Mapping Uncharted Territory: Predicting Evasive Tradecraft with Cartographer

$whoami

- Lots of blue team roles
  - SOC operations, Incident Response, Threat Intelligence, Engineering, RE, etc.
- Currently the Architect for Defensive Services at SpecterOps
- Interested in Windows internals and reverse engineering
- Twitter: @michaelbarclay_

Acknowledgements

- Concepts and Research:
  - Jared Atkinson
  - Jonny Johnson
- Answering my Windows internals questions in great detail:
  - Evan McBroom
  - Lee Christensen
- Everyone else at SpecterOps

Agenda

Detection, Evasion, and Predictions

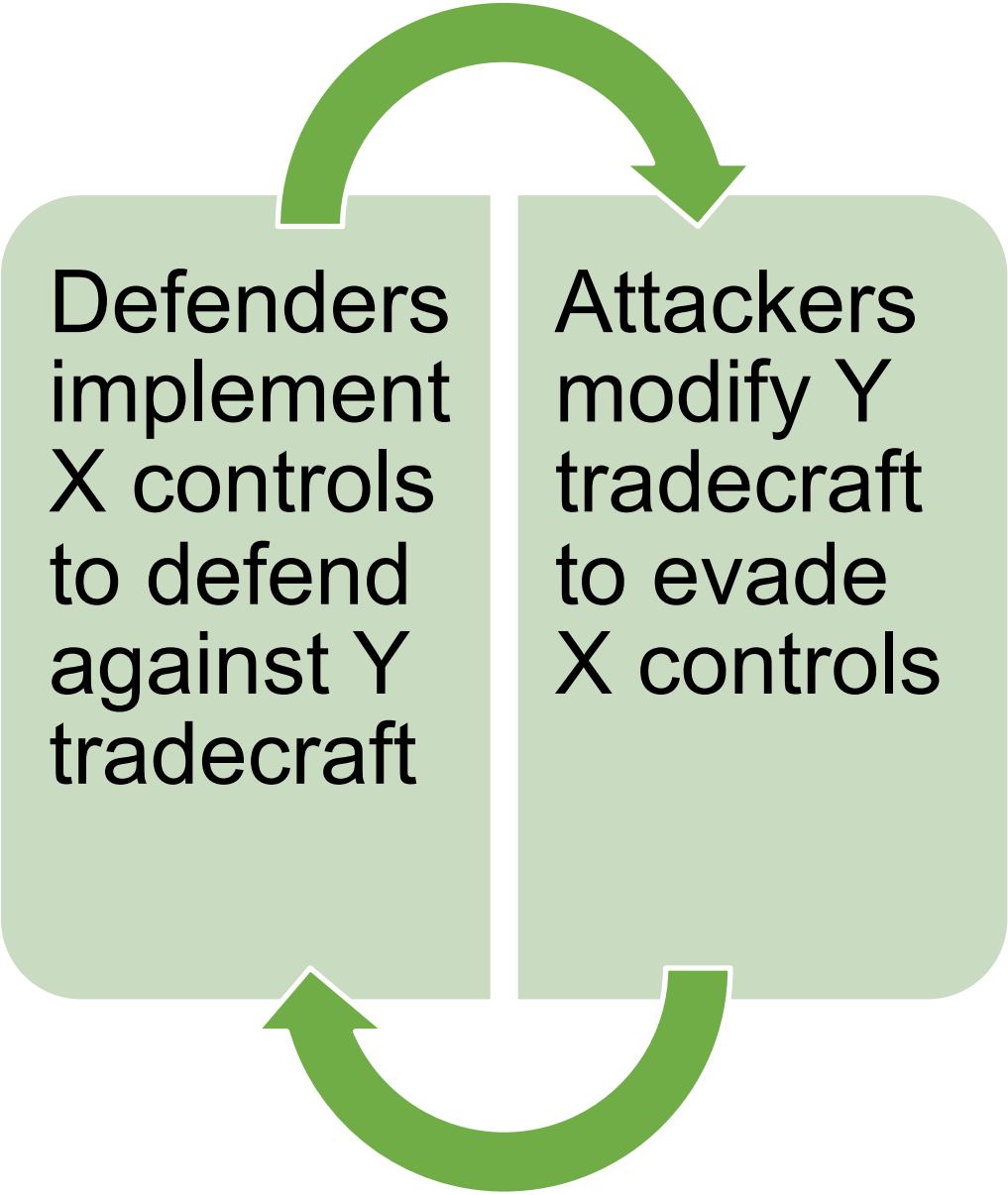
Cartographer's Data Model

The Future of Cartographer

Demo

Predictions

- Information security is adversarial and cyclical
- We are all in the business of making predictions



What is a prediction?

- A prediction is based on some set of **assumptions**

What is a prediction?

- A prediction is based on some set of **assumptions**

Uncertainty

- Assumptions suffer from **uncertainty**. Failure to address uncertainty leads to **inaccurate predictions**.

Tradecraft Uncertainty

Telemetry Uncertainty

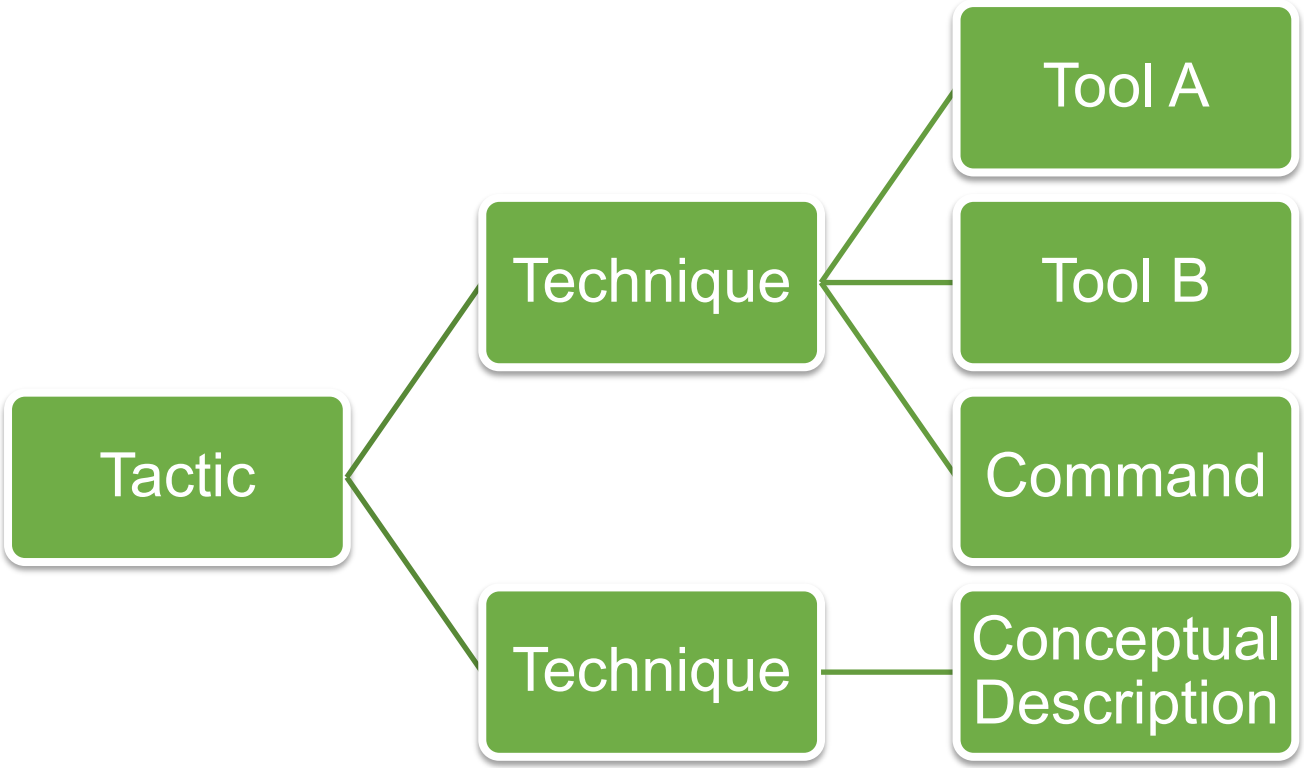
Introducing Cartographer

Cartographer helps to predict possible procedures for a given technique to reduce both types of uncertainty

The Data Model

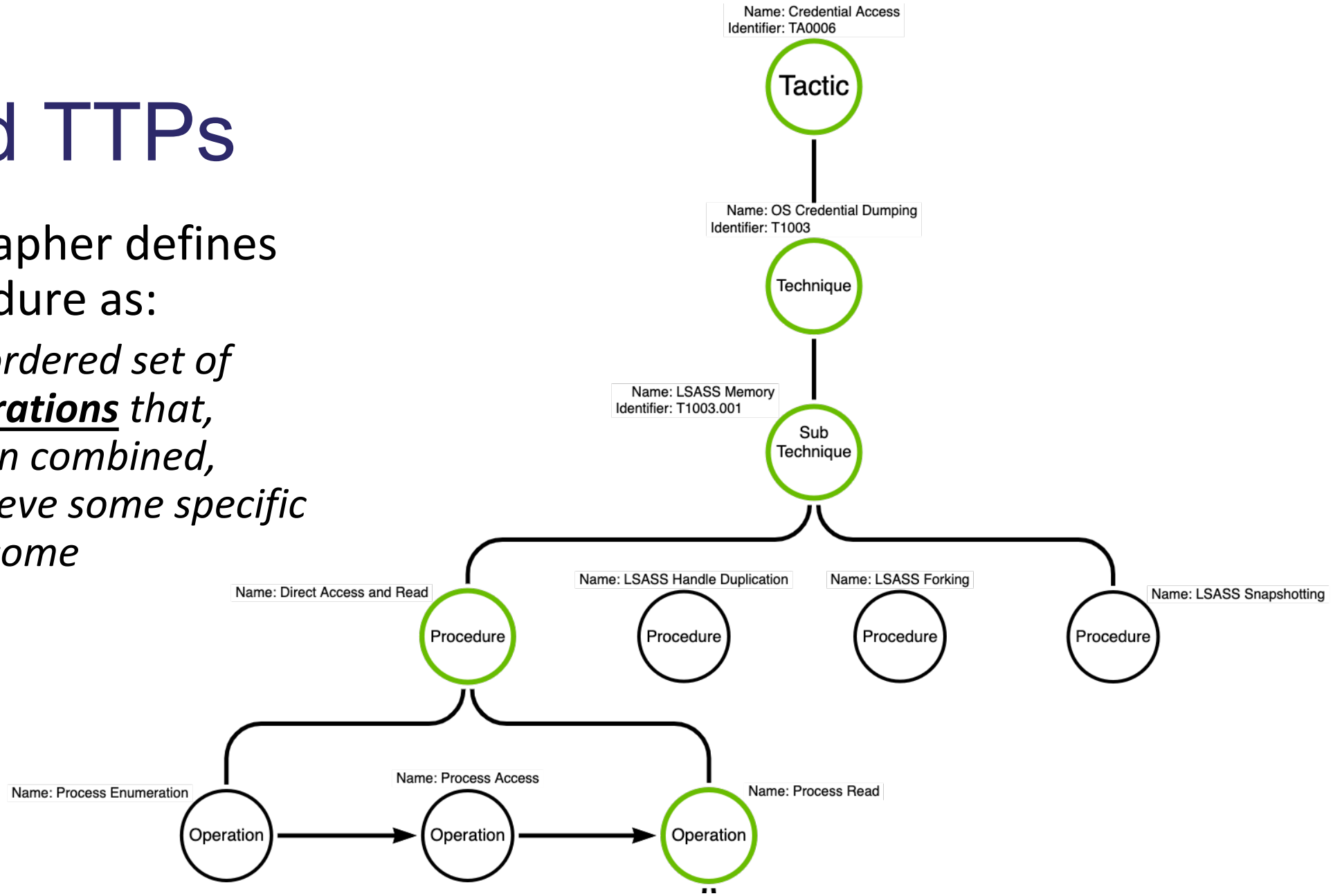
Beyond TTPs

- Tactics, Techniques, and Procedures are all ***categorical*** labels
- No clear definition of a procedure
- Difficult to compare the distinctions between known procedures



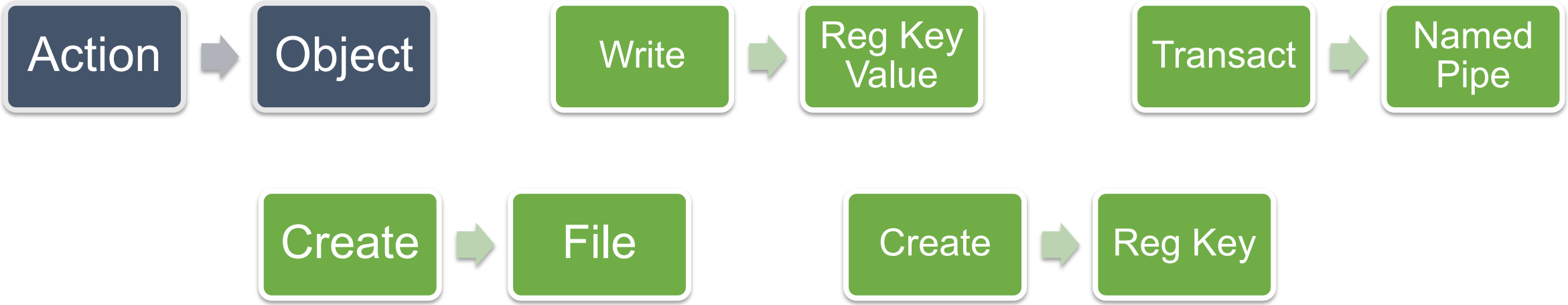
Beyond TTPs

- Cartographer defines a procedure as:
  - *An ordered set of **<u>operations</u>** that, when combined, achieve some specific outcome*



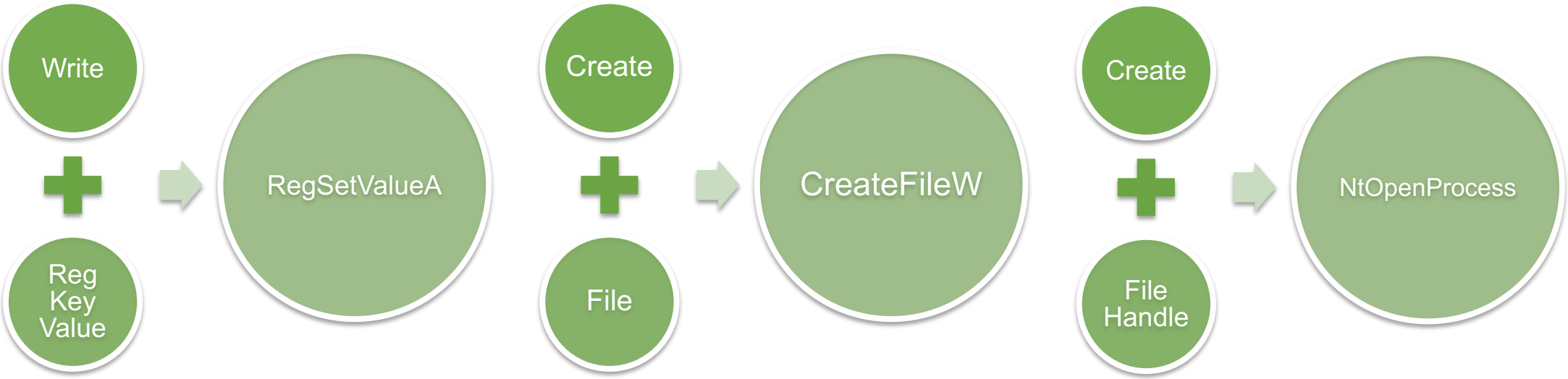
Operations

- An operation is an **action** that targets an **object**



Operations

- An operation is carried out by invoking specific **functions**

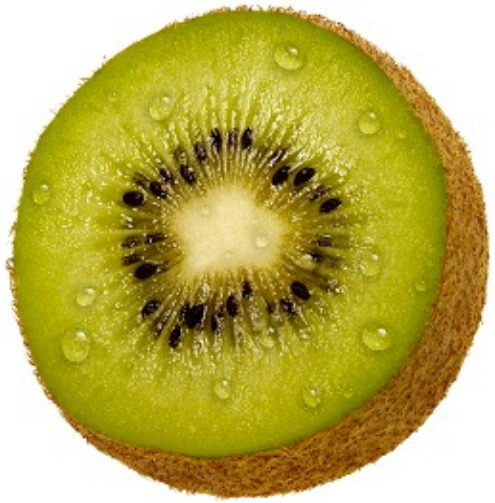


Some Operational Observations

- Tools that implement the same **procedure** often call different functions for a given **operation**
  - These choices have a direct impact on telemetry
- Some operations are more important than others
  - Cyclical innovation is likely to focus on operations that are unavoidable for attackers, but useful to defenders
  - When we write detection rules that target a ”technique”, we are often targeting a specific operation

A Tale of Two Procedures

- Two tools that represent distinct procedures for the same technique (OS Credential Dumping: LSASS Memory)
  - Direct Open: *Mimikatz sekurlsa::logonpasswords*
  - Handle Duplication: *pypykatz live lsa --method handledup*



Direct Open and Read Memory

Operations

1. Obtain the Process ID (PID) for lsass.exe

Abstraction Map: ☒ Function Node Map: ☑	Target Object: Process Action: Enumerate

Show Operation Details

2. Open a handle to the lsass.exe process with the necessary access rights

Abstraction Map: ☒ Function Node Map: ☑	Target Object: Handle Action: Open

Show Operation Details

3. Read lsass.exe process virtual memory

Abstraction Map: ☑ Function Node Map: ☑	Target Object: Process Memory Action: Read

Show Operation Details

Duplicate Handle and Read Memory

1. Enumerate currently open handles to identify a handle that has the PROCESS_VM_READ access right assigned to it

Abstraction Map: ☒	Target Object: Handle
Function Node Map: ☑	Action: Enumerate

Show Operation Details

2. Open a handle to the process that has the currently open handle we want to duplicate

Abstraction Map: ☒	Target Object: Handle
Function Node Map: ☑	Action: Open

Show Operation Details

3. Duplicate the handle while specifying the correct access rights for the duplicated copy of that handle

Abstraction Map: ☒	Target Object: Handle
Function Node Map: ☑	Action: Duplicate

Show Operation Details

4. Validate that the newly duplicated handle points to lsass.exe

Abstraction Map: ☒	Target Object: Handle
Function Node Map: ☑	Action: Query

Show Operation Details

5. Read lsass.exe process virtual memory

Abstraction Map: ☑	Target Object: Memory
Function Node Map: ☑	Action: Read

Show Operation Details

Distinct Procedures

1. Obtain the Process ID (PID) for lsass.exe

Abstraction Map: ☒	Target Object: Process
Function Node Map: ☑	Action: Enumerate

Show Operation Details

2. Open a handle to the lsass.exe process with the necessary access rights

Abstraction Map: ☒	Target Object: Handle
Function Node Map: ☑	Action: Open

Show Operation Details

1. Enumerate currently open handles to identify a handle that has the PROCESS_VM_READ access right assigned to it

Abstraction Map: ☒	Target Object: Handle
Function Node Map: ☑	Action: Enumerate

Show Operation Details

2. Open a handle to the process that has the currently open handle we want to duplicate

Abstraction Map: ☒	Target Object: Handle
Function Node Map: ☑	Action: Open

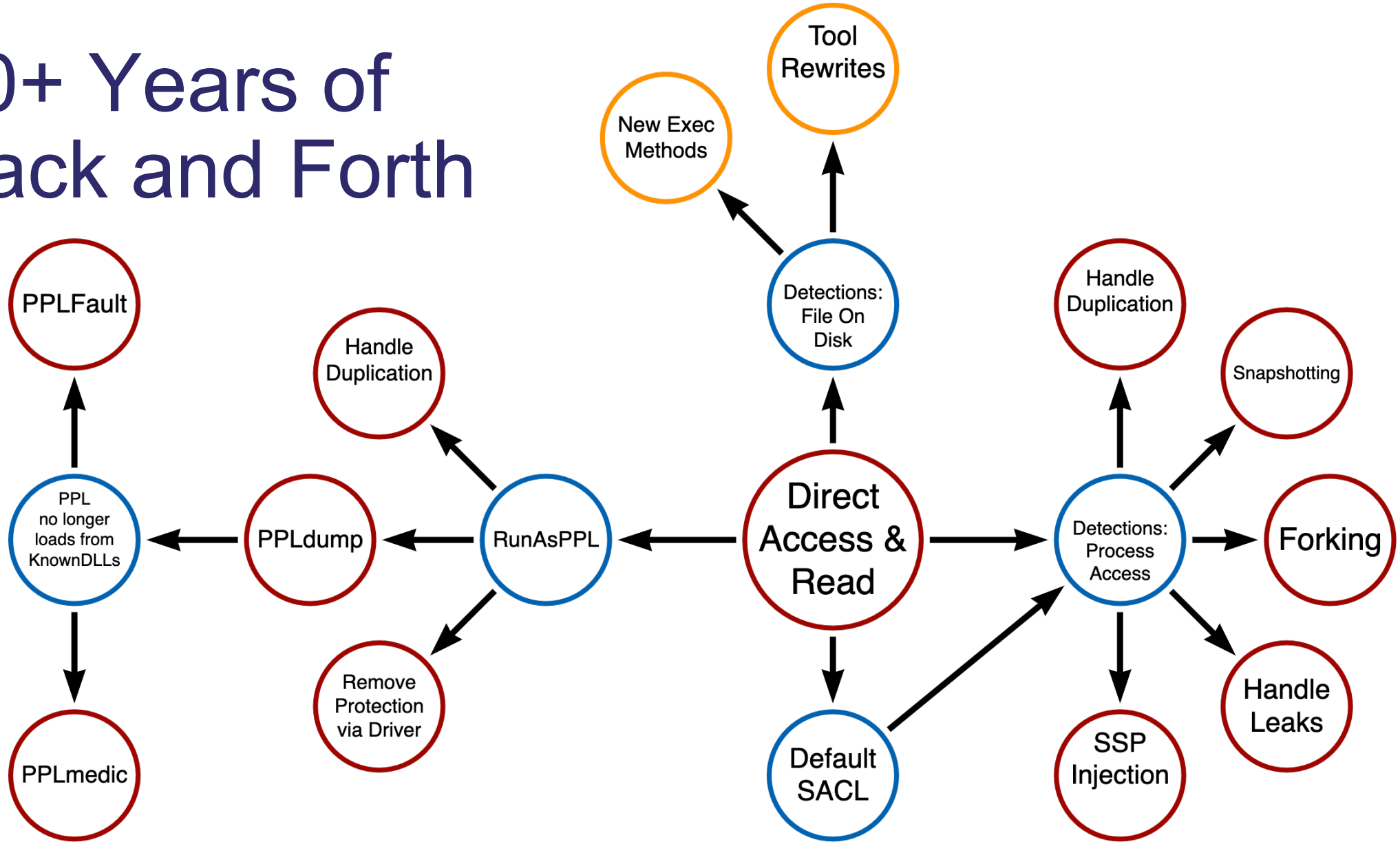
Show Operation Details

3. Duplicate the handle while specifying the correct access rights for the duplicated copy of that handle

Abstraction Map: ☒	Target Object: Handle
Function Node Map: ☑	Action: Duplicate

Show Operation Details

10+ Years of Back and Forth



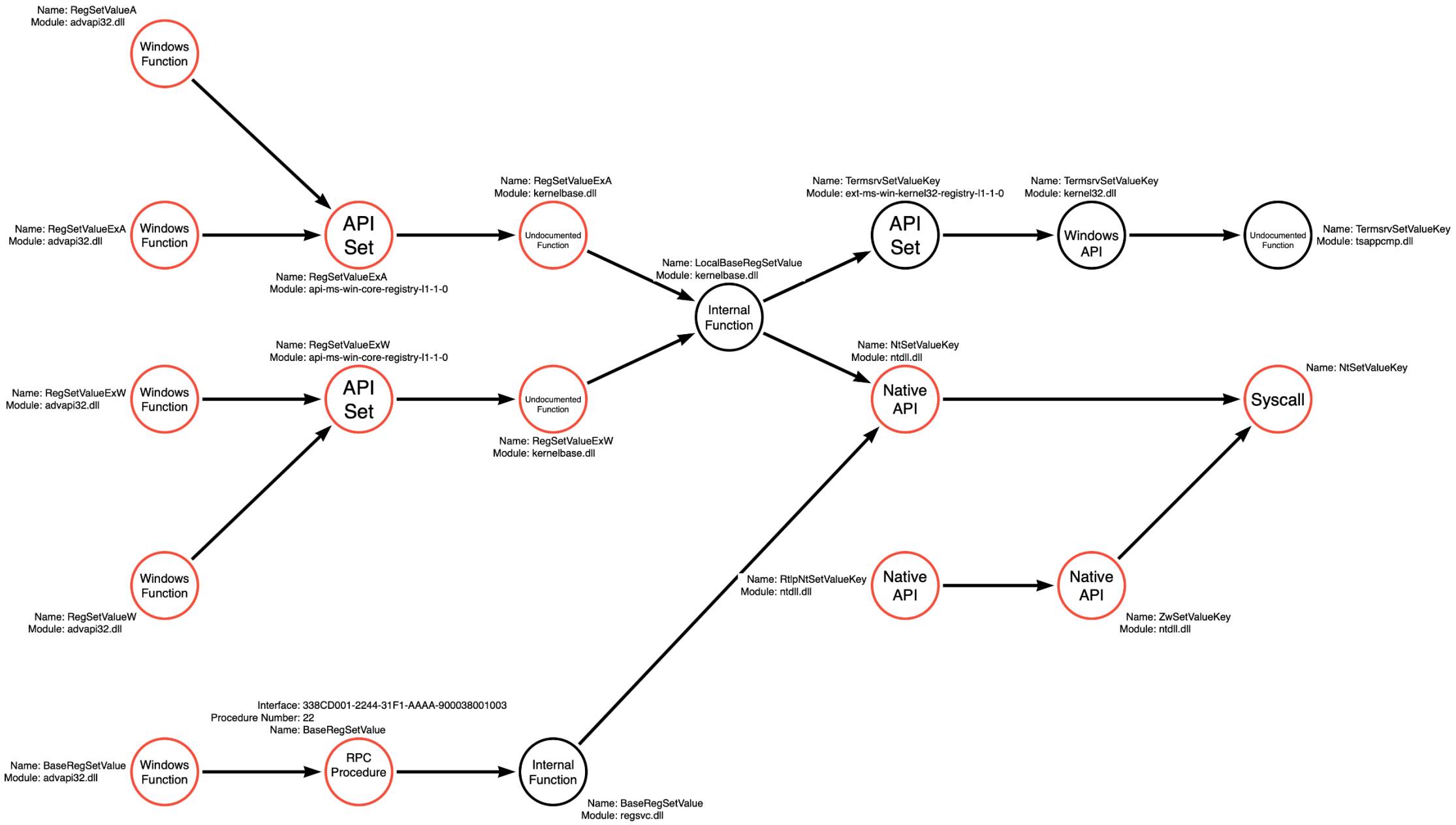
Dumping LSASS

- Modern tooling makes it easy to be selective about the way that we acquire (or avoid acquiring) a handle to LSASS
- Source: [Nanodump]

```
// get a handle to LSASS via multiple methods
BOOL obtain_lsass_handle(
    OUT PHANDLE phProcess,
    IN DWORD lsass_pid,
    IN BOOL duplicate_handle,
    IN BOOL elevate_handle,
    IN BOOL duplicate_elevate,
    IN BOOL use_seclogon_duplicate,
    IN DWORD spoof_callstack,
    IN BOOL is_seclogon_leak_local_stage_2,
    IN LPCSTR seclogon_leak_remote_binary,
    OUT PPROCESS_LIST* Pcreated_processes,
    IN BOOL use_valid_sig,
    IN LPCSTR dump_path,
    IN BOOL fork_lsass,
    IN BOOL snapshot_lsass,
    OUT PHANDLE PhSnapshot,
    IN BOOL use_seclogon_leak_local,
    IN BOOL use_seclogon_leak_remote,
    IN BOOL use_lsass_shtinkering)
```

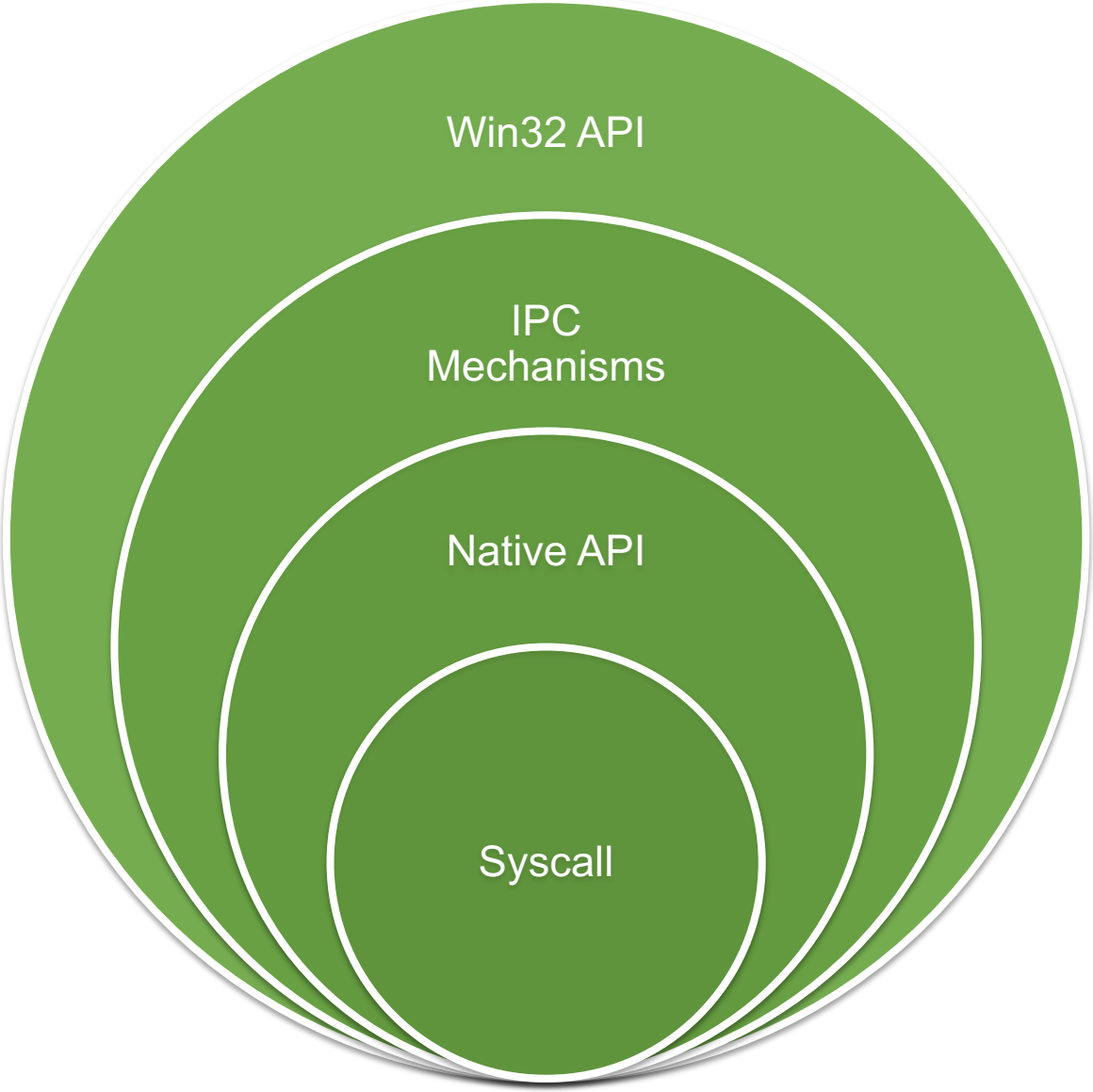
Operational Variation

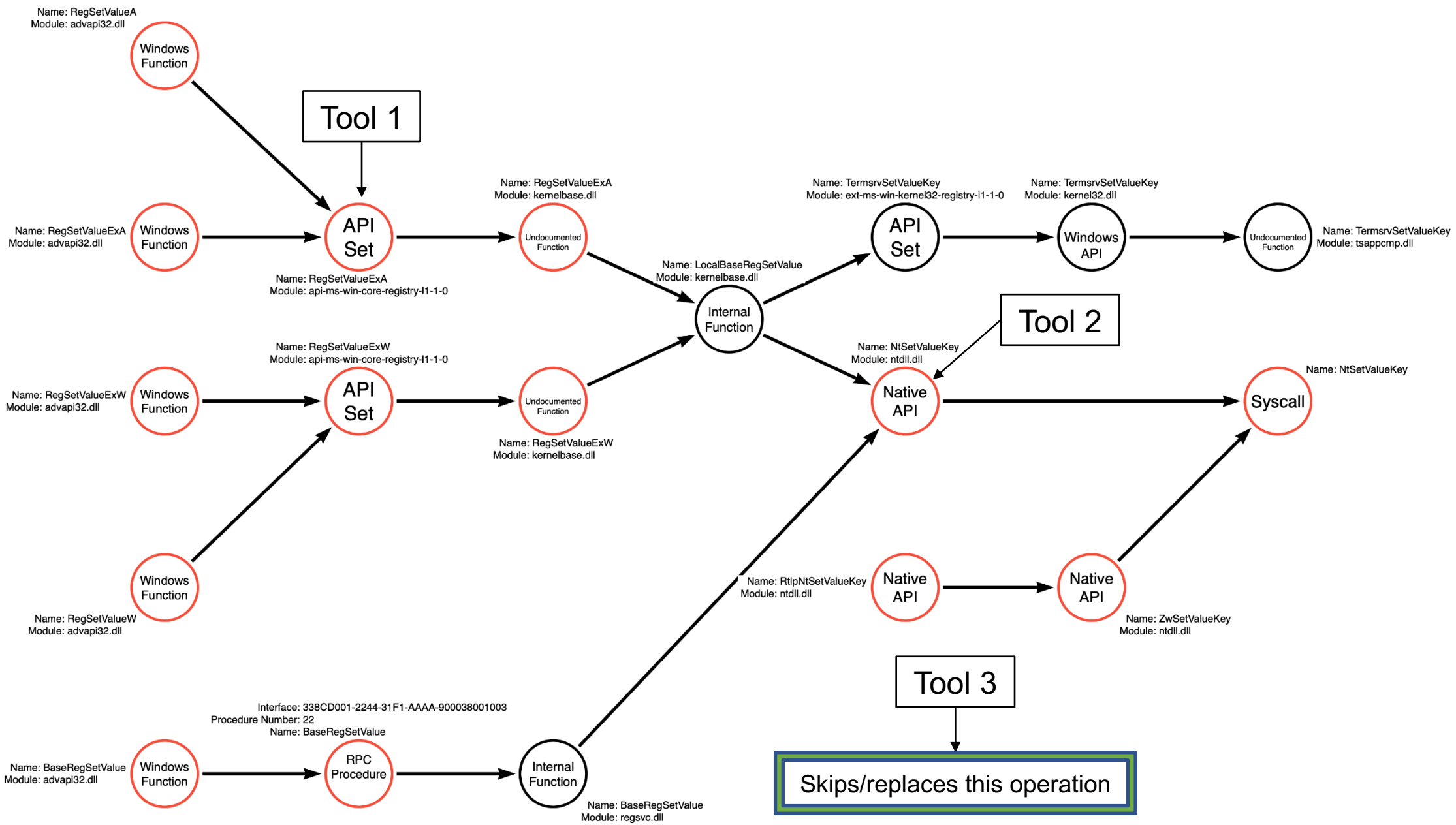
- **Function Maps** show the callable and internal functions that can be used to carry out an operation
- **Abstraction Maps** show how specific tools make use of functions to carry out an operation AND how those functions interact with operating system components


Name: RegSetValueA
Module: advapi32.dll
Windows Function
Name: RegSetValueExA
Module: advapi32.dll
Windows Function
API Set
Name: RegSetValueExA
Module: api-ms-win-core-registry-l1-1-0
Name: RegSetValueExA
Module: kernelbase.dll
Undocumented Function
Name: LocalBaseRegSetValue
Module: kernelbase.dll
Internal Function
Name: TermsrvSetValueKey
Module: ext-ms-win-kernel32-registry-l1-1-0
API Set
Name: TermsrvSetValueKey
Module: kernel32.dll
Windows API
Undocumented Function
Name: TermsrvSetValueKey
Module: tsappcmp.dll
Name: RegSetValueExW
Module: advapi32.dll
Windows Function
Name: RegSetValueExW
Module: api-ms-win-core-registry-l1-1-0
API Set
Undocumented Function
Name: RegSetValueExW
Module: kernelbase.dll
Name: NtSetValueKey
Module: ntdll.dll
Native API
Name: NtSetValueKey
Syscall
Windows Function
Name: RegSetValueW
Module: advapi32.dll
Name: RtlpNtSetValueKey
Module: ntdll.dll
Native API
Native API
Name: ZwSetValueKey
Module: ntdll.dll
Interface: 338CD001-2244-31F1-AAAA-900038001003
Procedure Number: 22
Name: BaseRegSetValue
Name: BaseRegSetValue
Module: advapi32.dll
Windows Function
RPC Procedure
Internal Function
Name: BaseRegSetValue
Module: regsvc.dll

Functional Paths

- Windows provides a hierarchy of functions to interact with OS components, services, and objects
- Tool authors MUST make decisions about where to enter this hierarchy




Name: RegSetValueA
Module: advapi32.dll
Windows Function
Tool 1
Name: RegSetValueExA
Module: advapi32.dll
Windows Function
API Set
Name: RegSetValueExA
Module: api-ms-win-core-registry-l1-1-0
Name: RegSetValueExA
Module: kernelbase.dll
Undocumented Function
Name: LocalBaseRegSetValue
Module: kernelbase.dll
Internal Function
Name: TermsrvSetValueKey
Module: ext-ms-win-kernel32-registry-l1-1-0
API Set
Name: TermsrvSetValueKey
Module: kernel32.dll
Windows API
Undocumented Function
Name: TermsrvSetValueKey
Module: tsappcmp.dll
Tool 2
Name: NtSetValueKey
Module: ntdll.dll
Native API
Syscall
Name: NtSetValueKey
Name: RegSetValueExW
Module: advapi32.dll
Windows Function
API Set
Name: RegSetValueExW
Module: api-ms-win-core-registry-l1-1-0
Undocumented Function
Name: RegSetValueExW
Module: kernelbase.dll
Windows Function
Name: RegSetValueW
Module: advapi32.dll
Name: RtlpNtSetValueKey
Module: ntdll.dll
Native API
Native API
Name: ZwSetValueKey
Module: ntdll.dll
Tool 3
Skips/replaces this operation
Interface: 338CD001-2244-31F1-AAAA-900038001003
Procedure Number: 22
Name: BaseRegSetValue
Name: BaseRegSetValue
Module: advapi32.dll
Windows Function
RPC Procedure
Internal Function
Name: BaseRegSetValue
Module: regsvc.dll

Abstraction Maps

Service Creation					
Tools	PsExec	PowerShell New-Service	PSReflect-Functions	NtObjectManager	regedit
Windows API Functions	AdvApi32!CreateServiceW RpcRt4!NdrClientCall3				
Interprocess Communication	367ABB81-9844-35F1-AD32-98F038001003 [MS-SCMR] Microsoft Service Control Manager Remote Protocol Opnum 12 (RCreateServiceW) C:\Windows\System32\services.exe				
Network	RPC over Named Pipes (ncacn_np): PIPE\\svcctl RPC over TCP_IP (ncacn_ip_tcp)				
Securable Object *(Chokepoint)*	Registry Key Creation HKLM\SYSTEM\CurrentControlSet\Services				

Chokepoints

- Cartographer will identify the highest layer of the Abstraction Map where all known functional paths converge

Securable Object *(Chokepoint)*	Registry Key Creation HKLM\SYSTEM\CurrentControlSet\Services

- No matter what functional path a tool takes to create a service, there will be a registry key created
  - Chokepoints can change as we discover additional functional paths

Making Predictions with Cartographer

Build Detection Strategies

- Cartographer can answer important questions necessary to build a robust detection strategy

Which operations can't be avoided by attackers for a given procedure?

Which operations generate consumable telemetry?

Which parts of my rule logic are subject to evasion opportunities?

Purple Teaming

- Purple teaming is concerned with **thoroughly** testing a given technique in multiple ways to **expose detection and protective control gaps**

Do my test cases include all known iterations of a given technique?

How do I describe the difference between procedures of a given technique?

What kind of telemetry can I expect when I run a given procedure?

The Present and Future of Cartographer

Current State

- Current content reachable at https://cartographer.run
- Read-only display of currently covered techniques and procedures
  - Exclusively covers Windows tradecraft
  - Not a full library – think of this as a preview of the data model

Future Work

Integrations	Enrichment	Open Source	Instructional	UI
• Telemetry • Detection Rules	• Code Snippets • MSFT Docs	• Community contributions • Open Data Model	• Docs • Blogs • Video content	• More comparison tools • Make it pretty

Some Challenges

Research Intensive Content

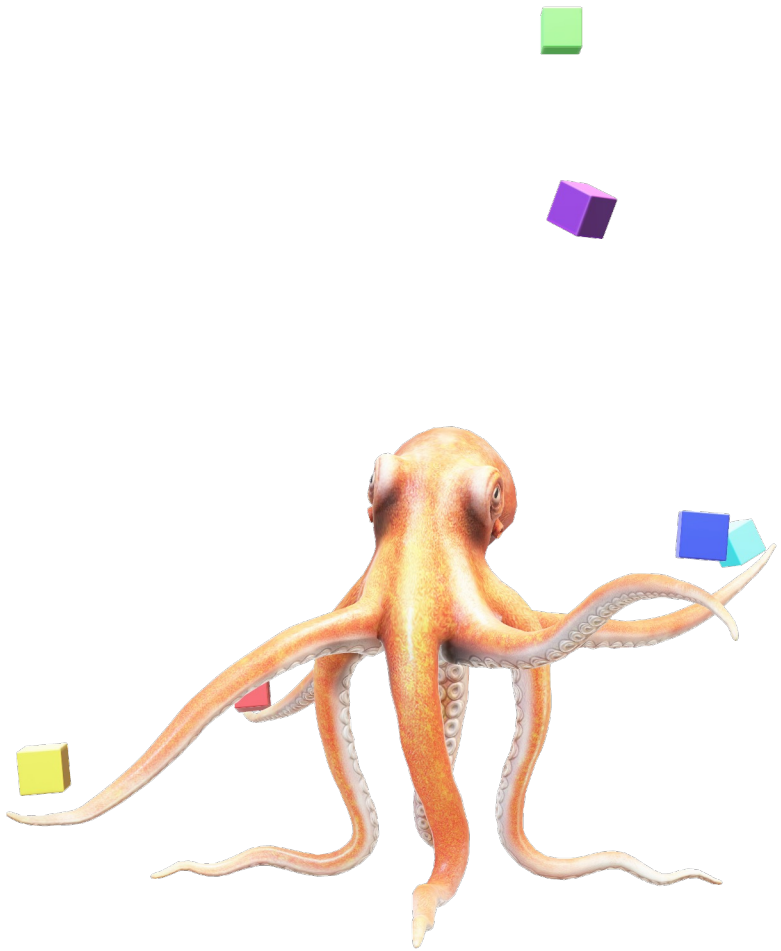
Tricky Input Format

Data Model Edge Cases

Further Reading

- On Detection: Tactical to Functional (Part 1 of 5) - Jared Atkinson
- Capability Abstraction Case Study: Detecting Malicious Boot Configuration Modifications – Michael Barclay
- Identifying RPC Telemetry for Detection Engineers - Jonny Jonson
- Exploring Mimikatz part II - Adam Chester

Questions & Contributions

- DM me on twitter @michaelbarclay_
- Email me at mbarclay@specterops.io
- GitHub repo will be made public and open to contributions in the coming months

Demo Time!