String Obfuscation in the Hamweq IRC-bot

blag.nullteilerfrei.de/2020/05/31/string-obfuscation-in-the-hamweq-irc-bot/

born

In this blog post, we will follow one of herrcore's awesome videos and re-implement the automation as a Ghidra script in Java. The video in question is part of a series about a legacy malware family called Hamweq. CERT Polska published an extensive analysis of Hamweq: The malware implements a IRC-based botnet with worm-like capabilities. In this post we will solely focus on the string deobfuscation functionality in the malware. ## Identifying the String Deobfuscation Method Instantly after opening the sample `4eb33ce768def8f7db79ef935aabf1c712f78974237e96889e1be3ced0d7e619` in Ghidra, you can see four calls to `GetProcAddress` . This method resolves an API function dynamically, basically turning a string referencing a name of an API function into a pointer to the corresponding function. According to the documentation, the second argument to `GetProcAddress` is that string. Following the memory address in Ghidra (by double-clicking) does not lead to any printable strings though. Hence _before_ these four calls to `GetProcAddress` , these memory regions have to be modified during runtime. Otherwise, `GetProcAddress` would return the null pointer and calling that pointer, would crash the program. The only two functions that can do this deobufscation step are `FUN_00402781` and `FUN_004027e1` . The first of the two seems to be doing something related to privileges, but since we want to focus on string obfuscation right now, we will not waste any time reverse engineering it but take a look at the function `FUN_004027e1` . This function accepts one string argument which is hard-coded to be `I0L0v3Y0u0V1rUs` at this call. This is probably a reference to the famous ILOVEYOU virus from 2000 left by the malware author to our amusement. Because we are feeling lucky, let's rename `FUN_004027e1` to `pr_StringDeobfusaction` . ## Optimizing Crappy Crypto `pr_StringDeobfusaction` references the data at `0x00405020` and interprets it as an array of pointers to strings. Each of these strings is then deobfuscated with a custom Xor-algorithm using the passed argument as a key. The deobfuscation algorithm is called on each of the referenced strings separately: it first Xors each byte of the passed key onto each byte of the obfuscated data and then inverts every byte of the result. Since the Xor-operation is associative, the key can be reduced to a single-byte Xor-key: For simplicity's sake, let us assume, the Xor key is not `I0L0v3Y0u0V1rUs` but the sequence of numbers 23, 42 and 36. Now let x be a single byte to be deobfuscated and let $\otimes$ denote bit-wise Xor, then the following equation is true: \[((x \otimes 23) \otimes 42) \otimes 36 = x \otimes (23 \otimes 42 \otimes 36) = x \otimes 25 \] so instead of using the key 23, 42, 36 one could simply use the key 25. Similarly, the key `I0L0v3Y0u0V1rUs` can be reduced to 95. The following Java function implements this key-reduction:

```
private byte[] reduceKey(String key) {
    byte ret[] = new byte[1];
    for (byte b : key.getBytes()) {
        ret[0] ^= b;
    }
    return ret;
}
```

Scripting We now want to write a script where the user specifies the address of the array of pointers to the obfuscated strings and Ghidra should then deobfuscate them all, print the result, patch the data in memory, set the correct data-type and create bookmarks for all deobfuscated strings:

```
public void run() throws Exception {
    byte[] key = reduceKey("I0L0v3Y0u0V1rUs");
    Address stringTable = askAddress("Enter Address", "Specify address of string
table");
    while (true) {
        Address stringAddress = unpackAddressLE(getOriginalBytes(stringTable, 4));
        if (stringAddress.getOffset() == 0)
            break;
        byte data[] = getOriginalBytes(stringAddress, 0x40);
        if (data == null) {
            break;
        }
        byte cypherText[] = readUntilZeroByte(data);
        byte plainText[] = cryptXorAndInvert(cypherText, key);
        println(String.format("0x%08X %s", stringAddress.getOffset(), new
String(plainText)));

        setBytes(stringAddress, plainText);
        clearListing(stringAddress, stringAddress.add(plainText.length - 1));
        createData(stringAddress, new ArrayDataType(CharDataType.dataType,
plainText.length, 1));
        createBookmark(stringAddress, "DeobfuscatedString", new String(plainText));

        stringTable = toAddr(stringTable.getOffset() + 4);
    }
}
```

The only missing part now is the actual decryption routine:

```
private byte[] cryptXorAndInvert(byte[] data, byte[] key) {
    final byte[] ret = new byte[data.length];
    for (int k = 0; k < data.length; k++)
        ret[k] = (byte) (~(data[k] ^ key[k % key.length]));
    return ret;
}
```

As always, the complete script to deobfuscate strings from a Hamweq sample can be found on GitHub.