BazarLoader Technical Analysis Report

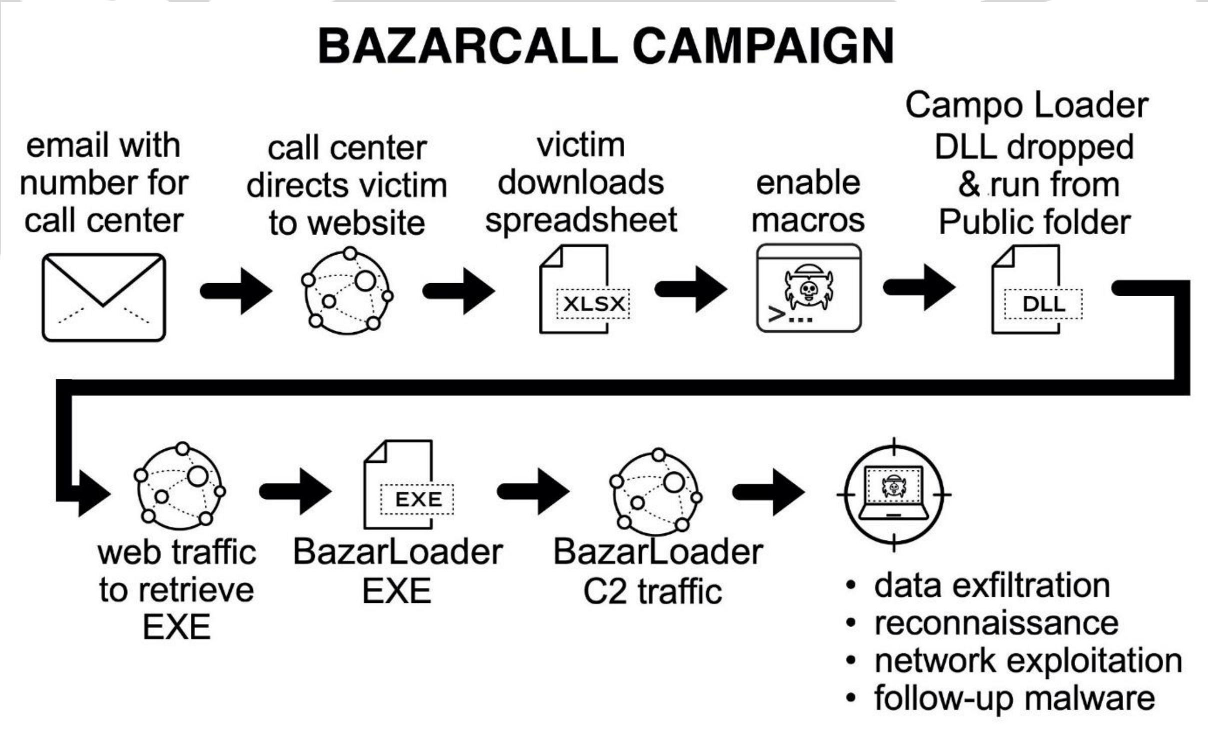
ZAYOTEM

Table of Contents

INTRODUCTION

BazarLoader (also known as BazaLoader) is a family of malware that creates a backdoor for infected Windows host systems. Developed by TA800. By creating a backdoor, it tries to install malware and infiltrate other systems on the network to exploit these vulnerabilities by finding vulnerabilities in the systems entered.

BazarLoader, which is spread with different vectors, is generally transmitted to users by e-mail, but in February 2021, phishing attacks with call centers were transmitted to users. Such phishing calls are also called "**BazarCall**". They try to inject malware into systems by calling users and offering free trials.



File Name	1f6e8b2f989cc0ce80baa52acc0b3986.dll
MD5	1F6E8B2F989CC0CE80BAA52ACC0B3986
SHA256	bc8407aa092b9b316e72b6082699dd1432521f739eacfb57109bb1d759d89802
SHA1	6fc636cd696a77c590727f512cd4ce02da55d984
First Seen	2021-07-12 06:28:35 UTC

As mentioned in the introduction, the malware infects the user's device in various ways and starts to perform its harmful operations by running it as follows.

Since the malware we have is a DLL, it needs a host application, so **rundll32.exe** is given as a parameter to the legal Windows application and run by **cmd.** After this run, it was seen that, together with many techniques, it also connected to The Command and Control Servers by running **svchost.exe** a legal application of Windows, and running the code injected into it through thread. As is known, **svchost** is a legal application that runs to run system services in Windows. By injecting the malware into this application, it also provides persistence. You can see the process tree created by the malware below.

(System Win7 x64)

- cmd.exe (cmdline: cmd.exe /C rundll32.exe 'C:\Users\user\Desktop\1f6e8b2f989cc0ce80baa52acc0b3986.dll',#1 MD5: 4E2ACF4F8A396486AB4268C94A6A245F)
  - rundll32.exe (cmdline: rundll32.exe 'C:\Users\user\Desktop\1f6e8b2f989cc0ce80baa52acc0b3986.dll',#1 MD5: 73C519F050C20580F8A62C849D49215A)
- rundll32.exe (cmdline: C:\Windows\System32\rundll32.exe C:\Users\user\Desktop\1f6e8b2f989cc0ce80baa52acc0b3986.dll,StartW 2791350475 MD5: 73C519F050C20580F8A62C849D49215A)
  - svchost.exe (cmdline: C:\Windows\system32\svchost.exe -k UnistackSvcGroup MD5: 32569E403279B3FD2EDB7EBD036273FA)

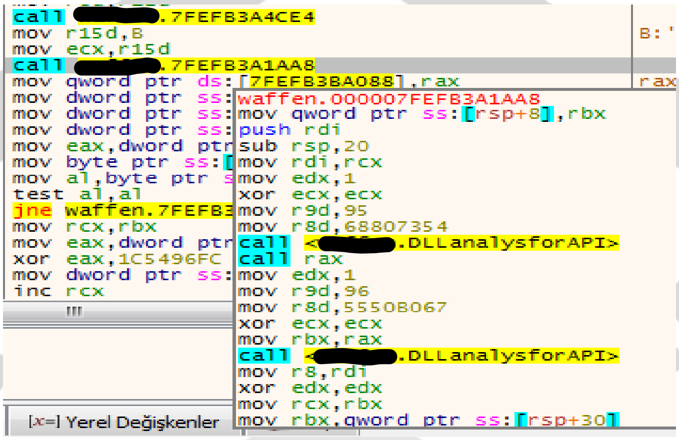
API Hammering

API Hammering is a technique used to delay sandbox analysis and reduce the capacity of malware technical analyses. It makes analysis difficult by using certain APIs tens of thousands of times as variables. When looking at sandbox algorithms, algorithms based on record keeping prevent the actual block of malicious code, called **delay execution,** from running with overload. For example, a malware that makes 2 million calls is encoded to run the actual block of malicious code as a result of these calls. After a certain period of time, as a result of so many calls, the records sandbox keeps will be filled with completely unnecessary data, and the actual malicious code will not work.

The number of API calls received from a sample that used this technique:

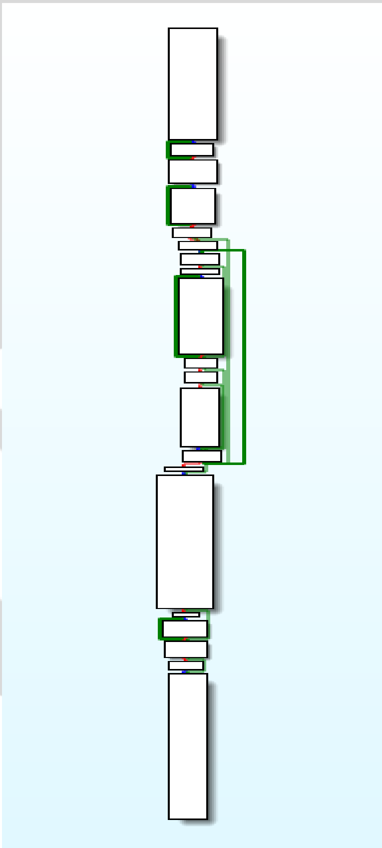
API Name	Number of Calls
➢ KERNEL32.dll.GetLastError	49739
➢ USER32.dll.GetDlgItem	34446
➢ KERNEL32.dll.TlsGetValue	34434
➢ KERNEL32.dll.SetLastError	34434
➢ dbghelp.dll.SymCleanup	30608
➢ USER32.dll.ShowWindow	30608
➢ KERNEL32.dll.GetCurrentProcess	30608
➢ KERNEL32.dll.LeaveCriticalSection	15306
➢ KERNEL32.dll.EnterCriticalSection	15306
➢ KERNEL32.dll.CloseHandle	15305
➢ USER32.dll.FindWindowExA	15304
➢ GDI32.dll.MoveToEx	15304
➢ USER32.dll.GetClassNameA	15304
➢ PSAPI.DLL.GetPerformanceInfo	15304
➢ USER32.dll.SetWindowPlacement	15304
➢ KERNEL32.dll.GlobalMemoryStatusEx	15304
➢ USER32.dll.PostMessageA	15304
➢ PSAPI.DLL.EnumProcesses	15304
➢ KERNEL32.dll.GetVersionExA	15304
➢ dbghelp.dll.SymInitialize	15304…

In this way, it is getting difficult to making manual analyzing the uses and parameters of the malicious APIs used.

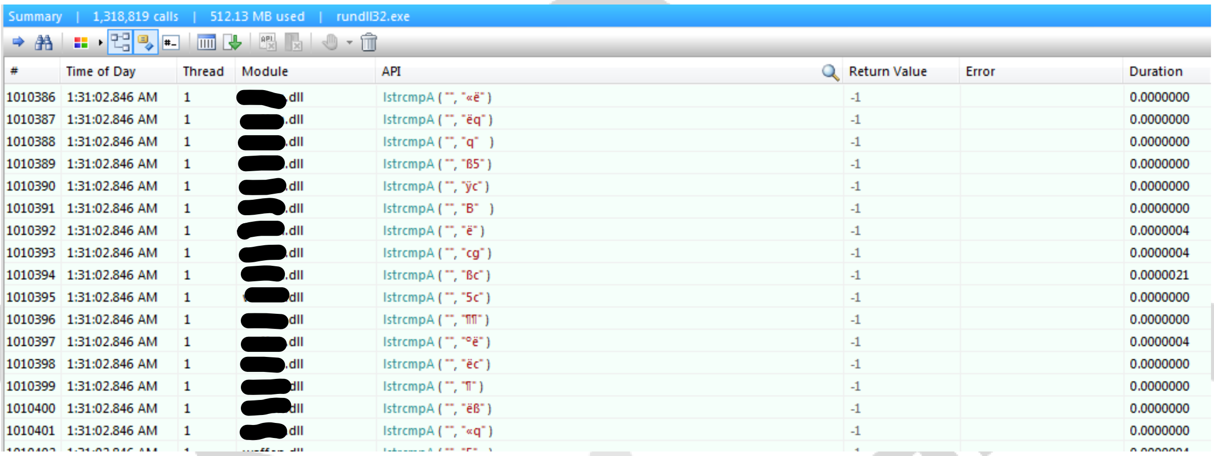


At the same time, it dynamically uses DLL interpretation and "parse" in malware to hide from analysts which block of code the actual malicious APIs will be used and when.

As in the IDA image seen below, **API Hammering** is applied by making hundreds of thousands of API calls with many near-infinity loops.



The number of CALLS made by the malware in a short time and the memory size used appear in this photo:



Summary | 1,318,819 calls | 512.13 MB used | rundll32.exe

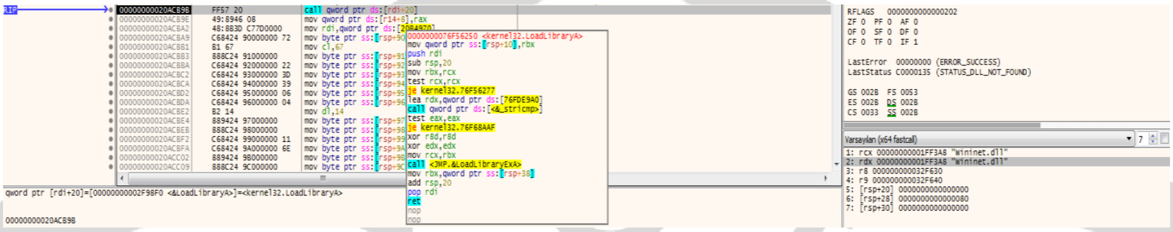
#	Time of Day	Thread	Module	API	Return Value	Error	Duration
1010386	1:31:02.846 AM	1	[redacted].dll	lstrcmpA ("", "«ë")	-1		0.0000000
1010387	1:31:02.846 AM	1	[redacted].dll	lstrcmpA ("", "ëq")	-1		0.0000000
1010388	1:31:02.846 AM	1	[redacted].dll	lstrcmpA ("", "q")	-1		0.0000000
1010389	1:31:02.846 AM	1	[redacted].dll	lstrcmpA ("", "ß5")	-1		0.0000000
1010390	1:31:02.846 AM	1	[redacted].dll	lstrcmpA ("", "ÿc")	-1		0.0000000
1010391	1:31:02.846 AM	1	[redacted].dll	lstrcmpA ("", "B")	-1		0.0000000
1010392	1:31:02.846 AM	1	[redacted].dll	lstrcmpA ("", "ë")	-1		0.0000004
1010393	1:31:02.846 AM	1	[redacted].dll	lstrcmpA ("", "cg")	-1		0.0000004
1010394	1:31:02.846 AM	1	[redacted].dll	lstrcmpA ("", "ßc")	-1		0.0000021
1010395	1:31:02.846 AM	1	[redacted]dll	lstrcmpA ("", "5c")	-1		0.0000000
1010396	1:31:02.846 AM	1	[redacted]dll	lstrcmpA ("", "¶¶")	-1		0.0000000
1010397	1:31:02.846 AM	1	[redacted]dll	lstrcmpA ("", "ºë")	-1		0.0000004
1010398	1:31:02.846 AM	1	[redacted].dll	lstrcmpA ("", "ëc")	-1		0.0000000
1010399	1:31:02.846 AM	1	[redacted]dll	lstrcmpA ("", "¶")	-1		0.0000000
1010400	1:31:02.846 AM	1	[redacted]dll	lstrcmpA ("", "ëß")	-1		0.0000000
1010401	1:31:02.846 AM	1	[redacted]dll	lstrcmpA ("", "«q")	-1		0.0000000

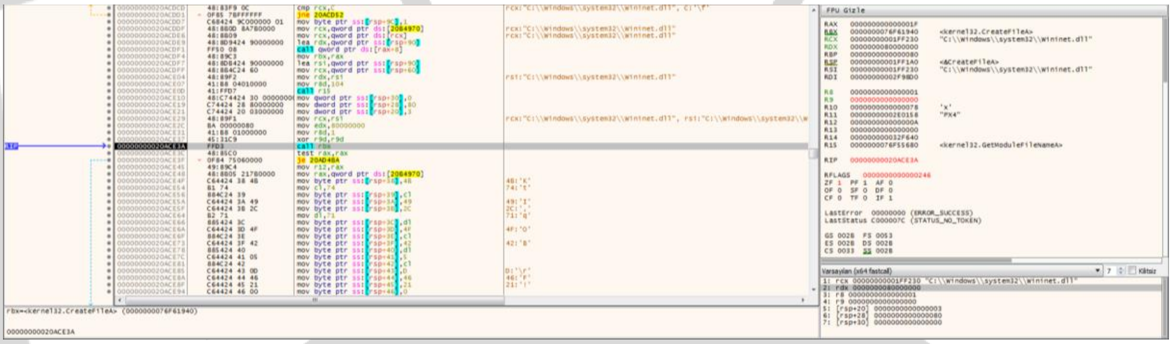
The following table shows the DLLs that are loaded into their memory block by our malware.

File Path	API
C:\Windows\System32\kernel32.dll	ReadFile
C:\Windows\System32\wininet.dll	ReadFile
C:\Windows\System32\advapi32.dll	ReadFile
C:\Windows\System32\ole32.dll	ReadFile
C:\Windows\System32\ntdll.dll	ReadFile
C:\Windows\System32\shell32.dll	ReadFile
C:\Windows\System32\bcrypt.dll	ReadFile
C:\Windows\System32\crypt32.dll	ReadFile
C:\Windows\System32\dnsapi.dll	ReadFile
C:\Windows\System32\netapi32.dll	ReadFile
C:\Windows\System32\shlwapi.dll	ReadFile
C:\Windows\System32\user32.dll	ReadFile
C:\Windows\System32\ktmw32.dll	ReadFile

So how does it upload these DLLs to his memory?

After dynamic DLL analysis, the malware that loads the DLLs into its memory in the **LoadLibrary-CreateFile-ReadFile** order keeps the initial addresses of the required APIs in its memory by parsing these DLLs after loading them into their memory.





It does not keep the APIs to be used in this way, it keeps in a **hard-coded** way in its own memory, making static analysis difficult. It also hides when it will combine with API Hammering technique and then use it.

Thread content injected and operated into Svchost:

ntdll.dll!ZwWaitForSingleObject
KernelBase.dll!WaitForSingleObjectEx
wininet.dll!InternetSetStatusCallbackW
wininet.dll!GetUrlCacheHeaderData
wininet.dll!InternetSetStatusCallbackA
wininet.dll!HttpOpenDependencyHandle
wininet.dll!InternetCanonicalizeUrlW
wininet.dll!AppCacheGetManifestUrl
wininet.dll!InternetConfirmZoneCrossingW
wininet.dll!HttpSendRequestA
kernel32.dll!BaseThreadInitThunk
ntdll.dll!RtlUserThreadStart
wininet.dll!InternetSetStatusCallbackA

BazarLoader is known to be a family of malware that creates a backdoor. So how to provide this backdoor?

The injected code periodically discards **HTTP** requests to certain command and control servers that it stores in Svchost's memory. As shown in the procmon image below, it periodically discards requests by creating threads.

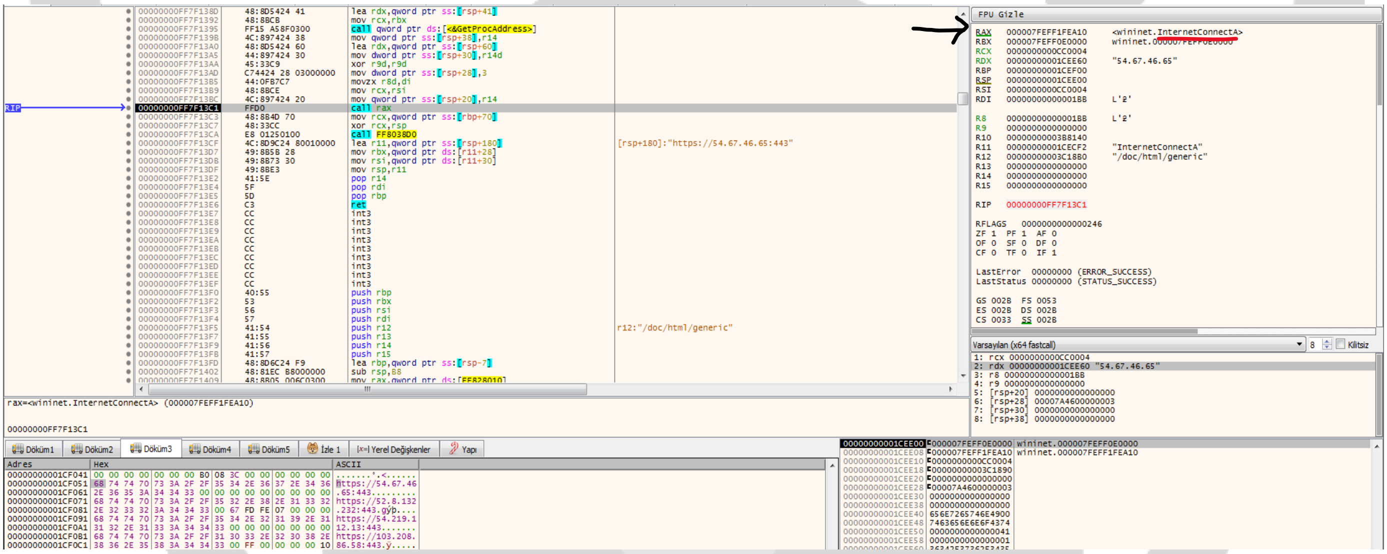
svchost.exe	2216	TCP Reconnect	[redacted]localdomain:62135 -> ec2-54-219-112-13.us-west-1.compute.amazonaws.com:https
svchost.exe	2216	TCP Reconnect	[redacted]localdomain:62135 -> ec2-54-219-112-13.us-west-1.compute.amazonaws.com:https
svchost.exe	2216	TCP Reconnect	[redacted]localdomain:62136 -> ec2-54-219-112-13.us-west-1.compute.amazonaws.com:https
svchost.exe	2216	TCP Reconnect	[redacted]localdomain:62136 -> ec2-54-219-112-13.us-west-1.compute.amazonaws.com:https
svchost.exe	2216	Thread Exit	
svchost.exe	2216	Thread Exit	
svchost.exe	2216	RegQueryKey	HKCU
svchost.exe	2216	RegCreateKey	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Connections
svchost.exe	2216	RegQueryValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Connections\DefaultConnectionSettings
svchost.exe	2216	RegQueryValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Connections\DefaultConnectionSettings
svchost.exe	2216	RegQueryValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Connections\DefaultConnectionSettings
svchost.exe	2216	RegCloseKey	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Connections
svchost.exe	2216	RegQueryKey	HKLM
svchost.exe	2216	RegOpenKey	HKLM\SYSTEM\CurrentControlSet\Services\NetBT\Linkage
svchost.exe	2216	RegOpenKey	HKLM\System\CurrentControlSet\Services\NetBT\Linkage
svchost.exe	2216	RegQueryValue	HKLM\System\CurrentControlSet\services\NetBT\Linkage\Export
svchost.exe	2216	RegQueryValue	HKLM\System\CurrentControlSet\services\NetBT\Linkage\Export
svchost.exe	2216	RegQueryValue	HKLM\System\CurrentControlSet\services\NetBT\Linkage\Export
svchost.exe	2216	RegCloseKey	HKLM\System\CurrentControlSet\services\NetBT\Linkage
svchost.exe	2216	TCP Reconnect	[redacted]localdomain:62137 -> ec2-54-219-112-13.us-west-1.compute.amazonaws.com:https
svchost.exe	2216	Thread Create	
svchost.exe	2216	TCP Reconnect	[redacted]localdomain:62137 -> ec2-54-219-112-13.us-west-1.compute.amazonaws.com:https
svchost.exe	2216	TCP Reconnect	[redacted]localdomain:62138 -> ec2-54-219-112-13.us-west-1.compute.amazonaws.com:https

Network Analysis

BazarLoader malware is known to be a backdoor provider. **How to provide this backdoor and what is it?**

As a result of the command received on the leaked device, it is used for many purposes such as running code, stealing information from the system, monitoring system movements, keylogger. In fact, all traffic of the actual device passes through the command and control server because all its movements can be monitored and interfered with by this server. Different malware can be downloaded and run to cause greater damage at any time. For this, the command is expected by keeping the connection active continuously.

The Backdoor appears to have contacted certain command and control servers with the **InternetConnectA** API to run commands and keep the connection alive.



With thread that created in Svchost, malware connects "xxx.xxx.xxx.xxx.us-west[.]-1[.]compute[.]amazonaws[.]com" domains periodically.

Process	PID	Protocol	Local Address	Local Port	Remote Address	Remote Port	State
lsass.exe	496	TCP		49155		0	LISTENING
lsass.exe	496	TCPV6		49155		0	LISTENING
rundll32.exe	3504	TCP		62134	ec2-34-213-41-242.us-west-2.compute.amazonaws.com	https	ESTABLISHED
services.exe	488	TCP		49156		0	LISTENING
services.exe	488	TCPV6		49156		0	LISTENING
svchost.exe	684	TCP		epmap		0	LISTENING
svchost.exe	764	TCP		49153		0	LISTENING
svchost.exe	864	TCP		49154		0	LISTENING
svchost.exe	2432	UDP		ssdp	*	*	
svchost.exe	2432	UDP		ssdp	*	*	
svchost.exe	840	UDP		ws-discovery	*	*	
svchost.exe	840	UDP		ws-discovery	*	*	
svchost.exe	2432	UDP		ws-discovery	*	*	
svchost.exe	2432	UDP		ws-discovery	*	*	
svchost.exe	308	UDP		llmnr	*	*	
svchost.exe	2432	UDP		55522	*	*	
svchost.exe	840	UDP		58361	*	*	
svchost.exe	2432	UDP		62618	*	*	
svchost.exe	2432	UDP		62619	*	*	
svchost.exe	684	TCPV6		epmap		0	LISTENING
svchost.exe	764	TCPV6		49153		0	LISTENING
svchost.exe	864	TCPV6		49154		0	LISTENING
svchost.exe	764	UDPV6		546	*	*	
svchost.exe	2432	UDPV6		1900	*	*	
svchost.exe	2432	UDPV6		1900	*	*	
svchost.exe	2432	UDPV6		3702	*	*	
svchost.exe	840	UDPV6		3702	*	*	
svchost.exe	2432	UDPV6		3702	*	*	
svchost.exe	840	UDPV6		3702	*	*	
svchost.exe	308	UDPV6		5355	*	*	
svchost.exe	2432	UDPV6		55523	*	*	
svchost.exe	840	UDPV6		58362	*	*	
svchost.exe	2432	UDPV6		62616	*	*	
svchost.exe	2432	UDPV6		62617	*	*	
svchost.exe	2216	TCP		62135	ec2-54-219-112-13.us-west-1.compute.amazonaws.com	https	SYN_SENT
System	4	TCP		netbios-ssn		0	LISTENING
System	4	TCP		microsoft-ds		0	LISTENING
System	4	TCP		wsd		0	LISTENING
System	4	UDP		netbios-ns	*	*	
System	4	UDP		netbios-dgm	*	*	

MITRE ATT&CK Table

Execution	Persistence	Privilege Escalation	Defense Evasion	Discovery	Command and Control	Collection
Shared Modules	Application Shimming	Process Injection	Masquerading	System Time Discovery	Encrypted Channel	Archive Collected Data
		Application Shimming	Virtualization/Sandbox Evasion	Security Software Discovery	Application Layer Protocol	
			Process Injection	Virtualization / Sandbox Evasion		
			Obfuscated Files or Information	Process Discovery		
			Rundll32	File and Directory Discovery		
			Software Packing	System Information Discovery		

Solution Suggestions

There are ways to protect against backdoor-type BazarLoader malware:

- Use of up-to-date, reliable anti-virus software in systems,
- Careful attention to incoming e-mails, not to open attachments unconsciously without analysis,
- Disregard of spam emails,
- Solutions such as creating Mutex objects on the system,

It can prevent backdoor type BazarLoader malware from infecting the system.

YARA Rule

```
import "hash"
import "pe"

rule FirstFile{
	meta:
		description="1f6e8b2f989cc0ce80baa52acc0b3986.dll"
	strings:
		$str1="LoadLibraryW"
		$str2="us-west-1.compute.amazonaws.com"
		$str3="54.67.46.65"
		$str4="52.8.132.232"
		$str5="54.219.112.13"
		$str6="103.208.86.56"
		$str7="InternetConnectA"
		$str8="InternetOpenA"
		$str9="HttpOpenRequestA"
		$str10="CreateMutex"
		$str11=”VirtualAllocA”

	condition:
		hash.md5(0,filesize) =="1F6E8B2F989CC0CE80BAA52ACC0B3986" or all of them
}
```

Fatih YILMAZ

https://www.linkedin.com/in/fatih-yilmaz-f8/