COMP SCI 7412 Secure Software Engineering Assignment 2

Tinson Lai
a1812422

1 Part 2

1.1 meetecho/janus-gateway

Repository: meetecho/janus-gateway
Commit: 3f41a662f4fd4f00b90c307678211fd94f90e103
Stars: 4.6k
Contributors: 194
File: rtp.c
Language: C
CVE ID: CVE-2020-14034
CWE ID: 120 (Classic Buffer Overflow => Code Injection + (potentially) Arc Injection)
Lines: 94
Fix: dacb4edfad8e77f73b64d8c175cca0a7796ebf80
Comments: This is a typical example of misusing `sscanf`. A malicious attack can be launched by crafting a long extension (over 99 characters in this circumstance) in the request. This will also provide a better environment for arc injection attack based on code injection since the succeeding address for overflow is predictable and fixed (local stack variable). It will be easier to inject malicious flow controls. The fix imposed later has solved this problem by simply restricting the length to a constant number, it can then guarantee it will not write to memory region out of the allocated space. This is also the most common way to avoid buffer overflow.

1.2 netdata/netdata

Repository: netdata/netdata
Commit: a35bd00fbf8f2578ec0c660b532e925911191b43
Stars: 49k
Contributors: 411
File: web/api/web_api_v1.c
Language: C
CVE ID: CVE-2018-18836
CWE ID: CWE-94 (Code Injection)
Lines: 380-382
Fix: 92327c9ec211bd1616315abcb255861b130b97ca
Comments: A malicious client can launch an attack to update values within the database. This can be achieved by adding commas or some special characters inside the request which will change the JSON string to a different unexpected content. In its fixing commit, the contributor add an extra

sttep of filtering out special characters so that only alphanumeric characters will be kept inside the received parameters, and actually, neutralisation in the fixing commit is the most common way to prevent this kind of flaw.

1.3 LibVNC/libvncserver

Repository: LibVNC/libvncserver
Commit: 4616de68e664a8136b16c3003d33ba37d0438df3
Stars: 655
Contributors: 71
File: libvncserver/rre.c
Language: C
CVE ID: CVE-2020-14404
CWE ID: 119 (Improper Restriction of Operations within the Bounds of a Memory Buffer)
Comments: This is actually a minor bug since the bound checking should always be inserted before the operation but no after the operation. This bug appears to be an accidental mistake but not a conceptual error from my perspective. Anyway, the fixing commit has corrected the order of operations in the condition. This will only affect the integrity of the software since the bug will potentially crash the program.

1.4 Summary

Honestly, I've been using C++ (so as C) for around ten years. Careful handling of low-level system details is always the focused topic. Although most of the C++ language features has enhanced the protection from the language design level, considering the interoperability with C and Assembly, it is still more vulnerable than other interpreter-based or VM-based languages. It is really a basic manner to (correctly) add bound checking to most of the uncertain operations to these two languages, or otherwise, segmentation fault is usually the best circumstance it could happen. What even worse is data breaches or unauthorised executions.

2 Part 3

2.1 ex1

2.1.1 Original

```
fc hoh,

fow ct grgxqdfcsz zocsz?

nxg qoa ixgg dfct ixcjnq sczfd?
wg nxg pbnsscsz do fnrg n pnxdq nd kfnxbcg't pbnkg.
dfg cjgn ct do hxcsz wfnd qoa wnsd do gnd nsj znlgt dfnd qoa wnsd do pbnq.
nsj fnrg n bod oi ias dozgdfgx!
iggb ixgg do csrcdg loxg pgopbg.

fopg do tgg qoa dfgxg!
```

```
kfggxt,
nbckg
```

2.1.2 Decrypted

```
hi bob,

how is everything going?

are you free this friday night?
we are planning to have a party at charlie's place.
the idea is to bring what you want to eat and games that you want to play.
and have a lot of fun together!
feel free to invite more people.

hope to see you there!

cheers,
alice
```

2.1.3 Steps

It involves multiple guesses. First, this is obviously a letter, thus we can infer that

- `fc` is `hi`
- `kfggxt` is `cheers`

Based on the first inference, we can further deduce that

- `fow` is `how`
- `nxg` is `are`
- `qoa` is `you`
- `ixgg` is `free`
- `dfct` is `this`
- `ixcjnq` is `friday`
- `sczfd` is `night`

Eventually, we can solve the rest of the unknown characters

- `hoh` is `bob`
- `grgxqdfcsz` is `everything`

- `pbnsscsz` is `planning`
- `loxg` is `more`

Then all used characters and words are decrypted. There are some characters were not presented in the text, so the eventual ciphertext key may have numerous possibilities. The replacement is done by a script:

```
#!/usr/bin/env python3

mapping = {
    e: d
    for enc, dec in (
        ("fc", "hi"),
        ("kfggxt", "cheers"),
        ("fow", "how"),
        ("nxg", "are"),
        ("qoa", "you"),
        ("ixgg", "free"),
        ("dfct", "this"),
        ("ixcjnq", "friday"),
        ("sczfd", "night"),
        ("hoh", "bob"),
        ("grgxqdfcsz", "everything"),
        ("pbnsscsz", "planning"),
        ("loxg", "more")
    )
    for e, d in zip(enc, dec)
}

with open("ex1.enc", "r") as rf:
    with open("ex1.dec", "w") as wf:
        wf.write("".join(mapping.get(e, e) for e in rf.read()))
```

2.2 ex2

This part is a bit tricky. Based on the hint, I used a script to enumerate all possible weekday name and the encrypted word with matching lengths:

```
#!/usr/bin/env python3

from itertools import product

def diff(cs):
    return (ord(cs[0]) - ord(cs[1])) % 26

def run(s1, s2):
    re = list(map(diff, zip(s1, s2)))
```

```
    conv = "".join(chr(ord("a") + d) for d in re)
    print(f"{s1}:{s2} {re} {conv}")

lists = [("vgfjicf", "tuesday")] + list(
    product(["evfsgm", "qqzvea"], ["monday", "friday", "sunday"])
)
for s1, s2 in lists:
    run(s1, s2)
```

The result is:

```
vgfjicf:tuesday [2, 12, 1, 17, 5, 2, 7] cmbrfch
evfsgm:monday [18, 7, 18, 15, 6, 14] shspgo
evfsgm:friday [25, 4, 23, 15, 6, 14] zexpgo
evfsgm:sunday [12, 1, 18, 15, 6, 14] mbspgo
qqzvea:monday [4, 2, 12, 18, 4, 2] ecmsec
qqzvea:friday [11, 25, 17, 18, 4, 2] lzrsec
qqzvea:sunday [24, 22, 12, 18, 4, 2] ywmsec
```

So the key should be `ecms`. By writing another simple script:

```
#!/usr/bin/env python3

from itertools import cycle

enc = "evfsgm aj vgfjicf oekf mrvud qqzvea"
key = cycle(map(ord, "ecms"))

print(
    " ".join(
        "".join(
            chr((ord(v) - k) % 26 + ord('a'))
            for v, k in zip(word, key)
        )
        for word in enc.split()
    )
)
```

It is obvious that the original message is:

```
attack or retreat wait until monday
```

2.3 ex3

According to the hint, the cracking can be done by (run for around 2 seconds on my desktop):

```
#!/usr/bin/env python3
```

```
from itertools import product
from multiprocessing import Pool
from string import ascii_letters

from Crypto.PublicKey import RSA

with open("rsa_key.pub", "rb") as f:
    key = RSA.importKey(f.read(), 'PEM');

with open("ex3.enc", "rb") as f:
    encrypted = f.read()

def e(s):
    s = "".join(s)
    return s, key.encrypt(s.encode(), 0)[0]

def f(s):
    return encrypted == s[1]

if __name__ == "__main__":
    with Pool() as pool:
        ae = pool.map(e, product(ascii_letters, repeat=3))
    print([s[0] for s in filter(f, ae)])
```

And the plain text is `sun`.

3 Part 4

As a Chinese, I think the following IMEs will be more or less familiar to us:

- Baidu Pinyin (百度输入法)
- Sogou Pinyin (搜狗输入法)
- QQ Pinyin from Tencent (QQ 拼音)
- etc.

And actually, for faster typing, some may also learn to use Wubi (五笔, namely, five-strokes, a shape-based input method), but I just hate this since it needs so much memorisation. I am using none of these third-party IMEs. In fact, I have three types of devices in use:

- Apple iOS
- Microsoft Windows
- Google Android

I also used to have Ubuntu with KDE Plasma installed on my desktop, but I deleted that several months ago since its performance doesn't differ too much from WSL (Windows Subsystem for Linux). As far as I know, there is a third-party IME for Linux called `fcitx` (小企鹅输入法) and its newer version `fcitx-5`. I haven't installed any third-party IMEs for around 6 years since Microsoft has incorporated and enhanced their own IME starting from Windows 8. Currently, what I am using is

- Microsoft Pinyin for Windows
- Google Keyboard (Pinyin) for Android
- Apple Pinyin for iOS & macOS
- `fcitx` as mentioned.

Thus I don't have any third-party privacy concerns. However, a potential privacy concern here is that the built-in IME will also share its data to the other departments within the same corporation, for instance, search engines like Google or Bing. Apple doesn't participate in the search engines competition, but it uses these data to improve suggestions made by Siri as Apple claimed. Apple has been criticised for their poor privacy protection with respect to data on Siri, and the most recent scandal occurred in 2019[1]. Google has a data breach scandal in 2018, and it has been criticised on covering up the discovery of the bug which causes the data leakage[2]. Generally, however, most of the privacy concern is more relative to Android system, as pointed out in the given reference[3].

3.1 `fcitx`

The IME, `fcitx`, is an open-source project[4] initially created by a Chinese C++ programmer. It is only an IME engine in lieu of the native `ibus` available on most of the Linux distributions. Basically, it has three different add-ons for the actual typing,

- `fcitx-cloudpinyin` uses the Baidu Pinyin
- `fcitx-sunpinyin` uses the open-source statistical language model `sunpinyin`[5]
- `fcitx-googlepinyin` uses the cloud-based Google Pinyin service[6]

Since the development progress of Pinyin on Linux is very slow, none of these has the ability to perform cloud synchronisation.

[1] `https://www.forbes.com/sites/jeanbaptiste/2019/07/30/confirmed-apple-caught-in-siri-privacy-scandal-let-contractors-listen-to-private-voice-recordings/#4034d3f57314`

[2] `https://www.wsj.com/articles/google-exposed-user-data-feared-repercussions-of-disclosing-to-public-1539017194`

[3] `https://doi.org/10.1145/2666356.2594299`

[4] `https://github.com/fcitx`

[5] `https://github.com/sunpinyin/sunpinyin`

[6] `https://www.google.com/inputtools/services/features/input-method.html`

3.2 Microsoft/Google/Apple Pinyin

A general concern amongst these big companies is that they will use data recorded from their own IME input methods to improve their services, and they will also supply these data to other components within the ecosystem they built. They also try to fetch user's personal data from both the running devices and the associated account, such as contacts or emails, to enhance the user experience of the input method. They will also stored custom phrases input by the user into a dictionary, which will potentially be synchronised to the cloud so that it can be accessed across devices. All of these data collected by each company will form a unique identity of the user in their own ecosystem. A data breach will, therefore, easily expose these data.

Another potential way of leaking data is that, due to the evolution of modern Machine Learning, most of the phrases will not only be recorded but also fed to their machine learning model for phrase candidates ranking. This indeed encourages the development of the modern smart Pinyin, which greatly improves the user experience. The negative effect is that it will also expose certain characteristics of the user.

Let's pretend a user called San Zhang (张三). He may try to shorten the typing to simply 'z' then 's' when he tries to type his name because of the support of smart (ML-based) fuzzy Pinyin. On my desktop (Windows), it will be something like:

Since I just typed this word, it is even now ranked higher than most of the other common words. If the user itself type this shorthand will usually get his name as the first choice amongst the candidates. Similarly, some specific terminologies within some specific fields (such as Computer Science) which less frequently appears in daily life will potentially reflect the characteristics such as jobs etc. of the user.