SODA

A System for Cyber Deception Orchestration and Automation

Md Sajidul Islam Sajid*, Jinpeng Wei*, Basel Abdeen†, Ehab Al-Shaer‡, Md Mazharul Islam*, Walter Diong‡, Latifur Khan†

University of North Carolina at Charlotte*
University of Texas at Dallas†
Carnegie Mellon University ‡

UNIVERSITY OF NORTH CAROLINA CHARLOTTE | COLLEGE OF COMPUTING AND INFORMATICS

Demo can be found here: https://youtu.be/Xv8PkUJ72d0

Agenda

- Motivation
- System Overview
    - Deception playbook creation
    - Real time orchestration
- Evaluations
- Conclusion
- Q&A

Demo can be found here: https://www.youtube.com/watch?v=Xv8PkUJ72d0

Motivation

Malware attacks leverage the asymmetry in cyber warfare

- **Role:** Adversaries can inflict harm on systems, but defenders are limited on defending them.
- **Fe**
- **Ki**
- **C** ck path, but th

We can not simply rely on traditional detection system!
A deception-oriented system can complement the traditional defense mechanisms to overcome their limitations

Malv rtunity" to at

- **Di**
- **Distortion**: generating ambiguity in the attacker's mind about what is and is not real ➔ to slow down the attacker.
- **Depletion**: consuming the attacker's resources ➔ to inflict harm and increase attack cost.
- **Discovery**: enabling the malware to progress in order to learn the motive/intention/goal of the attack

Motivation

Understanding malware behavior on the runtime.

- **MSGs:** Finding out Malicious Subgraphs (MSGs) responsible for malicious behaviors.
- **MSG-to-MITRE mapping:** Mapping MSGs to the MITRE ATT&CK framework to determine the malware's behaviors at the kill chain tactical level which helps selecting correct deception actions.

Automatic orchestration

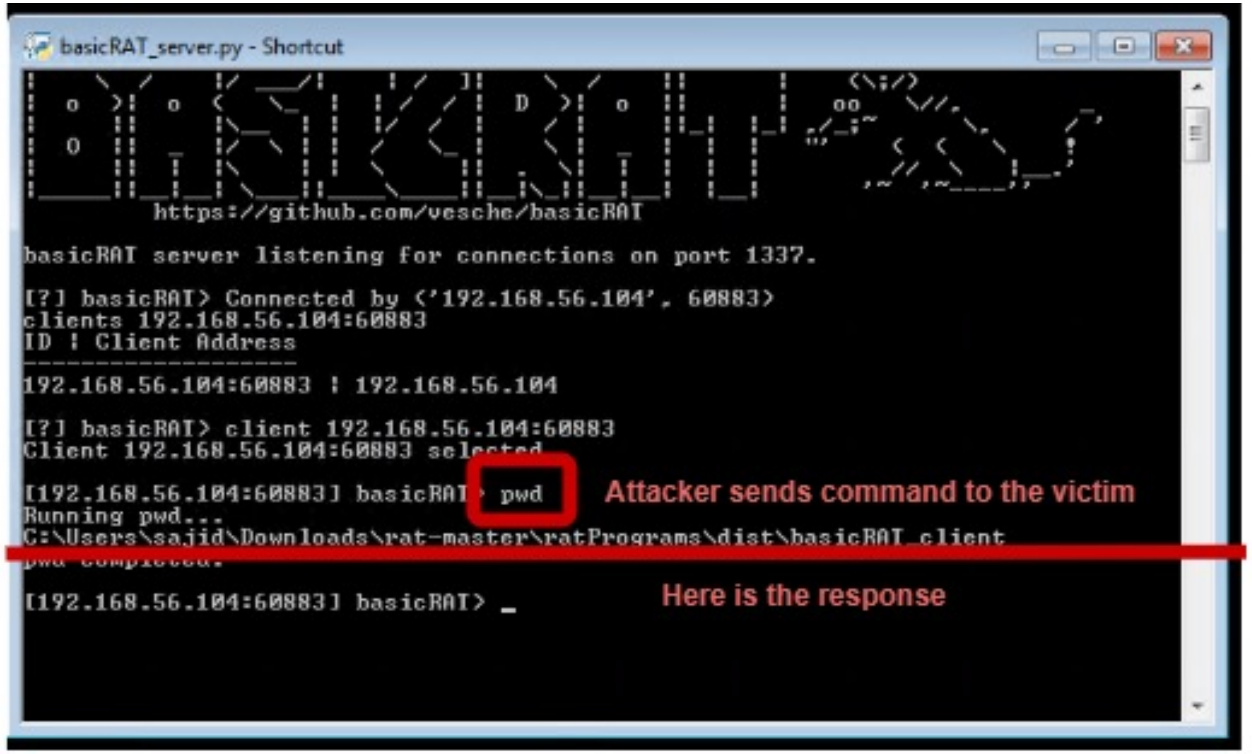
- **Manual:** Existing deception approaches are manual and lack in agility and automation.
- **Customization:** Existing deception approaches (example: **DodgeTron***) are mostly rule based and do not give the users option to customize them accordingly to their need. SODA provides an automated customizable deception playbooks against arbitrary malware.

*** Dodgetron: Towards autonomous cyber deception using dynamic hybrid analysis of malware. In *2020 IEEE Conference on Communications and Network Security (CNS)* (pp. 1-9). IEEE.**

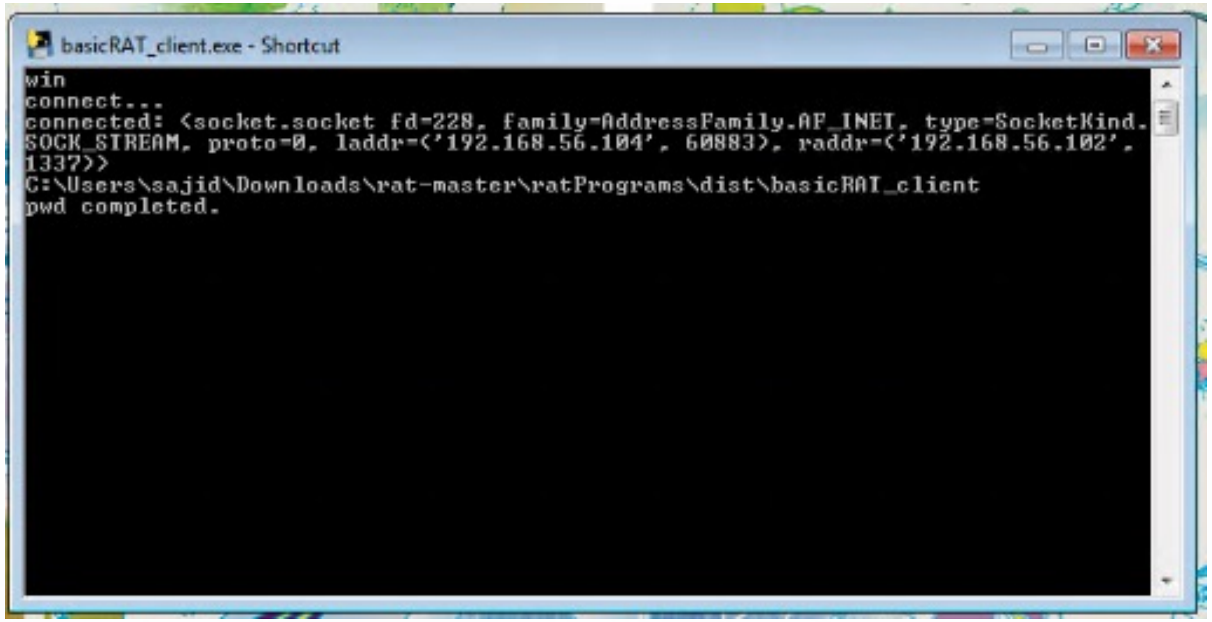
Contributions

- We propose a dynamic security orchestration, automation, and deception system, SODA, enabling users to orchestrate deception ploys with appropriate strategies and goals dynamically.
- We propose an automated MSG extraction and MSG-to-MITRE mapping, allowing SODA to understand malware behaviors at the run time to activate relevant deception ploys.
- We propose an embedded deception technique based on API hooking, allowing SODA to execute deception ploys in real-time.
- We evaluated SODA with recent malware to determine the accuracy and the scalability of our approach. We observed an accuracy of 95% in deceiving malware with negligible overhead and deployment time. Furthermore, our approach successfully extracted MSGs with a 97% recall value and MSG-to-MITRE achieved a top-1 accuracy of 88.75%.

Screenshots (Scenario 1: Regular attack without SODA)



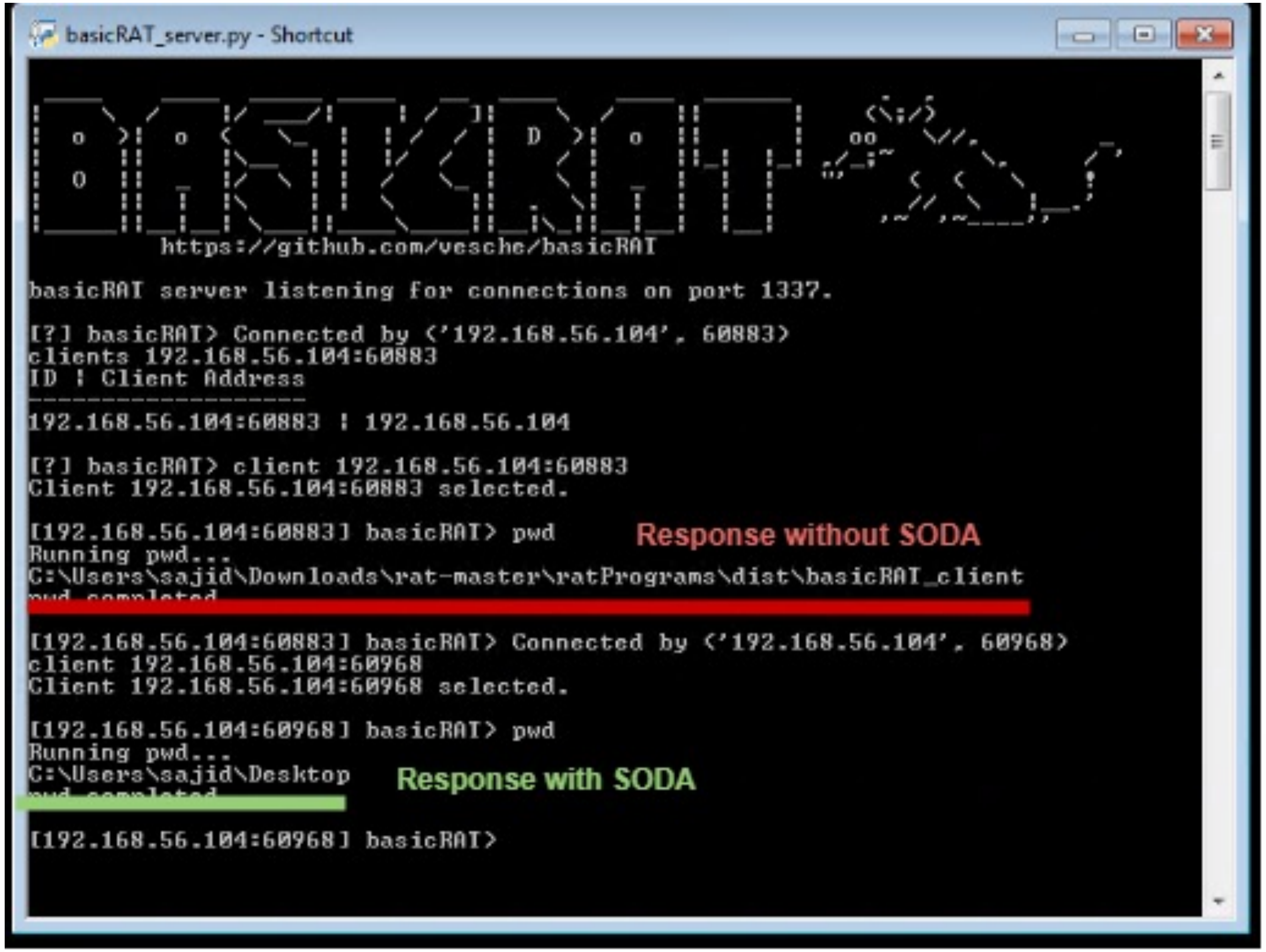
Malware – Attacker's side



Malware – Victim's side

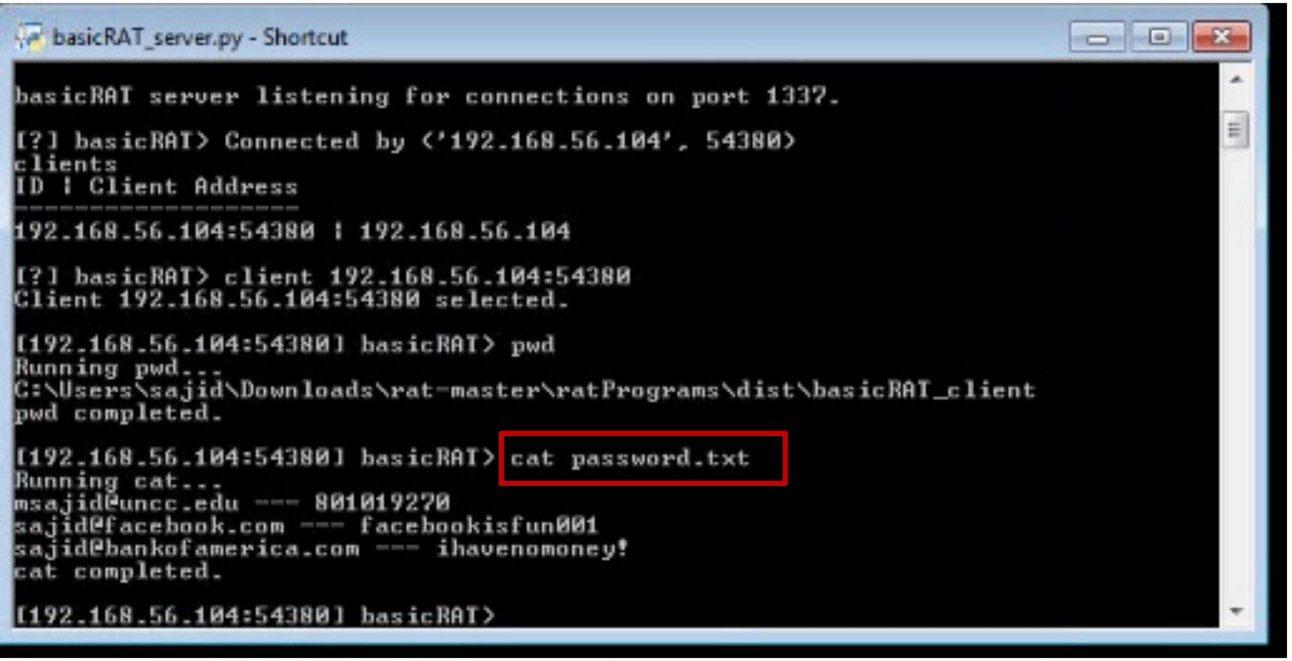
Attacker is trying to identify the "current working directory"

Screenshots (Scenario 1: With SODA)

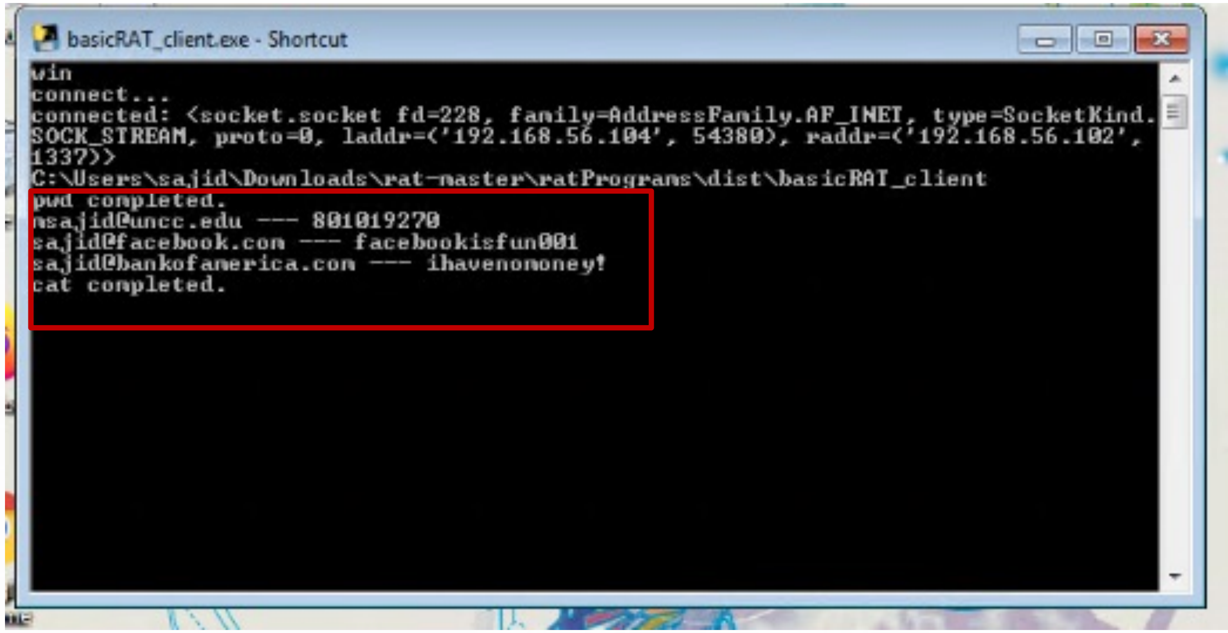


Malware – Attacker's side after SODA in action

Screenshots (Scenario 2: Regular attack without SODA)



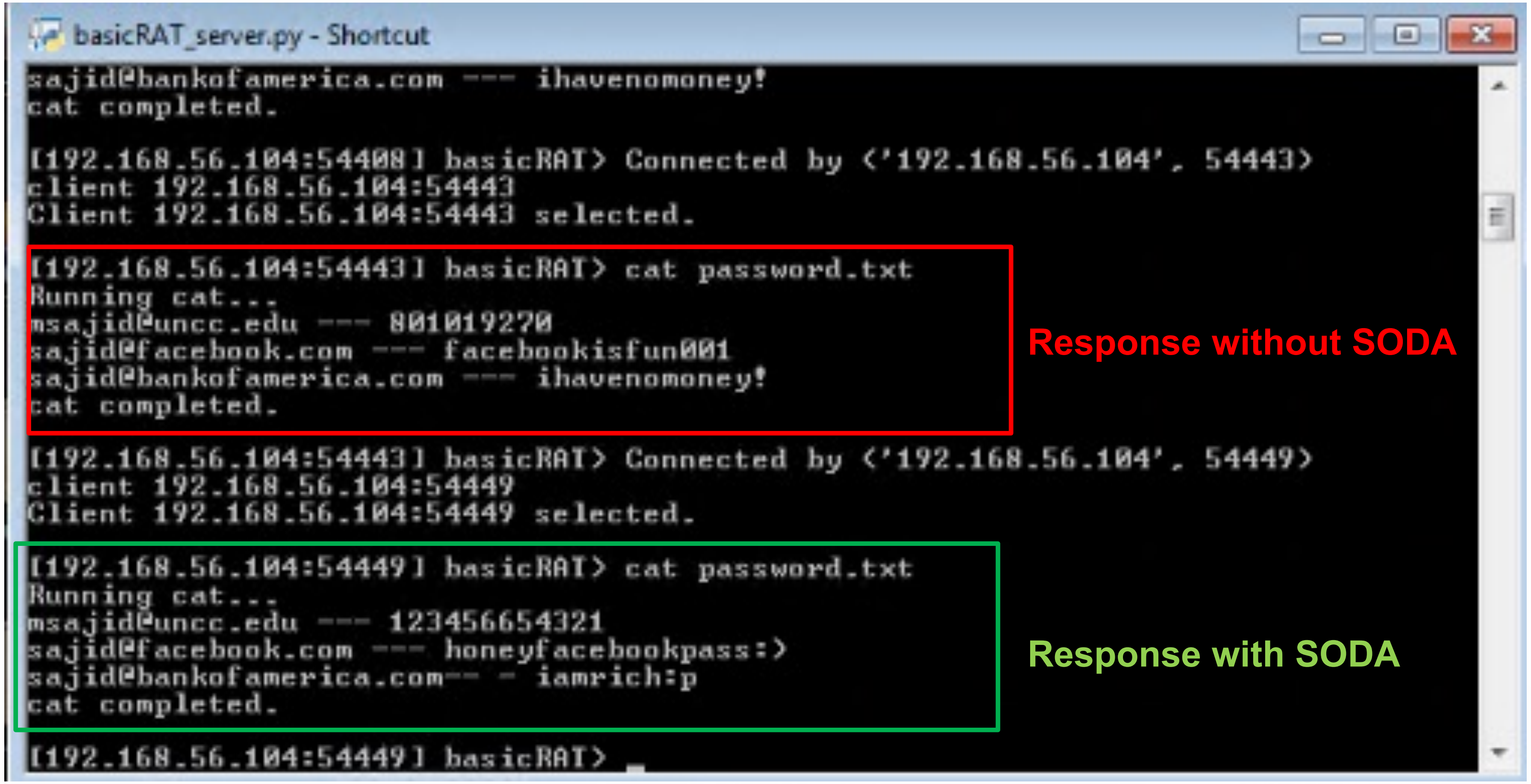
Malware – Attacker's side



Malware – Victim's side

Attacker is trying to "steal passwords" from well known files

Screenshots (Scenario 2: With SODA)



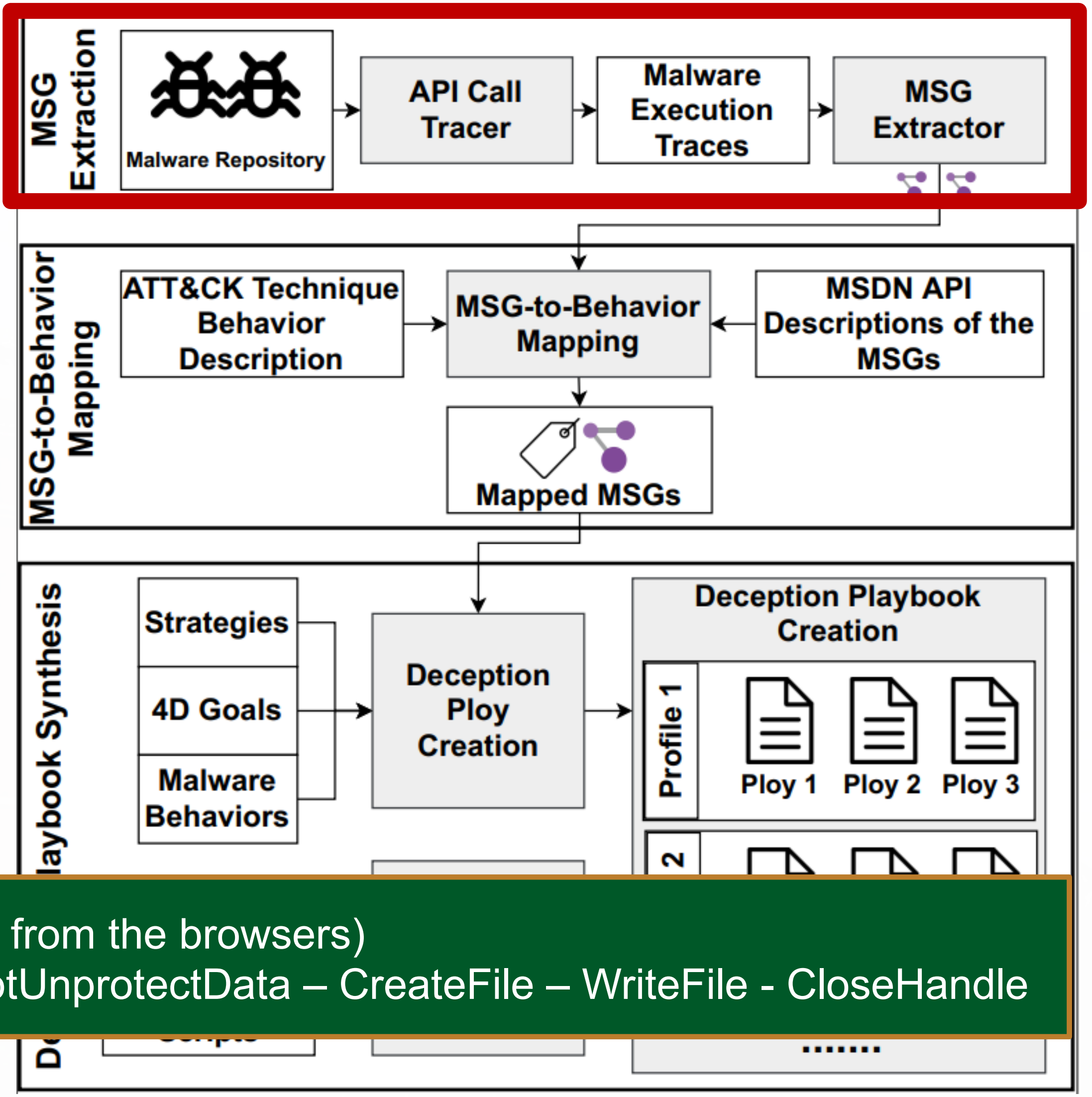
Response without SODA

Response with SODA

Malware – Attacker's side after SODA in action

SODA: DECEPTION PLAYBOOK CREATION

- Extract Malicious subgraphs (MSGs)
  - A malicious sub-graph (MSG) represents a sequence of WinAPI calls that work together to perform a malicious task.
- Map MSGs to MITRE/Malware behaviors to understand malware goals.
- Create deception ploys based on deception 4D goals and strategies
- Create deception playbook profiles
  - Each profile incorporates ploys for the behaviors that are likely to happen together
- Implement deception ploys inside API hook functions

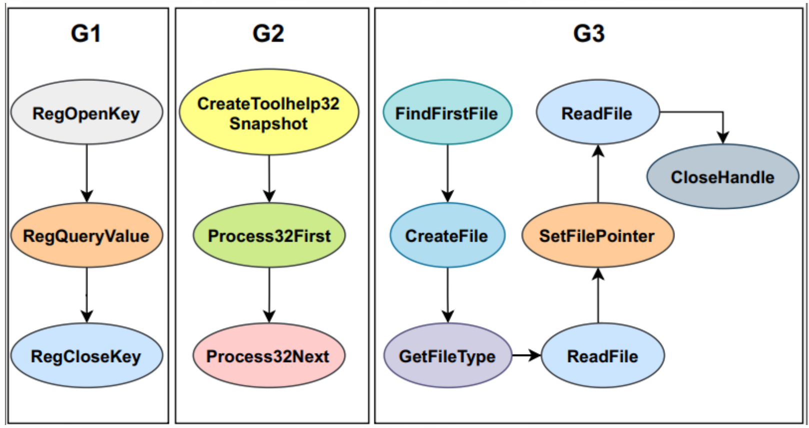


MSG example: (Steal from the browsers)
CreateFile – GetFileSize – VirtualAlloc – ReadFile – CryptUnprotectData – CreateFile – WriteFile - CloseHandle

MSG Extraction

```
RegOpenKey {'key_handle': '0x000000bc', 'regkey': 'HKEY_LOCAL_MACHINE\\SOFTWARE\\Mozilla\\Mozilla Firefox'}
RegQueryValue {'key_handle': '0x000000bc', 'value': '41.0.2 (x86 en-US)', 'regkey': 'HKEY_LOCAL_MACHINE\\SOFTWARE\\Wow6432Node\\Mozilla\\Mozilla
Firefox\\CurrentVersion'}
RegCloseKey, 0, {'key_handle': '0x000000bc'}
......
CreateToolhelp32Snapshot{'Out_Handle': '0x000000f8'}
Process32First {'process_name': '[System Process]', 'snapshot_handle': '0x000000f8'}
Process32Next{'process_name': 'System', 'snapshot_handle': '0x000000f8'}
Process32Next ...
.....
FindFirstFile {'filepath': 'C:\\Users\\Administrator\\AppData\\Local\\Google\\Chrome\\User Data\\Default\\Login Data'}
CreateFile {'out_file_handle': '0x000000dc', 'create_disposition': 1, 'filepath': 'C:\\Users\\Administrator\\AppData\\Local\\Google\\Chrome\\User
Data\\Default\\Login Data'}
GetFileType {'file_handle': '0x000000dc'}
ReadFile {'file_handle': '0x000000dc', 'buffer':'FileContent', 'length': 260, 'offset': 0}
SetFilePointer {'file_handle': '0x000000dc', 'move_method': 0, 'offset': 4096}
ReadFile {'file_handle': '0x000000dc', 'buffer':'FileContent', 'length': 4096, 'offset': 0}
CloseHandle {'handle': '0x000000dc'}
......
```

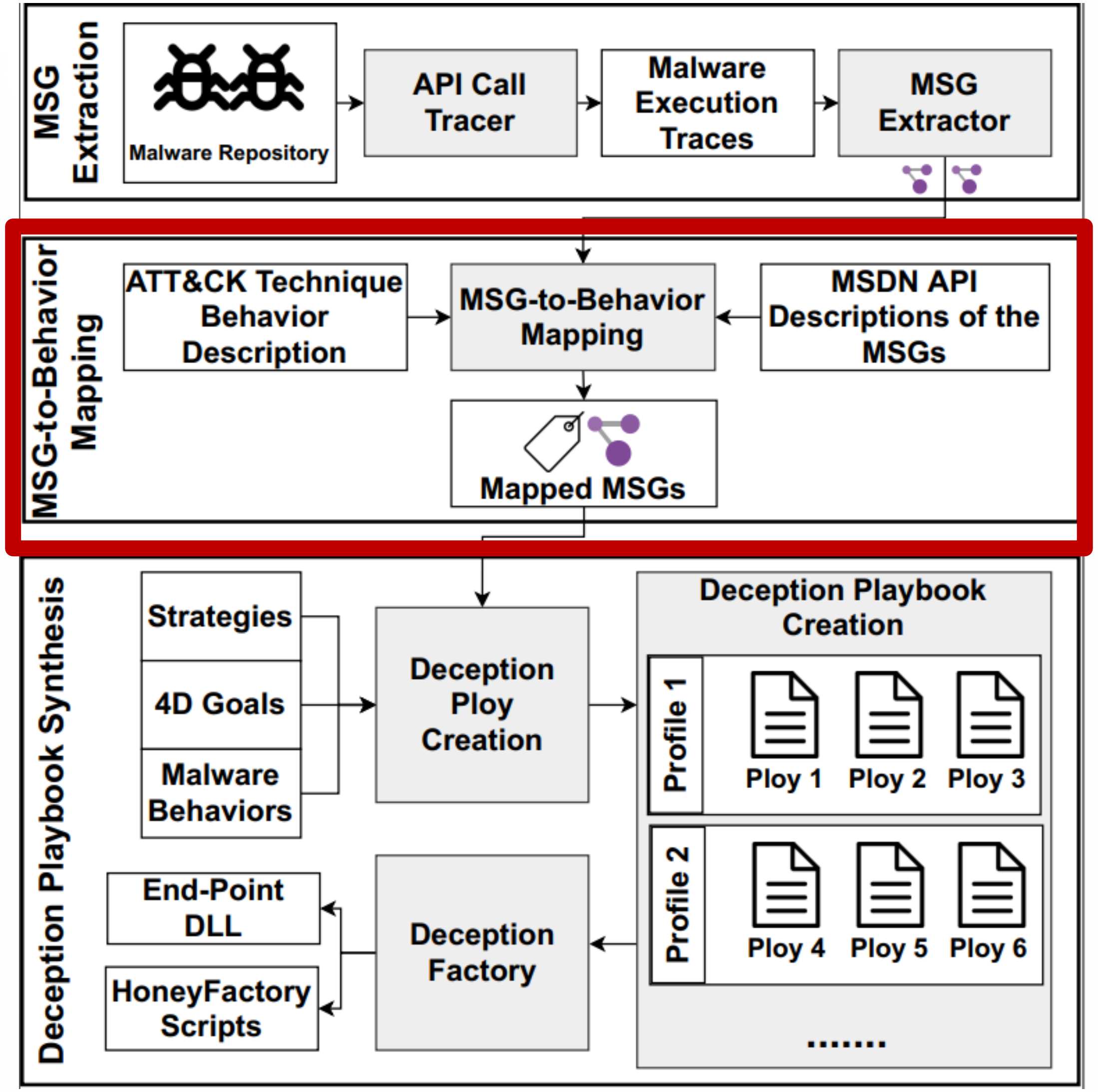
Malware execution trace log



MSGs extracted from the trace above

SODA: DECEPTION PLAYBOOK CREATION

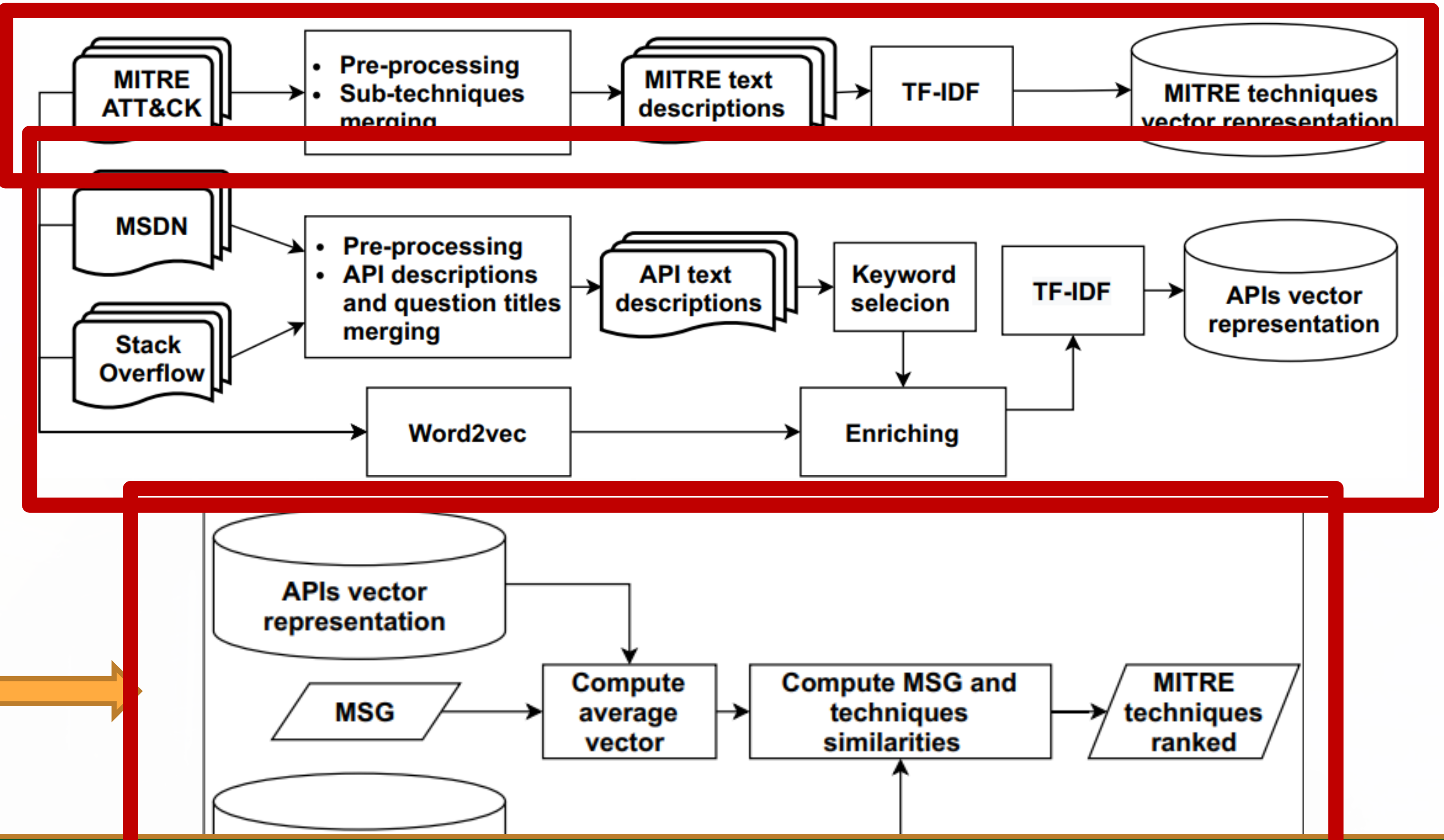
- Extract Malicious subgraphs (MSGs)
- Map MSGs to MITRE/Malware behaviors to understand malware goals.
- Create deception ploys to (connect it with goal) based on deception 4D goals and strategies
- Create deception playbook profiles
  - Each profile incorporates ploys for the behaviors that are likely to happen together
- Implement deception ploys inside API hook functions



How MSG-to-MITRE Mapping Works

MITRE techniques and APIs vector representation extraction

MSG to MITRE technique classification

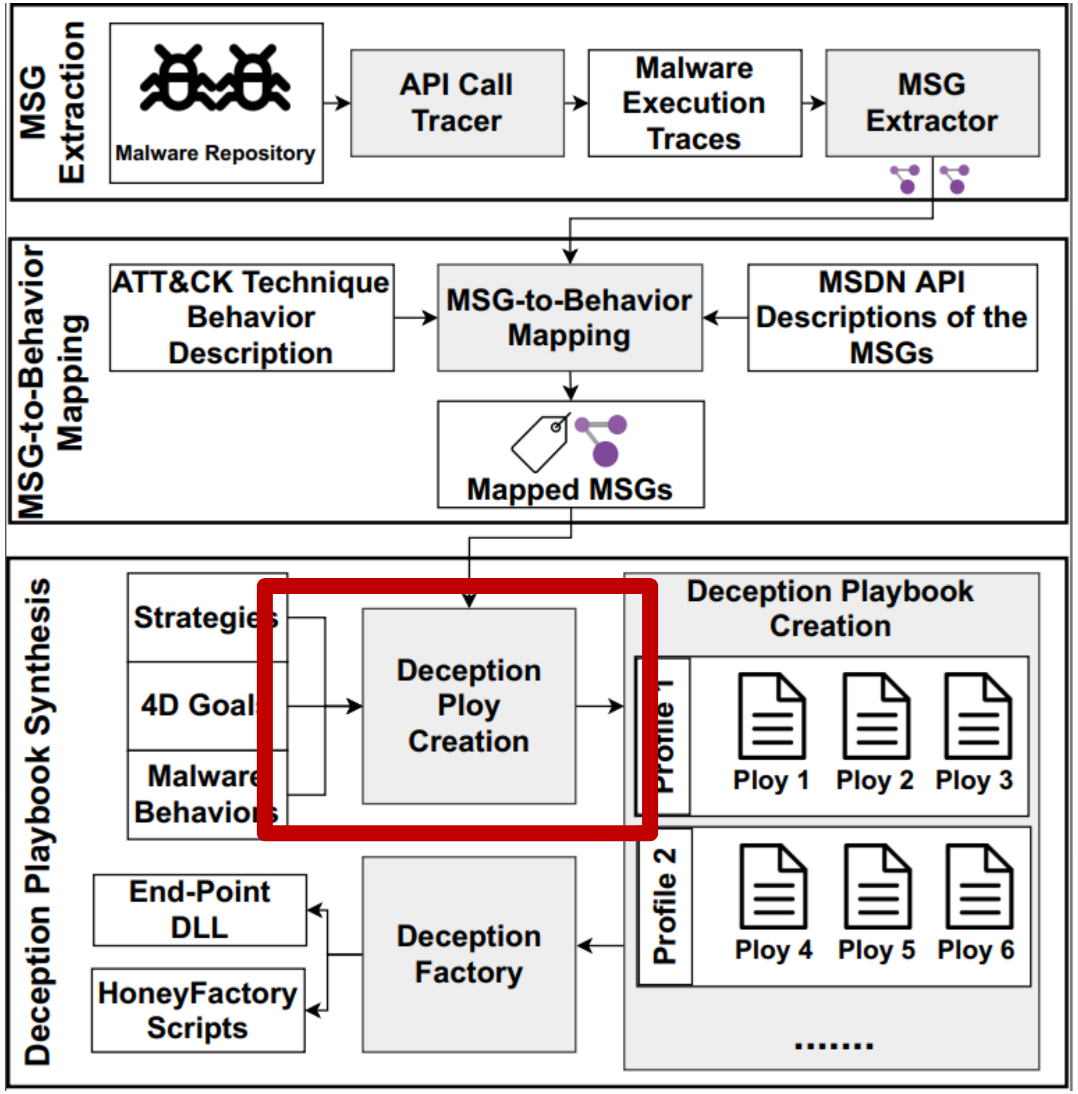


MSG example: CreatePipe – CreateProcess - ReadFile

1. Command and Scripting Interpreter
2. Ingress Tool Transfer (Execute file)

SODA: DECEPTION PLAYBOOK CREATION

- Extract Malicious subgraphs (MSGs)
- Map MSGs to MITRE/Malware behaviors to understand malware goals.
- Create deception ploys based on deception 4D goals and strategies
- Create deception playbook profiles
  - Each profile incorporates ploys for the behaviors that are likely to happen together
- Implement deception ploys inside API hook functions



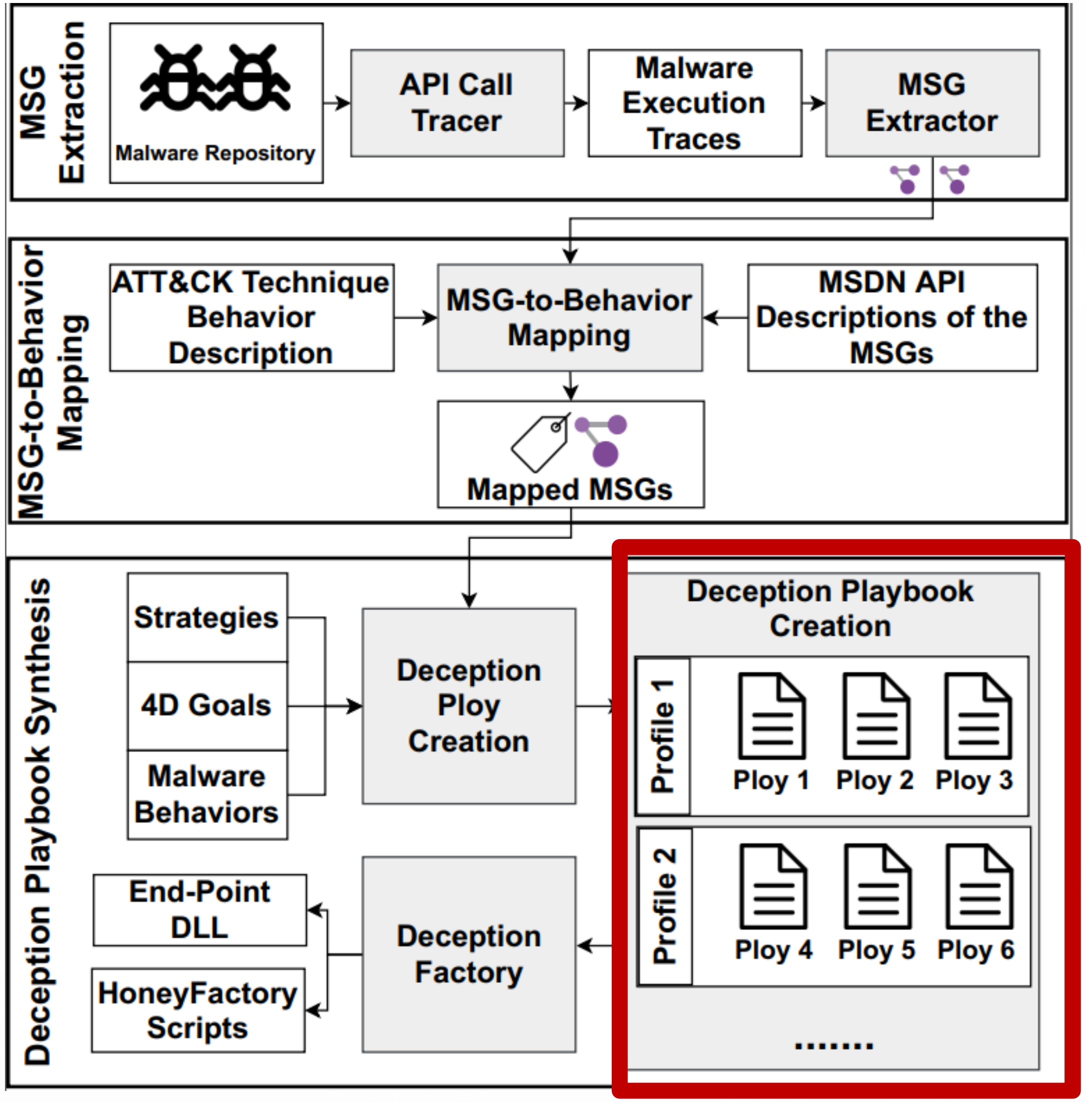
Deception ploys creation

Malware Behavior	Mapped MSG (API Sequence)	Strategy	Deception Goal				Deception Actions
			D_1	D_2	D_3	D_4	
Stealing from credentials files	1) Search file and steal: FindFirstFile, PathFileExist, CreateFile, GetFileSize, VirtualAlloc, ReadFile, CloseHandle, VirtualFree, FindNextFile, FindClose 2) Read sensitive file (known file). CreateFile, ReadFile 3) Steal from the browsers. CreateFile, GetFileSize, VirtualAlloc, ReadFile, CryptUnprotectData, CreateFile, WriteFile, CloseHandle	FakeFaliure	✓	-	-	-	Diversion: pretend the File doesn't exist by returning false when PathFileExist is called
		FakeSuccess	-	-	✓	✓	1) Depletion: replace sensitive file reading with static HoneyFile containing Honey Credentials. 2) Discovery: watch out for Exfiltration.
		FakeExecute	✓	-	✓	✓	1) Diversion: forward the execution to HoneyFactory (false target). 2) Depletion: communicate with HoneyFactory. Ask for HoneyFile containing Honey Credentials. Replace sensitive data with the content of HoneyFile. 3) Discovery: watch out for Exfiltration.
		NativeExecute	-	-	-	✓	Discovery: watch out for Exfiltration

Table 1: Deception ploy creation and verification (D_1 = diversion, D_2 = distortion, D_3 = depletion, D_4 = discovery).

SODA: DECEPTION PLAYBOOK CREATION

- Extract Malicious subgraphs (MSGs)
- Map MSGs to MITRE/Malware behaviors to understand malware goals.
- Create deception ploys to (connect it with goal) based on deception 4D goals and strategies
- Create deception playbook profiles
  - Each profile incorporates ploys for the behaviors that are likely to happen together
- Implement deception ploys inside API hook functions



Deception playbook profile creation

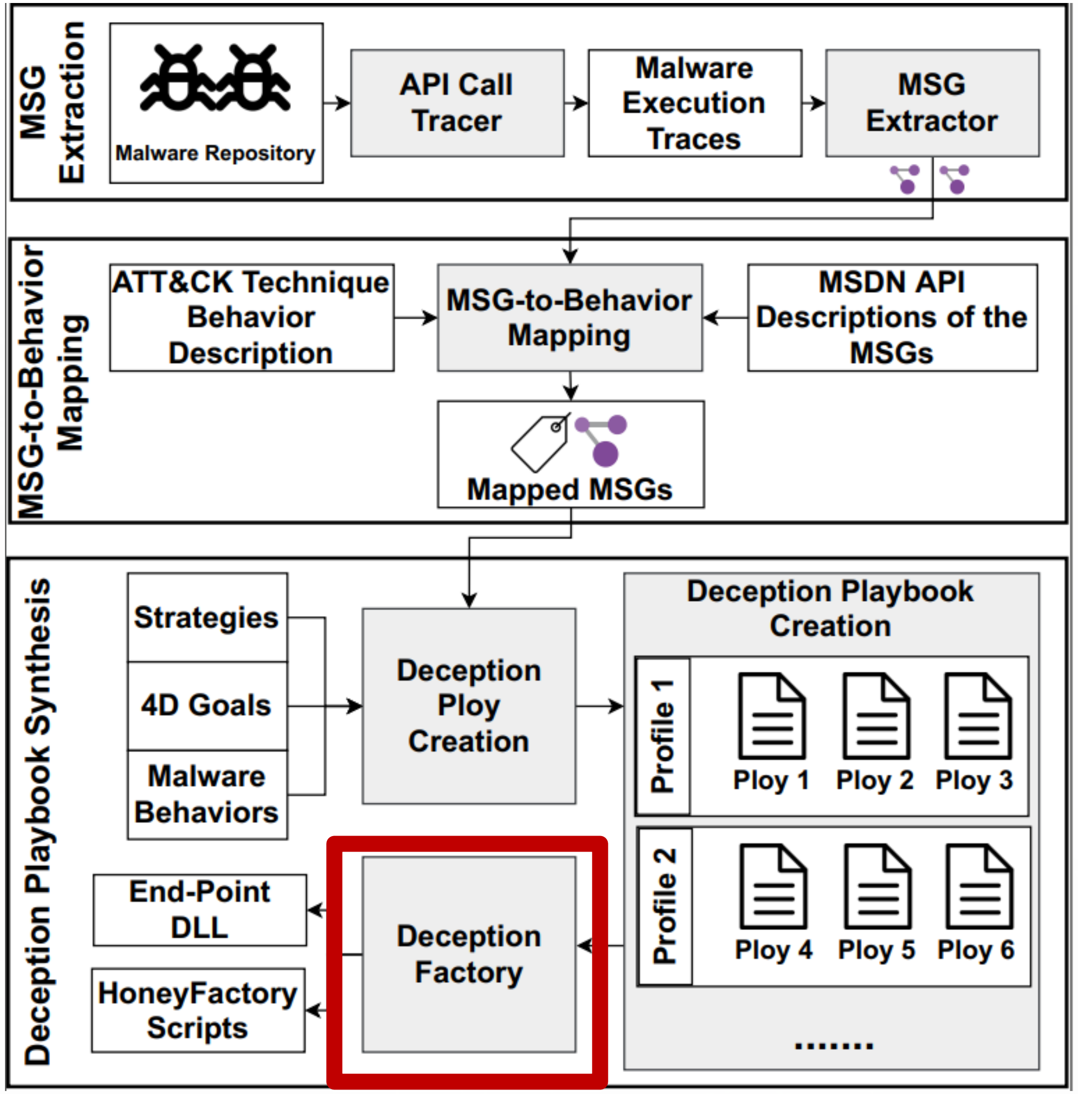
- We perform frequent item-set mining to identify highly associated MSGs.
- For example, if deception ploys $T1$,$T2$,$T3$, ...,$T6$ are created for malicious behaviors $B1$,$B2$,$B3$, ...,$B6$, respectively. If $B1$,$B2$ and $B3$ are in a frequent itemset then we create profile $P1$ containing $T1$,$T2$ and $T3$.
- The target is to provide a generic profile for each malware type such as InfoStealer, Ransomware, Spyware etc.
- However, user can create their own profile based on requirement.

$$Support(B) = \frac{Number\ of\ malware\ in\ which\ B\ appears}{Total\ number\ of\ Malware\ in\ the\ dataset}$$

$$Confidence(B1 \rightarrow B2) = \frac{Support(B1 \cup B2)}{Support(B1)}$$

SODA: DECEPTION PLAYBOOK CREATION

- Extract Malicious subgraphs (MSGs)
- Map MSGs to MITRE/Malware behaviors to understand malware goals.
- Create deception ploys to (connect it with goal) based on deception 4D goals and strategies
- Create deception playbook profiles
  - Each profile incorporates ploys for the behaviors that are likely to happen together
- Implement deception ploys inside API hook functions



Deception factory creation

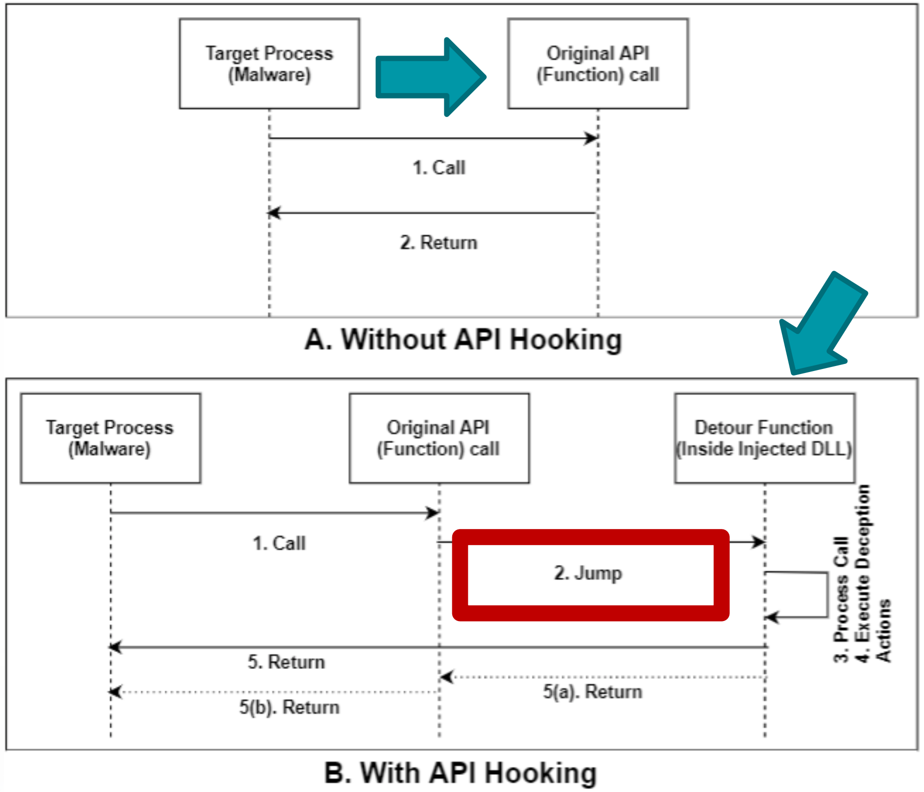


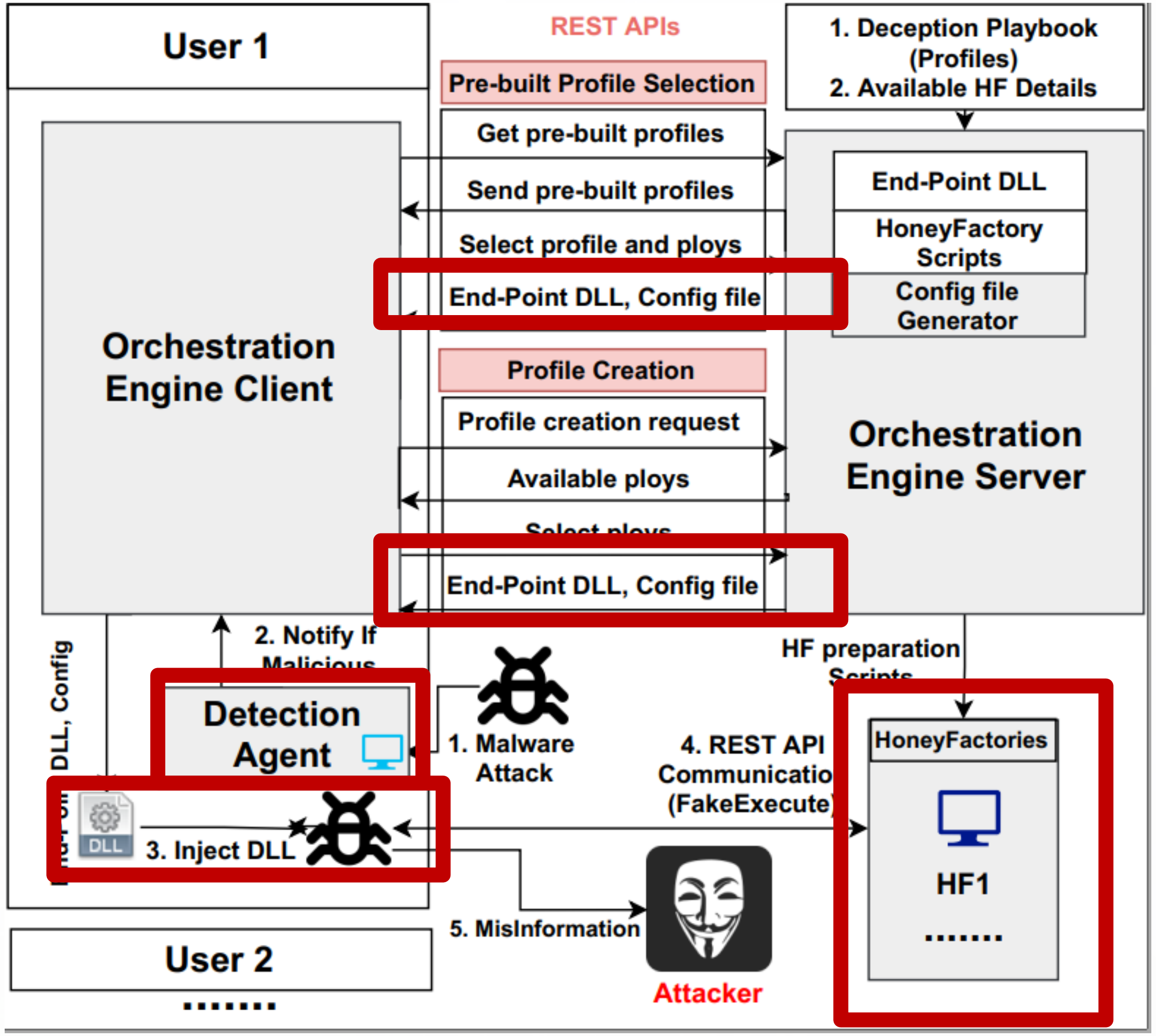
Figure 12: Call flow with and without API Hooking. In some case, Response is return via the original API (5a, 5b), otherwise, Detour function responds directly (5).

```
known_sensitive_files_browsers = {"Login Data", ....}
malicious_behavior = 0
buffer_under_observation = NULL
HookFunction_CreateFile(){
        Check if this Hook is in the activation_ploy_id List, If not exit without doing anything
                Check lpFileName //The name of the file opened
                it's in known_sensitive_files_browsers {
                        //(for internal understanding), 5 indicates stealing credent   ls from browsers
                        malicious_behavior = 5
                        Handle_under_observation = CurrentFile_Handle
                }
                else{
                        -----
                }
}

HookFunction_ReadFile(){
        Check if this Hook is in the activation_ploy_id List, If not exit without doing anything
                Check Handle //The name of the file opened
                If it's same as Handle_under_observation {
                        //this buffer holds sensitive data
                        buffer_under_observation = buffer_under_observation.append(lpBuffer)
                }
                else{
                        -----
                }
}

HookFunction_CryptUnprotectData(){
                                                                          hout doing anything
                If malicious_behavior == 5{

                        Check pDataIn->pbData and buffer_under_observation
                        If same{
                                // change the buffer value that holds the decrypted data
                                If the strategy is FakeExecute {

                                        honey_file_content_from_HoneyFactory =
                                                AlterData("context": "5",
                                                        "browser": "Google Chrome", "filename": "Login Data")
                                        pDataOut->pbData = honey_file_content_from_HoneyFactory

                                }
                }
}
```

Figure 7: A code snippet: How the deception technique is implemented inside API hooks

SODA: REAL-TIME ORCHESTRATION

- Playbook profile creation using a user interface
  - Requests are handled using REST APIs
  - Profile information is sent to Orchestration Engine Server (OES)
  - OES responses with End-point DLL and a configuration file
- Detection agent:
  - Detects the malware
  - Injects the End-point DLL into the malware
- Finally, Realtime deception takes place
  - Deception actions are implemented within embedded API-Hookings
  - Which deception actions to take is determined from the configuration file



Screenshots

C:/Users/msajid/Dropbox/transfer/table.html

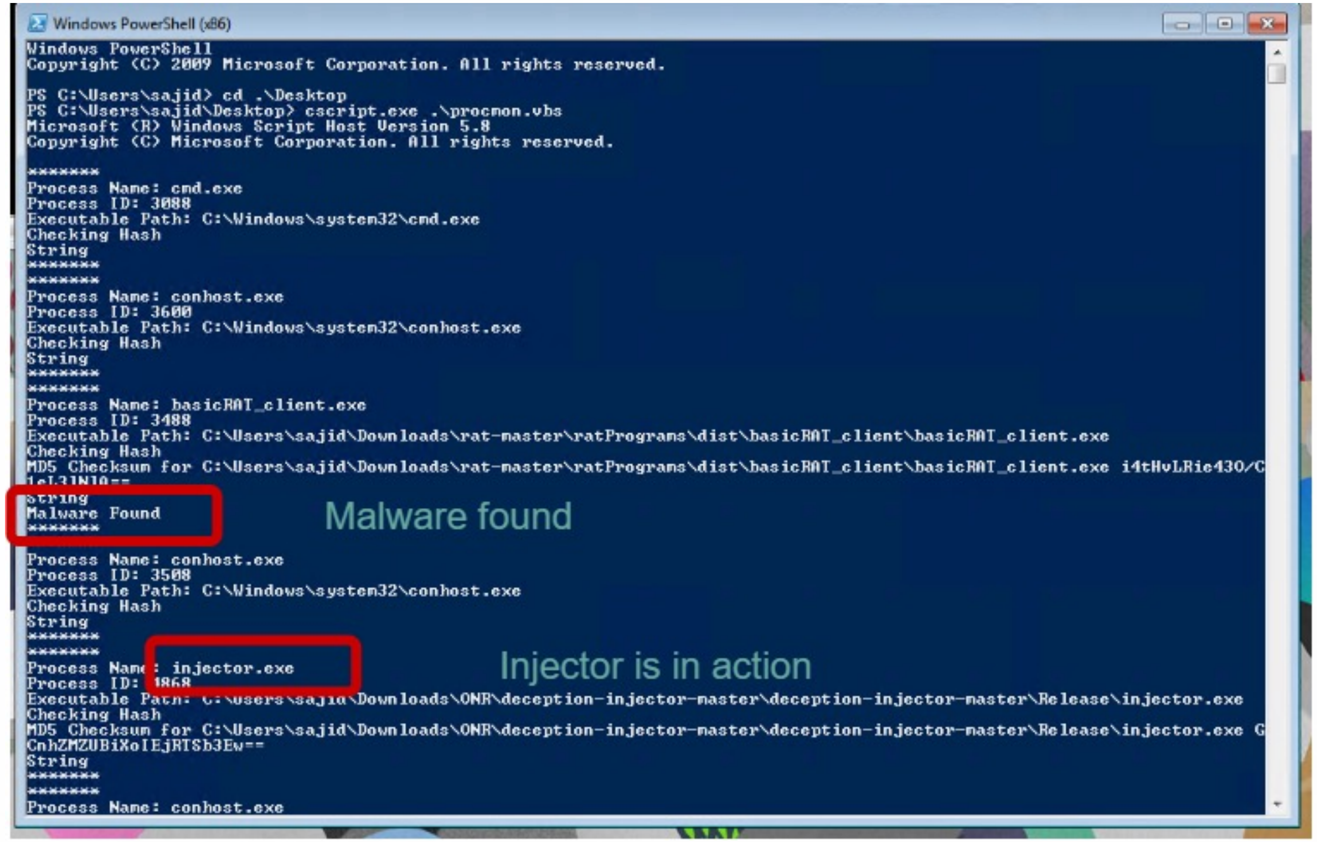
HoneyFactory
IP address: 192.168.56.102

Malware Behavior	Strategy	Goals			
		Diversion	Distortion	Depletion	Discovery
Remote execution	FakeFailure	○			
	FakeSuccess		○		○
	Honey	○			○
	Allow				◉
Screen Capture	FakeFailure	○			
	FakeSuccess		○		○
	Honey	○			○
	Allow				◉
Keylogging	FakeFailure	○			
	FakeSuccess		○		○
	Honey	○			○
	Allow				◉
Data collection from Clipboard	FakeFailure	○			
	FakeSuccess		○		○
	Honey	○			○
	Allow				◉
Stealing from credentials files	FakeFailure	○			
	FakeSuccess		○		○
	Honey	○			○
	Allow				◉
Read remote files	FakeFailure	○			
	FakeSuccess		○		○
	Honey	○			○
	Allow				◉

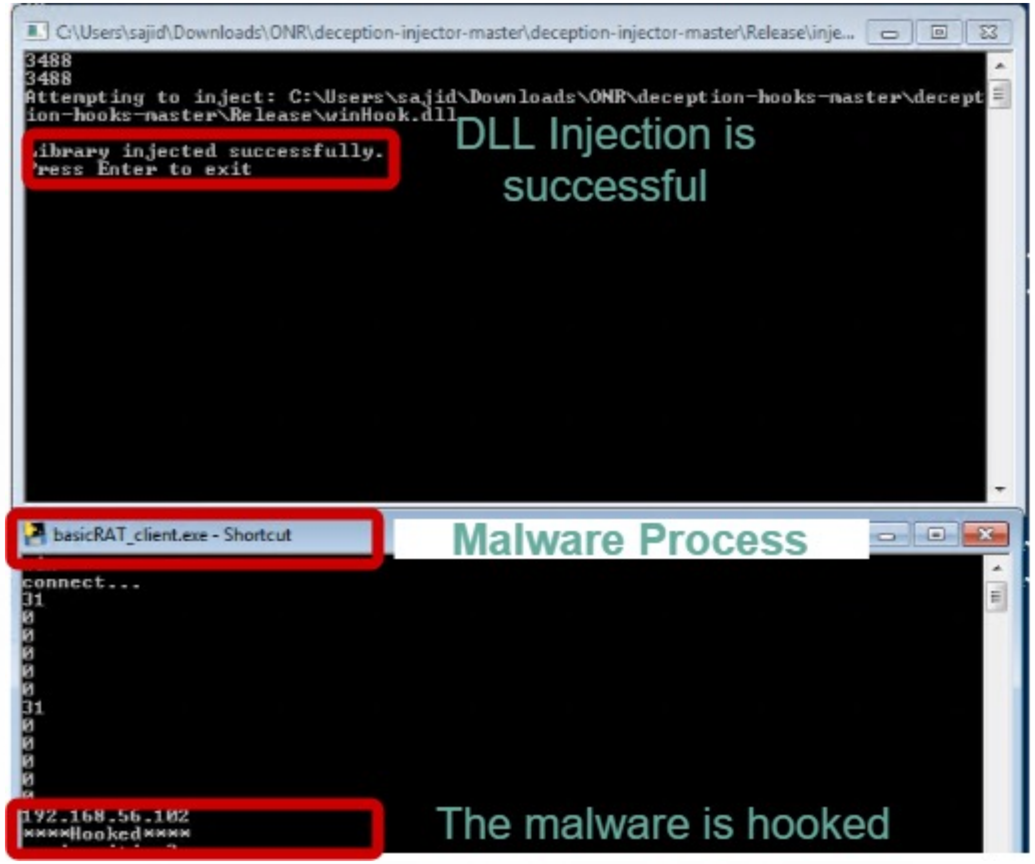
Submit

User Interface for Deception Playbook Profile creation

Screenshots (With SODA)



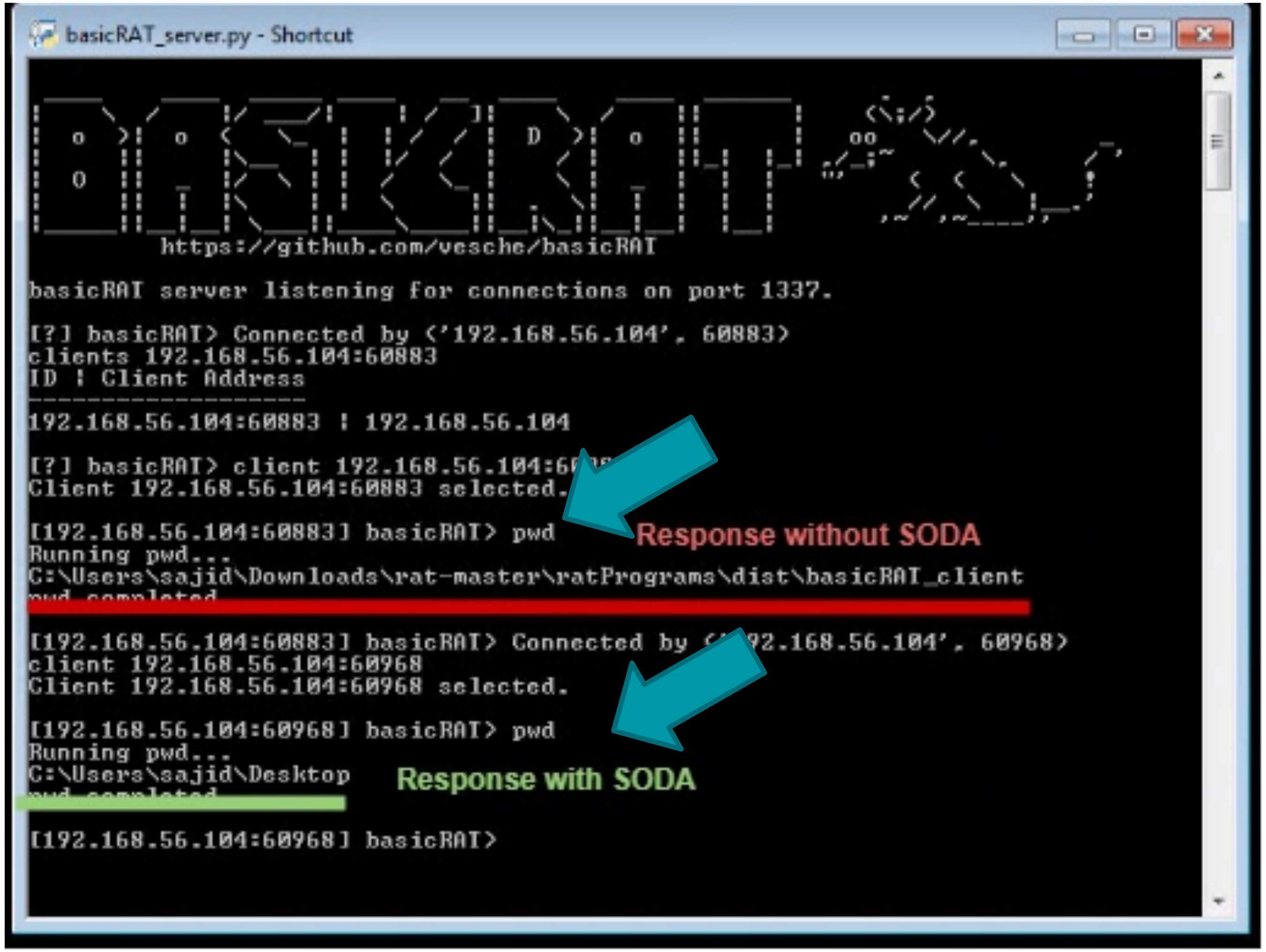
Detection Agent



End-Point DLL is injected

Screenshots (Scenario: With SODA)

- **Behavior:** List the current working directory
- **4D goal:** Diversion
- **Strategy:** FakeSuccess



Malware – Attacker's side after SODA in action

Evaluations

Recall of MSG extraction

- Ground-Truth (GT1):
  - We downloaded 42 malware source code from the GitHub along with the comments and descriptions explaining the malware's capabilities/behaviors.
  - Manually extracted 94 distinct MSGs.
- Experiment:
  - We obtain the binaries of these malware by building the source code
  - We run them in the API tracer (Our analyzer/Extended Cuckoo Sandbox)
  - Our MSG extractor was able to detected 113 unique MSGs.
  - We manually verified out of these 113, 91 of them are as expected (belonging to the GT1)

$$Recall_{GT1} = \frac{TP}{TP + FN} = \frac{91}{91 + 3} = 0.968$$

Evaluations

Comparison with other State-of-the-art tools and Sandboxes

Tools	Malware Behavior/Capability Identification (Using GT1)		
	Identified	Not Identified	Recall
Kris et al. [32]	58	36	0.62
FORECAST [11]	32	62	0.34
DodgeTron [37]	8	86	0.09
SODA	91	3	0.97

Comparison with other State-of-the-art tools in terms of discovering malware behaviors/capabilities using GT1 and their individual recall values

Family	Malware Family	Discovery	Cuckoo	Any.run	SODA
InfoStealer	Fareit	T	8	7	8
		P	39	126	149
	LokiBot	T	7	2	11
		P	21	173	243
	Pony	T	8	4	17
		P	231	191	582
	Racoon	T	7	8	16
		P	45	23	51
Ransomware	Ryuk	T	6	3	6
		P	27	32	102
	GandCrab	T	8	10	10
		P	192	109	245
RAT	Gh0st	T	2	2	6
		P	4	57	69
	VanilaRat	T	1	0	12
		P	1	0	12
	Quasar	T	5	2	13
		P	14	4	16

Number of techniques (T) and procedures (P) discovered by SODA compared to Cuckoo sandbox and Any.run.

Evaluations

MSG Classifier Evaluation (MSG-to-MITRE)

- Ground-Truth (GT2):
  - We used Remote Access Trojans (RATs) to create this ground truth.
  - We downloaded 13 different RATs from GitHub, capable of performing 33 distinct malicious behaviors.
  - We ran each malicious behavior at a time and manually collected MSGs
  - We manually mapped each of these MSGs to MITRE
  - Summary: we obtained 80 MSGs (correspond to 33 distinct malicious behaviors) and mapped them manually to 31 MITRE techniques.
- Experiment:
  - We feed these MSG to the MSG classifier.
  - MSG-to-MITRE mapping achieved a top-1 accuracy of 88.75%.

Minimum word frequency	4
API TF-IDF enriching threshold	20%
Word2Vec similarity threshold	70%
Maximum number of words per MITRE technique	40

Table 4: MSG Classifier Parameters

Experiment	Excluding Stackoverflow	Excluding Enriching	After Result Analysis
Top-1 Accuracy	48.75%	58.75%	88.75%
Top-2 Accuracy	62.5%	73.75%	96.25%

Table 6: Top-n accuracies after analysis as well as excluding StackOverflow and enriching component.

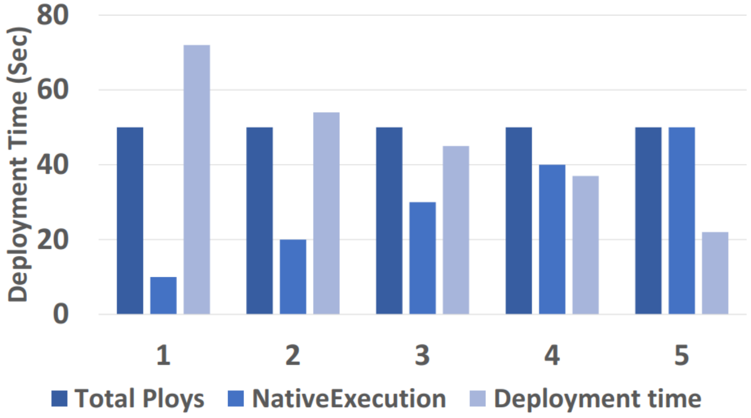
Evaluations - Performance Analysis of SODA

Deployment Time:

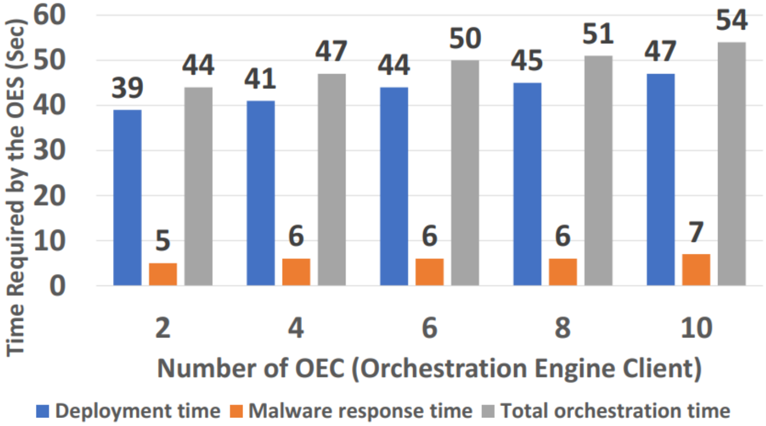
- Deployment consists the following three aspects:
  - generating the configuration file.
  - preparing necessary HF and
  - Forwarding the End-Point DLL and the configuration file to the OEC.
- Experimental setup and result:
  - User creates deception playbook profile with 50 deception ploys.
  - We performed the experiment 5 times by varying the ploys.
  - The maximum deployment time recorded is 72 sec.

Scalability Measurement:

- Simultaneous request from multiple clients
- Experimental setup and result:
  - We used the same profile to keep the consistency.
  - We performed the experiment using 2-10 clients.
  - **Note:** The execution time of the malware is 127 sec without SODA.
  - From this experimental result, we concluded that even though the orchestration time increased, OES still was able to serve its service successfully with negligible overhead (maximum of 7s) compared to the entire execution time of the malware (127s).



SODA deployment time with different ploys



Avg time of a single OES to orchestrate and serve multiple OECs

Evaluations - Performance Analysis of SODA

Overhead/Response delay time:

- We find out the overhead/response delay time of SODA by running the malware with and without SODA and the let the malware reach to the same execution point.
- We performed the experiment with four different types of malware (RAT, InfoStealer, Ransomware and Spyware)
- Our data shows that the maximum overhead time was 18 seconds (14% increment compared to the normal malware execution) which is minimal/insignificant compared to the total running/campaign period of an APT/malware.

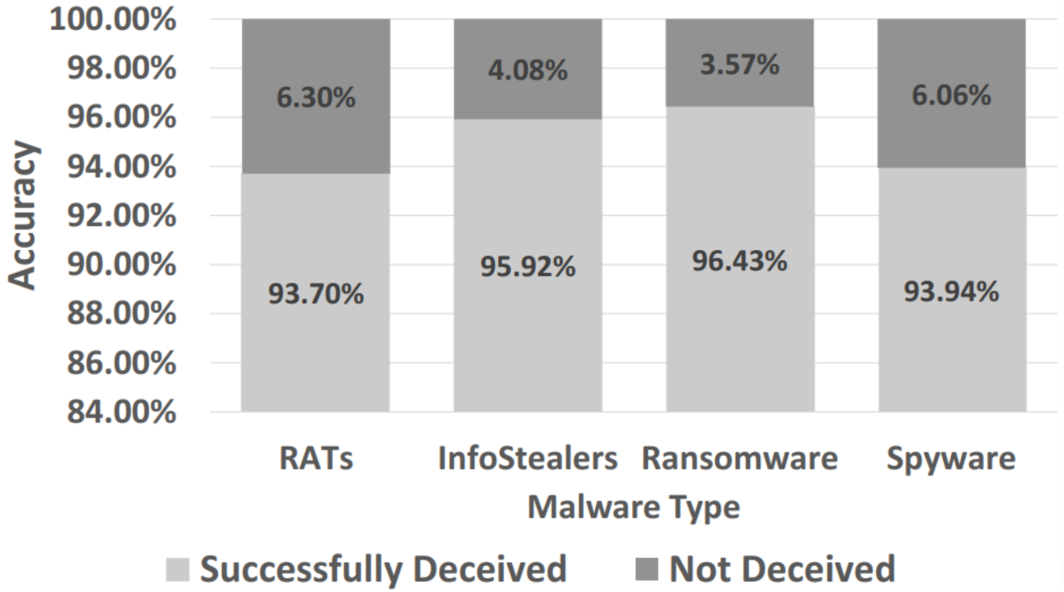
Malware Type	Focused Malicious Behavior	Strategy	Deception Goal	Deception Action	Expectation (Observance to consider deception ploy worked)	T1 (sec)	T2 (sec)	O (%)
RAT	Remote command Execution	FakeExecute	Depletion	Execute the remote command in HF and show it to the malware	Command executed in HF and is shown to the attacker (C&C server is in our control)	13	15	15%
InfoStealer	Steal credentials from the browsers	FakeExecute	Depletion	Show honey credentials from the HF	Honey credentials is seen to be exfiltrated (using packet capture)	38	43	13%
Ransomware	Encrypt files for Impact	FakeSuccess	Diversion	Pretend the encryption took place without performing it	This malware creates a ransom note after successful encryption (observe the note being created)	126	144	14%
Spyware	Capture screen	FakeExecute	Discovery	Capture screen from the HF and send it to the attacker	Captured screen of the HF is uploaded to our FTP server (redirected using ApateDNS)	61	65	7%

Table 7: Malware deception overhead (T_1 = time without deception, T_2 = time with SODA deception, O = Overhead).

Evaluations – End-to-End Accuracy of SODA

- We find out the E2E accuracy of SODA, we used 6 RATs with 37 distinct malicious behavior.
- Based of different deception strategies and goals, we identified 116 valid deception ploys that can be deployed to deceive these RATs.
- We observed SODA could deceive the RATs in 107 cases out of those 116 valid deception ploys.

Malware Type	Number of malware	Total number of valid deception ploys	Total number deception ploys where SODA successfully deceives malware
RATs	6	116	107
InfoStealers	122	49	47
Ransomware	96	28	27
Spyware	31	33	31
Total	**255**	**237**	**224 (95%)**



Accuracy of SODA across different malware types

Conclusion

- We propose an **autonomous cyber deception orchestration** system capable of analyzing real-world malware, discovering attack techniques, constructing Deception Playbooks, and orchestrating the environment to deceive malware.
- SODA **advances the state-of-the-art** by providing **dynamic real-time deception** and **customization** options to the users to choose their own deception ploys.
- Our proposed method of MSG extraction, followed by MSG-to-MITRE mapping, showed a promising result in **bridging the gap** between **malware traces** and the **MITRE ATT&CK framework**.
- Our extracted MSGs and MSG-to-MITRE mapping can play a vital role in **improving** the **existing tools**.
- We conducted rigorous evaluations to validate SODA's efficiency and scalability against 225 recent malware and observed:
    - An accuracy of **95%** in deceiving them.
    - A recall of **97%** in extracting MSGs
    - An accuracy of **88.75%** in mapping MSG-to-MITRE

Q&A