Threat Spotlight: Rombertik - Gazing Past the Smoke, Mirrors, and Trapdoors

By Talos Group,

Published: 2015-05-04 · Archived: 2026-04-05 16:49:47 UTC

This post was authored by Ben Baker and Alex Chiu.

Executive Summary

Threat actors and security researchers are constantly looking for ways to better detect and evade each other. As researchers have become more adept and efficient at malware analysis, malware authors have made an effort to build more evasive samples. Better static, dynamic, and automated analysis tools have made it more difficult for attackers to remain undetected. As a result, attackers have been forced to find methods to evade these tools and complicate both static and dynamic analysis.

> **Table of Contents**
> Executive Summary
> The 10,000 Foot View at Rombertik
> Analysis
> A Nasty Trap Door
> The Actual Malware
> Coverage and Indicators of Compromise
> Conclusion

It becomes critical for researchers to reverse engineer evasive samples to find out how attackers are attempting to evade analysis tools. It is also important for researchers to communicate how the threat landscape is evolving to ensure that these same tools remain effective. A recent example of these behaviors is a malware sample Talos has identified as Rombertik. In the process of reverse engineering Rombertik, Talos discovered multiple layers of obfuscation and anti-analysis functionality. This functionality was designed to evade both static and dynamic analysis tools, make debugging difficult. If the sample detected it was being analyzed or debugged it would ultimately destroy the master boot record (MBR).

Talos' goal is to protect our customer's networks. Reverse engineering Romberik helps Talos achieve that goal by better understanding how attackers are evolving to evade detection and make analysis difficult. Identifying these techniques gives Talos new insight and knowledge that can be communicated to Cisco's product teams. This knowledge can then be used to harden our security products to ensure these anti-analysis techniques are ineffective and allow detection technologies to accurately identify malware to protect customers.

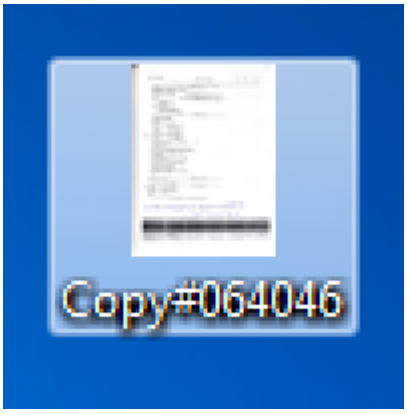
The 10,000 Foot View at Rombertik

At a high level, Romberik is a complex piece of malware that is designed to hook into the user's browser to read credentials and other sensitive information for exfiltration to an attacker controlled server, similar to Dyre. However, unlike Dyre which was designed to target banking information, Rombertik collects information from all websites in an indiscriminate manner.

Rombertik has been identified to propagate via spam and phishing messages sent to would-be victims. Like previous spam and phishing campaigns Talos has discussed, attackers use social engineering tactics to entice users to download, unzip, and open the attachments that ultimately result in the user's compromise.

Figure 1: Email messages such as the one seen here are spammed out in the hopes that users will open them and open the attachments. This is one sample that was used to propagate Rombertik.

In this sample, the message observed in Rombertik's distribution appears to come from the "Windows Corporation," an organization that possesses "state-of-the-art manufacturing quality processes." The attackers attempt to convince the user to check the attached documents to see if their business aligns with the target user's organization. If the user downloads and unzips the file, the user then sees a file that looks like a document thumbnail.



While this file may appears to be some sort of PDF from the icon or thumbnail, the file actually is a .SCR screensaver executable file that contains Rombertik. Once the user double clicks to open the file, Rombertik will begin the process of compromising the system.

The process by which Rombertik compromises the target system is a fairly complex with anti-analysis checks in place to prevent static and dynamic analysis. Upon execution, Rombertik will stall and then run through a first set of anti-analysis checks to see if it is running within a sandbox. Once these checks are complete, Rombertik will proceed to decrypt and install itself on the victims computer to maintain persistence. After installation, it will then launch a second copy of itself and overwrite the second copy with the malware's core functionality. Before Rombertik begins the process of spying on users, Rombertik will perform once last check to ensure it is not being analyzed in memory. If this check fails, Rombertik will attempt to destroy the Master Boot Record and restart the computer to render it unusable. The graphic below illustrates the process.

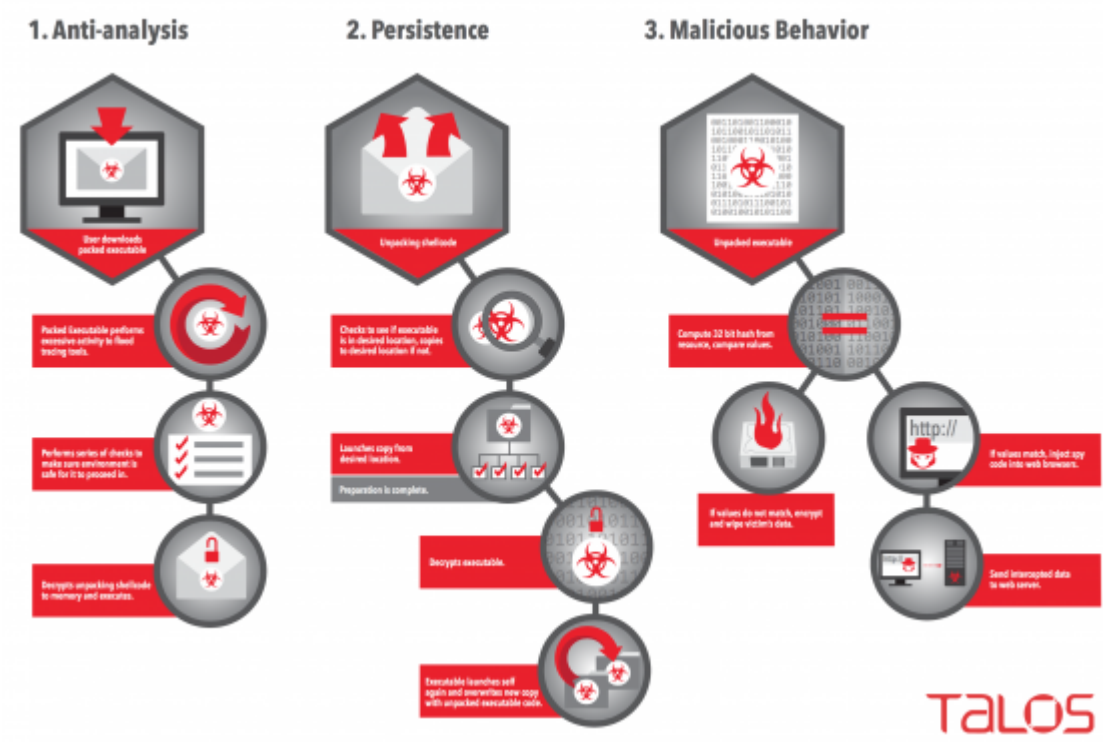


Figure 2: An illustration of the step-by-step process Rombertik follows to compromise the target system.

Analysis

From the beginning, Rombertik incorporates several layers of obfuscation along with anti-analysis functionality. Obfuscating the functionality of a malware sample can be accomplished in many different ways. A common method is to include garbage code to inflate the volume of code an analyst might have to review and analyze. In this case, the unpacked Rombertik sample is 28KB while the packed version is 1264KB. Over 97% of the packed file is dedicated to making the file look legitimate by including 75 images and over 8000 functions that are never used. This packer attempts to overwhelm analysts by making it impossible to look at every function.

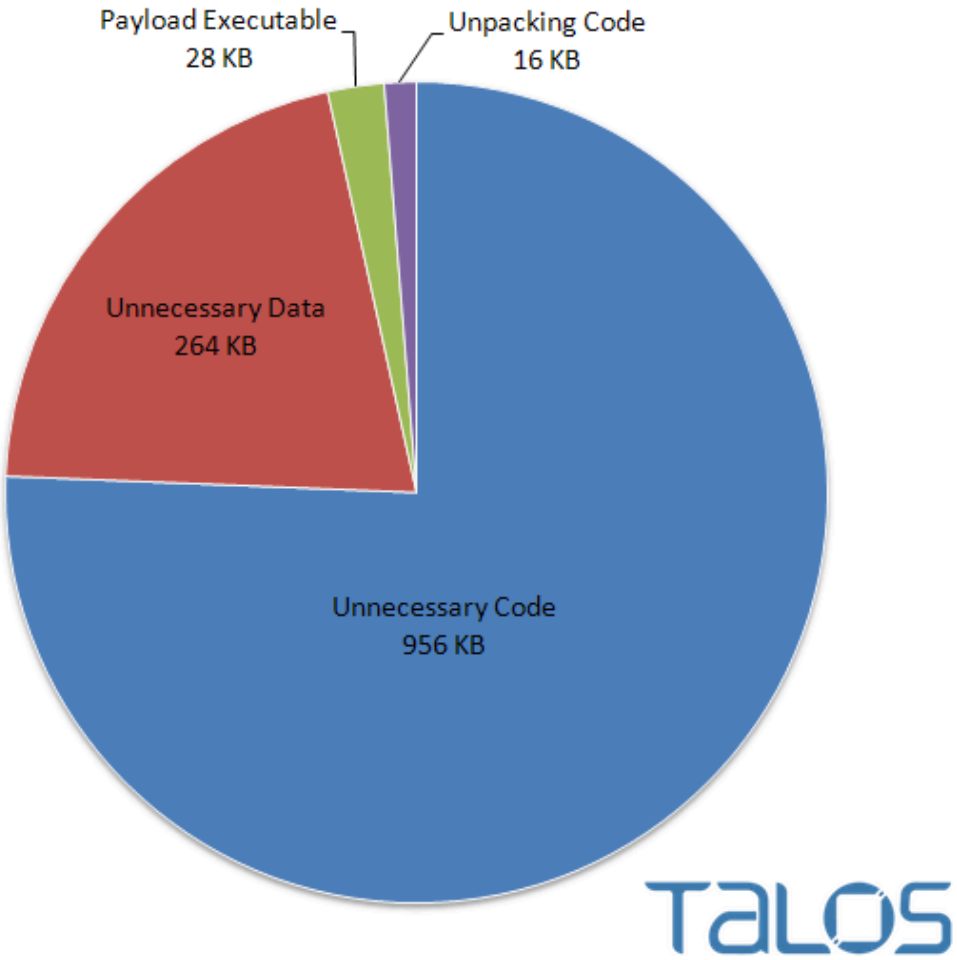


Figure 3: This chart shows the breakdown of the Rombertik executable and how it contains a large amount of unnecessary code and data.

As we started to slowly peel back the layers and focus on the subset of functions that are actually used, we observed an interesting sandbox evasion technique. A common technique to evade sandboxes is to sleep for extended lengths of time with the intention of forcing the sandbox to time out before the malware "wakes up" and begins executing. In response, sandboxes got better at detecting and responding when malware slept for extended periods of time. Rombertik employs a similar approach to delay execution, but does so without sleeping. Rombertik instead writes a byte of random data to memory 960 Million times. This is designed to consume time, like sleeping, but presents a couple disadvantages for sandboxes and application tracing tools. Sandboxes may not be able to immediately determine that the application is intentionally stalling since it's not sleeping. The other disadvantage is that the repetitive writing would flood application tracing tools. If an analysis tool attempted to log all of the 960 Million write instructions, the log would grow to over 100 gigabytes. Even if the analysis environment was capable of handling a log that large, it would take over 25 minutes just to write that much data to a typical hard drive. This complicates analysis.

After intentionally stalling by writing to memory repeatedly, Rombertik checks to see if analysis tools have modified code in the Windows API ZwGetWriteWatch routine. It does this by calling ZwGetWriteWatch with invalid arguments. If the routine does **not** return with a specific error, Rombertik terminates. The assumption behind checking for a specific error versus a generic error is to check for sandboxes that suppress errors returned from API routine calls. Once the sandbox check is complete, Rombertik calls the Windows API

OutputDebugString function 335,000 times as an anti-debugging mechanism. Finally, an anti-analysis function within the packer is called to check the username and filename of the executing process for strings like “malwar”, “sampl”, “viru”, and “sandb”. If the packer detects any of these substrings, it will stop unpacking and terminate. At this point, the initial anti-analysis checks are complete.

Once the packer has run through initial anti-analysis checks, it will check to see if it is executing from %AppData%\rsr\yfoye.exe. If the packer is not executing from there, it will proceed to install itself in order to ensure persistence across system reboots before continuing on to execute the payload. To install itself, Rombertik first creates a VBS script named “fgf.vbs”, which is used to kick off Rombertik every time the user logs in, and places the script into the user’s Startup folder. Rombertik then creates %AppData%\rsr\yfoye.bat and moves the packed version of itself into %AppData%\rsr\yfoye.exe.

If Rombertik detects it is already executing from %AppData%\rsr\yfoye.exe, the malware will then begin decrypting and executing the main unpacking code in memory. Rombertik will then proceed to execute yfoye.exe a second time to create a new instance of the process. Once the unpacking is complete, Rombertik will overwrite the memory of the new process with the unpacked executable code. The unpacking code is monstrous and has many times the complexity of the anti-analysis code. The code contains dozens of functions overlapping with each other and unnecessary jumps added to increase complexity. The result is a nightmare of a control flow graph with hundreds of nodes. Figure 4 helps illustrate how complex the unpacking code is in comparison to the all the code that performs anti-analysis checks.

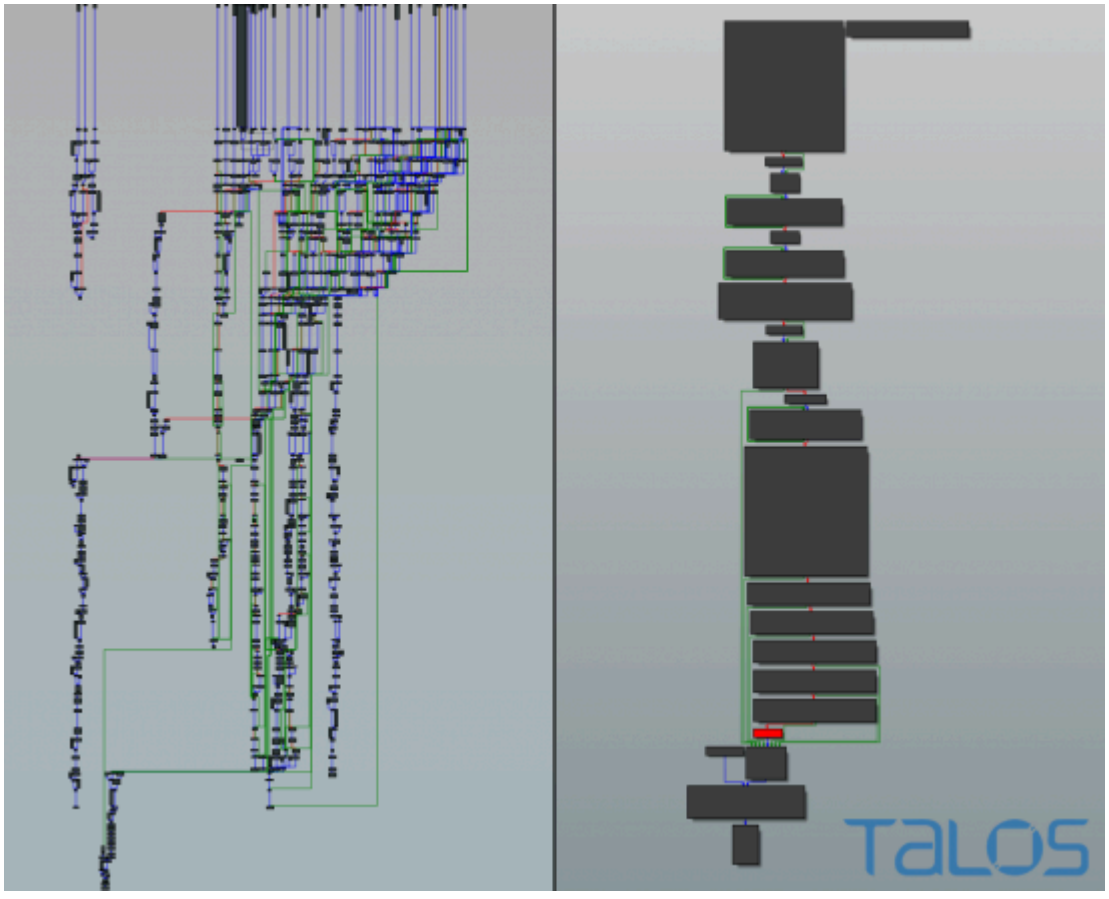


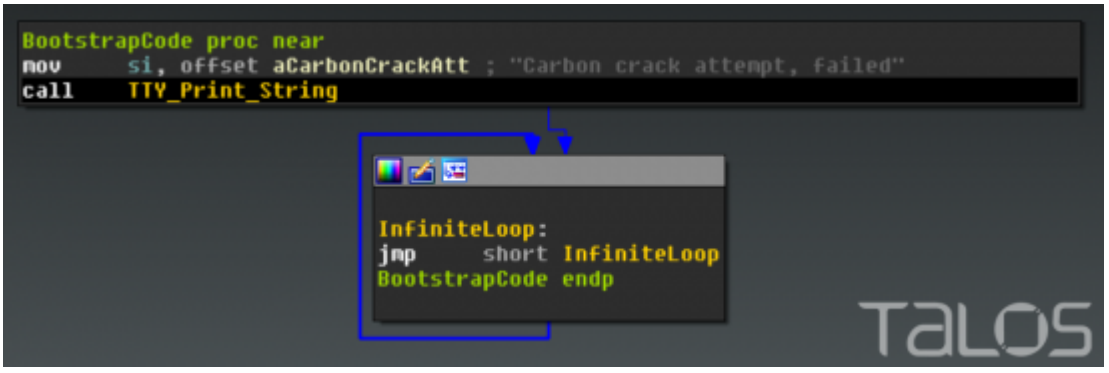
Figure 4: The control flow graph on the left represents the interwoven functions within the unpacking code that’s decrypted to memory. The control flow graph on the right represents the

previously mentioned anti-analysis checks. These 23 basic blocks represent the 930 million writes, 335 thousand API calls, checking ZwGetWriteWatch, and checking file and usernames. All of this functionality fits in this rather simple graph, where the red block is only executed if all of the checks were satisfied. A typical function has less than 20 nodes (basic blocks) and would normally be easy to see how all basic blocks relate to each other.

A Nasty Trap Door

Once the unpacked version of Rombertik within the second copy of yfoye.exe begins executing, one last anti-analysis function is run — which turns out to be particularly nasty if the check fails. The function computes a 32-bit hash of a resource in memory, and compares it to the PE Compile Timestamp of the unpacked sample. If the resource or compile time has been altered, the malware acts destructively. It first attempts to overwrite the Master Boot Record (MBR) of PhysicalDisk0, which renders the computer inoperable. If the malware does not have permissions to overwrite the MBR, it will instead destroy all files in the user's home folder (e.g. C:\Documents and Settings\Administrator\) by encrypting each file with a randomly generated RC4 key. After the MBR is overwritten, or the home folder has been encrypted, the computer is restarted.

The Master Boot Record starts with code that is executed before the Operating System. The overwritten MBR contains code to print out "Carbon crack attempt, failed", then enters an infinite loop preventing the system from continuing to boot.



The MBR also contains information about the disk partitions. The altered MBR overwrites the bytes for these partitions with Null bytes, making it even more difficult to recover data from the sabotaged hard drive.

Once the computer is restarted, the victim's computer will be stuck at this screen until the Operating System is reinstalled:

Effectively, Rombertik begins to behave like a wiper malware sample, trashing the user's computer if it detects it's being analyzed. While Talos has observed anti-analysis and anti-debugging techniques in malware samples in the past, Rombertik is unique in that it actively attempts to destroy the computer's data if it detects certain attributes associated with malware analysis.

The Actual Malware

At this point, Rombertik will assume that all anti-analysis checks have passed and will actually begin doing what was originally intended — stealing user data. Rombertik will scan the user's currently running process to determine if a web browser is currently running. If Rombertik detects an instance of Firefox, Chrome, or Internet Explorer, it will inject itself into the process and hook API functions that handle plain text data. Once accomplished, Rombertik is then able to read any plain-text data the user might type into their browser and capture this input before it gets encrypted if the input is to be sent over HTTPS. This enables the malware to collect data such as usernames and passwords from almost any website. Rombertik does not target any site in particular, such as banking sites, but instead, attempts to steal sensitive information from as many websites as possible. The collected data is then Base64 encoded and forwarded to www.centozos.org.in/don1/gate.php (in this

example) over HTTP with no encryption.

Coverage and Indicators of Compromise

Sample Analysed (SHA256)

0d11a13f54d6003a51b77df355c6aa9b1d9867a5af7661745882b61d9b75bccf

Command-and-Control Servers

www.centozos[.]org[.]in

Conclusion

Rombertik is a complex piece of malware with several layers of obfuscation and anti-analysis functionality that is ultimately designed to steal user data. Good security practices, such as making sure anti-virus software is installed and kept up-to-date, not clicking on attachments from unknown senders, and ensuring robust security policies are in place for email (such as blocking certain attachment types) can go a long way when it comes to protecting users. However, a defense in depth approach that covers the entire attack continuum can help identify malware and assist in remediation in the event that an attacker finds a way to evade detection initially.

For Talos, understanding how malware changes and evolves is essential to developing detection content and ensuring that static, dynamic, and automated analysis tools remain effective. We must adapt, change, and respond accordingly to address the evolving threat landscape. Looking forward, Talos expects these methods and behaviors to be adopted by other threat actors in the future.

Protecting Users From These Threat

Product	Protection
AMP	✔
CWS	✔
ESA	✔
Network Security	✔
WSA	✔

We encourage organizations to consider security best practices, starting with a threat-centric approach that implements protections across the extended network and across the full attack continuum.

ESA can block malicious emails, including phishing and malicious attachments sent by threat actors.

CWS/WSA web scanning prevents access to websites hosting malicious content.

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

Network Security appliances, such as NGIPS and NGFW, have signatures to detect and block malicious network activity by threat actors.

Source: http://blogs.cisco.com/security/talos/rombertik