cybereason

Lab Analysis

Webmail Server APT: A New Persistent Attack Methodology Targeting Microsoft Outlook Web Application (OWA)

Written by:
Yonatan Striem-Amit, CTO, Co-Founder
Yoav Orot, Senior Researcher

A New Approach to APT

This document analyzes a real and unique APT technique detected by the Cybereason platform in one of our customer's environment. The attack involved a malicious module that was loaded onto Microsoft Outlook Web Application (OWA), an internet-facing web-mail server, which enabled the attackers to record authentication credentials and be provided with complete backdoor capabilities. By using this approach, the hackers managed to collect and retain ownership over a large set of credentials, allowing them to maintain persistent control over the organization's environment.

Introduction

A Cybereason POC customer suspected that it had an infected server due to several behavioral abnormalities spotted by its security team. They reached out to Cybereason and together we decided to deploy our platform across the customer's entire environment of 19,000 endpoints.

Discovering a Suspicious DLL

Within several hours, the Cybereason platform found a suspicious DLL loaded into the Outlook Web App (OWA) server (a webmail component of Microsoft Exchange Server), with several interesting characteristics. Although it had the same name as another benign DLL, the suspicious DLL went unsigned and was loaded from a different

directory. Since OWA servers typically load only legitimately signed DLLs, the Cybereason behavioral engine immediately elevated this event to a suspicion.

Backdooring a world-accessible server in the Demilitarized Zone

Contrary to other web servers that typically have only a web interface, OWA is unique: it is a critical internal infrastructure that also faces the Internet, making it an intermediary between the internal, *allegedly* protected DMZ, and the web. The customer was using OWA to enable remote user access to Outlook. This configuration of OWA created an ideal attack platform because the server was exposed both internally and externally. Moreover, because OWA authentication is based on domain credentials, ***whoever gains access to the OWA server becomes the owner of the entire organization's domain credentials.*** Later, we will see how the attacker extracted credentials from the OWA server using their malicious module.

Persistence, Hiding and Loading

The hackers installed a backdoored malicious OWAAUTH.dll which was used by OWA as part of the authentication mechanism, and was responsible for authenticating users against the Active Directory (A/D) server used in the environment. In addition, the malicious OWAAUTH.DLL also installed an ISAPI filter into the IIS server, and was filtering HTTP requests.

This enabled the hackers to get all requests in cleartext after SSL/TLS decryption. The malware replaced the OWAAUTH by installing an IIS filter in the registry, which enabled the malware to automatically load and persist on every subsequent server restart.

The final touch was the hackers choosing the .NET assembly cache. This folder was used to store locally compiled native binaries in order to accelerate the loading and execution of .NET applications. These locally compiled binaries were only used on the computer in which they were generated, and thus had no reputation or digital signatures. It is interesting to note that the hackers chose this folder exactly for that reason, in order to avoid human-driven inspection. The hackers attempted to fool the hunters into thinking that it was simply another locally generated file, as if they were Obi-Wan practicing a little Jedi magic, convincing the defender to think: *these are not the files you're looking for, move along.*

Hooking the Request Handlers

Once loaded, the DLL implemented an HTTP Module and registered two request handling function callbacks on its initialization:

- Application_BeginRequest() registered as BeginRequest - Not implemented
- Application_EndRequest() registered as EndRequest - Called at the end of the request handling chain. Implements the main malware logic.

At the end of each HTTP request handling in the OWA server, Application_EndRequest() was called, executing the malware backdoor main function.

Capturing OWA Authentication Tokens

The hacker's first goal was to use the visibility they had gained into the OWA authentication process to steal the passwords of users logging into OWA - namely everyone.

While most security professionals understand the sensitivity of data in the A/D server, the OWA server serves as a focal point for the exact same sensitive data.

Whenever Application_EndRequest() was called, it read the request variables using the Request sub-object of the HttpApplication object. It then checked whether there were two variables named "username" and "password". These variables were passed in the request query URL whenever a user logged into the OWA service.

If the function found the username and password, it recorded them together with a timestamp, the client host address, and the client user agent to an encrypted file named log.txt, stored in C:\.

The encryption algorithm was DES, implemented by the DESCryptoServiceProvider class, with both a password and IV values of the ASCII string "12345678". Following this discovery, we wrote a decryption tool for this file, and found more than 11,000(!) user/password pairs. **This treasure trove essentially gave the hackers complete access to every identity and therefore every asset in the organization.**

A Complete Backdoor

In order to get access to the usernames and passwords , and to ensure that the hackers could gain more control over the environment, the malware also possessed covert backdoor functionalities. The attacker used HTTP request query URL variables in order to use the backdoor functionality remotely.

The malware was searching every incoming request for a special parameter, in our case “<CustomerName>XXX”. When the parameter was found, the backdoor functionality would parse the remainder of the parameters. **Naturally, the fact that the hacker embedded the customer name in the parameters proves that this malware was tailored for this particular target.**

If the variable “<CustomerName>XXX” was found in the HTTP query URL, the code also looked for the two other variables - Z1 and Z2. Together, they represented a function call with specific parameters, as “<CustomerName>XXX” represented a function ID of a single ASCII character with possible values of “A” through “Q,” while Z1/Z2 were parameters of the function.

If the called function returned a value to the client, it would be in the form of a string formatted as: “->|ResultData|<-”, with ResultData as the returned data. The formatted result string would then be returned to the client through the HTTP response data.

As you can see in the following functionality map (Table-1), some basic backdoor capabilities were implemented in the malware. Also, there were several SQL control functions, enabling the attacker to read, write and execute command on SQL servers inside and outside the targeted organization.

Naturally, the facilities below allowed the hacker to write and execute any code on the OWA server, and by using the previously collected passwords, impersonate any user and perform lateral movement.

Action	<CustomerName>XXX	Z1	Z2
Get web root path and all logical devices on machine	'A'	None	None
Get directory file info list (path, last write time, length)	'B'	Dir path	None
Print given string back to client	'C'	A string	None
Write a text file with given path and data	'D'	File path	Data string
Delete directory/file	'E'	File path	None
Return to client the contents of a file at given path	'F'	File path	None
Write a binary file with given path and data	'G'	File path	Hex data text
Copy directory	'H'	Source dir path	Destination dir path
Move file/directory	'I'	Source path	Destination path
Create directory	'J'	Dir path	None
Set file/directory creation & last write & access times	'K'	File path	Date time
Send HTTP GET request with content type of	'L'	URL	File path

application/x-www-form-urlencoded to given url and write the response to given file path			
Run process as ProcessPath Param1 Param2. Return stdout & stderr to client.	'M'	Param1(len: 2B)+ProcessPath	Param2
Get SQL DB name.	'N'	SQL connection string	None
Get SQL table list (and no. of columns for each)	'O'	SQL connection string	None
Dump SQL columns names and types from a specified table catalog and table name	'P'	SQL connection string\n Catalog name\n table name	None
Dump SQL table or execute command	'Q'	SQL connection string	Table name or command

Table-1 - Backdoor functions map

Conclusions

The hackers in this case managed to gain a foothold into a highly strategic asset: the OWA server. Almost by definition, OWA requires organizations to define a relatively lax set of restrictions; and in this case, OWA was configured in a way that allowed internet-facing access to the server. This enabled the hackers to establish persistent control over the entire organization's environment without being detected for a period of several months.

By combining endpoint level visibility and the Cybereason Malop hunting engine's contextual analysis, the organization was finally able to visualize the entire attack and pinpoint the root cause of the threat.

Our customer's instincts were spot on when his "Spidey Sense" told him there was more to the OWA server's odd behavior than what met the eye, but it would have likely taken him weeks of uninterrupted analysis to investigate the situation manually. Only by automating detection and analysis was this clever hack detected, understood, and contained.

Herein lies the magic of Endpoint Detection and Response (EDR). The Cybereason platform immediately pinpointed how the OWA server was compromised and provided clear visibility to the affected machine, enabling our customer to clean and fix it. Cybereason was designed to empower smart people with good instincts to be top-notch cyber-defenders, and this is a perfect example of what that looks like.

Cybereason:
Let the Hunt Begin.

222 Berkeley Street
13th Floor
Boston, MA 02116 USA
www.cybereason.com

About Cybereason

Cybereason was founded in 2012 by a team of ex-military cybersecurity experts to revolutionize detection and response to cyber attacks. The Cybereason Malop Hunting Engine identifies signature and non-signature based attacks using big data, behavioral analytics, and machine learning. The Incident Response console provides security teams with an at-your-fingertip view of the complete attack story, including the attack's timeline, root cause, adversarial activity and tools, inbound and outbound communication used by the hackers, as well as affected endpoints and users. This eliminates the need for manual investigation and radically reduces response time for security teams. The platform is available as an on premise solution or a cloud-based service. Cybereason is privately held and headquartered in Boston, MA with offices in Tel Aviv, Israel.