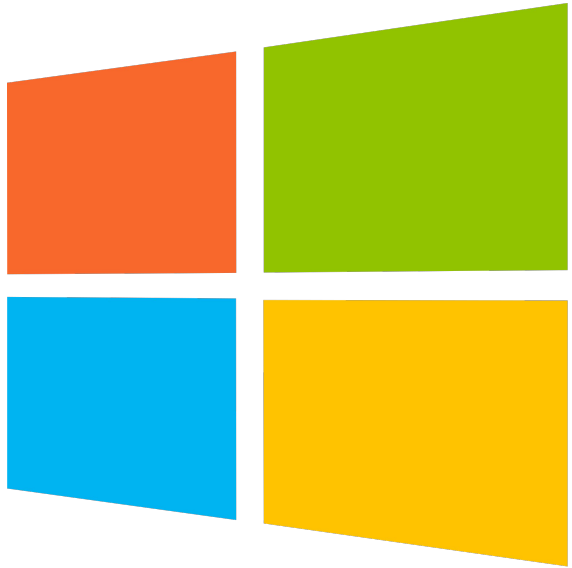
Windows Research Kernel

Niklas Schilli

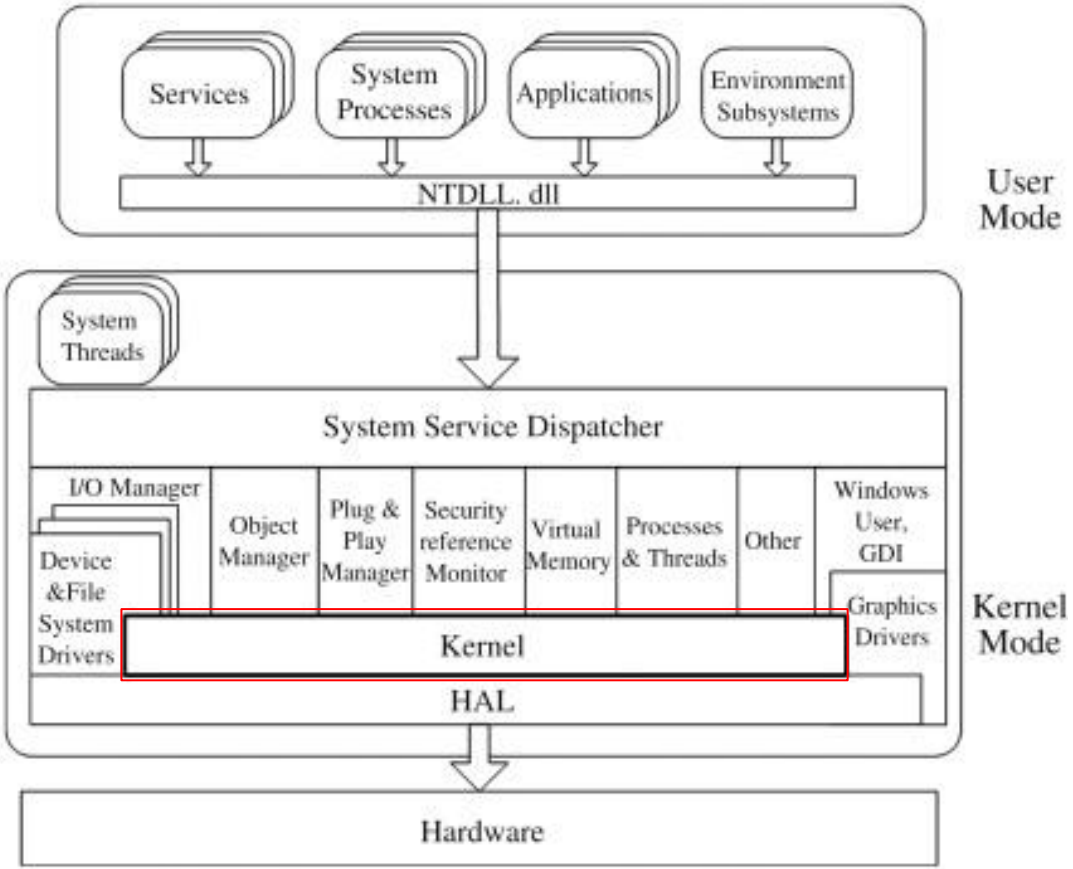


Überblick

- Aufbau des WRK
- Toolgestützte analyse
- CVE Mapping
- CVE Statistics
- Fuzzing

WRK

- Windows Kernel auf dem Stand von XP SP2/Windows Server 2003
- Größtenteils C Code, etwa 7 % Assembly
- Prebuilt Binaries für viele Komponenten wie z.B. HAL
- Über 800k LOC aus ca. 50.000k LOC für das gesamte OS
- Einige fehlende Komponenten wie Power Management, Driver Verifier, Branding
- Beinhaltet Design Documents / Source Reference


Services
System Processes
Applications
Environment Subsystems
NTDLL. dll
User Mode
System Threads
System Service Dispatcher
I/O Manager
Device &File System Drivers
Object Manager
Plug & Play Manager
Security reference Monitor
Virtual Memory
Processes & Threads
Other
Windows User, GDI
Graphics Drivers
Kernel
HAL
Kernel Mode
Hardware

Static Analyzer

- Erlauben einfach Klassen an Bugs zu finden
- Zusätzlich zu Compilerflags wie -wall, -wextra -pedantic,ubsan/asan
- CppCheck, PVS-Studio, Clang, Sonarqube
- Use after free, double free, malloc leak, freeing non heap allocated objects
- Microsoft benutzte Prefast, intern entwickeltes Tool
- Jetzt Teil von MSVC mit /analyze
- Für Kerneltreiber gibt es den Static Driver Verifier!

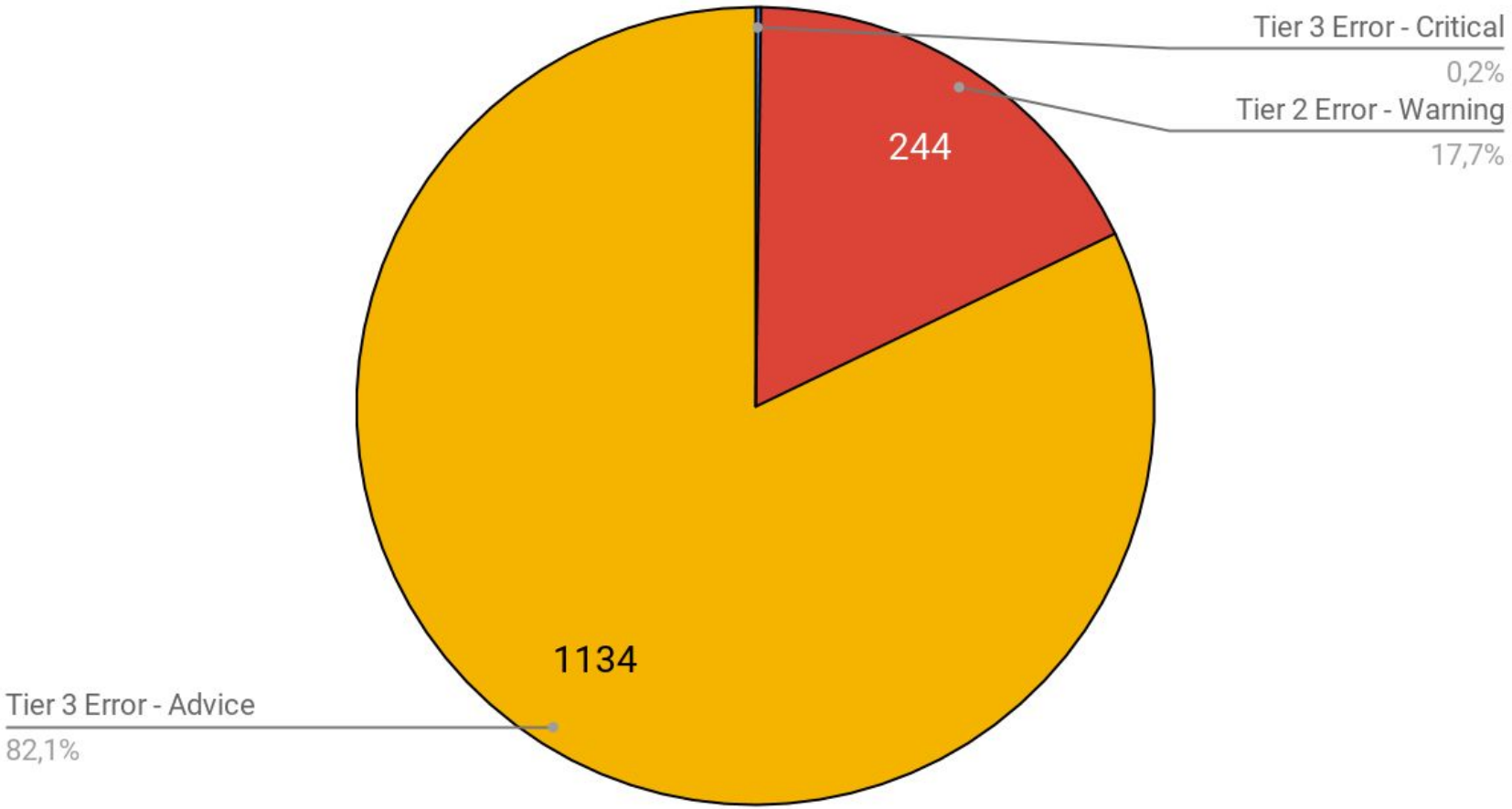
Hürden

- Großteil der Exploits targeted code der vorkompiliert ist und nicht im WRK liegt
- Logikfehler oder fehlende Validierung an unterschiedlichen stellen wird nicht erkannt (Check von Privilegien o.Ä.)
- Analyzer hat Probleme mit C->ASM Übergängen
- Custom Free Funktionen verhindern analyse von use after free/double free etc.

Viva64-PVS Studio

- Static Code Analyzer for C,C++,C# und Java
- Build Integration via Visual Studio
- Microsoft ist Kunde, analysiert wurden
  - Visual C++ Runtime
  - Windows 8 Treiber
  - Roslyn, MSBuild u.v.m.

PVS Studio Results



PVS-Studio Ergebnisse

- Tier 1 warnings sind größtenteils identisch mit pedantischen Compilerwarnings
- Comparison von unsigned mit signed Werten, if then Conditions sollen gruppiert werden etc.
- Tier 2&3 warnings sind annotiert, meistens fehlende Validierung
- PVS Studio ist kein Kernel Analyzer!
- Spezifische Fehler sollten nur analysiert werden
- Windows 10 wird mit neuerem Compiler gebaut -> besseres Tooling

```
else if (A == &inbound)
  str = "inbound";
else if (A == &inbound)
  str = "outbound";
```

Branch duplication

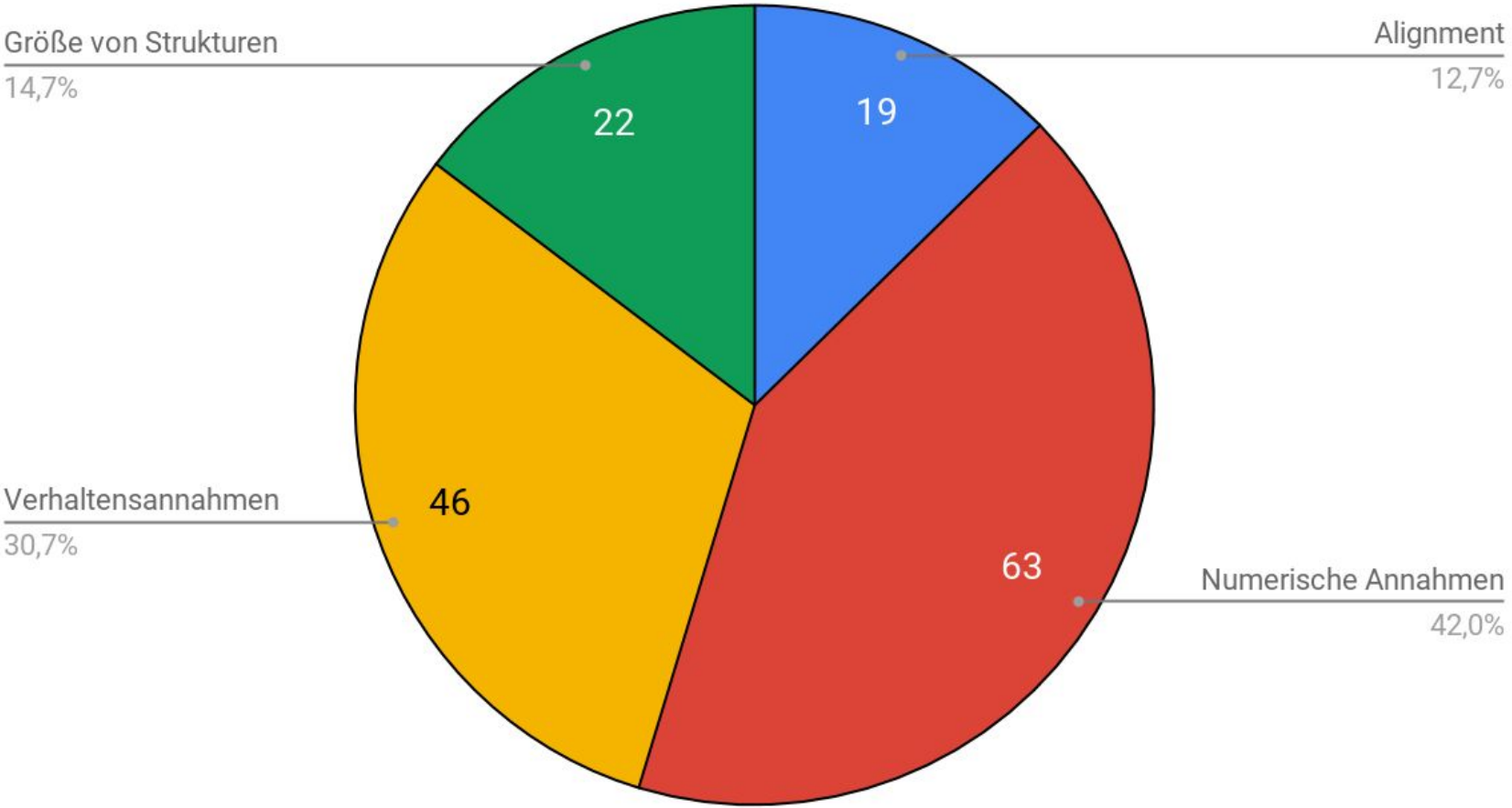
```
HRESULT GetNextSubscribedMessage()
{
  ....
  m_pWdfRequest = pWdfRequest;
  m_pWdfRequest->MarkCancelable(pCallbackCancel);
  if (m_pWdfRequest != NULL)
  {
    CompleteOneArrivalEvent();
  }
  ....
}
```

Possible NPR

```
This routine assumes that the input and output buffers has been
probed enough to determine that it is at least as large as
MinWnodeSize and MinWnodeSize must be at least as large as
sizeof(WNODE_HEADER)

WNODE Rules:

9. For outbound data WnodeDataBlockOffset != 0
5. For inbound Wnode->DataBlockOffset must be 0 (implying no data) or
   Wnode->DataBlockOffset must be <= incoming buffer size and >=
   sizeof(WNODE_SINGLE_INSTANCE), that is
   the data block must start in the incoming buffer, but after the
   WNODE_SINGLE_INSTANCE header.
6. Wnode and Wnode->DataBlockOffset must be aligned on an 8 byte boundray.
7. For inbound data (SetSingleInstance) (Wnode->DataBlockOffset +
   Wnode->DataBlockSize) < incoming buffer length. That is the entire
   data block must fit within the incoming buffer.
8. For outbound data (QuerySingleInstance) Wnode->DataBlockOffset
   must be <= outgoing buffer length. That is the start of the outgoing
   data block must fit within the outgoing data buffer. Note that it is
   the provider's responsibility to determine if there will be enough
   space in the outgoing buffer to write the returned data.

10. Wnode->OffsetInstanceNames must be aligned on a 2 byte boundary
11. Wnode->OffsetInstanceNames must be <= (incoming buffer size) +
    sizeof(USHORT), that is it must start within the incoming buffer and
    the USHORT that specifies the length must be within the incoming
    buffer.
12. The entire instance name string must fit with the incoming buffer
13. For outbound data (QuerySingleInstance) the entire instance name
    must start and fit within the output buffer.
14. Wnode->DataBlockOffset must be placed after any instance name and
    not overlap the instance name.
```

```
must be 0
must be <
must be smaller than
must be >
must be greater than
must be zero
must be non-zero
must be aligned
must fit within
at least as large
is expected to
is required to
```

Annahmen in Kommentaren (150)

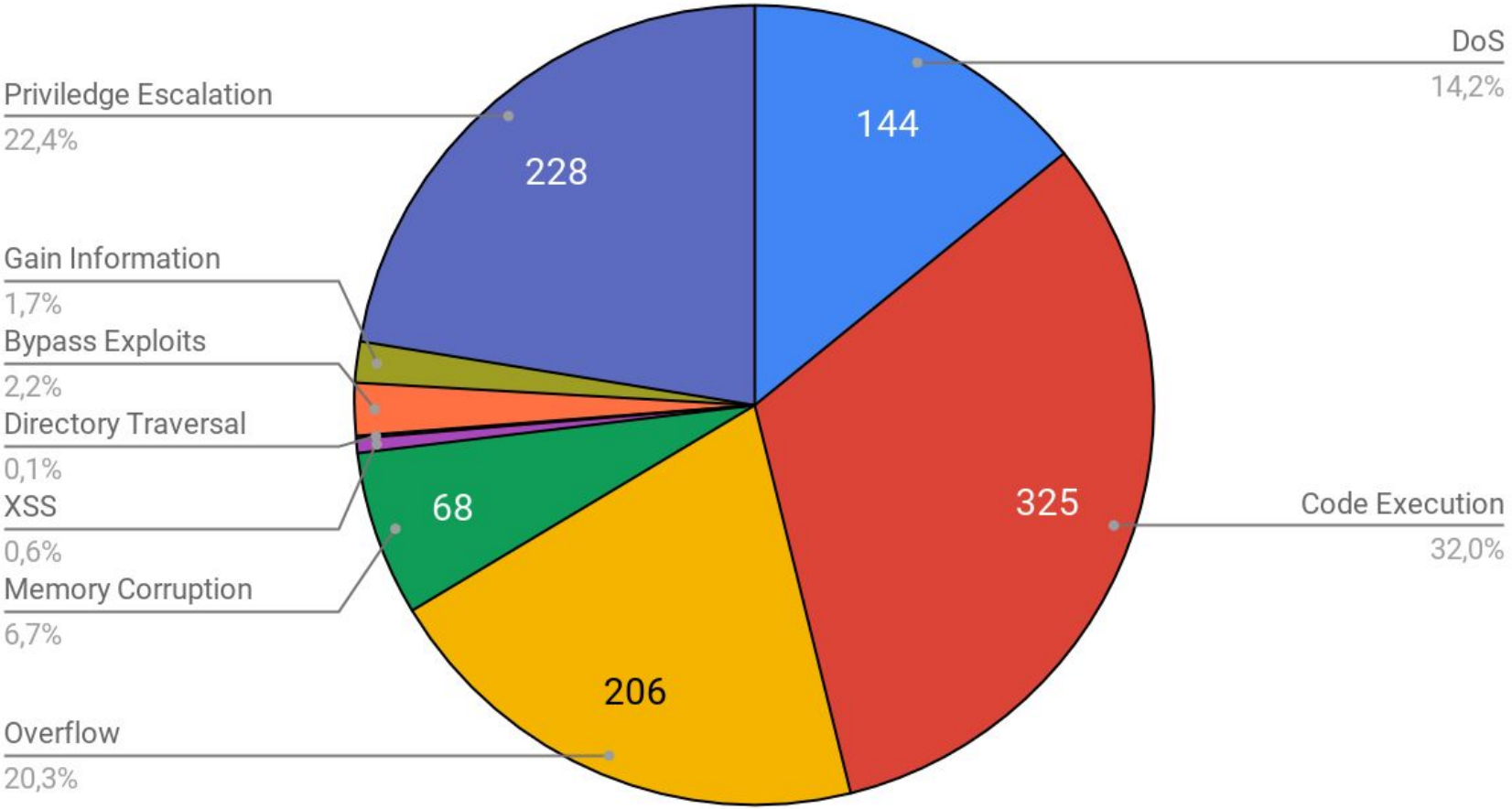


Mapping CVE->WRK

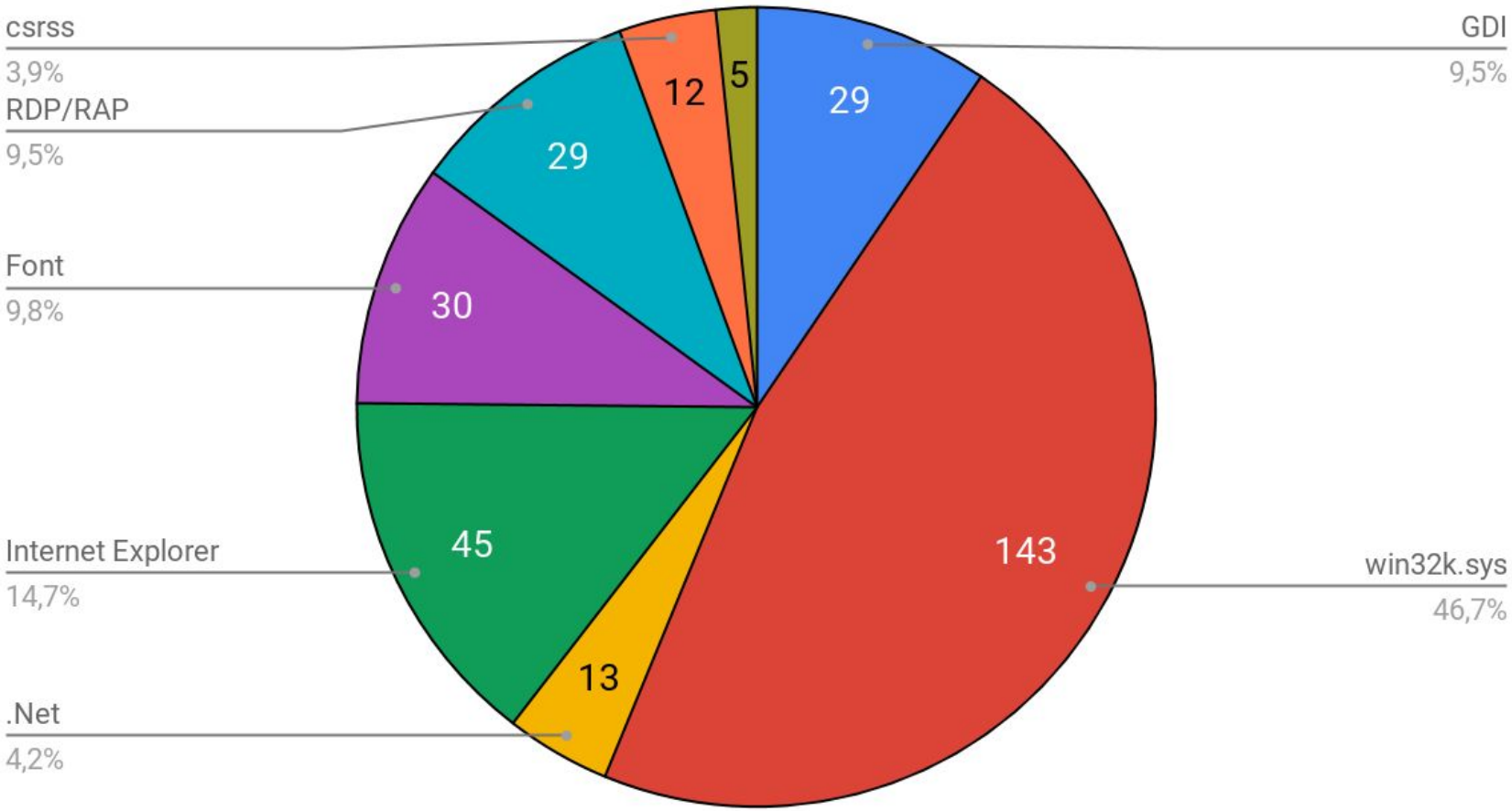
- 85% der Denial of Service Exploits in Windows Treibern
- Komponentenübergreifendes zusammenspiel von Fehlern[1]
- Top 100 CVE’s für Windows XP sind nicht im WRK zu finden
- Fuzzing ist wahrscheinlich die bessere Alternative bei limitierten Quellcode

1: https://dl.acm.org/doi/10.1145/1218063.1217943

Exploit origins



Exploit descriptions containing the Term



CVE-2005-2827

- Local privilege escalation
- Kompletter code ist im WRK
- Über Umwege wird Speicher bei einem vom User definierten Pointer ausgeführt.
- Unauffindbar für static analyzer

```
_ PspExitThread
__ KeFlushQueueApc
__ (detaches APC queues from ETHREAD_ApcState_ApcListHead)
__ (APC free loop begins)
__ ExFreePool(1st_APC -- queued by exited_process)
___ ExFreePoolWithTag(1st_APC)
____ ObfDereferenceObject(exited_process)
_____ ObpRemoveObjectRoutine
______ PspProcessDelete
_______ KeStackAttachProcess(exited_process)
________ KiAttachProcess
_________ KiMoveApcState(ETHREAD_ApcState --> duplicate)
_________ KiSwapProcess
________ PspExitProcess(0)
________ KeUnstackDetachProcess
_________ KiMoveApcState(duplicate --> ETHREAD_ApcState)
_________ KiSwapProcess
__ ExFreePool(2nd_APC)
__ ExFreePool(ETHREAD + 30h)
__ (APC free loop ends)
```

```
//
// flush user-mode APC queue
//

FirstEntry = KeFlushQueueApc (&Thread->Tcb, UserMode);

if (FirstEntry != NULL) {

    Entry = FirstEntry;
    do {
        Apc = CONTAINING_RECORD (Entry, KAPC, ApcListEntry);
        Entry = Entry->Flink;

        //
        // If the APC has a rundown routine then call it. Otherwise
        // deallocate the APC
        //

        if (Apc->RundownRoutine) {
            (Apc->RundownRoutine) (Apc);
        } else {
            ExFreePool (Apc);
        }

    } while (Entry != FirstEntry);
}
```

psinit.c pspExitThread

Fuzzing

Fuzzing [...] ist eine automatisierte Technik für Softwaretests, bei der das zu testende Programm an [...] Eingabeschnittstellen immer wieder mit Zufallsdaten beschickt wird. Mit zufälligen Daten können meistens Situationen im Betrieb des Programms erzeugt werden, die mit anderen Testverfahren nicht erreicht werden.

Kernel Fuzzing

- Generations- oder Mutationsbasiert
- “Dumb” oder “smart”, je nachdem ob er die Inputstruktur kennt
- White-, grey- oder black-box, je nachdem ob er die Programmstruktur kennt
- Kernel fuzzer sind meistens smart, da der Kernel sehr strukturierte Daten erwartet
- Größere Hürden als normale Programme zu fuzzen
- Erfolgreiche Beispiele sind kAFL, Syzkaller
- Win32k.sys sehr attraktiv für angreifer, da Sandbox escapes darüber einfach möglich sind

Syzcaller on Windows

```
int main(int argc, char **argv) {
    WNDCLASSEX WindowClass = { 0 };
    WindowClass.cbSize = sizeof(WNDCLASSEX);
    WindowClass.cbClsExtra = 0x2771;
    WindowClass.lpfnWndProc = DefWindowProc;
    WindowClass.lpszClassName = "Class";
    RegisterClassExA(&WindowClass);
    return 0;
}
```

Remainders of 16 bit Windows still remain a threat to new Windows Versions

Typical kernel bug: CVE-2018-0744

```
int main(int argc, char **argv) {
    WNDCLASSEX WindowClass = {0};
    HWND WindowA, WindowB, WindowC;
    ATOM Atom;
    WindowClass.cbSize         = sizeof(WNDCLASSEX);
    WindowClass.lpfnWndProc    = DefWindowProc;
    WindowClass.lpszClassName  = "Class";
    Atom = RegisterClassEx(&WindowClass);
    WindowA = CreateWindowEx(0, MAKEINTATOM(Atom), "One", 0, CW_USEDEFAULT,
                             0, 128, 128, NULL, NULL, NULL, NULL);
    SetClassLong(WindowA, GCL_STYLE, CS_CLASSDC);
    WindowB = CreateWindowEx(0, MAKEINTATOM(Atom), "Two", 0, CW_USEDEFAULT,
                             0, 128, 128, NULL, NULL, NULL, NULL);
    GetDC(WindowA);
    SetClassLong(WindowA, GCL_STYLE, CS_CLASSDC | CS_OWNDC);
    WindowC = CreateWindowEx(0, MAKEINTATOM(Atom), "Three", 0, CW_USEDEFAULT,
                             0, 128, 128, NULL, NULL, NULL, NULL);
}
```

https://msrnd-cdn-stor.azureedge.net/bluehat/bluehatil/2019/assets/doc/Bugs%20on%20the%20Windshield%20Fuzzing%20the%20Windows%20Kernel.pdf

Quellen

- https://ars.els-cdn.com/content/image/1-s2.0-S0895717709003409-gr2.jpg, Zugriff 11.05.2020 14:00
- https://www.viva64.com/en/a/0077/#ID0EJIBI, Zugriff 11.05.2020 15:00
- https://docs.microsoft.com/en-us/cpp/build/reference/analyze-code-analysis?view=vs-2019,Zugriff 12.05.2020 11:00
- https://www.cvedetails.com/vulnerability-list/vendor_id-26/product_id-739/Microsoft-Windows-Xp.html, Zugriff 12.05.2020 12:00
- https://www.cvedetails.com/product/739/Microsoft-Windows-Xp.html?vendor_id=26 Daten für Grafik 1, Zugriff 22.07.2020
- https://www.securityfocus.com/archive/1/419377/100/0/threaded Zugriff 22.07.2020
- https://dl.acm.org/doi/10.1145/1218063.1217943 Zugriff 23.07.2020
- https://www.blackhat.com/presentations/bh-usa-07/Lindsay/Presentation/bh-usa-07-lindsay.pdf 23.07.2020
- M. M. Swift, B. N. Bershad, and H. M. Levy. Improving the reliability of commodity operating systems. In SOSP 03: Symposium on Operating System Principles, Seiten 207--222, Juni 2003 Zugriff am 23.07.2020
- https://msrnd-cdn-stor.azureedge.net/bluehat/bluehatil/2019/assets/doc/Bugs%20on%20the%20Windshield%20Fuzzing%20the%20Windows%20Kernel.pdf 23.07.2020
- http://www0.cs.ucl.ac.uk/staff/b.cook/pdfs/thorough_static_analysis_of_device_drivers.pdf Zugriff 28.07.2020