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‡

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=Xv8PkJ72d0
Motivation

Malware attacks leverage the asymmetry in cyber warfare

- **Role**: Adversaries can inflict harm on systems, but defenders are limited on defending them.
- **Feedback**: Malware detection/prevention provides advantage to adversaries to simply make a better one!
- **Knowledge**: 0-day malware have full knowledge for exploit, but we have 0-knowledge of their behavior.
- **Cost**: Defense is more expensive and it takes much more effort than attacks (defenders must block every attack path, but the malware only needs to find one).

Malware Deception as Game Changing: let’s consider a malware as “opportunity” to attack the adversaries rather than a “threat” (4D Deception Goals)

- **Diversion**: misleading the attacker to false targets → reduce attack exposure.
- **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

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

---

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

Attacker sends command to the victim

Malware – Victim's side

Here is the response

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)

Malware – Attacker’s side after SODA in action

Response without SODA

Response with SODA
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
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

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

<table>
<thead>
<tr>
<th>Malware Behavior</th>
<th>Mapped MSG (API Sequence)</th>
<th>Strategy</th>
<th>Deception Goal</th>
<th>Deception Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stealing from credentials files</td>
<td>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</td>
<td>FakeFailure</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Diversion: pretend the File doesn’t exist by returning false when PathFileExist is called</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FakeSuccess</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1) Depletion: replace sensitive file reading with static HoneyFile containing Honey Credentials. 2) Discovery: watch out for Exfiltration.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FakeExecute</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NativeExecute</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Discovery: watch out for Exfiltration</td>
</tr>
</tbody>
</table>

Table 1: Deception ploy creation and verification ($D_1 = \text{diversion}, D_2 = \text{distortion}, D_3 = \text{depletion}, D_4 = \text{discovery}$).
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 $T_1, T_2, T_3, ..., T_6$ are created for malicious behaviors $B_1, B_2, B_3, ..., B_6$, respectively. If $B_1, B_2$ and $B_3$ are in a frequent itemset then we create profile $P_1$ containing $T_1, T_2$ and $T_3$.
- 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.

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

$$\text{Confidence}(B_1 \rightarrow B_2) = \frac{\text{Support}(B_1 \cup B_2)}{\text{Support}(B_1)}$$
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).

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

#### User Interface for Deception

**Playbook Profile creation**

<table>
<thead>
<tr>
<th>Honey Factory</th>
<th>IP address: 192.168.56.102</th>
</tr>
</thead>
<tbody>
<tr>
<td>Malware Behavior</td>
<td>Strategy</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Remote execution</td>
<td>FakeFailure</td>
</tr>
<tr>
<td></td>
<td>FakeSuccess</td>
</tr>
<tr>
<td></td>
<td>Honey</td>
</tr>
<tr>
<td></td>
<td>Allow</td>
</tr>
<tr>
<td>Screen Capture</td>
<td>FakeFailure</td>
</tr>
<tr>
<td></td>
<td>FakeSuccess</td>
</tr>
<tr>
<td></td>
<td>Honey</td>
</tr>
<tr>
<td></td>
<td>Allow</td>
</tr>
<tr>
<td>Keylogging</td>
<td>FakeFailure</td>
</tr>
<tr>
<td></td>
<td>FakeSuccess</td>
</tr>
<tr>
<td></td>
<td>Honey</td>
</tr>
<tr>
<td></td>
<td>Allow</td>
</tr>
<tr>
<td>Data collection from Clipboard</td>
<td>FakeFailure</td>
</tr>
<tr>
<td></td>
<td>FakeSuccess</td>
</tr>
<tr>
<td></td>
<td>Honey</td>
</tr>
<tr>
<td></td>
<td>Allow</td>
</tr>
<tr>
<td>Stealing from credentials files</td>
<td>FakeFailure</td>
</tr>
<tr>
<td></td>
<td>FakeSuccess</td>
</tr>
<tr>
<td></td>
<td>Honey</td>
</tr>
<tr>
<td></td>
<td>Allow</td>
</tr>
<tr>
<td>Read remote files</td>
<td>FakeFailure</td>
</tr>
<tr>
<td></td>
<td>FakeSuccess</td>
</tr>
<tr>
<td></td>
<td>Honey</td>
</tr>
<tr>
<td></td>
<td>Allow</td>
</tr>
</tbody>
</table>

Submit
Screenshots (With SODA)

Detection Agent

Malware found

Injector is in action

End-Point DLL is injected

DLL Injection is successful

The malware is hooked

Malware Process
Screenshots (Scenario: With SODA)

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

<table>
<thead>
<tr>
<th>Family</th>
<th>Malware Family</th>
<th>Discovery</th>
<th>Cuckoo</th>
<th>Any.run</th>
<th>SODA</th>
</tr>
</thead>
<tbody>
<tr>
<td>InfoStealer</td>
<td>Fareit</td>
<td>T 8</td>
<td>7</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>P 39</td>
<td>126</td>
<td>149</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>T 7</td>
<td>2</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P 21</td>
<td>173</td>
<td>243</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>T 8</td>
<td>4</td>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P 231</td>
<td>191</td>
<td>582</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Raccoon</td>
<td>T 7</td>
<td>8</td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P 45</td>
<td>23</td>
<td>51</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ransomware</td>
<td>Ryuk</td>
<td>T 6</td>
<td>3</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>P 27</td>
<td>32</td>
<td>102</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>T 8</td>
<td>10</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P 192</td>
<td>109</td>
<td>245</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RAT</td>
<td>Ghost</td>
<td>T 2</td>
<td>2</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>P 4</td>
<td>57</td>
<td>69</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>T 1</td>
<td>0</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P 1</td>
<td>0</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>T 5</td>
<td>2</td>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P 14</td>
<td>4</td>
<td>16</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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%.

---

Table 4: MSG Classifier Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum word frequency</td>
<td>4</td>
</tr>
<tr>
<td>API TF-IDF enriching threshold</td>
<td>20%</td>
</tr>
<tr>
<td>Word2Vec similarity threshold</td>
<td>70%</td>
</tr>
<tr>
<td>Maximum number of words per MITRE technique</td>
<td>40</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Excluding Stackoverflow</th>
<th>Excluding Enriching</th>
<th>After Result Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top-1 Accuracy</td>
<td>48.75%</td>
<td>58.75%</td>
<td>88.75%</td>
</tr>
<tr>
<td>Top-2 Accuracy</td>
<td>62.5%</td>
<td>73.75%</td>
<td>96.25%</td>
</tr>
</tbody>
</table>
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).
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.

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

Table 7: Malware deception overhead ($T_1 = \text{time without deception}$, $T_2 = \text{time with SODA deception}$, $O = \text{Overhead}$).
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.

<table>
<thead>
<tr>
<th>Malware Type</th>
<th>Number of malware</th>
<th>Total number of valid deception ploys</th>
<th>Total number deception ploys where SODA successfully deceives malware</th>
</tr>
</thead>
<tbody>
<tr>
<td>RATs</td>
<td>6</td>
<td>116</td>
<td>107</td>
</tr>
<tr>
<td>InfoStealers</td>
<td>122</td>
<td>49</td>
<td>47</td>
</tr>
<tr>
<td>Ransomware</td>
<td>96</td>
<td>28</td>
<td>27</td>
</tr>
<tr>
<td>Spyware</td>
<td>31</td>
<td>33</td>
<td>31</td>
</tr>
<tr>
<td>Total</td>
<td>255</td>
<td>237</td>
<td>224 (95%)</td>
</tr>
</tbody>
</table>

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