Large Scale Studies: Malware Needles in a Haystack

Giovanni Bertão[1], Marcus Botacin[2], André Grégio[2], Paulo Lício de Geus[1]

[1] University of Campinas (Unicamp)

`{bertao, paulo}@lasca.ic.unicamp.br`

[2]Federal University of Paraná (UFPR)

`{mfbotacin, gregio}@inf.ufpr.br`

Abstract. *Malware overview reports are valuable information to understand threats behavior and develop proper countermeasures. Currently, most of these studies are focused on either fine-grained, individual sample analysis or coarse-grained landscapes. On the one hand, only the first allows professionals to handle specific security breaches. On the other hand, only the second allows understanding threat scenario as a whole. We claim a complete security treatment is only possible when combining both approaches. Therefore, in this work, we present an analysis of a large malware dataset, showing the distinctions between coarse-grained and fine-grained analysis results. We present both a general threat scenario based on coarse-grained results as well as we detail our fine-grained results to identify particular malicious constructions to antecipate incident response of future threats.*

1. Introduction

Malware is a constant threat to modern computer systems. To counter such kind of threat, analysis procedures are employed, thus allowing vaccine development, remediation and enabling forensic procedures,

Most of the current malware analysis research is presented in two forms: i) a coarse-grained overview, highlighting only major aspects, discarding samples details; ii) a fine-grained, specific view, focusing on implementation details, but not stating the risk of such sample in the overall scenario.

We believe such approaches are complementary and security analyses must consider both to provide a complete threat understanding. To support this claim, this work presents a comparison of both approaches to highlight their differences and how they can be integrated.

For our evaluation, we have considered a dataset of 135 thousand unique, real binary samples. They were all submitted to static and dynamic analysis procedures and their results were analyzed in both coarse and fine-grained ways. Our results demonstrate that a coarse-grained analysis procedure allows us to draw a threat landscape but only a fine-grained analysis procedure allows us to identify rare constructions.

This work is organized as follows: we motivate our work in Section 2; in Section 3, we present our assumptions, data collection and analysis methods; in Section 4, we present the threat landscape; in Section 5, we discuss the impact of our discoveries; finally, we draw our conclusions in Section 6.

2. Motivation & Related Work

To exemplify how coarse and fine-grained analysis differ, consider the ransomware case. An overview report states ransomware attacks have grown more than 50% [BBC 2017]. On the one hand, such kind of information is valuable for security planners as they need to know what kind of attack their organizations will face in the future. On the other hand, such report says nothing about how and where ransomware is growing—i.e. which vulnerabilities are being exploited and/or techniques are being leveraged by attackers. In summary, they knew what they need to do (protect against ransomware), but not how and where.

Now, consider the opposite case. A fine-grained AV report evaluates the Erebus ransomware [TrendMicro 2017], a Linux threat. The report details how the sample is implemented, the affected directories, and so on. However, the report does not state the impact of such sample, as it does not position Linux threats in the overall scenario of computer systems. In a summary, it says where and how Linux servers should be protected, but does not state its impact.

Given the above, we claim that only an integrated view can enable full understanding of malicious threats. The hereby presented research is an attempt towards such direction. Therefore, this work is related to both coarse-grained and fine-grained research work. Among the first group, we highlight the Bayer's [Bayer et al. 2009] evaluation of Windows binaries and the Lindorfer's [Lindorfer et al. 2014] for Android ones. In this work, we establish a global scenario as done in these. For the last group, enumeration is not possible, as AV reports are released for many individual samples.

3. Methodology

In this section, we present the general methodology adopted in our work, such as sample collection, separation and the performed analysis.

3.1. Dataset

In this work, we considered samples collected from the Malshare[1] database, a collection of worldwide samples uploaded by security professionals. To investigate current threats, we daily crawled the most recently available samples in the period from Dec/2016 until May/2018. In total, we collected 242 thousand malicious objects, among which 141 thousand (58.5%) were unique binaries. Over these, 135 thousand (95.5%) were valid binaries and exhibited some behavior during dynamic analysis.

3.2. Analysis Method

For both coarse and fine-grained analysis, we proceeded in the same way. All samples were both statically and dynamically analyzed [Sikorski and Honig 2012]. For static analysis, we considered packers, identified using `PEId`[2], anti-analysis techniques, identified using `Peframe`[3], and antivirus detection, by submitting samples to `Virustotal`[4]. For dynamic analysis, we considered the logs from an internal sandbox solution [Botacin et al. 2017]. The analyses were conducted as soon as the samples were collected and further compiled in the hereby presented paper.

[1] `malshare.com` [2] `aldeid.com/wiki/PEiD` [3] `github.com/guelfoweb/peframe`
[4] `virustotal.com`

3.3. Large Scale Support

Serial procedures are not efficient to handle a large collection of malware samples and results as they consume much time. Therefore, we developed a parallel analysis framework for such task. Our pipeline is implemented in two stages: i) the first works on a producer-consumer way, providing samples for analysis; ii) the second works on a map-reduce way, consuming analysis results. A general view is presented in the Figure 1.

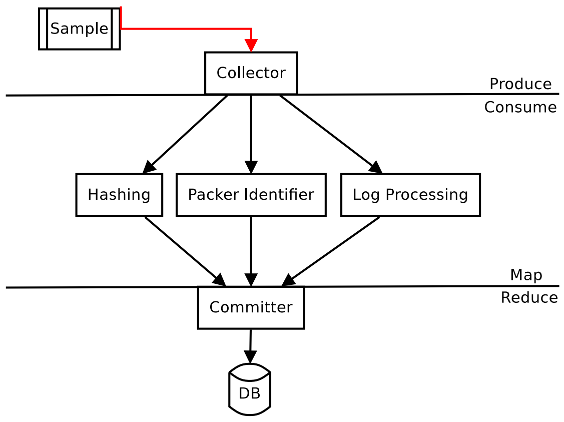

Figure 1. Parallel Processing Architecture. Samples are independently analyzed and their results are stored on a centralized database.

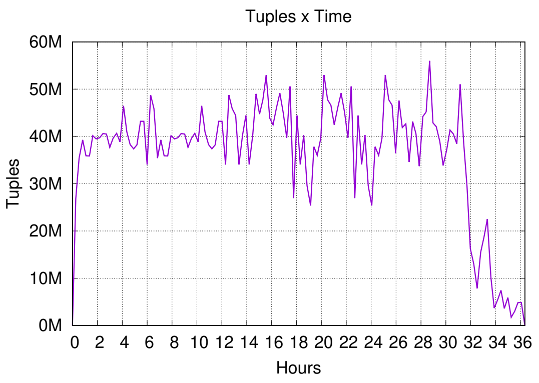

Figure 2. Parallel Processing Time. The complete analysis took 36 hours.

The collector is the mechanism responsible for crawling samples (producer) and scheduling them for analysis (mapper). The scheduler works by inserting the sample identifier on a queue of each proceeding step. By having individual queues, we allow each analyzer to run on their own speed, consuming data without delaying other analyzers. In total, we have 7 analyzers (binary hasher, static analyzer, AV analyzer, network protocol dissector, file-system log parser, process log parser and registry log parser), but we opted to represent only 3 of them for the sake of simplicity. When an analyzer finishes processing a given sample, the final information is sent to the committer (reduce), which stores such information on the database. By having a single committer, we avoid blocking the database for synchronization. The solution was implemented using `Python` and an in-memory `SQLite` database running on a multi-core server.

In total, the analysis procedure of all samples took 36 hours, as shown in Figure 2. When the solution starts, few tuples are inserted as the first stage have not parsed enough data yet to fill the whole analysis buffer. As the analysis proceeds, the insertion rate is increased. On average, our system was able to sustain a rate of 30M tuple insertions per hour.

4. Analysis Results

In this section, we present and compare results of performing both coarse and fine-grained sample analysis.

4.1. Coarse-grained Analysis

In this section, we present an overview of our dataset using the coarse-grained analysis procedure. Our goal is to identify the main characteristics present in the samples set. To do so, we considered static and dynamic analysis tools outputs as well as the network traffic generated during samples execution.

Our dataset is entirely composed by 32-bit samples. Among these, only 0.35% are DLLs, indicating an attackers' preference by self-contained executables. Attackers also prefer to rely on standard system libraries (60.42%) instead on third party ones. The functions in these libraries are used to implement a myriad of behaviors, as presented in Table 1.

Table 1. Most Imported Functions. Many distinct malicious behaviors are implemented using default system libraries.

Class	Function	Samples (%)	Class	Function	Samples (%)
Termination	Exitprocess	80.86%	Window	Messageboxa	50.70%
	Getexitcodeprocess	37.16%		Dispatchmessagea	50.25%
	Deletefilea	35.90%		Createwindowexa	48.76%
	Terminateprocess	35.26%		Showwindow	46.32%
	Exitwindowsex	34.87%		Getwindowrect	27.22%
	Enddialog	27.44%		Setwindowpos	26.80%
	Closeclipboard	24.63%		Iswindowvisible	25.72%
Timing	Sleep	76.48%		Iswindowenabled	24.60%
	Settimer	28.30%	Low Level	Getprocaddress	94.22%
Performance	Gettickcount	67.02%		Getmodulehandlea	73.72%
	Queryperformancecounter	43.02%		Getcommandlinea	73.60%
Fingerprinting	Getsystemmetrics	39.29%		Getcurrentprocess	69.36%
	Isprocessorfeaturepresent	28.26%		Isvalidcodepage	30.90%
Process	Getcurrentthreadid	53.23%		Encodepointer	26.82%
	Getcurrentprocessid	40.29%		Decodepointer	26.80%
Removal	Removedirectorya	34.44%	Anti Debug	Isdebuggerpresent	31.14%
	Deleteobject	27.46%	Modularization	Createprocessa	35.00%

We notice that most samples implement some kind of protection and/or evidence removal procedures. By performing process termination, a sample may, for instance, kill an AV solution. In addition, samples may also delete log files to cover their infection steps.

Moreover, samples may also protect themselves by evading analysis procedures. We observe both direct and indirect evasion attempts. Some samples opt to directly query system about attached debuggers. Other samples opt to indirectly identify whether they are under analysis, such as by querying system time and detecting possible performance penalties imposed by monitoring solutions.

Regarding implementation, we discovered that most samples opt for implementing graphic interfaces for user interaction instead of relying on stealth, background-based processes. We also discovered that they opt to directly handle system internal structures, such as pointers and pages, than relying on high-level abstractions.

Nevertheless, the aforementioned rates of malicious behaviors may only be considered as lower bounds, as samples may be obfuscated, thus hiding their real libraries and functions imports. In total, 81.90% of all samples presented some code generation signature, either from a compiler (53.01%) or from a packer (46.99%). The most prevalent signatures are presented in Table 2.

Table 2. Most Prevalent Signatures. Compilers are more prevalent than packers.

Signature	Type	Occurrence (%)
Microsoft	Compiler	37.95%
Nullsoft PIMP	Installer	25.51%
Borland Delphi	Compiler	15.06%
UPX	Packer	4.23%
MSLHR	Packer	2.25%
PEcompact	Packer	1.66%

We observe that most samples are not packed, thus presenting ordinary compilers signatures, such as `Microsoft` and `Delphi`. The `Microsoft` signature is the most common, as most samples are written in C/C++. In addition, whereas most samples are not packed, a significant number of samples present at least one packer signature. The `PIMP` installer may be considered as a packer. Its use is associated to trojanized applications installations. In addition, standalone malware samples are distributed by using non-installer packers, such as `UPX`.

To overcome the challenge of analyzing obfuscated samples, we leveraged dynamic analysis procedures. Table 3 presents identified malicious behaviors and their associated use rate.

Table 3. Dynamic Analysis. Identified Malicious Behaviors.

Subsystem	Operation	Samples (%)	Target	Samples (%)
File Subsystem	Create Files	91.56	Internet Explorer	10.14%
	Read Files	89.18%	.DLL	86.80%
			Internet Explorer	7.01%
			.SYS	1.26%
	Write Files	81.74%	.EXE	46.20%
			.DLL	31.62%
			Internet Explorer	<0.01%
			Host	0.00%
	Delete Files	62.45%	Internet Explorer	0.00%
Process Subsystem	Create Process	22.84%		
	Delete Process	23.38%		
Registry Subsystem	Set Registry Values	74.73%	Proxy	68.36%
			Autorun	5.66 %
	Delete Registry Values	55.43%		

We observe that the samples in our dataset present high level of system interaction, as most samples interact with more than one distinct OS subsystem. Whereas some observed behaviors are generic, such as reading and writing on temporary files, others are clearly suspicious. For instance, reading DLL files may be considered as legitimate, as process need to load them in their program space. Reading `.sys` files, in turn, may be associated to rootkit loading. Moreover, whereas reading a DLL may be considered legitimate, writing to them is considered malicious, as it is related to native system libraries replacement. In addition to DLLs, other binary files were also written. Such combination derives from the execution of downloader samples [Rossow et al. 2013].

Despite presenting this downloader characteristic, few processes were created.

Furthermore, few other expected malicious behaviors were not prevalent, such as writing the `host` file. It is explained by attackers moving their redirection strategy to OS proxies, as observed in the prevalent written registry keys.

As for static analysis results, the identified behaviors should also be considered as lower bounds, as dynamic analysis may also be evaded. We identified 80% of samples may implement some kind of anti-debug technique. Table 4 presents the identified techniques and their distribution among all identified tricks.

Table 4. Identified Anti-Debugger Techniques.

Technique	Occurrence (%)
Getlasterror	95.41%
Unhandledexceptionfilter	57.76%
Raiseexception	47.77%
Terminateprocess	43.99%
Isprocessorfeaturepresent	35.24%
Isdebuggerpresent	31.14%
Findwindowex	30.90%
Outputdebugstring	23.48%

Table 5. Identified Anti-VM techniques.

Technique	Occurrence(%)
VMcheck.dll	93.55%
Bochs & QEMU cpuid	14.59%
VMware	3.63%
VirtualBox	3.29%
VirtualPC	0.03%

Furthermore, 18% of samples present some kind of anti-virtualization technique. Table 5 presents the identified anti-VM techniques and their distribution among all techniques,

Regarding their network behavior, 60.69% of samples contacted at least 1 IP address, indicating most samples rely on Internet to achieve their malicious goals. On average, each sample contacts 2 distinct IP addresses.

Figure 3 shows the protocol distribution for all samples. We observe that most traffic is carried in TCP protocols, mainly due to HTTP connections. Similarly, DNS traffic dominates UDP traffic.

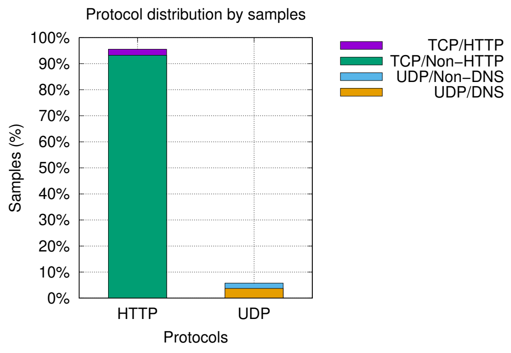

Figure 3. Protocol Distribution by Sample. HTTP and DNS are prevalent.

Table 6. Most Contacted Domains. We observe the presence of cloud providers among the most accessed domains.

Domain	Accesses (%)
Cloudfront	4.39%
Amazonaws	3.32%
Kirov	3.20%
Kerch	3.11%
Comcast	1.75%
Akamaitechnologies	1.48%
Sbcglobal	1.10%
Broadband	1.08%

Table 6 presents the most contacted domains. We notice that cloud providers are among the most accessed domains, which indicates an attackers trend of moving their operations to these environments.

The number of host providers also affects the list of accessed top-level domains, presented in Figure 4. The generic ones (`.net` and `.com`) are the most prevalent whereas the country-related ones are more well distributed, indicating malware creators are ready to exploit vulnerabilities in multiple countries.

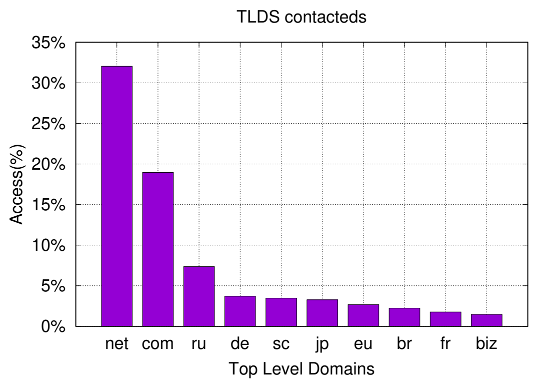

Figure 4. Most Accessed TLDs Distribution. Generic domains are prevalent.

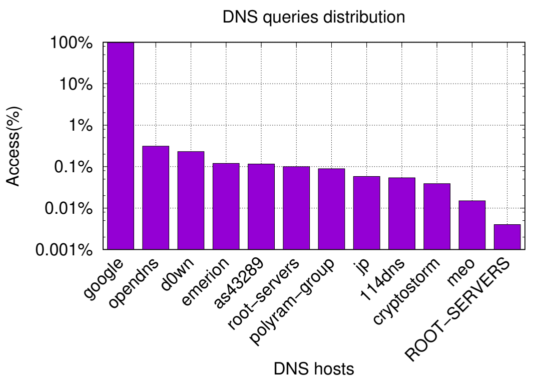

Figure 5. DNS Resolvers Distribution. Default sandbox DNS (`google`) was the most used one.

Regarding the DNS queries, most samples query the default server (google), as presented in Figure 5. This is an import project decision as relying on the system standard server makes samples more easily subject to subversion, making analysis procedures easier.

4.2. Fine-Grained Analysis

In this section, we consider the same aforementioned analysis outputs to perform a fine-grained evaluation. Our goal is to explain the observed behavior and trends in addition to pinpointing them.

To better understand how coarse and fine-grained analysis differ, consider the identified results for the registry subsystem. The coarse-grained analysis identified ≈5% of samples write on `AutoRun` keys. However, such kind of analysis does not specify where such keys are located. We present, in Table 7, distinct written key paths.

Table 7. Modified AutoRun Keys. Some keys are more affected than others.

Key	Samples(%)
HKCU\ID\Software\Microsoft\Windows\CurrentVersion\Run	46.33%
HKCU\.DEFAULT\SOFTWARE\Microsoft\Windows\CurrentVersion\Run	< 0.01%

The identified paths reflect distinct project decisions. Whereas majority of samples writes affect the current user (ID), thus having local impact, a single identified sample writes on the `.default` key. By doing so, it schedules its binary to be executed in the local context, affecting, for instance, the logon procedure [Microsoft 2007].

The presented distinction between the two analysis methods regarding the registry subsystem extends for the remaining data. Coarse-grained analysis identified each sample contacts 2 distinct IP addresses, on average. However, individual samples present characteristics which may significantly differ from the average. This is true even for samples which are supposed to implement the same malicious behavior. Table 8 shows the number of distinct IPs contacted by 3 distinct families ransomware samples.

Table 8. Contacted IPs by Sample. Ransomware may be implemented in distinct ways.

MD5 Hash	Number of Distinct IPs	Label
c1abb496deb7bd51a4ad2f8a43113b13	16386	Ransomware.Cerber
bc88096e7cc09f02f11deec35f84d5cd	16385	Ransomware.AWA
a801cdef09a61d3ba7969015a8bffec0	1	Ransomware.VirLock

We notice that although all samples are ransomware, their strategies are very distinct. The two first samples propagate by scanning the network, thus the high number of contacted IPs. The last one presents a distinct spreading mechanism, contacting only its C&C for infection notification.

Similarly, coarse-grained analysis states HTTP traffic dominates TCP payloads. Fine-grained analysis, however, may specify which payload is carried by such traffics. Table 9 shows the number of distinct HTTP requests performed by the most network-intensive samples.

Table 9. Prevalent HTTP Requests. Downloaders perform many distinct connections.

MD5 Hash	Total of Distinct HTTP Request	Label
ede13f40a96a8b6e5de1029200c0b15e	394	Downloader
e5f4116d08c343623d5ee3af5553cbee	353	Downloader
47a328b0b903bb68147facc3a084172c	310	Downloader
28c4e2a48d9ddfffa01a943ca1ba1262	304	Downloader

We notice that these samples perform many HTTP requests, which is explained by their downloaders nature, which makes them to contact multiple servers to retrieve their multiple payloads. This result shows that combining both approaches allows not only identifying the average use of network resources but also the behavior of specific malware families within the dataset. Particular malware families may be also identified through fine-grained analysis of their DNS traffic, as shown in Table 10.

Table 10. The percentage a query was sent to a DNS resolver by a sample and the description of the resolver

Query	Contacted(%)	Description
bmp.pilenga.co.uk.	12.29%	Hijacked Subdomain — Andromeda Botnet
tgr.tecnoagenzia.eu.	5.96%	Hijacked Subdomain — Andromeda Botnet
tds.repack.it.	2.48%	Andromeda Botnet
rxxl.tecnoagenzia.eu.	2.06%	Andromeda Botnet
and31.blllaaaaaazblaaa1.com.	0.92%	Andromeda Botnet

We discovered the most prevalent resolved domains are associated to a large botnet named Andromeda [Avast 2016]. Botnets rely on frequent C&C communication to receive commands from their botmasters.

5. Discussion

In this work, we presented analysis results of a large scale malware dataset to compare coarse-grained and fine-grained approaches. While adopting coarse-grained analysis methods, we were able to draw a broad panorama about the collected samples: Our evaluated dataset is mainly composed by executables—with few libraries—, which rely on system native libraries—with few external ones—, graphical user interfaces and present strong system interactions (file and registry creation/deletion).

Drawing panoramas allows us to compare multiple scenarios to understand their differences. For instance, we can compare our worldwide-collected samples to the ones found in the Brazilian scenario [Botacin et al. 2015]. In such, samples are presented as a mix of binaries and DLL, also relying on system native functions, but with background activity and fewer system interactions. Such characteristics are tied to Brazilian cultural characteristics not present in our global dataset.

Despite allowing drawing panoramas, coarse-grained analysis is not able to explain many samples project decisions. For instance, coarse-grained analysis of network results identifies each sample contacts, in average, 2 distinct IP addresses. Fine-grained analysis, in turn, is able to identify that a given sample (c1abb496deb7bd51a4ad2f8a43113b13) contacts 16386 IPs whereas other (a801cdef09a61d3ba7969015a8bffec0) accesses only a single one and to explain such distinction is due to samples distinct goals: The first sample is a ransomware which propagates by scanning the network, thus contacting many IPs. The second sample, in turn, despite also being a ransomware, presents a distinct spreading mechanism, thus contacting only its own C&C.

The previously presented cases illustrate that only a combined analysis approach allows us to achieve complete threat knowledge. In addition, we highlight that such fact is empowered by a large scale dataset, which exacerbates analysis distinctions. By relying on many samples, we were able to identify registry keys written only by one sample on a universe of more than 100 thousand samples.

6. Conclusion

In this work, we presented analysis results of a large scale malware dataset. We compared the results of two approaches—coarse and fine grained—, highlighting their differences. We discovered that whereas coarse-grained analysis allows us to draw threat panoramas, only fine-grained analysis enables us to understand samples internals. Therefore, only a combined approach allows complete security analysis treatment.

Acknowledgments

This work is supported by PIBIC-CNPQ 138239/2017-7 and FORTE-CAPES 23038.007604/2014-69.

References

Avast (2016). Andromeda under the microscope. `https://blog.avast.com/andromeda-under-the-microscope`.

Bayer, U., Habibi, I., Balzarotti, D., Kirda, E., and Kruegel, C. (2009). A view on current malware behaviors. In *USENIX LEET*.

BBC (2017). Ransomware attacks around the world grow by 50 `http://www.bbc.com/news/technology-39730407`.

Botacin, M., Grégio, A., and de Geus, P. (2015). Uma visão geral do malware ativo no espaço nacional da internet entre 2012 e 2015. `https://siaiap34.univali.br/sbseg2015/anais/WFC/artigoWFC02.pdf`.

Botacin, M. F., de Geus, P. L., and Grégio, A. R. A. (2017). The other guys: automated analysis of marginalized malware. *Journal of Computer Virology and Hacking Techniques*.

Lindorfer, M., Neugschwandtner, M., Weichselbaum, L., Fratantonio, Y., Veen, V. v. d., and Platzer, C. (2014). Andrubis – 1,000,000 apps later: A view on current android malware behaviors. In *BADGERS '14*. IEEE.

Microsoft (2007). The .default user is not the default user. `https://blogs.msdn.microsoft.com/oldnewthing/20070302-00/?p=27783`.

Rossow, C., Dietrich, C., and Bos, H. (2013). Large-scale analysis of malware downloaders. In *DIMVA*.

Sikorski, M. and Honig, A. (2012). *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. No Starch Press, San Francisco, CA, USA, 1st edition.

TrendMicro (2017). Erebus linux ransomware: Impact to servers and countermeasures. `https://www.trendmicro.com/vinfo/us/security/news/cyber-attacks/erebus-linux-ransomware-impact-to-servers-and-countermeasures`.