POPKORN: Popping Windows Kernel Drivers At Scale

Rajat Gupta, Lukas Dresel, Noah Spahn, Giovanni Vigna, Christopher Kruegel, Taesoo Kim

lukas.dresel@cs.ucsb.edu
@dreselli

Dec. 9th, 2022, Austin, TX, USA
Summary

- Vulnerable Windows Kernel drivers are crucial for modern malware
- Straight-forward analysis techniques can effectively find high-impact vulnerabilities
- Our Prototype POPKORN found 27 vulnerabilities
  - 2 CVEs and 6 acknowledgements from driver vendors
Robinhood Ransomware Borrow Vulnerable Driver To Kill Antivirus and Encrypt Windows System Files
AvosLocker Ransomware Variant Abuses Driver File to Disable Antivirus, Scans for Log4shell

We found an AvosLocker ransomware variant using a legitimate antivirus component to disable detection and blocking solutions.

By: Christophe Grandjean, Alex Nicco
May 02, 2022
Read time: 7 min (1825 words)
Lazarus APT Abuses Vulnerable Dell Drivers to Bypass Windows Security

AvosLocker Ransomware Variant Abuses Driver File to Disable Antivirus, Scans for Log4shell
Robinhood Ransomware Borrow Vulnerable Driver To Kill Antivirus System

AvosLocker Ransomware Variant Abuses Driver File to Disable Antivirus, Scans for Log4shell

Lazarus APT Abuses Vulnerable Dell Drivers to Bypass Windows Security

BlackByte Ransomware Abuses Vulnerable Windows Driver to Disable Security Solutions
Microsoft releases Driver Signature enforcement

Microsoft releases HVCI with Windows 10

Windows requires MS signature for drivers

Windows 11 enables HVCI by default
POPKORN

```c
void* SectionHandle = ZwOpenSection(
    GENERIC_WRITE,
    "\\Device\\PhysicalMemory",
);

void* user_input = IRP->SystemBuffer;

(SectionOffset, BaseAddress) = ZwMapViewOfSection(
    SectionHandle,
    ZwCurrentProcess(),
    *(size_t*)user_input, // offset
    *(size_t)user_input+8, // size
);

*(size_t*)user_input = SectionOffset;
*(void**)user_input + 8 = BaseAddress;
```
void* SectionHandle = ZwOpenSection(
    GENERIC_WRITE,
    "\\Device\\PhysicalMemory",
);

void* user_input = IRP->SystemBuffer;

(SectionOffset, BaseAddress) = ZwMapViewOfSection(
    SectionHandle,
    ZwCurrentProcess(),
    *(size_t*)user_input, // offset
    *(size_t)user_input+8, // size
);

*(size_t*)user_input = SectionOffset;
*(void**)user_input + 8 = BaseAddress;
POPKORN

```c
void* SectionHandle = ZwOpenSection(
    GENERIC_WRITE,
    "\\Device\\PhysicalMemory",
);

void* user_input = IRP->SystemBuffer;

(SectionOffset, BaseAddress) = ZwMapViewOfSection(
    SectionHandle,
    ZwCurrentProcess(),
    *(size_t*)user_input, // offset
    *(size_t)user_input+8, // size
);

*(size_t*)user_input = SectionOffset;
*(void**)user_input + 8 = BaseAddress;
```
```c
void* SectionHandle = ZwOpenSection(
    GENERIC_WRITE,
    "\\Device\\PhysicalMemory",
);

void* user_input = IRP->SystemBuffer;

(SectionOffset, BaseAddress) = ZwMapViewOfSection(
    SectionHandle,
    ZwCurrentProcess(),
    *(size_t*)user_input, // offset
    *(size_t)user_input+8, // size
);

*(size_t*)user_input = SectionOffset;
*(void**)user_input + 8 = BaseAddress;
```
POPKORN - Goals

- Lightweight analysis that can scale to large datasets of drivers
POPKORN - Goals

- Lightweight analysis that can scale to large datasets of drivers

- Exploitable & high-impact bugs
  - Arbitrary physical memory access
  - Arbitrary process access
POPKORN - Goals

- Lightweight analysis that can scale to large datasets of drivers

- Exploitable & high-impact bugs
  - Arbitrary physical memory access
  - Arbitrary process access

- Easily verifiable bug reports with high-confidence
Figure 2: POPKORN System Overview.
Figure 2: POPKORN System Overview.
Figure 2: POPKORN System Overview.
Evaluation

- 90,000 software packages
- 5,000 Windows kernel drivers
- 3,094 WDM drivers (62%)
- 271 drivers with sink functions
  - 212 unique
### Evaluation

<table>
<thead>
<tr>
<th>Kernel Function</th>
<th>Using Function</th>
<th>Vulnerable Usages</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total</td>
<td>Dedup.</td>
</tr>
<tr>
<td>MmMapIoSpace</td>
<td>240 (7.76%)</td>
<td>188</td>
</tr>
<tr>
<td>ZwMapViewOfSection</td>
<td>40 (1.29%)</td>
<td>32</td>
</tr>
<tr>
<td>ZwOpenProcess</td>
<td>14 (0.45%)</td>
<td>11</td>
</tr>
</tbody>
</table>

**Total (Unique)**          | **271 (8.76%)** | **212**          | **38** | **27** |

**Table 2:** Drivers using specific target functions, and drivers found vulnerable by POPKORN, out of a total of 3,094 WDM drivers.
Evaluation - Known vulnerabilities

- False negative analysis was performed using known-vulnerable drivers from the physmem repository [https://github.com/namazso/physmem_drivers](https://github.com/namazso/physmem_drivers)
- 30 unique vulnerable drivers with

<table>
<thead>
<tr>
<th>Analysis Result</th>
<th>#Drivers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reported vulnerability</td>
<td>20 (+1*)</td>
</tr>
<tr>
<td>Timed out</td>
<td>6</td>
</tr>
<tr>
<td>Analysis terminated</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 3: POPKORN results for physmem_drivers
Conclusion
Conclusion

- Implemented prototype analysis POPKORN to demonstrate how targeted analyses can scale to automated deployment
Conclusion

- Implemented prototype analysis POPKORN to demonstrate how targeted analyses can scale to automated deployment
- Detected 27 unique vulnerable kernel drivers in our real-world dataset
Conclusion

- Implemented prototype analysis POPKORN to demonstrate how targeted analyses can scale to automated deployment
- Detected 27 unique vulnerable kernel drivers in our real-world dataset
- Received two CVEs and six acknowledgements from vendors
Conclusion

- Implemented prototype analysis POPKORN to demonstrate how targeted analyses can scale to automated deployment
- Detected 27 unique vulnerable kernel drivers in our real-world dataset
- Received two CVEs and six acknowledgements from vendors
- Code and evaluation dataset on GitHub under `ucsb-seclab/popkorn-artifact`
Conclusion

- Implemented prototype analysis POPKORN to demonstrate how targeted analyses can scale to automated deployment
- Detected 27 unique vulnerable kernel drivers in our real-world dataset
- Received two CVEs and six acknowledgements from vendors
- Code and evaluation dataset on GitHub under ucsb-seclab/popkorn-artifact

<table>
<thead>
<tr>
<th>Kernel Function</th>
<th>Using Function</th>
<th>Vulnerable Usages</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total Dedup.</td>
<td>Total Dedup.</td>
</tr>
<tr>
<td>MmMapIoSpace</td>
<td>240 (7.76%)</td>
<td>24 (14)</td>
</tr>
<tr>
<td>ZwMapViewOfSection</td>
<td>40 (1.29%)</td>
<td>17 (12)</td>
</tr>
<tr>
<td>ZwOpenProcess</td>
<td>14 (0.45%)</td>
<td>0 (0)</td>
</tr>
<tr>
<td><strong>Total (Unique)</strong></td>
<td>271 (8.76%)</td>
<td>38 (27)</td>
</tr>
</tbody>
</table>

Table 2: Drivers using specific target functions, and drivers found vulnerable by POPKORN, out of a total of 3,094 WDM drivers.

Table 3: POPKORN results for physmem_drivers

<table>
<thead>
<tr>
<th>Analysis Result</th>
<th>#Drivers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reported vulnerability</td>
<td>20 (+1*)</td>
</tr>
<tr>
<td>Timed out</td>
<td>6</td>
</tr>
<tr>
<td>Analysis terminated</td>
<td>4</td>
</tr>
</tbody>
</table>
Conclusion

- Implemented prototype analysis POPKORN to demonstrate how targeted analyses can scale to automated deployment
- Detected 27 unique vulnerable kernel drivers in our real-world dataset
- Received two CVEs and six acknowledgements from vendors
- Code and evaluation dataset on GitHub under `ucsb-seclab/popkorn-artifact`

<table>
<thead>
<tr>
<th>Kernel Function</th>
<th>Using Function</th>
<th>Vulnerable Usages</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total</td>
<td>Dedup.</td>
</tr>
<tr>
<td>MmMapIoSpace</td>
<td>240 (7.76%)</td>
<td>188</td>
</tr>
<tr>
<td>ZwMapViewOfSection</td>
<td>40 (1.29%)</td>
<td>32</td>
</tr>
<tr>
<td>ZwOpenProcess</td>
<td>14 (0.45%)</td>
<td>11</td>
</tr>
<tr>
<td><strong>Total (Unique)</strong></td>
<td>271 (8.76%)</td>
<td>212</td>
</tr>
</tbody>
</table>

Table 2: Drivers using specific target functions, and drivers found vulnerable by POPKORN, out of a total of 3,094 WDM drivers.

<table>
<thead>
<tr>
<th>Analysis Result</th>
<th>#Drivers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reported vulnerability</td>
<td>20 (+1*)</td>
</tr>
<tr>
<td>Timed out</td>
<td>6</td>
</tr>
<tr>
<td>Analysis terminated</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 3: POPKORN results for `physmem_drivers`
References & Resources

- g_CiOptions in a Virtualized World - XPN InfoSec Blog
- The Swan Song for Driver Signature Enforcement Tampering | Fortinet Blog
- Microsoft signed a malicious Netfilter rootkit
- Defeating Windows Driver Signature Enforcement #1: default drivers | j00ru//vx tech blog
- BlueHat IL 2018 David Weston Windows Hardening with Hardware
- https://github.com/ucsb-seclab/polkorn-artifact
Kernel Driver Attack Surface in CVEs
“A non-administrative user mode process cannot access or tamper with kernel code and data. Administrator-to-kernel is not a security boundary.“

- Microsoft’s Security Servicing Criteria
NTSTATUS DeviceControlDispatch(_DEVICE_OBJECT *DeviceObject,
    _IRP *Irp)
{
    auto IoStackLocation = Irp->Tail.Overlay.CurrentStackLocation;
    auto Params = IoStackLocation->Parameters.DeviceIoControl;
    auto Buf = Irp->AssociatedIrp.SystemBuffer;
    auto Status = STATUS_UNSUCCESSFUL;

    if (Params.IoControlCode == 0x9C402530
        && Params.InputBufferLength >= 8
        && Params.OutputBufferLength >= 8)
    {
        // Map one page as determined by request
        PHYSICAL_ADDRESS* Address = (PHYSICAL_ADDRESS*)Buf;

        void* res = MmMapIoSpace(*Address, 0x1000, MmNonCached);

        if (res)
        {
            *(void**)Buf = res;
            IoCompleteRequest(Irp, 0);
            Status = STATUS_SUCCESS;
        }
    }
    return Status;
}
NTSTATUS DeviceControlDispatch(_DEVICE_OBJECT *DeviceObject, 
    _IRP *Irp)
{
    auto IoStackLocation = Irp->Tail.Overlay.CurrentStackLocation;
    auto Params = IoStackLocation->Parameters.DeviceIoControl;
    auto Buf = Irp->AssociatedIrq.SystemBuffer;
    auto Status = STATUS_UNSUCCESSFUL;

    if (Params.IoControlCode == 0x9C402530 
        && Params.InputBufferLength >= 8 
        && Params.OutputBufferLength >= 8)
    {
        // Map one page as determined by request
        PHYSICAL_ADDRESS * Address = (PHYSICAL_ADDRESS *)Buf;

        void * res = MmMapIoSpace(*Address, 0x1000, MmNonCached);

        if (res) {
            *(void **)Buf = res;
            IoCompleteRequest(Irp, 0);
            Status = STATUS_SUCCESS;
        }
    }
    return Status;
}