Amit Waisel
Offensive Cyber Security Expert
Technology lead, Security Research @ XM Cyber
Trusted Security Advisor
Favorite bit: 1
Private Pilot 🛩, Skipper ⚓️ and cat lover 🐱

Hila Cohen
Security Researcher @ XM Cyber
Passionate about Windows Internals and Malware Analysis
Love to dance, travel the world 🌍 and capture moments with my camera 📸
Malproxy - A new technique to bypass endpoint protections

Demo

Mitigations

TL;DR
Organizations heavily rely on endpoint protection solutions in their security stack.

Unfair cat-and-mouse game.

Security solutions evolved over time, so are the viruses.
What do you know about your endpoint protection solutions?
not great, not terrible

Anatoly Dyatlov
Endpoint Protection 101

malicious activity detection mechanisms

1. Static signatures
2. Heuristics
3. Behavioral signatures
//testbin.c
int main()
{
    char *user = "adm.user";
    printf("%s\n",user);
    return 0;
}
rule APT_adm_corp : apt //apt is just a tag, it doesn't affect the rule.
{
    meta:  //Metadata, they don't affect the rule
        author = "xgusix"

    strings:
        $adm = "adm."
        $corp = "corp."
        $elf = { 7f 45 4c 46 } //ELF file's magic numbers

    condition:
        $elf in (0..4) and ($adm or $corp)
// If $elf in the first 4 bytes and it matches $adm or $corp
}
Static signatures

Behavioral signatures

Heuristics

# yara -s -m -g rules.yar testbin

APT_adm_corp [apt] [author="xgusix"] testbin
0x0:$elf: 7F 45 4C 46
0x4c0:$adm: adm.
HackTool:Win32/OurCoolMimikatzSignature:
"A La Vie, A L'Amour" - (oe.eo)
Benjamin DELPY `gentilkiwi`
Vincent LE TOUX
## / \ ##
sekurlsa
logonpasswords
### Static Signatures

<table>
<thead>
<tr>
<th>Property</th>
<th>.text</th>
<th>.data</th>
<th>UPX2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raw-address</td>
<td>0x00000400</td>
<td>0x00000400</td>
<td>0x00003400</td>
</tr>
<tr>
<td>Raw-size</td>
<td>0x0 bytes</td>
<td>0x3000 bytes</td>
<td>0x200 bytes</td>
</tr>
<tr>
<td>Virtual-address</td>
<td>0x00401000</td>
<td>0x00407000</td>
<td>0x0040A000</td>
</tr>
<tr>
<td>Virtual-size</td>
<td>0x6000 bytes</td>
<td>0x3000 bytes</td>
<td>0x1000 bytes</td>
</tr>
<tr>
<td>Executable</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Writable</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
</tbody>
</table>

### Heuristics

- Property: Raw-address, Raw-size, Virtual-address, Virtual-size, Executable, Writable
- Values:
  - Raw-address: 0x00000400, 0x00000400, 0x00003400
  - Raw-size: 0x0 bytes, 0x3000 bytes, 0x200 bytes
  - Virtual-address: 0x00401000, 0x00407000, 0x0040A000
  - Virtual-size: 0x6000 bytes, 0x3000 bytes, 0x1000 bytes
  - Executable: +, -, +
  - Writable: +, +, -
Static signatures

Heuristics

Behavioral signatures
1 Static signatures

2 Heuristics

3 Behavioral signatures
Endpoint protection solutions bypass
Endpoint protection solutions bypass MALPROXY
Looking for my code somewhere?

You will never get this!
Malicious code interacts with the underlying OS using API function calls. Those actions can be detected and blocked by any security solution.
Proxy the malicious operations over the network

Never deploying the actual malicious code on the target side

Emulating needed API calls
Target & attacker stubs

Load the PE file and hook system API functions

Execution flow – hook, serialize, send, execute, serialize, send, return. Repeat.
Target & attacker stubs

Load the PE file and hook system API functions

Execution flow – hook, serialize, send, execute, serialize, send, return. Repeat.
Target & attacker stubs

Load the PE file and hook system API functions

Execution flow – hook, serialize, send, execute, serialize, send, return. Repeat.
Target & attacker stubs

Load the PE file and hook system API functions

Execution flow – hook, serialize, send, execute, serialize, send, return. Repeat.
Target & attacker stubs

Load the PE file and hook system API functions

Execution flow – hook, serialize, send, execute, serialize, send, return. Repeat.
Key terms:

SYSTEM CALLS

OVERVIEW

USER MODE

Windows Application

Kernel32.dll CreateFile

Ntdll.dll NtCreateFile

KERNEL MODE

Ntoskrnl ZwCreateFile

Call CreateFile

Call NtCreateFile

SYSENTER\SYSCALL
Find relevant function in SSDT and executes it
Key terms:

SYSTEM CALLS

OVERVIEW

USER MODE

Kernel32.dll CreateFile

CALL

Call CreateFile

CALL

Call NtCreateFile

SYSENTER\SYSCALL

Find relevant function in SSDT and executes it

KERNEL MODE

Ntoskrnl ZwCreateFile
Key terms:

**SYSTEM CALLS**

**OVERVIEW**

**USER MODE**

- **Kernel32.dll**
  - CreateFile

**KERNEL MODE**

- **Ntoskrnl**
  - ZwCreateFile

Call CreateFile

Call NtCreateFile

SYSENTER\SYSCALL

Find relevant function in SSDT and executes it
Key terms:

**SYSTEM CALLS**

**OVERVIEW**

**USER MODE**

- Kernel32.dll
  - CreateFile

**KERNEL MODE**

- Ntoskrnl
  - ZwCreateFile

Find relevant function in SSDT and executes it.
Key terms: SYSTEM CALLS

OVERVIEW

USER MODE
- Windows Application
  - Kernel32.dll
    - CreateFile
  - Ntdll.dll
    - NtCreateFile

KERNEL MODE
- Ntoskrnl
  - ZwCreateFile

Call CreateFile
Call NtCreateFile
SYSENTER\SYSCALL
Find relevant function in SSDT and executes it
**Key terms:**

**SYSTEM CALLS**

**OVERVIEW**

**USER MODE**
- Windows Application
- Kernel32.dll
  - CreateFile
- Ntdll.dll
  - NtCreateFile

**KERNEL MODE**
- Ntoskrnl
  - ZwCreateFile

- Call CreateFile
- Call NtCreateFile
- Ntoskrnl
  - ZwCreateFile

- Find relevant function in SSDT and executes it

**COMPUTER OS**

**Innocent code**

**Process**
Redirect system API calls to our code

Imported system API function addresses are resolved during PE load process and can be overridden later – IAT hooking

Control all arguments & return value

This allows us to separate the code’s logic from its interaction with the OS

<table>
<thead>
<tr>
<th>IMPORT ADDRESS TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NtQuerySystemInformation</td>
</tr>
<tr>
<td>OpenProcess</td>
</tr>
<tr>
<td>ReadProcessMemory</td>
</tr>
<tr>
<td>BCryptGenerateSymetricKey</td>
</tr>
<tr>
<td>ConvertSidToStringSidW</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>RtlAdjustPrivilege</td>
</tr>
<tr>
<td>NtQueryInformationProcess</td>
</tr>
<tr>
<td>RtlEqualUnicodeString</td>
</tr>
</tbody>
</table>
Key terms:

FUNCTION

BOOL stdcall ReadProcessMemory(HANDLE hProcess, LPCVOID lpBaseAddress, LPVOID lpBuffer, SIZE_T nSize, SIZE_T *lpNumberOfBytesRead);

Return Type  Calling Convention  Function arguments
Proxying Win32 API

Dealing with all aspects of different prototypes

- Calling convention – same for all Win32API and Native API calls

  Input Arguments:
  - Primitives
  - Pointers to primitives
  - User-allocated buffers

  Output Arguments:
  - User-allocated output buffer
  - System-allocated output buffer

  Return values

YO Dawg, heard you love pointers
So I made you a pointer to a pointer to a pointer...
Handling

ARGUMENTS

NTSTATUS NtQueryInformationProcess(
    IN HANDLE ProcessHandle,
    IN PROCESSINFOCLASS ProcessInformationClass,
    OUT PVOID ProcessInformation,
    IN ULONG ProcessInformationLength,
    OUT PULONG ReturnLength
);

Handling Arguments

ATTACKER SIDE

Request Message

```
NTSTATUS NtQueryInformationProcess(
    IN HANDLE ProcessHandle,
    IN PROCESSINFOCLASS ProcessInformationClass,
    OUT PVOID ProcessInformation,
    IN ULONG ProcessInformationLength,
    OUT PULONG ReturnLength
);
```
ATTACKER SIDE

Request Message

```
NTSTATUS NtQueryInformationProcess(
    IN HANDLE ProcessHandle,
    IN PROCESSINFOCLASS ProcessInformationClass,
    OUT PVOID ProcessInformation,
    IN ULONG ProcessInformationLength,
    OUT PULONG ReturnLength
);
```
**ATTACKER SIDE**

Request Message

```c
NTSTATUS NtQueryInformationProcess(
    IN HANDLE ProcessHandle,
    IN PROCESSINFOCLASS ProcessInformationClass,
    OUT PVOID ProcessInformation,
    IN ULONG ProcessInformationLength,
    OUT PULONG ReturnLength
);
```
ATTACKER SIDE

Request Message

NTSTATUS NtQueryInformationProcess(
    IN HANDLE ProcessHandle,
    IN PROCESSINFOCLASS ProcessInformationClass,
    OUT PVOID ProcessInformation,
    IN ULONG ProcessInformationLength,
    OUT PULONG ReturnLength
);
Handling ARGUMENTS

ATTACKER SIDE

Request Message

NTSTATUS NtQueryInformationProcess(
    IN HANDLE ProcessHandle,
    IN PROCESSINFOCLASS ProcessInformationClass,
    OUT PVOID ProcessInformation,
    IN ULONG ProcessInformationLength,
    OUT PULONG ReturnLength
);
Handling ARGUMENTS

ATTACKER SIDE

TARGET SIDE

Request Message

NTSTATUS NtQueryInformationProcess(
    IN HANDLE ProcessHandle,
    IN PROCESSINFOCLASS ProcessInformationClass,
    OUT PVOID ProcessInformation,
    IN ULONG ProcessInformationLength,
    OUT PULONG ReturnLength
);

ProcessHandle
ProcessInformationClass
ProcessInformation
ProcessInformationLength
ReturnLength
ATTAACKER SIDE

NTSTATUS NtQueryInformationProcess(
    IN HANDLE ProcessHandle,
    IN PROCESSINFOCLASS ProcessInformationClass,
    OUT PVOID ProcessInformation,
    IN ULONG ProcessInformationLength,
    OUT PULONG ReturnLength
);

TARGET SIDE

Request Message

ProcessHandle
ProcessInformationClass
ProcessInformationLength

NTSTATUS NtQueryInformationProcess(
    IN HANDLE ProcessHandle,
    IN PROCESSINFOCLASS ProcessInformationClass,
    OUT PVOID ProcessInformation,
    IN ULONG ProcessInformationLength,
    OUT PULONG ReturnLength
);
Handing
ARGUMENTS

ATTACKER SIDE

NTSTATUS NtQueryInformationProcess(
    IN HANDLE ProcessHandle,
    IN PROCESSINFOCLASS ProcessInformationClass,
    OUT PVOID ProcessInformation,
    IN ULONG ProcessInformationLength,
    OUT PULONG ReturnLength
);
Handling ARGUMENTS

ATTACKER SIDE

NTSTATUS NtQueryInformationProcess(
    IN HANDLE ProcessHandle,
    IN PROCESSINFOCLASS ProcessInformationClass,
    OUT PVOID ProcessInformation,
    IN ULONG ProcessInformationLength,
    OUT PULONG ReturnLength
);

TARGET SIDE

NTSTATUS NtQueryInformationProcess(
    IN HANDLE ProcessHandle,
    IN PROCESSINFOCLASS ProcessInformationClass,
    OUT PVOID ProcessInformation,
    IN ULONG ProcessInformationLength,
    OUT PULONG ReturnLength
);

Response Message

ReturnLength
Handling ARGUMENTS

ATTACKER SIDE

NTSTATUS NtQueryInformationProcess(
    IN HANDLE ProcessHandle,
    IN PROCESSINFOCLASS ProcessInformationClass,
    OUT PVOID ProcessInformation,
    IN ULONG ProcessInformationLength,
    OUT PULONG ReturnLength
);

TARGET SIDE

Response Message

NTSTATUS NtQueryInformationProcess(
    IN HANDLE ProcessHandle,
    IN PROCESSINFOCLASS ProcessInformationClass,
    OUT PVOID ProcessInformation,
    IN ULONG ProcessInformationLength,
    OUT PULONG ReturnLength
);
Handling Arguments

**ATTACKER SIDE**

```c
NTSTATUS NtQueryInformationProcess(
    IN HANDLE ProcessHandle,
    IN PROCESSINFOCLASS ProcessInformationClass,
    OUT PVOID ProcessInformation,
    IN ULONG ProcessInformationLength,
    OUT PULONG ReturnLength
);
```

**TARGET SIDE**

```c
NTSTATUS NtQueryInformationProcess(
    IN HANDLE ProcessHandle,
    IN PROCESSINFOCLASS ProcessInformationClass,
    OUT PVOID ProcessInformation,
    IN ULONG ProcessInformationLength,
    OUT PULONG ReturnLength
);
```

Response Message

- ProcessInformation
- ReturnLength
- Return value
Handling Arguments

ATTACKER SIDE

Response Message

NTSTATUS NtQueryInformationProcess(
  IN HANDLE ProcessHandle,
  IN PROCESSINFOCLASS ProcessInformationClass,
  OUT PVOID ProcessInformation,
  IN ULONG ProcessInformationLength,
  OUT PULONG ReturnLength
);

TARGET SIDE

NTSTATUS NtQueryInformationProcess(
  IN HANDLE ProcessHandle,
  IN PROCESSINFOCLASS ProcessInformationClass,
  OUT PVOID ProcessInformation,
  IN ULONG ProcessInformationLength,
  OUT PULONG ReturnLength
);
Handling ARGUMENTS

**ATTACKER SIDE**

Response Message

- ProcessInformation
- ReturnLength
- Return value

```c
NTSTATUS NtQueryInformationProcess(
    IN HANDLE ProcessHandle,
    IN PROCESSINFOCLASS ProcessInformationClass,
    OUT PVOID ProcessInformation,
    IN ULONG ProcessInformationLength,
    OUT PULONG ReturnLength
);
```

**TARGET SIDE**

```c
NTSTATUS NtQueryInformationProcess(
    IN HANDLE ProcessHandle,  // ProcessHandle,
    IN PROCESSINFOCLASS ProcessInformationClass, // ProcessInformationClass,
    OUT PVOID ProcessInformation, // ProcessInformation,
    IN ULONG ProcessInformationLength, // ProcessInformationLength,
    OUT PULONG ReturnLength  // ReturnLength
);
```
RECAP

- Target & attacker stubs
- Load the PE file and hook system API functions
- Execution flow – hook, serialize, send, execute, serialize, send, return. Repeat.

Attacker OS

- Process
- Malicious code

Target OS

- Process
- Innocent code

API

Target & attacker stubs
Running MALPROXY

ATTACKER SIDE

TARGET SIDE
## ATTACKER SIDE

### IMPORT ADDRESS TABLE

<table>
<thead>
<tr>
<th>Function</th>
<th>DLL</th>
</tr>
</thead>
<tbody>
<tr>
<td>NtQuerySystemInformation</td>
<td>Kernel32.dll</td>
</tr>
<tr>
<td>OpenProcess</td>
<td>Kernel32.dll</td>
</tr>
<tr>
<td>ReadProcessMemory</td>
<td>Ntdll.dll</td>
</tr>
<tr>
<td>BCryptGenerateSymetricKey</td>
<td>Bcrypt.dll</td>
</tr>
<tr>
<td>ConvertSidToStringSidW</td>
<td>Advapi32.dll</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>RtlAdjustPrivilege</td>
<td>Ntdll.dll</td>
</tr>
<tr>
<td>NtQueryInformationProcess</td>
<td>Ntdll.dll</td>
</tr>
<tr>
<td>RtlEqualUnicodeString</td>
<td>Ntdll.dll</td>
</tr>
</tbody>
</table>
### IMPORT ADDRESS TABLE

<table>
<thead>
<tr>
<th>Function</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>NtQuerySystemInformation</td>
<td>Malproxy</td>
</tr>
<tr>
<td>OpenProcess</td>
<td>Malproxy</td>
</tr>
<tr>
<td>ReadProcessMemory</td>
<td>Malproxy</td>
</tr>
<tr>
<td>BCryptGenerateSymetricKey</td>
<td>Bcrypt.dll</td>
</tr>
<tr>
<td>ConvertSidToStringSidW</td>
<td>Advapi32.dll</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>RtlAdjustPrivilege</td>
<td>Malproxy</td>
</tr>
<tr>
<td>NtQueryInformationProcess</td>
<td>Malproxy</td>
</tr>
<tr>
<td>RtlEqualUnicodeString</td>
<td>Ntdll.dll</td>
</tr>
</tbody>
</table>
Running M ALPROXY

ATTACKER SIDE

RtlAdjustPrivilege
NtQuerySystemInformation
RtlEqualUnicodeString
OpenProcess
NtQueryInformationProcess
ReadProcessMemory
BCryptDecrypt

TARGET SIDE
Running MALPROXY

ATTACKER SIDE

RtlAdjustPrivilege
NtQuerySystemInformation
RtlEqualUnicodeString
OpenProcess
NtQueryInformationProcess
ReadProcessMemory
BCryptDecrypt

TARGET SIDE

RtlAdjustPrivilege
Running MALPROXY

ATTACKER SIDE

RtlAdjustPrivilege
NtQuerySystemInformation
RtlEqualUnicodeString
OpenProcess
NtQueryInformationProcess
ReadProcessMemory
BCryptDecrypt

TARGET SIDE

RtlAdjustPrivilege
NtQuerySystemInformation
Chrome.exe, explorer.exe
Calc.exe, lsass.exe
Running MALPROXY

ATTACKER SIDE

RtlAdjustPrivilege
NtQuerySystemInformation
RtlEqualUnicodeString
OpenProcess
NtQueryInformationProcess
ReadProcessMemory
BCryptDecrypt

TARGET SIDE

RtlAdjustPrivilege
NtQuerySystemInformation
Running MALPROXY

ATTACKER SIDE

RtlAdjustPrivilege
NtQuerySystemInformation
RtlEqualUnicodeString
OpenProcess
NtQueryInformationProcess
ReadProcessMemory
BCryptDecrypt

TARGET SIDE

RtlAdjustPrivilege
NtQuerySystemInformation
OpenProcess

Handle 0x00000080

PID 1234
Running MALPROXY

ATTACKER SIDE

RtlAdjustPrivilege
NtQuerySystemInformation
RtlEqualUnicodeString
OpenProcess
NtQueryInformationProcess
ReadProcessMemory
BCryptDecrypt

TARGET SIDE

RtlAdjustPrivilege
NtQuerySystemInformation
OpenProcess
NtQueryInformationProcess

Handle 0x00000080
PEB at 0xdeadbeef
Running MALPROXY

ATTACKER SIDE

- RtlAdjustPrivilege
- NtQuerySystemInformation
- RtlEqualUnicodeString
- OpenProcess
- NtQueryInformationProcess
- ReadProcessMemory
- BCryptDecrypt

TARGET SIDE

- RtlAdjustPrivilege
- NtQuerySystemInformation
- OpenProcess
- NtQueryInformationProcess
- ReadProcessMemory

Read 0xdeadbeef
[0x12, 0x34, 0x56, 0x78]
ATTACKER SIDE

- RtlAdjustPrivilege
- NtQuerySystemInformation
- RtlEqualUnicodeString
- OpenProcess
- NtQueryInformationProcess
- ReadProcessMemory
- BCryptDecrypt

TARGET SIDE

- RtlAdjustPrivilege
- NtQuerySystemInformation
- OpenProcess
- NtQueryInformationProcess
- ReadProcessMemory
Running MALPROXY

ATTACKER SIDE

RtlAdjustPrivilege
NtQuerySystemInformation
RtlEqualUnicodeString
OpenProcess
NtQueryInformationProcess
ReadProcessMemory
BCryptDecrypt

TopSecretPassword

TARGET SIDE

RtlAdjustPrivilege
NtQuerySystemInformation
OpenProcess
NtQueryInformationProcess
ReadProcessMemory

PWNED!
DEMO
You came off as a naive idiot. And naive idiots are not a threat.
Endpoint protections

BYPASS

- Bypassing Static Signatures
- Bypassing Heuristic Rules
- Behavioral Signatures
<table>
<thead>
<tr>
<th>Security Solution</th>
<th>Mimikatz sekurls::logonpasswords</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Defender</td>
<td>Malproxied!</td>
</tr>
<tr>
<td>Symantec Norton Security</td>
<td>Malproxied!</td>
</tr>
<tr>
<td>Kaspersky Internet Security</td>
<td>Blocks ReadProcessMemory without a verdict</td>
</tr>
<tr>
<td>ESET Smart Security</td>
<td>Malproxied!</td>
</tr>
<tr>
<td>Avast Free Antivirus</td>
<td>Blocks OpenProcess on lsass.exe without a verdict</td>
</tr>
<tr>
<td>Bitdefender Total Security</td>
<td>Malproxied!</td>
</tr>
<tr>
<td>McAfee Total Protection</td>
<td>Malproxied!</td>
</tr>
</tbody>
</table>
Why worry about something that isn't going to happen?
Hunt and sign the target-side proxy stub

Improve the behavioral signature engines to handle their known weaknesses

Any more ideas?
Hunt and sign the target-side proxy stub.

Improve the behavioral signature engines to handle their known weaknesses.

Any more ideas?

/dev/null
The Crazy Ideas Section - Remote Syscalls by Yaron Shani:

Syscall Proxying - Simulating remote execution by Maximiliano Caceres:

Syscall Proxying || Pivoting Systems by Filipe Balestra and Rodrigo Rubira Branco:
https://www.kernelhacking.com/rodrigo/docs/H2HCIII.pdf
Questions?