Technical Document for CVE-2018-0202

Clamscan in ClamAV before 0.99.4

601.443/643 Security & Privacy, Fall 2018

Jay Chow, Harry Luo, Benfang Wang

Introduction:

The National Vulnerability Database(NVD) is a listing of computer vulnerabilities maintained by the US government. Each listing is known as a CVE and contains a description of the vulnerability, its relative impact and the solutions to patch the vulnerability. It is important to stay informed about the NVD in order to identify and respond to new attacks. Through this project, we will learn about and interface with how real-world attacks are mitigated and patched. **This is the GitHub link of our public repository containing the code of our software project is https://github.com/jaychowjingjie/CVE-2018-0202.git.**

Tasks:

We will replicate this exploit and associated patch for this CVE.

1) First, we have to browse the CVE to find information about the exploit and why it exists.
2) Second, we have to do additional searches of GitHub to find the version of the source necessary to build the vulnerable program. We will attempt to exploit it.
3) Then, we have to patch the program, re-run the exploit and figure why the patch works.

We know that there are vulnerability databases with sample code. There are detailed CVE reports by prominent researches. It is acceptable to build off others' exploits and patches but we will give credit in the form of citations. We will have a bibliography of all the code and resources that we used for this project as part of the technical document.

Deliverables

Code and Technical Document (50%)

The code and technical document are combined as the technical document brings meaning to the code. As such, they must be accessed together. The code portion will be a link to a Github public repository that contains all the source files associated with the project. This includes the vulnerable program, exploit code, patches for the program, and a README type of file with instructions on how to run the exploit.

This CVE will be exploited and patched within the SEED Lab VM. We will ensure that we include precise build instructions. The technical document will document every step of the process, including initial setup tasks. It should be at least as long as the lab reports and should incorporate annotated code, screenshots. We will discuss they why along with how, and have considerable depth of discussion.

We will look at what others have written about the attack, security advisories, research papers and news items that may be critical in finishing the exploit.

At a high level, it should have the outline:

1) Abstract
2) Introduction
3) Vulnerability
4) Exploitation
5) Solution
6) Conclusion
7) References

Presentation(50%)

The second half entails distilling our code and technical document into an effective presentation. My team will talk about this project with slides detailing our steps. We will include demos of our exploit and create

graphics showing our architecture. The presentation will be between 10-12 minutes long to describe our team's work in enough depth to be meaningful.

Submission Information

All project materials such as code link, technical document and presentation slides must be submitted to BlackBoard by December 3, 2018 at 10pm EST. The presentation days are on December 4 2018 and December 6 2018. We make sure the link to our git repo is public and that each of member is committing the code.

Note: We used SEEDUbuntu VM on Virtual Box for this group project:

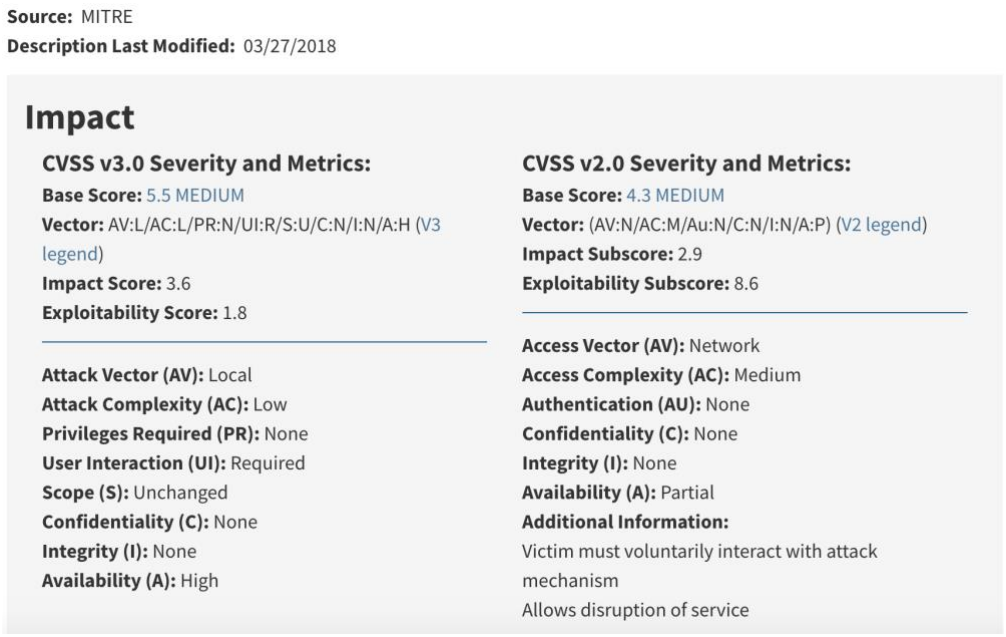
```
[11/26/18]seed@VM:~/.../clamscan$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 16.04.2 LTS
Release:        16.04
Codename:       xenial
```

Abstract

Clamscan in ClamAV before 0.99.4 contains a vulnerability that could **allow an unauthenticated, remote attacker to cause a denial of service (DoS) condition on an affected device**. This is caused by improper input validation checking mechanism when handling Portable Document Format(.pdf) files sent to an affected device. As a result, an unauthenticated, unauthenticated, **remote attacker could exploit**

this vulnerability by sending a crafted .pdf file to an affected device. This action could cause an out-of-bounds (segfault) read when ClamAV scans the malicious file, allowing the attacker to cause a DoS condition. This concerns **pdf_parse_array** and **pdf_parse_string** in **libclamav/pdfng.c**. Cisco Bug IDs: CSCvh91380, CSCvh91400. This vulnerability type is a CWE-125 Out-of-bounds Read vulnerability, allowing attackers to read sensitive information from other memory locations or cause a crash. A crash occurs when the code reads data and expects a sentinel to stop the read such as a null byte in a string. The software may modify an index or perform pointer arithmetic that references a memory location outside the allocated buffer, and a subsequent read will produce undefined or unexpected results.

Note that reading out of an allocated array will cause a warning from the GCC compiler but the program will still be able to be compiled and executed.



Source: MITRE
Description Last Modified: 03/27/2018

Impact

CVSS v3.0 Severity and Metrics:
Base Score: 5.5 MEDIUM
Vector: AV:L/AC:L/PR:N/UI:R/S:U/C:N/I:N/A:H (V3 legend)
Impact Score: 3.6
Exploitability Score: 1.8

Attack Vector (AV): Local
Attack Complexity (AC): Low
Privileges Required (PR): None
User Interaction (UI): Required
Scope (S): Unchanged
Confidentiality (C): None
Integrity (I): None
Availability (A): High

CVSS v2.0 Severity and Metrics:
Base Score: 4.3 MEDIUM
Vector: (AV:N/AC:M/Au:N/C:N/I:N/A:P) (V2 legend)
Impact Subscore: 2.9
Exploitability Subscore: 8.6

Access Vector (AV): Network
Access Complexity (AC): Medium
Authentication (AU): None
Confidentiality (C): None
Integrity (I): None
Availability (A): Partial
Additional Information:
Victim must voluntarily interact with attack mechanism
Allows disruption of service

Introduction

According to ClamAV's official website, ClamAV is an open source anti-virus engine used in email scanning, web scanning, and end-point security. It provides utilities including a multi-threaded daemon, a command line scanner and an advanced tool for automatic database updates. The core is an anti-virus engine in a form of a shared library.

Features:

1) Licensed under GNU General Public License, Version 2
2) POSIX compliant, portable
3) Fast scanning
4) Supports on-access scanning(Linux Only)
5) Detects over 1 million viruses, worms, and trojans, including Microsoft Office macro viruses, mobile malware
6) Built-in bytecode interpreter allows ClamAV signature writers to create and distribute complex detection routines and remotely enhance the scanner's functionality
7) Command-line scanner
8) Milter interface for sendmail
9) Advanced database updater with support for scripted updates and digital signatures
10) Virus databases updated multiple times a day
11) Built-in support for all standard mail file formats
12) Built-in support for archive formats including Zip, RAR, Dmg, Tar, Gzip, Bzip2, OLE2, Cabinet, CHM, BinHex and SIS
13) Built-in support for Elf executables(32 and 64 bit), mach-O files(32 and 64 bit) and Portable Executable files packed with UPX, FSG, Petite, NsPack, wwpack32, MEW, Upack and obfuscated with SUE and Y0da Cryptor
14) Built-in support for popular document formats including MS Office, MacOffice files, HTML, Flash, RTF and PDF.

Base Package

Supported platforms

ClamAV is tested on:

1) GNU/Linux
2) Solaris
3) FreeBSD
4) macOS
5) Windows

Binary packages

The up-to-date list of binary packages is at www.clamav.net/download.html#otherversions

Installation

Requirements

The following components are required to compile ClamAV under UNIX:

1) zlib and zlib-devel packages
2) openssl version 0.9.8 or higher and libssl-devel packages
3) gcc compiler suite(tested with 2.9x, 3.x and 4.x series). The default optimization level is -O2 for gcc. There may exist mis-optimization if we compile with higher optimization levels.
4) GNU make(gmake)

The following packages are highly recommended:

1) bzip2 and bzip2-devel library

2) libxml2 and libxml2-dev library
3) check unit testing framework

These packages are optional but required for bytecode JIT support(if not available, ClamAV will fall back to an interpreter):

1) gcc C and C++ compilers(minimum 4.1.3, recommended 4.3.4 or newer). The packages for these compilers are called: gcc, g++ or gcc-c++.
2) OSX Xcode versions prior to 5.0 use a g++ compiler frontend(llvm-gcc) that is not compatible with ClamAV JIT. It is recommended to either compile ClamAV JIT with clang++ or to compile ClamAV without JIT
3) Supported CPU for JIT is X86, X86-64, PowerPC and PowerPC64

The following packages are optional but needed for JIT unit tests:

1) GNU Make(version 3.79, recommended 3.81)
2) Python(version 2.5.4 or newer) for running JIT unit tests

The follow packages are optional, but required for clamsubmit:

1) libcurl-devel library
2) Libjson-c-dev library

Components of ClamAV

Clamd

Clamd is a multi-threaded daemon that uses libclamav to scan files for viruses. It may work in 1 or both modes listening on:

1) Unix(local) socket
2) TCP socket

Clamdscan

clamdscan is a clam daemon client. We could use it as a a clamscan replacement.

On-access Scanning

There is a special thread in clamd that performs on-access scanning under Linux and shares internal virus database with the daemon. The thread will only notify when potential threats are discovered.

Clamdtop

clamdtop is a tool to monitor one or multiple instances of clamd. It has a color interface that shows the jobs in clamd's queue, memory usage and information about the loaded signature databases. By default, it will attempt to connect to the local clamd as defined in clamd.conf.

Clamscan

Clamscan is ClamAV's command line virus scanner. It can be used to scan files and directories for viruses. In order for clamscan to work proper, the ClamAV virus database files must be installed on the system we are using clamscan on.

The general usage of clamscan is : clamscan <options> <file/directory>

For more detailed help, type "man clamscan" or "clamscan -help"

ClamBC

Clambc is ClamAV's bytecode testing tool. It can be used to test files which contain bytecode.

Freshclam (this was important in running clamscan)

Freshclam is ClamAV's virus database update tool and reads its configuration from the file freshclam.conf. The default behavior is to attempt to update databases that are paired with downloaded cdiffs. Potentially corrupted databases are not updated and automatically fully replaced after several failed attempts unless otherwise specified.

There were modifications that were needed for our clamscan on terminal to work.

Clamconf

This is ClamAV's configuration utility, used for displaying values of configuration options in ClamAV, which will show the contents of clamd.conf, contents of freshclam.conf and display information about software settings, database, platform and build information.

Output format

Clamscan writes all regular program messages to stdout and errors/warnings to stderr. We can use the option --stdout to redirect all program messages to stdout. Warnings and error messages from libclamav are always printed to stderr. An example output is:

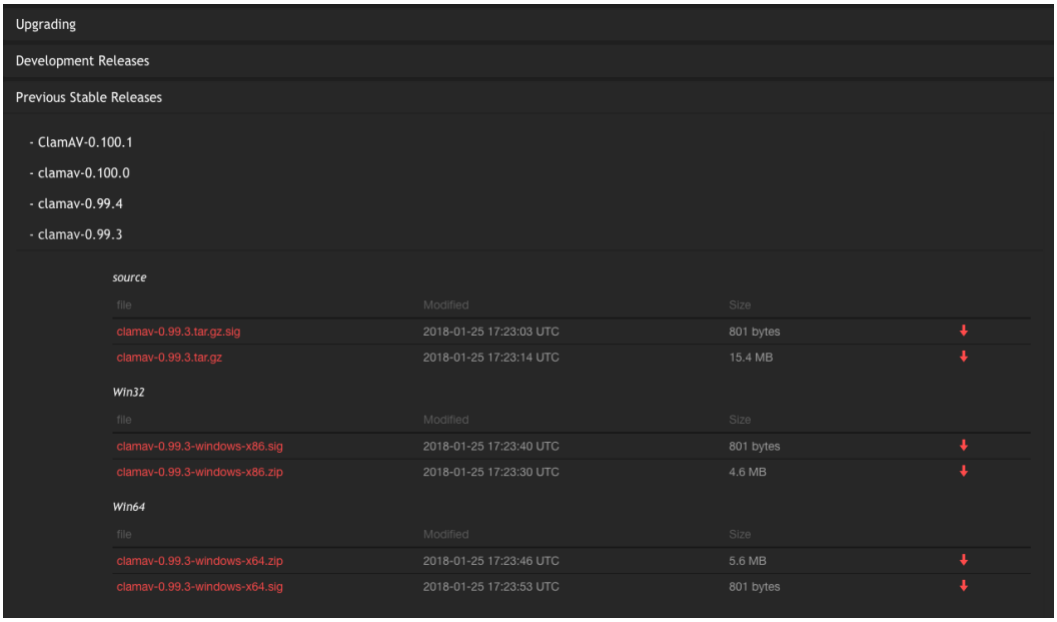
```
/tmp/test/removal-tool.exe: Worm.Sober FOUND
/tmp/test/md5.o: OK
/tmp/test/blob.c: OK
/tmp/test/message.c: OK
/tmp/test/error.hta: VBS.Inor.D FOUND
```

It is key to note that when a virus is found, the name is printed between the <filename:> and FOUND strings. For example:

```
$ clamscan malware.zip
malware.zip: Worm.Mydoom.U FOUND
```

Installation of ClamAV (**Using SEEDUbuntu VM, Ubuntu 32 bit OS**)

From https://www.clamav.net/downloads, we scrolled down and went to Previous Stable Releases -> clamav-0.99.3 and downloaded clamav-0.99.3.tar.gz(since the vulnerable versions are before 0.99.4):



After downloading this .tar.gz, we extracted the folder and clicked on INSTALL found within the clamav-0.99.3. This was the contents of INSTALL, an ACSII text file:

```
The simplest way to compile this package is:

  1. `cd' to the directory containing the package's source code and type
     `./configure' to configure the package for your system.

     Running `configure' might take a while.  While running, it prints
     some messages telling which features it is checking for.

  2. Type `make' to compile the package.

  3. Optionally, type `make check' to run any self-tests that come with
     the package.

  4. Type `sudo make install' to install the programs and any data files and
     documentation.

  5. You can remove the program binaries and object files from the
     source code directory by typing `make clean'.  To also remove the
     files that `configure' created (so you can compile the package for
     a different kind of computer), type `make distclean'.  There is
     also a `make maintainer-clean' target, but that is intended mainly
     for the package's developers.  If you use it, you may have to get
     all sorts of other programs in order to regenerate files that came
     with the distribution.
```

First, we ran "./configure" in within the folder clamav-0.99.3:

```
checking for the tclsh program in tclinclude directory... none
checking for tclsh8.4... no
checking for tclsh8.4.8... no
checking for tclsh8.4.7... no
checking for tclsh8.4.6... no
checking for tclsh8.4.5... no
checking for tclsh8.4.4... no
checking for tclsh8.4.3... no
checking for tclsh8.4.2... no
checking for tclsh8.4.1... no
checking for tclsh8.4.0... no
checking for tclsh8.3... no
checking for tclsh8.3.5... no
checking for tclsh8.3.4... no
checking for tclsh8.3.3... no
checking for tclsh8.3.2... no
checking for tclsh8.3.1... no
checking for tclsh8.3.0... no
checking for tclsh... /usr/bin/tclsh
checking for zip... /usr/bin/zip
checking for ocamlc... no
checking for ocamlopt... no
checking for ocamldep... no
checking for ocamldoc... no
checking for gas... no
checking for as... /usr/bin/as
checking for linker version... 2.26.1
checking for compiler -Wl,-R<path> option... yes
checking for compiler -Wl,-export-dynamic option... yes
checking for compiler -Wl,--version-script option... yes
checking for an ANSI C-conforming const... yes
checking for dirent.h that defines DIR... yes
```


Second, we ran "make" within the folder clamav-0.99.3. This took some time:

```
In file included from ./llvm/include/llvm/BasicBlock.h:18:0,
                 from ./llvm/include/llvm/Function.h:23,
                 from ./llvm/include/llvm/Analysis/Dominators.h:25,
                 from ./llvm/include/llvm/Analysis/LoopInfo.h:39,
                 from bytecode2llvm.cpp:46:
./llvm/include/llvm/SymbolTableListTraits.h: In member function 'ItemParentClass* llvm::SymbolTableListTraits<
ValueSubClass, ItemParentClass>::getListOwner()':
./llvm/include/llvm/SymbolTableListTraits.h:49:53: warning: typedef 'Sublist' locally defined but not used [-W
unused-local-typedefs]
     typedef iplist<ValueSubClass> ItemParentClass::*Sublist;
                                                     ^

In file included from bytecode2llvm.cpp:46:0:
./llvm/include/llvm/Analysis/LoopInfo.h: In member function 'BlockT* llvm::LoopBase<N, M>::getLoopPredecessor(
) const':
./llvm/include/llvm/Analysis/LoopInfo.h:288:34: warning: typedef 'BlockTraits' locally defined but not used [-
Wunused-local-typedefs]
     typedef GraphTraits<BlockT*> BlockTraits;
                                  ^

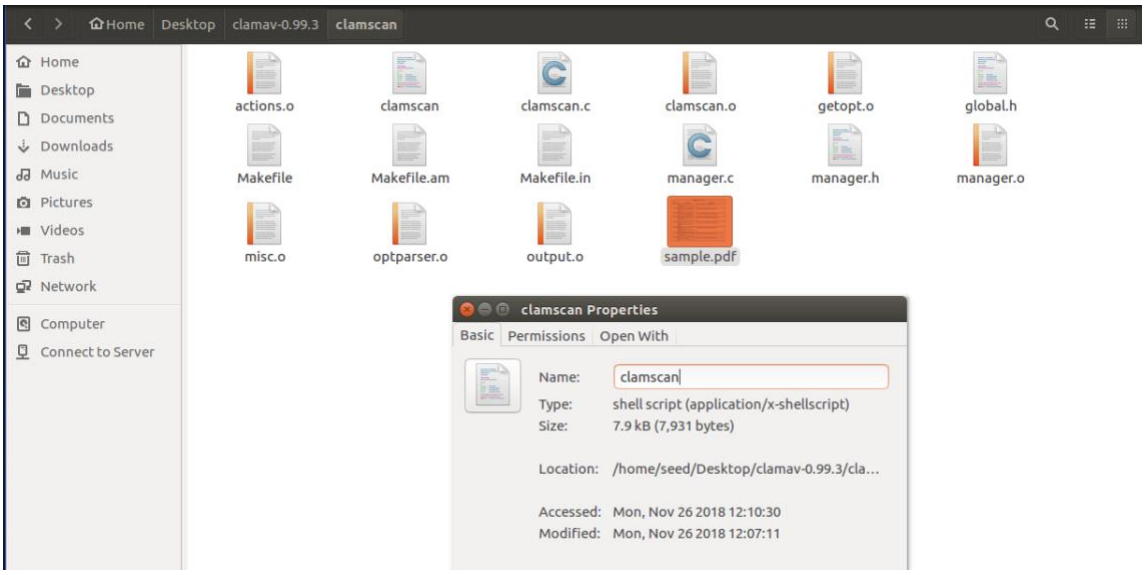
bytecode2llvm.cpp: In function 'void setGuard(unsigned char*)':
bytecode2llvm.cpp:2432:49: warning: deprecated conversion from string constant to 'char*' [-Wwrite-strings]
     cl_hash_data("md5", salt, 48, guardbuf, NULL);
                                             ^

  CXX    libclamavcxx_la-ClamBCRTChecks.lo
In file included from ./llvm/include/llvm/BasicBlock.h:18:0,
                 from ./llvm/include/llvm/Function.h:23,
                 from ./llvm/include/llvm/Analysis/CallGraph.h:54,
                 from ClamBCRTChecks.cpp:31:
./llvm/include/llvm/SymbolTableListTraits.h: In member function 'ItemParentClass* llvm::SymbolTableListTraits<
ValueSubClass, ItemParentClass>::getListOwner()':
./llvm/include/llvm/SymbolTableListTraits.h:49:53: warning: typedef 'Sublist' locally defined but not used [-W
unused-local-typedefs]
     typedef iplist<ValueSubClass> ItemParentClass::*Sublist;
                                                     ^
```

Last, we ran "sudo make install" within the folder clamav-0.99.3:

```
make[2]: Nothing to be done for 'install-exec-am'.
make[2]: Nothing to be done for 'install-data-am'.
make[2]: Leaving directory '/home/seed/Desktop/clamav-0.99.3/test'
make[1]: Leaving directory '/home/seed/Desktop/clamav-0.99.3/test'
Making install in clamdtop
make[1]: Entering directory '/home/seed/Desktop/clamav-0.99.3/clamdtop'
make[2]: Entering directory '/home/seed/Desktop/clamav-0.99.3/clamdtop'
make[2]: Leaving directory '/home/seed/Desktop/clamav-0.99.3/clamdtop'
make[1]: Leaving directory '/home/seed/Desktop/clamav-0.99.3/clamdtop'
Making install in clambc
make[1]: Entering directory '/home/seed/Desktop/clamav-0.99.3/clambc'
make[2]: Entering directory '/home/seed/Desktop/clamav-0.99.3/clambc'
 /bin/mkdir -p '/usr/local/bin'
  /bin/sh ../libtool   --mode=install /usr/bin/install -c clambc '/usr/local/bin'
libtool: install: /usr/bin/install -c .libs/clambc /usr/local/bin/clambc
make[2]: Nothing to be done for 'install-data-am'.
make[2]: Leaving directory '/home/seed/Desktop/clamav-0.99.3/clambc'
make[1]: Leaving directory '/home/seed/Desktop/clamav-0.99.3/clambc'
Making install in unit_tests
make[1]: Entering directory '/home/seed/Desktop/clamav-0.99.3/unit_tests'
make[2]: Entering directory '/home/seed/Desktop/clamav-0.99.3/unit_tests'
make[2]: Nothing to be done for 'install-exec-am'.
make[2]: Nothing to be done for 'install-data-am'.
make[2]: Leaving directory '/home/seed/Desktop/clamav-0.99.3/unit_tests'
make[1]: Leaving directory '/home/seed/Desktop/clamav-0.99.3/unit_tests'
make[1]: Entering directory '/home/seed/Desktop/clamav-0.99.3'
make[2]: Entering directory '/home/seed/Desktop/clamav-0.99.3'
 /bin/mkdir -p '/usr/local/bin'
 /usr/bin/install -c clamav-config '/usr/local/bin'
 /bin/mkdir -p '/usr/local/lib/pkgconfig'
 /usr/bin/install -c -m 644 libclamav.pc '/usr/local/lib/pkgconfig'
make[2]: Leaving directory '/home/seed/Desktop/clamav-0.99.3'
make[1]: Leaving directory '/home/seed/Desktop/clamav-0.99.3'
```

As a result, when we got into the clamscan folder, we could see clamscan, which a shell script, ready for usage.



First, inside the clamscan folder, when we tried running clamscan on sample.pdf, we encountered this:

```
[11/26/18]seed@VM:~/.../clamscan$ sudo clamscan sample.pdf
[sudo] password for seed:
clamscan: error while loading shared libraries: libclamav.so.7: cannot open shared object file: No such file o
r directory
```

To solve this, we did this:

```
[11/26/18]seed@VM:~/.../clamscan$ sudo apt-get install --reinstall libclamav7
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libllvm3.6v5 libmspack0
Suggested packages:
  libclamunrar7
The following NEW packages will be installed:
  libclamav7 libllvm3.6v5 libmspack0
0 upgraded, 3 newly installed, 0 to remove and 5 not upgraded.
Need to get 9,853 kB of archives.
After this operation, 38.2 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://us.archive.ubuntu.com/ubuntu xenial/main i386 libmspack0 i386 0.5-1 [40.9 kB]
Get:2 http://us.archive.ubuntu.com/ubuntu xenial/main i386 libllvm3.6v5 i386 1:3.6.2-3ubuntu2 [8,992 kB]
Get:3 http://us.archive.ubuntu.com/ubuntu xenial/main i386 libclamav7 i386 0.99+dfsg-1ubuntu1 [820 kB]
Fetched 9,853 kB in 0s (9,870 kB/s)
Selecting previously unselected package libmspack0:i386.
(Reading database ... 194199 files and directories currently installed.)
Preparing to unpack .../libmspack0_0.5-1_i386.deb ...
Unpacking libmspack0:i386 (0.5-1) ...
Selecting previously unselected package libllvm3.6v5:i386.
Preparing to unpack .../libllvm3.6v5_1%3a3.6.2-3ubuntu2_i386.deb ...
Unpacking libllvm3.6v5:i386 (1:3.6.2-3ubuntu2) ...
Selecting previously unselected package libclamav7.
Preparing to unpack .../libclamav7_0.99+dfsg-1ubuntu1_i386.deb ...
Unpacking libclamav7 (0.99+dfsg-1ubuntu1) ...
Processing triggers for libc-bin (2.23-0ubuntu5) ...
Setting up libmspack0:i386 (0.5-1) ...
Setting up libllvm3.6v5:i386 (1:3.6.2-3ubuntu2) ...
Setting up libclamav7 (0.99+dfsg-1ubuntu1) ...
```

Next, we encountered this:

```
[11/26/18]seed@VM:~/.../clamscan$ sudo clamscan sample.pdf
LibClamAV Error: cl_load(): No such file or directory: /usr/local/share/clamav
ERROR: Can't get file status

----------- SCAN SUMMARY -----------
Known viruses: 0
Engine version: 0.99.3
Scanned directories: 0
Scanned files: 0
Infected files: 0
Data scanned: 0.00 MB
Data read: 0.00 MB (ratio 0.00:1)
Time: 0.000 sec (0 m 0 s)
```

To solve this, we did this:

```
[11/26/18]seed@VM:~/.../clamscan$ sudo mkdir -p /usr/local/share/clamav
```

Now, when we went into freshclam folder, and tried to update the AV database:

```
[11/26/18]seed@VM:~/.../freshclam$ sudo freshclam
ERROR: Can't open/parse the config file /usr/local/etc/freshclam.conf
```

To solve this, we figured that we had to install these 2 packages:

```
[11/26/18]seed@VM:~/.../clamscan$ sudo apt-get install clamav
```

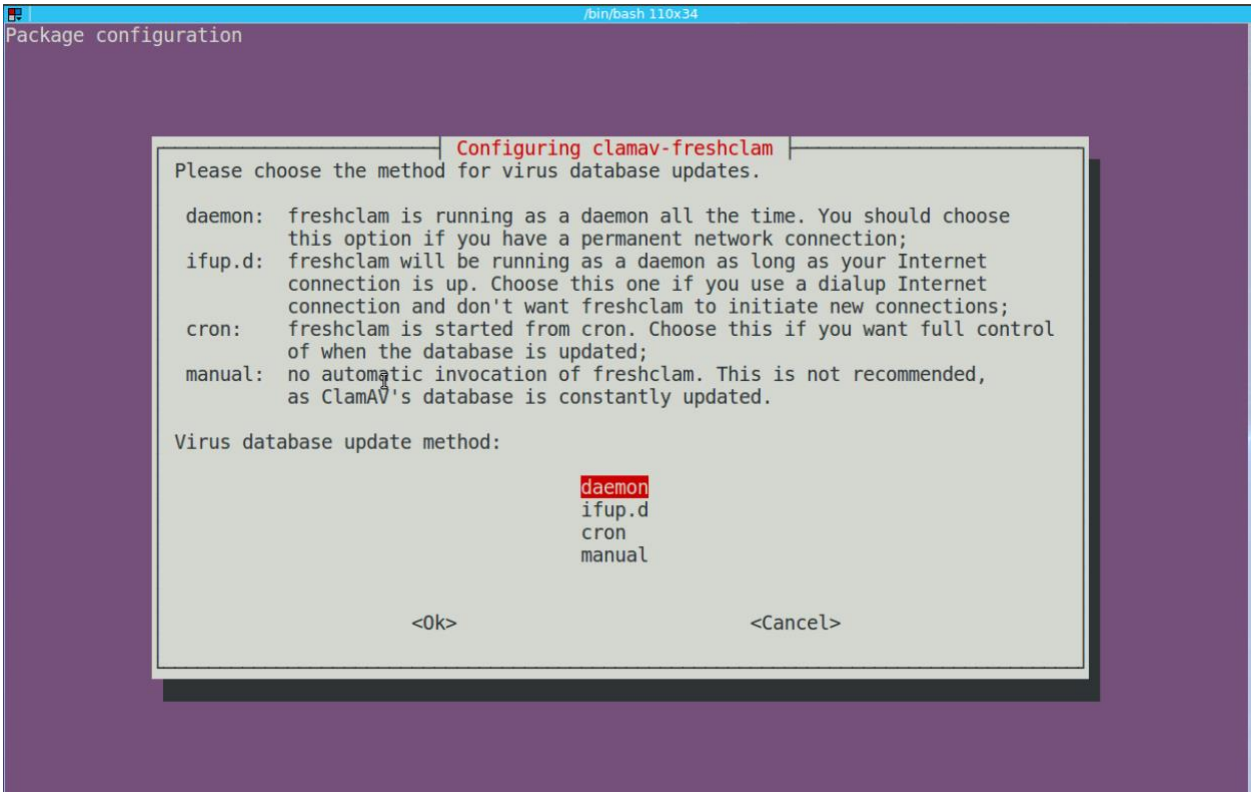
```
[11/26/18]seed@VM:~/.../clamscan$ sudo apt-get install clamtk
```

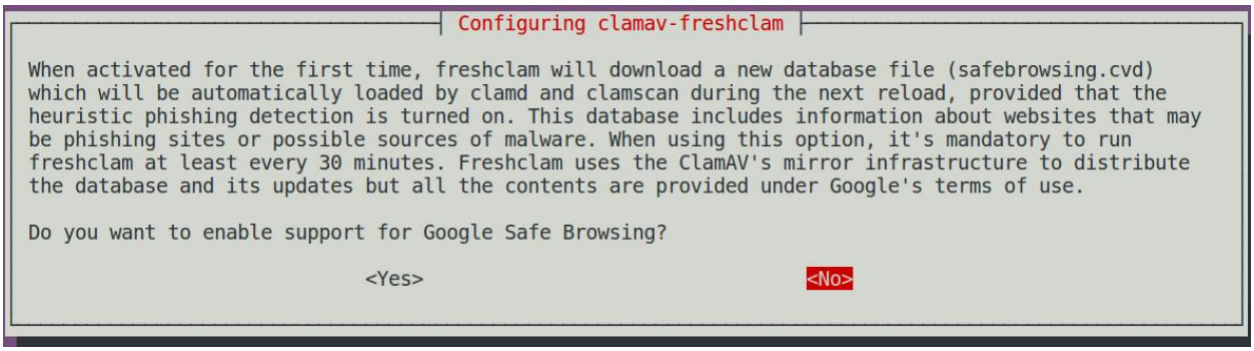
Note: clamtk is GUI interface for the ClamAV

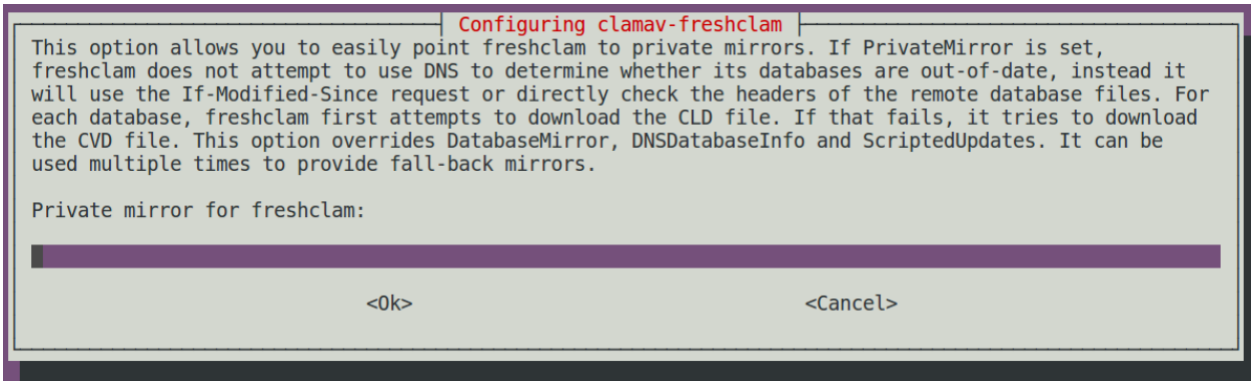
Then, we proceeded with this step:

```
[11/26/18]seed@VM:~/.../clamscan$ sudo dpkg-reconfigure clamav-freshclam
```

And we immediately got this GUI to configure freshclam:







Then, we run these:

```
[11/26/18]seed@VM:~$ sudo rm -f /usr/local/etc/freshclam.conf
[11/26/18]seed@VM:~$ sudo ln -s /etc/clamav/freshclam.conf /usr/local/etc/freshclam.conf
```

Note: we can update the anti-virus database by running “sudo freshclam”.

Now, we add a non-malicious sample.pdf file inside this folder and run “sudo clamscan sample.pdf”:

```
LibClamAV Warning: cli_loadldb: logical signature for Img.Malware.Rocke-6681160-2 uses PCREs but support is di
sabled, skipping
LibClamAV Warning: cli_loadldb: logical signature for Unix.Malware.Agent-6687840-0 uses PCREs but support is d
isabled, skipping
LibClamAV Warning: cli_loadldb: logical signature for Win.Trojan.RevengeRAT-6690354-0 uses PCREs but support i
s disabled, skipping
LibClamAV Warning: cli_loadldb: logical signature for Unix.Malware.Agent-6696064-0 uses PCREs but support is d
isabled, skipping
LibClamAV Warning: cli_loadldb: logical signature for Txt.Tool.ADAPE-6714123-0 uses PCREs but support is disab
led, skipping
LibClamAV Warning: cli_loadldb: logical signature for Html.Exploit.CVE_2018_8174-6700490-0 uses PCREs but supp
ort is disabled, skipping
LibClamAV Warning: cli_loadldb: logical signature for Win.Downloader.DDEObfuscatedCmdExec-6715127-0 uses PCREs
 but support is disabled, skipping
LibClamAV Warning: cli_loadldb: logical signature for Win.Downloader.DDEObfuscatedCmdExec-6715128-0 uses PCREs
 but support is disabled, skipping
LibClamAV Warning: cli_loadldb: logical signature for Win.Exploit.CVE_2018_12833-6731045-0 uses PCREs but supp
ort is disabled, skipping
LibClamAV Warning: cli_loadldb: logical signature for Win.Trojan.Zebrocy-6743852-2 uses PCREs but support is d
isabled, skipping
sample.pdf: OK

----------- SCAN SUMMARY -----------
Known viruses: 6718942
Engine version: 0.99.3
Scanned directories: 0
Scanned files: 1
Infected files: 0
Data scanned: 0.08 MB
Data read: 0.02 MB (ratio 5.00:1)
Time: 22.930 sec (0 m 22 s)
```

This shows that we got the vulnerable version of clamscan running on our VM and that sample.pdf is not a virus. In addition, we tested the anti virus software using a malware sample, and it worked by alerting the user:

```
EICAR_test.txt: Eicar-Test-Signature FOUND

----------- SCAN SUMMARY -----------
Known viruses: 6724000
Engine version: 0.99.3
Scanned directories: 0
Scanned files: 1
Infected files: 1
Data scanned: 0.00 MB
Data read: 0.00 MB (ratio 0.00:1)
Time: 18.368 sec (0 m 18 s)
```

Vulnerability

The main cause of the vulnerability is the two functions in **clamav-0.99.3/libclamav/pdfng.c** that allow a heap overflow to be exploited. The first one is pdf_parse_array() function(line 908 - 1076 in pdfng.c) in which it's likely that a pointer called 'end' could be incremented to point to whatever is after the object. This will cause an Out-of-Bounds read problem. The first read OOB occurs at line 966 below:

```
908 struct pdf_array *pdf_parse_array(struct pdf_struct *pdf, struct pdf_obj *obj, size_t objsz, char *begin,
    char **endchar)
909 {
910     struct pdf_array *res=NULL;
911     struct pdf_array_node *node=NULL;
912     const char *objstart;
913     char *end;
914     int in_string=0, ninner=0;
915
916     /* Sanity checking */
917     if (!(pdf) || !(obj) || !(begin))
918         return NULL;
919
920     objstart = obj->start + pdf->map;
921
922     if (begin < objstart || (size_t)(begin - objstart) >= objsz)
923         return NULL;
924
925     if (begin[0] != '[')
926         return NULL;
927
928     /* Find the end of the array */
929     end = begin;
930     while ((size_t)(end - objstart) < objsz) {
931         if (in_string) {
932             if (*end == '\\') {
933                 end += 2;
934                 continue;
935             }
936
937             if (*end == ')')
938                 in_string = 0;
```

```
939
940             end++;
941             continue;
942         }
943
944         switch (*end) {
945             case '(':
946                 in_string=1;
947                 break;
948             case '[':
949                 ninner++;
950                 break;
951             case ']':
952                 ninner--;
953                 break;
954         }
955
956         if (*end == ']' && ninner == 0)
957             break;
958
959         end++;
960     }
```

```
961
962     /* More sanity checking */
963     if ((size_t)(end - objstart) == objsz)
964         return NULL;
965
966     if (*end != ']')
967         return NULL;
968
969     res = cli_calloc(1, sizeof(struct pdf_array));
970     if (!(res))
971         return NULL;
972
973     begin++;
974     while (begin < end) {
975         char *val=NULL, *p1;
976         struct pdf_array *arr=NULL;
977         struct pdf_dict *dict=NULL;
978
979         while (begin < end && isspace(begin[0]))
980             begin++;
981
982         if (begin == end)
983             break;
984
985         switch (begin[0]) {
986             case '<':
987                 if ((size_t)(begin - objstart) < objsz - 2 && begin[1] == '<') {
988                     dict = pdf_parse_dict(pdf, obj, objsz, begin, &begin);
989                     begin+=2;
990                     break;
991                 }
```

```
992
993                 /* Not a dictionary. Intentially fall through. */
994             case '(':
995                 val = pdf_parse_string(pdf, obj, begin, objsz, NULL, &begin, NULL);
996                 begin += 2;
997                 break;
998             case '[':
999                 /* XXX We should have a recursion counter here */
1000                arr = pdf_parse_array(pdf, obj, objsz, begin, &begin);
1001                begin+=1;
1002                break;
1003            default:
1004                p1 = end;
1005                if (!is_object_reference(begin, &p1, NULL)) {
1006                    p1 = begin+1;
1007                    while (p1 < end && !isspace(p1[0]))
1008                        p1++;
1009                }
1010
1011                val = cli_calloc((p1 - begin) + 2, 1);
1012                if (!(val))
1013                    break;
1014
1015                strncpy(val, begin, p1 - begin);
1016                val[p1 - begin] = '\0';
1017
1018                begin = p1;
1019                break;
1020        }
1021
```

```
1022        /* Parse error, just return what we could */
1023        if (!(val) && !(arr) && !(dict))
1024            break;
1025
1026        if (!(node)) {
1027            res->nodes = res->tail = node = calloc(1, sizeof(struct pdf_array_node));
1028            if (!(node)) {
1029                if (dict)
1030                    pdf_free_dict(dict);
1031                if (val)
1032                    free(val);
1033                if (arr)
1034                    pdf_free_array(arr);
1035
1036                break;
1037            }
1038        } else {
1039            node = calloc(1, sizeof(struct pdf_array_node));
1040            if (!(node)) {
1041                if (dict)
1042                    pdf_free_dict(dict);
1043                if (val)
1044                    free(val);
1045                if (arr)
1046                    pdf_free_array(arr);
1047
1048                break;
1049            }
1050
1051            node->prev = res->tail;
1052            if (res->tail)
1053                res->tail->next = node;
```

```
1054                res->tail = node;
1055            }
1056
1057            if (val != NULL) {
1058                node->type = PDF_ARR_STRING;
1059                node->data = val;
1060                node->datasz = strlen(val);
1061            } else if (dict != NULL) {
1062                node->type = PDF_ARR_DICT;
1063                node->data = dict;
1064                node->datasz = sizeof(struct pdf_dict);
1065            } else {
1066                node->type = PDF_ARR_ARRAY;
1067                node->data = arr;
1068                node->datasz = sizeof(struct pdf_array);
1069            }
1070        }
1071
1072        if (endchar)
1073            *endchar = end;
1074
1075        return res;
1076    }
```

The second vulnerable function is pdf_parse_string() function(**line 378 - 640 in pdfng.c**) in which pointer called 'p2' keeps getting incremented to an address that it should not have been allowed to read from. This problem may cause a heap overflow:

```
char *pdf_parse_string(struct pdf_struct *pdf, struct pdf_obj *obj, const char *objstart, size_t objsize, const char
    *str, char **endchar, struct pdf_stats_metadata *meta)
{
    const char *q = objstart;
    char *p1, *p2;
    size_t len, checklen;
    char *res = NULL;
    uint32_t objid;
    size_t i;

    /*
     * Yes, all of this is required to find the start and end of a potentially UTF-* string
     *
     * First, find the key of the key/value pair we're looking for in this object.
     * Second, determine whether the value points to another object (NOTE: this is sketchy behavior)
     * Third, attempt to determine if we're ASCII or UTF-*
     * If we're ASCII, just copy the ASCII string into a new heap-allocated string and return that
     * Fourth, Attempt to decode from UTF-* to UTF-8
     */

    if (str) {
        checklen = strlen(str);

        if (objsize < strlen(str) + 3)
            return NULL;

        for (p1=(char *)q; (size_t)(p1 - q) < objsize-checklen; p1++)
            if (!strncmp(p1, str, checklen))
                break;
```

```
        if ((size_t)(p1 - q) == objsize - checklen)
            return NULL;

        p1 += checklen;
    } else {
        p1 = (char *)q;
    }

    while ((size_t)(p1 - q) < objsize && isspace(p1[0]))
        p1++;

    if ((size_t)(p1 - q) == objsize)
        return NULL;

    /*
     * If str is non-null:
     *     We should be at the start of the string, minus 1
     * Else:
     *     We should be at the start of the string
     */

    p2 = (char *)(q + objsize);
    if (is_object_reference(p1, &p2, &objid)) {
        struct pdf_obj *newobj;
        char *begin, *p3;
        STATBUF sb;
        uint32_t objflags;
        int fd;
        size_t objsize2;

        newobj = find_obj(pdf, obj, objid);
        if (!(newobj))
            return NULL;

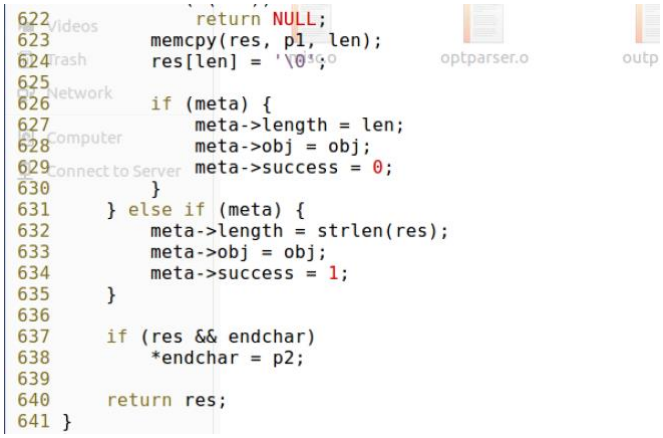
        if (newobj == obj)
            return NULL;
```

```
443
444         /*
445          * If pdf_handlename hasn't been called for this object,
446          * then parse the object prior to extracting it
447          */
448         if (!(newobj->statsflags & OBJ_FLAG_PDFNAME_DONE))
449             pdf_parseobj(pdf, newobj);
450
451         /* Extract the object. Force pdf_extract_obj() to dump this object. */
452         objflags = newobj->flags;
453         newobj->flags |= (1 << OBJ_FORCEDUMP);
454
455         if (pdf_extract_obj(pdf, newobj, PDF_EXTRACT_OBJ_NONE) != CL_SUCCESS)
456             return NULL;
457
458         newobj->flags = objflags;
459
460         if (!(newobj->path))
461             return NULL;
462
463         fd = open(newobj->path, O_RDONLY);
464         if (fd == -1) {
465             cli_unlink(newobj->path);
466             free(newobj->path);
467             newobj->path = NULL;
468             return NULL;
469         }
470
471         if (FSTAT(fd, &sb)) {
472             close(fd);
473             cli_unlink(newobj->path);
474             free(newobj->path);
475             newobj->path = NULL;
476             return NULL;
477         }
478
479         if (sb.st_size) {
```

```
480             begin = calloc(1, sb.st_size+1);
481             if (!(begin)) {
482                 close(fd);
483                 cli_unlink(newobj->path);
484                 free(newobj->path);
485                 newobj->path = NULL;
486                 return NULL;
487             }
488
489             if (read(fd, begin, sb.st_size) != sb.st_size) {
490                 close(fd);
491                 cli_unlink(newobj->path);
492                 free(newobj->path);
493                 newobj->path = NULL;
494                 free(begin);
495                 return NULL;
496             }
497
498             p3 = begin;
499             objsize2 = sb.st_size;
500             while ((size_t)(p3 - begin) < objsize2 && isspace(p3[0])) {
501                 p3++;
502                 objsize2--;
503             }
504
505             switch (*p3) {
506                 case '(':
507                 case '<':
508                     res = pdf_parse_string(pdf, obj, p3, objsize2, NULL, NULL, meta);
509                     break;
510                 default:
511                     res = pdf_finalize_string(pdf, obj, begin, objsize2);
512                     if (!res) {
513                         res = cli_calloc(1, objsize2+1);
514                         if (!(res)) {
515                             close(fd);
```

```
516                 cli_unlink(newobj->path);
517                 free(newobj->path);
518                 newobj->path = NULL;
519                 free(begin);
520                 return NULL;
521             }
522             memcpy(res, begin, objsize2);
523             res[objsize2] = '\0';
524
525             if (meta) {
526                 meta->length = objsize2;
527                 meta->obj = obj;
528                 meta->success = 0;
529             }
530         } else if (meta) {
531             meta->length = strlen(res);
532             meta->obj = obj;
533             meta->success = 1;
534         }
535     }
536     free(begin);
537 }
538
539 close(fd);
540 cli_unlink(newobj->path);
541 free(newobj->path);
542 newobj->path = NULL;
543
544 if (endchar)
545     *endchar = p2;
546
547 return res;
548 }
549
```

```
586
587 /* We should be at the start of a string literal (...) here */
588 if (*p1 != '(')
589     return NULL;
590
591 /* Make a best effort to find the end of the string and determine if UTF-* */
592 p2 = ++p1;
593 while (p2 < objstart + objsize) {
594     int shouldbreak=0;
595
596     switch (*p2) {
597         case '\\':
598             p2++;
599             break;
600         case ')':
601             shouldbreak=1;
602             break;
603     }
604
605     if (shouldbreak) {
606         p2--;
607         break;
608     }
609
610     p2++;
611 }
612
613 if (p2 == objstart + objsize)
614     return NULL;
615
616 len = (size_t)(p2 - p1) + 1;
617
618 res = pdf_finalize_string(pdf, obj, p1, len);
619 if (!res) {
620     res = cli_calloc(1, len+1);
621     if (!(res))
```



```
622             return NULL;
623         memcpy(res, p1, len);
624         res[len] = '\0';
625
626         if (meta) {
627             meta->length = len;
628             meta->obj = obj;
629             meta->success = 0;
630         }
631     } else if (meta) {
632         meta->length = strlen(res);
633         meta->obj = obj;
634         meta->success = 1;
635     }
636
637     if (res && endchar)
638         *endchar = p2;
639
640     return res;
641 }
```

The code in line 596 causes the first out-of-bounds read to occur at the switch statement when p2 is de-referenced. In the while loop (lines 593 to 611), p2 keeps incrementing until it is equal or greater than objstart+objsize. Once this happens, the while loop condition is exited. The problem is when objsize is a large value and p2 keeps getting incremented unintentionally. In this case, the while loop is still running since the check is against the objsize+objstart. The pointer eventually gets incremented to an address that it should not be reading from, but ends up reading something from an address it should not be reading from.

For example, if p2 points to the string below that corresponds to a construction of a PDF below, p2 should stop after the first \r\n ideally, but due to the bug, it went past the first /r/n in the string below, and read from a place it should not be allowed to, causing a crash in the clamscan program.

>>>>0\t[1/j0] R\r\n>>\r\nstartxref\r\n4006\r\n%%EOF\r\n""

Note: This is a sample of the internal data of a PDF file

Exploitation

Based on https://bugzilla.clamav.net/show_bug.cgi?id=11973,

```
Backtrace at crash time:
```c
    #0 0x819aa90 in pdf_parse_string /tmp/clamav-devel/libclamav/pdfng.c:596
    #1 0x819bb3b in pdf_parse_dict /tmp/clamav-devel/libclamav/pdfng.c:797
    #2 0x819bbe9 in pdf_parse_dict /tmp/clamav-devel/libclamav/pdfng.c:807
    #3 0x819cacb in pdf_parse_array /tmp/clamav-devel/libclamav/pdfng.c:988
    #4 0x819bb6a in pdf_parse_dict /tmp/clamav-devel/libclamav/pdfng.c:801
    #5 0x819cacb in pdf_parse_array /tmp/clamav-devel/libclamav/pdfng.c:988
    #6 0x819cb2f in pdf_parse_array /tmp/clamav-devel/libclamav/pdfng.c:1000
    #7 0x819cb2f in pdf_parse_array /tmp/clamav-devel/libclamav/pdfng.c:1000
    #8 0x819bb6a in pdf_parse_dict /tmp/clamav-devel/libclamav/pdfng.c:801
    #9 0x818e212 in pdf_extract_obj /tmp/clamav-devel/libclamav/pdf.c:959
    #10 0x8196bab in cli_pdf /tmp/clamav-devel/libclamav/pdf.c:2517
    #11 0x80911b8 in cli_scanpdf /tmp/clamav-devel/libclamav/scanners.c:1925
    #12 0x809c1fc in magic_scandesc /tmp/clamav-devel/libclamav/scanners.c:3389
    #13 0x809e101 in cli_base_scandesc /tmp/clamav-devel/libclamav/scanners.c:3616
    #14 0x809e251 in cli_magic_scandesc /tmp/clamav-devel/libclamav/scanners.c:3625
    #15 0x809f8c4 in scan_common /tmp/clamav-devel/libclamav/scanners.c:3891
    #16 0x809fa48 in cl_scandesc_callback /tmp/clamav-
devel/libclamav/scanners.c:4032
    #17 0x809fcdc in cl_scanfile_callback /tmp/clamav-
devel/libclamav/scanners.c:4099
    #18 0x809fc6b in cl_scanfile /tmp/clamav-devel/libclamav/scanners.c:4082
```
```

```
Detailed backtrace during the last loop prior to crash

```c
#0  pdf_parse_string (pdf=0xffffd520, obj=0xf3d03c80, objstart=0xf6404446 "
(>>>>0\t[1/j0] R\r\n>>\r\nstartxref\r\n4006\r\n%%EOF\r\n", objsize=13410, str=0x0,
endchar=0xffffc980, meta=0x0) at pdfng.c:605
#1  0x0819bb3c in pdf_parse_dict (pdf=0xffffd520, obj=0xf3d03c80, objsz=13410,
begin=0xf6404446 "(>>>>0\t[1/j0] R\r\n>>\r\nstartxref\r\n4006\r\n%%EOF\r\n",
endchar=0xffffca70) at pdfng.c:797
#2  0x0819bbea in pdf_parse_dict (pdf=0xffffd520, obj=0xf3d03c80, objsz=13410,
begin=0xf640443b "<</C.2\371er)>(>>>>0\t[1/j0]
R\r\n>>\r\nstartxref\r\n4006\r\n%%EOF\r\n", endchar=0xffffcbbc) at pdfng.c:807
#3  0x0819cacc in pdf_parse_array (pdf=0xffffd520, obj=0xf3d03c80, objsz=13410,
begin=0xf6403ac5 "<<\030Type/OCG/Name (\377\376v", endchar=0xffffcc40) at
pdfng.c:988
#4  0x0819bb6b in pdf_parse_dict (pdf=0xffffd520, obj=0xf3d03c80, objsz=13410,
    begin=0xf64038f9 "
[4\005\036Jh\ni\276h\202\024j\345Y\375\nT\212u\023\314\342y2\361\341\330\223\004\061\062\225\025ZF\233k\263\211\215\270\274YI\207L\354\066\322!X

\365S\023\375\324\330\241\356b\252\371\020\251J\351\026\016\034\277\017\303=bL\276H/\323\304s\377tor
 5 0 R\r\n>>\r\ne\210dobj\r\n8 0 obj\r\n[\r\n7 0 R \r\n]\r\nendobj\r\n9 /Rnt
\r\n/BaseFont \016SBDGDD+OCHAIconsBound\001", endchar=0xffffcd8c)
    at pdfng.c:801
#5  0x0819cacc in pdf_parse_array (pdf=0xffffd520, obj=0xf3d03c80, objsz=13410,
begin=0xf6401bd1 "<</T`pe/OCG/Name (\377\376v", endchar=0xffffce6c) at pdfng.c:988
#6  0x0819cb30 in pdf_parse_array (pdf=0xffffd520, obj=0xf3d03c80, objsz=13410,
begin=0xf64015a0 "[\r\n7 0\313n\252\372", endchar=0xffffcf4c) at pdfng.c:1000
#7  0x0819cb30 in pdf_parse_array (pdf=0xffffd520, obj=0xf3d03c80, objsz=13410,
    begin=0xf6401181 "[-1 -33
ilterF5O;\r+F\276\360\241\371\362\270\332;:\321\377s\214n\361\224m\241\360\2551\330\353(\307\245\316\323Kn\252-
1.s<</Colimns 4/IF/P\001", endchar=0xffffcfd0) at pdfng.c:1000
#8  0x0819bb6b in pdf_parse_dict (pdf=0xffffd520, obj=0xf3d03c80, objsz=13410,
begin=0xf6401048 "[n\237 213/Root 12f/W[1 2 1]>>stream\r\nh\336bed\020``",
endchar=0x0) at pdfng.c:801
#9  0x0818e213 in pdf_extract_obj (pdf=0xffffd520, obj=0xf3d03c80, flags=1) at
pdf.c:959
#10 0x08196bac in cli_pdf (dir=0xf5f00e00 "/tmp/clamav-
8d8c23b61eda37c98aca7636c053763f.tmp", ctx=0xffffdad0, offset=0) at pdf.c:2517
#11 0x080911b9 in cli_scanpdf (ctx=0xffffdad0, offset=0) at scanners.c:1925
#12 0x0809c1fd in magic_scandesc (ctx=0xffffdad0, type=CL_TYPE_PDF) at
scanners.c:3389
#13 0x0809e102 in cli_base_scandesc (desc=3, ctx=0xffffdad0, type=CL_TYPE_ANY) at
scanners.c:3616
#14 0x0809e252 in cli_magic_scandesc (desc=3, ctx=0xffffdad0) at scanners.c:3625
#15 0x0809f8c5 in scan_common (desc=3, map=0x0, virname=0xffffdc90,
scanned=0xffffdcd0, engine=0xf4303c80, scanoptions=54551095, context=0x0) at
```
```

As shown above, the backtrace of the crash is deeply nested and complex to fully understand. Therefore, we, as a group, tried to replicate this exploit.

First, we received this hint

```
For example initially p2 (which is the argument objstart+1) points to the string
below, it should really stop after the first \r\n in an ideal scenario. However
objsize is 13410 initially as well,
```
```
>>>>0\t[1/j0] R\r\n>>\r\nstartxref\r\n4006\r\n%%EOF\r\n""
```
```
This poc under gdb, `0xf64078a8` is the objstart+objsize address P2(0xf640444) is
compared against. The first oob occurs at 0xf605000 in my case.

Unfortunately  I don't quiet have a proposed patch for this. Obviously in this
specific case objsize should not be used as check before leaving loop, however that
might break other scenarios or the recursion done with this function so I will let
you guys figure out what the best patch is.
```

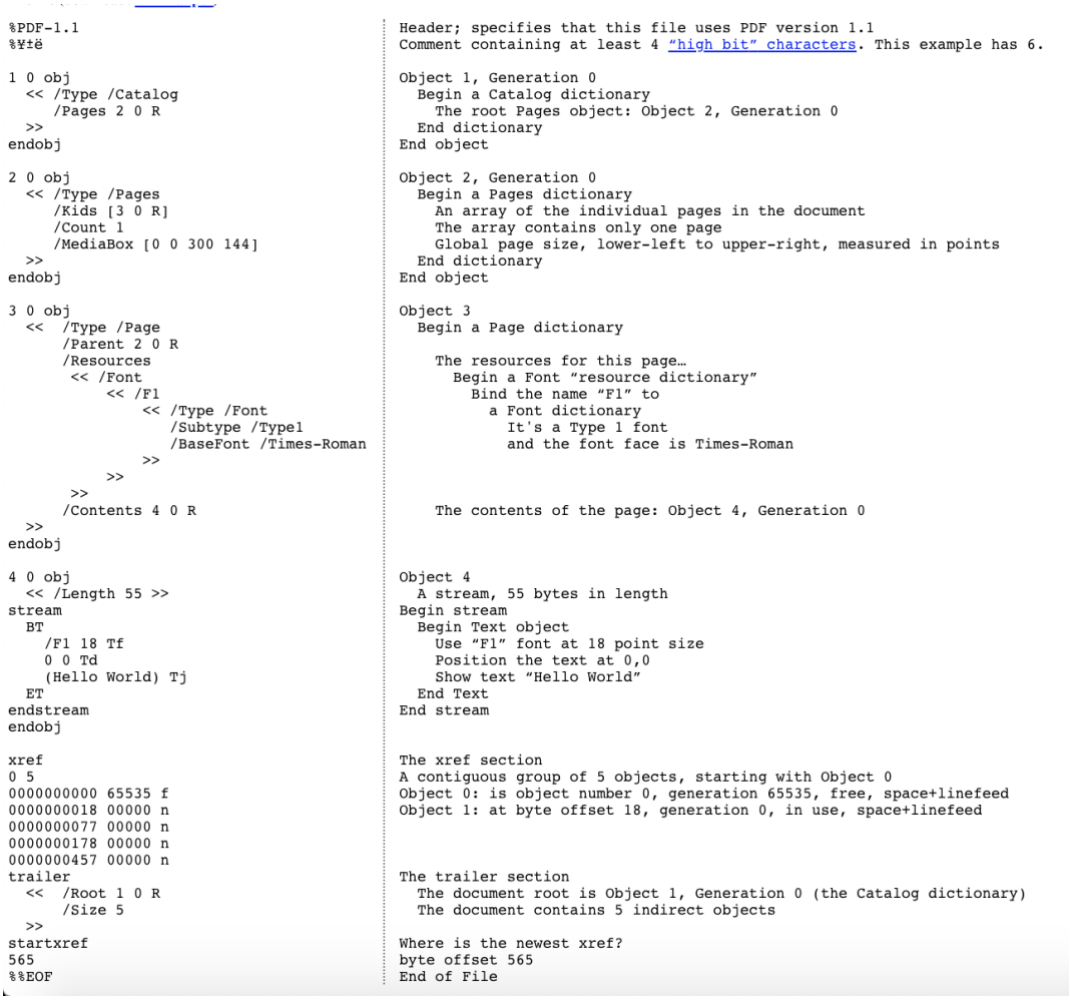
```
scanners.c:3891
#16 0x0809fa49 in cl_scandesc_callback (desc=3, virname=0xffffdc90,
scanned=0xffffdcd0, engine=0xf4303c80, scanoptions=54551095, context=0x0) at
scanners.c:4032
#17 0x0809fcdd in cl_scanfile_callback (filename=0xffffdee8
"/data_out/clamavpdf/newcrash1", virname=0xffffdc90, scanned=0xffffdcd0,
engine=0xf4303c80, scanoptions=54551095, context=0x0) at scanners.c:4099
#18 0x0809fc6c in cl_scanfile (filename=0xffffdee8 "/data_out/clamavpdf/newcrash1",
virname=0xffffdc90, scanned=0xffffdcd0, engine=0xf4303c80, scanoptions=54551095) at
scanners.c:4082
#19 0x0804bdf8 in main (argc=2, argv=0xffffddd4) at clamavfuzz.c:15
```

```
This vulnerability has been found by Offensive Research at Salesforce.com:
Alberto Garcia (@algillera), Francisco Oca (@francisco_oca) & Suleman Ali
(@Salbei_)
```

First, we learned that a PDF file is organized into 4 parts:

1) Header - The first line of the PDF file specifies the version number of the PDF specification to which the document adheres (i.e %PDF-1.3)
2) Body - Consist of objects that compromise the document's contents, includes text streams, image data, fonts, and annotations. The body could also contain invisible objects that contribute to the document's interactivity, security features or logical structure.
3) Cross-Reference Table - contains offsets to all of the objects in the file, so that it is not necessary to scan the large portions of a file to locate elements. New sections are added every time the file is modified.
4) Trailer - Enables an application reading the file to find the cross-reference table and certain special objects. Applications read a PDF file from the end. The last line of the PDF file contains the end-of-file, %%EOF.

A sample of a minimal pdf is shown:



```
%PDF-1.1
%¥±ë

1 0 obj
  << /Type /Catalog
     /Pages 2 0 R
  >>
endobj

2 0 obj
  << /Type /Pages
     /Kids [3 0 R]
     /Count 1
     /MediaBox [0 0 300 144]
  >>
endobj

3 0 obj
  <<  /Type /Page
      /Parent 2 0 R
      /Resources
       << /Font
           << /F1
               << /Type /Font
                  /Subtype /Type1
                  /BaseFont /Times-Roman
               >>
           >>
       >>
      /Contents 4 0 R
  >>
endobj

4 0 obj
  << /Length 55 >>
stream
  BT
    /F1 18 Tf
    0 0 Td
    (Hello World) Tj
  ET
endstream
endobj

xref
0 5
0000000000 65535 f
0000000018 00000 n
0000000077 00000 n
0000000178 00000 n
0000000457 00000 n
trailer
  <<  /Root 1 0 R
      /Size 5
  >>
startxref
565
%%EOF
```

```
Header; specifies that this file uses PDF version 1.1
Comment containing at least 4 "high bit" characters. This example has 6.

Object 1, Generation 0
  Begin a Catalog dictionary
    The root Pages object: Object 2, Generation 0
  End dictionary
End object

Object 2, Generation 0
  Begin a Pages dictionary
    An array of the individual pages in the document
    The array contains only one page
    Global page size, lower-left to upper-right, measured in points
  End dictionary
End object

Object 3
  Begin a Page dictionary

    The resources for this page…
      Begin a Font "resource dictionary"
        Bind the name "F1" to
          a Font dictionary
            It's a Type 1 font
            and the font face is Times-Roman



    The contents of the page: Object 4, Generation 0



Object 4
  A stream, 55 bytes in length
Begin stream
  Begin Text object
    Use "F1" font at 18 point size
    Position the text at 0,0
    Show text "Hello World"
  End Text
End stream


The xref section
A contiguous group of 5 objects, starting with Object 0
Object 0: is object number 0, generation 65535, free, space+linefeed
Object 1: at byte offset 18, generation 0, in use, space+linefeed



The trailer section
  The document root is Object 1, Generation 0 (the Catalog dictionary)
  The document contains 5 indirect objects

Where is the newest xref?
byte offset 565
End of File
```

As shown above, there are only five objects in minimal.pdf(including the object 0).

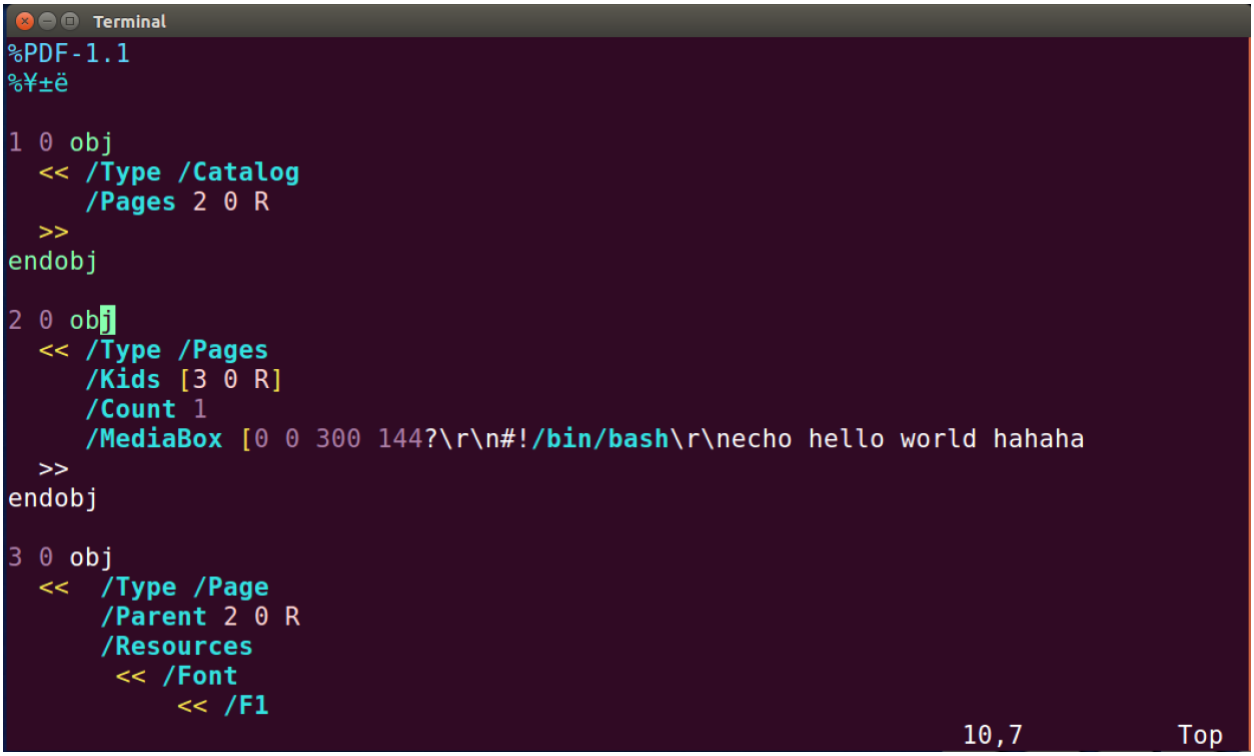
Based on the limited resources that we could get online and the complexity of the large code base, we found this clue(bugzilla.clamav.net/show_bug.cgi?id=11980) to develop our own exploit by crafting our

```
For example initially p2 (which is the argument objstart+1) points to the string
below, it should really stop after the first \r\n in an ideal scenario. However
objsize is 13410 initially as well,
```
```
>>>>0\t[1/j0] R\r\n>>\r\nstartxref\r\n4006\r\n%%EOF\r\n""
```

Attempt 1: Compare the example which is given in Bug 11980 with the last part of the minimal.pdf, we figured out that the code given above is the end part of this malicious pdf. Accordingly, we use that code to change the last part of minimal.pdf just like the figure shows below. Unfortunately, it didn't work.

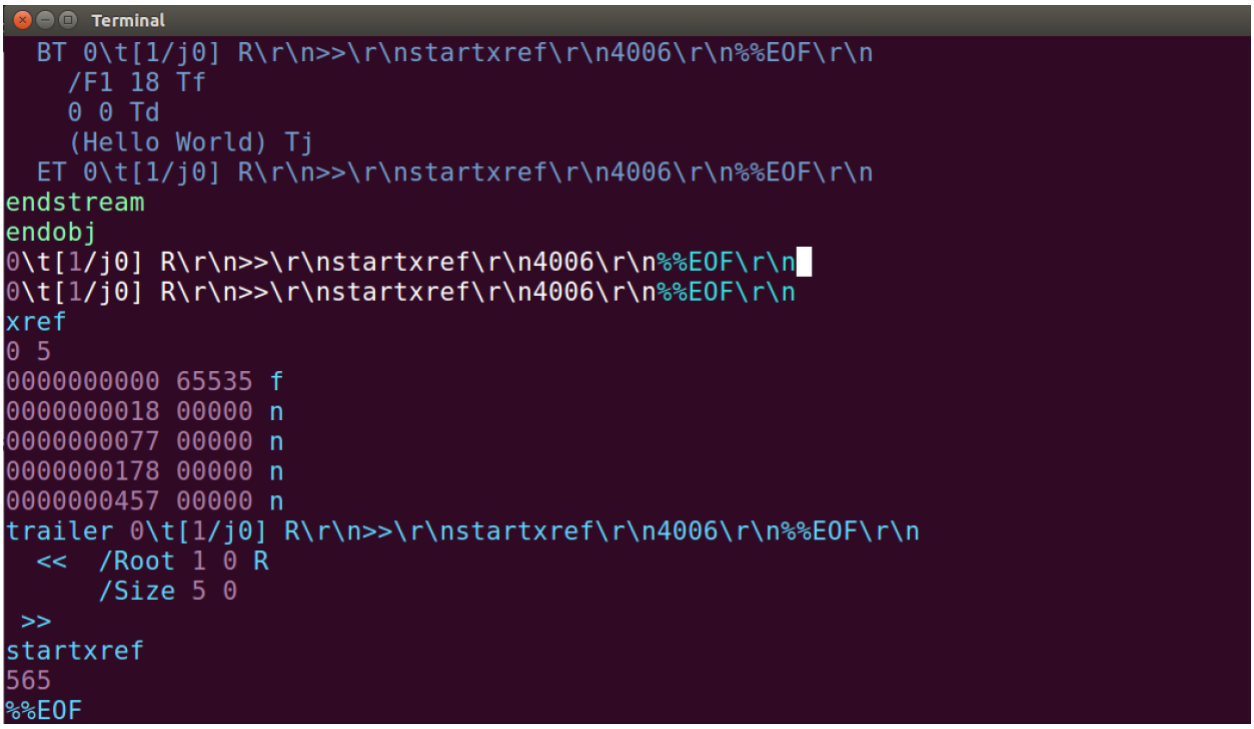
```
xref
0 5
0000000000 65535 f
0000000018 00000 n
0000000077 00000 n
0000000178 00000 n
0000000457 00000 n
trailer
  <<  /Root 1 0 R
      /Size 5 0\t[1/j0] R
 >>
startxref
565
%%EOF
```

Attempt 2: Combined with the information provided in Bug 11973, we thought maybe we could exploit the vulnerability by changing the end of a project. So as you can see in the figure below, we remove '] ' which is at the end of the second project as well as add some shell code. Even though it didn't work, we still get a better understanding of the structure of pdf.

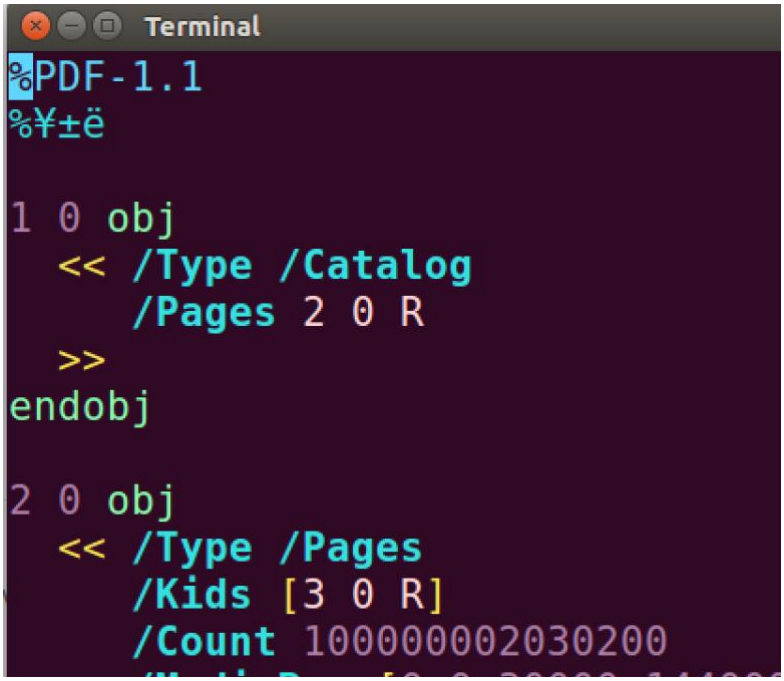


Attempt 3: As we can know from the information provided by Bug 11980, if p2 points past the first /r/n, it will read from a place it shouldn't be allowed and this will crash the program. So we force it to read the

end of the file by adding the code which is given in Bug 11980 randomly just like the figure below shows. As a result, this attempt failed too.



Attempt 4: The vulnerability existed because the *objsize* is a high value in some case. This results in that p2 to be incremented to an address it shouldn't have access to. So we attempt to exploit the vulnerability by changing the size of a random object just like the figure below shows. At last, this attempt failed either.



Solution

Recall that the first out-of-bound read occurs here when the pointer is dereferenced:

```
966     if (*end != ']')
967         return NULL;
```

The problem occurs as it is possible for the end pointer to be incremented to point to whatever is after the object. In the while loop to find the end of the array as shown below, if (end - objstart) is greater than objsz, the while loop will be escaped.

```
928     /* Find the end of the array */
929     end = begin;
930     while ((size_t)(end - objstart) < objsz) {
```

To fix the vulnerability below, we could change from:

```
962     /* More sanity checking */
963     if ((size_t)(end - objstart) == objsz)
964         return NULL;
```

To this:

```
962     /* More sanity checking */
963     if ((size_t)(end - objstart) >= objsz)
964         return NULL;
```

So if (end - objstart) is indeed greater than objsz (bad behavior), it will skip the earlier while loop and get into this sanity checking if statement to return NULL, which is safe behavior.

Conclusion

From this vulnerability, we learned the importance of proper input validation and boundary checks. User input is dangerous and without proper validation, it could lead to severe consequences. In this vulnerability, a malformatted user-crafted .pdf file can exploit the out-of-bounds vulnerability in ClamAV, DoSsing the antivirus program, and possibly causing the operating system to halt, which demonstrates how a simple glitch in the input validation and boundary check mechanism can result in a breakdown of a modern antivirus program. To fix the vulnerability, we patched the code of function pdf_parse_array() and pdf_parse_string() in libclamav/pdfng.c of the ClamAV antivirus engine. Potential countermeasures against the vulnerability include always keeping all software up-to-date, always validating user inputs properly, and enforcing thorough boundary checks.

References

Clam AntiVirus 0.100.0 User Manual

https://www.clamav.net/about

https://askubuntu.com/questions/526317/cant-run-freshclam

http://forum.directadmin.com/showthread.php?t=48957

https://askubuntu.com/questions/720928/clamav-not-working

https://en.wikipedia.org/wiki/EICAR_test_file

https://bugzilla.clamav.net/show_bug.cgi?id=11980

https://bugzilla.clamav.net/show_bug.cgi?id=11973

https://brendanzagaeski.appspot.com/0004.html

https://brendanzagaeski.appspot.com/0005.html