



Authors: Or Yair, Security Research Team Lead | Shahak Morag, Research Lead

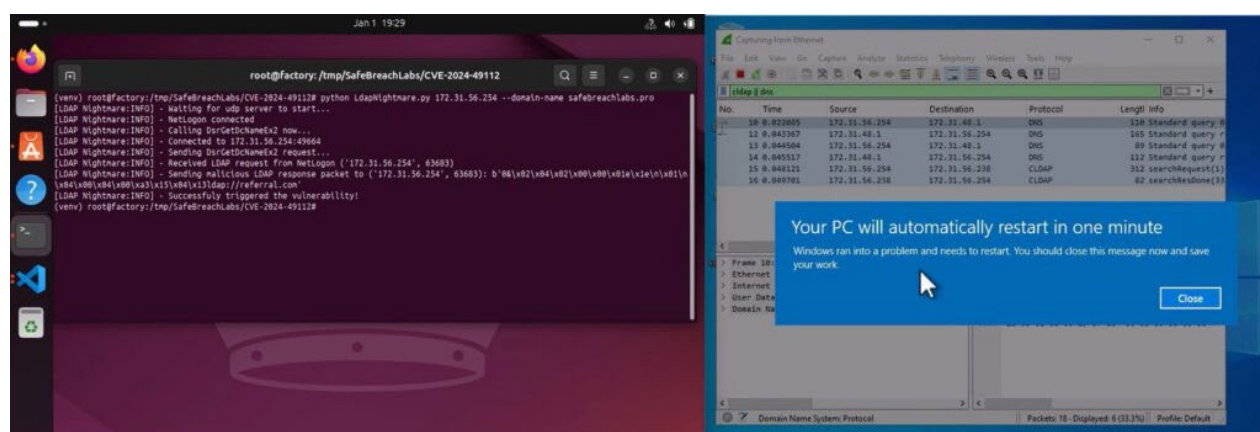
Active Directory Domain Controllers (DCs) are considered to be one of the crown jewels in organizational computer networks. Vulnerabilities found in DCs are usually much more critical than those found in usual workstations. The ability to run code on a DC or crash Windows servers heavily affects network security posture;.

On December 10, 2024, two LDAP vulnerabilities were found by Yuki Chen (@guhe120): a remote code execution (RCE) and a Denial Of Service/Info Leak that both affect any DC was published on the Microsoft Security Response Center (MSRC) website as part of the latest Patch Tuesday update. The RCE vulnerability was assigned as CVE-2024-49112 and was given the CVSS severity score of 9.8 out of 10; the DOS vulnerability was assigned CVE-2024-49113. However, a public exploit or blogpost explaining the vulnerability or exploitation path was not published anywhere for either.

The SafeBreach Labs team regularly undertakes projects that we feel can help both our customers, who represent some of the largest brands in the world, and the security community at large. Given the severity of this vulnerability's consequences and the attention it has received from both since it was fixed, we decided as a team to prioritize it and are proud of the findings we have identified that will help enterprises address any potential exposures.

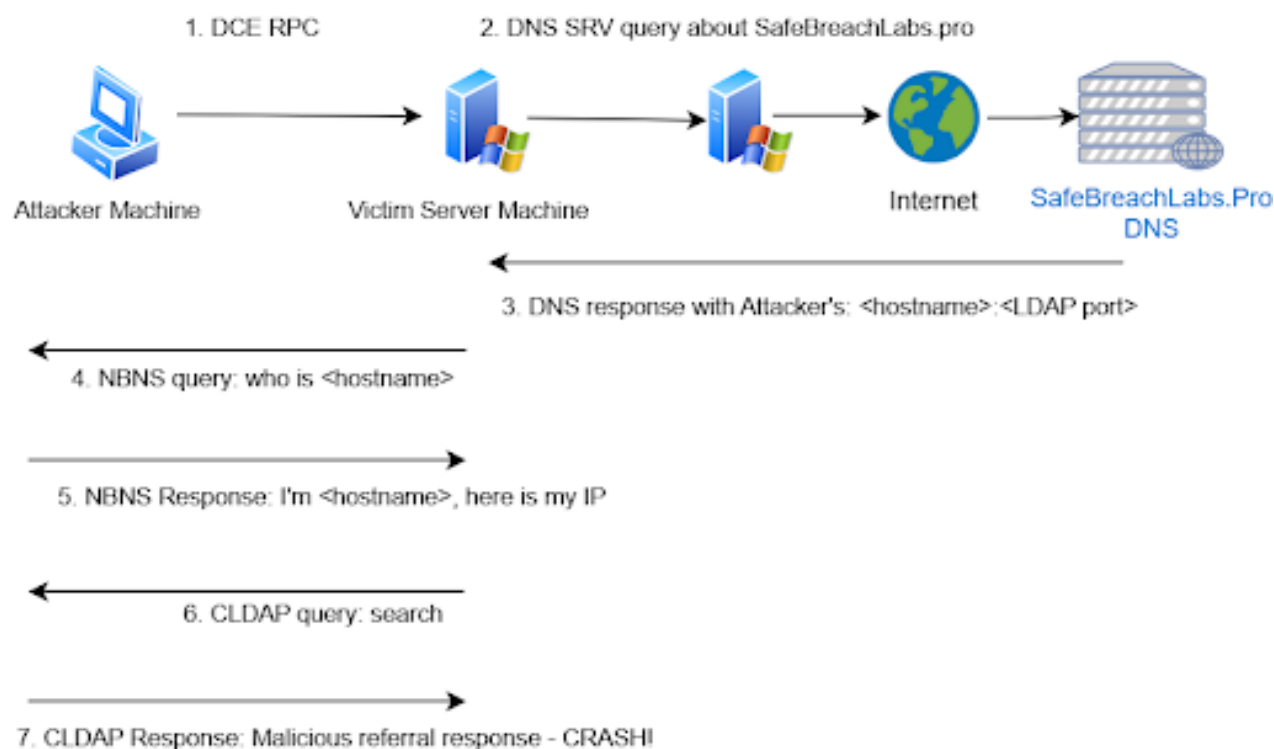
High-Level Summary

SafeBreach Labs developed a proof of concept exploit for CVE-2024-49113 that crashes any unpatched Windows Server (not just DCs) with no pre-requisites except that the DNS server of the victim DC has Internet connectivity.



The attack flow:

1. The attacker sends a DCE/RPC request to the Victim Server Machine
2. The Victim is triggered to send a DNS SRV query about SafeBreachLabs.pro
3. The Attacker's DNS server responds with the Attacker's hostname machine and LDAP port
4. The Victim sends a broadcast NBNS request to find the IP address of the received hostname (of the Attacker's)
5. The Attacker sends an NBNS response with its IP Address
6. The Victim becomes an LDAP client and sends a CLDAP request to the Attacker's machine
7. The Attacker sends a CLDAP referral response packet with a specific value resulting in LSASS to crash and force a reboot of the Victim server



We believe this same attack vector may be leveraged to achieve an RCE; the entire chain noted above, including the first six steps, should be similar, but the last CLDAP packet sent should be modified.

We believe our findings are significant for a number of reasons. First, we have shown the criticality of this vulnerability by proving that it can be used to crash multiple unpatched Windows servers. According to Microsoft's classification, this vulnerability can be further exploited to lead to remote code execution. Second, we did verify that Microsoft's patch fixes the out-of-bounds vulnerability and the exploit is not capable of crashing patched servers. Finally, we provided a public PoC that organizations can use to test and verify that their servers are protected. For more details, please see the GitHub repository noted at the end of this blog.

The vulnerability that the SafeBreach Labs PoC exploits affects technology that is in widespread use across enterprise networks, and this flaw could help attackers propagate more easily and effectively. SafeBreach helps large enterprises identify and address potential exposures, including vulnerabilities like CVE-2024-49113, and SafeBreach customers will soon have access to new capabilities to test their internal networks against this and other vulnerabilities. Keep an eye on www.safebreach.com for news to come.

Technical Deep Dive

Below, we will explain the exact technical details of how the SafeBreach Labs research team identified the exploitation path that triggers the vulnerability and crashes a DC (or any Windows Server), provide a step-by-step exploitation summary, and share a proof-of-concept (PoC) tool that executes these steps.

CVE-2024-49113

CVE-2024-49113 was titled as “Windows Lightweight Directory Access Protocol (LDAP) Denial of Service Vulnerability”. LDAP is the protocol that workstations and servers in Microsoft’s Active Directory use to access and maintain directory services information. The title of the vulnerability means that the vulnerability probably has something to do with LDAP-related code. On MSRC’s page for the CVE, Microsoft provided a few details, but on the RCE vulnerability there was additional interesting data:

“How could an attacker exploit this vulnerability?”

*A remote unauthenticated attacker who successfully exploited this vulnerability would gain the ability to **execute arbitrary code within the context of the LDAP service**. However successful exploitation is dependent upon what component is targeted.*

In the context of exploiting a domain controller for an LDAP server, to be successful an attacker must send specially crafted RPC calls to the target to trigger a lookup of the attacker’s domain to be performed in order to be successful.

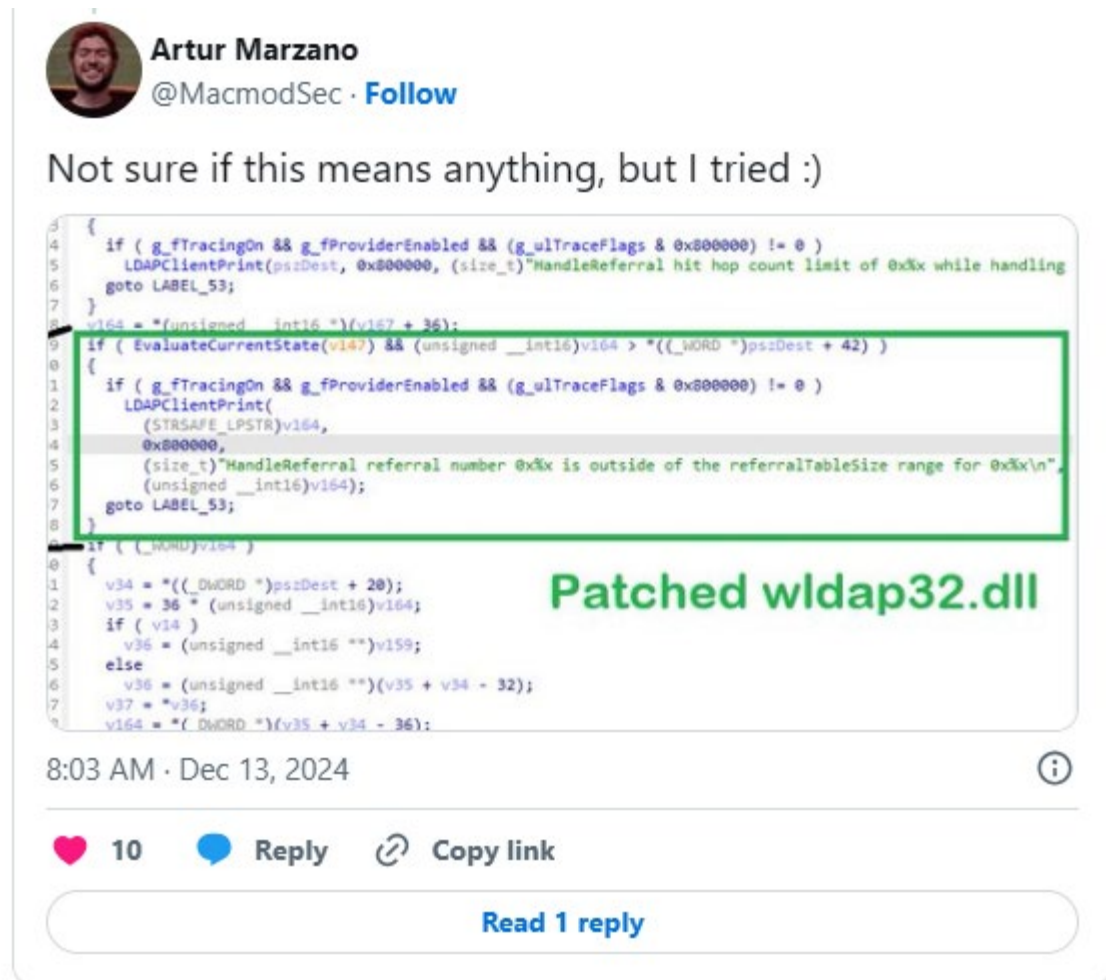
In the context of exploiting an LDAP client application, to be successful an attacker must convince or trick the victim into performing a domain controller lookup for the attacker’s domain or into connecting to a malicious LDAP server. However, unauthenticated RPC calls would not succeed.”

Based on this information—and assuming the accuracy of Microsoft’s documentation—we made the following assumptions:

1. The attacker does not need to authenticate
2. The vulnerability is an integer overflow type and is sourced in an executable or a Dynamic-Linked Library (DLL) that implements an LDAP client logic
3. There are RPC calls that we can leverage in order to affect a DC to query an LDAP server controlled by an attacker

4. In the context of a DC, the vulnerability probably lies in `/sass.exe` or in one of the DLLs that it loads, as `/sass.exe` implements the LDAP service on a DC
5. Thus, the RPC interface with the RPC call that has the vulnerable LDAP client code is located in `Isass.exe` or in one of the DLLs that it loads as well

In addition, we also found an interesting insight by Artur Marzano (@MacmodSec) on X that suggested the potential location where Microsoft's patch for the vulnerability was made, in `wldap32.dll`:



This insight fits perfectly with the documentation in MSRC's website, as `wldap32.dll` implements the logic of an LDAP client.

Triggering a Remote LDAP Request

We started with proving the first step of exploitation against a DC—affecting it to query an LDAP server controlled by us. We needed to find an RPC call sourced in `Isass.exe` itself or in a DLL loaded into `Isass.exe` that imports functions from `wldap32.dll`. Using

RpcView, we listed the available RPC interfaces loaded into lsass.exe:

svchost.exe4088C:\Windows\System32\svchost.exe

lsass.exe924C:\Windows\System32\lsass.exe

fontdrvhost.exe4772C:\Windows\System32\fontdrvhost.exe

777

Interfaces

Pid	Uuid	Ver	Type	Procs	Stub	Callback	Name	Base	Locati
924	0b1c2170-5732-...	0.0	RPC	7	Interpreted	0x00007ff8a03d...		0x00007ff8a03c...	C:\Win
924	11220835-5b26-...	1.0	RPC	3	Interpreted	0x00007ff8a04a...		0x00007ff8a04a...	C:\Win
924	12345678-1234-...	1.0	RPC	59	Interpreted	0x00007ff8a08d...		0x00007ff8a08b...	C:\Win
924	12345778-1234-...	0.0	RPC	134	Interpreted	0x00007ff8a108...		0x00007ff8a105...	C:\Win
924	12345778-1234-...	1.0	RPC	74	Interpreted	0x00007ff8a0eb...		0x00007ff8a0eb...	C:\Win
924	18f70770-8e64-...	0.0	RPC	5	Interpreted	0x00007ff8a397...		0x00007ff8a389...	C:\Win
924	3919286a-b10c-...	0.0	RPC	1	Interpreted			0x00007ff8a105...	C:\Win
924	3dde7c30-165d-...	1.0	RPC	1	Interpreted	0x00007ff8a04c...		0x00007ff8a04a...	C:\Win
924	4f32adc8-6052-...	1.0	RPC	16	Interpreted			0x00007ff8a05f0...	C:\Win
924	51a227ae-825b-...	1.0	RPC	1	Interpreted	0x00007ff8a97a7...		0x00007ff8a97a7...	C:\Win
924	5cbe92cb-f4be-...	1.0	RPC	3	Interpreted	0x00007ff8a04a...		0x00007ff8a04a...	C:\Win
924	7f1317a8-4dea-...	1.0	RPC	2	Interpreted	0x00007ff8a04d...		0x00007ff8a04a...	C:\Win
924	8fb74744-b2ff-4...	1.0	RPC	12	Interpreted	0x00007ff8a97a7...		0x00007ff8a97a7...	C:\Win
924	ace1c026-8b3f-...	1.0	RPC	2	Interpreted	0x00007ff8a109...		0x00007ff8a105...	C:\Win
924	afc07e2e-311c-...	1.0	RPC	3	Interpreted	0x00007ff8a112...		0x00007ff8a105...	C:\Win
924	b25a52bf-e5dd-...	2.0	RPC	32	Interpreted	0x00007ff8a97a7...		0x00007ff8a97a7...	C:\Win
924	c681d488-d850-...	1.0	RPC	21	Interpreted			0x00007ff8a04f0...	C:\Win
924	c9acd8b5-82b7-...	1.0	RPC	15	Interpreted	0x00007ff8a9f3b...		0x00007ff8a9f3b...	C:\Win
924	d25576e4-00d2-...	0.0	RPC	3	Interpreted			0x00007ff8a105...	C:\Win
924	e3514235-4b06-...	4.0	RPC	31	Interpreted	0x00007ff8a9f5d...		0x00007ff8a9f4b...	C:\Win
924	fb8a0729-2d04-...	1.0	RPC	7	Interpreted	0x00007ff7b0b4...		0x00007ff7b0b4...	C:\Win

Procedures

Index	Name	A
-------	------	---

Out of these RPC interfaces, we listed only the ones sourced in DLLs that are dependent on wldap32.dll and use its exported functions. We were looking for RPC interfaces that did not require authentication, as we assumed that the attacker does not need to authenticate. Two interesting interfaces we found that offered several interestingly named RPC calls that seemed related to LDAP queries and could possibly trigger one were located in lsasrv.dll and netlogon.dll:

```
dq offset NetrLogonControl2Ex
dq offset NetrEnumerateTrustedDomains
dq offset DsrGetDcName
dq offset NetrLogonGetCapabilities
dq offset NetrLogonSetServiceBits
dq offset NetrLogonGetTrustRid
dq offset NetrLogonComputeServerDigest
dq offset NetrLogonComputeClientDigest
dq offset NetrServerAuthenticate3
dq offset DsrGetDcNameEx
dq offset DsrGetSiteName
dq offset NetrLogonGetDomainInfo
dq offset NetrServerPasswordSet2
dq offset NetrServerPasswordGet
dq offset NetrLogonSendToSam
dq offset DsrAddressToSiteNamesW
dq offset DsrGetDcNameEx2
dq offset NetrLogonGetTimeServiceParentDomain
dq offset NetrEnumerateTrustedDomainsEx
```

```
dq offset LsarEnumeratePrivileges
dq offset LsarQuerySecurityObject
dq offset LsarSetSecurityObject
dq offset LsarChangePassword
dq offset LsarOpenPolicyRPC
dq offset LsarQueryInformationPolicy
dq offset LsarSetInformationPolicy
dq offset LsarClearAuditLog
dq offset LsarCreateAccount
dq offset LsarEnumerateAccounts
dq offset LsarCreateTrustedDomain
dq offset LsarEnumerateTrustedDomains
dq offset LsarLookupNames
dq offset LsarLookupSids
dq offset LsarCreateSecret
dq offset LsarOpenAccount
dq offset LsarEnumeratePrivilegesAccount
dq offset LsarAddPrivilegesToAccount
```

Using IDA, we searched from the bottom up for RPC calls that actively use one of the functions imported from wldap32.dll. After a long search, we found DsrGetDcNameEx2. According to Microsoft's documentation:

"The DsrGetDcNameEx2 method SHOULD return information about a domain controller (DC) in the specified domain and site. If the AccountName parameter is not NULL, and a DC matching the requested capabilities (as defined in the Flags parameter) responds during this method call, then that DC will have verified that the DC account database contains an account for the AccountName specified.

```
NET_API_STATUS DsrGetDcNameEx2(
```

```
[in, unique, string] LOGONSRV_HANDLE ComputerName,
```

```
[in, unique, string] wchar_t* AccountName,
```

```
[in] ULONG AllowableAccountControlBits,
```

```
[in, unique, string] wchar_t* DomainName,
```

```
[in, unique] GUID* DomainGuid,  
  
[in, unique, string] wchar_t* SiteName,  
  
[in] ULONG Flags,  
  
[out] PDOMAIN_CONTROLLER_INFO* DomainControllerInfo  
  
);  
  
”
```

This function looked pretty promising. It actively retrieves a hostname of a domain controller, in addition to verifying that a specific account exists in it. Both the domain name and the account are specified by the caller. That means that if the function uses LDAP in order to fulfil its purpose, we have what we need.

Moving forward, we needed to understand each one of DsrGetDcNameEx2 ‘s arguments and the values that we would set for them:

- ComputerName: The hostname of the target DC – This would be set to the victim’s hostname (further research revealed that this value does not matter at all for the function)
- AccountName: The account name that will be searched in the queried attacker’s domain —it can be any name—we don’t care if it exists or not
- AllowableAccountControlBits: Controls what will be queried about “AccountName” – can be 0 – we don’t really care about the queried account
- DomainName: The domain that will be queried – we set this to **the domain name of the attacker**
- SiteName: The site in which the DC must be located – can be set to NULL
- Flags – extra configuration for the call – we wanted the default behavior first, so we set it to 0
- DomainControllerInfo – Output parameter, where the returned information will be placed

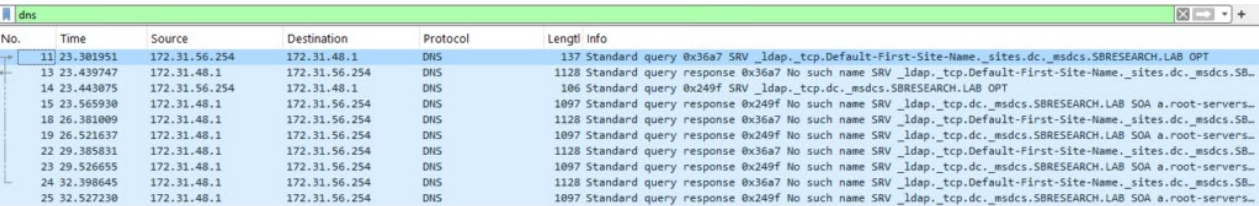
For testing purposes, we installed two new DCs in the same subnet, and created two new root domains in each one of them. One was called SBRESEARCH.LAB and the other TESTDOMAIN.LAB. The goal was to run on the DC at SBRESEARCH.LAB (the

attacker), and get the DC at TESTDOMAIN.LAB (the victim) to query the LDAP server on the DC at SBRESEARCH.LAB.

So running on the attacker DC, we called the DsrGetDcNameEx2 function on the victim DC. The arguments for the call were:

- ComputerName – WIN-ELD41*****
- AccountName – user1
- AllowableAccountControlBits – 0
- DomainName – SBRESEARCH.LAB
- SiteName – NULL
- Flags – 0

Unfortunately, this was not enough. Looking in Wireshark at the packets that were sent and received by the victim, we did not see any LDAP request initiated by the victim. However, Wireshark did show us something else very interesting. We saw that the victim sent a DNS query to its DNS server about a subdomain of SBRESEARCH.LAB. The DNS query was replied with an error code specifying that the DNS server did not find any record about that domain. Then it made perfect sense why the call failed. The only way for the victim DC to get a successful answer about this query is if the attacker DC was its DNS server. But we can't just change a DNS server of a DC; that alone is likely to be considered a vulnerability:



No.	Time	Source	Destination	Protocol	Length	Info
11	23.381951	172.31.56.254	172.31.48.1	DNS	137	Standard query 0x36a7 SRV _ldap._tcp.Default-First-Site-Name._sites.dc._msdcs.SBRESEARCH.LAB OPT
13	23.439747	172.31.48.1	172.31.56.254	DNS	1128	Standard query response 0x36a7 No such name SRV _ldap._tcp.Default-First-Site-Name._sites.dc._msdcs.SB-
14	23.443075	172.31.56.254	172.31.48.1	DNS	106	Standard query 0x249f SRV _ldap._tcp.dc._msdcs.SBRESEARCH.LAB OPT
15	23.565930	172.31.48.1	172.31.56.254	DNS	1097	Standard query response 0x249f No such name SRV _ldap._tcp.dc._msdcs.SBRESEARCH.LAB SOA a.root-servers-
18	26.381009	172.31.48.1	172.31.56.254	DNS	1128	Standard query response 0x36a7 No such name SRV _ldap._tcp.Default-First-Site-Name._sites.dc._msdcs.SB-
19	26.521637	172.31.48.1	172.31.56.254	DNS	1097	Standard query response 0x249f No such name SRV _ldap._tcp.dc._msdcs.SBRESEARCH.LAB SOA a.root-servers-
22	29.385831	172.31.48.1	172.31.56.254	DNS	1128	Standard query response 0x36a7 No such name SRV _ldap._tcp.Default-First-Site-Name._sites.dc._msdcs.SB-
23	29.526655	172.31.48.1	172.31.56.254	DNS	1097	Standard query response 0x249f No such name SRV _ldap._tcp.dc._msdcs.SBRESEARCH.LAB SOA a.root-servers-
24	32.398645	172.31.48.1	172.31.56.254	DNS	1128	Standard query response 0x36a7 No such name SRV _ldap._tcp.Default-First-Site-Name._sites.dc._msdcs.SB-
25	32.527230	172.31.48.1	172.31.56.254	DNS	1097	Standard query response 0x249f No such name SRV _ldap._tcp.dc._msdcs.SBRESEARCH.LAB SOA a.root-servers-

The specific DNS query that was sent was of type SRV. DNS SRV queries specify a domain name, to which another domain name and a port are mapped in the response. The specific full domain name of the two queries sent by the victim DC were:

- _ldap._tcp.Default-First-Site-Name._sites.dc._msdcs.SBRESEARCH.LAB
- _ldap._tcp.dc._msdcs.SBRESEARCH.LAB

Great! It looked like the victim DC really was looking for an LDAP server on our attacker domain. If we could just get this DNS query to be solved successfully, then the LDAP query by the victim would potentially happen. But if we couldn't change the victim's DNS server, what else could we do?

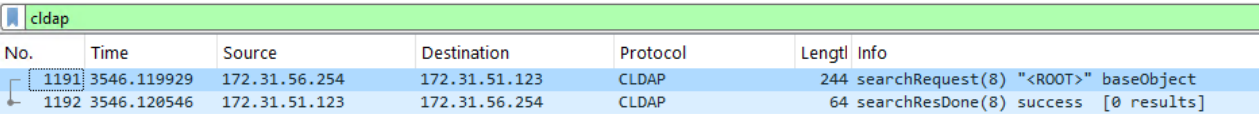
Do we have to control the victim’s DNS server in order to get the query to be solved successfully? The victim’s DNS server does not know SBRESEARCH.LAB, but it does know other domains. Not all the domain names that are known to the victim’s DNS server were manually configured on it. This DNS server knows “google.com” of course. What did Google do in order to be known by this DNS server? They bought a domain on the Internet, so this is exactly what we did as well.

We bought the domain “safebreachlabs.pro” to create two SRV records for:

- `_ldap._tcp.Default-First-Site-Name._sites.dc._msdcs.safebreachlabs.pro`
- `_ldap._tcp.dc._msdcs.safebreachlabs.pro`

These SRV records need to return a domain name (IP is not supported) and a port, which are likely to be contacted by the victim as the LDAP server. At first sight, it looks like this might mean that the victim will have to contact an LDAP server that has a public IP on the Internet, which is a requirement that we prefer not to have, as a firewall might block such communication. But, if we already have access to the DC’s subnet, we can maybe set the domain name that the SRV record returns to be the hostname of the computer that we control in the subnet. So, we mapped both SRV records to the hostname of the attacker DC, and port 389 (its LDAP server).

Following that, we ran another test. Running on the attacker DC, we again called the DsrGetDcNameEx2 function on the victim DC, but this time changed the DomainName parameter to be “safebreachlabs.pro” instead of “SBRESEARCH.LAB”, and it worked. The victim DC issued an LDAP query to our attacker DC.



No.	Time	Source	Destination	Protocol	Length	Info
1191	3546.119929	172.31.56.254	172.31.51.123	CLDAP	244	searchRequest(8) "<ROOT>" baseObject
1192	3546.120546	172.31.51.123	172.31.56.254	CLDAP	64	searchResDone(8) success [0 results]

As you can see in the image above, the query is sent in Connectionless LDAP (CLDAP) and uses UDP instead of TCP. Using Windbg, we were able to verify that even though this request is being made in CLDAP, it is still performed by wldap32.dll.

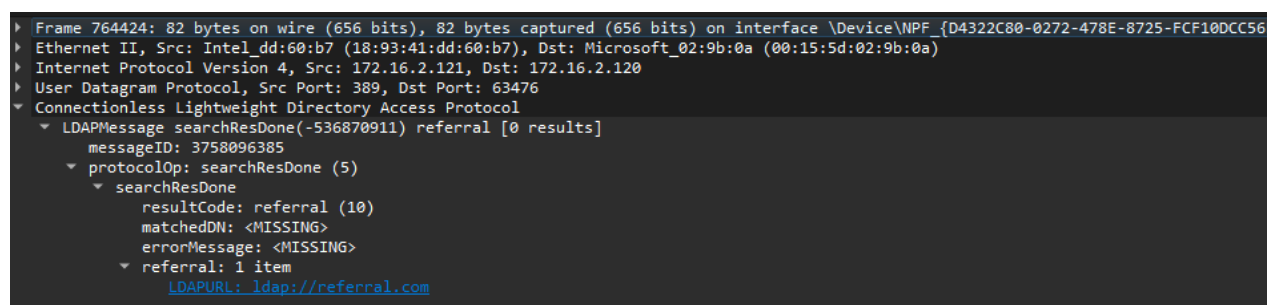
Sending a Malicious LDAP Response

Once we managed to get the victim DC to query our attacker LDAP server, we could move on to understand what needed to be in the response for that query. That is in order to get the victim to execute the assumed vulnerable function found by Artur Marzano –

LdapChaseReferral.

Referrals allow an Active Directory tree to be partitioned between multiple LDAP servers. When an LDAP server can't answer a request, it can reply with referrals to other servers that may provide the answers for the query. Then, the client can "chase" these referrals and query the referred servers instead. It's important to note that a client is not obligated to "chase" these referrals. However, in our case it does chase them.

In order for a server to indicate that it does not have the answer for the query and refer the client to different servers, it needs to reply with the "referral" LDAP result code (equals to 10). The response must also contain valid LDAP URLs (starts with "ldap://" or "ldaps://").



```

Frame 764424: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface \Device\NPF_{D4322C80-0272-478E-8725-FCF10DCC56F}
Ethernet II, Src: Intel_dd:60:b7 (18:93:41:dd:60:b7), Dst: Microsoft_02:9b:0a (00:15:5d:02:9b:0a)
Internet Protocol Version 4, Src: 172.16.2.121, Dst: 172.16.2.120
User Datagram Protocol, Src Port: 389, Dst Port: 63476
Connectionless Lightweight Directory Access Protocol
  LDAPMessage searchResDone(-536870911) referral [0 results]
    messageID: 3758096385
    protocolOp: searchResDone (5)
    searchResDone
      resultCode: referral (10)
      matchedDN: <MISSING>
      errorMessage: <MISSING>
      referral: 1 item
        LDAPURL: ldap://referral.com
  
```

Going back to our exploitation scenario, in order to trigger the LdapChaseReferral function, we created our own custom LDAP UDP server that allowed us to send such a referral response packet.

Looking at the logic of the patch, Microsoft added a condition that verifies that a certain value is not bigger than another value. Based on the logs printed next to this logic, these compared values are named as "lm_referral" and "referral table size". "lm_referral" is taken from an "ldap_message" struct (probably our response message) and "referral table size" is taken from an "ldap_connection" struct. The condition checks whether the "lm_referral" value is inside the range of the "referral table". This range is the "referral table size".

In the vulnerable version without the patch, this "lm_referral" value is indeed used to access a certain offset inside the referral table:

```

if ( message->lm_referral )
{
    ref_table_1 = *((_QWORD *)request + 21);
    v33 = (unsigned __int64)message->lm_referral << 6;
    v34 = (unsigned __int16 **)(v33 + ref_table_1 - 56);
    v168 = *((_QWORD *)v33 + ref_table_1 - 64);
}
  
```

In our tests with Windbg, we saw that the value in “lm_referral” is always equal to 0, while the pointer to the referral table is also equal to 0. However, the condition determining whether the code accesses the “referral table” only verifies whether the “lm_referral” value is not zero. That means that in order to trigger the vulnerability we must control the “lm_referral” variable and make it non-zero. If we succeed, then the code will dereference a pointer that we can control using lm_referral’s value.

Searching for where the “lm_referral” variable inside the “ldap_message” struct is populated, we looked for other occurrences in wldap32.dll where the offset of “lm_referral” inside the “ldap_message” struct is being used (+0x3C). This resulted in two functions: LdapInitialDecodeMessage and LdapChaseReferral:

```
.text:000000000000FC0A ?LdapInitialDecodeMessage... mov [r14+3Ch], cx
.text:00000000000029A69 ?LdapChaseReferral@@YAK... cmp [r15+3Ch], di
.text:00000000000029AAF ?LdapChaseReferral@@YAK... movzx ecx, word ptr [r15+3Ch]
```

Then we identified the code that sets “lm_referral” in LdapInitialDecodeMessage:

```
value_from_response_packet = value_from_response_packet2;
a2->lm_referral = (unsigned int)value_from_response_packet2 >> 25;
a2->lm_msgid = value_from_response_packet & 0x1FFFFFFF;
```

What we saw is that “msgid” and “lm_referral” are taken from the same part of the packet.

In the above screenshot, the “value_from_response_packet” must be a 4-byte DWORD in order to make “lm_referral” a non-zero WORD, due to the shift by 25.

In the default response packet that we sent using our custom UDP LDAP server, we can fully control the value of “value_from_response_packet” (seen in the above screenshot), and it is one byte long. What we learned is that this value is prefixed with its length.

Then we understood what we needed to do in order to set a non-zero value for “lm_referral”:

- Change the byte that represents the length of “value_from_response_packet” (combination of “lm_referral” and “lm_msgid”) to 4 instead of 1.
- Now “value_from_response_packet” is 4 bytes long, and we can set the most significant byte from it, which will affect “lm_referral”. Keep in mind that we can set this byte only to a value that can be divided equally by 2, or

otherwise we affect the value of “lm_msgid”

These two actions will point the flow of the code into the scope of the vulnerable code and create an access violation once the dereferencing happens:

```
if ( a2->lm_referral )
{
    ref_table = *(_QWORD *)&a1->ref_table;
    lm_referral_shifted_left_by_6 = (unsigned __int64)a2->lm_referral << 6;
    v34 = (unsigned __int16 **)(lm_referral_shifted_left_by_6 + ref_table - 56);
    v171 = *(void **)(lm_referral_shifted_left_by_6 + ref_table - 64);
}
```

Since “ref_table” is equal to NULL, and “lm_referral” at this point is a non-zero value, the last line of code in the above image will trigger a dereference for a non-existent address resulting in out-of-bounds read and crashing LSASS and the entire operating system.

```
0:000> g
(194c.1a04): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
WLDAP32!LdapChaseReferral+0x393:
00007ffb`ec039abf 4c8b4c11c0      mov     r9,qword ptr [rcx+rdx-40h] ds:00000000`00001bc0=????????????????
```

Next Steps: On the Way to Achieve RCE

Based on the initial research outlined here, we continued working towards the implementation of a full RCE chain (CVE-2024-49112). We haven’t achieved that yet, but we have made some good progress towards it. In this section, we will describe this progress.

In order not to crash, and continue the exploit, we are planning to find a way to trigger an integer overflow in the code that parses the CLDAP response, as MSRC documented that CVE-2024-49112 is triggered by an integer overflow.

We analyzed the diff between the patched and the unpatched version of the DLL and found the integer overflow fixes. The fixes are achieved by adding calls to the functions ULongMult, ULongAdd, and ULongSub. These functions perform the addition/multiplication/substitution and check whether an overflow happened. In addition, they store the result in an output parameter.

These functions are known to be Microsoft’s usual mitigation for previous integer overflow vulnerabilities, for example MS16-098. See our previous analysis of this patching method [here](#).

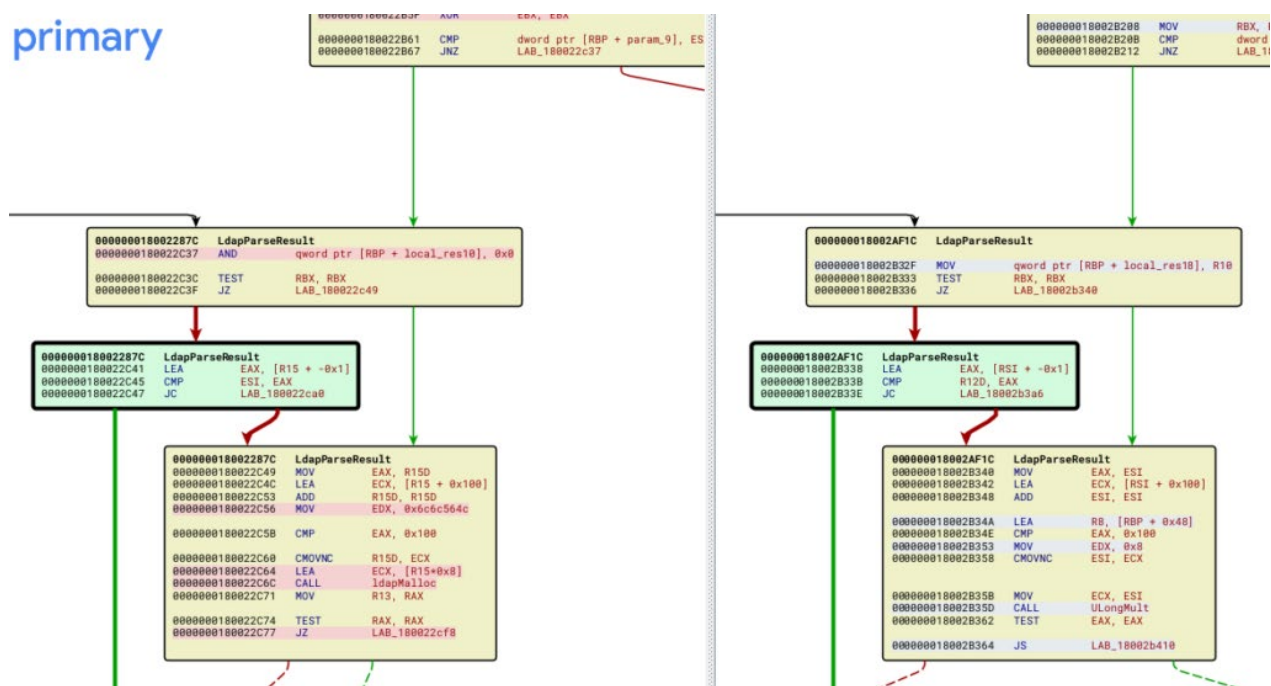
Here are the xrefs to these three new added functions in the patched DLL:

xrefs to ULONGAdd			
Direction	Typ	Address	Text
Up	p	LdapBind+329	call ULONGAdd
Up	p	CLdapBer::HrAddBinaryValue(uchar *,ulong,ulong)+3F	call ULONGAdd
Up	p	CLdapBer::HrAddValue(long,ulong)+6D	call ULONGAdd
Up	p	CLdapBer::HrAddValue(char const *,ulong)+C6	call ULONGAdd
Up	p	CLdapBer::HrAddValue(ushort const *,ulong)+FA	call ULONGAdd
Up	p	ldap_escape_filter_element+58	call ULONGAdd
	p	ldap_escape_filter_element+77	call ULONGAdd

xrefs to ULONGMult			
Direction	Typ	Address	Text
Up	p	LdapBind+34B	call ULONGMult
Do...	p	LdapGetValues+477	call ULONGMult
Do...	p	LdapGetValues+5CA	call ULONGMult
Do...	p	LdapGetValues+67C	call ULONGMult
Do...	p	LdapParseResult+32A	call ULONGMult
Do...	p	LdapParseResult+441	call ULONGMult
Do...	p	SendLdapSearch(ldap_request *,lda...	call ULONGMult
Do...	p	add_string_to_list+9B	call ULONGMult
Do...	p	add_string_to_list+E2	call ULONGMult
Do...	p	ldap_escape_filter_elementW+3D	call ULONGMult
Do...	p	ldap_escape_filter_elementW+DA	call ULONGMult

xrefs to ULONGSub		
Direction	Typ	Address
Do...	p	CLdapBer::HrLoadMoreBer(uchar const *,ulong,ulong *)+8C

We started going through the fixes, and at least for now, concluded that the ones we checked will probably not lead to the RCE. We started with LdapParseResult. On the left, we can see the unpatched version of the function LdapParseResult and on the right we can see the patched version, which has the added call to ULONGMult before calling ldapMalloc.



We verified that our exploit code triggers this function if we send back a regular CLDAP response instead of the one crashing LSASS, but it was not enough. This function will get to this patched code only if the sixth parameter is not zero. This sixth parameter is a unicode string representing referrals. Based on our check, the RPC method that we call leads the parameter to be always zero. There were other xrefs to LdapParseResult, but we didn't succeed to find a flow that calls it and sets the sixth parameter to a non-zero value using the RPC call that we use. It is possible that it can be triggered with a different RPC call than the one we use – DsrGetDcNameEx2. For now, we are going to try to use DsrGetDcNameEx2.

We decided to shift to a different attack flow. We analyzed the other calls to ULONGMult and focused on the function SendLdapSearch.

Just below the call to ULONGMult, we found an error print that verifies an integer overflow mitigation:

```
if ( ULONGMult(v26, 2u, &pulResult) < 0 )
{
    v19 = 90;
    if ( !g_fTracingOn || !g_fProviderEnabled || !(g_ulTraceFlags & 0x40000) )
        goto LABEL_126;
    v27 = "ldap_search : integer overflow for filter with remaining size 0xx.\n";
    goto LABEL_83;
}
v28 = (unsigned __int16 *)ldapMalloc(pulResult, 1920226124i64);
```

This is the unpatched code:

```
if ( LDAPFilter && (v24 = strlenW(LDAPFilter), filterLength = v24 + 1, v24) )
{
    v26 = (unsigned __int16 *)ldapMalloc(2 * filterLength, 1920226124i64);
    v27 = v26;
    if ( !v26 )
    {
        v19 = 90;
        if ( g_fTracingOn && g_fProviderEnabled && g_ulTraceFlags & 0x40000 )
            LDAPClientPrint(0x40000i64, "ldap_search : could not allocate memory 0x%x.\n", filterLength);
        goto LABEL_120;
    }
    memcpy(v26, LDAPFilter, 2i64 * filterLength);
    v19 = LdapEncodeFilter(v15, v27, 0);
}
```

We were able to reach this function and even reached the call to ldapMalloc:

```
0:042> u
WLDAP32!SendLdapSearch+0x494:
00007ff9`e3177528 e8c73f0000 call WLDAP32!ldapMalloc (00007ff9`e317b4f4)
```

In the relevant code, we understood that the variable that is passed to strlenW is an LDAP filter sent as part of a search that is resulted from our RPC call. If filterLength is multiplied by 2, and filterLength is long enough, an overflow will happen and the allocation will be very small compared to the filter length. An important thing to note is that we control some elements in this filter such as the username and DnsDomain:

```
0:042> du poi(rsp + 80)
00000185`e5103bc8 "((&(DnsDomain=safebreachlabs\2Epr"
00000185`e5103c08 "o)(Host=WIN\2DVM51N9F5JCR)(User="
00000185`e5103c48 "Administrator)(AAC=\10\00\00\00)"
00000185`e5103c88 "(DomainGuid=\00\00\00\00\00\00\0"
00000185`e5103cc8 "0\00\00\00\00\00\00\00\00)(Nt"
00000185`e5103d08 "Ver=\16\00\00\01)(DnsHostName=WI"
```

We tried sending a very long username, but it seems like the required length for the username value to trigger this overflow is too big to be passed to the DsrGetDcNameEx2 RPC call.

We are still in the progress of investigating the other references to the three integer overflow mitigation functions, and we will provide additional updates here once we have new findings.

Exploit PoC

We have created a [research repository](#) that includes a PoC of the LDAP Nightmare exploit that organizations can use to test and verify that their servers are protected

against this vulnerability.

Affected Windows Servers

While our research focused on the testing of a Windows Server 2022 (DC) and Windows server 2019 (non DC), we believe this exploit path and PoC are applicable for any Windows Server version until the patch point.

Mitigation

To mitigate the risk of this vulnerability, organizations should implement the patch released by Microsoft detailed [here](#). As noted above, SafeBreach Labs verified that the patch sufficiently prevents the exploitation and crashing of the tested servers. We believe patching this vulnerability is time critical, but also understand that patching a DC and Windows Servers must be done carefully and with the proper caution.

As such, we suggest organizations implement detections to monitor suspicious CLDAP referral responses (with the specific malicious value set), suspicious DsrGetDcNameEx2 calls, and suspicious DNS SRV queries until a patch can be applied.

Conclusion

This research set out to explore whether the LDAP CVE-2024-49113 vulnerability could be exploited. Our research proved that not only can it be exploited against Domain Controllers, it also affects any unpatched Windows Server.

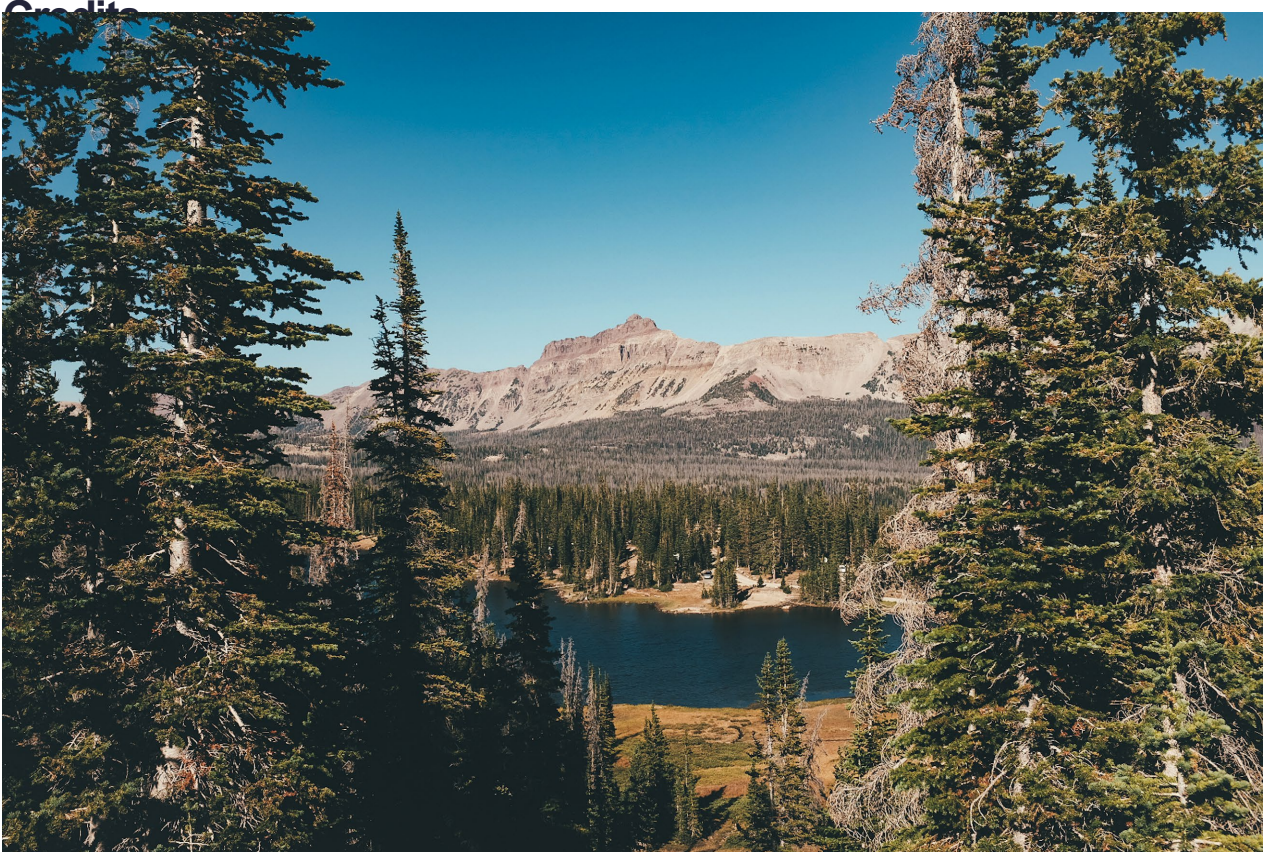
In addition, we provided an exploit PoC for testing purposes, noted in the section above. We also believe that this will make exploitation of CVE-2024-49112 more likely in the near future, so we recommend patching both vulnerabilities.

For more in-depth information about this research, please:

- Contact your customer success representative if you are a

current SafeBreach customer

- [Schedule a one-on-one](#) discussion with a SafeBreach expert
- Contact [Kesselring PR](#) for media inquiries



You Might Also Be Interested In

Leverage the Attack to Improve Your Defenses

Why SafeBreach

Breach and Attack Simulation Platform

No-Code Red Team Platform

Service & Support

SafeBreach-as-a-Service

RansomwareRx

Threat Assessment

Security Control Validation

Cloud Security Assessment

Healthcare

Finance

Life Sciences

IT/OT Environments

Labs

Research Hub

CVE Discoveries

- Blog
- Reports & White Papers
- Solution Briefs
- Videos & Podcasts
- Case Studies
- On-Demand Webinars
- Events & Webinars
- Breach and Attack Simulation
- MITRE ATT&CK®: The Complete Guide

- Our Partners
- Partner Portal

- About
- Leadership
- Newsroom
- Careers
- Contact Us
- Security Portal

- Support Portal
- SafeBreach Labs
- Request a Demo

