and Anti-Debugging checks are over it proceeds to download the encrypted binary from the domain stated and copies it to a buffer space and decrypts it as depicted in Figure 8.

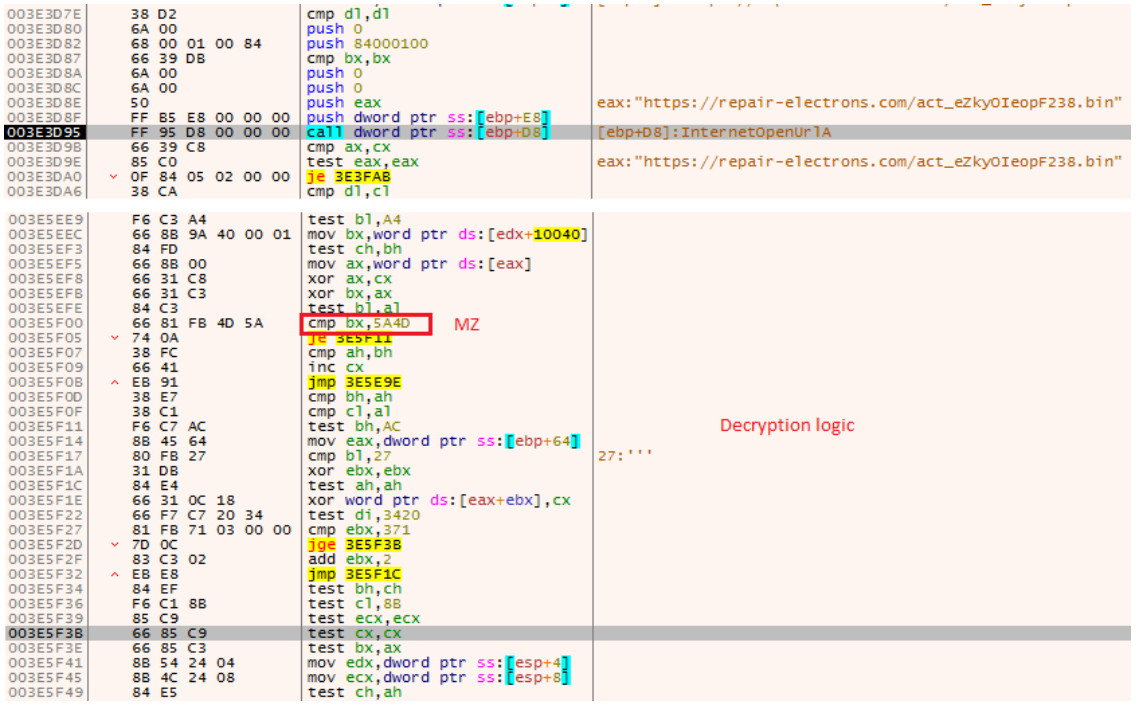


Figure 8: Downloading after Decrypting the Binary

The domain is still live and seems to be bogus because the domain name mentioned in the contact section of the page is **repair-electronics**[.]com whereas the domain name active is **repair-electrons**[.]com and the "**created by Mohamad Chedid**" line under copyright symbol has a HTML href tag, which is blank and doesn't redirect anywhere. When viewing the source of the page there is a commented line saying "**Free HTML5 template developed by FREEHTML5.CO**" as depicted in Figure 9.

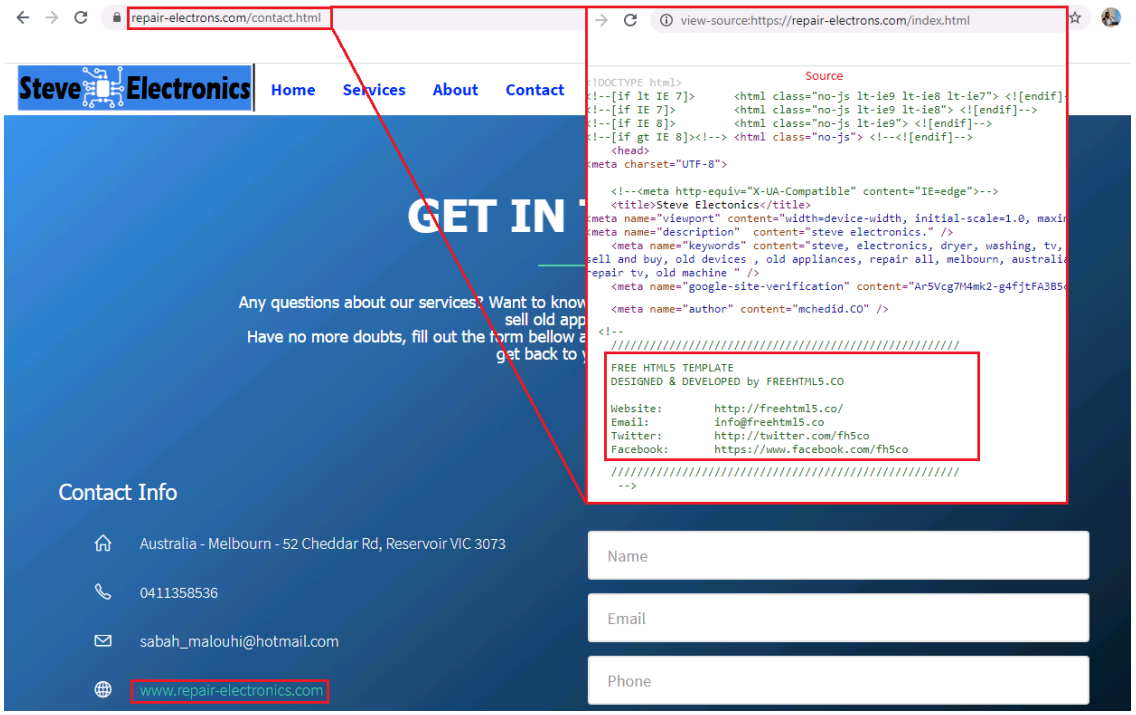


Figure 9: Bogus Domain Name

Threat actors are always evolving by modifying their tools with improved techniques and tricks to evade detection and make the analysis harder. Here at K7 Labs we actively monitor such malware and have proactive detection for all the files. So stay safe from these kinds of attacks in this pandemic situation by using a reputed AV product such as K7 products.

Indicators Of Compromise (IOCs)

MD5: 1C8B24FCF8143C9035EE722EC8714EB0

File Name: EXTERNAL RFP – PAN India Epoxy PU – 2021.exe

K7 Detection Name: Trojan (005774081)

URL

hxxps[:]//www[.]repair-electrons[.]com

Source: https://labs.k7computing.com/?p=21725Lokesh