UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science

Review of CVE-2016-6903 vulnerability

Kristjan Vedler

Supervisor: Meelis Roos

Tartu 2017

Introduction

Vulnerability CVE-2016-6903 is a bypass of the limited shell privileges in a semi-popular Python shell application for UNIX systems [1], giving the attacker the possibility of breaking out of the limited shell to execute arbitrary commands that the user is not supposed to have access to. The National Institute of Standards and Technology (NIST) has given it a CVSS v3.0 rating of 9.9 (critical), as this attack vector could be used to take complete control of the target computer through executing any arbitrary commands and for also having a low access complexity [2].

Impact

NIST has given a summary of possible usages for this vulnerability [2]:

- Allows unauthorized disclosure of information
- Allows unauthorized modification
- Allows disruption of service

As this vulnerability gives full access to execute any arbitrary commands, the possible usages for this vulnerability are in a very large scope. The most notable ones, as listed by NIST here, are the most likely usages for attackers attempting to do any nefarious activities. The damage caused by this vulnerability is hard to judge, as the usage of the whole application is not well document and it is not clear how long the vulnerability has been available or even viable.

Exploit

The exploit itself is not very complicated and is akin to the most classical SQL injection vulnerabilities in style. Simple combination of characters and commands gives the user full access to the target computer, given that the attacker already has access to this application. Although the vulnerability is network exploitable, the attacker still needs SSH or other type of authorization first to even access the lshell application through command-line.

The exploit itself was reported together with another similar vulnerability (CVE-2016-6902 [3]), coming from the same part of faulty code later seen in the fix commits. As such, it would be reasonable to look at both of them at the same time being that they are both almost exactly the same exploits.

The two ways to achieve shell outbreak were through bad syntax parsing and bad multiline parsing, with them both being different enough to warrant their own separate CVE entries. As mentioned in the initial CVE request, two methodologies were already mentioned for exploiting this attack, but the bug itself could introduce the need for other CVE IDs for other vulnerabilities possible in the same scope [4]. It was also mentioned, that the fix for this bug might solve other reported issues in their GitHub issue tracker. [5]

The vulnerability itself works by entering an allowed or authorized command, like *'echo'*, followed by a keyboard shortcut sequence or a string of characters that would allow the second command to be executed, such as the *'&&'* shell operator or in case of the former methodology, keyboard combination of CTRL+V followed by CTRL+J [5][6].

For example, the two issues reported by GitHub user Snawoot show a sequence of operations as follows [5][6]:

1)

```
vladislav@dt1:~$ su - testuser
Password:
You are in a limited shell.
Type '?' or 'help' to get the list of allowed commands
testuser:~$ ?
cd  clear  echo  exit  help  history  ll  lpath  ls  lsudo
testuser:~$ echo'/1.sh'
testuser@dt1:~$ cat echo/1.sh
#!/bin/bash

/bin/bash
testuser@dt1:~$
```

2)

```
vladislav@dt1:~$ getent passwd testuser
testuser:x:1001:1002:,,,:/home/testuser:/usr/bin/lshell
vladislav@dt1:~$ su - testuser
Password:
You are in a limited shell.
Type '?' or 'help' to get the list of allowed commands
testuser:~$ ?
cd  clear  echo  exit  help  history  ll  lpath  ls  lsudo
testuser:~$ bash
*** forbidden command: bash
testuser:~$ echo<CTRL+V><CTRL+J>
bash

testuser@dt1:~$ which bash
/bin/bash echo || 'bash'
```

And another methodology as reported by the GitHub user Oloremo and confirmed by other users [7]:

```
echo && 'bash'
```

Patch and reasons for the vulnerability

Being a very simplistic character or command escaping problem, the fix itself was only not very complicated. The two commits that fixed both of the CVE-s were `a686f71` and `e72dfcd` [8][9]. A common line splitting technique was used to parse the commands given by the user when the application did its security checks. However, it turned out that the sequence in which this was implemented was vulnerable to escaping out of it by passing it a newline character by first telling the terminal to insert the next character literally ("verbatim insert") with CTRL-V followed by the newline command CTRL-J [10][11]. This meant, that the `line = " ".join(line.split())` line of code only checked the command that was before the newline operator, meaning if you passed a valid command like `echo` to it and then split the line, the security check was never going to look at anything past the newline character, meaning that full bash access was available with a command sequence of two commands. Using `line.strip()` however meant that the newline characters and anything after that would still be parsed. Another security check was made in the former of the two commits listed here, where the security checker would specifically check for these control characters in the command and warn the user if they were present.

In parallel to this, the security parser also ignored checking any quoted strings for possible commands, which meant that another, very simple vulnerability exploited the security checker not checking for commands inside quotes, which was also basically just removed from the code, eliminating the issue.

Several unit-tests and functional tests were also added to make sure these exact vulnerabilities would not reappear in the future.

Conclusion

Unbeknownst to many junior developers, big security holes might not even come from very complicated faults in code. Some very quick and simple string parsing, as shown here for example, that has been ingrained in developers' minds from when they started learning software development might make them more likely to introduce such security flaws in their code long after they've completed their studies, regardless of their experience.

Such security flaws, however, could pose a serious threat if used in any large scale production environment. As such, the maturity of the application should be one of the key points in the consideration of using said software, by performing security audits or with just time, as in open-source applications like this, these kinks do get ironed out gradually over time if no actual audits are performed. However even if the system relying on pieces of software like this have a reliable way of updating it with new patches, a small time window would still remain present for any attackers to perform nefarious activities through these vulnerabilities.

Used resources

[1] https://github.com/ghantoos/lshell

[2] https://nvd.nist.gov/vuln/detail/CVE-2016-6903

[3] http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-6902

[4] http://www.openwall.com/lists/oss-security/2016/08/22/17

[5] https://github.com/ghantoos/lshell/issues/149

[6] https://github.com/ghantoos/lshell/issues/148

[7] https://github.com/ghantoos/lshell/issues/147

[8] https://github.com/ghantoos/lshell/commit/a686f71732a3d0f16df52ef46ab8a49ee0083c68

[9] https://github.com/ghantoos/lshell/commit/e72dfcd1f258193f9aaea3591ecbdaed207661a0

[10] https://superuser.com/a/421468

[11] https://ss64.com/bash/syntax-keyboard.html