

You have become the very
thing you swore to destroy:

Remotely exploiting an
AntiVirus engine

echo \$USER

Simon, Cloud Vulnerability Researcher at Google



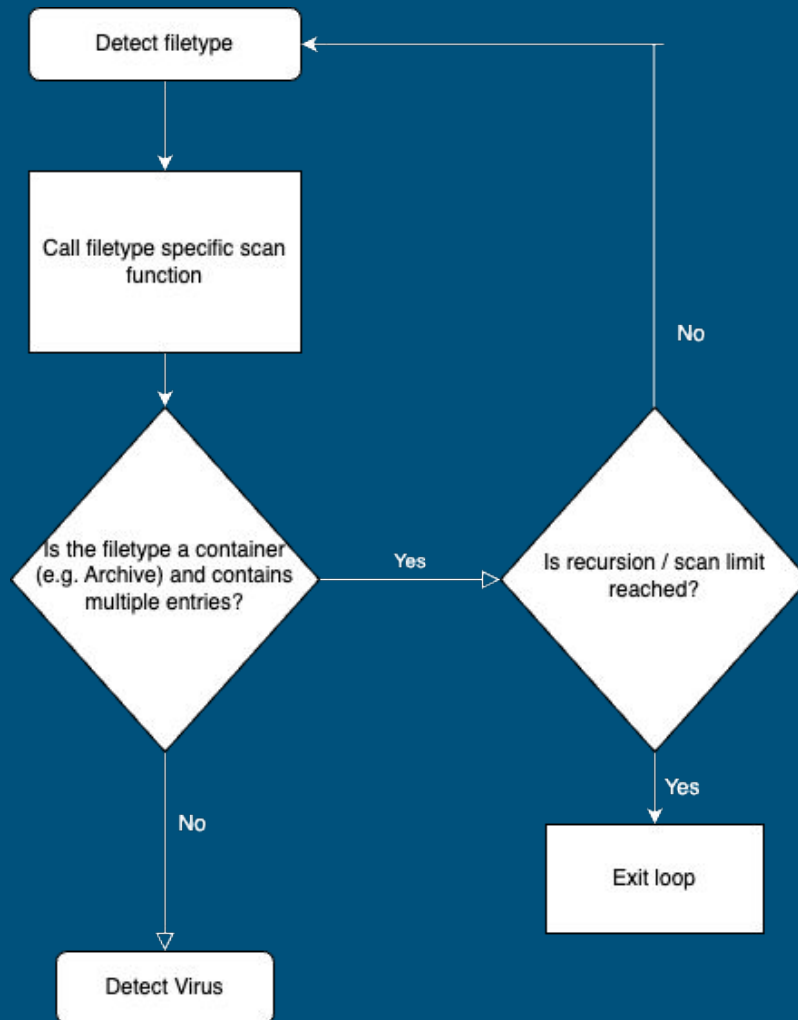
ClamAV: Why would an attacker pwn it?

- Open-Source AV engine for Linux, MacOS and Windows
- Popular for Linux clients
- Widely used in server-side contexts:
 - Email servers (e.g. Zimbra)
 - Appliances (e.g. Cisco Secure Web Appliance)
 - Cloud Storage etc.
- Written in C

=> Achieving Remote-Code-Execution would give an attacker privileged access to highly sensitive data such as emails and (up|down)loaded files

ClamAV High-Level Overview

- Supports file-type detection and recursive scanning of many file-formats
- e.g. unpacks ZIP or 7zip files, detects each files' type and either unpacks them or scans them for known viruses
- Recursive depth and number of files scanned depends on configuration settings



ClamAV: The Threat Model

- An external or internal attacker can upload files which are scanned in a backend by a ClamAV instance
- The attacker discovers a vulnerability in a parser for one of the many file formats ClamAV supports
 - e.g. HFS+, Autolt, CPIO, DMG and many more
- Trigger a vulnerability by getting a maliciously crafted file

Agenda for today

- 1) Why the usual approaches to info leaks won't work with remote ClamAV
- 2) Bugs we found
- 3) Exploitation strategy
- 4) Real-World case study: unauthenticated RCE on an email server

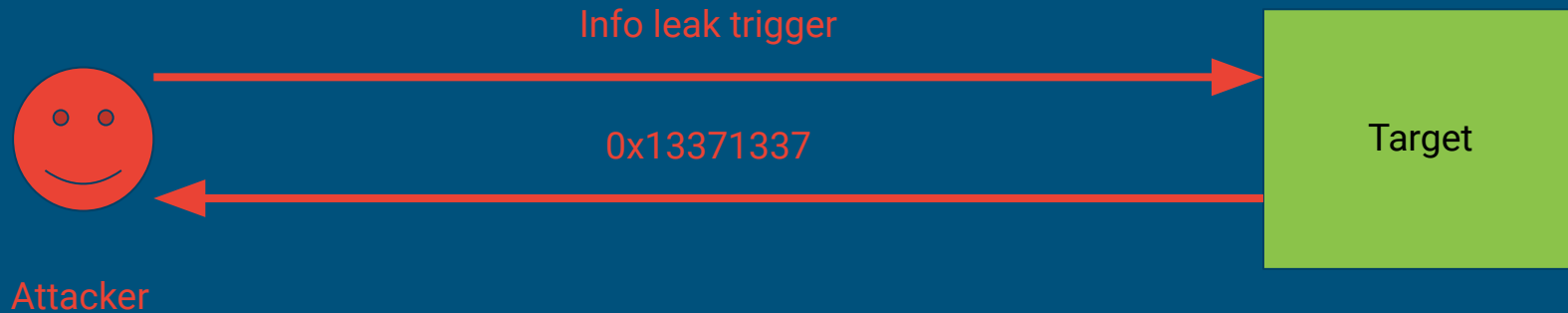
Why common Info-Leak strategies don't apply to ClamAV

The assumed environment

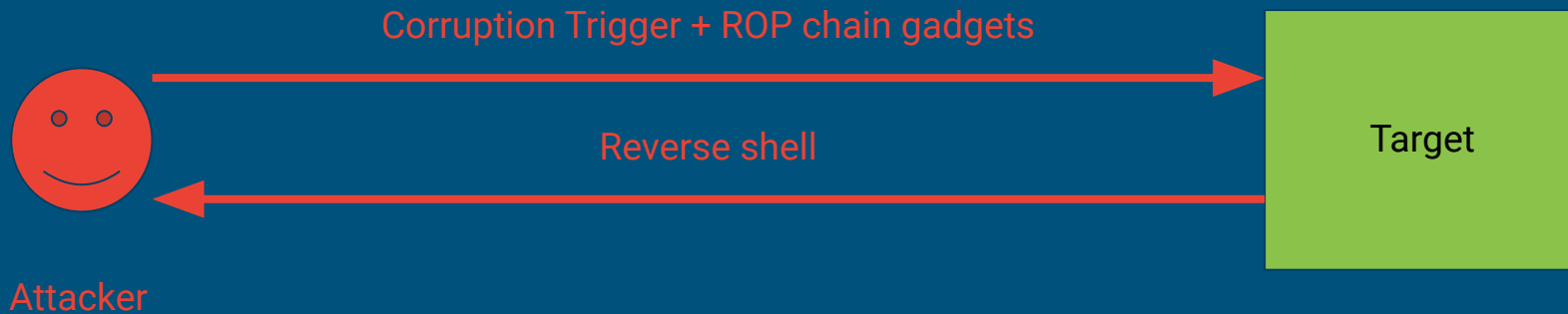
The assumed environment:

- Full ASLR applied to all loaded libraries and files
- An attacker gets to see the scan results (virus or no virus)
- Multi-threaded Server (ClamD)
 - A single segfault will crash all threads

Stage 1

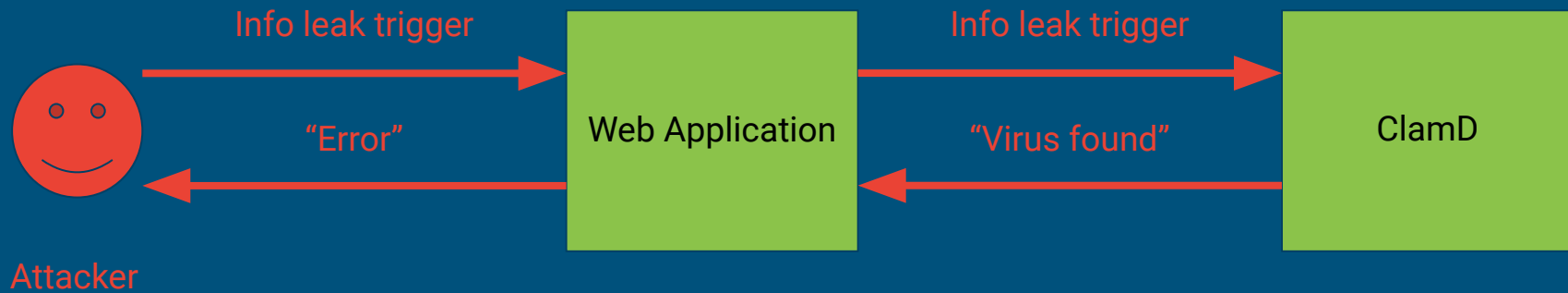


Stage 2



This doesn't apply to ClamAV

- ClamAV is a non-interactive target. At most, an attacker can see if a scan resulted in a virus being detected or not.
- In most environments an intermediary server communicates directly with ClamAV and gives us a custom error message. No direct interaction with ClamAV



Alternative approaches

- Partial pointer overwrite
 - Could overwrite a function pointer partially and call a function like `system()`
 - We didn't find a gadget for this, at least none that would have worked reliably
- Data only exploits
 - ClamAV dumps files to `/tmp` using a randomized filename before scanning them
 - If we can overwrite the path and get an arbitrary file-write primitive we might be able to get RCE
 - The files are created as `clamav` user and this won't work in sandboxed environments
 - Still: It could work for targets where the environment is known and can be exploited
- Bruteforce ASLR
 - By default runs in multi-threaded server mode
 - Server restarts are very costly. Just not feasible

The Bugs

Bug #1 - Heap Buffer Overflow

In February 2023, Cisco released an [advisory](#) on a Heap Buffer Overflow we reported:

- CVE-2023-20032
- CVS Score: 9.8
- Due to missing bounds check while parsing HFS+, a file-system by Apple
- Can be triggered remotely by an unauthenticated attacker
- Default configuration

Bug #1 - Heap Buffer Overflow

```
fileOffset = realFileBlock * volHeader->blockSize;  
readSize   = volHeader->blockSize;
```

```
if (curBlock == startBlock) {  
    fileOffset += startOffset;  
} else if (curBlock == endBlock) {  
    readSize = endSize;  
}
```

```
if (fmap_readn(ctx->fmap, buff + buffOffset, fileOffset, readSize) != readSize) {  
    cli_dbgmsg("hfsplus_fetch_node: not all bytes read\n");  
    return CL_EFORMAT;  
}
```


Bug #1 - Heap Buffer Overflow

The bug is powerful as:

- Overflow size can be controlled (must be in range of 4KB-2MB)
- Overflow contents can be arbitrarily controlled
- Can be triggered repeatedly in a loop

Bug #2 - OOB Read

We reported an Out-Of-Bounds Read bug to the maintainers of libmspack in the same timeframe

- libmspack is used by ClamAV to parse Microsoft Cabinet files (CAB and CHM)
- CVE-2023-29077
- Can be triggered remotely in default configurations
- Due to integer truncation

libmspack CHM files background

- CHM are Microsoft Compiled HTML Help Files
- Archive that contains compressed and uncompressed files
- Consists of:
 - CHM file header
 - Array of file headers
 - System files (contain Metadata about Compression)
 - Content files (the help files)
- ClamAV uses libmspack to extract CHM files

CHM Header

File Header Array



Uncompressed Data

Compressed Data

```
*/
struct mschmd_file {
    /**
     * A pointer to the next file in the list, or NULL if this is the final
     * file.
     */
    struct mschmd_file *next;

    /**
     * A pointer to the section that this file is located in. Indirectly,
     * it also points to the CHM helpfile the file is located in.
     */
    struct mschmd_section *section;

    /** The offset within the section data that this file is located at. */
    off_t offset;

    /** The length of this file, in bytes */
    off_t length;

    /** The filename of this file -- a null terminated string in UTF-8. */
    char *filename;
};
```

libmspack CHM files background

When a compressed file is extracted, things get more complicated...

- The file offset is not used as an offset into the file but as an index into the *Reset Table*
- libmspack reads the Reset Table from the attacker supplied file into memory and accesses it like an array to obtain the real offset within the file
- The Reset Table is an uncompressed file with a special name:
`::DataSpace/Storage/MSCompressed/Transform/{7FC28940-9D31-11D0-9B27-00A0C91E9C7C}/InstanceData/ResetTable`

CHM Header

File Header Array



Uncompressed Data

Compressed Data

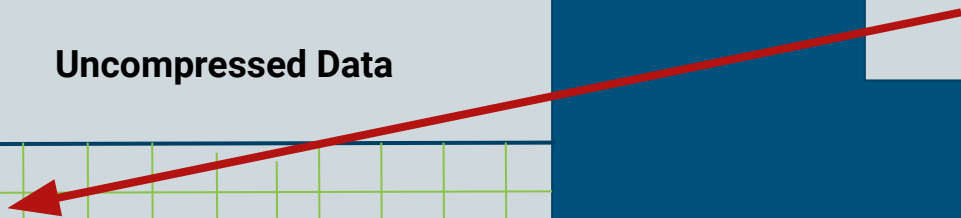
Reset Table File

1337

4096

8000

...



Bug #2 - OOB Read in Reset Table lookup

libmspack treats the Reset Table like a regular uncompressed file. It allocates **length** bytes of memory according to the file entry (attacker controlled)

However, it truncates the length field from 8 bytes to 4 bytes before allocating memory

```
len = (int) file->length;
```



```
if (!(data = (unsigned char *) sys->alloc(sys, (size_t) len))) {  
    self->error = MSPACK_ERR_NOMEMORY;  
    return NULL;  
}
```


Bug #2 - OOB Read in Reset Table lookup

A length of value $1 \ll 32$ | 1337 or 4294968633 would be truncated into 1337.

The resulting Reset Table buffer is 1337 bytes large

```
len = (int) file->length;

if (!(data = (unsigned char *) sys->alloc(sys, (size_t) len)))
    self->error = MSPACK_ERR_NOMEMORY;
    return NULL;
}
```



Bug #2 - OOB Read in Reset Table lookup

Following the example of a length of $1 \ll 32 \mid 1337$, the Reset Table Buffer is 1337 bytes large.

However, when the bounds check is made, the untruncated value of $1 \ll 32 \mid 1337$ is used as the upper bounds. Therefore we can read beyond the index 1337.

Bug #2 - OOB Read in Reset Table lookup

pos = 0xdeadbeef

sec->rtable->length = 4294968633 data = 1337

```
/* ensure reset table entry for this offset exists */
if (entry < EndGetI32(&data[lzxrt_NumEntries]) &&
    pos <= (sec->rtable->length - entrysize))
{
    switch (entrysize) {
    case 4:
        *offset_ptr = EndGetI32(&data[pos]);
```

Bug #2 - OOB Read in Reset Table lookup

Strong OOB Read primitive as:

- Size of the Reset Table Buffer can be arbitrarily controlled, Heap Feng Shui is easier
- OOB Read Size can be controlled (4 or 8 bytes)
- OOB index can be controlled

The resulting value of the OOB Read is interpreted as the offset into the CHM archive file

Let's pwn ClamAV

CHM Header

File Header Array



Uncompressed Data

Compressed Data

Known virus

To detect the known, compressed virus, the CHM file must be well-formed and libmspack must seek to it correctly

CHM Header

File Header Array



Uncompressed Data

Compressed Data

Known virus

Value read out of bounds (e.g. function pointer)

```
/* read the reset table entry */
if (read_reset_table(self, sec, entry, &length, &offset)) {
    // ...

    /* get offset of compressed data stream:
     * = offset of uncompressed section from start of file
     * + offset of compressed stream from start of uncompressed section
     * + offset of chosen reset interval from start of compressed stream */
    self->d->inoffset = file->section->chm->sec0.offset +
                      sec->content->offset +
                      offset;
```

Addition with fully attacker-controlled values

CHM Header

File Header Array



Uncompressed Data

Compressed Data

Known virus

```
/* seek to input data */  
if (sys->seek(self->d->infh, self->d->inoffset, MSPACK_SYS_SEEK_START)) {  
    self->error = MSPACK_ERR_SEEK;  
    break;  
}
```

CHM Header

File Header Array



Uncompressed Data

Compressed Data

Known virus

Let's assume:

- We can make a target ClamAV instance scan multiple files
- We can reliably prepare the heap and always read the same function pointer via our Out-Of-Bounds Read
- The only thing we change is the value of the offsets that are added to the function pointer that is read Out-Of-Bounds

CHM Header

File Header Array



Uncompressed Data

Compressed Data

Known virus

3 places we can **seek** to:

- 1) Outside of the file: **seek** returns an error, ClamAV stops the scan and no crash occurs. No virus is found
- 2) Somewhere inside the file but not at the beginning of the known virus: No virus is found
- 3) Exactly at the beginning of the virus: The virus is detected

CHM Header

File Header Array



Uncompressed Data

Compressed Data

Known virus



CHM Header

File Header Array



Uncompressed Data

Compressed Data

Known virus

At some point, the sum of the offset addition with the function pointer is exactly the offset of the known virus in the file. In this case it is detected

- This allows us to bruteforce ASLR without ever triggering the buffer overflow and crashing the server
- When the virus is detected, we know what the value of the function pointer was
- However, still need to bruteforce 28 bits of ASLR.. not feasible, right?

Known data

Function pointer

0x00	0x00	0x00	0x00	0x10	0x1a	0x88	0x7b
------	------	------	------	------	------	------	------

Out Of Bounds Read

Known data

Function pointer

0x00	0x00	0x00	0x00	0x10	0x1a	0x88	0x7b
------	------	------	------	------	------	------	------

Out Of Bounds Read

Known data

Function pointer

0x00	0x00	0x00	0x00	0x10	0x1a	0x88	0x7b
------	------	------	------	------	------	------	------

Out Of Bounds Read

Summary ASLR bypass oracle

- We used the value of an OOB Read to change the logic flow of the target to side-channel the value that was read


```
value = read_oob_trigger(attacker_index)
if logic(value):
    # enter logic path with output 1
    path_one(value)
elif other_logic(value):
    # enter logic path output 2
    path_two(value)
else:
    # error path can also be used to sidechannel value
    error_out(value)
```

Summary ASLR bypass oracle

- We used the value of an OOB read to change the logic flow of the target to side-channel the value that was read
- By aligning the OOB read offset with known or attacker-controlled data, we reduced the number of required attempts to ~1000. Bytes are brute forced individually. This yielded in more control over the logic flow we manipulated

Putting it all together

- 1) Leak ASLR with aforementioned Oracle
- 2) Prepare ROP chain and embed it into a nested archive to achieve a desired heap layout
- 3) Trigger buffer overflow and overwrite function pointer on heap

```
struct rtf_state {  
    rtf_callback_begin cb_begin; /* must be  
    rtf_callback_process cb_process;   
    rtf_callback_end cb_end;  
    void* cb_data; /* data set up by cb_be  
    size_t default_elements;
```

Real-World case study

The Target: Zimbra

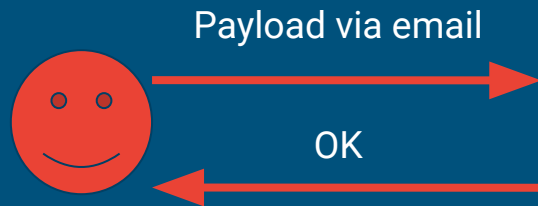
- Open-Source, Enterprise-Ready Email Suite
- Comes with ClamAV installed. Every incoming and outgoing email is scanned. No sandboxing is deployed
- Commonly used by governments
- Mass exploitation of CVE-2022-27925 against government, military and billion dollar corporations [1]
- Russian APT target NATO-aligned government servers to access military and diplomat's emails using CVE-2022-27926 [2]

[1] <https://www.volexity.com/blog/2022/08/10/mass-exploitation-of-unauthenticated-zimbra-rce-cve-2022-27925/>

[2] <https://www.bleepingcomputer.com/news/security/cisa-warns-of-zimbra-bug-exploited-in-attacks-against-nato-countries/>

Why the oracle won't work in an email context

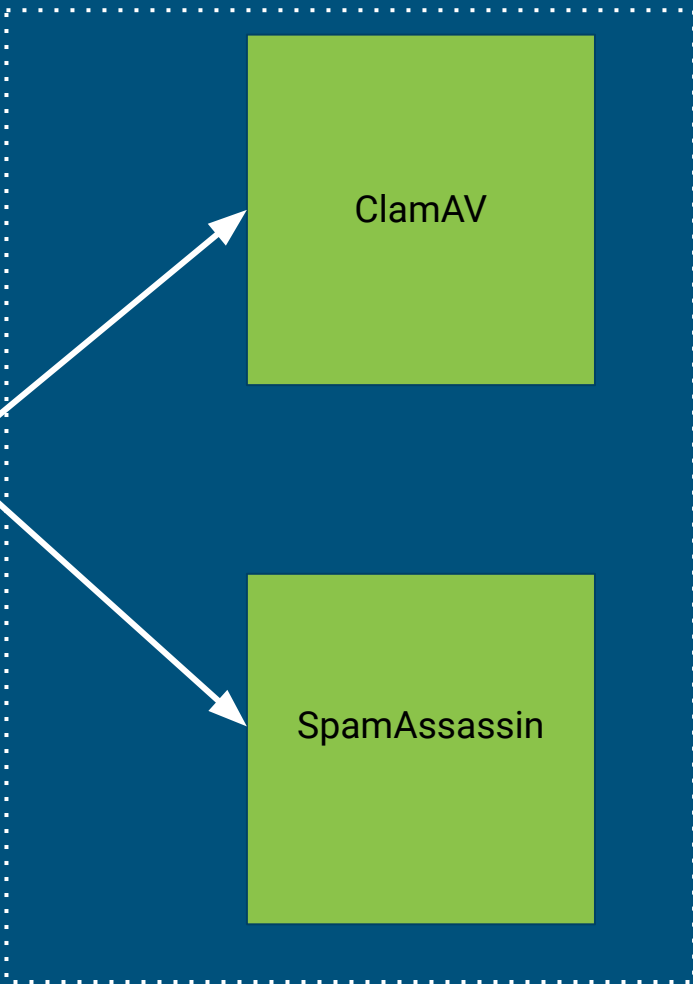
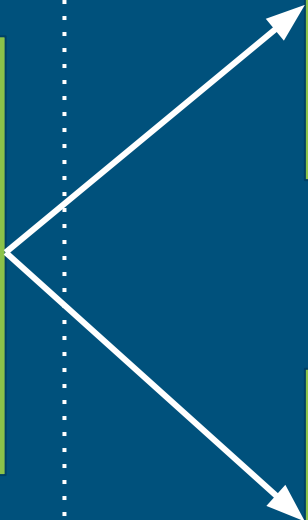
- So far we have been assuming a setup where an attacker will be notified of the scan result (virus or no virus)
- This is not usually the case for email servers as an external attacker



Attacker



Zimbra SMTP Server



ClamAV

SpamAssassin

Why the oracle won't work in an email context

- The SMTP server simply receives the email and responds to the client, then queues the email for virus scanning and spam checking
- Timing based attacks impossible as queue is used
- We need another side-channel. Maybe we can exploit the fact that virus scanning comes before spam-checking?

Side-channel via Email Spoofing Validation

- When an attacker sends an email to an SMTP server, they can simply claim to be "trusted@example.com"
- To validate this claim, the email server makes a SPF DNS request to the responsible DNS server for **example.com**. The DNS server responds with a list of IPs that are allowed to send emails for this domain

The Collaborator server received a DNS lookup of type TXT for the domain name **_dmarc.g4lkd17pvol62voq1kfjif19sfl3arz.oastify.com**.

The lookup was received from IP address **172.217.2.100** at 2023-May-25 13:32:22.818 UTC.

Side-channel via Email Spoofing Validation

In Zimbra's configuration, the email server bails if a virus was detected in the email. No further spam checks are made. This means:

- An attacker can host a DNS server
- Generate a unique subdomain per email that is sent to the target instance
- When no virus is found (the ASLR oracle did not succeed), a DNS request is made
- If a virus is found, no DNS request is made. An attacker can side-channel the oracle success through this behaviour


```
simonscannell@simonscannell:~$ ./clamav-zimbra-exploit target.com
```

X

Conclusions

- Just because the result of an OOB Read is not reflected, it can be side-channeled by observing how the application behaves based on the value
- A similar approach can be applied to scenarios where you target a load-balanced service and need to ensure that your payload is triggered against the worker for which ASLR has been defeated

Questions?

Feel free to reach out:

Twitter: @scannell_simon

Email: simonscannell at google com