Qakbot | ThreatLabz

zscaler.com/blogs/security-research/tracking-15-years-qakbot-development

Javier Vicente

Security Research

Introduction

Qakbot (aka QBot or Pinkslipbot) is a malware trojan that has been used to operate one of the oldest and longest running cybercriminal enterprises. Qakbot has evolved from a banking trojan to a malware implant that can be used for lateral movement and the eventual deployment of ransomware. In August 2023, the Qakbot infrastructure was dismantled by law enforcement. However, just several months later in December 2023, the fifth (and latest) version of Qakbot was released, marking more than 15 years of development. In this blog, we will analyze Qakbot from the first version dating back to 2008 through the most recent version that continues to be updated as of January 2024. Our analysis demonstrates the threat actor behind Qakbot is resilient, persistent, and innovative.

Key Takeaways

- Qakbot originated in 2008 as a banking trojan designed to steal credentials and conduct ACH, wire, and credit card fraud.
- In recent years, Qakbot has become an initial access broker delivering Cobalt Strike for lateral movement and ultimately resulting in second-stage infections including ransomware like BlackBasta.
- Over the years, Qakbot's anti-analysis techniques have improved to evade malware sandboxes, antivirus software, and other security products.
- The malware is modular and can download plugins that enable it to dynamically add new functionality.
- The threat group behind Qakbot has now released five distinct versions of the malware with the latest release in December 2023.

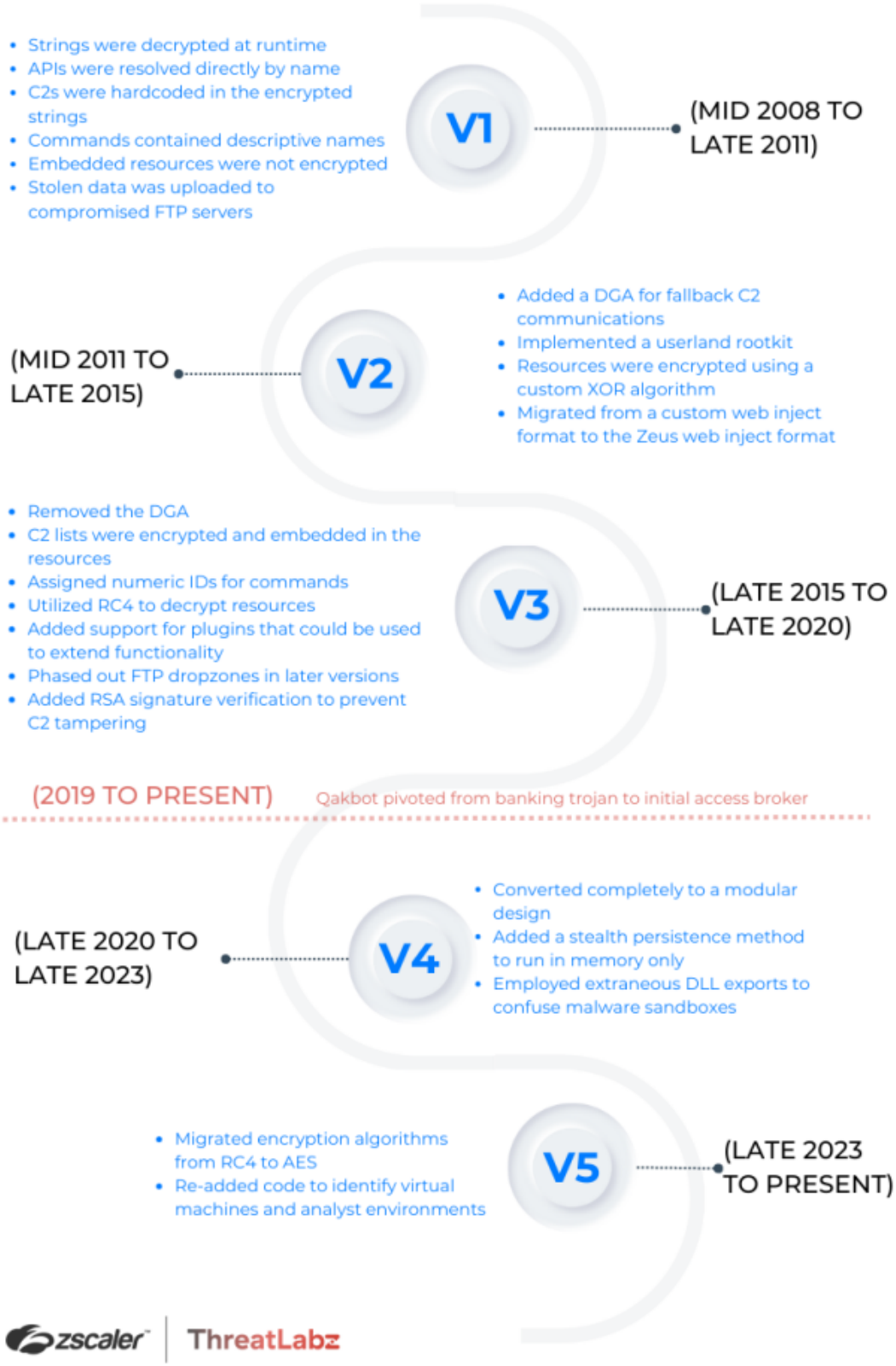
A Brief History of Qakbot

ThreatLabz researchers have been tracking Qakbot for more than a decade and our analysis started with samples that date back to 2008. These early versions of Qakbot contained a date timestamp rather than a version number. However, we will refer to these samples as version 1.0.0 for clarity and consistency with subsequent versions. At that time, Qakbot leveraged a dropper with two embedded components in the resource section that consisted of a malicious DLL and a tool to inject the DLL into running processes. The Qakbot DLL implemented a wide variety of features including: a SOCKS5 server, stealing passwords, harvesting web browser cookies, and spreading via SMB. These early versions were heavily developed and even had a feature to report crash dumps. In 2011, Qakbot introduced a versioning system that started with 2.0.0 that has signified major developmental milestones over time. The Qakbot major version number is a three-digit hexadecimal value with 0x500 (or 5.0.0) being the most recent.

Qakbot was largely used for banking fraud until 2019, when the threat actor pivoted to serving as an initial access broker for ransomware including Conti, ProLock, Egregor, REvil, MegaCortex, and BlackBasta.

The following timeline illustrates the key developments for each version of Qakbot.

Evolution of Qakbot



zscaler | ThreatLabz

Each version of Qakbot represents a snapshot in time and is indicative of the threat landscape during that period. For instance, early versions contained hardcoded command-and-control (C2) servers. As time progressed, law enforcement and malware researchers worked successfully with domain registrars to suspend malicious domains. In response, the Qakbot threat actor added network encryption and implemented a solution to remove the C2 server's single point of failure by adding a domain generation algorithm (DGA). While a DGA addressed the single point of failure issue, it also created significant noise when querying for a large number of domains. As a result, the Qakbot developer devised a new multi-tiered architecture that leveraged compromised systems to act as proxy servers that relay network traffic between other infected systems and the backend C2 infrastructure. This design update addressed the single point of failure problem, reduced network traffic, and effectively hid the subsequent C2 tiers.

In the following sections, we will analyze key areas where Qakbot has evolved significantly including anti-analysis techniques, network communication, and the implementation of a modular design.

Anti-Analysis Techniques

Qakbot has implemented anti-analysis techniques from the beginning of its development including string obfuscation, API obfuscation, and malware sandbox evasion.

String obfuscation

Every version of Qakbot since its inception has obfuscated the malware's important strings with a simple XOR algorithm. The XOR key (and most recently, the derivation of an XOR key) is used to decrypt strings. Moreover, the reference structure to the strings has also evolved across versions.

In the first two versions (1.0 and 2.0), the malware decrypted a block of strings from the data section, overwriting the original encrypted block, and the unencrypted strings remained in memory as shown in Figure 1. This simple design was likely an attempt to evade static antivirus signatures.

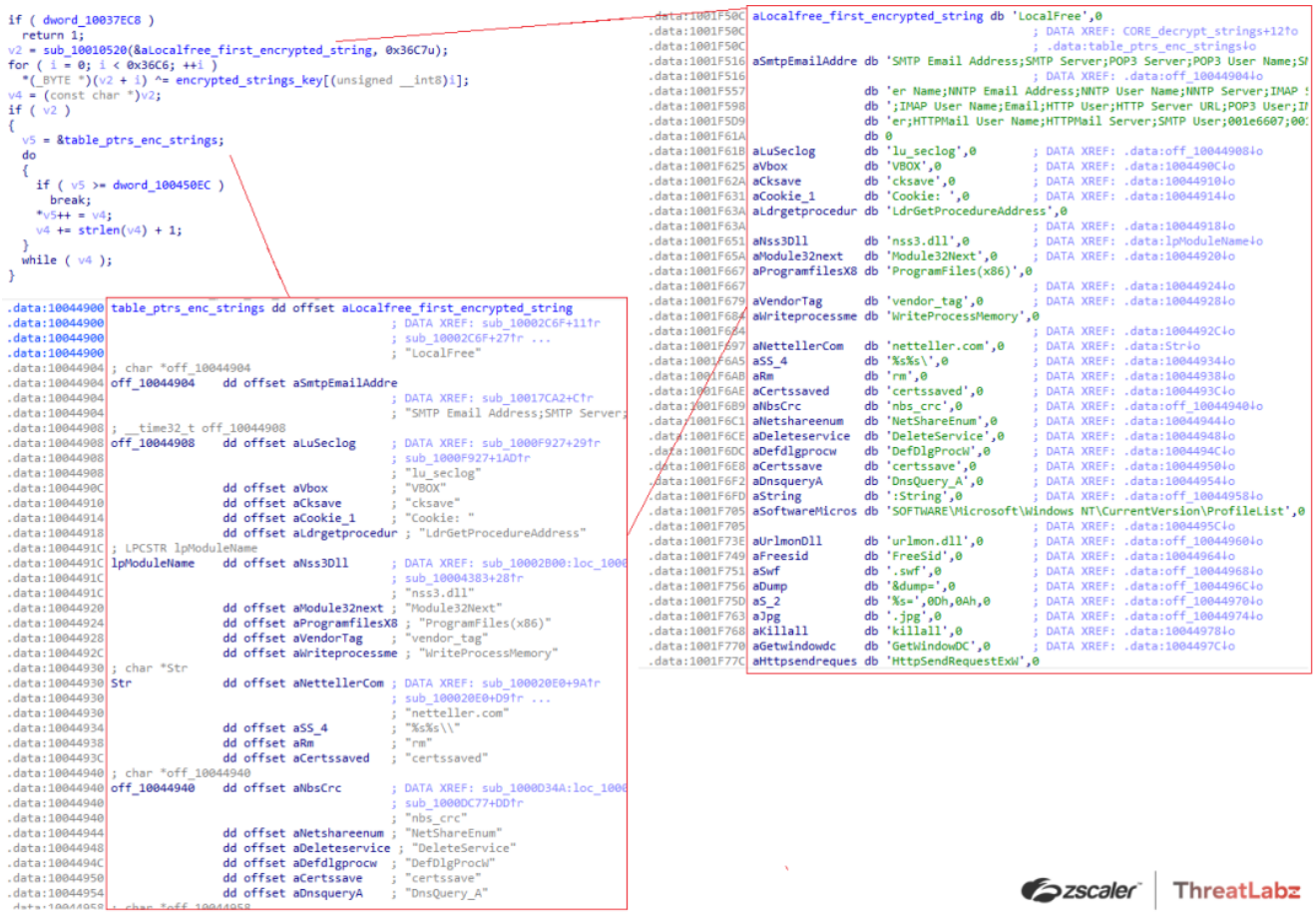


Figure 1. Early versions of Qakbot string obfuscation

In later versions of Qakbot, the XOR key length was significantly increased, and strings were decrypted and copied to a newly allocated buffer. Qakbot version 5.0 made perhaps the most significant change to the string encryption algorithm. The strings are still encrypted with a simple XOR key. However, the XOR key is no longer hardcoded in the data section. Instead the XOR key is encrypted with AES, where the AES key is derived by performing a SHA256 hash of a buffer. A second buffer contains the AES initialization vector (IV) as the first 16 bytes, followed by the AES-encrypted XOR key. Once the XOR key has been decrypted, the block of encrypted strings can then be decrypted as shown in Figure 2.

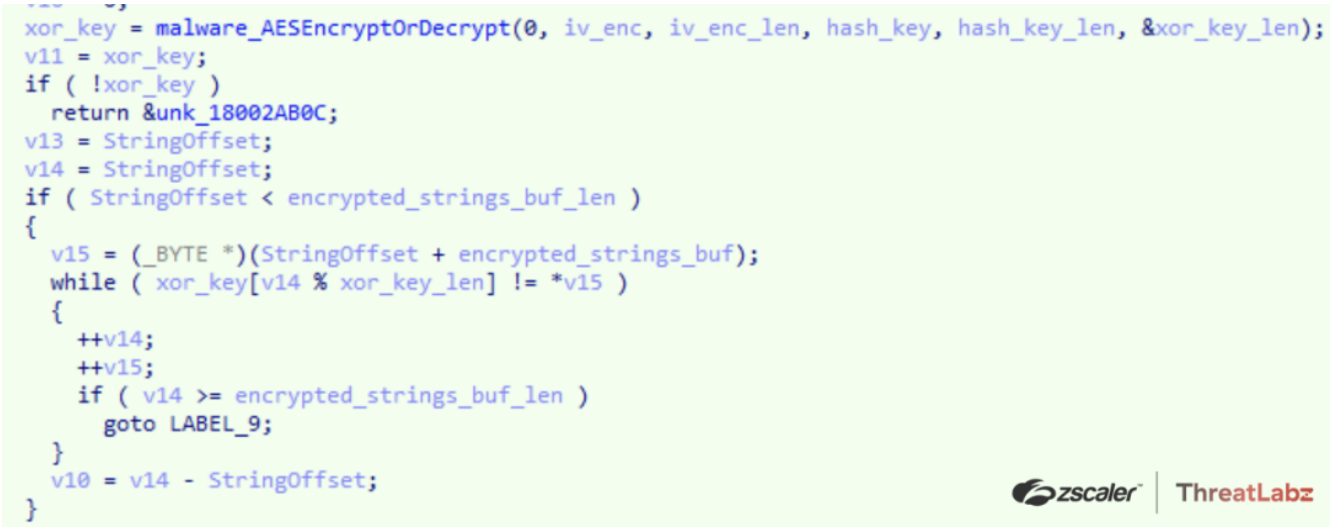


Figure 2. Qakbot 5.0 string decryption

API obfuscation

In versions 1 and 2, Qakbot carried a list of Windows API names used by the malware in the encrypted strings table. After the strings table was decrypted, the code would dynamically resolve the address of each API at runtime and then initialize a table of pointers that could then be used by Qakbot to invoke the corresponding function when required. This implementation made it harder for malware researchers and antivirus software to statically determine the APIs used at runtime.

In more modern versions, the Qakbot developer further obfuscated the use of APIs by resolving the imports using a CRC32 hash rather than a string. At first, Qakbot used the CRC hashes of the API name directly, and subsequent versions performed an XOR with a hardcoded value and the CRC hash. Figure 3 shows an example of this dynamic API import hashing algorithm.

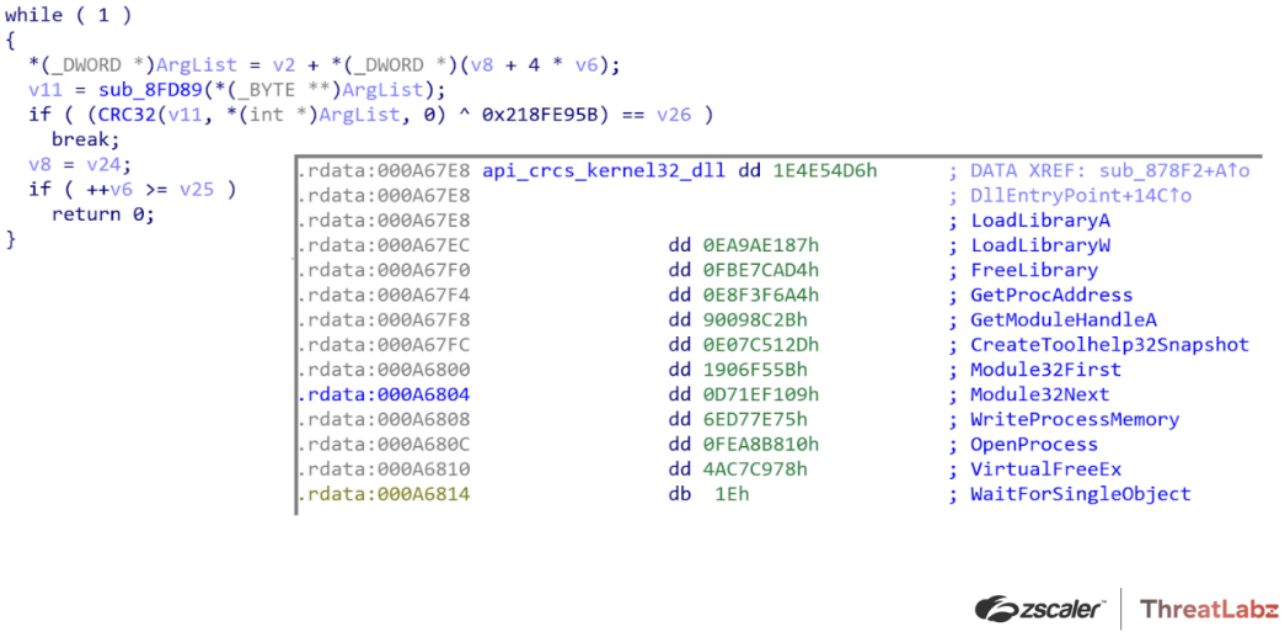


Figure 3. Example Qakbot API obfuscation

Junk code

Over time, Qakbot has introduced blocks of code that are deliberately non-functional to defeat static antivirus signatures as shown in Figure 4. In the example below, a block of junk code was added prior to an RC4 initialization routine.

```
int __fastcall mw_rc4_prga(_BYTE *ptr_data, int a2, __int64 key)
{
  __int64 v4; // rsi
  const char *v6; // rax
  unsigned int v7; // r9d
  unsigned int v8; // ecx
  WCHAR *v9; // rdx
  char v10; // al
  __int64 v11; // rax
  unsigned __int8 v12; // r9
  __int64 v13; // r11
  unsigned __int8 v14; // r10
  char v15; // dl
  WCHAR String[7]; // [rsp+20h] [rbp-18h] BYREF
  char v18; // [rsp+2Fh] [rbp-9h]

  v4 = a2;
  v6 = "business.doc";
  v7 = 0;
  do
  {
    ++v6;
    ++v7;
  }
  while ( *v6 );
  v18 = 0;
  if ( v7 > 0xF )
    v7 = 15;
  v8 = 0;
  if ( v7 )
  {
    v9 = String;
    do
    {
      v10 = v8++ + 65;
      *(_BYTE *)v9 = v10;
      v9 = (WCHAR *)((char *)v9 + 1);
    }
    while ( v8 < v7 );
  }
  LODWORD(v11) = lstrlenW(String);
  v12 = *(_BYTE *)(key + 256);
  v13 = v4;
  v14 = *(_BYTE *)(key + 257);
  if ( (int)v4 > 0 )
  {
    do
    {
      v15 = *(_BYTE *)(++v12 + key);
      v14 += v15;
      *(_BYTE *)(v12 + key) = *(_BYTE *)(v14 + key);
      *(_BYTE *)(v14 + key) = v15;
      v11 = (unsigned __int8)(*(_BYTE *)(v12 + key) + v15);
      *ptr_data++ ^= *(_BYTE *)(v11 + key);
```

zscaler | ThreatLabz

Figure 4. Example of Qakbot junk code block in an RC4 initialization function

Anti-sandbox techniques

Qakbot has implemented numerous detection mechanisms to identify researcher environments and malware sandboxes since the earliest versions. In particular, Qakbot has attempted to identify processes, system artifacts, and the underlying virtual machines associated with an analysis environment. Figure 5

shows an example of Qakbot's implementation to identify whether an infected system is running on a VMWare virtual machine from a sample dating back to September 2009.

```
.text:1000F4F1 ;   __try { // __except at loc_1000F518
.text:1000F4F1                 and     [ebp+ms_exc.registration.TryLevel], 0
.text:1000F4F5                 push    eax
.text:1000F4F6                 push    ebx
.text:1000F4F7                 push    ecx
.text:1000F4F8                 push    edx
.text:1000F4F9                 mov     eax, 'VMXh'
.text:1000F4FE                 mov     ecx, 0Ah
.text:1000F503                 mov     dx, 'VX'
.text:1000F507                 in      eax, dx
.text:1000F508                 mov     [ebp+var_1C], ebx
.text:1000F50B                 mov     [ebp+var_20], ecx
.text:1000F50E                 pop     edx
.text:1000F50F                 pop     ecx
.text:1000F510                 pop     ebx
.text:1000F511                 pop     eax
.text:1000F512                 jmp     short loc_1000F51B
```

zscaler | ThreatLabz

Figure 5. Qakbot implementation to identify VMWare

Qakbot has continuously added code to identify analysis environments by checking system information such as the name of BIOS vendors, processes, drivers, etc. for strings as shown in Table 1.

vmxnet	vmx_svga	vmrawdsk	vmdebug	vm3dmp
vSockets	srootkit	sbtisht	ansfltr	Xen
XENVIF	XENSRC	XENCLASS	XENBUS	Vmscsi
VirtualBox	Virtual Machine	Virtual HD	VirtIO	VRTUAL
VMware server memory	VMware SVGA	VMware SCSI	VMware Replay	VMware Pointing
VMware Accelerated	VMware	VMW	VMAUDIO	VIRTUAL-DISK
VBoxVideo	QEMU	PROD_VIRTUAL_DISK	MS_VM_CERT	CWSandbox
20202020				

Table 1. Qakbot virtual machine string-based detections

The following processes in Table 2 are frequently used by malware analysts and are also detected by Qakbot:

frida-winjector-helper-32.exe	packetcapture.exe	filemon.exe	proc_analyzer.exe	sniff_hit.exe

frida-winjector-helper-64.exe	capturenet.exe	procmon.exe	sysAnalyzer.exe	sysAnalyzer.exe
tcpdump.exe	qak_proxy	idaq64.exe	sniff_hit.exe	BehaviorDumper.exe
windump.exe	dumpcap.exe	loaddll32.exe	joeboxcontrol.exe	processdumperx64.exe
ethereal.exe	CFF Explorer.exe	PETools.exe	joeboxserver.exe	anti-virus.EXE
wireshark.exe	not_rundll32.exe	ImportREC.exe	ResourceHacker.exe	sysinfoX64.exe
ettercap.exe	ProcessHacker.exe	LordPE.exe	x64dbg.exe	sctoolswrapper.exe
rtsniff.exe	tcpview.exe	SysInspector.exe	Fiddler.exe	sysinfoX64.exe
FakeExplorer.exe	apimonitor-x86.exe	idaq.exe	dumper64.exe	user_imitator.exe

Table 2. Malware analyst process names detected by Qakbot

Around version 404.510, the malware developer added extraneous exports to the Qakbot stager DLL to confuse malware sandboxes as shown in Figure 6. In this example, the export name Wind (or ordinal #458) is the actual entry point.

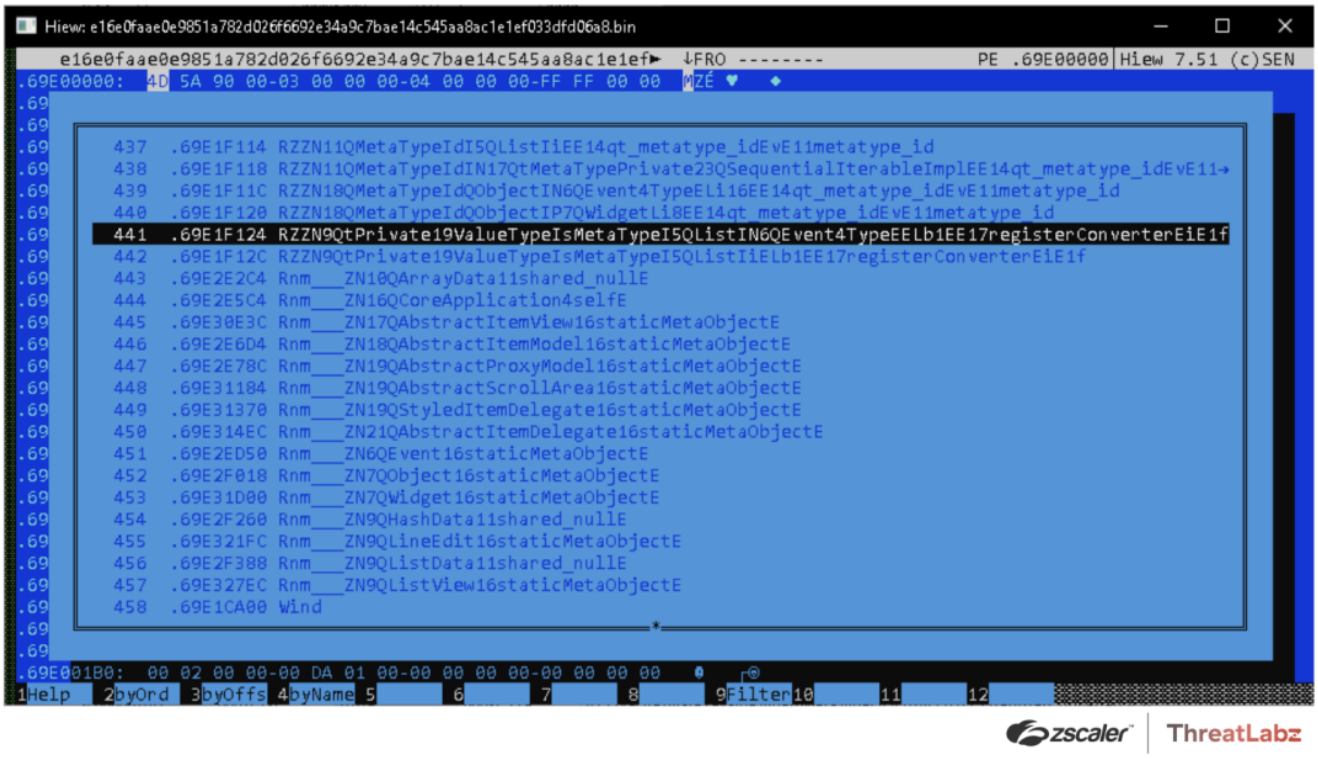


Figure 6. Qakbot 404.510 sample with 458 entries in the exports directory

Network Communication

Qakbot has leveraged HTTP for C2 communication from the beginning. However, the network protocol on top of HTTP has changed significantly over the years with encryption, RSA signature verification, and the addition of a JSON-based message format.

Network protocol and encryption

Qakbot has continuously updated its message protocol with version 19 being the latest. The protocol specifies the format of the message. In version 3, Qakbot sent requests in a format similar to the following:

```
protoversion=9&r=1&n=kvtjmq970452&os=6.1.1.7601.1.0.0100&bg=b&it=3&qv=
0300.288&ec=1453922906&av=0&salt=qrTMyfvpr
```

zscaler | ThreatLabz

However, this protocol format was later replaced with a JSON-based protocol with integer key values that denote specific fields as shown below:

```
{
   "8":1,
   "5":1035,
   "1":19, // protocol version
   "59":0,
   "3":"obama259",
   "4":1028,
   "10":1683022694,
   "2":"kqsvfc505763",
   "6":59661,
   "14":"Xgd1KxZQKTHGB6IxwtIy2e0RAq4iFNE6w6",
   "7":16759,
   "101":1,
   "26":"WORKGROUP",
   "73":0
}
```

zscaler | ThreatLabz

This encoding adds a layer of obfuscation for each of the message fields.

Qakbot's network encryption has used RC4 with the key consisting of 16 random bytes concatenated with a hardcoded salt and hashed using SHA1. The most recent version of Qakbot now uses AES encryption with the key consisting of 16 random bytes concatenated with a hardcoded salt and hashed using SHA256. After encryption, the data is Base64 encoded and prepended to a variable in the body of an HTTP POST request.

Domain generation algorithm

The first versions of Qakbot only used hardcoded C2s as shown in Figure 7.

Address	Length	Type	String
.data:0040F...	00000010	C	http://nt16.in/1
.data:0040F...	00000025	C	http://redserver.com.ua/cgi-bin/ss.pl
.data:0040F...	00000033	C	http://swallowthewhistle.com/cgi-bin/clientinfo3.pl
.data:0040F...	00000023	C	http://www.cdcdcdcdc2121cdsfdfd.com
.data:00410...	00000028	C	http://nt16.in/cgi-bin/jl/jloader.pl?r=q
.data:00410...	0000000B	C	http://%s%s
.data:00410...	00000025	C	http://nt16.in/cgi-bin/jl/jloader.pl?
.data:00411...	0000002A	C	http://nt002.cn/cgi-bin/jl/jloader.pl?r=3d
.data:00411...	0000002D	C	http://redserver.com.ua/cgi-bin/exhandler4.pl

zscaler | ThreatLabz

Figure 7. Example of hardcoded Qakbot C2s

However, in version 2.0.1 a DGA was added as a backup C2 channel in the event that the hardcoded C2s were unreachable. Qakbot used a time-based DGA to generate up to 5,000 C2 domains for a specific date interval as shown in Figure 8.

```
int __cdecl DGA_main_with_callback(int a1, int (__cdecl *DGA_callback)(int, int *, int), int
{
  int result; // eax
  int date_seed; // eax
  unsigned int i; // esi
  int dga_seed[625]; // [esp+Ch] [ebp-E38h] BYREF
  int v7[256]; // [esp+9D0h] [ebp-474h] BYREF
  char date[100]; // [esp+DD0h] [ebp-74h] BYREF
  int v9; // [esp+E34h] [ebp-10h]
  unsigned int v10; // [esp+E38h] [ebp-Ch] BYREF
  unsigned int total_domains_generated; // [esp+E3Ch] [ebp-8h]
  const char **domains_list; // [esp+E40h] [ebp-4h] BYREF

  if ( !g_var_dga
    || time(0) >= dword_100282DC + 43200
    || (sub_1000FFE5(&byte_10024168, (int)v7, 0x400, "http://"),
        result = DGA_callback((int)&g_var_dga, v7, arg_8),
        result < 0) )
  {
    if ( DGA_get_date_from_legit_domains_http_response(date, 0x64u) )//
                                            // queries a domain from
                                            // google.com;microsoft.com;cnn.com
                                            // and gets the date from the response
    {
      date_seed = DGA_make_final_date_and_crc32(date);
      DGA_make_seed(date_seed, dga_seed);
      total_domains_generated = 0;
      while ( 2 )
      {
        v10 = 5;
        domains_list = (const char **)DGA_gen_n_domains((unsigned int *)dga_seed, 5u, 1);
        if ( domains_list )
        {
          for ( i = 0; i < 5; ++i )
          {
            sub_1000FFE5(&byte_10024168, (int)v7, 0x400, "http://");
            v9 = DGA_callback((int)domains_list[i], v7, arg_8);// foreach generated domain,
                                            // call the given callback
            if ( v9 >= 0 )
            {
              delete_domains_list(&domains_list, &v10);
              return v9;
            }
          }
          delete_domains_list(&domains_list, &v10);
          total_domains_generated += 5;
          if ( total_domains_generated < 0x1388 )// max 5000 domains
            continue;
        }
        break;
      }
    }
    return -1;
  }
```

```
int *__cdecl DGA_gen_n_domains(unsigned int *dga_seed, unsigned int n_domains, int
{
  unsigned int n_domains_cp; // edi
  int *domains_pointers_table; // ebx
  char *domain; // eax
  int *v6; // esi
  int v7; // ebx
  int v8; // eax
  int v9; // ecx
  int *v10; // eax
  int v12; // [esp+Ch] [ebp-Ch] BYREF
  int *domains_pointers_table_retval; // [esp+10h] [ebp-8h]
  int tlds; // [esp+14h] [ebp-4h] BYREF
  unsigned int counter; // [esp+24h] [ebp+Ch]

  tlds = 0;
  n_domains_cp = n_domains;
  v12 = (int)do_split("com;net;org;info;biz;org", 0x3B, 1, &tlds);
  domains_pointers_table = (int *)docalloc(4 * n_domains);
  domains_pointers_table_retval = domains_pointers_table;
  if ( !domains_pointers_table )
    return 0;
  for ( counter = 0; counter < n_domains_cp; ++domains_pointers_table )
  {
    domain = (char *)docalloc(0x100u);
    *domains_pointers_table = (int)domain;
    if ( !domain )
      return 0;
    DGA_gen_domain(domain, 256, dga_seed, v12, tlds);
    ++counter;
  }
  if ( flag_1 && n_domains_cp )
  {
    v6 = domains_pointers_table_retval;
    v7 = n_domains_cp - 1;
    do
    {
      v8 = RNG(dword_10042784, 0, v7);
      v9 = *v6;
      v10 = &domains_pointers_table_retval[v8];
      *v6++ = *v10;
      --n_domains_cp;
      *v10 = v9;
    }
    while ( n_domains_cp );
  }
  delete_domains_list((const char ***)&v12, (unsigned int *)&tlds);
  return domains_pointers_table_retval;
}
```

```
char *__cdecl DGA_gen_domain(char *dest_domain, int a2, unsigned int *seed, int tlds, ir
{
  int v5; // edi

  v5 = RNG(seed, 0, ntlds - 1);
  dest_domain[RND_string((int)dest_domain, 8, 25, seed)] = 0;
  strcat(dest_domain, ".");
  return strcat(dest_domain, *(const char **)(tlds + 4 * v5));
}
```

zscaler | ThreatLabz

Figure 8. Qakbot DGA code

Interestingly, some versions of Qakbot would generate fake domains if an analysis environment was detected in an effort to mislead researchers, as shown in Figure 9.

```
if ( DGA_get_date_from_legit_domains_http_response(date, 0x64) )
{
  date_seed = DGA_make_final_date_and_crc32(date);
  if ( ANTI_netmonitor_tools() > 0 )
    ++date_seed;
  DGA_make_seed(date_seed, dga_seed);
  v12 = 0;
  while ( 2 )
  {
    v11 = 5;
    v13 = DGA_gen_n_domains((int)dga_seed, 5u, 1);
```

```
int ANTI_netmonitor_tools()
{
  int v0; // esi
  int splitted_tools; // [esp+4h] [ebp-10h] BYREF
  int v3; // [esp+8h] [ebp-Ch] BYREF
  char *wpcap; // [esp+Ch] [ebp-8h] BYREF
  char *tools; // [esp+10h] [ebp-4h] BYREF

  tools = CORE_strings_decryptor(0x436u);       // .tcpdump.exe;windump.exe;ethereal.exe;
                                                // wireshark.exe;ettercap.exe;rtsniff.exe;
                                                // packetcapture.exe;capturenet.exe;
                                                // wireshark.exe
  splitted_tools = split(tools, ';', 0, &v3);
  wpcap = CORE_strings_decryptor(0x5CFu);       // wpcap.dll
  v0 = ((int (__cdecl *)(_DWORD, int *))UTIL_procfind_with_callback)(pcallback, &splitted_tools);
  UTIL_free_decstr(&wpcap);
  UTIL_free_list_of_ptrs(&splitted_tools, &v3);
  UTIL_free_decstr(&tools);
  return v0;
}
```

zscaler | ThreatLabz

Figure 9. Example of Qakbot generating fake domains if network monitoring tools were detected

Data exfiltration to compromised FTP servers

Qakbot versions 3.0.0 and earlier used compromised FTP servers to exfiltrate data rather than sending the data directly to their C2 server. The FTP credentials were stored in Qakbot's configuration files as shown below:

```
22=<ip address 1>:xxx@credsuser1.com:<credspassword1>:
23=<ip address 2>:xxx@credsuser2.com:<credspassword2>:
24=<ip address 3>:xxx@credsuser3.com:<credspassword3>:
25=<ip address 4>:xxx@credsuser4.com:<credspassword4>:
26=
3=1581496845
```

zscaler | ThreatLabz

This design had an inherent weakness since anyone with the FTP credentials could potentially have accessed and recovered the stolen information. To address this weakness, Qakbot was later updated to send the stolen data directly to Qakbot's C2 infrastructure.

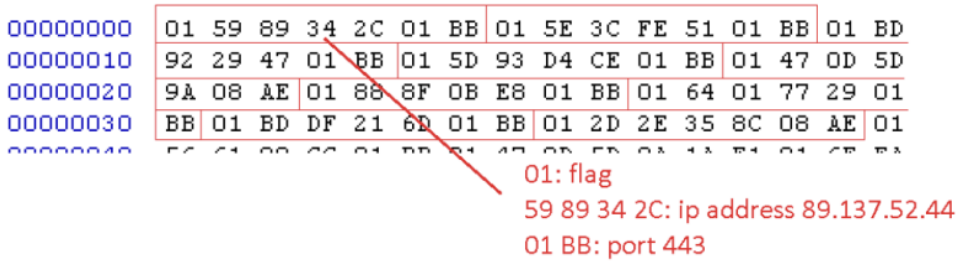
Using compromised systems as relays

After version 3.2.4.8, Qakbot ceased using the DGA. Instead, Qakbot started using compromised systems themselves as C2 servers, and embedded a list of IP addresses and port numbers in the malware configuration. Before version 4.0.3.2, the configuration file (stored as an encrypted resource) contained the list of IP addresses in a text-based format:

```
45.45.105.94;0;443
86.107.20.14;0;443
99.228.5.106;0;443
184.191.62.24;0;995
47.153.115.154;0;995
206.169.163.147;0;995
96.35.170.82;0;2222
73.210.114.187;0;443
75.70.218.193;0;443
...
```

zscaler | ThreatLabz

However, after version 4.0.3.2, the Qakbot C2 list evolved into a binary format as shown in Figure 10.



zscaler | ThreatLabz

Figure 10. Qakbot C2 list binary format

Commands

In the first versions of Qakbot, the server sent commands in a descriptive text-based format. The following commands were supported in Qakbot versions 1.0 and 2.0:

- certssave
- ckkill
- cksave
- clearvars
- cron
- cronload
- cronsave
- forceexec
- ftpwork
- getip
- install3
- instwd
- kill

- killall
- loadconf
- nbscan
- psdump
- reload
- rm
- saveconf
- sleep
- socks
- sxordec
- sxorenc
- sysinfo
- thkill
- thkillall
- uninstall
- update
- update_finish
- uploaddata
- var
- wget

In order to obfuscate these commands, the Qakbot author replaced these string commands with integer values starting in the later builds of version 3.

Addition of RSA signature verification

Qakbot version 3.0.0.443 introduced RSA digital signatures (initially using the MatrixSSL library) to prevent tampering. This was especially important when the DGA and compromised systems were used as C2 servers.

Modular Structure

The design of Qakbot has changed significantly from versions 1 through 5. In particular, the malware has become more modular with the ability to dynamically add new features without releasing a new version of Qakbot. Modern versions use a lightweight stager responsible for initializing, maintaining persistence, and establishing C2 communication to request commands and modules.

Embedded resources

Prior to version 4.0.2.19, Qakbot frequently used the resource section to store configuration information (such as web injects and application parameters) as well as DLLs that performed malicious behavior. Initially, in version 1.0, these resources were not encrypted. However, Qakbot's code evolved with various encryption algorithms to protect these resources.

Qakbot version 2.0 implemented a custom XOR-based algorithm as shown in Figure 11.

```
  v9 = *(_WORD *)(resource_cp + 7);
  if ( v9 )
  {
    memcpy((void *)(a1 + 512), (const void *)(resource_cp + 9), v9);
    *(_WORD *)(a1 + 544) = *(_WORD *)(resource_cp + 7);
  }
  else
  {
    ALGOS_init_sha1(a1, Str_null);
  }
  nullsub_2(resource_cp + 9, *(unsigned __int16 *)(resource_cp + 7), aSxor2DecryptFu);
  result = ALGOS_xor_dec(a1 + 512, *(_WORD *)(a1 + 544), resource, *(_DWORD *)(a1 + 548), size_of_resource_cp);
}
*(_DWORD *)(a1 + 552) = result;
*(_DWORD *)(a1 + 556) = result;
return result;
```

```
unsigned int __cdecl ALGOS_xor_dec(int key, unsigned __int16 key_sz, int resource, int dest, unsigned int resource_sz)
{
  _BYTE *v5; // esi
  unsigned __int8 v6; // al
  char v7; // dl
  unsigned int i; // [esp+0h] [ebp-4h]

  for ( i = 0; i < resource_sz; *v5 = v7 )
  {
    v5 = (_BYTE *)(i + dest);
    v6 = *(_BYTE *)(resource + i) ^ *(_BYTE *)(i % key_sz + key);
    v7 = (v6 >> (i & 7)) | ((v6 & (unsigned __int8)(255 >> (8 - (i & 7)))) << (8 - (i & 7)));
    ++i;
  }
  return resource_sz;
}
```


Figure 11. Custom encryption algorithm used by Qakbot 2.0 to protect resources

In this example, the offset 0x7 in the encrypted resource contained a WORD that was the size of the XOR key. The XOR key was located at offset 0x9 in the resource. Encrypted data was then concatenated after the XOR key. Python code that replicates this algorithm is shown below:

```
import struct

def custom_xor(data, key):
    out = b""
    for i, eb in enumerate(data):
        v6 = eb ^ key[i % len(key)]
        v1 = (v6 & (0xff&(255 >> (8 - (i & 7))))) << (8 - (i & 7))
        v2 = (v6 >> (i & 7))
        v3 = v1 | v2
        out += struct.pack("B", v3)
    return out

s = open('res.bin', 'rb').read()
key_sz = struct.unpack('H', s[7:9])[0]
s = custom_xor(s[9+key_sz:], s[9:9+key_sz])
print(s)
```


Qakbot version 3.0 and later used an RC4-based algorithm to decrypt the resources.

The initial 0x14 bytes in the resource served as the RC4 key for decrypting the remaining data. A slightly modified version of the BriefLZ library was later added to compress specific resources to reduce the overall file size.

In version 4.0.2.1, the resource encryption algorithm changed slightly. The first 0x14 bytes of the resource were no longer used as an RC4 key. Instead, the code contained a salt value in the encrypted strings table that was then hashed using SHA1 to derive the RC4 key used to decrypt the resource. In version 4.0.3.902 this was improved again, which added two layers of RC4 to decrypt the resource. The first RC4 layer was decrypted using the SHA1 hash of the salt string. The second layer used the first 0x14 bytes of the result as the key to decrypt the following data. Example Python code for this algorithm is shown below:

```
seed = b"Muhcu#YgcdXubYBu2@2ub4fbUhuiNhyVtcd"
key = SHA1(seed).digest()
dec_data = RC4(resource_data, key)[0x14:]
data = RC4(dec_data[0x14:], dec_data[0:0x14])[0x14:]
```

zscaler | ThreatLabz

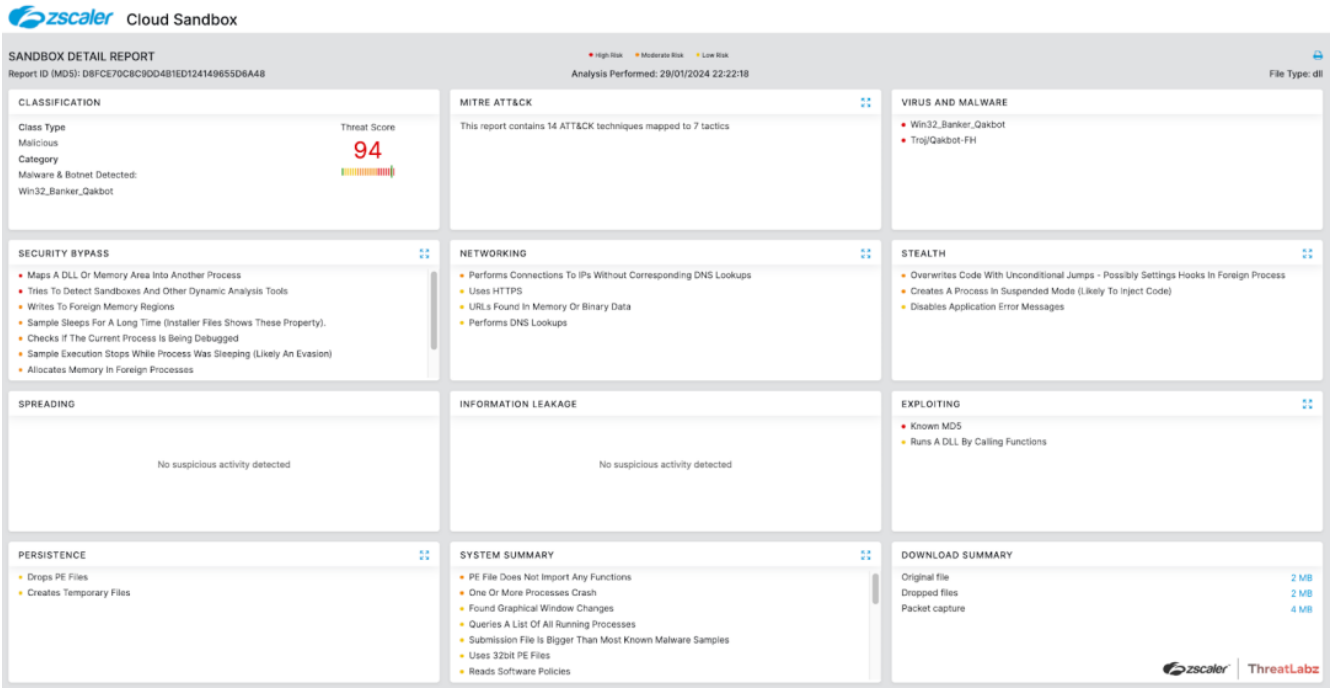
Plugins

In version 4.0.1, Qakbot was modified to split various functionality into separate modules. This allowed Qakbot to use a stager to download additional modules from Qabkot's C2 servers to add functionality on-demand. Qakbot has built modules to hook web browsers, steal email addresses (and email), harvest stored credentials, deploy Cobalt Strike, and act as a C2 server that relays traffic between other infected systems and the backend infrastructure.

Conclusion

Qakbot is a sophisticated trojan that has evolved significantly over the past 15 years, and remains remarkably persistent and resilient. Despite the significant disruption to Qakbot in August 2023, the threat group remains active and recently updated their codebase to support 64-bit versions of Windows, improved the encryption algorithms, and added more obfuscation. This demonstrates that Qakbot will likely remain a threat for the foreseeable future and ThreatLabz will continue to add detections to protect customers.

Zscaler Cloud Sandbox



Zscaler's multilayered cloud security platform detects payloads with the following threat names:

Win32.Banker.Qakbot

Indicators Of Compromise (IOCs)

Date	Version	Sample Hash
2008-08-28	1.0.0	34588857312371e4b789fb49d2606386
2009-11-16	1.0.0	8c33780752e14b73840fb5cff9d31ba1
2009-12-29	1.0.0	37bbdaf1d14efa438f9ff34d8eeaa5e7
2010-10-12	1.0.0.63	d02252d88c3eab14488e6b404d2534eb
2011-05-12	2.0.0.685	b9e23bc3e496a159856fd60e397452a0
2012-05-31	2.0.1.1432	570547fa75c15e6eb9e651f2a2ee0749
2013-07-08	2.0.1.1457	42e724dc232c4055273abb1730d89f28
2014-06-24	2.0.1.2544	9160ea12dbce912153b15db421bb87da
2015-01-28	2.0.1.2674	945ba16316c8a6a8428f0b50db0381dc

Date	Version	Sample Hash
2015-12-17	3.0.0.116	dca0ef26493b9ac3172adf931f1a3499
2016-01-04	3.0.0.180	6718c6af4b89cffd9b6e0c235cf85bd2
2016-01-04	3.0.0.275	8fbb43dc853d0b95829112931493fe22
2016-01-13	3.0.0.262	72125013ac58d05adb32b7406b02c296
2016-01-29	3.0.0.322	3b4a2e984a51210d0594c9b555ba4e0d
2016-02-09	3.0.0.333	f952dc1e942ebdfb95a2347263265438
2016-02-12	3.0.0.352	b849381ab6a4e97d32580bb52d15cb7d
2016-03-08	3.0.0.443	dc8b137d5d61b23dbbb6085ce46bfcdb
2016-04-05	3.0.0.468	327a5e491d6db899d9db4c6bdc8f5367
2016-04-05	3.0.0.473	e3b0e54777ca9fd9863e3563a1b7dd59
2016-04-06	3.0.0.506	2e9261e75e15540ef88327a480a5b10e
2016-04-26	3.0.0.580	a472b9dd64198d739c6e415bbcae8a6f
2016-05-19	3.0.0.739	8609e6e4d01d9ef755832b326450cbe9
2016-06-01	3.0.0.743	a7cc19cde3a1a78b506410e4ffafdbef
2017-04-27	3.1.0.723	581016035f95327e7e1daac3ad55ae0e
2017-05-16	3.1.0.733	361d46f32a93786b34b2ac225efc0f79
2018-02-06	3.2.2.381	89e6f171c29255d6b4490774c630ad14
2019-09-16	3.2.3.91	ff186a1ef9e83c229940ff2dd4556eaf
2020-01-22	3.2.4.8	bea66da7088bd20adbfed57cf350a6a4
2020-01-22	3.2.4.8	1cd7a95064515625ad90464a65ea4d94

Date	Version	Sample Hash
2020-03-03	3.2.4.53	08c51514a42eec6ccbbc7a09a8258419
2020-03-20	3.2.4.70	d8ff9d18cd622c545d21b199a2d17594
2020-04-01	3.2.4.75	2e658f5fa658651331cb5b16447bdbe2
2020-04-29	3.2.4.136	ca22283396dbe21fa2ef5e27c85ffae6
2020-05-07	3.2.4.141	e9d0e767a5c5284ab33a3bb80687cf63
2020-05-07	3.2.4.141	d8841201c9d32b5e885f4d035e32f654
2020-05-28	3.2.4.401	82d7c5ea49c97059bbec02161b36f468
2020-08-07	3.2.5.42	163ee88405bccc383c7b69c39028bf9a
2020-08-07	3.2.5.42	acf65632b7cdc40091daec58bf8830bc
2020-08-11	3.2.5.43	455c543243f5216e21ba045814311971
2020-08-11	3.2.5.43	cfc77e4421d830e73c6f6040a4baedd4
2020-11-03	3.2.5.83	40a9bdac882285ab844917d8b5b75188
2020-11-24	4.0.1.29	6b1771b883c0b3ffdc3f5923f45c1f93
2020-12-15	4.0.1.138	0a3caa2845251b8fb5ab72f450edd488
2021-03-12	4.0.1.194	4a6e7f055d5bf4fd6d2a401c1b3d18ab
2021-04-12	4.0.2.1	dc2acf1704456880208146c91692cfc8
2021-04-15	4.0.2.12	3ca1f0e708283f21c9a10ef4acf40990
2021-04-15	4.0.2.12	1e71ea79c5a70bb8c729037132855b5a
2021-04-22	4.0.2.12	66a87dbc24af866849646911f4841a28
2021-04-29	4.0.2.68	25984af48fa27ec36bd257f8478aa628

Date	Version	Sample Hash
2021-04-29	4.0.2.68	c1849c1ee3b8146c6fb836dae0b64652
2021-05-06	4.0.2.68	d45e04df3c9270a01e9fb9e4e8006acc
2021-09-20	4.0.2.318	9a1c1497428743b4e199f2583f3d8390
2021-09-27	4.0.2.363	0865757dfe54c2d01c5cef5bfd3162c5
2021-09-27	4.0.2.363	c6dea1f4e6ee1ed4c0383cd1af456649
2021-11-03	4.0.3.1	1d4952cbe998312fd2bf810535db8a20
2021-11-03	4.0.3.1	6cce1ec83d1428de9fcb0c3791efabd1
2021-11-04	4.0.3.2	e111d982dc0c12f23fa3f446d674600b
2021-11-04	4.0.3.2	751f7d8ad6b2308cd1750fc23f606b53
2021-12-09	4.0.3.10	8bb4208a50c041f9cdfc26815905eab3
2022-02-10	4.0.3.490	bcb8e64c5a69c7a572ca34450712fb2f
2022-02-14	4.0.3.491	54e3f20f74c1089e89841798ffaac084
2022-02-14	4.0.3.503	95adeb6a1c1e0a9d9ee4ecafb6079b37
2022-02-15	4.0.3.509	da206d25fddf3286f42ec7626d8bb676
2022-02-18	4.0.3.532	3ba490216d4cdf92661444d896fefac3
2022-02-24	4.0.3.549	8fa26ff07c3b5e1653e55b8a567b7623
2022-02-24	4.0.3.549	1253695c63136edb1f6b37bbfd83db55
2022-04-06	4.0.3.573	2853985cab3c5b83eec38ae1f3a890be
2022-04-29	4.0.3.573	5e7deb4acb4429498693bc45db68978a
2022-05-04	4.0.3.674	2273dd59ca71c4f078cab09d93093294

Date	Version	Sample Hash
2022-05-04	4.0.3.675	40d5e775a52c94842c97d012eb94efdc
2022-05-04	4.0.3.683	f1d47a4dc1d11b17e51419299dc282e4
2022-05-12	4.0.3.684	2f17bd9f4b9edd91a7fd80ef32981f70
2022-05-18	4.0.3.686	7dcbd74778754eee85810a4393d8e3ef
2022-05-18	4.0.3.688	e9e9d194f3ee9822852309cc83455eea
2022-05-23	4.0.3.689	019117f66e43de489b3ff56377f9907b
2022-05-24	4.0.3.690	28f84ffa14c7ef3936a00d3bd751bdb3
2022-06-07	4.0.3.694	d88ee89344d04f83eacd3614785560ef
2022-08-31	4.0.3.780	3ff9d9dbf8c7a6865faeb43188afa6b4
2022-09-06	4.0.3.858	3e86ac10b4e7d818e0f410130bb7f237
2022-09-08	4.0.3.860	377acb7149fdfa56c090d9a12619a53c
2022-09-15	4.0.3.892	e5ebdec7417ad847e4325c4114e41809
2022-09-20	4.0.3.894	c23d2cd7d10a5f88032ddfcab4cfe146
2022-09-28	4.0.3.895	050ce5fb25ffd3e907a5c81a6711fcea
2022-10-04	4.0.3.914	b857efb30d9e35bc83a294580ad8cc3a
2022-10-10	4.0.3.967	6dc027269262b93351633eb8af4623ef
2022-10-11	4.0.3.973	e5eb07b009ca666f91ef5fe48269ca52
2022-10-25	4.0.3.1051	0971b8e78fcc6f9158e279376116c8c4
2022-10-26	4.0.4.2	4fbebc9879ec1f95e759cb8b5d9fb89d
2022-10-28	4.0.4.14	66a0741f8f43b584e387459b367097c1

Date	Version	Sample Hash
2022-10-31	4.0.4.20	6d61a88890be4ab5116cb712ff7788f4
2022-11-08	4.0.4.26	da75924c717524a8d17de126f8368ec4
2022-11-08	4.0.4.27	5971c4a485e881268ca28f24fdedc4e5
2022-11-16	4.0.4.30	22e45a212998d2ee264b6756b2972901
2022-11-28	4.0.4.46	accc6d9ba88040c89df34ef1749944d1
2022-12-13	4.0.4.52	22b3cb9b0bacd525a83aab5b1a853f63
2022-12-20	4.0.4.60	bebebd4e16a88f43f16e4c6c811c9894
2022-12-20	4.0.4.62	cafb7b2f8383cf9686f144dc2082f287
2022-12-22	4.0.4.66	6e3b4252903c0f3a153e011445ad2179
2023-01-31	4.0.4.432	3e3bc981a7fdbae10b40cd6683edacbb
2023-01-31	4.0.4.432	a12dd4324bbf1129d9fae1b3d1e6b9ca
2023-05-02	4.0.4.1035	ebec03d53d716cd780c92c5c29a95e6b
2023-05-10	4.0.4.1038	5e4c95b2c1b14a8a0f425576189fae60
2023-12-11	5.0.0.326	8aec3f3ef66e4ff118bfdab1d031eadb
2023-12-13	5.0.0.361	46e169516479d0614b663f302b5d1ace
2023-12-19	5.0.0.370	795319d48ce1f680699beb03317c6bff
2024-01-22	5.0.0.484	de1d9ed6da4f34b4444b13442aac5033
2024-01-22	5.0.0.486	f382d0f92221831eeb39c108f8ccfa26

Thank you for reading

Was this post useful?

Yes, very!Not really

Get the latest Zscaler blog updates in your inbox

By submitting the form, you are agreeing to our privacy policy.