Automated Cyber Defense Courses of Actions – Reference Implementation

Kevin Miller

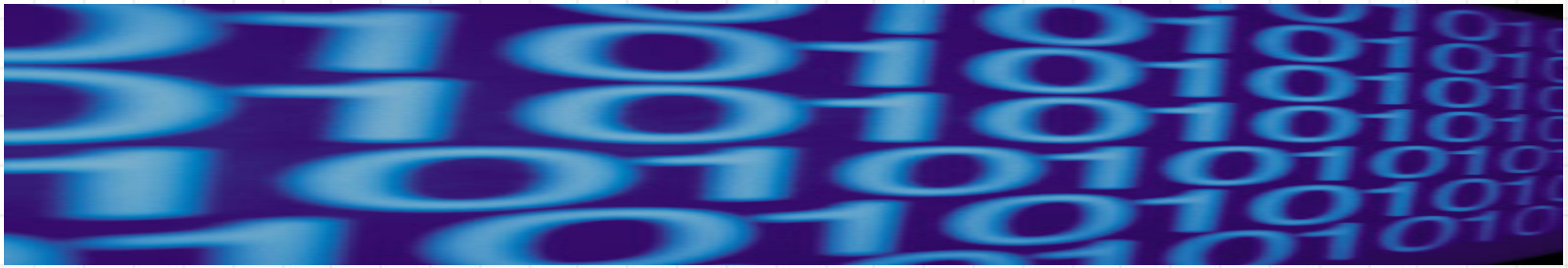
September 29, 2016

Agenda

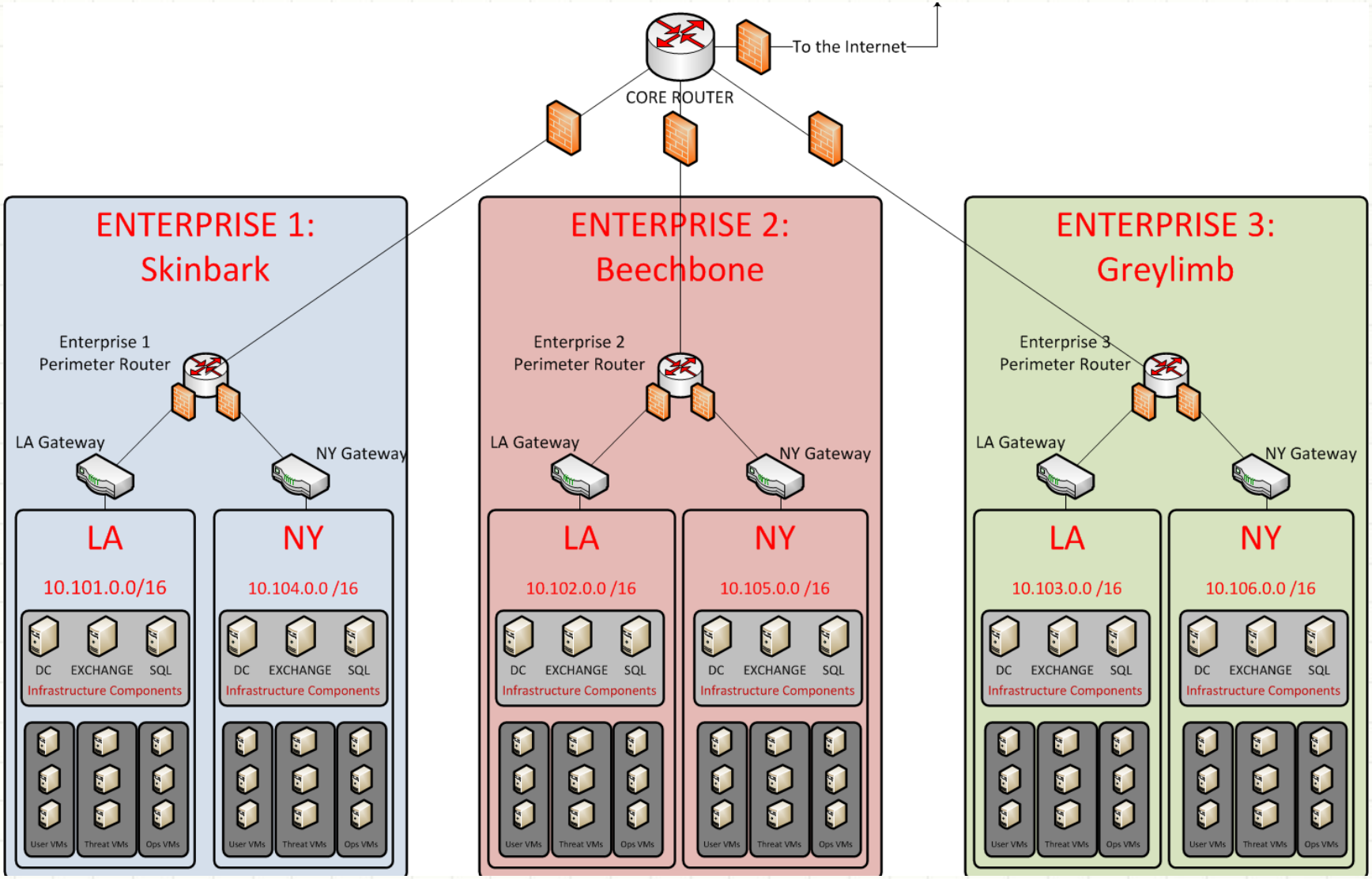
- Objectives
- Approach
  - Initial Architecture
  - OpenC2 Library
  - Use Cases
- Findings
- Questions

Objectives

- Automating proactive defensive measures
- Leveraging OpenC2 language as a standardized interoperability layer
- Understanding relationships amongst sequential response actions



Initial Architecture



ADA RI - Applications

splunk>

Microsoft®
System Center™

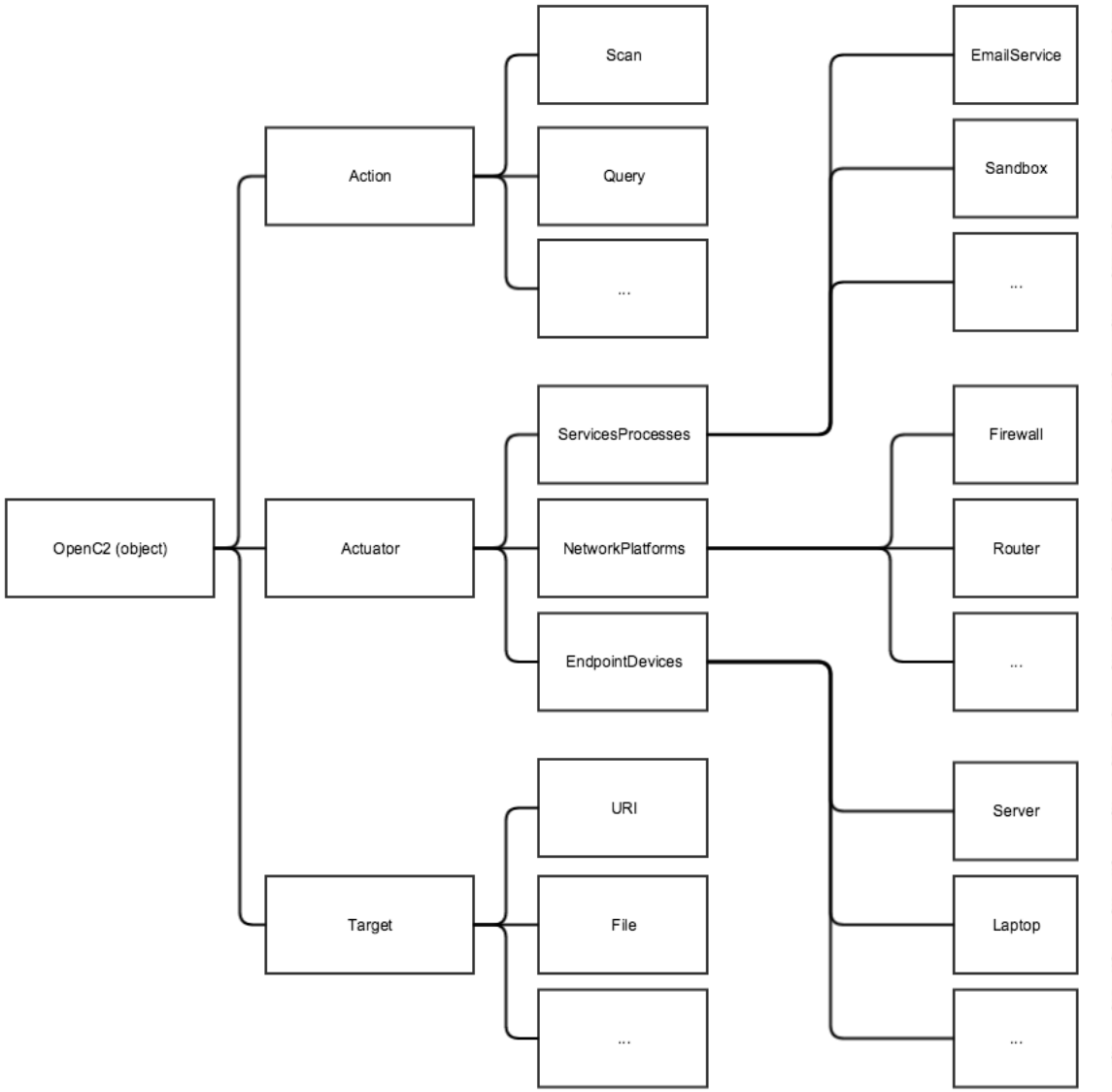
ActiveMQ

INVOTAS

OpenC2 Library

- Why a library?
  - Minimize pairwise integration code
  - Write core OpenC2 code once, reuse many times
  - Maintain consistent code across multiple hosts with ease
  - Enable mapping from OpenC2 commands to custom callback functions containing application API calls

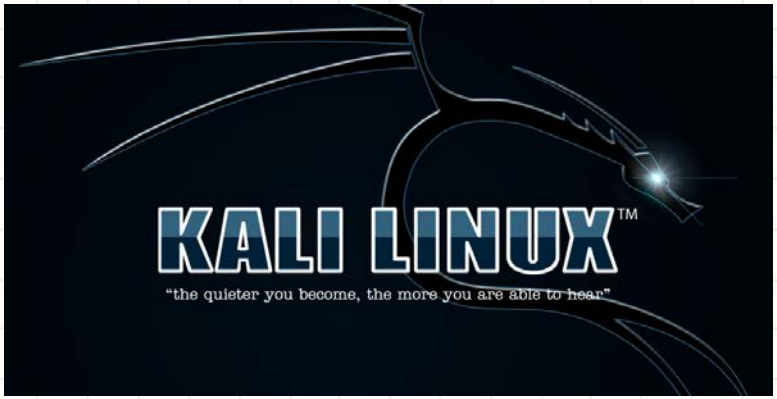
OpenC2 Library – Software Architecture



OpenC2 Library – Command Status

Command	Code	Scenario	Potential Actuators
RESTART	✓	✓	CarbonBlack, ePO
GET	✓	✓	CarbonBlack
DETONATE	✓	✓	Cuckoo
DENY	✓	✓	iptables
INVESTIGATE	✓	✓	VirusTotal
QUERY	✓	✓	CarbonBlack
SCAN	✓		Nessus
SET	✓	✓	SCCM
UPDATE	✓	✓	SCCM
COPY	✓	✓	CarbonBlack, ePO
MODIFY	✓	✓	SCCM
DELETE	✓	✓	CarbonBlack, ePO
LOCATE	✓	✓	CarbonBlack
REMEDIATE	✓	✓	Invotas, Phantom
STOP	✓		SCCM

Use Cases – Web Drive By



- Spear-phishing → navigation to compromised web site → artifact dropped on host
- Snort alerts trigger automated defense sequence
- Where possible, orchestrator to actuator or actuator manager commands leverage OpenC2

OpenC2

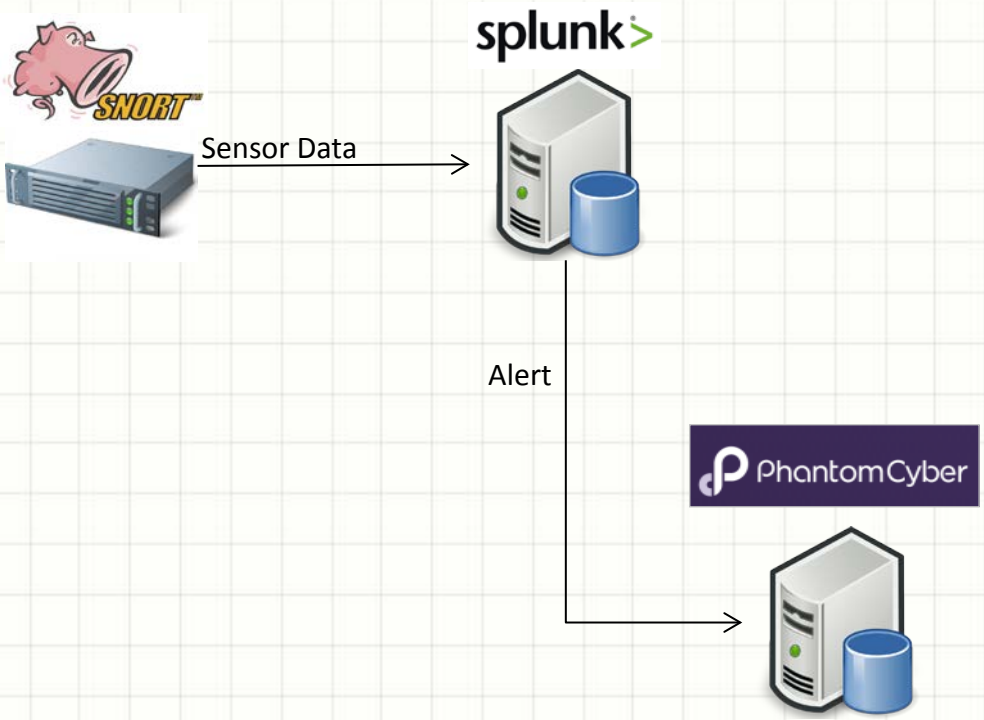
Use Cases – Web Drive By

1. QUERY
   - Find processes associated with suspicious IP address
2. COPY
   - Copies a file from affected endpoint to a quarantined area
   - Finding: CybOX does not seem to support full network paths to files
3. INVESTIGATE
   - Commands a dedicated machine to execute a VirusTotal lookup
4. DETONATE
   - Commands Cuckoo to pull and detonate file

Use Cases – Web Drive By

5. RESTART
   - Restart affected endpoint
6. DENY
   - Commands netfilter/iptables to block the threat
7. LOCATE
   - Commands CarbonBlack to find the suspicious file by hash across all endpoints
8. DELETE
   - Commands CarbonBlack to delete suspicious files from affected endpoints

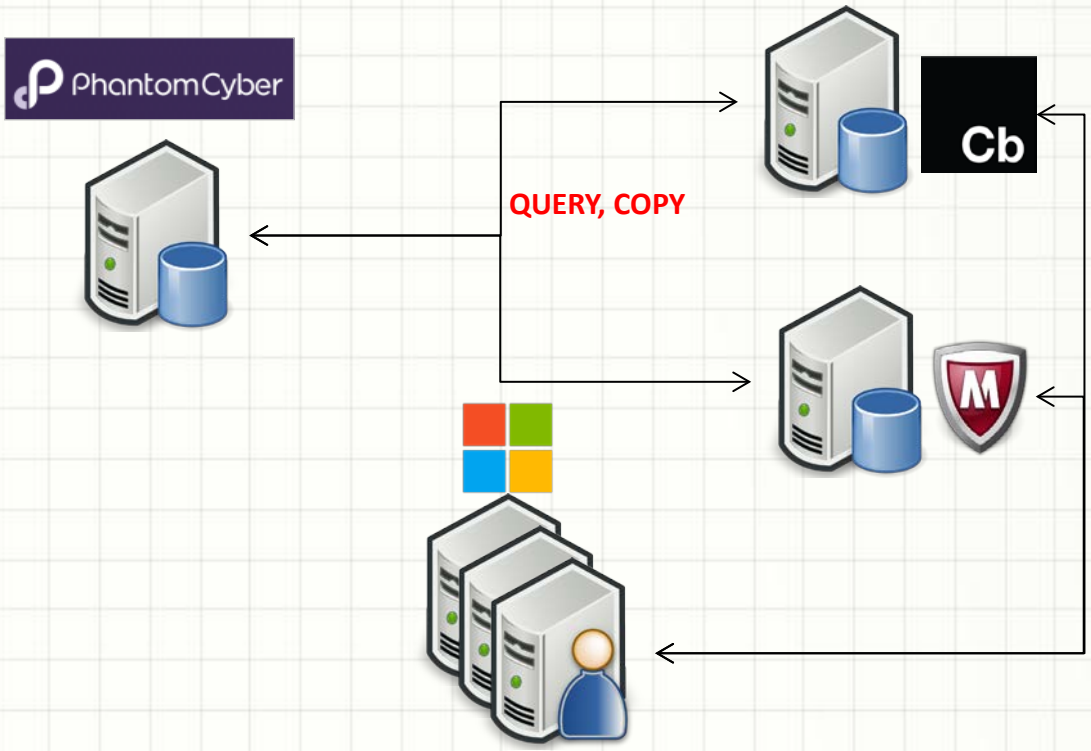
Use Cases – Web Drive By



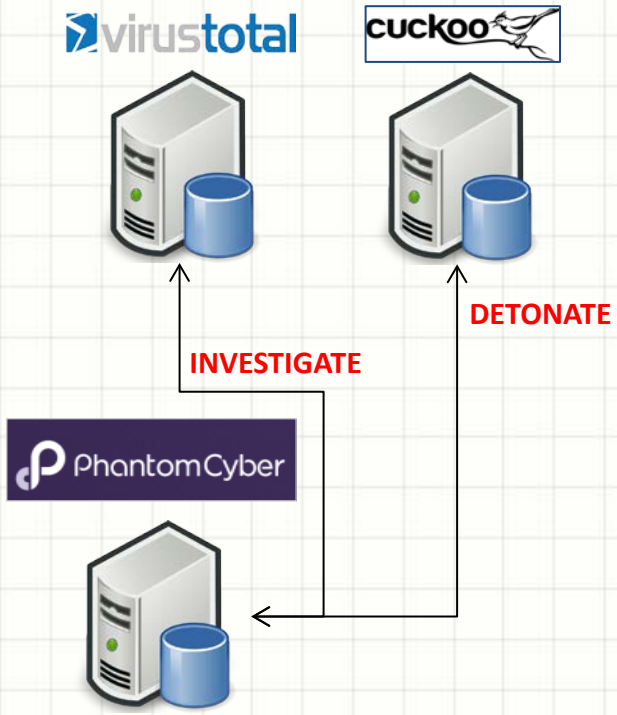
- Snort NIDS forwards sensor data to Splunk
- Splunk generates an alert, triggering Phantom

Use Cases – Web Drive By

- Phantom sends a QUERY for running processes to CarbonBlack and ePO
- Phantom sends a COPY to CarbonBlack or ePO for a forensic copy to protected location



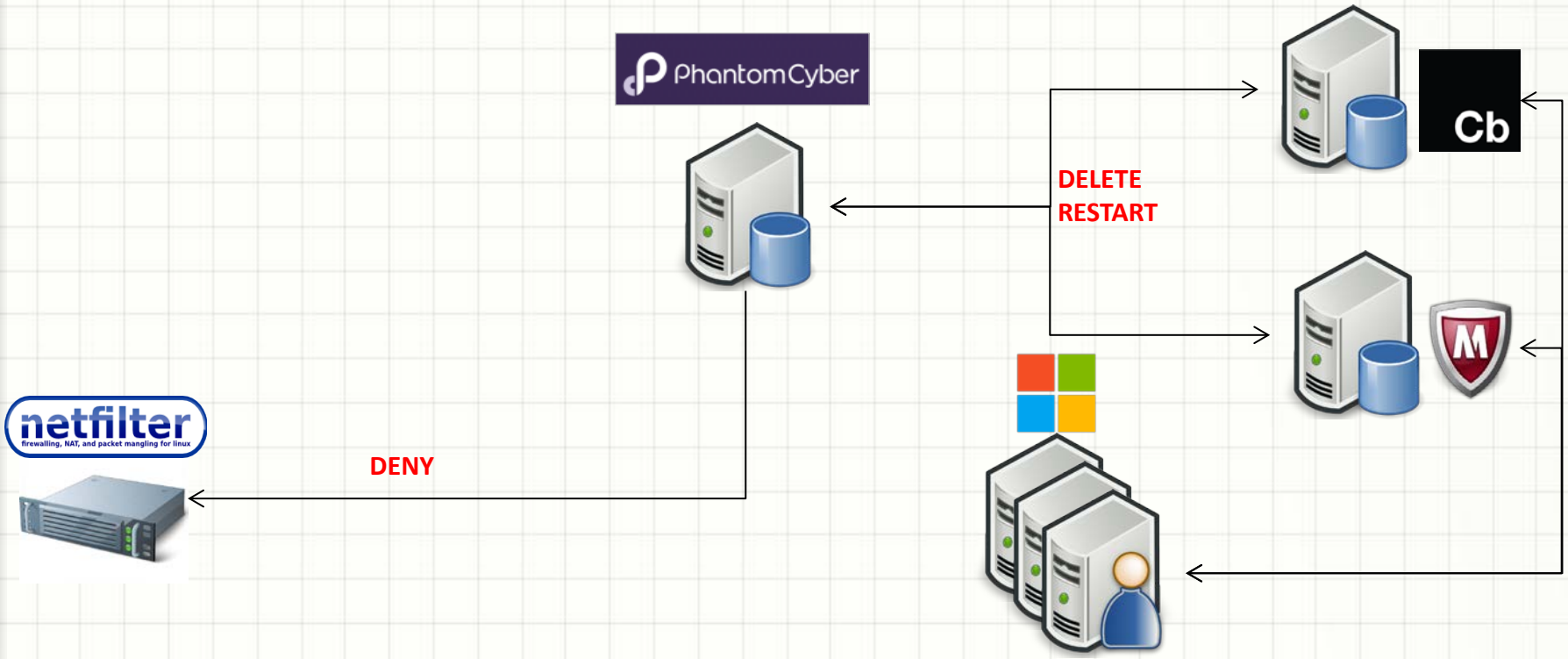
Use Cases – Web Drive By



- Phantom sends INVESTIGATE to VirusTotal (internal relay server) with hash of file
- Phantom sends DETONATE to Cuckoo for forensic information gathering of file

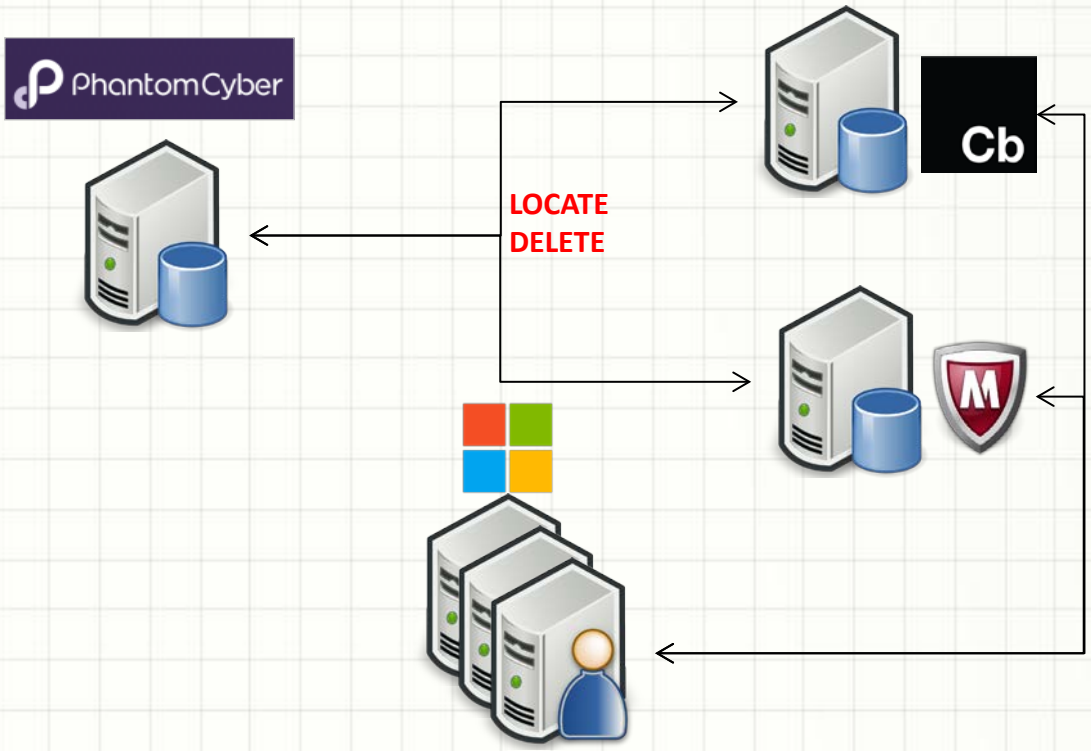
Use Cases – Web Drive By

- Phantom determines file is malicious, sends DENY to perimeter firewall (netfilter/iptables)
- Phantom tasks CarbonBlack or ePO to remove the file and reboot via DELETE and RESTART respectively

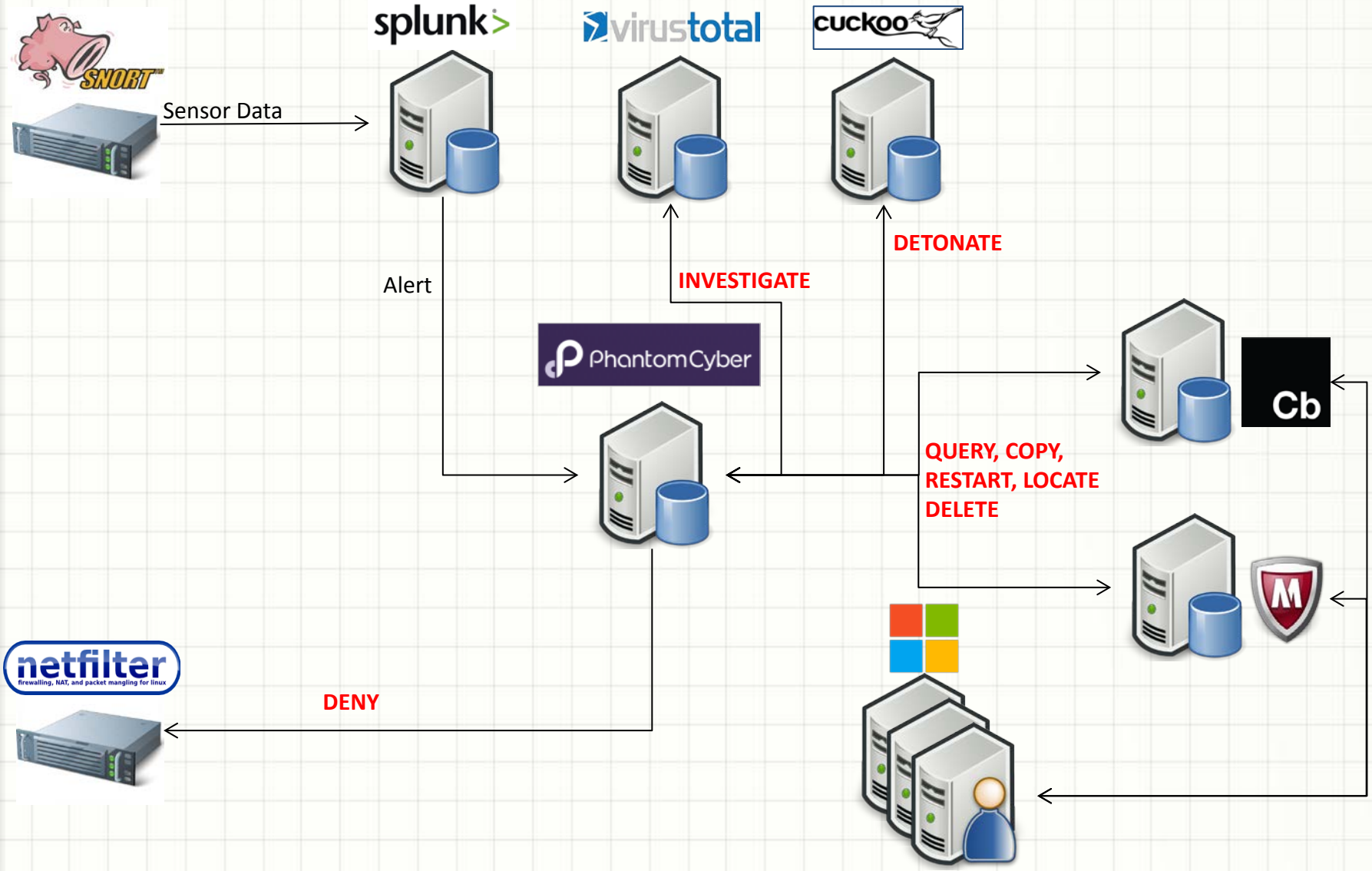


Use Cases – Web Drive By

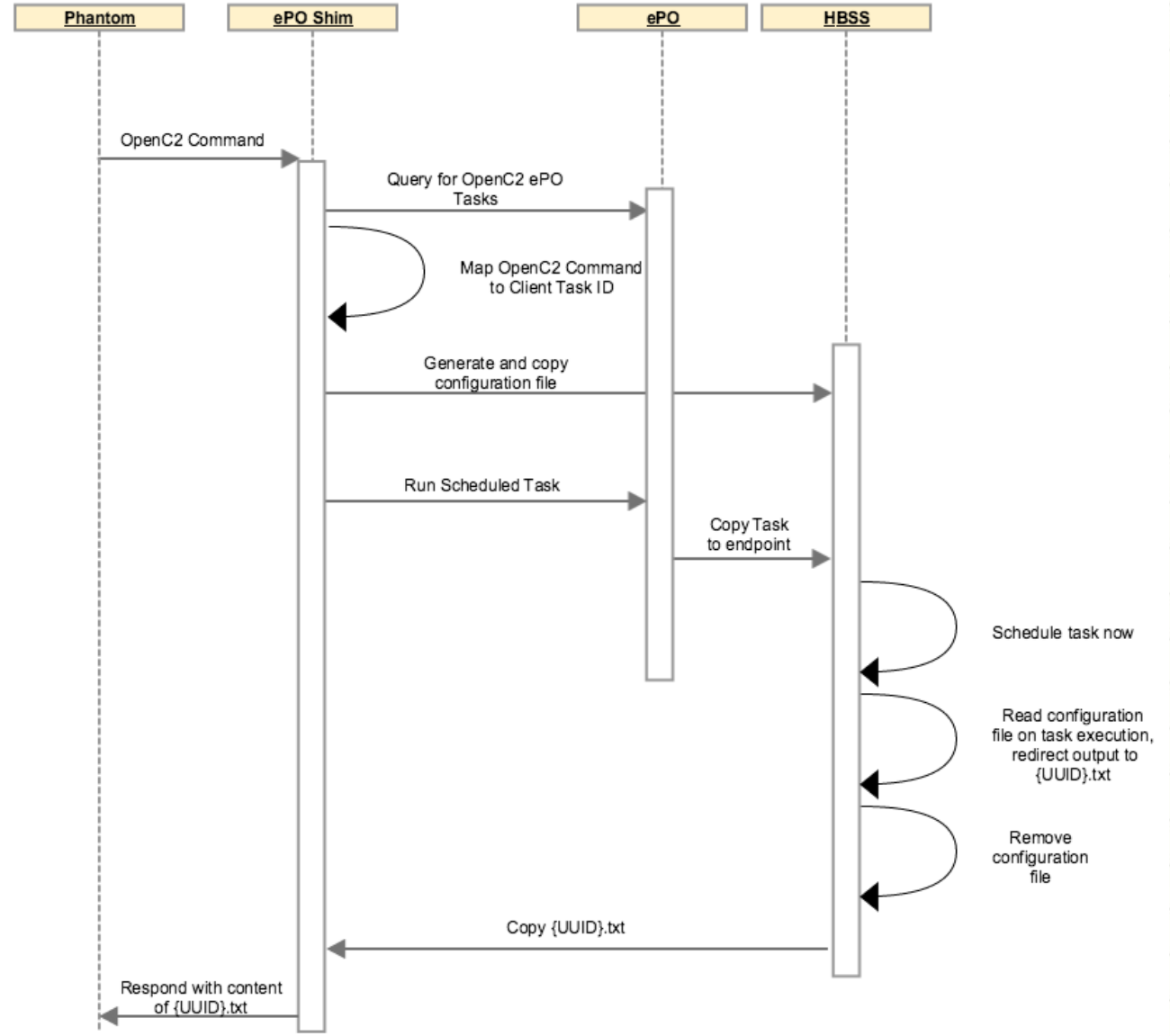
- Phantom hunts for other instances by sending LOCATE to CarbonBlack or ePO
- If any files are found, Phantom sends DELETE to CarbonBlack or ePO



Use Cases – Web Drive By



ePO Workaround



Use Cases – Web Drive By

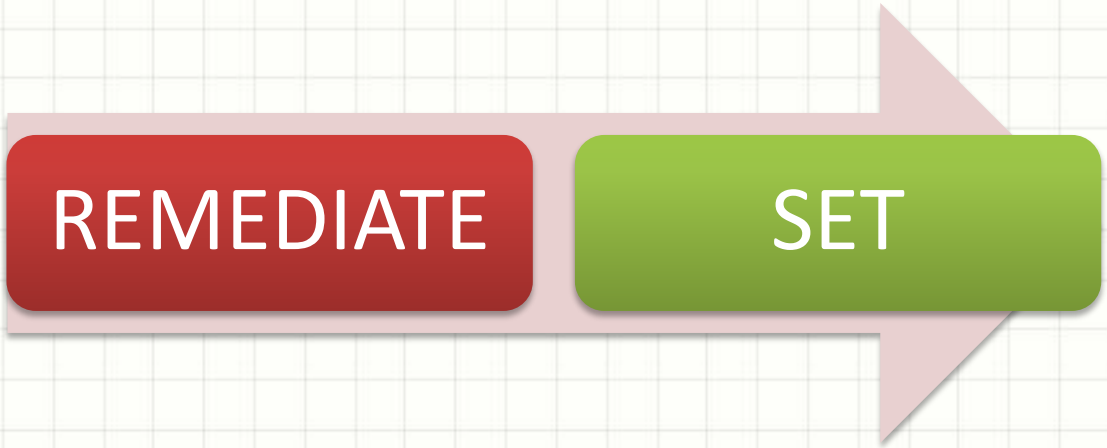
- Currently CarbonBlack executes all commands integrated into Phantom
- No problem integrating OpenC2 into Phantom
- Intend to “catch up” ePO to the same state as CarbonBlack

Use Cases – CTO – Block auto-run

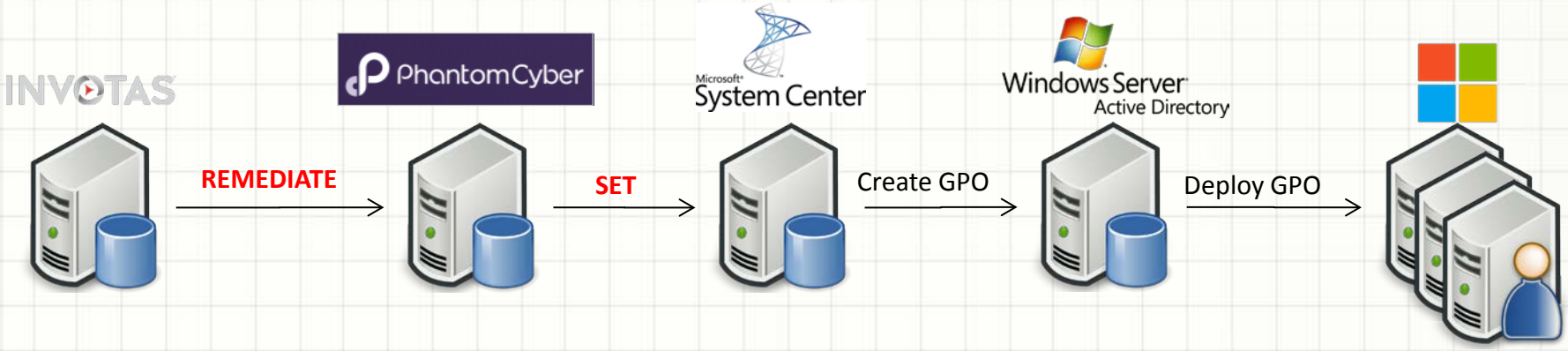
- Command Task Order scenario
- What does it mean to take an abstract, generic OpenC2 command from a trusted authority and actually implement it with the proper intent in a subordinate enterprise?
- Orchestrator to orchestrator
  - Invotas → Phantom → SCCM → DC → Windows Domain PCs

Use Cases – CTO – Block auto-run

- Programmatically create and deploy GPO to block auto-run in a Windows environment
- Updates registry on workstations through configuration management



Use Cases – CTO – Block auto-run



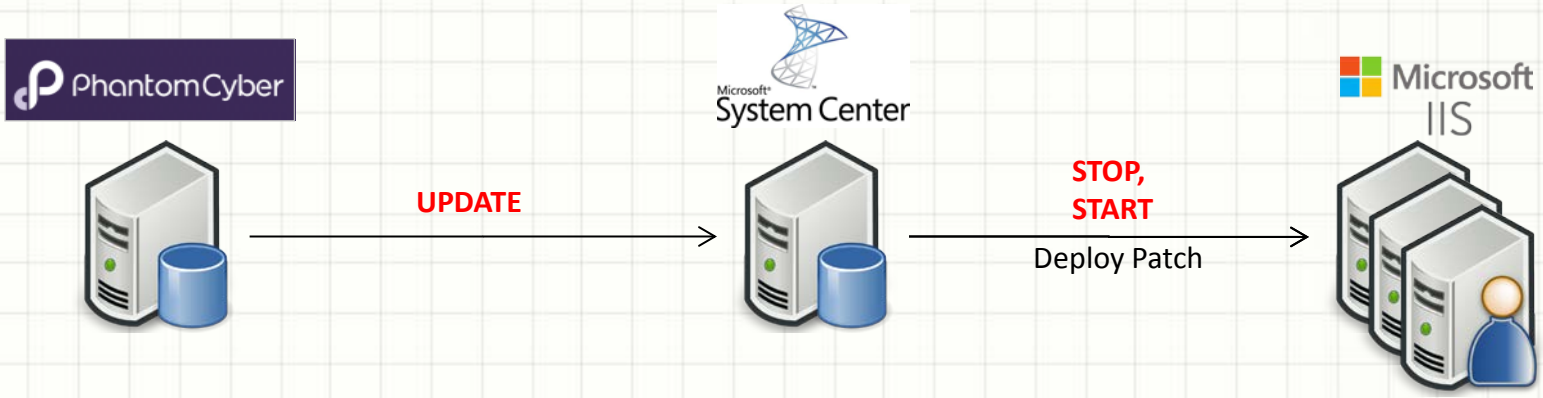
- Invotas sends REMEDIATE to Phantom to block autorun across the enterprise
- Phantom sends SET to System Center Configuration Manager to create a GPO

Use Cases – Automated Patching

- Given a vulnerable web server (IIS)
  - Determine if a patch is available
  - Patch it

- Phantom → Microsoft SCCM

Use Cases – Automated Patching



- Phantom sends a STOP to IIS to stop the web server
- Phantom sends UPDATE to System Center Configuration Manager to patch IIS
- Phantom sends START to IIS to bring it back online

Findings – Verbosity

- Originally agreed to flexibility in design
- Eschewed
  - XML in favor of JSON
  - Aligning field names with CybOX object names
- JSON made the Python library significantly easier to develop
- Using fields:
  - Significantly cut down on CybOX verbosity
  - Added flexibility for different transports
  - Easily mapped intent back to CybOX without using it

Findings – Flexibility

- Started implementing by mapping to CarbonBlack's REST API
- Migrated to using its Python API
- OpenC2 commands did not change
  - Mappings are not always 1:1
  - Currently requires a robust, open, well documented API

Findings – Intent

- Some commands need more detail or require explicitly defined default values
- Consider blocking an IP at a firewall
  - Given currently required elements of OpenC2, how?
    - Silently drop packets
    - Block with icmp
- Bottom line: some commands either need to require specific modifiers or need explicitly defined default behaviors

Findings – Responses

- Limited definition and discussion on handling responses to OpenC2 commands
- Originally ignored RESPONSE due to lack of discussion in working group
- Even as specified, combinations of responses need to be considered:
  - Where to redirect output of executed command (acknowledging INVESTIGATE's modifier "report-to")
  - Where to respond with status of command
- Effectively, how should we nest transport-agnostic (message bus queue name?) response requirements (status and output to different places) into one command?

Findings – Responses

- How should we handle uniquely identifying OpenC2 commands?
  - Metadata wrapper object
  - Packet header
- We used:
  - Metadata JSON wrapper with OpenC2 command nested in
  - Added UUID and more abstract reply-to fields
- Implemented actions/responses with a message fabric
  - Commands go out on a topic logically separating classes of actuators
  - Response queue name aligned with the UUID of the command

Findings – Responses

- OpenC2 responses…
  - Where (already covered)
  - How?

- RI: redirect output from an API response back to the orchestrator
  - Standardizing response payload will be a significant challenge across the vendor space
  - Leaves text or object parsing up to the orchestrator
  - Might be out of scope for the OpenC2 language, but worth noting nonetheless

QUESTIONS?