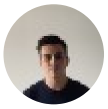
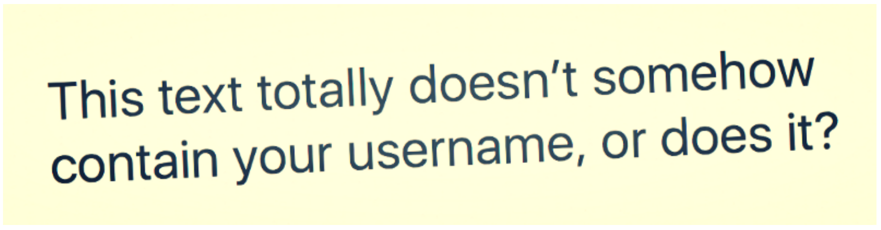
Be careful what you copy: Invisibly inserting usernames into text with Zero-Width Characters



Tom Ross Follow
Apr 3, 2018 · 4 min read

Don't want to read? Try the demo

Zero-width characters are invisible, 'non-printing' characters that are not displayed by the majority of applications. For example, I've inserted 10 zero-width spaces into this sentence, can you tell? (Hint: paste the sentence into Diff Checker to see the locations of the characters!). These characters can be used to 'fingerprint' text for certain users.



This text totally doesn't somehow contain your username, or does it?

It certainly can do, and you would have no idea

Why?

Well, the original reason isn't too exciting. A few years ago I was a member of a team that participated in competitive tournaments across a variety of video games. This team had a private message board, used to post important announcements amongst other things. Eventually these announcements would appear elsewhere on the web, posted to mock the team and more significantly; ensuring the message board was redundant for sharing confidential information and tactics.

The security of the site seemed pretty tight so the theory was that a logged-in user was simply copying the announcement and posting it elsewhere. I created a script that allowed the team to invisibly fingerprint each announcement with the username of the user it is being displayed to.

I saw a lot of interest in zero-width characters from a recent post by Zach Aysan so I thought I'd publish this method here along with an interactive demo to share with everyone. The code examples have been updated to use modern JavaScript but the overall logic is the same.

How?

The exact steps and logic are detailed below but to put it simply; the username string would be converted into its binary form and then the binary would be converted into a series of zero-width characters representing each binary digit. The zero-width string could then be invisibly inserted into the text. If said text was posted elsewhere the zero-width character string could be extracted and the process reversed to figure out the username of the person who copied it!

Fingerprinting the Text:

1: Taking the logged-in users' username and converting it to binary
Here we are just converting each letter of the username into its binary equivalent.

```
const zeroPad = num => ‘00000000’.slice(String(num).length)
+ num;

const textToBinary = username => (
  username.split('').map(char =>
    zeroPad(char.charCodeAt(0).toString(2))).join(' ')
);
```

2: Taking the binary-converted username and further converting it into zero-width characters
This iterates through the binary string and converts each 1 into a zero-width space and each 0 into a zero-width non-joiner character. Once we have converted the letter we insert a zero-width joiner character before moving onto the next.

```
const binaryToZeroWidth = binary => (
  binary.split('').map((binaryNum) => {
    const num = parseInt(binaryNum, 10);
    if (num === 1) {
      return ''; // zero-width space
    } else if (num === 0) {
      return ''; // zero-width non-joiner
    }
    return ''; // zero-width joiner
  }).join('') // zero-width no-break space
);
```

3: Inserting the zero-width 'username' into the confidential text
This was just inserting the block of zero-width characters into the confidential text.

Extracting the username from the fingerprinted text

Reversing the logic.

1: Extract the zero-width 'username' from the confidential text
Removing the expected confidential text from the string, leaving only the zero-width characters.

2: Convert the zero-width 'username' back into binary
Here we split the string based on the zero-width no-break spaces we added earlier. This will give us the zero-width equivalent of the binary equivalent of each letter of the username! We iterate through the zero-width characters and return 1 or 0 to recreate the binary string. If we don't find a corresponding 1 or 0 we must have hit the zero-width joiner, and thus have completed the binary for a character; we can then append a single space to the string and move onto the next character.

```
const zeroWidthToBinary = string => (
  string.split('').map((char) => { // zero-width no-break
space
    if (char === '') { // zero-width space
      return '1';
    } else if (char === '') {  // zero-width non-joiner
      return '0';
    }
    return ' '; // add single space
  }).join('')
);
```

3: Convert the binary username back into text
Finally we parse the binary string and convert each series of 1's and 0's into its corresponding character.

```
const binaryToText = string => (
  string.split(' ').map(num =>
    String.fromCharCode(parseInt(num, 2))).join('')
);
```

Conclusion

Companies are doing more than ever to avoid information leakage and stop whistleblowers, this trick is just one of many that can be used. **Depending on your line of work, it could be vitally important to understand the risks associated with copying text.** Very little applications will try to render the zero-width characters. For example, you would hope your terminal would attempt to display them (mine doesn't!).

To go back to the message board scenario, the plan worked as expected. A new announcement was released soon after the script had been deployed. Within a few hours the text had been shared elsewhere with a zero-width string attached. The username of the culprit was correctly identified and they were banned; a successful project!

There are some caveats to this method of course. For example, if a user knew of the script they could theoretically insert their own zero-width characters and accuse someone else. A better solution would be to insert a unique user ID that is not publicly available instead of the username.

To have a play around check out the demo or view the source code

Want to discuss this some more? Get in touch on Twitter