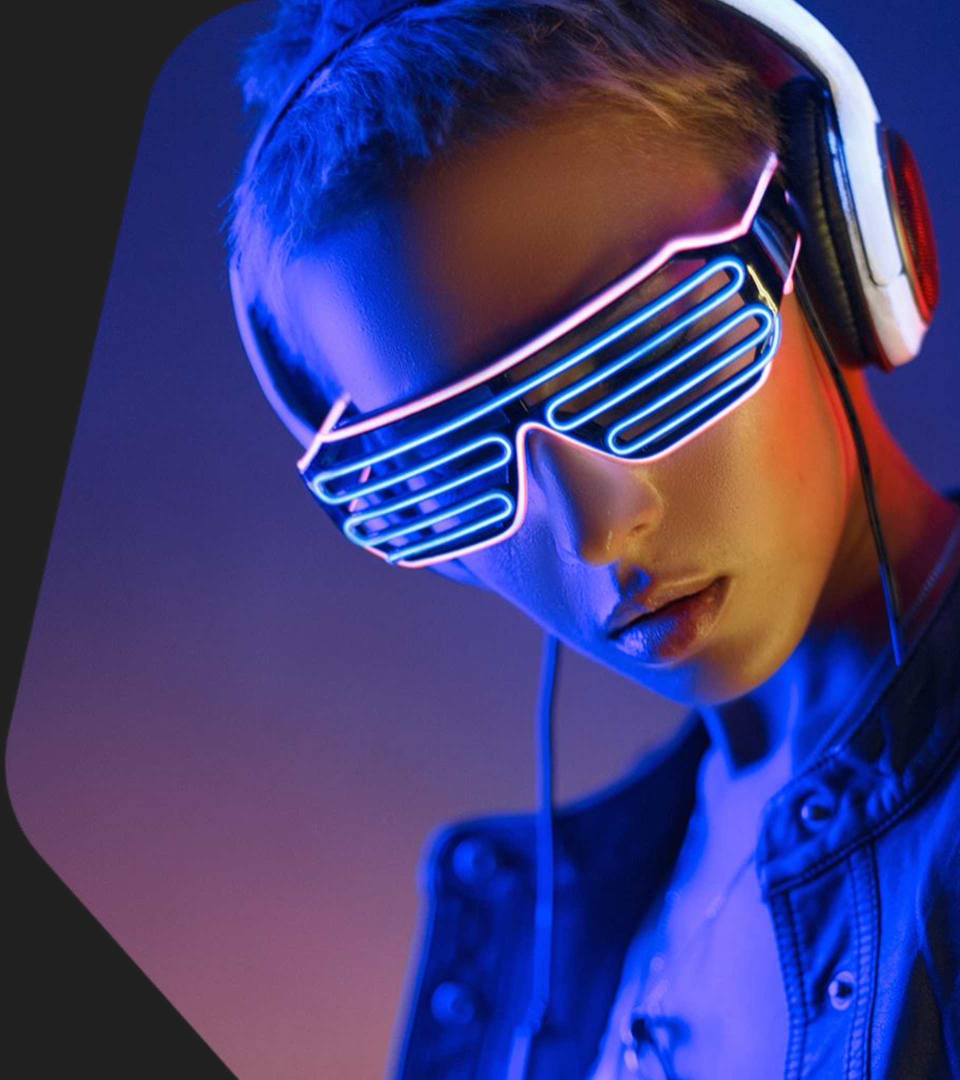
Advanced Malware Analysis Techniques

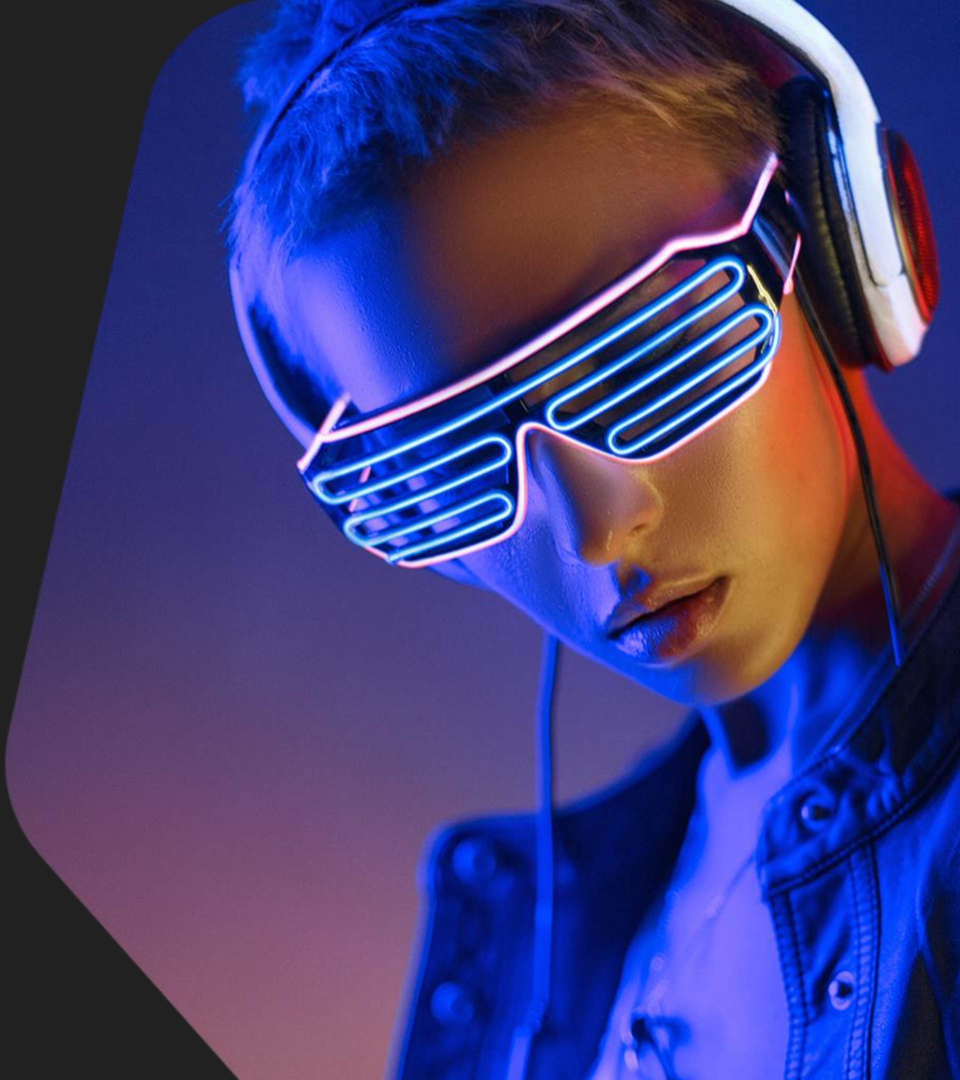
Course training manual

kaspersky



Welcome

Course overview



kaspersky

About your trainer - Igor Kuznetsov

- Joined Kaspersky in 2001 as a virus analyst
- Specializes in: reverse engineering, forensics, incident response
- Analyzed: Flame, Gauss, Red October, ATM protocols, etc.
- The exercises are from Igor's work / portfolio.

Copy for Lee Wei Yeong

The course - Main focus

- Reverse engineering disassembled code in IDA Pro
- Automating decryption, decoding and other processing of the samples
- Automating routine tasks
- Preserving most important steps of the analysis in code
- Most activities apply to the generic analysis workflow
- Exercises include unique corner cases that require special treatment

Copy for Lee Wei Yeong

The course - Overview

- x86/64 Intel code
- Windows PE, Mac OS X Mach-O files, raw shellcode for Windows
- RTF, OLE2, PDF documents
- All exercises are hands-on
- Most tracks include code templates that need to be filled in / modified to solve the exercises
- The scripts are written in Python 3 and most are standalone, but most of the tasks can also be done via the IDAPython scripting interface

Copy for Lee Wei Yeong

The course - The disclaimer

- There is no single correct way to do reverse engineering
- The code and scripts are used for real-life analysis
- The workflow displayed is how we do it

Copy for Lee Wei Yeong

Library of useful links

IDA tips and tricks - hex-rays.com/blog/tag/idatips/

IDA Pro Shortcuts - hex-rays.com/products/ida/support/freefiles/IDA_Pro_Shortcuts.pdf

Sysinternals - Windows Internals, Part 1 - amazon.com/dp/0735684189

Nostarch - IDA Pro book - nostarch.com/idapro2.htm

MSDN: PE format - PE Format - Win32 apps | Microsoft Docs

X86 Opcode and Instruction Reference - X86 Opcode and Instruction Reference (x86asm.net)

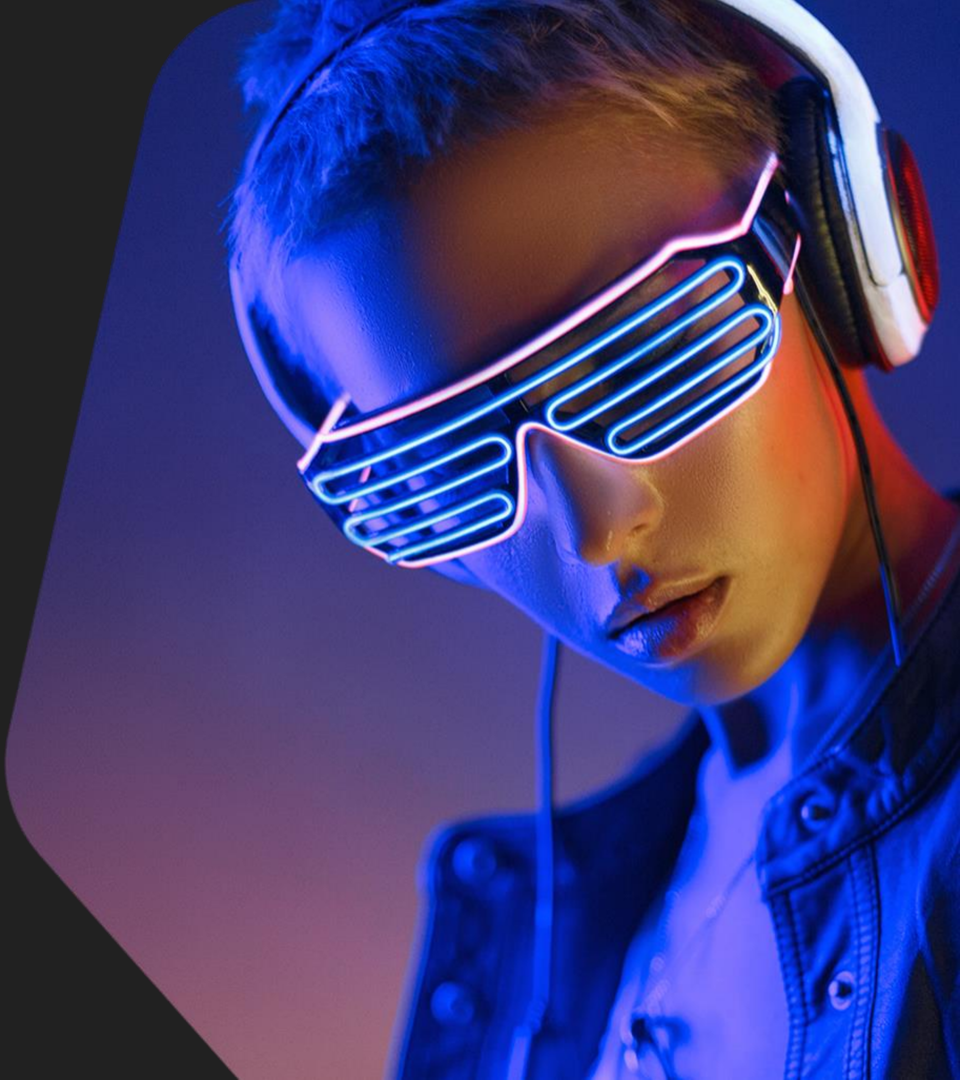
Capstone Disassembler - The Ultimate Disassembly Framework – Capstone – The Ultimate Disassembler (capstone-engine.org)

Intel® 64 and IA-32 Architectures Software Developer's Manual Combined Volumes https://software.intel.com/content/www/us/en/develop/download/intel-64-and-ia-32-architectures-sdm-combined-volumes-1-2a-2b-2c-2d-3a-3b-3c-3d-and-4.html

Copy for Lee Wei Yeong

Track 1

Intro



kaspersky

Intro - Overview

In this track you will practice:

- Routine IDA Pro tasks: navigation, functions, code and data manipulation
- Advanced features of IDA Pro: structure types, fields, shifted structure pointers
- Code and data flow analysis
- Stack arithmetics

Copy for Lee Wei Yeong

Intro - Mission briefing

Initial context:

- A bank is being robbed in real-time
- A suspected victim machine is analysed during IR
- Artifact: RWX memory page in a 32-bit Windows process
- Memory is dumped and passed for analysis

Copy for Lee Wei Yeong

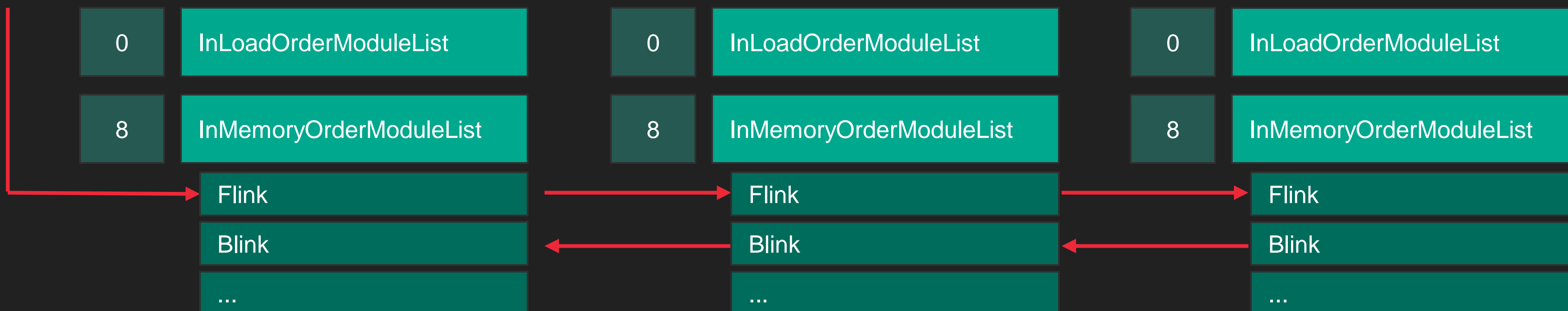
Intro - IDA Pro cheatsheet

- Make code - **C**
- Undefine - **U**
- Follow the reference - **Enter** / Double click
- Go back - **Esc**
- Make function - **P**
- Type libraries - **Shift-F11**
- Structures - **Shift-F9**
- Add (structure, type, etc.) - **Ins**
- Apply struct offset - **T**
- Apply struct offset advanced - select**+T**
- Set type - **Y**
- Parse C header file - **Ctrl+F9**

Copy for Lee Wei Yeong

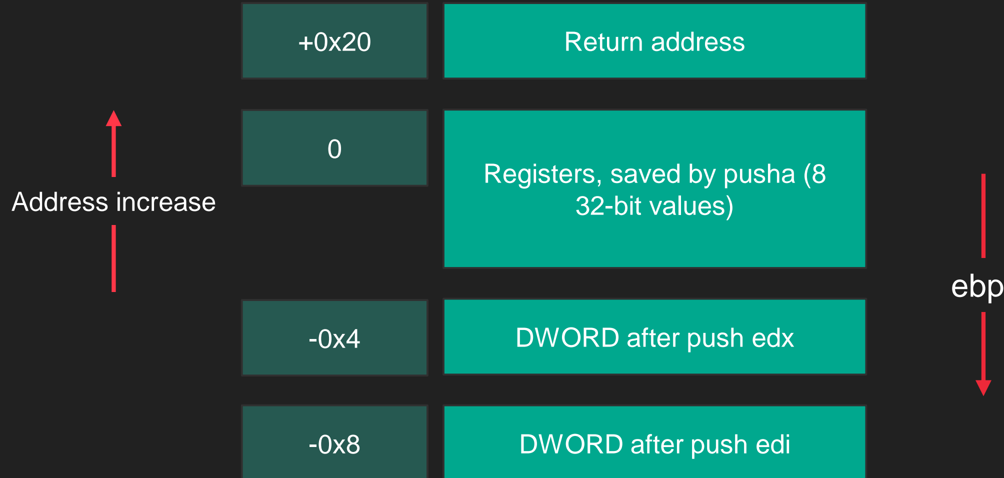
Intro - Pointer into the middle of a structure

Copy for Lee Wei Yeong



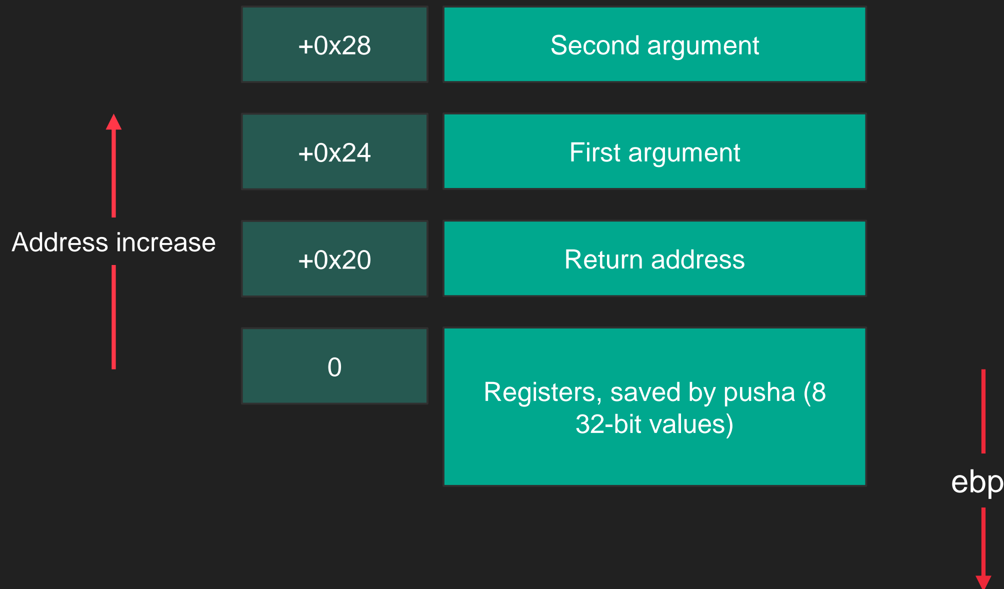
- Widely used by the MSVC compiler for iterating over structure pointers
- Unavoidable for “generic” linked list structures

Intro - Stack frame and stack pointer



Copy for Lee Wei Yeong

Intro - Solution for exercise 4



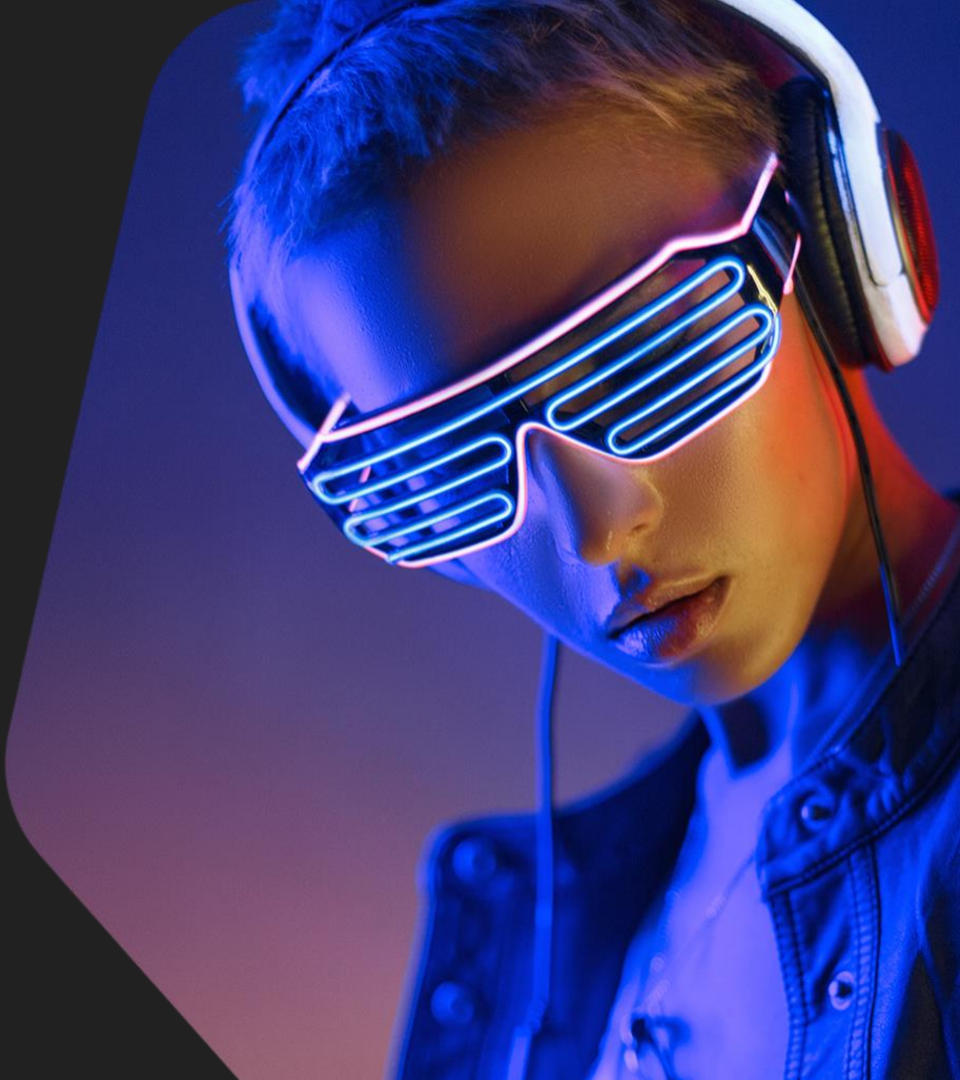
ebp

Copy for Lee Wei Yeong

kaspersky

Track 2

Shell



kaspersky

Shell - Overview

In this track you will practice:

- Code and data flow analysis
- Stack mechanics and data layout
- Manual reconstruction of data structures

Copy for Lee Wei Yeong

Shell - Mission briefing

Initial context:

- Another bank is being robbed (it happens all the time)
- The infected machine is located, it is being analysed during IR
- Artifact: RWX memory page in a 32-bit Windows process
- Memory is dumped and passed for analysis, and it looks familiar

Copy for Lee Wei Yeong

Shell - Solution for exercise 1

```
int WSAAPI connect(
  SOCKET         s,
  const sockaddr *name,
  int            namelen
);
```

- Win32 API use the **stdcall** calling convention
- All arguments are passed on the stack, in the reverse order: **first** argument is pushed **last**
- For Win64, the ABI is different: rcx, rdx, r8, r9, then stack in the reverse order

https://docs.microsoft.com/en-us/windows/win32/api/winsock2/nf-winsock2-connect
https://docs.microsoft.com/en-us/cpp/cpp/calling-conventions?view=msvc-160
https://docs.microsoft.com/en-us/cpp/build/x64-calling-convention?view=msvc-160

Copy for Lee Wei Yeong

Shell - Solution for exercise 2

Address	Opcode bytes	Disassembly
000000B3 000000B8	68 0A 01 03 D2 68 02 00 11 5C	push 0D203010Ah push 5C110002h

The opcodes modify the stack

Addr	Stack data
x - 4 x	02 00 11 5C 0A 01 03 D2

Stack growth

Address increase

Start of the structure

Decompose into the structure fields

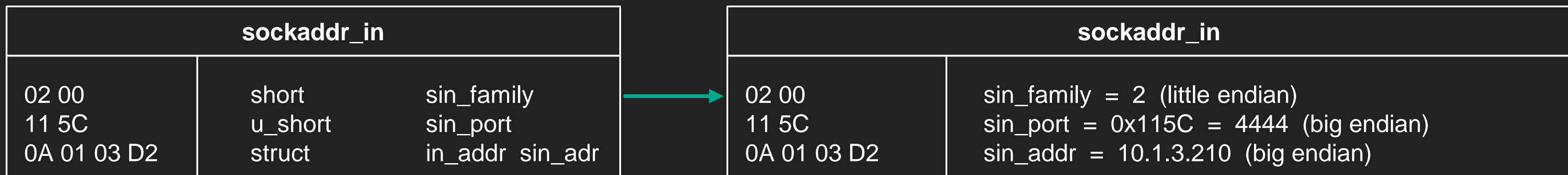
sockaddr_in		
02 00 11 5C 0A 01 03 D2	short u_short struct	sin_family sin_port in_addr sin_adr

Copy for Lee Wei Yeong

Shell - Solution for exercise 2

Copy for Lee Wei Yeong

"The sockaddr structure varies depending on the protocol selected. Except for the *sin*_family* parameter, sockaddr contents are expressed in **network byte order.**"



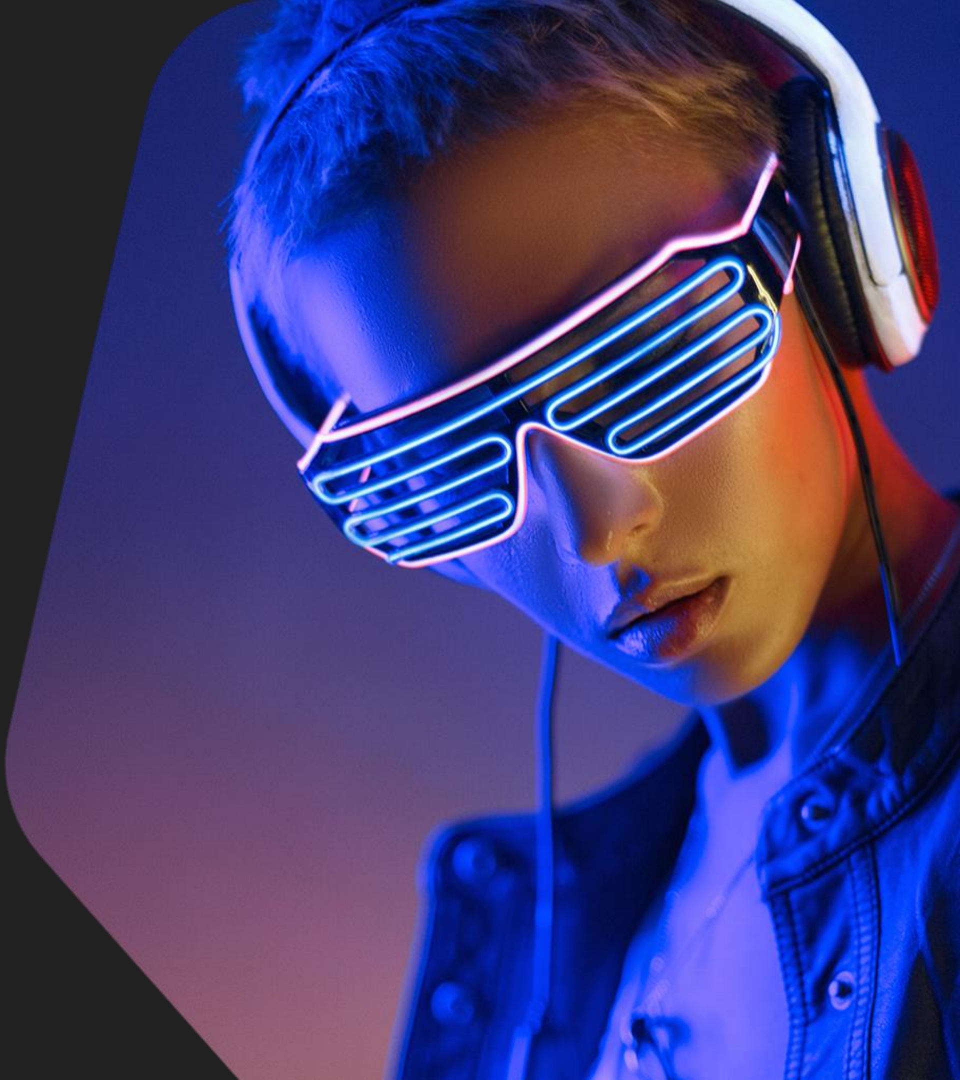
MSDN links:

https://docs.microsoft.com/en-us/windows/win32/winsock/sockaddr-2

Copy for Lee Wei Yeong

Track 3

Msfvenom



kaspersky

Msfvenom - Overview

Copy for Lee Wei Yeong

In this track you will practice:

- Analyzing Powershell scripts
- Decoding Msfvenom (Metasploit) payloads
- Manual reconstruction of data structures

Msfvenom - Mission briefing

Initial context:

- During the IR activities in the bank (track Shell) you discover traces of lateral movement
- The attackers installed a service remotely
- Artifact: event log record (“service was created”)
- The command line for the service executes a Powershell script

Copy for Lee Wei Yeong

MSDN links:

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_pwsh?view=powershell-7.1

Copy for Lee Wei Yeong

Msfvenom - Conclusion

- Powershell wrappers and initial shellcode loaders are likely used for pivoting
- Knowing the inner workings of the shellcode allows automation
- Can be easily integrated in a general digital forensics / incident response, SOC workflow
- Automated extraction of IOCs in offline mode, on-premises, under NDA constraints

Copy for Lee Wei Yeong

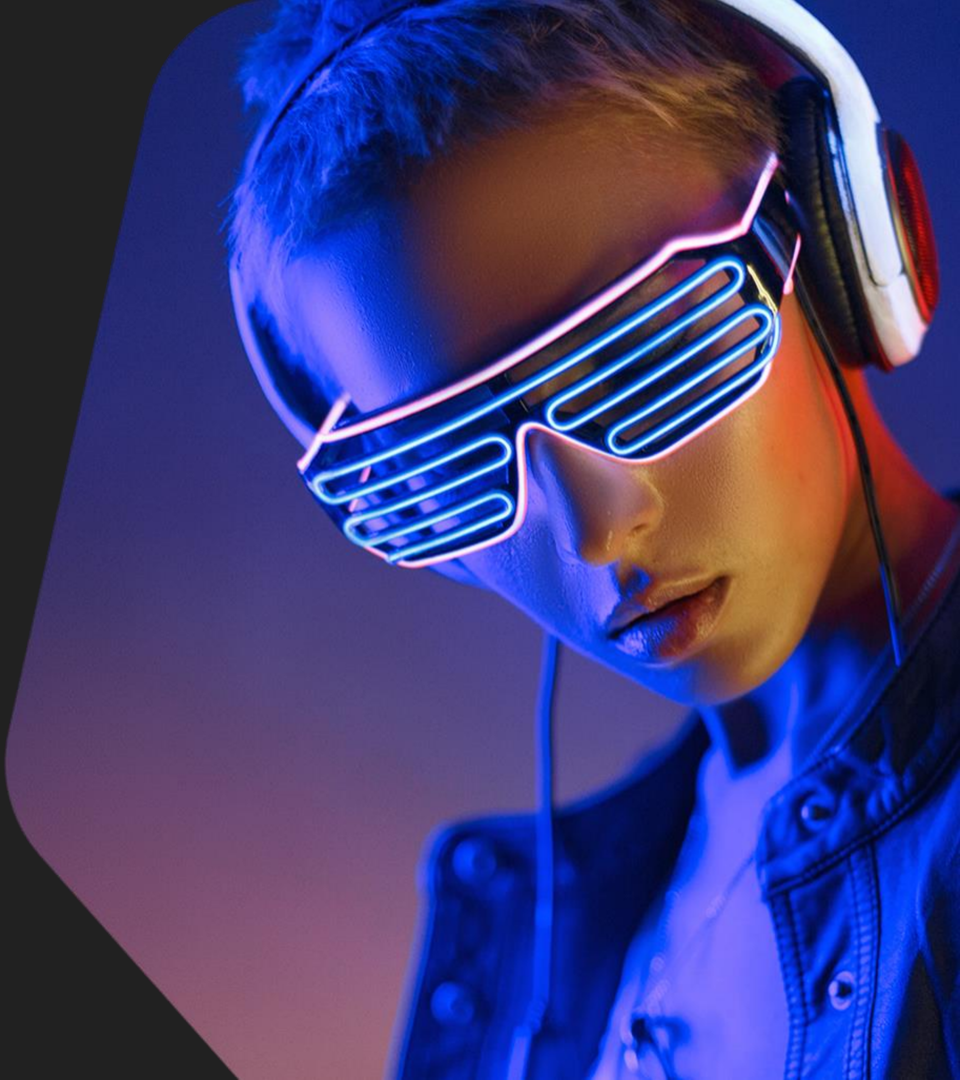
MSDN links:

https://docs.microsoft.com/en-us/windows/win32/winsock/sockaddr-2
https://docs.microsoft.com/en-us/windows/win32/api/winsock2/nf-winsock2-bind

Copy for Lee Wei Yeong

Track 4

Bangladesh GPCA



kaspersky

Bangladesh GPCA - Overview

Copy for Lee Wei Yeong

In this track you will practice:

- Code and data flow analysis
- Recognizing a well-known encryption algorithm
- Automating decryption with a decoding framework

Bangladesh GPCA - Mission briefing

Copy for Lee Wei Yeong

Initial context:

- The Bangladesh (central) bank heist, happened in 2016
- The attackers tried to transfer funds with SWIFT transactions
- Artifacts: nroff_b.exe (1d0e79feb6d7ed23eb1bf7f257ce4fee) and gpca.dat
- We need to decrypt gpca.dat, that is known to be used by nroff_b.exe

Bangladesh GPCA - Tracking the value

Copy for Lee Wei Yeong

Data flow: tracking values at various points in the program

"gpca.dat" - obvious string value to look for

- read/write/offset cross-references
- "what reads this value?"
- "what writes this value?"
- "what uses the address of this value?"

Code (control) flow: determining possible sequences of execution

- call/jump/offset cross-references
- "what calls this code?"
- "what jumps to this code?"
- "what uses the address of this code?"

Bangladesh GPCA - Key features to look for

Copy for Lee Wei Yeong

- Most important: what is the algorithm?
    - homebrew XOR, SUB, ADD
    - well-known ciphers: DES, AES, RC4, RC5, RC6, Salsa20, ChaCha20, RSA, Tea, Xtea, XXtea
- Key
- IV
- Padding
- Magic numbers?

Bangladesh GPCA - How to recognize ciphers

Copy for Lee Wei Yeong

- Specific constant values
  - DES, AES, Camellia: **S-Boxes**
  - RC5, RC6: P32=0xb7e15163, Q32=0x9e3779b9
- Specific code sequences
  - XTea: specific constant + algorithm
  - RC4: algorithm only

Bangladesh GPCA - Why code?

Preserve the decryption sequence

- Easily searchable artifacts:
  - code of the homebrew algorithm
  - magic constants
  - specific key / data constraints
  - custom data and executable formats
- Reproducible analysis
- Retargetable for similar samples
- Code can be migrated to tools, products

Copy for Lee Wei Yeong

Bangladesh GPCA - Why automate?

- For every file...

```
with open('input', 'rb') as f:
    data = f.read()
    data = decrypt(data)
    with open('output', 'wb') as f2:
        f2.write(data)
```

- Format parsers (PE, ELF, Mach-O, Golang, etc.)
- Code analysis

Copy for Lee Wei Yeong

Bangladesh GPCA - Conclusion

- Decoders preserve your analysis
- Can be reproduced and adapted for new samples
- Well-known ciphers can be easily recognized
- Use a framework to automate routine tasks
- Use off-the-shelf crypto (compression, etc.) libraries

Copy for Lee Wei Yeong

Track 5

Regin driver

kaspersky

Regin driver - Overview

Copy for Lee Wei Yeong

In this track you will practice:

- Analyzing a homebrew crypto algorithm
- Raw offset - virtual address conversions
- Automating decryption of PE files

Regin driver - Mission briefing

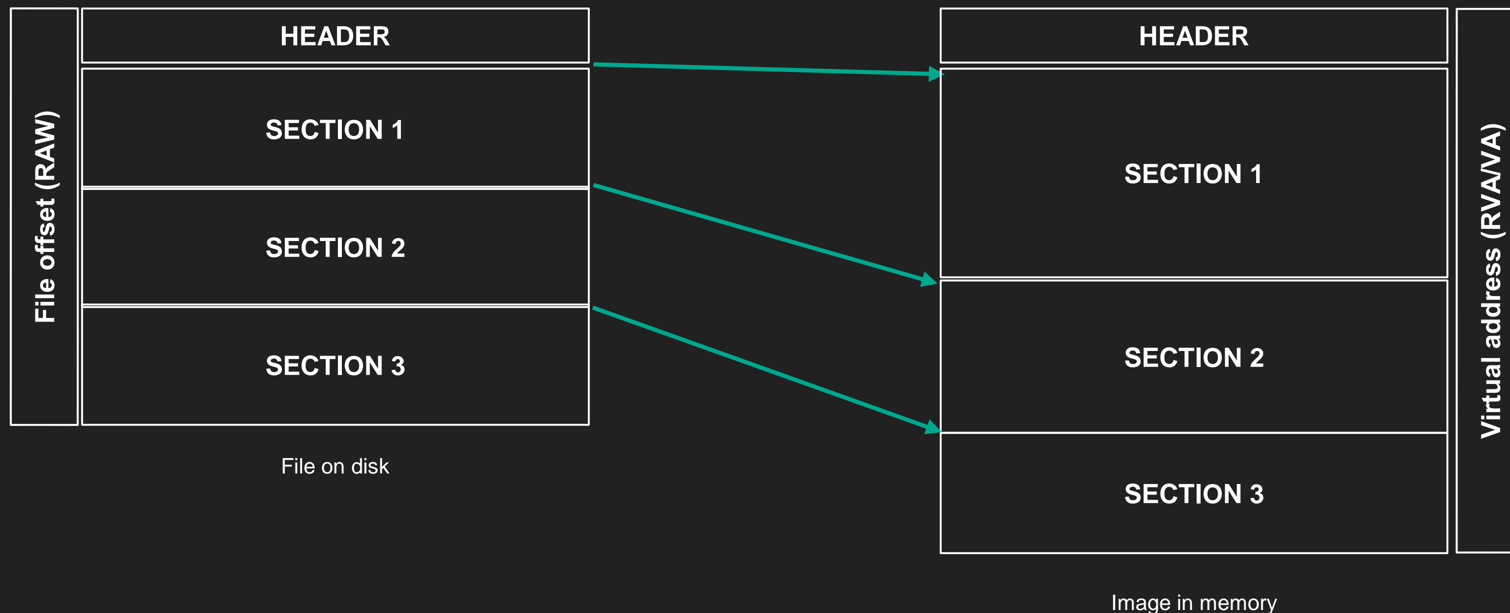
Copy for Lee Wei Yeong

Initial context:

- Regin, a sophisticated APT platform
- The driver is the first stage, just a loader
- The location of the next stage is unique and encrypted within the driver's body
- There are dozens of such samples, need to decrypt them all

Regin driver - Virtual addresses and file offsets

Copy for Lee Wei Yeong



kaspersky

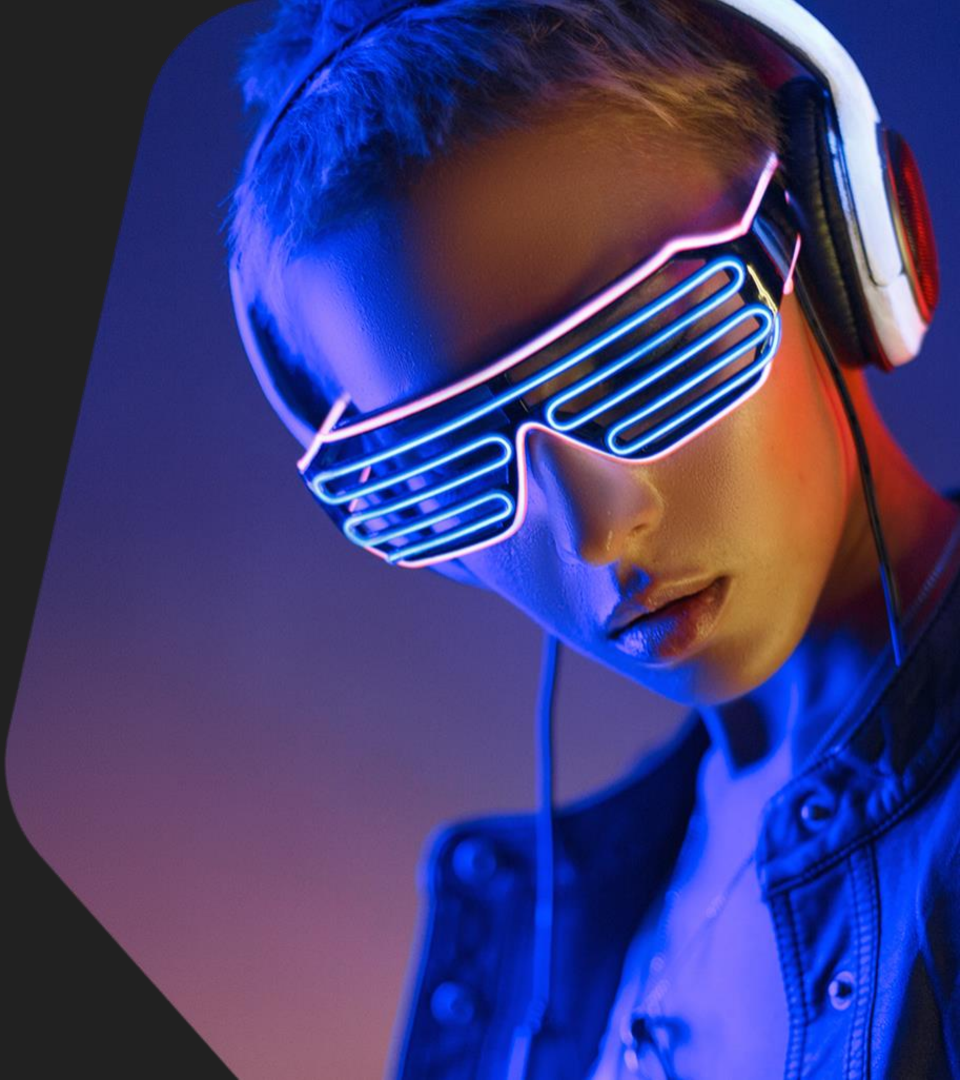
Regin driver - Conclusion

- Typical example: data is encrypted in a contiguous block
- Decryption is usually required for the analysis
- Automation enables scaling

Copy for Lee Wei Yeong

Track 6

Decrypt strings



kaspersky

Decrypt strings - Overview

Copy for Lee Wei Yeong

In this track you will practice:

- Analyzing a homebrew crypto algorithm
- Automating decryption of Mach-O files
- Processing multiple encrypted strings, referenced as function arguments

Decrypt strings - Mission briefing

Copy for Lee Wei Yeong

Initial context:

- Sofacy, a sophisticated APT platform
- Mach-O 64-bit x86_64 executable
- Need to process the sample for high-level analysis

Decrypt strings - Evolution of the tools

- **Single sample**: step with the debugger / create a tiny script / **manual** decryption

- **Many samples**: wrap common steps in a module, **reuse**

- **Many complicated samples**: upgrade to a static analysis **framework**

Copy for Lee Wei Yeong

Decrypt strings - Static analysis, automated

- Process common binary formats: PE, ELF, Mac-O, raw binary
- Use a disassembler to extract generic code flow information
  - call, jmp, j*, ret
- Light emulation: stack arguments, registers, simple arithmetic operations
  - push, pop, sub/add esp, mov, xor, …
- Handle key events
  - call to the decryption function
- Can be implemented with IDA python API, Ghidra as backends

Copy for Lee Wei Yeong

Copy for Lee Wei Yeong

Decrypt strings - Static analysis, an example

Event: call to the decryption function

```
call     Coder::decryptString(char const*,int)
```

Prerequisite: function arguments

```
lea      rsi, FILE_NAME
mov      edx, 8
```

Handler:

- extract the location and size of the string
- if the string is already decrypted, skip
- decrypt the string

Track 7

Driver

kaspersky

Driver - Overview

In this track you will practice:

- Processing encrypted strings
  - Preparing the sample for the analysis
- Applying structures, enumerations
- Re-creating a C++ class/structure
- In-depth reverse engineering of a sample

Copy for Lee Wei Yeong

Driver - Mission briefing

Copy for Lee Wei Yeong

Initial context:

- Equation, “the death star of malware galaxy”
- x86 Windows Native PE = driver
- Need a one-liner description of the sample
- All-in-one: pre-process first, then analyze

Driver - General workflow

1. Is the sample ready for analysis?

    Format: does the format require pre-processing (custom binary format, stripped headers, memory dump with relocations)?
    Code: is the code encrypted, compressed, encoded in any way?
    Data: are there encrypted entities (strings, resources, etc.)

1. Decode, decrypt, pre-process
    (manual, third-party tools, custom scripts)

1. Analyze the code. When more processing is required, **goto** 2.

Copy for Lee Wei Yeong

Driver - String decryption

Copy for Lee Wei Yeong

Look for:

- Unique code sequences that load addresses of encrypted data in **operands** (combinations of lea, push, mov, ..)
- Calls to functions that lead to decryption and get addresses of encrypted data as **arguments**
- **Tables** of virtual, relative or any other addresses of encrypted entities

Finding a way to extract the addresses is the key to automating the decryption.

Driver - Conclusion

Copy for Lee Wei Yeong

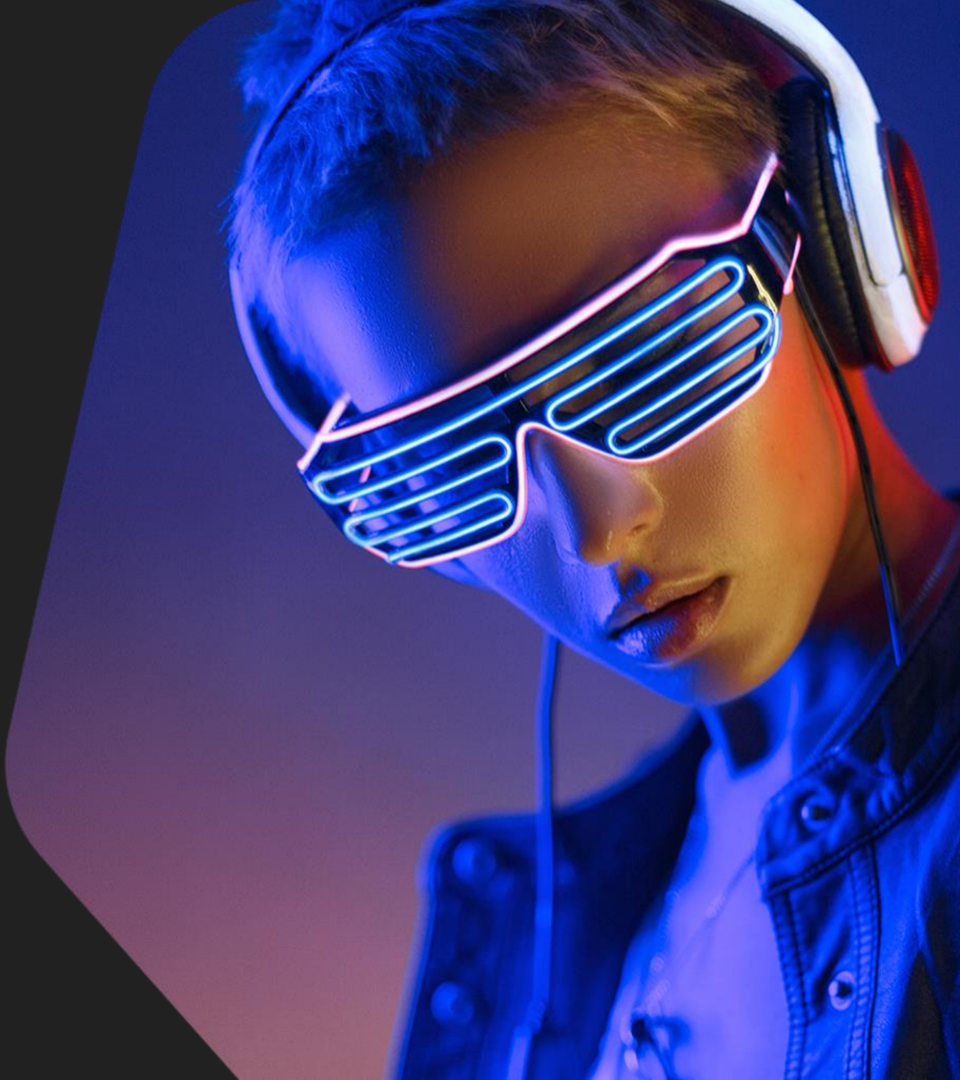
The driver injects a DLL, specified in its “Parameters\Excluded” registry value, or a default “msvcp73.dll”, in one of the running processes: “services.exe”, “lsass.exe”, “winlogon.exe”

Complete cycle of analysis:

- Processed encrypted entities with a decryption script
- Multiple iterations of “decrypt-reload-analyze”
- Reverse engineered the code to produce the description

Track 8

Miniduke



kaspersky

Miniduke - Overview

In this track you will practice:

- Processing a custom assembly-coded shellcode
- Extracting opcode information without a disassembler
- Reconstructing a custom API hashing algorithm
- Exporting information to IDA via an IDC script

Copy for Lee Wei Yeong

Miniduke - Mission briefing

Initial context:

- Miniduke
- x86 Windows shellcode, extracted from a PDF exploit
- Hand-written assembly
- Strings are already decrypted
- Need to resolve API hashes to enable analysis

Copy for Lee Wei Yeong

Miniduke - The plan

To resolve the API hashes, we need:

- Identify the code sequences that load the API hash values
- Find a way to extract the hash values from these code sequences
- Reconstruct the hashing function and pre-calculate the hashes for known names
- Resolve all the hash values discovered
- Transfer resolved function names into the IDB

Copy for Lee Wei Yeong

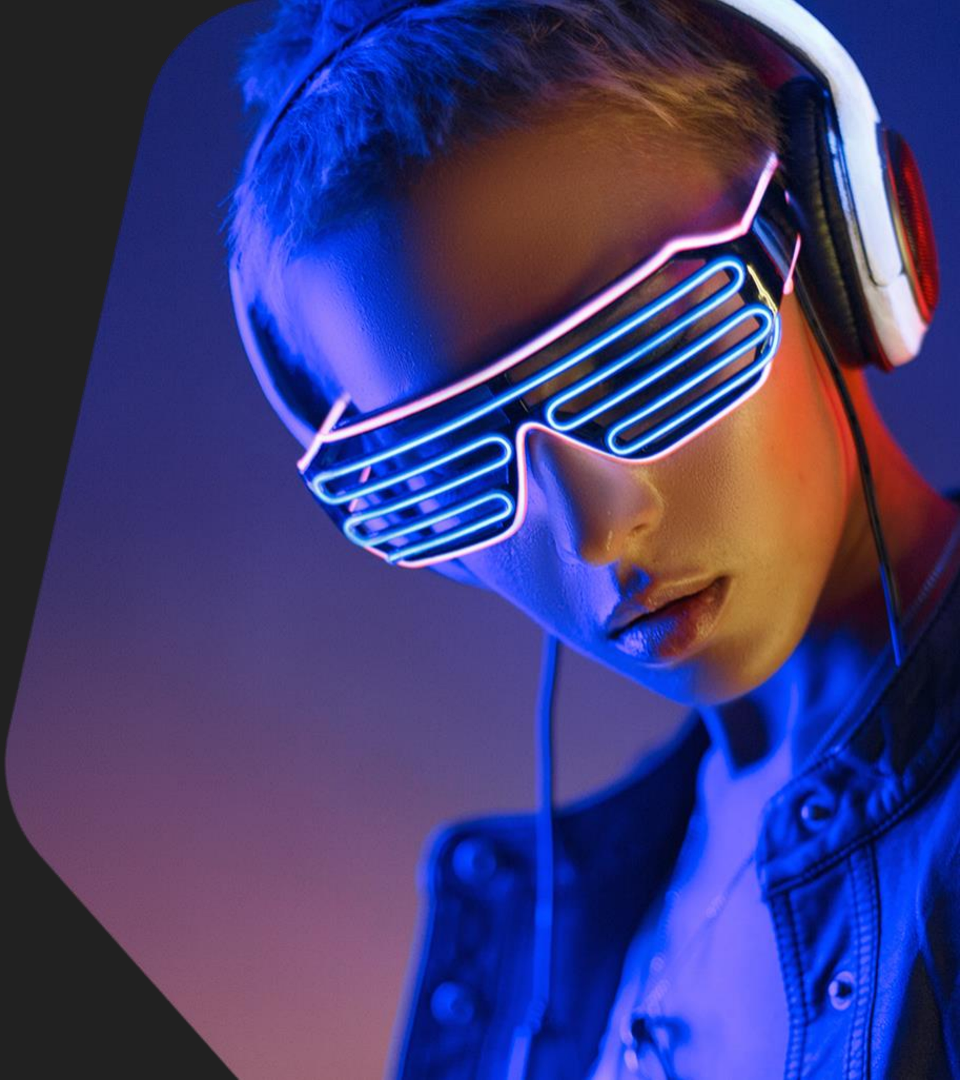
Miniduke - Conclusion

Copy for Lee Wei Yeong

- Where *clever* automation fails, simpler code may succeed
- Manual extraction of operands - know the format
- Reconstruction of a custom API hashing function
  - Hashes are good IOCs
- IDC is a universal medium for exporting to IDA

Track 9

Rocra



kaspersky

Rocra - Overview

Copy for Lee Wei Yeong

In this track you will practice:

- Extracting a binary payload from the RTF document
- Analyzing an exploit’s shellcode payload
- Extracting the final payload from the document

Rocra - Mission briefing

Initial context:

- Red October, a large scale APT operation
- Victims received a weaponized document
- CVE-2010-3333 pFragments vulnerability
- Need to analyze the exploit's code to extract the final payload

Copy for Lee Wei Yeong

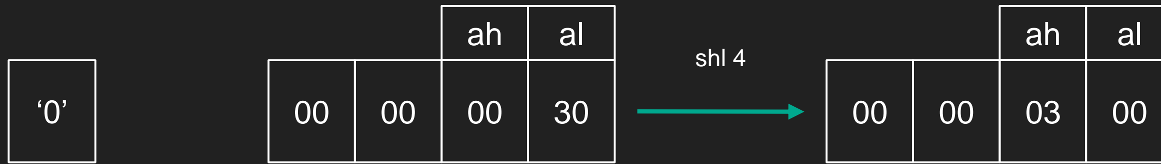
Rocra - RTF format

Basic rules:

- Control words (tags) are placed in groups, surrounded by {braces}
- Each control word starts with the **backslash symbol **
- Most of the binary data, except for the **\bin** control words, is stored as hexadecimal strings

Copy for Lee Wei Yeong

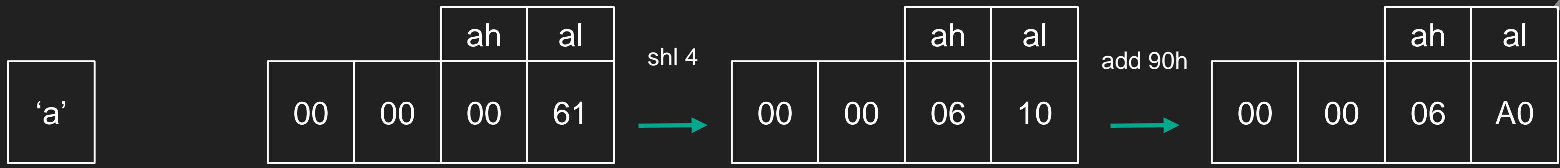
Rocra - Solution for exercise 2

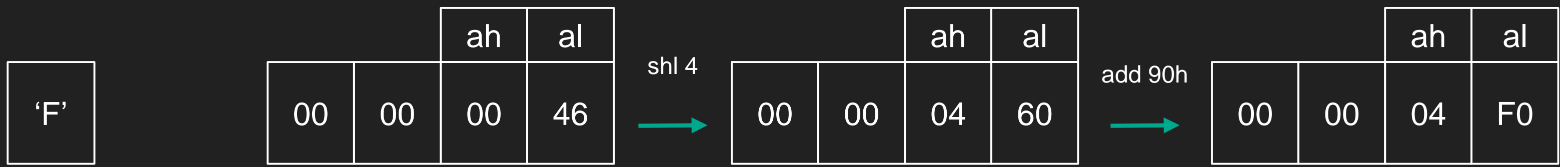


Copy for Lee Wei Yeong

Rocra - Solution for exercise 2

Copy for Lee Wei Yeong





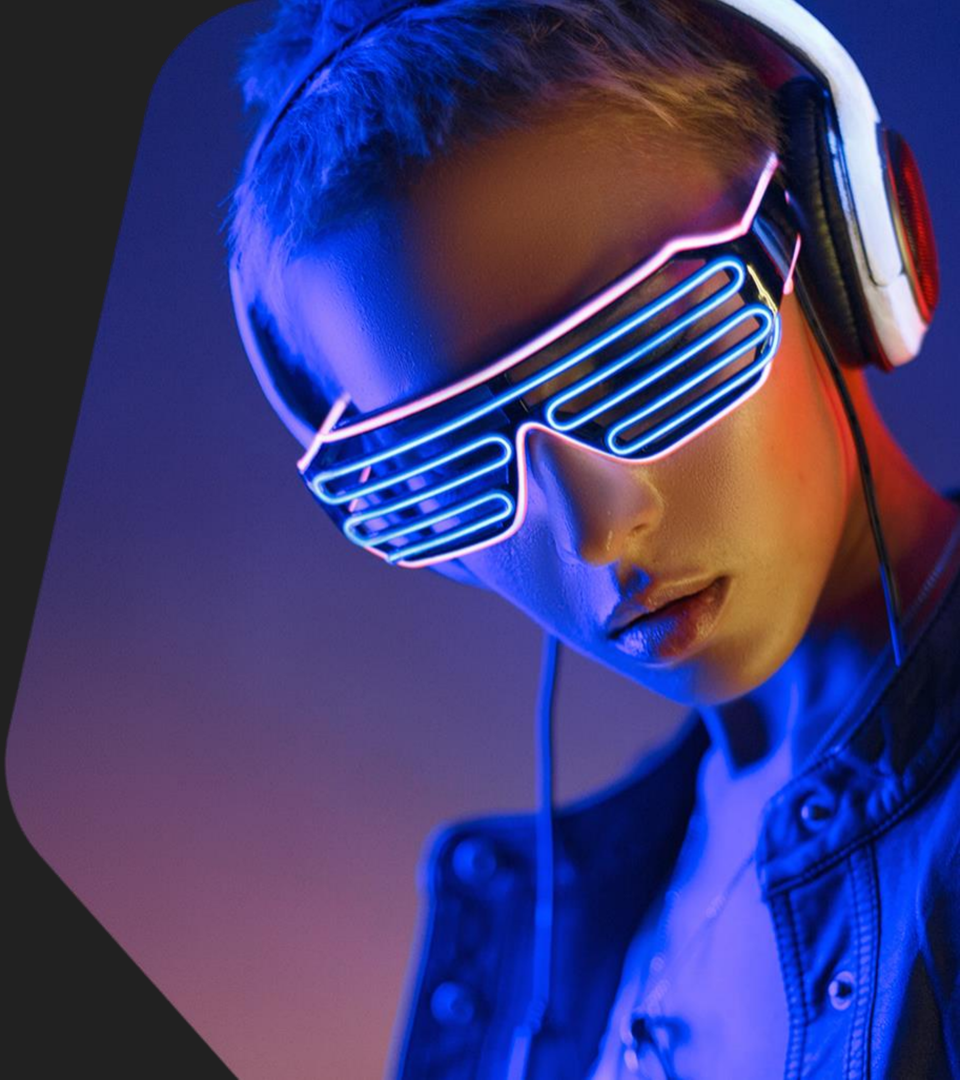
Rocra - Conclusion

- Extracting binary payloads from RTF files
- Exploit payloads are usually similar
- Analyzing the code allows to automate payload extraction
- API hash values and algorithm provide input to hunting and clustering

Copy for Lee Wei Yeong

Track 10

Cobalt



kaspersky

Cobalt - Overview

In this track you will practice:

- Using oletools to inspect an OLE2 container

Copy for Lee Wei Yeong

Cobalt - Mission briefing

Copy for Lee Wei Yeong

Initial context:

- Cobalt, a financially motivated APT group
- OLE2 file embedded in an RTF document
- CVE-2017-11882, Equation editor exploit
- Extract the payload

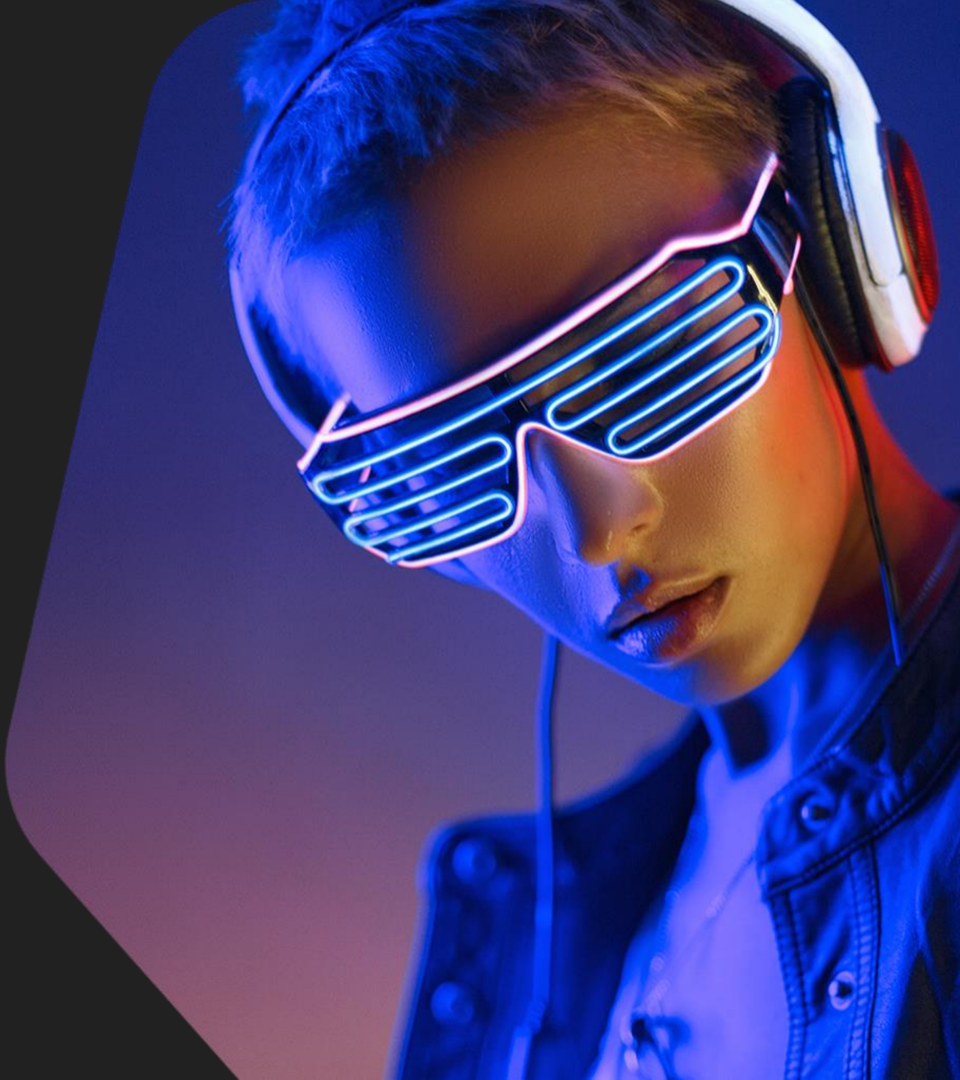
Cobalt - Conclusion

- RTF documents are used to deliver embedded payloads
- OLE2 objects require more complicated tools (oletools, for example)
- DOC, XLS, PPT, MSI and many other formats are OLE2-based

Copy for Lee Wei Yeong

Track 11

Cloud Atlas



kaspersky

Cloud Atlas - Overview

Copy for Lee Wei Yeong

In this track you will practice:

- Extracting binary data from a crafted RTF document
- Using oletools to inspect an OLE2 container
- Analyzing binary and scriptable (VBS) payloads

Cloud Atlas - Mission briefing

Initial context:

- Cloud Atlas, advanced cyber espionage actor
- Initial infection vector: malicious RTF document
- Multiple layers: binary payload, VBS wrapper
- Extract the payload

Copy for Lee Wei Yeong

Cloud Atlas - Conclusion

Copy for Lee Wei Yeong

- Multi-stage attacks are frequently used by malicious actors
- Dissecting all the stages may provide additional indicators
- External and internal IDA scripting can replace routine tasks
- Unique complicated RTF samples can be processed by hand

Track 12

Miniduke PDF

kaspersky

Miniduke PDF - Overview

In this track you will practice:

- Analyzing a malicious PDF document
- Inspecting a ROP-building Javascript
- Reconstructing a ROP chain

Copy for Lee Wei Yeong

Miniduke PDF - Mission briefing

Initial context:

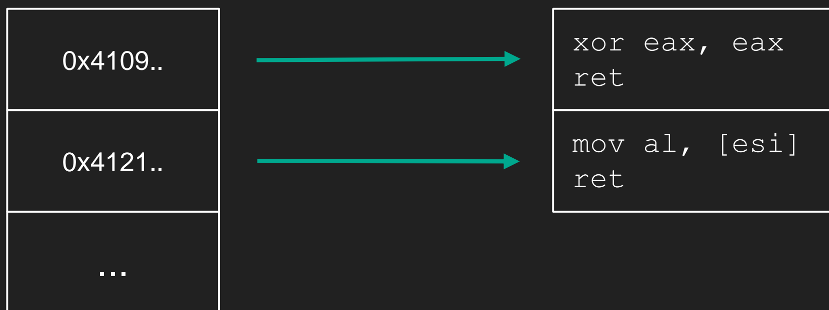
- Miniduke, the one from Track 08
- Initial infection vector: malicious PDF document
- No executable code, but a complete ROP chain
- Practice reconstructing the code execution chain

Copy for Lee Wei Yeong

Miniduke PDF - ROP chain

Copy for Lee Wei Yeong

- **R**eturn **O**riented **P**rogramming
- When the stack is not executable but still writable, you can change return addresses
- Set return addresses to tiny code sequences - “rop gadgets” - chained by a series or RETs
- Combine the gadgets to emulate the required business logic



kaspersky

Miniduke PDF - ROP chain

- ASLR was created to mitigate the risk, now the modules have random base addresses
- Leaked pointer into the module (i.e. data section) allows to bypass **ASLR**
- **No ASLR:** look for arrays of static return addresses into known image base of non-relocatable libraries
- **ASLR:** look for the code that builds such arrays using a computed image base value (our JS!)

Copy for Lee Wei Yeong

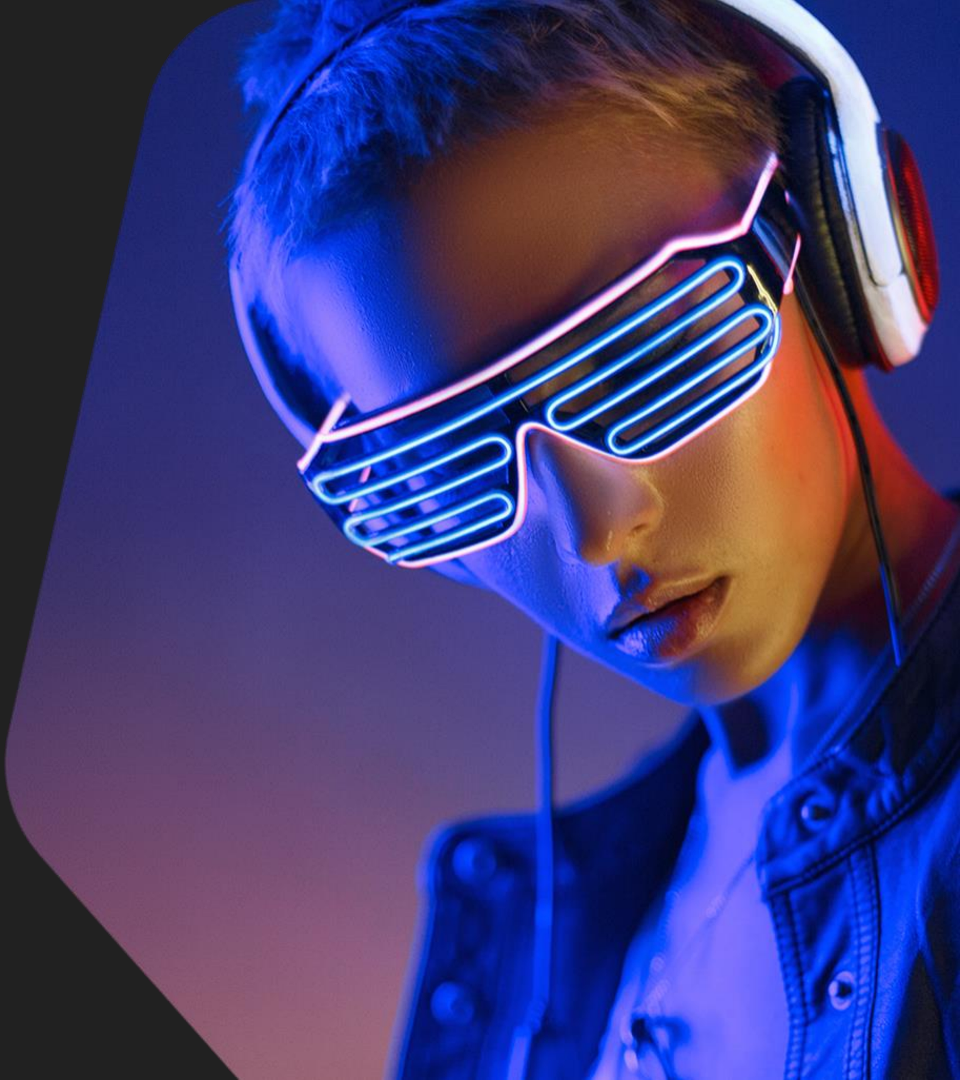
Miniduke PDF - Conclusion

- Analysis of PDF files
- Basics of return-oriented programming
- Extracting the ROP gadgets
- Reconstructing the ROP chain

Copy for Lee Wei Yeong

Track 13

Ragua Py2exe



kaspersky

Ragua - Overview

In this track you will practice:

- Extracting a py2exe binary
- Decompiling Python bytecode

Copy for Lee Wei Yeong

Ragua Py2exe - Mission briefing

Copy for Lee Wei Yeong

Initial context:

- Ragua, an extensive espionage framework in Python
- Binaries are Py2exe compiled bytecode
- Need to extract the bytecode (.pyc)
- Then, need to decompile the bytecode

Ragua Py2exe - Decompiling bytecode

- **Python**: uncompyle6, python-decompile3
- **.NET**: dnSpy, ILspy, .NET reflector
- **Java**: jad(old), Jeb decompiler
- **Visual Basic**: VB decompiler
- **Lua**: luadec

Copy for Lee Wei Yeong

Ragua Py2exe - Conclusion

- Bytecode: decompilers required
- Py2exe binaries can be extracted using the same Python version
- Resulting source code is almost identical to the original

Copy for Lee Wei Yeong

Track 14

Cridex

kaspersky

Cridex - Overview

In this track you will practice:

- Dynamically unpacking / decrypting Windows executables

Copy for Lee Wei Yeong

Cridex - Mission briefing

Copy for Lee Wei Yeong

Initial context:

- Major financial cyberthreat (later Dridex)
- Malicious attachments sent by e-mail
- Binaries are protected with a polymorphic layer
- Need to extract the payload

Cridex - Dynamic decoding: the plan

Copy for Lee Wei Yeong

Major types of protections (protection and/or compression):

- Replaces the current EXE image with the payload:
  - Set **write breakpoints** on major locations: MZ/PE **header**, start of the code section
  - Trace the source of the memory that is copied there

- Allocates dynamic memory for the original image, updates the existing section:
  - Set code **execution breakpoints** on major image startup events:
    - user mode: **LoadLibraryA, GetProcAddress**
    - kernel mode: **PsCreateSystemThread**
  - Trace the caller's location and look for non-image, RWE and/or newly allocated memory

Cridex - The Disclaimer

Beware that after this exercise you will need to revert your virtual environment to the initial state and your progress on previous exercises in the virtual environment will not be saved.

Copy for Lee Wei Yeong

Cridex - Conclusion

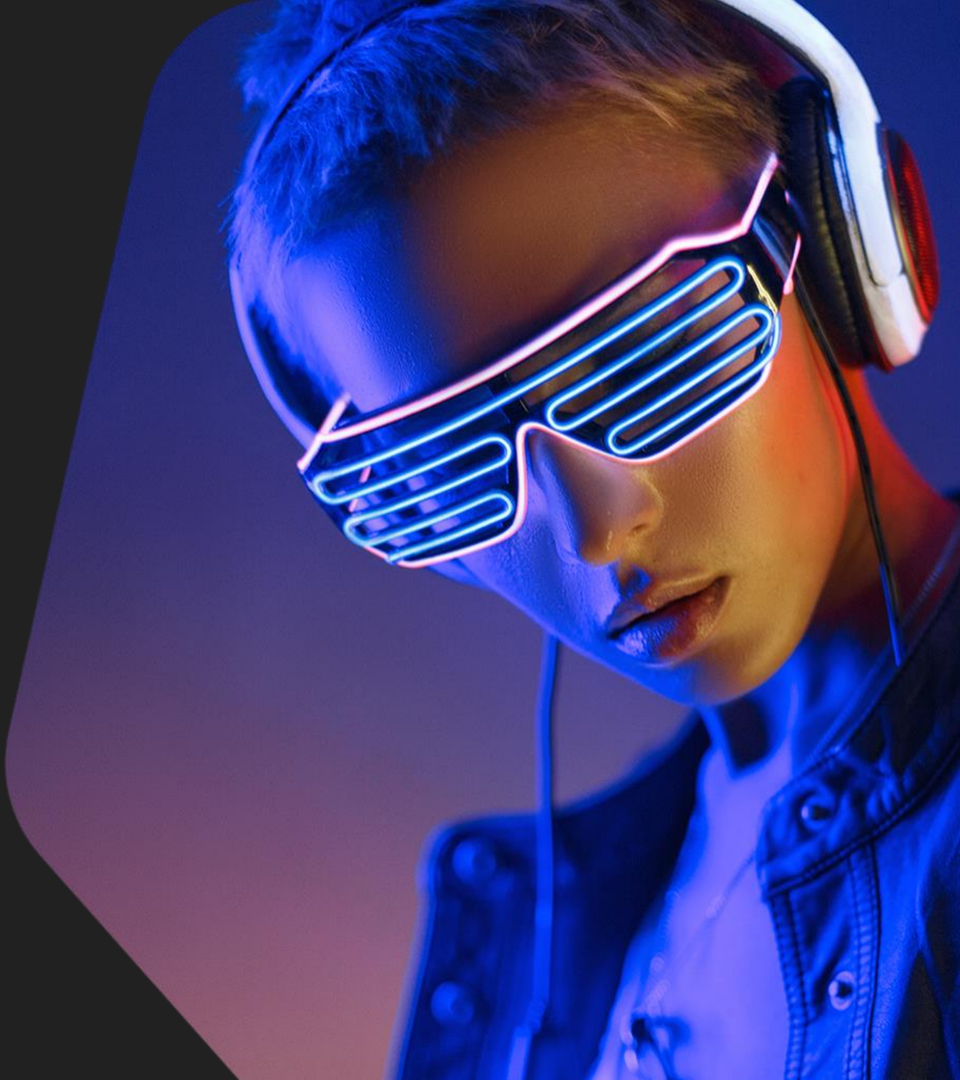
- Polymorphic machine-generated protections may be removed dynamically
- Most of the process can be automated in a debugger-based tool, debugger plugin or emulators
- Approach is universal, the key is in the correct breakpoint location

Copy for Lee Wei Yeong

Track 15

Carbanak

kaspersky



Carbanak - Overview

In this track you will practice:

- Analyzing and dynamically unpacking / decrypting Windows .NET executables

Copy for Lee Wei Yeong

Carbanak - Mission briefing

Copy for Lee Wei Yeong

Initial context:

- APT-style financially motivated actor
- Malicious samples were sent by e-mail
- Payload is wrapped in protective layers
- Need to extract the payload

Carbanak - The Disclaimer

Copy for Lee Wei Yeong

Beware that after this exercise you will need to revert your virtual environment to the initial state and your progress on previous exercises in the virtual environment will not be saved.

Carbanak - Conclusion

- Analyzing .NET code: “analyze”, cross-references
- Debugging multiple stages
- Loading order may need to be preserved

Copy for Lee Wei Yeong

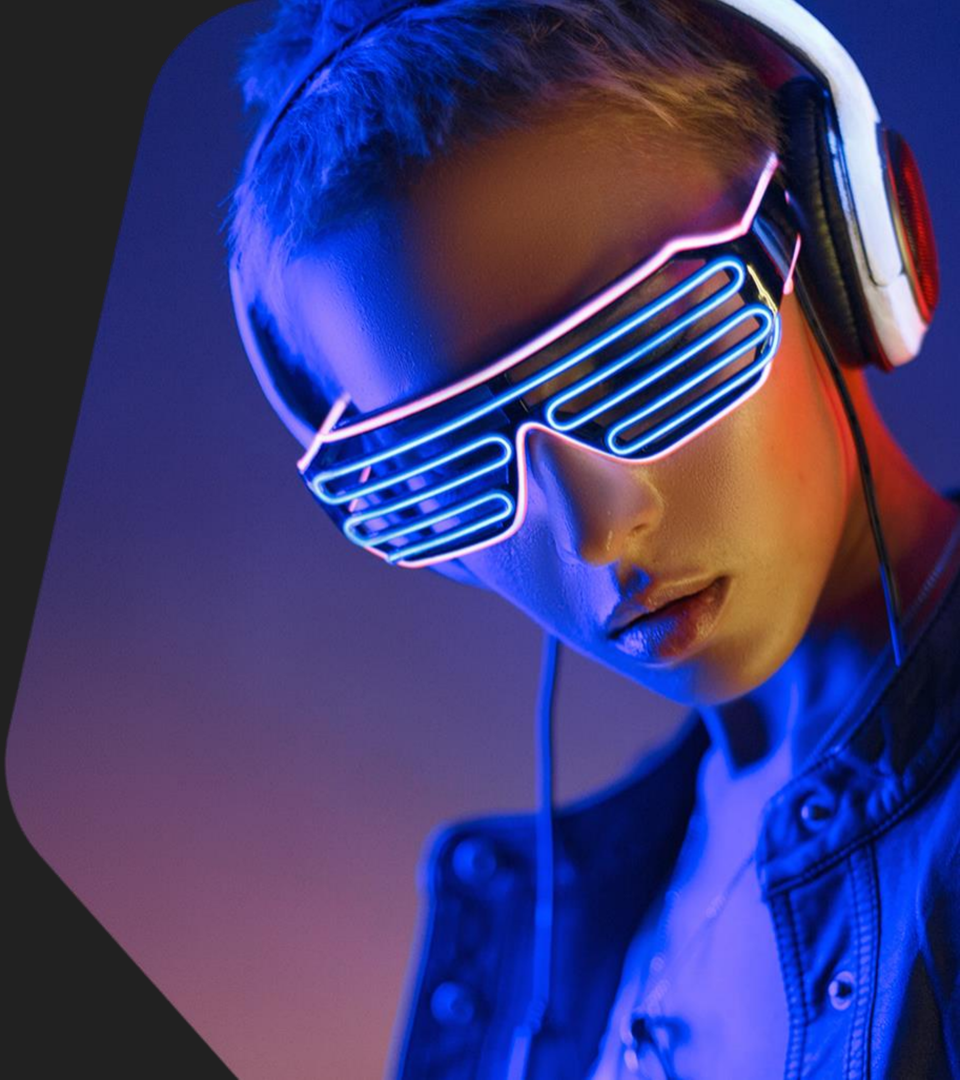
Carbanak - Conclusion

- Analyzing .NET code: “analyze”, cross-references
- Debugging multiple stages
- Loading order may need to be preserved

Copy for Lee Wei Yeong

Track 16

Snake



kaspersky

Snake - Overview

In this track you will practice:

- Analyzing Golang samples
- Mapping basic Golang structures
- Extracting and decrypting Golang string literals

Copy for Lee Wei Yeong

Snake - Mission briefing

Copy for Lee Wei Yeong

Initial context:

- Ransomware, used to attack industrial companies
- Samples are unique, generated for each victim
- Strings are encrypted
- Written in Golang
- Need to decrypt the strings

Snake - Reversing Golang

- Multiple return values

```
func Parse(layout, value string) (Time, error)
```

- Strings are condensed in a single buffer
- Arrays and [ ]slices of arbitrary types
- The runtime and compiler are open source
- IDA’s decompiler leads to more confusion

Copy for Lee Wei Yeong

Snake - Let’s **re**

- Python’s regular expressions can be applied to binary data
- Find the prolog instruction: **b“\x64\x8B\x0D\x14\x00\x00\x00”** (mov ecx, large fs:14h)
- Parentheses **()** form groups that can be addressed within each match
- Dots **.** match any byte, and **.{exact}** or **.{min,max}** allow to match/skip ranges of bytes

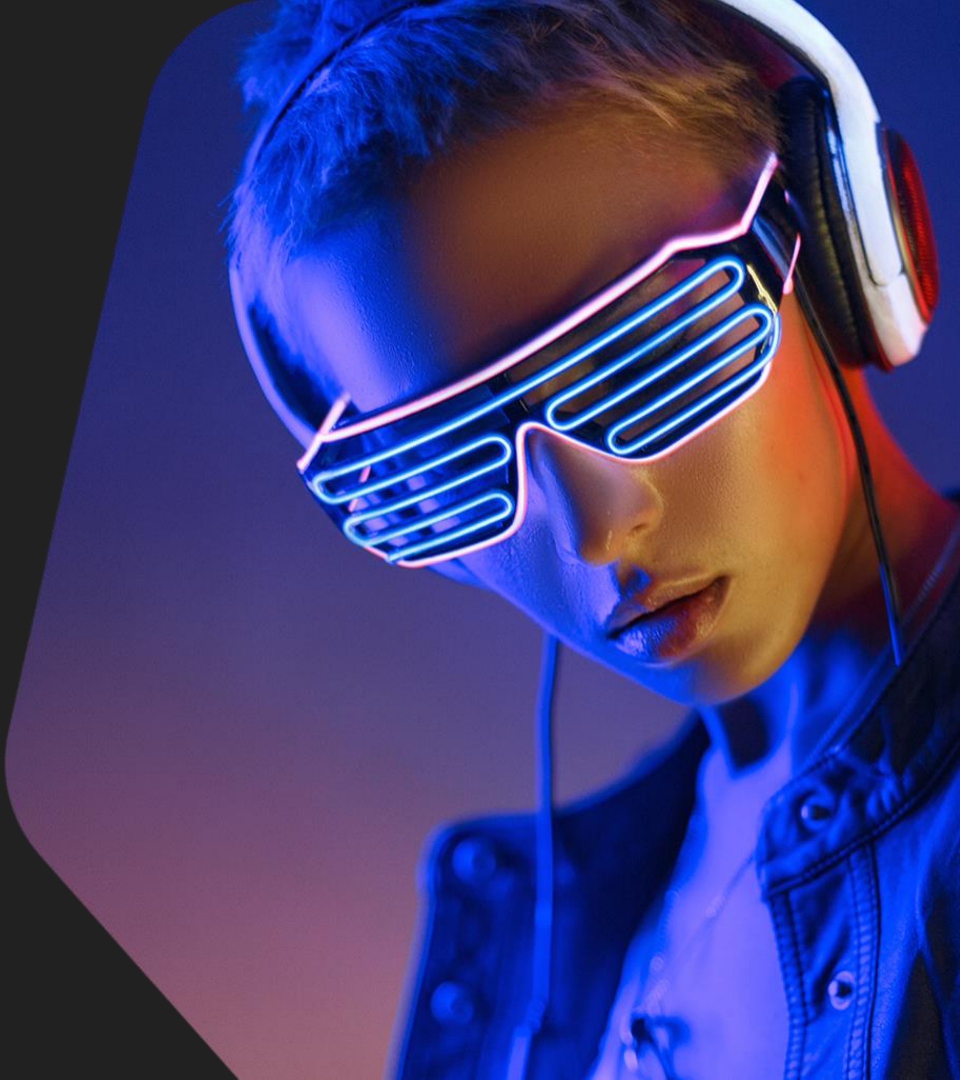
Copy for Lee Wei Yeong

Snake - Conclusion

- Basic Golang runtime structures
- String slices
- Addressing multiple return values
- Hard to analyze without auto-analysis

Copy for Lee Wei Yeong

The end



kaspersky