



IT-Security (ITS) B1

DIKU, E2021

Lecture plan

| | | | | |
|--------|--------|-------|-----|--|
| 36 | 06 Sep | 10-12 | TL | Introduction and security concepts |
| | 10 Sep | 10-12 | TL | Crypto |
| 37 | 13 Sep | 10-12 | CJ | User authentication and access control |
| | 17 Sep | 10-12 | CJ | Software and operating system security |
| → 38 | 20 Sep | 10-12 | TL | Vulnerabilities and exploits |
| | 24 Sep | 10-12 | CJ | Cloud and IoT security |
| 39 | 27 Sep | 10-12 | TL | Malware |
| | 01 Oct | 10-12 | CJ | Firewalls and denial-of-service attacks |
| 40 | 04 Oct | 10-12 | TL | Internet security protocols |
| | 08 Oct | 10-12 | TL | Intrusion detection |
| 41 | 11 Oct | 10-12 | TL | Incident response and forensics |
| | 15 Oct | 10-12 | CJ | IT security management and risk assessment |
| 42 | | | | Fall Vacation - No lectures |
| 43 | 25 Oct | 10-12 | CJ | Privacy |
| | 29 Oct | 10-12 | CJ | GDPR |
| 44 | 01 Nov | 10-11 | TL | Security in the industry and career paths |
| | | 11-12 | All | Q/A |



Today's agenda

Part 1: Some common and notable bugs

Part 2: Fuzzing – automatiing bug discovery



Definitions

Software contains **bugs**

A **vulnerability** is a bug that can be exploited by an attacker

Not all bugs can be exploited

Not all vulnerabilities matter the same

Vulnerabilities are exploited to run malware

This is a vulnerability

CVE-2021-38647

Unauthenticated RCE as root
(Severity: 9.8)

What is OMI?

Open Management Infrastructure (OMI) is an open source project sponsored by Microsoft in collaboration with The Open Group. Essentially, it's Windows Management Infrastructure (WMI) for UNIX/Linux systems.

The OMI agent runs as root with the highest privileges. Any user can communicate with it using a UNIX socket or via an HTTP API when configured to allow external access.





OMIGOD

The flow below illustrates the unexpected behavior of OMI when a command execution request is issued with no **Authorization header**.

Normal flow with valid password in the auth header—The OMICLI issues an HTTP request to the remote OMI instance, passing the login information in the Authorization header.

Exploit flow when passing a command without auth header—Surprise! The OMI server trusts the request even without an auth header and enables the perfect RCE: single-packet-to-rule-them-all.

Types of vulnerabilities

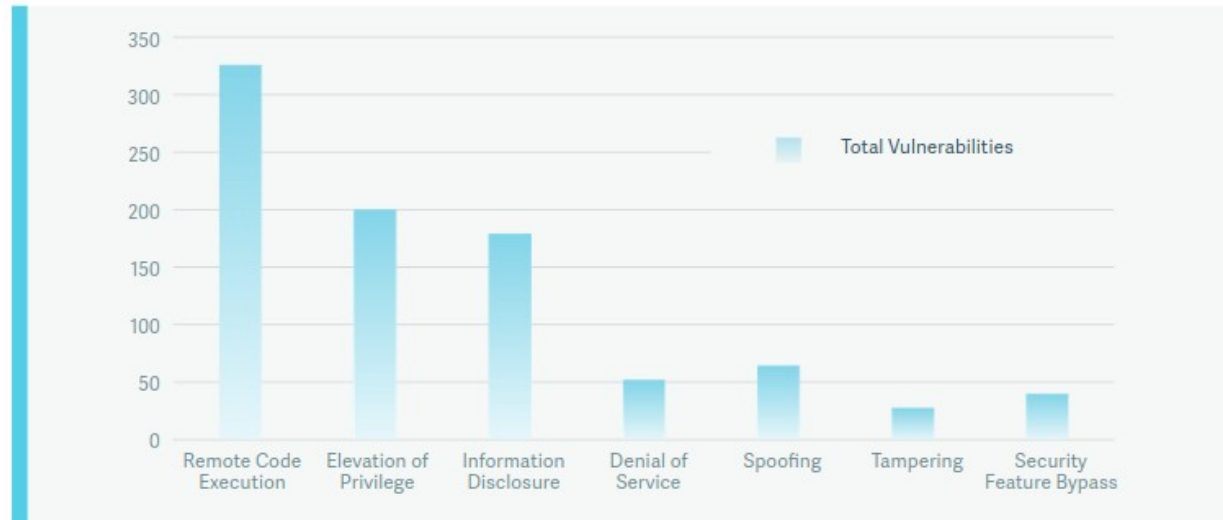


Figure 1: Breakdown of Microsoft Vulnerability Categories (2019)

Another vulnerability

BlueKeep

From Wikipedia, the free encyclopedia

Not to be confused with [BlueBEEP](#).

BlueKeep ([CVE-2019-0708](#)^[]) is a [security vulnerability](#) that was discovered in [Microsoft's Remote Desktop Protocol](#) implementation, which allows for the possibility of [remote code execution](#).

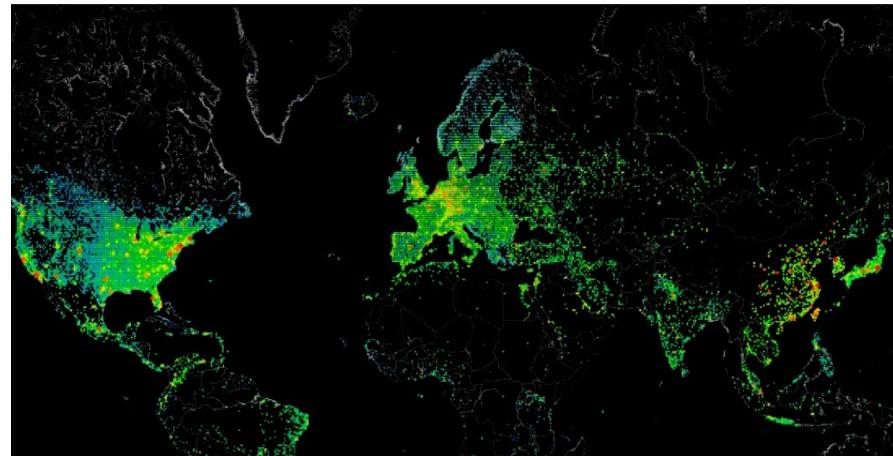
First reported in May 2019, it is present in all unpatched Windows NT-based versions of Microsoft Windows from [Windows 2000](#) through [Windows Server 2008 R2](#) and [Windows 7](#). Microsoft issued a security patch (including an out-of-band update for several versions of Windows that have reached their end-of-life, such as [Windows XP](#)) on 14 May 2019. On 13 August 2019, related Bluekeep security vulnerabilities, collectively named **DejaBlue**, were reported to affect newer Windows versions, including [Windows 7](#) and all recent versions up to [Windows 10](#) of the operating system, as well as the older Windows versions.^[3]

Contents ^[hide]

- [History](#)
- [Mechanism](#)
- [Mitigation](#)
- [See also](#)
- [References](#)
- [External links](#)

| BlueKeep | |
|--|--|
|  | |
| A logo created for the vulnerability, featuring a keep , a fortified tower built within castles. | |
| CVE identifier(s) | CVE-2019-0708 ^[] |
| Date patched | 14 May 2019; 3 months ago ^[1] |
| Discoverer | UK National Cyber Security Centre ^[2] |
| Affected software | pre- Windows 8 versions of Microsoft Windows |

Shodan and BlueKeep





Common Vulnerabilities Scoring System

Attack vector: Local vs. Remote

Attack complexity: High, Medium, Low

Authentication: Required vs. Not

Impact: C/I/A

CVSS v2.0 Severity and Metrics:

Base Score: 10.0 HIGH

Vector: (AV:N/AC:L/Au:N/C:C/I:C/A:C) (V2 legend)

Impact Subscore: 10.0

Exploitability Subscore: 10.0

Access Vector (AV): Network

Access Complexity (AC): Low

Authentication (AU): None

Confidentiality (C): Complete

Integrity (I): Complete

Availability (A): Complete

Additional Information:

Allows unauthorized disclosure of information

Allows unauthorized modification

Allows disruption of service



Another vulnerability

Vulnerability Summary for CVE-2010-2883

Original release date: 09/09/2010

Last revised: 08/04/2011

Source: US-CERT/NIST

Overview

Stack-based buffer overflow in CoolType.dll in Adobe Reader and Acrobat 9.x before 9.4, and 8.x before 8.2.5 on Windows and Mac OS X, allows remote attackers to execute arbitrary code or cause a denial of service (application crash) via a PDF document with a long field in a Smart INdependent Glyphlets (SING) table in a TTF font, as exploited in the wild in September 2010. NOTE: some of these details are obtained from third party information.

Impact

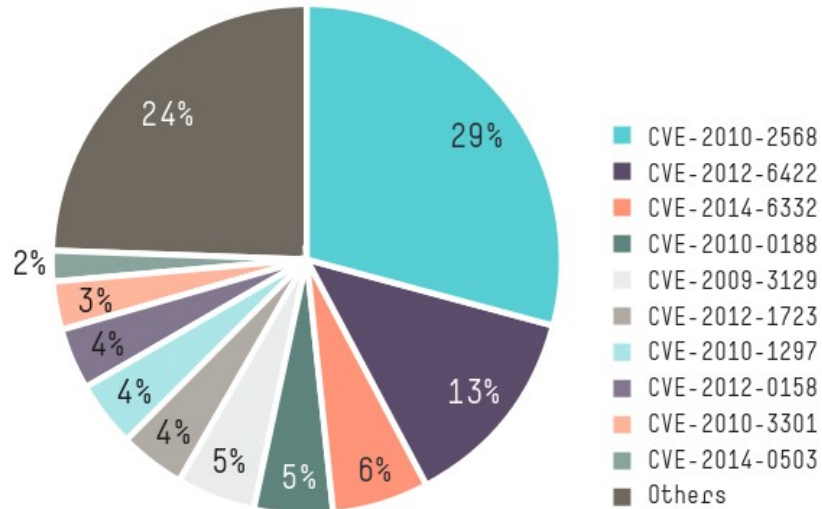
CVSS Severity (version 2.0):

CVSS v2 Base Score: 9.3 (HIGH) (AV:N/AC:M/Au:N/C:I/C/A:C) (legend)

Impact Subscore: 10.0

Exploitability Subscore: 8.6

Too old? Old exploits never die (2015)





Types of vulnerabilities, include:

Format string

Overflow

Over-read

Load order

Use-after-free

Dangling pointers

Code injection

Command injection

Race conditions

Typos, and more



Where's the bug?



```
#include <stdio.h>
```

```
int main () {  
    int i;  
    printf("Enter a value: ");  
    scanf("%d", &i);
```

```
    if (i < 0)  
        goto fail;  
    if (i > 100)  
        goto fail;  
    //goto fail;  
    if (i%2 == 0)  
        goto fail;
```

```
    return;
```


```
fail:  
    printf("Fail\n");  
    return;  
}
```

```
$ ./a.out  
Enter a value: 2  
Fail
```

```
$ ./a.out  
Enter a value: 3  
Fail
```

Apple iOS Goto Fail

```
1 static OSStatus
2 SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer signedParams,
3                                   uint8_t *signature, UInt16 signatureLen)
4 {
5     OSStatus      err;
6     ...
7
8     if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
9         goto fail;
10    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
11        goto fail;
12    if ((err = SSLHashSHA1.update(&hashCtx, &signature)) != 0)
13        goto fail;
14    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
15        goto fail;
16    ...
17 fail:
18    SSLFreeBuffer(&signedHashes);
19    SSLFreeBuffer(&hashCtx);
20    return err;
21 }
```

```
#include <stdio.h>
#include <string.h>
```

```
int main () {
```

```
    char buf[20] = "http://www.diku.dk";
    char shh[30] = "mumstheword";
    char out[64];
    int chars;
```

```
    printf("Buffer contents: %s\n", buf);
```

```
    printf("Chars to copy: ");
    scanf("%d", &chars);
```

```
    if (chars > sizeof(buf)) chars = sizeof(buf);
    memcpy(out, buf, chars);
```

```
    printf("Copied: ");
    fwrite(out, chars, 1, stdout);
    printf("\n");
```

```
$ ./a.out
```

```
Buffer contents: http://www.diku.dk
```

```
Chars to copy: 12
```

```
Copied: http://www.d
```

```
$ ./a.out
```

```
Buffer contents: http://www.diku.dk
```

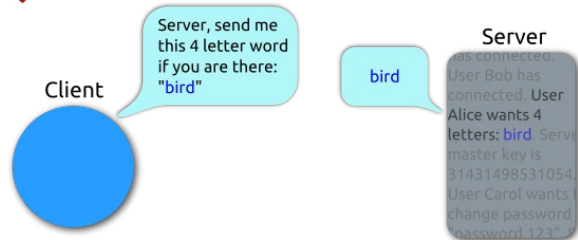
```
Chars to copy: 50
```

```
Copied: http://www.diku.dk0LHmumstheword
```

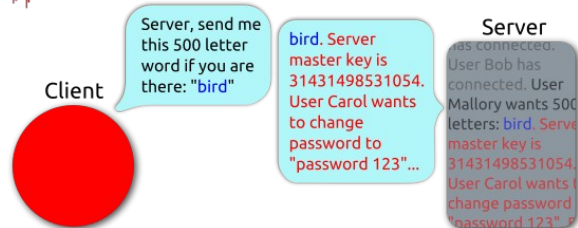
The HeartBleed Bug




Heartbeat – Normal usage



Heartbeat – Malicious usage





```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv)
{

    printf("Current time: ");
    fflush(stdout);
    system("/bin/date");
    return 0;

}
```

```
$ ./a.out
Current time: Fri Sep  6 09:30:47 CEST 2019
```

```
$ export PATH=`pwd`: $PATH
$ echo -e '#!/bin/sh\necho "Hello"' > date
$ chmod 700 date
```

```
$ ./a.out
Current time: Hello
```



Real-world example: PlugX

PlugX drops

- A legitimate NVIDIA file (NvSmart.exe)

- A malicious DLL (NvSmartMax.dll)

Normally, NvSmart.exe would load a legitimate NvSmartMax.dll

But if a (malicious) version the DLL file is located in the same directory, this will load instead



```
#!/usr/bin/perl
```

```
open(FH, "< ".$ARGV[0]); #force read open with '<'
```

```
while(<FH>)  
{  
    print $_;  
}
```

```
close(FH);
```

```
$ ./code.pl code.pl  
#!/usr/bin/perl
```

```
open(FH, $ARGV[0]);
```

```
while(<FH>)  
{  
    print $_;  
}
```

```
close(FH);
```


```
$ ./code.pl 'ls -l code.pl'  
-rwx----- 1 user user 79 Sep  1 10:45 code.pl
```



Explanation

According to the Perl documentation

If filename ends with a "|", filename is interpreted as a command which pipes output



```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char **argv)
{
    char buffer[64];
    strncpy(buffer, argv[1], sizeof(buffer));


    printf("You entered: ");
    printf("%s", buffer);
    printf("\n");
}
```

```
$ ./a.out A
You entered: A
```

```
$ ./a.out %s
You entered: You entered:
```

```
$ ./a.out %x
You entered: 510a2000
```

```
$ ./a.out %x%x
You entered: 437a00041569e0
```



```
#include <string.h>
```

```
void foo (char *bar)
{
    char c[12];
    strncpy(c, bar, sizeof(c));
}
```

```
int main (int argc, char **argv)
{
    foo(argv[1]);
}
```

```
$ ./6.out A
```

```
$ ./6.out AAAAAAAAAAAAAAA
```

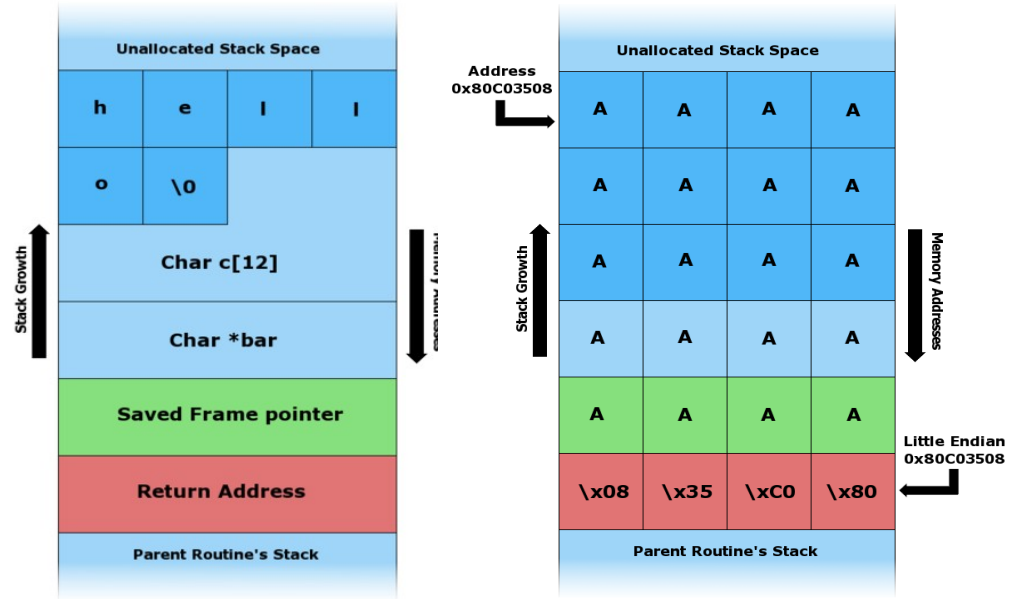
```
$ ./6.out AAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Segmentation fault
```



```
#include <string.h>
```

```
void foo (char *bar)
{
    char c[12];
    strcpy(c, bar);
}
```

```
int main (int argc, char **argv)
{
    foo(argv[1]);
}
```





Some countermeasures

Stack canaries

Check stack not altered when function returns

Data execution prevention (DEP)

Prevent the execution of data on the stack or heap

Address space layout randomization (ASLR)

Rearrange memory positions to make successful exploitation more difficult