CVE-2021-44142 Storytale

Mikhail Gudyrin, Prague, 2022

Overview

Primary goal of current project was to extract intangible knowledge about Out-of-Bound vulnerability in Samba disclosed by Nguyễn Hoàng Thạch and Billy Jheng Bing-Jhong in early 2022 and leverage obtained information to describe a business-friendly mitigation approach. In essence, this **was not** a vulnerability research project, published blogs and utilities were used as a basis.

Playground

Project environment required vulnerable version of Samba configured to provide a share with *vfs_fruit* module enabled. We picked a 4.5.16 version, which according to official advisory, is vulnerable, and prepared an isolated environment for the simulation.

We used Docker CE virtualization engine and an official minimal Ubuntu:20.04 image as a base:

```
┌──(kali㉿kali)-[~/cve-2021-44142]
└─$ docker run -it -p 445:445 --cap-add=SYS_PTRACE --privileged ubuntu:20.04
Unable to find image 'ubuntu:20.04' locally
20.04: Pulling from library/ubuntu
8e5c1b329fe3: Pull complete
Digest: sha256:4e9ed8dc49c4c21888f4053e59d7ef0959f57e77d0fbe47ba0063fddd6b70f2c
Status: Downloaded newer image for ubuntu:20.04
root@004cf69bcb5f:/#
```

According to official Samba build guideline, we installed additional packages required for proper compilation and further investigation:

```
root@004cf69bcb5f:/# apt-get -qq update && apt-get -y -qq install acl apt-utils attr autoconf bin
d9utils binutils bison build-essential ccache chrpath curl debhelper dnsutils docbook-xml docbook
-xsl flex gcc gdb git glusterfs-common gzip heimdal-multidev hostname htop krb5-config krb5-kdc k
rb5-user language-pack-en lcov libacl1-dev libarchive-dev libattr1-dev libavahi-common-dev libblk
id-dev libbsd-dev libcap-dev libcephfs-dev libcups2-dev libdbus-1-dev libglib2.0-dev libgnutls28-
dev libgpgme11-dev libicu-dev libjansson-dev libjs-jquery libjson-perl libkrb5-dev libldap2-dev l
iblmdb-dev libncurses5-dev libpam0g-dev libparse-yapp-perl libpcap-dev libpopt-dev libreadline-de
v libsystemd-dev libtasn1-bin libtasn1-dev libtracker-sparql-2.0-dev libunwind-dev lmdb-utils loc
ales lsb-release make mawk mingw-w64 patch perl perl-modules pkg-config procps psmisc python3 pyt
hon3-cryptography python3-dbg python3-dev python3-dnspython python3-gpg python3-iso8601 python3-m
arkdown python3-matplotlib python3-pexpect python3-pyasn1 python3-setproctitle rng-tools rsync se
d sudo tar tree uuid-dev wget xfslibs-dev xsltproc zlib1g-dev python-dev net-tools tcpdump nano
root@004cf69bcb5f:/#
```

Next, we downloaded Samba 4.5.16 source code, extracted gzipped tarball, compiled and installed Samba application:

```
root@004cf69bcb5f:~# cd /tmp && wget https://download.samba.org/pub/samba/stable/samba-4.5.16.tar
.gz
--2022-04-22 05:24:01--  https://download.samba.org/pub/samba/stable/samba-4.5.16.tar.gz
Resolving download.samba.org (download.samba.org)... 144.76.82.148, 2a01:4f8:192:486::2:3
Connecting to download.samba.org (download.samba.org)|144.76.82.148|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 21024396 (20M) [application/gzip]
Saving to: 'samba-4.5.16.tar.gz'

samba-4.5.16.tar.gz      100%[===================>]  20.05M  1.53MB/s    in 14s

2022-04-22 05:24:16 (1.45 MB/s) - 'samba-4.5.16.tar.gz' saved [21024396/21024396]

root@004cf69bcb5f:/tmp# tar xf samba-4.5.16.tar.gz
root@004cf69bcb5f:/tmp# cd /tmp/samba-4.5.16
root@004cf69bcb5f:/tmp/samba-4.5.16# ./configure --prefix=/usr --enable-fhs --sysconfdir=/etc --l
ocalstatedir=/var --with-privatedir=/var/lib/samba/private --with-smbpasswd-file=/etc/samba/smbpa
sswd --with-piddir=/var/run/samba --with-pammodulesdir=/lib/x86_64-linux-gnu/security --libdir=/u
sr/lib/x86_64-linux-gnu --with-modulesdir=/usr/lib/x86_64-linux-gnu/samba --datadir=/usr/share --
with-lockdir=/var/run/samba --with-statedir=/var/lib/samba --with-cachedir=/var/cache/samba --wit
h-socketpath=/var/run/ctdb/ctdbd.socket --with-logdir=/var/log/ctdb --without-ad-dc --enable-debu
g && make -j 4 && make install -j 4
```

Setup a local directory for Samba share to use:

```
root@004cf69bcb5f:/tmp/samba-4.5.16# mkdir /TimeMachineBackup
root@004cf69bcb5f:/tmp/samba-4.5.16# chown nobody:nogroup /TimeMachineBackup
root@004cf69bcb5f:/tmp/samba-4.5.16# chmod 2770 /TimeMachineBackup
root@004cf69bcb5f:/tmp/samba-4.5.16#
```

And finally, configured Samba to share this directory with writable access and *vfs_fruit* module enabled:

```
root@004cf69bcb5f:~# testparm -s
Load smb config files from /etc/samba/smb.conf
Processing section "[TimeMachineBackup]"
Loaded services file OK.
WARNING: You have some share names that are longer than 12 characters.
These may not be accessible to some older clients.
(Eg. Windows9x, WindowsMe, and smbclient prior to Samba 3.0.)
Server role: ROLE_STANDALONE

# Global parameters
[global]
        map to guest = Bad User
        idmap config * : backend = tdb
        map acl inherit = Yes
        vfs objects = acl_xattr


[TimeMachineBackup]
        path = /TimeMachineBackup
        ea support = Yes
        guest ok = Yes
        read only = No
        vfs objects = catia fruit streams_xattr
        fruit:time machine max size = 0M
        fruit:time machine = yes
root@004cf69bcb5f:~#
```

For the sake of project environment reproducibility, we also created a *Dockerfile*:

```
FROM ubuntu:20.04
ENV DEBIAN_FRONTEND=noninteractive
RUN apt-get update && apt-get -y install acl apt-utils attr autoconf bind9utils binutils bison build-essential ccache chrpath curl debhelper dnsutils docbook-xml docbook-xsl flex gcc gdb git glusterfs-common gzip heimdal-multidev hostname htop krb5-config krb5-kdc krb5-user language-pack-en lcov libacl1-dev libarchive-dev libattr1-dev libavahi-common-dev libblkid-dev libbsd-dev libcap-dev libcephfs-dev libcups2-dev libdbus-1-dev libglib2.0-dev libgnutls28-dev libgpgme11-dev libicu-dev libjansson-dev libjs-jquery libjson-perl libkrb5-dev libldap2-dev liblmdb-dev libncurses5-dev libpam0g-dev libparse-yapp-perl libpcap-dev libpopt-dev libreadline-dev libsystemd-dev libtasn1-bin libtasn1-dev libtracker-sparql-2.0-dev libunwind-dev lmdb-utils locales lsb-release make mawk mingw-w64 patch perl perl-modules pkg-config procps psmisc python3 python3-cryptography python3-dbg python3-dev python3-dnspython python3-gpg python3-iso8601 python3-markdown python3-matplotlib python3-pexpect python3-pyasn1 python3-setproctitle rng-tools rsync sed sudo tar tree uuid-dev wget xfslibs-dev xsltproc zlib1g-dev python-dev net-tools tcpdump nano
WORKDIR /tmp
RUN wget https://download.samba.org/pub/samba/stable/samba-4.5.16.tar.gz
RUN tar xf samba-4.5.16.tar.gz
WORKDIR /tmp/samba-4.5.16
RUN ./configure --prefix=/usr --enable-fhs --sysconfdir=/etc --localstatedir=/var --with-privatedir=/var/lib/samba/private --with-smbpasswd-file=/etc/samba/smbpasswd --with-piddir=/var/run/samba --with-pammodulesdir=/lib/x86_64-linux-gnu/security --libdir=/usr/lib/x86_64-linux-gnu --with-modulesdir=/usr/lib/x86_64-linux-gnu/samba --datadir=/usr/share --with-lockdir=/var/run/samba --with-statedir=/var/lib/samba --with-cachedir=/var/cache/samba --with-socketpath=/var/run/ctdb/ctdbd.socket --with-logdir=/var/log/ctdb --without-ad-dc --enable-debug && make -j 4 && make install -j 4
RUN mkdir /TimeMachineBackup
RUN chown nobody:nogroup /TimeMachineBackup
RUN chmod 2770 /TimeMachineBackup
RUN echo "[global]\nmap to guest = Bad User\nvfs objects = acl_xattr\nmap acl inherit = yes\n[TimeMachineBackup]\npath = /TimeMachineBackup\nguest ok = yes\nfruit:time machine = yes\nfruit:time machine max size = 0M\nvfs objects = catia fruit streams_xattr\nguest ok = yes\nwritable = yes\nea support = yes" > /etc/samba/smb.conf
```

Unambiguity. Talks

To ensure project succeeded, we couldn't rely on public claims like "Samba 4.5.16 is vulnerable to CVE-2021-44142". On the one hand, we required own technical insight to the space of possible attack implementations, on the other hand, this project was not about vulnerability research itself, so some breadth/depth balance of approach was kept in mind.

According to researcher's blogpost, examined Out-of-Bound vulnerability allows code execution. This is a *runtime memory corruption* vulnerability. In general, successful attack leveraging memory corruption uses malformed input to overwrite some memory locations aiming to transfer execution flow to injected payload. But if input is malformed enough to trigger the vulnerability, but instead of injecting "sophisticated bytes to proper addresses" just confirms memory corruption fact, we are still good. For us it means, that attack shouldn't necessarily end with reverse shell, required results can be extracted anyway.

Note: this chapter is called *Unambiguity*, and presence of actual unambiguity in such stuff like vulnerability exploitation can reliably be confirmed only by, vulnerability exploitation, not just triggering one. However. 1st, we agreed to not satisfy with researchers claims, but not to distrust them – CVE-2021-44142 was published as a code execution vulnerability and we trust them. 2nd, we kept in mind a breadth/depth balance of approach due to project's resources limitation and our goal was to obtain practical understanding of the attack to describe a business-friendly threat eradication procedure.

Unambiguity. CVE-2021-44142

SMB protocol can interact with Netatalk, an open-source implementation of the Apple Filing Protocol (AFP). It allows creation of Samba shares for Apple clients and is a part of Apple Sharing Services. For its operation, Netatalk uses some metadata, there is no reason to dive into meaning of their values from the protocol perspective, it is just worth to know that they are stored in a *user.org.netatalk.Metadata* extended file attribute (xattr).

The problem arises since *user.org.netatalk.Metadata* xattr is not in Samba private attribute name list. Anyone with writable permissions can use SMB2_SET_INFO request to write arbitrary Netatalk metadata. Later, when SMB will process such a file it will initialize an *AppleDouble* structure in memory and fill with *user.org.netatalk.Metadata* xattr value. All this functionality is handled by the *vfs_fruit* module.

According to researchers blogpost, attacker can leverage this situation to either read or write 31 bytes from heap or to cause a buffer overflow. *AppleDouble* structure contains a pointer to ADEID_FINDERI entry. Let's call this pointer *p*. There are next scenarios how an attack can progress:

1. ### Out-of-Bounds Read

   Attacker can overwrite *p* with address of the last byte of the data buffer, located in *AppleDouble* structure. Then, a call for a *fruit_pread_meta_adouble()* function will cause a read of 31 bytes from next to data buffer location on the heap. In offensive code, this call can be initiated by opening a malicious file with *:AFP_AfpInfo* stream for reading.

   Trigger steps:

   - ✓ Connect to vulnerable SMB share;
   - ✓ Create a new file or use existent one; write payload to *user.org.netatalk.Metadata* xattr;
   - ✓ Open malicious file with *:AFP_AfpInfo* stream for **reading**.

2. ### Out-of-Bounds Write

   Very similar to *read* scenario. Attacker should overwrite *p* with data buffer last byte address. Then a call for a *fruit_pwrite_meta_netatalk()* function will write 31 bytes next to data buffer location on the heap. In offensive code, this can be done by opening a malicious file with *:AFP_AfpInfo* stream for writing.

   Trigger steps:

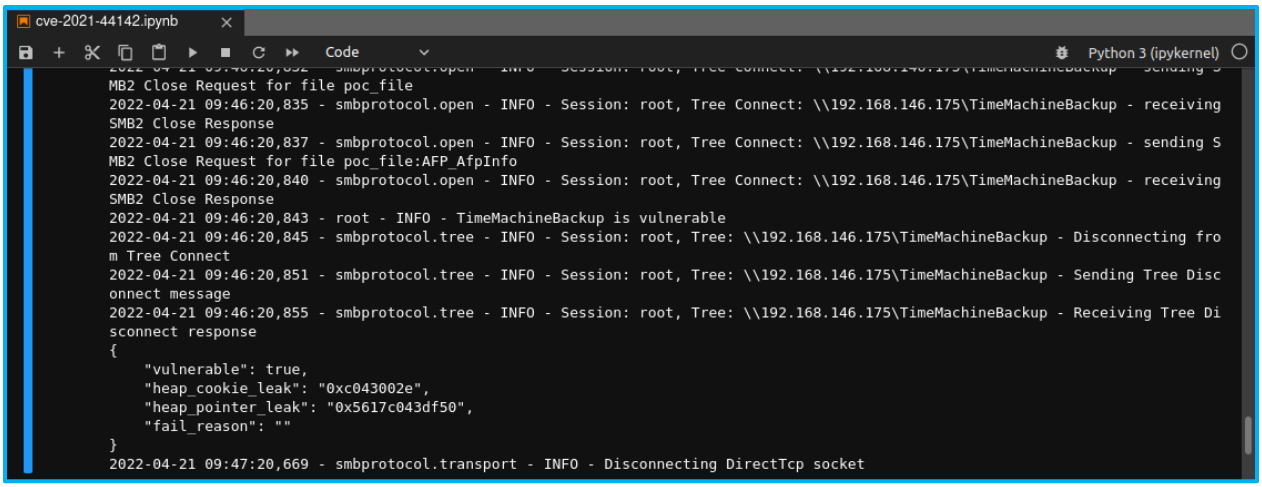
   - ✓ Connect to vulnerable SMB share;
   - ✓ Create a new file or use existent one; write payload to *user.org.netatalk.Metadata* xattr;
   - ✓ Open malicious file with *:AFP_AfpInfo* stream for **writing.**

3. ### Buffer Overflow

   To cause an overflow attacker should follow *write* scenario with next differences: instead of pointer to ADEID_FINDERI entry a pointer to ADEID_FILEDATESI should be overwritten, there will be 3 bytes overwrite and trigger function will be *ad_setdate()*.
   This scenario requires additional research and is out of scope of current project.

Unambiguity. Attack

To demonstrate and consequently confirm CVE-2021-44142 we used public tool, developed by *horizon3ai*, available here.

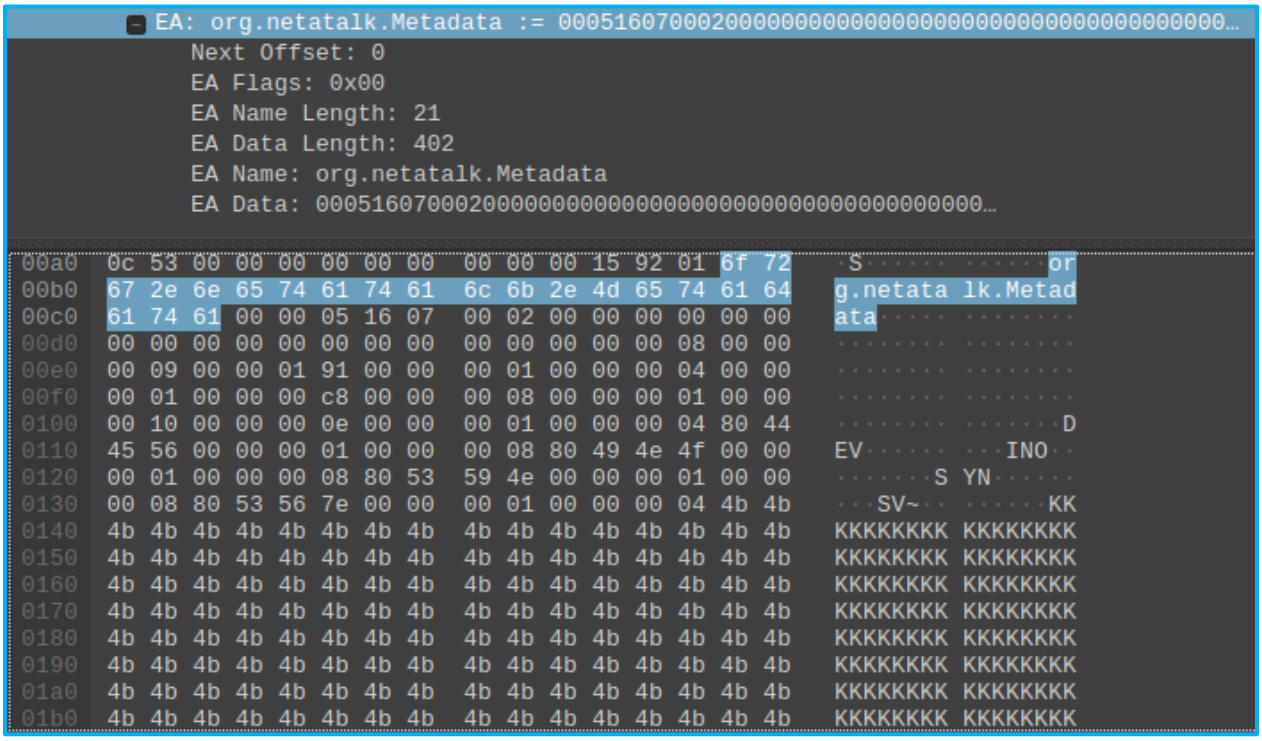


This script implements Out-of-Bound Read scenario and demonstrates dump of a *talloc()* cookie and linked list pointer.

To understand what was going on the network level during the attack, we configured real time packet capture inside our lab:

```
┌──(kali㉿kali)-[~/cve-2021-44142]
└─$ sudo tcpdump -p -s 0 -n -w attack-traffic port 445
tcpdump: listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
```

Malicious Netatalk metadata flowing the network could be easily identified:



Detection

This is without a doubt the most valuable part of the project. CVE-2021-44142 CVSS 3 score is High (8.8), what emphasize actual threat severity. So, businesses require specific knowledge to develop threat mitigation and/or remediation procedures. In this chapter we will focus on real-time detection of exploitation attempt.

Since attacker needs a network access to the target, packet inspection will be an obvious starting point. We will drop discussion of local exploitation due to the nature of such scenario – it is probably can occur within a targeted attack, which impact and progress should be investigated individually.

Naïve approach

The most simple strategy is to catch Netatalk metadata write requests during SMB sessions, what we already demonstrated. This strategy is based on the assumption, that during legitimate communication there is no need to modify *user.org.netatalk.Metadata* xattr and if such modification occurs, there is an exploitation attempt. To increase reliability, we also suggest a conjunction with two additional checks: 1) a vulnerable version of Samba is in use, 2) Samba share is configured to use *vfs_fruit* module.

This approach is very easy to implement (e.g., via YARA rules) and requires low computational resources, so it can be used in a high-load environments without overwhelm risk. But as always, disadvantages are a natural consequences of advantages. Simplicity of this approach doesn't allow to distinguish malicious manipulation with Netatalk metadata from a legitimate one, and hence, applies a usage limitation on a Samba share.

Magical approach

More complex but more promising strategy may use a sorcerence from magic school known as machine learning. Of course, designing a solution to such problem via machine learning deserves separate research and here we will describe general mindfalls.

Since attack always incorporates Netatalk metadata manipulation, strategy can be formulated as a binary classification task. Classifier may use *user.org.netatalk.Metadata* values from SMB network flow as input and produce a True/False answer to a question "*is this value malicious?*" as output. We recommend to focus only on classical machine learning models, because 1) deep learning predictions are far more computational heavy, so using of deep learning models in massive environments can be infeasible, 2) before magic could happen, deep learning models require dataset of immense size to learn on, what is almost impossible task in context of Netatalk metadata.

Also, probably a custom cost-function (it returns "*how much is it malicious*?" for an input sample) should be designed, because Netatalk metadata is of a mixed type – this is just a string which can be interpreted in every way (qualitative or quantitative). And to make it even more realistic let's specify – actually, even every **substring** of Netatalk metadata can be interpreted in every and different way, especially in case of a malicious one.

Final thoughts

In a software which installations is counted by millions a new vulnerability with remote code execution impact was disclosed. Of course, there is already a patch. Of course, mitigation is pretty straightforward.

Of course, organizations with security awareness already fixed it. But just to be clear – this is not enough to make the threat eradicated, this is only a preliminary mitigation. Let me remind you, that e.g. EternalBlue, a publicly available weaponized exploit for discovered in 2017 CVE-2017-0144, is still completely relevant to have in arsenal and this is not without a reason.

We suspect CVE-2021-44142 will remain with us for years due to IoT trends. There is a large and still increasing number of small devices which are not maintained by security aware administrators, and which will almost never be updated. All this effectively renders CVE-2021-44142 as a “forever-day” vulnerability.