Smuggling HTA files in Internet Explorer/Edge

Published: 2017-08-08 · Archived: 2026-04-06 00:55:42 UTC

In this blog post, we will demonstrate how attackers can serve malicious HTML Application (HTA) [1] files in a way that may bypass traditional proxy filtering. We will also cover some defensive mechanisms that can be used to prevent such attacks.

Background

When carrying out Red Team engagements for our clients, we often attempt to gain code execution using malicious payloads, much in the same way that real-life attackers would.

With defence technologies becoming more advanced and vendors focusing heavily on mitigation, attackers are having to find increasingly novel techniques in order to execute malicious code on their targets.

Gone are the days of using malicious Java Applets and Flash exploits, with browsers now implementing click-to-play or, in some cases, completely removing support [2] for these often-abused technologies.

You can now block Office Macros via a group policy [3], closing an attack vector heavily relied upon by attackers in the past. Object Linking and Embedding (OLE) - another popular attack-vector which enables an attacker to embed executable content within an Office document, has become more prevalent in the last year or so, probably due (at least in part), to the increased focus on mitigating macro-based threats.

Soon, Microsoft will be raising the bar even higher by locking-down OLE in Office, by restricting the file-types that can be embedded within Office documents [4]. This will hopefully significantly reduce the attack surface in Office even further.

With all this in mind, it may or may not come as a surprise to hear that HTA files are still supported in Internet Explorer and Edge. This is a very old attack, yet something attackers have been abusing more in recent times.

Recently, the Hancitor malspam actor has been observed using HTA as part of their attack chain, which drops password-stealing malware. HTA files were also used as an exploitation vector for CVE-2017-0199, which was found to be exploited in the wild [5].

With all the focus on mitigating Office Macros and OLE embedding, you may be wondering why HTA files are still an issue. The answer is that although they've been around for a while, they are only now coming under the spotlight as a result of increased effort (from vendors and third parties alike) to block these more widely abused 'features'. As a result, attackers are simply adapting to find the path of least resistance and, at the moment, HTA files still work, which is great for attackers.

What is a HTA File?

A HTA file is usually made up of HTML and script such as JScript or VBScript, much like a normal web page.

However, the difference with HTA files is that they run in full trust mode, with access to features that a normal web page wouldn't have, such as ActiveX controls usually marked 'unsafe for scripting'.

This means that if an attacker was to serve a HTA file (for example, via a malicious webpage) and convince the user to click through two warnings, then the attacker could run malicious code on the victim's computer. All without needing an exploit or bypassing any of the latest and greatest mitigations.

Attack

Thinking from an attacker's perspective, something that can often get in the way of your payload making it all the way to the user's desktop are security products, such as content-inspecting web proxies and URL-scanning 'sandboxes'.

These will often keep an eye out for executable content - such as *.exe* files or scripts being downloaded via a user's browser - and block them. Some products carry out sandboxing, which means your content might be downloaded by the security product and executed in a virtual machine to see how it behaves and if it appears to be malicious.

Both of these things could present an issue for attackers trying to get their payload to a user.

With these issues in mind, we created Demiguise.

We were recently carrying out a Red Team engagement for a client who had implemented many of the recommended controls and also had a sandboxing/content-inspecting web proxy in place. We needed a way to serve our HTA file to the users without it being caught by the web proxy and blocked as executable content. Ideally, we did not want it to be run in the sandbox at all.

Demiguise works by creating an HTML file which contains an encrypted version of your HTA payload. This means that the content is served as a single HTTP request (with a content type of html/text) which the proxy will happily allow.

When the HTML is rendered in the user's browser, embedded JavaScript will unpack and decrypt the HTA content before calling msSaveBlob [6] which downloads the unpacked file directly from the user's browser.

The user will then be prompted to run the HTA twice and, if they accept the two prompts, the HTA file will run successfully; all while making only a single HTTP request for a non-executable mime-type.

Environmental keying

To improve the attack, and in an attempt to avoid sandboxing products, the tool supports the notion of 'environmental keying'.

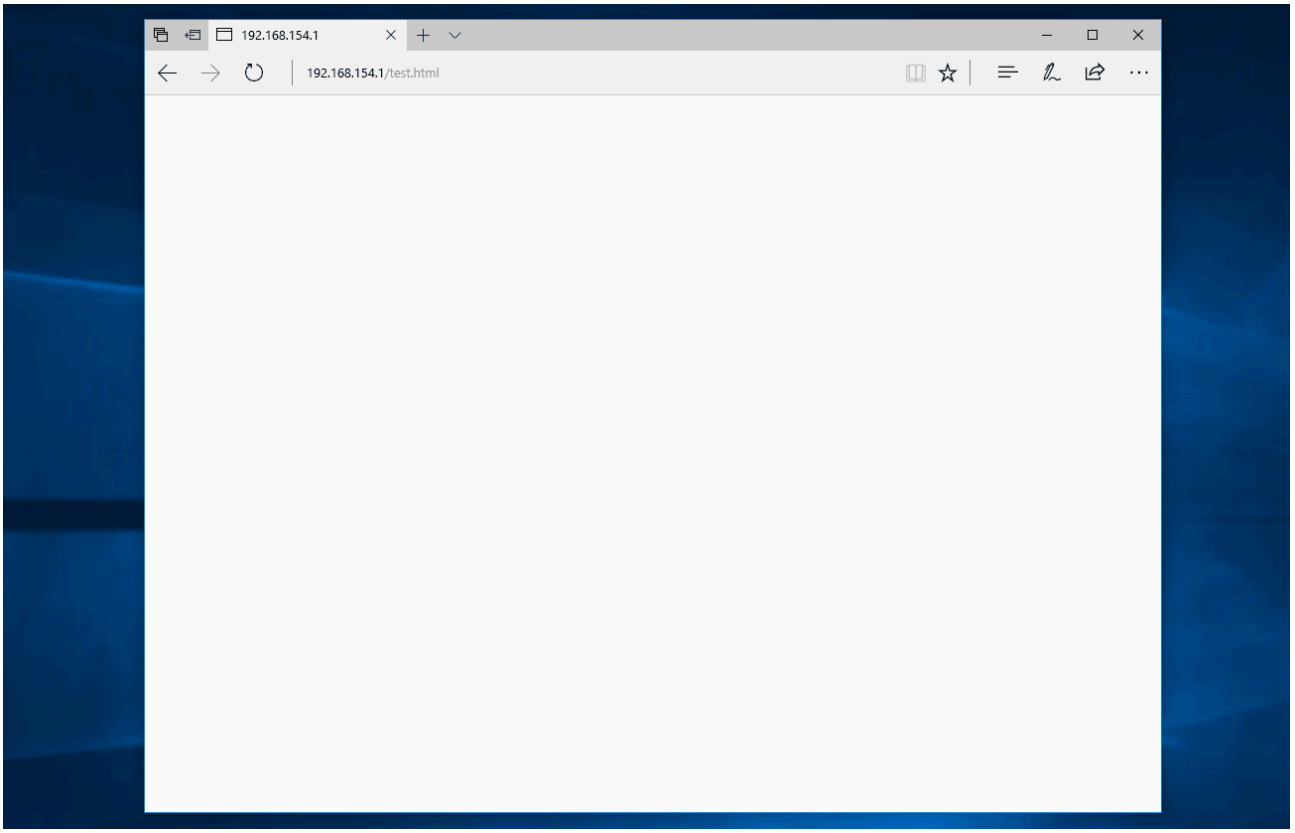
The idea with this concept is that rather than hard-coding the payload encryption key directly within the HTML source, we can instead derive the key from something within the user's environment. It should be something which can be determined from a user's machine via JavaScript that would not work elsewhere. It can be thought of like a signature, indicating that your JavaScript is executing on the intended target's browser, and nowhere else.

A good way to achieve this via JavaScript is to find something on the target network that can only be resolved within that network. For example, an image hosted on the intranet, or perhaps the client's external IP address.

As to how you may identify a good candidate for an environmental key, this is left as an exercise for the attacker. However, there are many good tools, such as BeEF [7] and WebFEET [8] which can be used to fingerprint a user's environment as part of a fingerprinting campaign before carrying out your main attack.

By deriving the encryption key from something that only exists in their network you can guarantee that your payload won't work elsewhere (like a sandbox). In fact, not only will it not work, but the sandbox will have no idea what the file is even supposed to be, as it will not decrypt correctly..

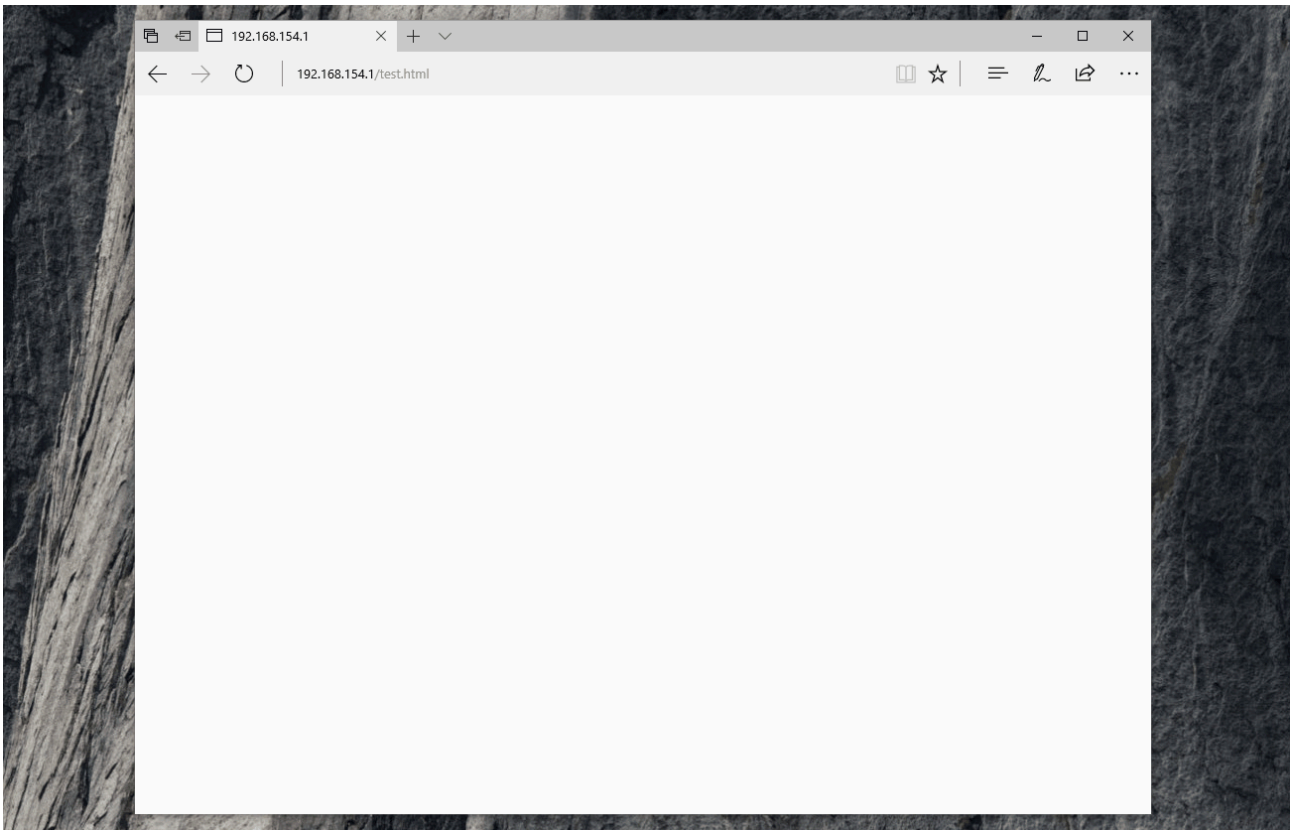
A full walkthrough of how the tool works, as well as an example of environmental keying, can be found on the tool's GitHub page [9]. This also includes a number of example videos which show the warnings that the user will be prompted with, along with examples of typical output from the tool.



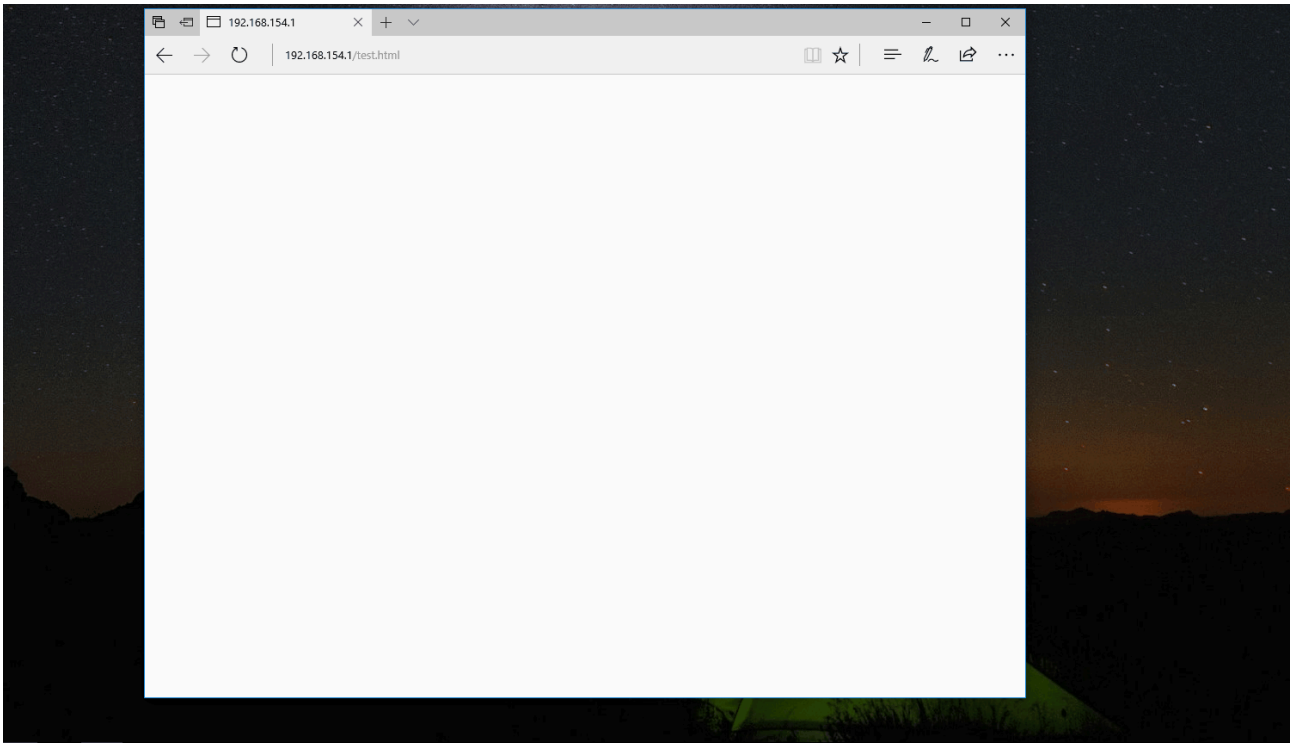
Defence

Due to an attacker's ability to obfuscate the calls to functions such as msSaveBlob, deriving signatures for attacks such as this may be difficult. A more holistic approach could be to block HTA files from executing at all.

This can be achieved through the use of Software Restriction Policies (SRP [10]), or Device Guard [11] (on Windows 10 and Server 2016), both of which can be configured to block *.hta* files from being executed.



Another slightly easier option may be to override the default file handler for *.hta* files so that they open with *notepad.exe* instead, thus rendering HTA files harmless.



Note: Using Applocker instead of SRP to block the use of *mshta.exe* did not appear to work successfully when testing in Windows 10. Applications such as Windows Update and Search bar appeared to not function correctly. This is believed to be due to its use as part of some internal Windows apps, therefore more research is suggested if you choose to use Applocker for blocking HTA files.

Conclusion

We hope that this blog post serves to demonstrate how, in spite of the great mitigations being implemented by vendors, some old attacks are still working well. Albeit with a few tweaks to suit the modern environment.

We have demonstrated the risk of HTA files and shown why the blocking of HTA files should be considered. Hopefully, the tool also helps fellow Red Teams to demonstrate that risk to organisations.

References

[1] https://msdn.microsoft.com/en-us/library/ms536471(VS.85).aspx

[2] https://blog.chromium.org/2017/07/so-long-and-thanks-for-all-flash.html?m=1

[3] https://blogs.technet.microsoft.com/mmpc/2016/03/22/new-feature-in-office-2016-can-block-macros-and-help-prevent-infection/

[4] https://twitter.com/enigma0x3/status/888443907595526144

[5] https://www.fireeye.com/blog/threat-research/2017/04/cve-2017-0199-hta-handler.html

[6] https://msdn.microsoft.com/en-us/library/hh779016(v=vs.85).aspx

[7] https://github.com/beefproject/beef

[8] https://github.com/nccgroup/WebFEET

[9] https://github.com/nccgroup/demiguise

[10] https://technet.microsoft.com/en-gb/library/bb457006.aspx

[11] https://docs.microsoft.com/en-us/sccm/protect/deploy-use/use-device-guard-with-configuration-manager

Published date: 08 August 2017

Written by: Richard Warren
