black hat USA 2021
August 4-5, 2021
BRIEFINGS

Rope: Bypassing Behavioral Detection of Malware with Distributed ROP-Driven Execution

Daniele Cono D'Elia, Lorenzo Invidia

WHO ARE WE

- Post-doc @ Sapienza
- Software and systems security
- A few Black Hat talks on malware

@dcdelia

- MSc graduate @ Sapienza
- Windows internals and reversing

@mattless_

SAPIENZA
UNIVERSITÀ DI ROMA

MALWARE DETECTION

- Flag untrusted software as malicious on end machines
- AV/EDR solutions rely on behavioral analyses to forestall new threats. What are the limits of current approaches?

Workflow
- monitor execution units
- match actions against «dynamic» signatures
- raise an alert

IN THIS TALK

- What we did
- Rope concept
- Prototype (+ new bypasses)
- Results
- Outlook

BEHAVIORAL 101

Approach

- attempt initial controlled execution
- monitor once running unleashed

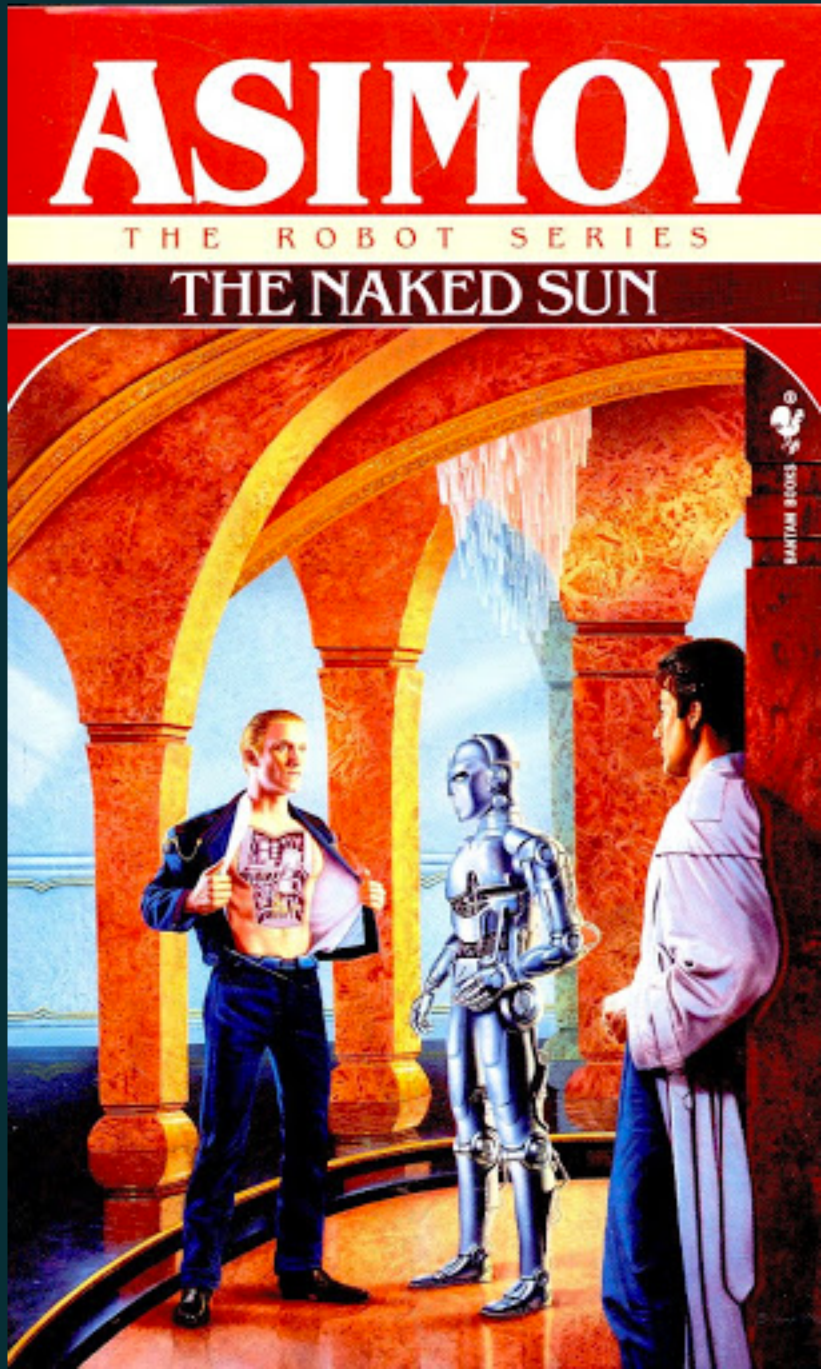
HOW? -> user-space hooks, mini-filters

WHO? -> process (w/ children)

DISTRIBUTED MALWARE

IDEA

- dilute temporal and spatial footprint
- multiple cooperating entities
- and no single entity alerts AV/EDRs!



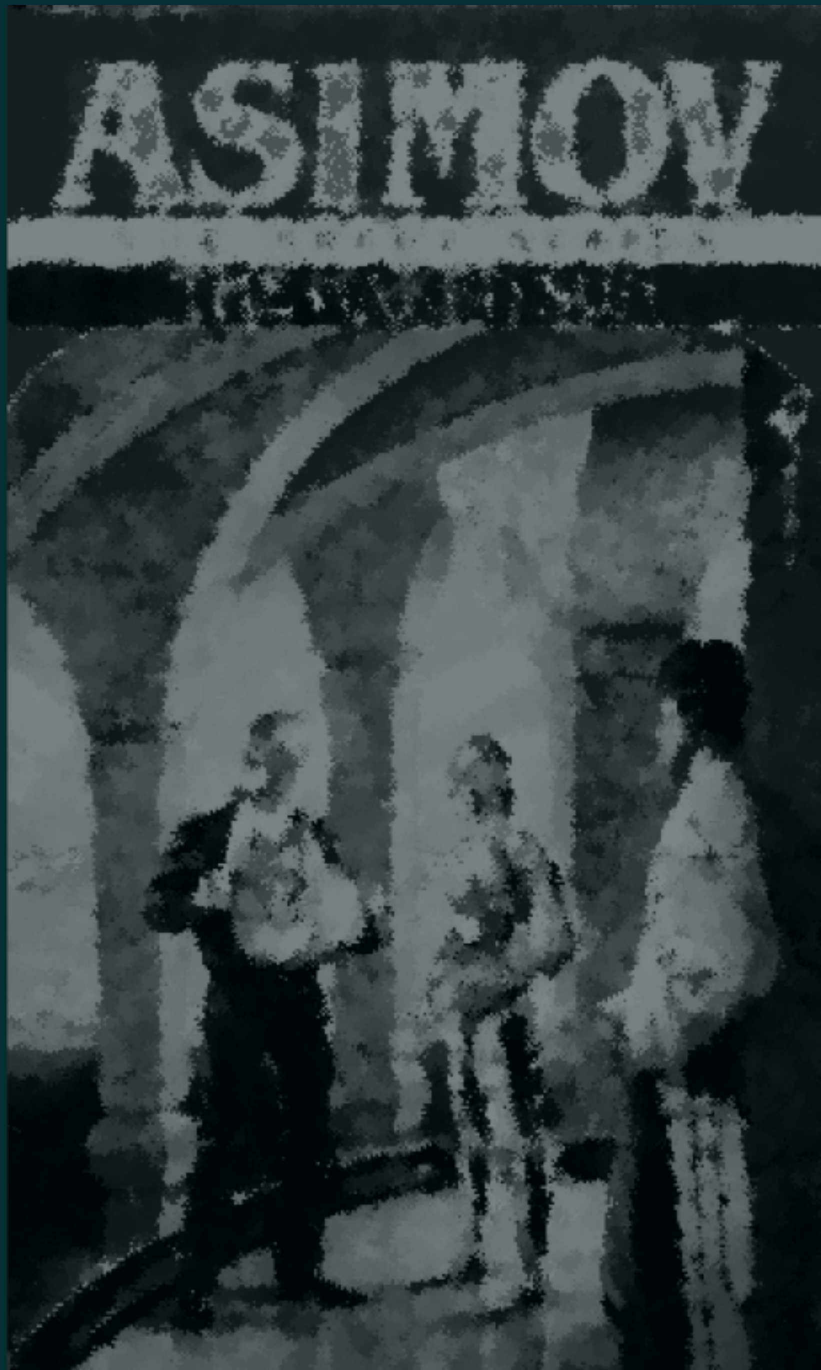
DISTRIBUTED MALWARE

Create ad-hoc processes?

- ✗ very high number required
- ✗ correlation is easy

Abuse existing processes?

- ✗ injecting code is noisy
- ✗ conspicuous regions



HARDENING MITIGATIONS

Windows now offers means for applications to
«reduce the attack surface against next-generation malware»

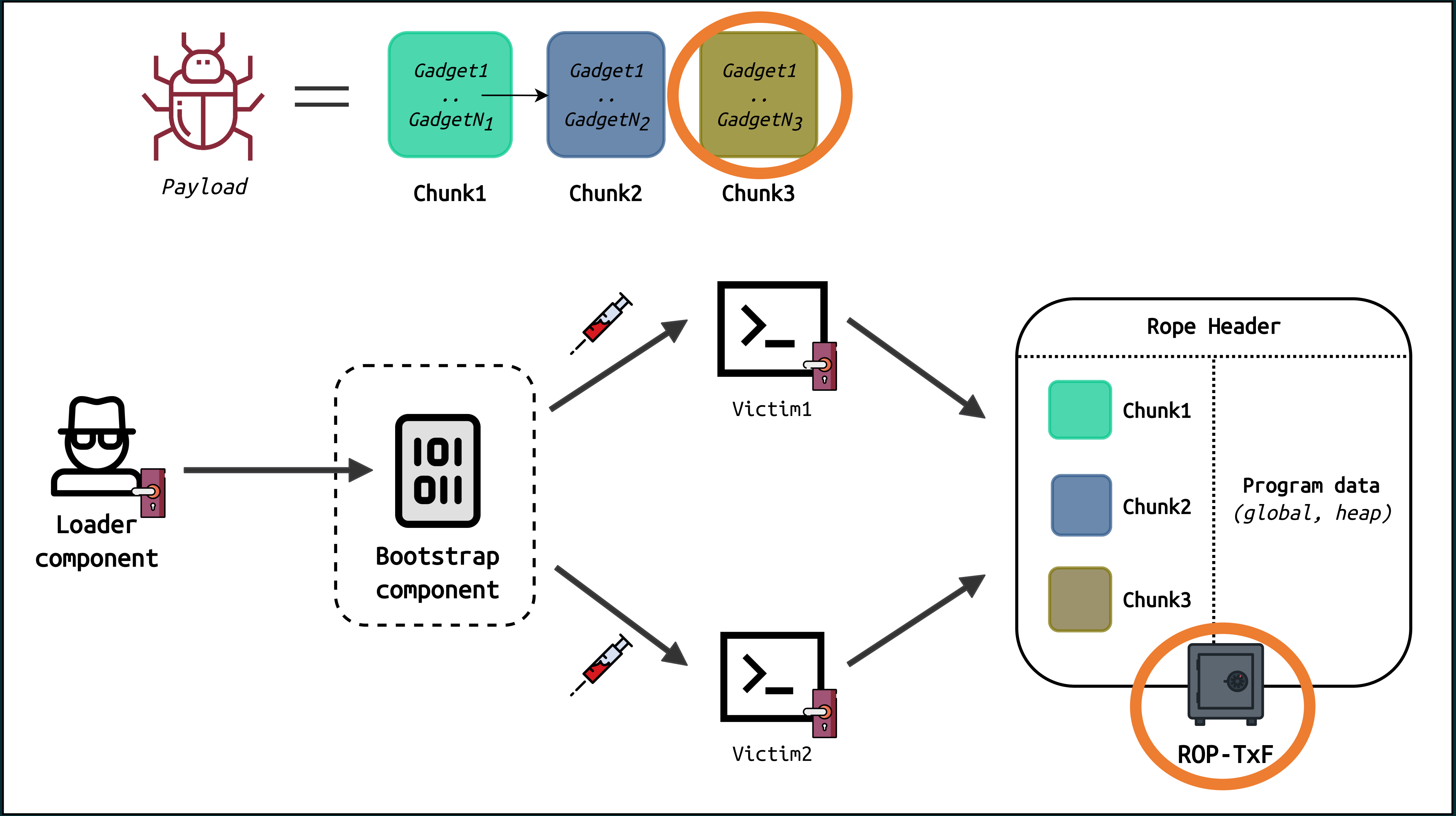
WINDOWS DEFENDER EXPLOIT GUARD

- Arbitrary Code Guard (ACG)
- Code Integrity Guard (CIG)
- Export & Import Address Filtering (EAF, IAF)
- and more...

DESIRED PROPERTIES

- ✓ flexible delivery of payload
- ✓ small footprint of distributed runtime
- ✓ comply with hardening mitigations
- ✓ keep code and data hidden as much as possible

WHAT WE DID



Payload
=
Gadget1
..
GadgetN1
Gadget1
..
GadgetN2
Gadget1
..
GadgetN3
Chunk1
Chunk2
Chunk3
Loader component
Bootstrap component
Victim1
Victim2
Rope Header
Chunk1
Chunk2
Chunk3
Program data (global, heap)
ROP-TxF

WHAT WE DID

Key #1: ***Return-Oriented Programming***

- encode distributed payload
- get around WDEG mitigations

Key #2: ***Transactional NTFS***

- non-inspectable covert channel
- payload sharing + communications

ROPE: distributed, ROP-driven Execution

DESIGN OF ROPE

Goals

- encode distributed payload
- get around WDEG mitigations

With code reuse we avoid any RWX memory! We borrow ROP gadgets from a shared library that all victims have loaded...

DESIGN OF ROPE

Goals

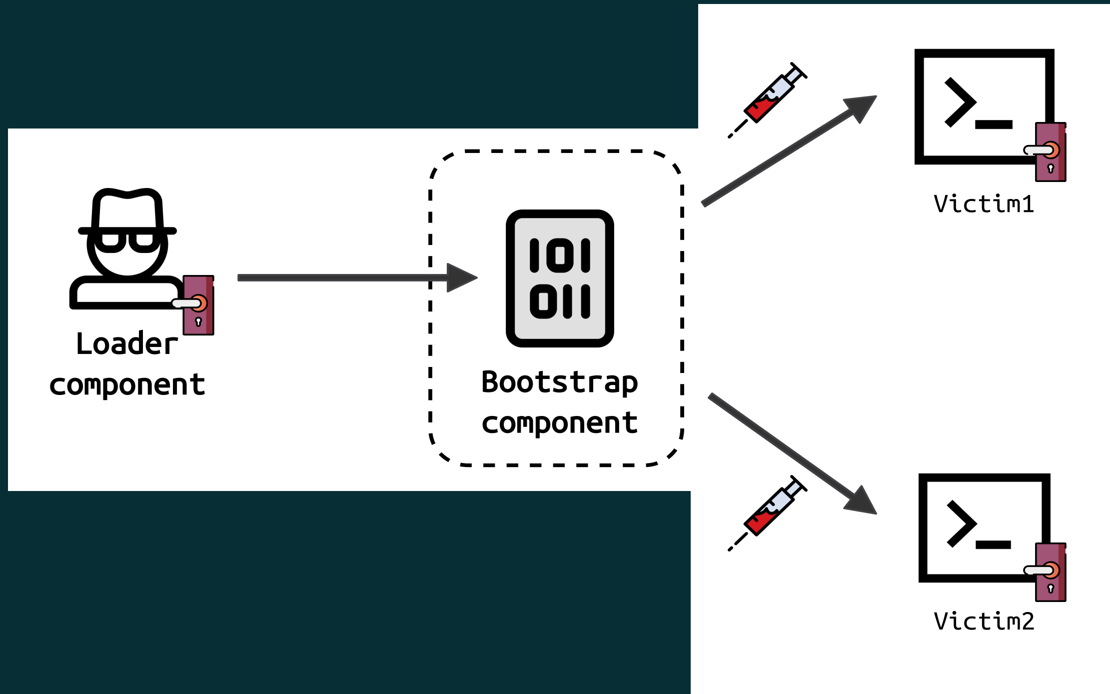
- non-inspectable covert channel
- payload sharing + communications

Thanks to TxF, only processes with the TxF handle can see the transient contents of the shared file. And ROP code is data!

ROPE: LOADER

TASKS

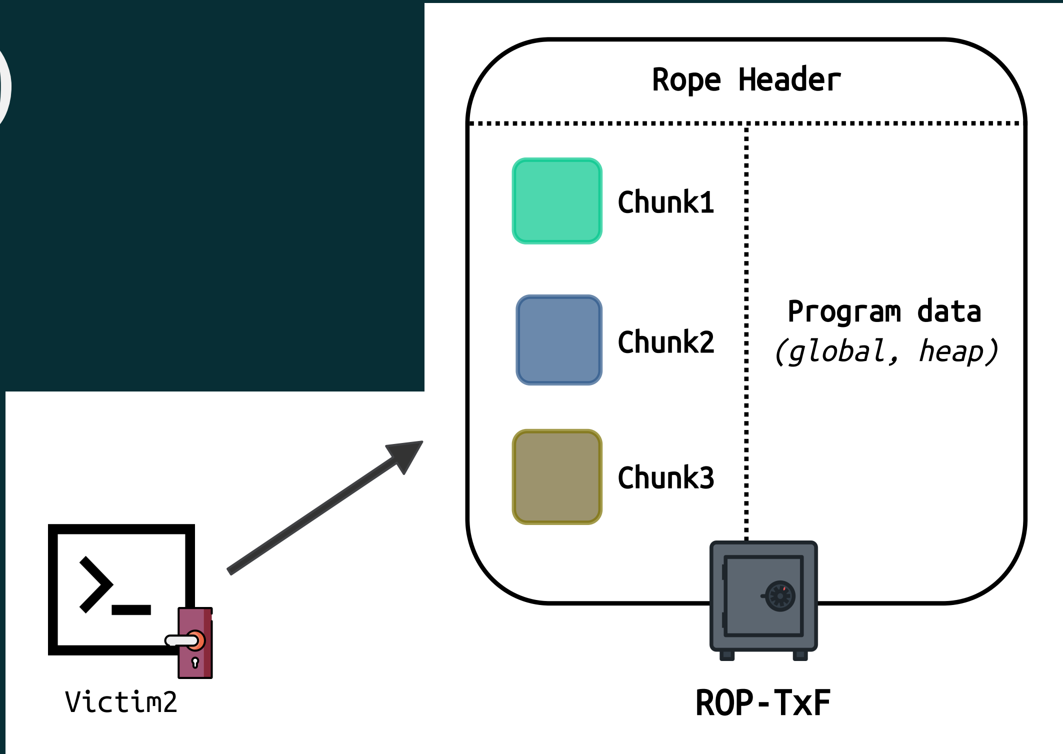
- pick victim processes
- create ROP-TxF on some file
- clear it, then fill with chains & metadata
- duplicate TxF handle for victims
- inject bootstrap component



ROPE: BOOTSTRAP

TASKS

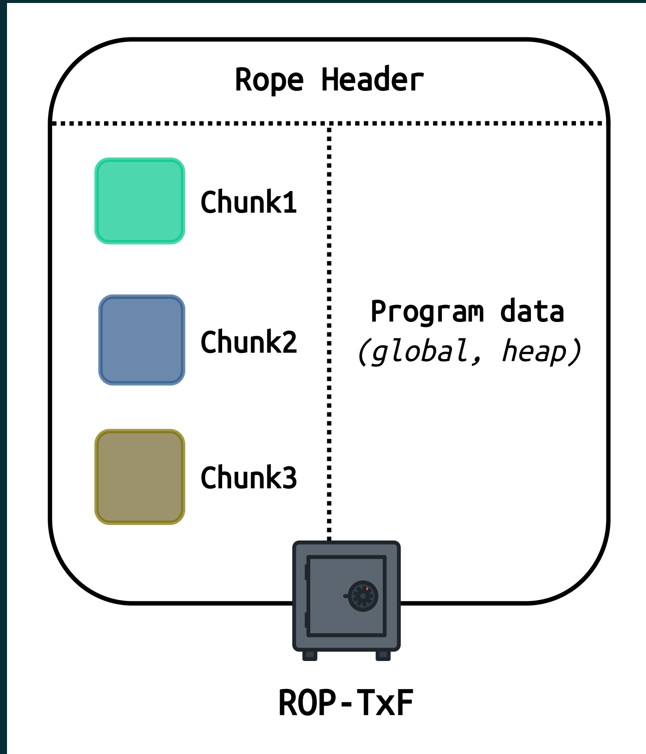
- make victim load ROP-TxF
- schedule execution of ROP code
- solve needed APIs covertly
- coordinate with other victims (if needed)



ROPE: ROP-TxF

Structure & contents

- ROP payload arranged in chunks
- a victim executes one or more chunks
- ROP-TxF hosts:
  - ➢ chunks + program memory
  - ➢ metadata for runtime (e.g., APIs, handles)



ROPE: EXECUTION

Mode 1: continuous

- any victim can execute any chunk
- Rope brings explicit coordination for chunks

Mode 2: staged

- sequences of chunks run by specific victims
- coordination may be also external

ADVANTAGES OF ROPE

- ✓ no need to allocate/modify executable memory
- ✓ in-memory inspection harder for AV/EDRs (ROP adds indirection)
- ✓ single shared medium for code and data
- ✓ compliance with ACG & CIG

CHALLENGES

- ❑ inject the bootstrap component
- ❑ find suitable gadget source
- ❑ comply with ROP mitigations
- ❑ encode the payload
- ❑ look up APIs in hardened victim

CHALLENGES

- ❑ inject the bootstrap component (bypass #1)
- ❑ find suitable gadget source
- ❑ comply with ROP mitigations
- ❑ encode the payload
- ❑ look up APIs in hardened victim (bypass #2)

INJECTION STAGE

- We have to deliver the bootstrap component to victims
- And Rope also needs a shared source of gadgets...

Restrictions

- can only use/load signed modules
- cannot use RWX memory
- Rope runtime should not spook AV/EDRs

PHANTOM DLL HOLLOWING

```
HANDLE hSection, hFile, hTransaction;
NtCreateTransaction(&hTransaction)
hFile = CreateFileTransactedW(dllPath, ..., hTransaction)
< parse file for suitable insertion region >
WriteFile(hFile);
NtCreateSection(&hSection, ..., SEC_IMAGE, hFile);
NtMapViewOfSection(hSection, hVictimProcess, ...);
```

Alerts AV/EDRs!

PHANTOM DLL HOLLOWING

0xC0000428 (STATUS_INVALID_IMAGE_HASH) for NtCreateSection when CIG enabled…

BYPASS #1: ACG/CIG

1. create DLL-TxF with a Windows DLL
2. create Section on it
3. duplicate TxF-ed Section for victims
4. inject ROP chain on victim’s stack
   - map view of Section handle
   - yield control to desired address

BYPASS #1: ACG/CIG

The ROP Chain

- host CONTEXT for resuming victim's activities
- set up arguments for NtMapViewOfSection
- add RVA of entrypoint to base address from loading
- run the desired code
- upon return, call NtContinue with CONTEXT

INJECTION STAGE

- The bypass just brought multiple advantages:
  - ✓ we can add gadgets to DLL-TxF
  - ✓ bootstrap component in DLL-TxF (as ROP chain or shellcode)
  - ✓ victim will spawn payload with own means (no remote threads)

- Rope can work with other injection primitives. Our bypass just offers an implementation shortcut...

CHALLENGES

- ✓ inject the bootstrap component (bypass #1)
- ✓ find suitable gadget source
- ❑ comply with ROP mitigations
- ❑ encode the payload
- ❑ look up APIs in hardened victim (bypass #2)

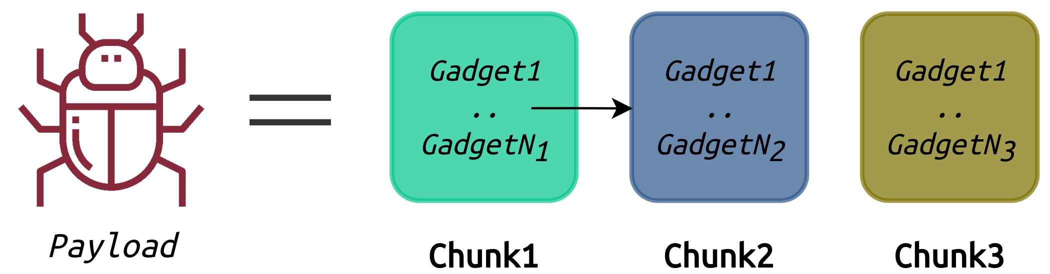
ROP MITIGATIONS

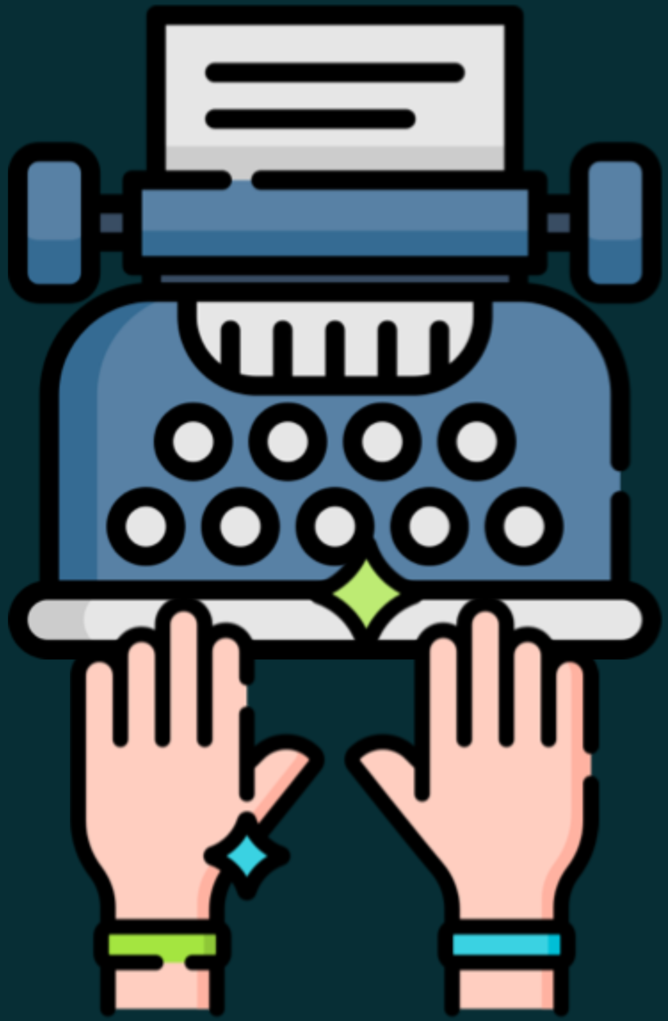
Rope chunks use standard means against WDEG

- ➢ StackPivot => make API calls from native stack
- ➢ CallerCheck & SimExec (32-bit)
  - gadgets that break analyses (Németh'15, Borrello'19)
  - Rite of Passage (Yair @ DEF CON 27)
  - issue calls from shellcode

As for the injection, WDEG ignores NtMapViewOfSection...

ROP ENCODING

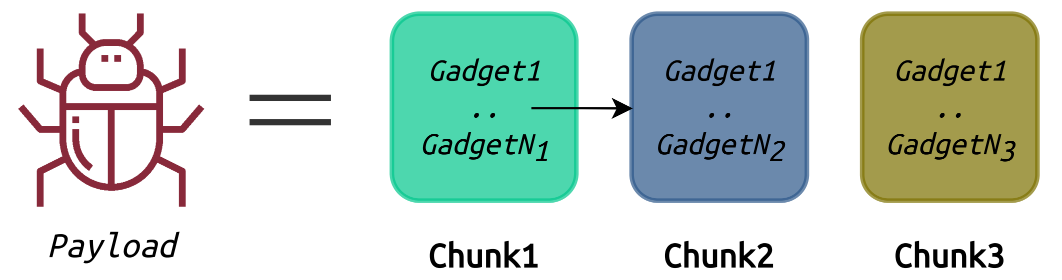


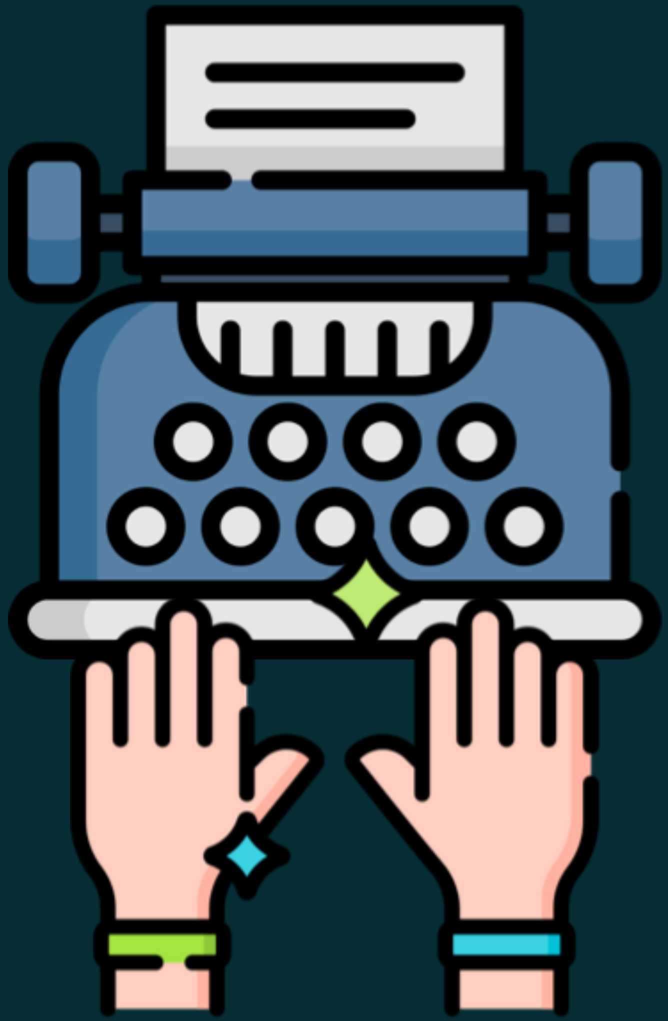


Some automation?

- ✗ manual writing doesn’t scale
- ✗ ROP tools meant for exploits

ROP ENCODING





Some automation!

1. promote stack variables to globals
2. globals as fields of a single struct
3. MSVC with optimization/canaries off

Output resembles a shellcode. Delimit chunks as basic blocks, look up gadgets, produce a chain skeleton...

Future work: use raindrop (DSN'21) for fully automated ROP binary rewriting
(«Hiding in the particles: When return-oriented programming meets program obfuscation»)

CHALLENGES

- ✓ inject the bootstrap component (bypass #1)
- ✓ find suitable gadget source
- ✓ comply with ROP mitigations
- ✓ encode the payload
- ❑ look up APIs in hardened victim (bypass #2)

API LOOKUP

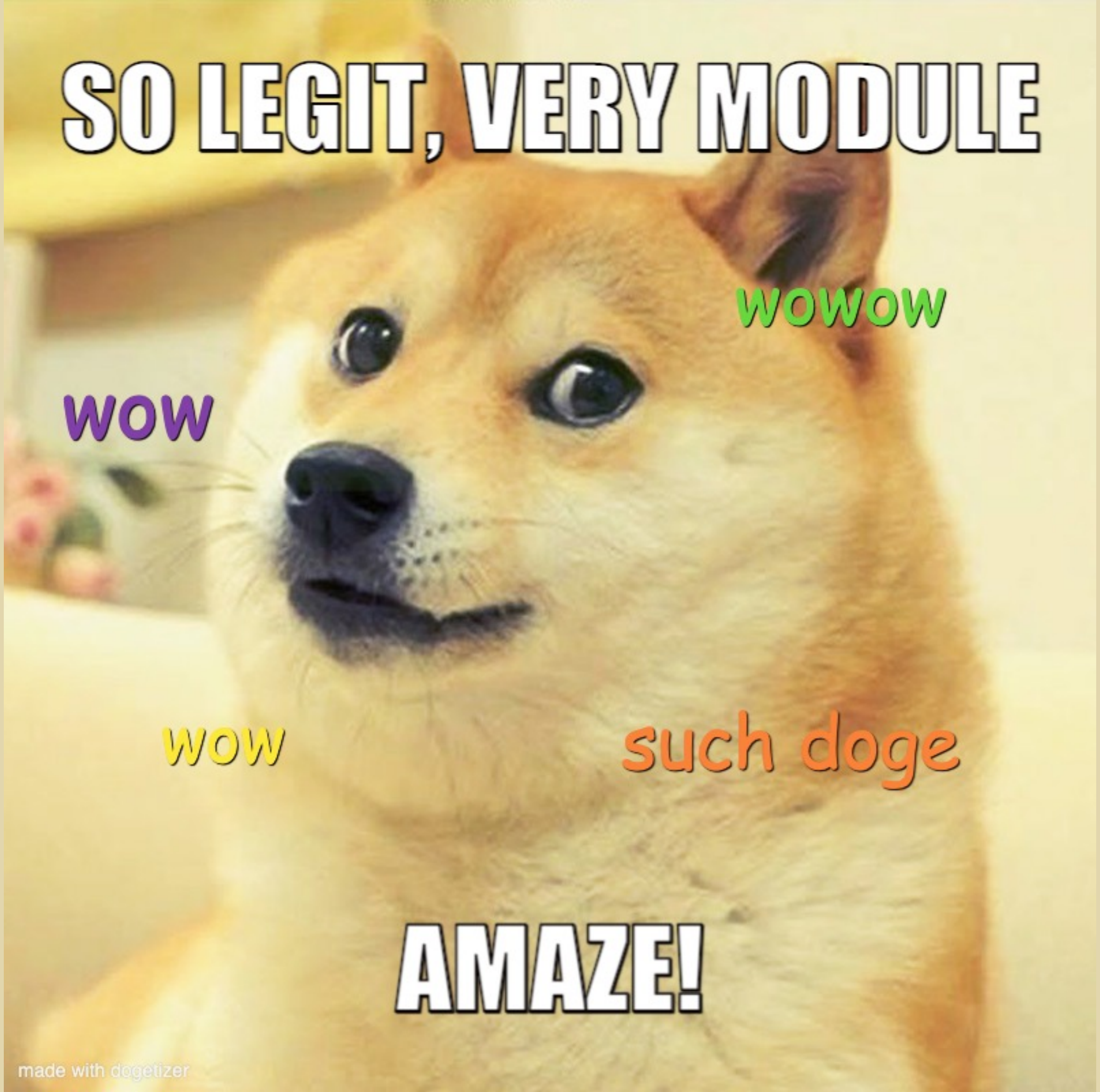
Locate APIs needed for boostrap & chunks

- GetProcAddress spooks AV/EDRs
- as imports of Rope loader would be suspicious
- manual search conflicts with WDEG defenses
    - Export Address Filtering
    - Import Address Filtering

EAF/IAF POLICY

EAF and IAF implement a simple policy:

- monitor Export/Import Address Table of PE modules
- guard page handler shepherds offending access
- allowed if instruction is from legit module...


SO LEGIT, VERY MODULE
wowow
wow
wow
such doge
AMAZE!
made with dogetizer

BYPASS #2: EAF/IAF

1. Locate .text of any loaded Windows DLL
2. Find gadget to make an arbitrary read
3. Adapt your GetProcAddress-like code
   - list of loaded PE modules is not guarded by EAF/IAF
   - wrap accesses to guarded regions so as to use the gadget when dereferencing memory

```
// 8b 00      mov eax, dword ptr [eax]
// c3         ret
```

kernel32.dll

Legit module to WDEG

We may also use JOP gadgets, or a write gadget for IAT hijacking...

BYPASS #2: EAF/IAF

```
DWORD readp(LPBYTE target, DWORD GADGET_read){
      DWORD res = NULL;
      __asm {     mov eax, target ;
                  call GADGET_read ;
                  mov res, eax ; }
      return res;
}

PDWORD pNames = (PDWORD)((LPBYTE)hModule + readp((LPBYTE)pExportDirectory +
      FIELD_OFFSET(IMAGE_EXPORT_DIRECTORY, AddressOfNames), GADGET_read));
```

Legit module to WDEG

EVALUATION

We evaluated Rope on 10 commercial solutions (6 AVs, 4 EDRs)

Setup

- ACG, CIG, EAF, IAF, ROP mitigations + OS defaults
- victim applications running with medium integrity level
- write in Rope payloads that alert AV/EDRs when run standalone
- compare with D-TIME (WOOT'19)

EVALUATION

We evaluated Rope on 10 commercial solutions (6 AVs, 4 EDRs)

Details of Setup

- WDEG mitigations: audit mode, different combinations (incompatibilities)
- two victims (from: Chrome, Skype, Telegram, Dropbox, Reader DC, ...)
- one PoC payload per execution mode
  - Mode 1: modify registry for persistence / play with bcdedit
  - Mode 2: download PS script, make another victim execute it


EXPECTATIONS
LIGHTYEAR
EXPECTATIONS EVERYWHERE

EVALUATION

We evaluated Rope on 10 commercial solutions (6 AVs, 4 EDRs)

RESULTS

✓ no WDEG mitigation triggered

✓ Rope completely deceived 8 out of 10 products

- ➢ two products block OpenProcess (Access Denied) and provide rogue outputs also to DuplicateHandle => *not a real detection, may be evaded...*

✗ D-TIME detected by 7 products

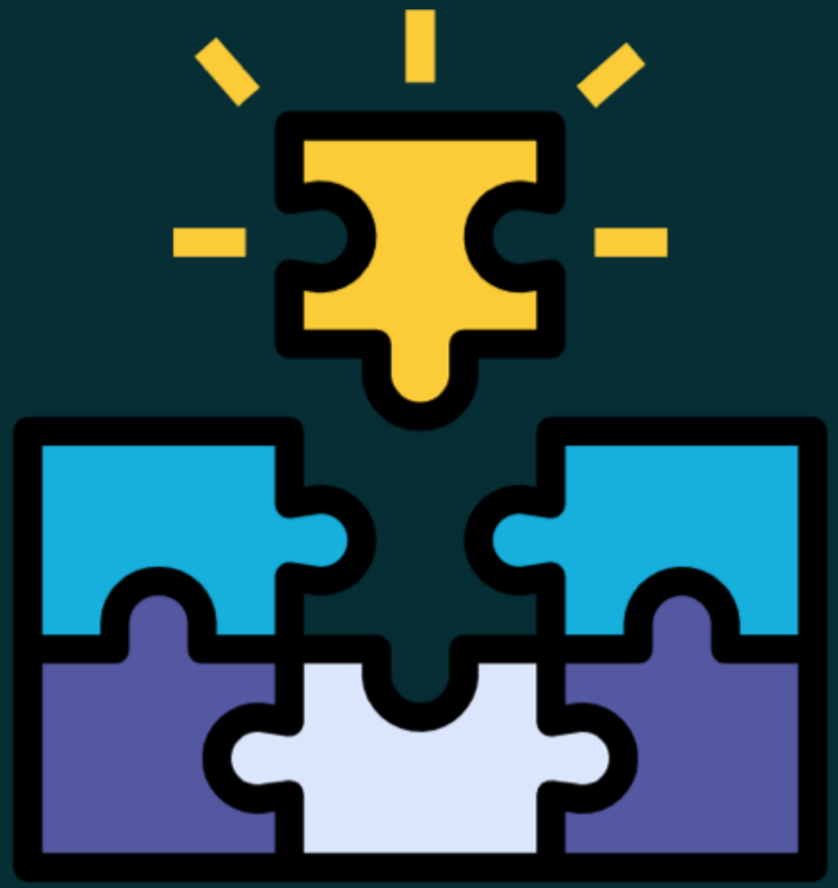
AFTERMATH

- Rope looked like a blind-side hit to AV/EDRs
- Evading user-mode API hooks useful only for injection
  (unnecessary for 7 products, 1 deceived with WOW64 APIs)
- EAF/IAF promising but gullible

OPPORTUNITIES

The architecture of Rope is extensible

- other code reuse flavors
- other covert medium than TxF
- other self-dispatch methods (e.g., APC, IAT hijacking)
- fileless paradigms



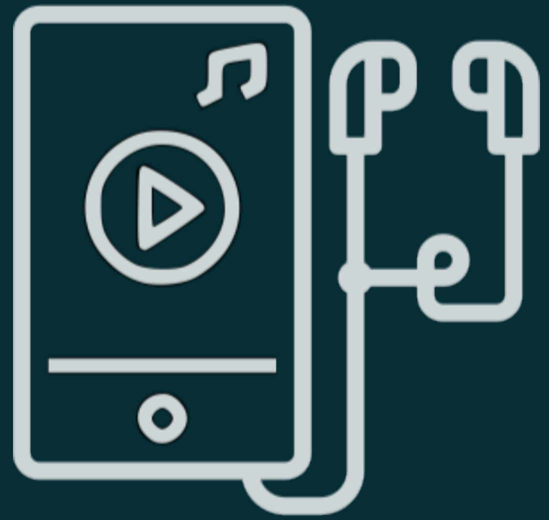
We may need defenses that see Rope & distributed malware as a whole....

DEFENSES

- Behavioral analyses that correlate execution units
  - ✗ tracking execution units faces scalability issues
  - ✗ new injection techniques keep appearing
  - ✓ suggestion: follow duplication and sharing of objects
- Code reuse-aware analyses for in-memory contents
  - ➢ ROPDissector, ROPMEMU
- Reliable means to intercept sensitive APIs

(we followed a responsible disclosure process for our bypasses)

BLACK HAT SOUND BYTES



- Distributed malware poses a tough challenge to AV/EDRs
- ROP is a Swiss-army knife. Also, it helps in many bypasses
- Legit OS features (TxF, handle duplication) need close monitoring

There is a White Paper available!
(and an upcoming ESORICS'21 paper)

@dcdelia

REFERENCES

- Rope: Covert multi-process malware execution with return-oriented programming (to appear in ESORICS 2021)
- malWASH: Washing malware to evade dynamic analysis (WOOT 2016)
- D-TIME: Distributed threadless independent malware execution for runtime obfuscation (WOOT 2019)
- The Naked Sun: Malicious cooperation between benign-looking processes (ACNS 2020)
- ROPInjector: Using return oriented programming for polymorphism and antivirus evasion (Black Hat USA 2015)
- ROPMEMU: A framework for the analysis of complex code-reuse attacks (ASIACCS 2016)
- Static analysis of ROP code (EUROSEC 2019)
- Hiding in the particles: When return-oriented programming meets program obfuscation (DSN 2021)