A Collaborative Forensic Framework for Detecting Advanced Persistent Threats

Weifeng Xu
Forensic Science: Cyber Investigations
University of Baltimore
Baltimore, USA
wxu@ubalt.edu

Jie Yan
Department of Computer Science
Bowie State University
Bowie, USA
jyan@bowiestate.edu

Daryl Stone
Department of Technology and Security
Bowie State University
Bowie, USA
dstone@bowiestate.edu

***Abstract*—An advanced persistent threat (APT) is one type of cybercrime that steals valuable information over an extended period through malicious activities. The paper proposes a collaborative framework to systematically detect APTs by analyzing the Cyber Forensic Evidence (CFE) collected from a System Under Investigation (SUI). It is a post-compromise analysis based on Forensic-Evidence-Driven Finite State Machines (FED-FSM) modeled from an SUI. A FED-FSM extends an FSM by defining a set of forensic evidence patterns as guided conditions that trigger the state changes of FSM. The approach consists of three tasks (1) collecting shared CFE and formalizing patterns of CFE, (2) modeling the security status of an SUI in a FED-FSM, and (3) building a Threat Activity Detection Engine to match the observed CFE from SUI logs with the CFE patterns in the FED-FSM. An empirical study shows the framework can be used to detect malicious activities of Poison Ivy, which utilizes a remote access tool to control computers.**

Keywords—advanced persistent threat, Structured Threat Information Expression, finite state machine, threat detection, ATT&CK framework, digital forensics

I. Introduction

An advanced persistent threat (APT) is one of the cybercrimes, which typically refers to a state-sponsored hacking group and cyberattacks associated with the group. APT strategizes its way to infiltrate an organization's network and exfiltrates valuable information. For example, one common attack executed by APTs is to place custom malicious code on one or multiple computers for specific stealth tasks. Various approaches have been proposed to study and detect APT activities, either through scanning signatures of malicious code [1] [2] [3] or analyzing evidence that malicious programs are left on systems or networks activity logs [4] [5]. A common issue of these approaches is that they act alone during the whole investigation process: they set their environments, collect evidence, observe patterns, and make conclusions based on their judgments. It is hard for the cybersecurity community to recreate their environments, verify their observations, and reuse their observation results. MITRE [6][7] has proposed an Intelligence-sharing-based approach to fight against cybercrimes with collaborative efforts. It developed a concept called shared Cyber Threat Intelligence (CTI) to facilitate knowledge sharing. For example, Figure 1 shows a shared scenario which consists of two predefined objects *indicator* and *malware*. The figure means that a cybercrime investigator has reported that he/she has found malware, named Poison Ivy (PIVY). PIVY is a remote access tool used by many APTs and cybercriminals for information infiltration. To support his/her findings, the investigator attaches the hash code of the malware to the *indicator* object, which defines the possible threat patterns of malware, e.g., the SHA hash code of the PIVY executable file.

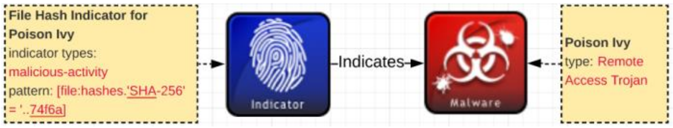


Figure 1. A simple indicator uses a file hash to indicate the presence of Poison Ivy

Shared CTI sounds promising for collaborative threat detection and therefore, has attracted much attention in cybersecurity communities. However, there are major challenges to apply them directly in practice due to: (1) Current CTI framework does not support presentable CTI in terms of forensic investigations. Detecting APT threats is the process of investigating cybercrimes with supporting digital evidence. A presentable shared CTI has to describe how threat evidence is acquired, preserved, identified, and validated to meet the requirements of the law, (2) the lack of a systematical approach to discover digital forensic evidence for a System Under Investigation (SUI). Current CTI only describes the evidence and the patterns of threat evidence using *Indicator*, it doesn't provide practical guidelines of how evidence can be extracted for a given SUI, and (3) Current CTI cannot describe the dynamical behaviors of an APT. Current CTI is a threat data model, which only describes static objects. To detect APTs, a dynamical model is needed to describe the known behaviors of APTs so that the model can be used for matching a given SUI.

The paper demonstrates a framework to detect APTs imposed on an SUI over time using Forensic-Evidence-Driven Finite state machines (FED-FSM). The approach first enhances CTI to Cyber Forensic Intelligence (CFI) and enables *indicator* objects to capture the properties of digital forensic evidence. The enhanced *indicator* object is called Cyber Forensic Evidence (CFE) object. We then propose a systematic approach to extract CFE-objects from an SUI and specify the SUI as a collection of CFE objects. Finally, we model the SUI in a FED-FSM with CFE objects as the model's guided conditions. We build a Threat Activity Detection Engine (TADE) to detect APTs that impose on the SUI by comparing the observed CFE objects with the

DOI reference number:10.18293/SEKE2021-052

CFE objects defined in FED-FSM. The contribution of the paper includes: (1) the formalization of CFE by leveraging ATT&CK knowledge model [6][7] (2) inferring dynamical behaviors of an APT using FED-FSM, and (3) designing a scalable threat-activity detection framework for detecting real-world APTs systematically.

The rest of the paper is outlined as follows: Sections II and III describe the difference between CTI and CFE and how CEF can be discovered systematically. Section IV shows the proposed architecture for detecting APT activities. Section V formally defines FED-FSM. Section VI describes an empirical study. Section VII summarizes the related work. Finally, Section VIII concludes this paper.

II. Shared-Cyber Threat Intelligence and Cyber Forensic Evidence

Intelligence-sharing is a critical strategy for cybersecurity defenders because it allows them to avoid the missteps of their peers within the security community and to deploy proven defensive measures. To systematically and effectively share security Intelligence, research groups [6][7] have realized that cybersecurity-related terminologies, measurements, and standards need to be defined to understand the security and share CTI. For example, to better understand security problems, MITRE has created an ATT&CK framework to document common Tactics, Techniques, and Procedures (TTP) based on real-world observations of adversaries' operations against computer networks. To share CTI, OASIS CTI technical committee [8] has developed Structured Threat Information Expression (STIX) to formalize CTI and enable interoperable sharing of CTI across organizational, technology, and geographic boundaries. STIX is considered a de facto standard for many industries and organizations [9]. Figure 2 shows a shareable CTI expressed in STIX objects based on Fireeye's Poison Ivy (PIVY) Report [10]. Colored icons in the figure represent various types of objects defined in STIX. Each STIX object defines various attributes to specify the object. For example, the indicator object shown in Figure 1 has two attributes, the type of indicator, e.g., malware activity, and the pattern of the indicator, i.e., the hash code of the malware file. Note that the PIVY data model shown in Figure 2 primarily relies on 25 indicator objects (i.e., blue circles with fingerprint symbols) to represent suspicious or malicious cyber activities. These indicators are spread out all over the model and there are no pre-defined semantic relations among them.

We extend shared CTI to CFE to align with the general process model of digital forensic investigation, including evidence collection, analysis, interpretation, and validation. Figure 3 shows the CFE object extends the Indicator object in CTI. The hollow arrow represents the extended relationship and the solid diamonds indicate collection tool, media, investigator, etc. are components of forensic evidence object. We use CFE object and its components to answer the questions related to the legal aspects of digital forensics [11] [12]:

- Where is forensic evidence collected, e.g., memory or disks? It can be addressed by Collection Tool and Media objects, which are used to describe the source of the evidence and how evidence is extracted and stored.
- Who does collect the evidence? When did a person access the original digital evidence? Is the person forensically competent? This information can be addressed by an investigator object.
- How does the analysis carry out? How to interpret the results? A solid forensic evidence analysis should be based on math and science. Besides the pattern match method defined in the Indicator object, we propose the Identification objects in CFE to record other methods

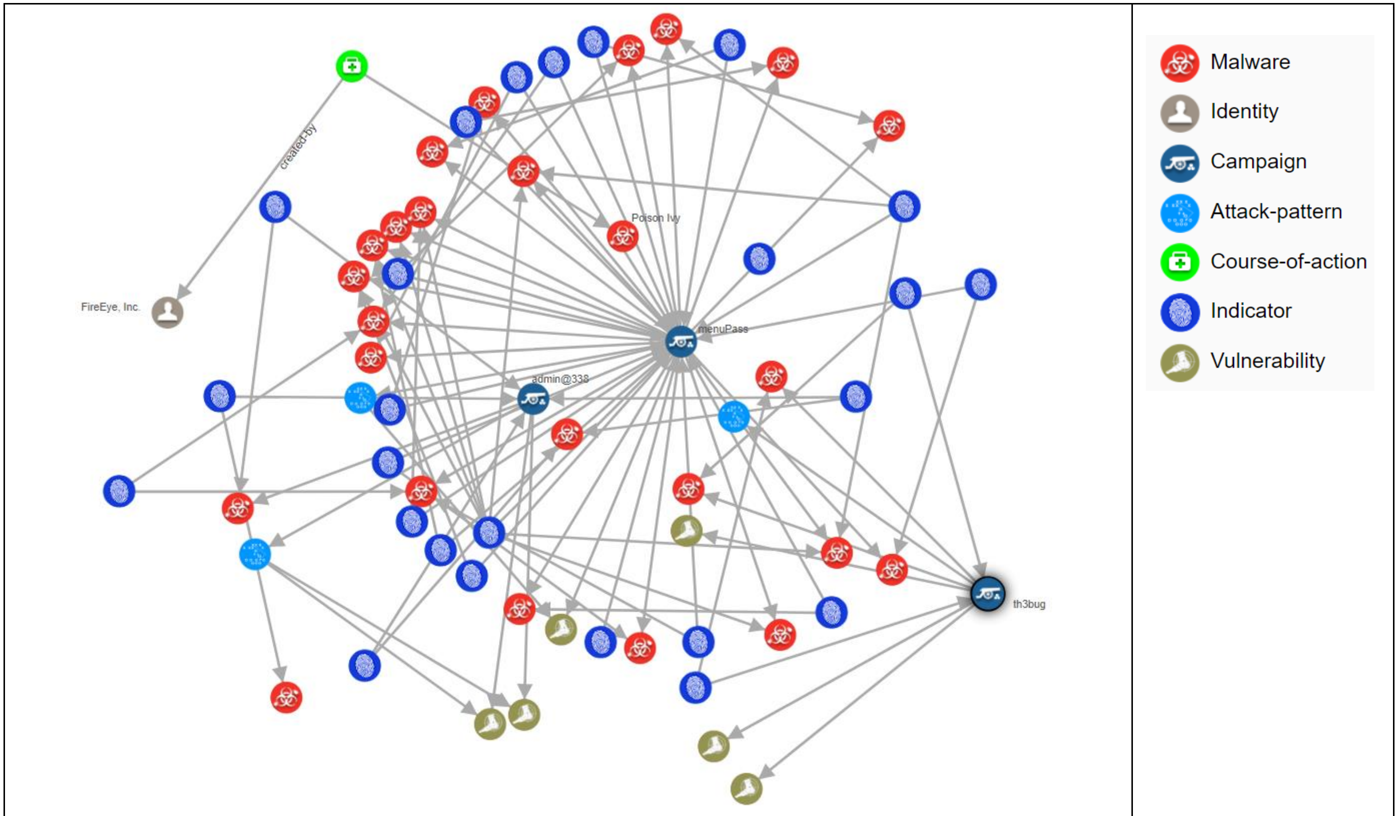


Figure 2. A threat data model based on Fireeye's PIVY report

and algorithms used for evidence analysis and explanation. For example, evidence patterns or new evidence can be discovered by artificial intelligence (AI).

- Most importantly, how the above process and results are validated? Why should courts trust the process and results? For example, upon seizing digital evidence, action should not change that evidence. We propose Preservation and Interpretation objects to validate the authenticity of evidence and explain the. For example, if evidence patterns or new evidence is discovered by artificial intelligence (AI), then explainable AI algorithms may show investigators and courts how much they can trust evidence.

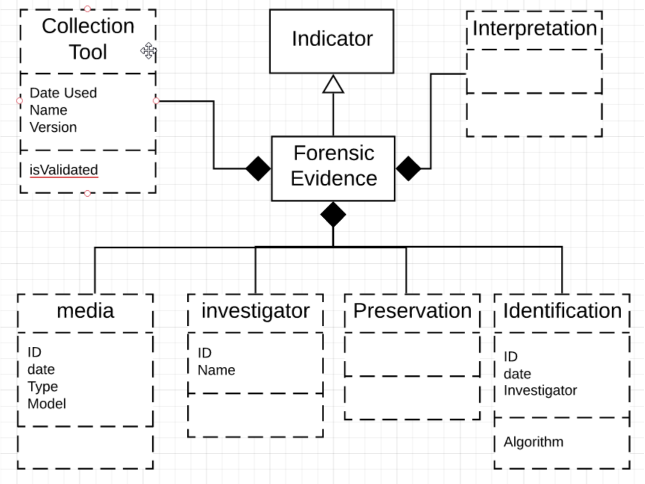


Figure 3. CFE object extends Indicator object in CTI

III. Cyber Forensic Evidence Discovery

We propose a systematic approach to search for possible CFE for an SUI. The main idea of the approach is based on the fact that all CFE is generated by software and hardware of the SUI that was used by cybercriminals. Figure 4 shows a bottom-up layered architecture to discover CFE. Layer one represents any cybercrimes involving computing-related devices. Layer two is called a forensic evidence generator layer. It represents a computing-related device used by cybercriminals and is considered a forensic evidence generator. All system components of the device, such as software and hardware, will either generate CFE or store generated CFE. Layers three and four classify CFE objects in a tree-like structure. In the structure, all CFE is the root of the CFE tree, layer three represent CFE categories, and layer four contains all CFE objects. The first level of layer three has three types of CEF, including application-generated CFE, operating system (OS)-generated CFE, and hardware-generated CFE.

The application-generated CFE has two sub-types, the application function-generated CFE and application non-function-generated CFE. From the software engineering perspective, an application is developed based on two types of requirements, functional and non-functional requirements. A functional requirement is a description of the service that the software must offer. It defines a function of a system or its component, where a function is described as a specification of behavior between outputs and inputs [13]. Non-functional requirements are often referred to as "quality attributes" of a system, including usability requirements, security requirements, reliability requirements, etc. For example, an online chatting application that is built from functional requirements will generate CFE objects related to the chat functions, such as chat text messages, images, audio, and videos. To meet the usability requirements, the online chat application may cache credentials in cookie or memory and the cookie is considered a classic CEF object. The security requirements may save the private key of the application in a folder. The key is a CFE too. These identified CFE objects are in layer four. Similarly, OS-generated CFO also can be classified into two sub-categories, OS-function and non-function generated CFE objects. A typical operating system has three main functions: (1) manage the computer's resources, such as the Central processing unit (CPU), memory, and disk drives, (2) provide a user interface, and (3) execute and provide services for applications software. The non-functional features of an OS can also have usability, security, reliability requirements, etc. Note that layer three can be further refined to form sub-types as needed.

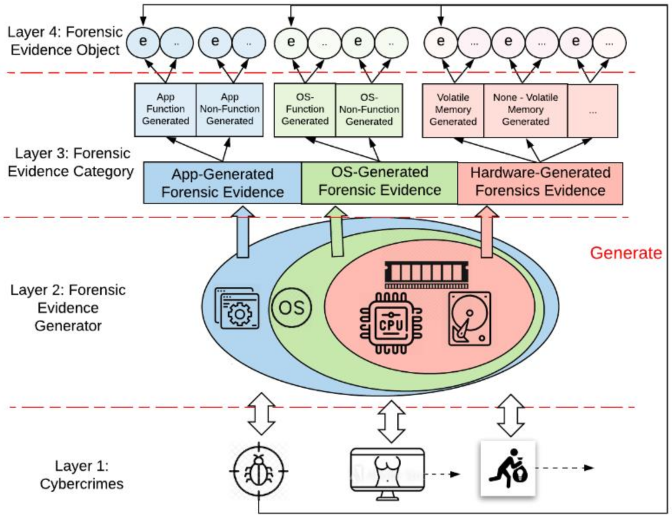

Layer 4: Forensic Evidence Object
e
..
...
App Function Generated
App Non-Function Generated
OS-Function Generated
OS-Non-Function Generated
Volatile Memory Generated
None - Volatile Memory Generated
Layer 3: Forensic Evidence Category
App-Generated Forensic Evidence
OS-Generated Forensic Evidence
Hardware-Generated Forensics Evidence
Generate
Layer 2: Forensic Evidence Generator
OS
CPU
Layer 1: Cybercrimes

Figure 4. A systematical approach to discover digital forensic evidence for a system under the investigations

The hardware-generated CFE is different from the software-generated CFE, including the application- and OS-generated CFE. It contains three sub-types of CEF related to the hardware components of commuting devices, CPU-generated CFE, hard drive-generated CFE, and memory-generated CFE. Memory forensics is a vital form of cyber investigation that allows an investigator to identify unauthorized and even malicious activity on a target computer or server. This is usually achieved by running special software, known as a memory dump, which captures the current state of the system's memory as a snapshot file. The memory-generated CFE presents some special states in memory dumps. There are two types of memory-generated CFE, volatile memory generated CFE and non-volatile memory generated CFE. Volatility Framework [14], a volatile memory extraction utility framework, can extract many CFE objects, such as *clipboard*, *cmdline*, and *iehistory*. The non-volatile memory CFE refers to firmware generated CFE. Firmware is a specific class of computer software that provides low-level control for a device's specific hardware. Typical firmware CFE objects include *Basic Input/Output System (BIOS)*, *hard driver*, and *routers and firewall firmware*. These CFE objects are in level four and the leaves of layer three.

Formally, we can model the relations between cybercrimes and their associated CFE.

- All committed cybercrimes C in layer one is a collection of crime $C= \{c_1, c_2, \ldots, c_n\}$, where each cybercrime $c_i \epsilon C$ and $1 \leq i \leq n.$
- All CFE objects E in layer four is a collection of CFE $E = \{e_1, e_2, \ldots, e_n\}$, where $e_i \epsilon E$ and $1 \leq i \leq n.$
- *CFE* generated from a crime c under the investigations is a set of CFE object $V = g(c) \subset E$ where g is a *CFE* generation function, which represents how the crime c is committed.

IV. An Architecture for detecting APT CyberCrime activities

Figure 5 shows the architecture of detecting APT cybercrimes. The architecture has three main components shown inside of the dashed rectangle: a CFE model repository, a FED-FSM model repository, and a TADE. The system takes shared APT CFI, SUIs, and observable CFE objects from the SUI as inputs to infer possible malicious activities.

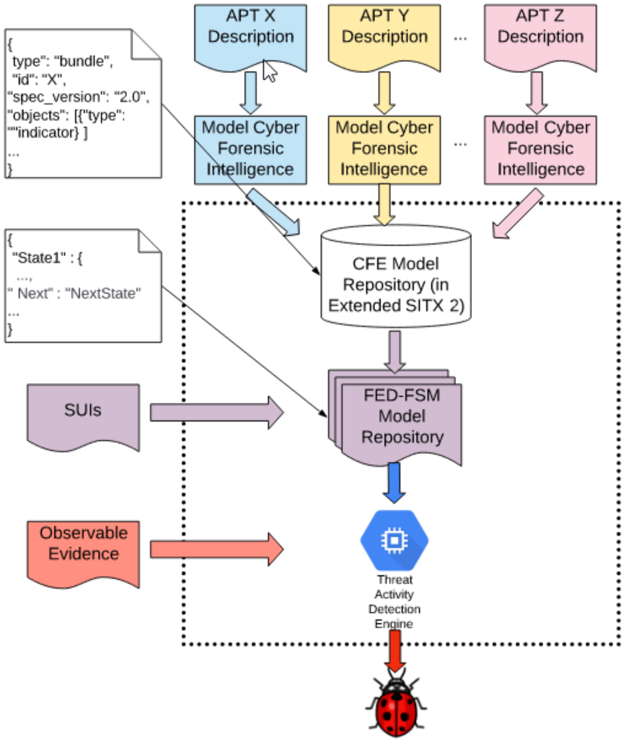


Figure 5. The architecture for detecting APT Crime activities with FED-FSM

A. A CFE Model Repository

A CFE model repository is a collection of sharable CFI from APTs expressed in JSON. Each CFE model consists of a list of CFE objects. Formally, let define c = PIVY and g is the function that generates all the CEF objects, i.e., $V = g(c) =$ *{PIVY file, IP address, Running process,…}*. PIVY file is a CFE object, which refers to the existence of the malware file itself (i.e., the static executable file). The IP address refers to the IP address of attackers, i.e., the client of PIVY is a CFE object because attackers need to communicate with the PIVY server. Also, the running process is another CFE object that indicates PIVY is running on victims' devices.

The following JSON file defines the PIVY file CFE object. The CFE object defines CFE attributes that are associated with a threat activity. These attributes include the CFE type, id, name, pattern, etc. Patterns are designed to assert suspicious or malicious cyber activities. Specifically, patterns use observable objects and their attributes to describe forensic evidence that is associated with known malicious activities. For example, the pattern `[file: name = 'Poison_Ivy_2.3.2.exe']` is to assert the existence of the PIVY file. Logic operations can be applied to multiple observable objects as well. For example, in addition to asserting the existence of the specific file, the following pattern can check the identity of the file (e.g., an SHA-256 hash) with the logical operator `AND`, `[file: name = 'Poison_Ivy_2.3.2.exe' AND file: hashes='SHA-256' = '…e9f5']`. Note that STIX defines a cyber-observable object dictionary. Indicators containing cyber observable objects can be collected from threat data model repositories available publicly or created to support the flexibility of the framework.

```
# Poison Ivy file CFE object in JSON
{
"type": "CFE-file obect",
"id": "PIVY-CFE-file-1",
"created": "2014-02-20T09:16:08.989Z",
"modified": "2014-02-20T09:16:08.989Z",
"name": "PIVY 2.3.2",
"description": "Assert the exists of
                PIVY process.",
"labels": "malicious-activity",
"pattern":"[file:name=
                'Poison_Ivy_2.3.2.exe']"
}
```

Figure 6. Code Snippet of a CFE object in PIVY

B. A FED-FSM Model Repository

A FED-FSM model repository contains a collection of FED-FSM models. The main design idea of the repository is that (1) FED-FSM models in the repository describe potential APT activities imposed on SUIs and (2) any CFI we have observed in SUIs to infer and monitor the potential APT activities in terms of FED-FSM models can be used.

Traditional FSM is a well-studied mathematical model of computation, and these mathematical models are suitable for process automation. Unlike static threat data models for CTI sharing, which only describe static threat information, FSM is commonly used for capturing dynamical behaviors of synchronous sequential machines or software systems, and it has been utilized for detecting security vulnerabilities [15][16][17][18]. A state in FSM models is a description of the status of a system. An FSM model often contains a list of its states and one initial state. A state of an FSM model can change from one state to another in response to some activities or external events. Such a change is called a transition.

However, traditional FSM models cannot be used for SUI threat detection directly because their transitions, such as attacking or threat activities, are unknown or unpredictable for threat analysis. Without knowing these threat activities, it will be very challenging for analysts to monitor and understand the status of SUI and to detect threat activities. To address the issue, we extend FSM by only allowing FSM states to change in response to forensic evidence left by APTs and patterned CFE objects. These CFE objects are from the CFE model repository. The status of a FED-FSM model is inferred by CFE objects instead of triggered by unknown threat activities. The formal definition of FED-FSM is discussed in the next section for process automation.

C. *A Threat Activity Detection Engine*

TADE shown in Figure 7 is another key component of the APT detection system. TADE consists of three different data types and threat activity detecting algorithms. These data types include CFE, cyber observed data, and sightings. The idea of the TADE is to use algorithms to infer the existence of malicious based on observed forensic evidence CFE extracted from an SUI and shared CFE fed on other cybersecurity intelligence resources, such as Anomali [19].

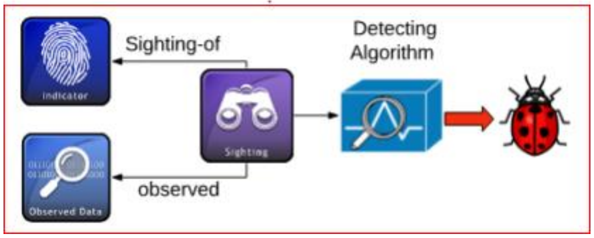


Figure 7. The APT activity detection engine (TADE) for detecting threat activities using CFE (e.g., Indicator), Cyber observed data, and Sightings

The detection engine has defined the following major functions:

- Collecting evidence logs. The log files include APT activity forensic evidence collected from files, disks, networking, and host system events as well as processes and signature strings in memory (Figure 4).
- Preprocessing logs. Logs will be cleaned, checked, organized, integrated, and stored in an evidence data repository or in memory for better performance.
- Formalizing observed evidence data. Similarly, the observed evidence data collected from an SUI will be specified in supporting STIX Domain Objects (SDO), named Observed Object or Observed CFE objects. Note that while CFE objects represent intelligence assertions behind attacks, raw observed information helps formulate the basis behind this intelligence, the observed CFE objects convey information that was observed on systems and networks. Multiple observed CFE objects can be used for crosschecking evidence and therefore increase the confidence of inferring results. The following code snippet shows an observed CFE file object, including its name, size, hash codes, etc.

```
# An observed file object in PIVY
{
“type": "observed CEF Object",
"id": "observed-data--1",
…
"objects": {
   "0": {
      "type": "file",
      "hashes": {
        "MD5": "CF7AB60B7948232C4
                47F284FC695A868",
        "SHA-256": "6cd85b478066
                     …
                    479d8f9f198be9f5"
       },
       "name":   "Poison_Ivy_2.3.2
                             .exe",
       "size": 54824
       }
}
```

Figure 8. Code Snippet of an observed file object in PIVY

- Determining threats. To detect a threat activity, we first use the STIX Relationship Object (SRO), i.e., *Sighting* object, to report observations of both CFE objects and observed CFE objects. *Sighting* objects use two references to capture: what indicator was sighted (i.e., `sighting_of_ref)` and what was seen on an SUI, (i.e., `observed_data_ref)`. SROs are also specified in JSON to facilitate the threat detection automation process. Based on information collected by *Sighting* objects, various detecting algorithms can be used for determining whether threat activities exist by using patterns in CFE objects against observed data attributes.

V. Forensic-Evidence-Driven FSM Model

A FED-FSM model extends an FSM model by integrating a CFE model into the FSM model. Specifically, transitions of an FSM are determined by threat indicators specified by CFE objects. Formally, a FED-FSM model is defined as a tuple $<S, T, F, I, L, \varphi, s_0>$, where the elements of the tuple are defined as follows:

1) *S* is a set of states of an SUI.
2) *T* is a set of transitions of an SUI.
3) *F* is a finite set of arcs from one transition to another, i.e., $F \subseteq S \times S$.
4) *I* is a set of threat indicators specified by CFE objects.
5) *L* is a threat indicators-selecting function on *T and I*, i.e., $L(t, I) \subseteq I$ and $t \in T$.
6) φ is a guard function on *T* and *L*. The guard condition of transition *t*, $\varphi(t, L)$, is a first-order logical formula, which can be evaluated as true or false. The element of the formula is a list of STIX patterns that represent CFE objects.
7) s_0 is an initial state. It is often defined as *Secure*, i.e., s_0 = *Secure* and $s_0 \in T$.

Figure 9 shows two states (i.e., *Secure* and *Penetrated*) and one transition of the PIVY FED-FSM model based on *Fireeye's PIVY* report [10]. The state *Secure* is an initial state and it indicates a system has not been compromised. *Penetrated* state indicates malicious code that has been successfully executed on an SUI by an attacker. The tuple t = *(Secure, Penetrated)* is a transition. The threat indicator-selecting function $L(t, I)$ selects a CFE object with a process ID, e.g., *observed-data—2*, from *I*, where *I* represents all available CFE objects in the CFE model repository. The guard function $\varphi(t, L)$ on the transition *t* defines the pattern formally, i.e.,

```
[ pattern":"[process:name=
                   myPoisonIvy_HK']" ]
```

TADE will use the pattern to evaluate observed CEF objects collected from an SUI and return true if the pattern matches observed CEF objects or false if it doesn't. The Boolean value determines whether the current state will change from *Secure* to *Penetrated*.

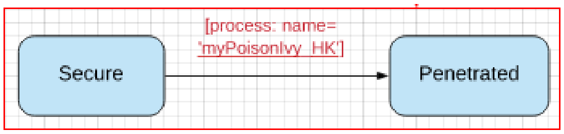


Figure 9. Two states and one transition from Poison Ivy FED-FSM model

FED-FSM models can also be expressed in Amazon States Language (ASL) in JSON format [20]. The following code snippet shows two states, `Secure` and `Penetrated`, as well as the lambda pseudo function (called `resource`), `DetectProcess`, for determining whether a threat exists in an SUI.

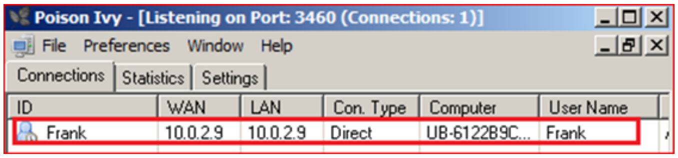
```
{
 "Comment": "A partial code snippet of
             Poison Ivey FED-FSM model
             in ASL",
 "StartAt": "Secure",
    "States": {
      "Secure": {
        "Type": "Task",
        "Resource": "DetectProcess",
        "Next": "Pentrated”
     },
      “Penetrated”: {…}
   }
};

# DetectProcess lambda pseudo
# function as a transition
exports.handler =
   function(event, context) {
       context.succeed(
            indicator.match(
                observedData));
};
```

VI. Empirical Study

The empirical study demonstrates the use of the framework to detect an APT that utilizes a customized PIVY against an SUI. We describe the following three artifacts related to the case study.

A. Case Study Environment Setting

The case study is conducted in a VirtualBox with two Windows virtual machines (VMs) and one security onion (https://securityonionsolutions.com/) Linux VM. One Windows VM acts as a PIVY client and another one acts as a PIVY server. The security onion is to monitor network traffics among three VMs. The client that is controlled by an attacker is configured on the attacker's machine. It will accept the server's connection and act as a command and control center of the server. The server or payload is created by the attacker using Poison Ivy 2.3.2 and then distributed to one victim's machine. Once the victim executes the payload, the payload will infect its machine and connect to the computer running the PIVY Client. Malicious activities, including internal reconnaissance and data exfiltration [21], will be carried out after the victim's machine has been infected. The environment setting instructions for the empirical study can be accessed at GitHub [22]. Figure 10 shows the PIVY client and a victim's machine that has been infected by a PIVY server. The PIVY client is listening on its port 3460.

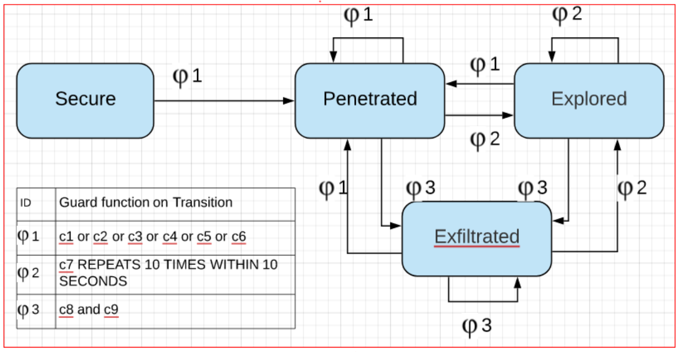

Poison Ivy - [Listening on Port: 3460 (Connections: 1)]

ID	WAN	LAN	Con. Type	Computer	User Name
Frank	10.0.2.9	10.0.2.9	Direct	UB-6122B9C...	Frank

Figure 10. A PIVY client is listening on its port 3460

B. PIVY FED-FSM Model

Figure 11 shows the PIVY FED-FSM model of an SUI. Besides the two aforementioned states, *Secure* and *Penetrated*, the FED-FSM model has two states: *Explored* and *Exfiltrated*. The state *Explored* indicates an SUI has been explored by attackers to gain a better understanding of the environment for future actions. The state *Exfiltrated* indicates the SUI has an unauthorized movement of data.



ID	Guard function on Transition
φ1	c1 or c2 or c3 or c4 or c5 or c6
φ2	c7 REPEATS 10 TIMES WITHIN 10 SECONDS
φ3	c8 and c9

Figure 11. A Poison Ivey FED-FSM model of an SUI

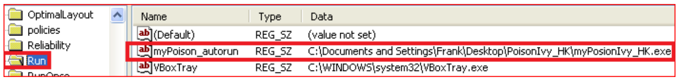
The model has three types of guard functions on ten transitions. Each guard function contains multiple comparison expressions. For example, the guard function $\varphi 3$ indicates that at any state except *Secure*, the destination state will be *Exfiltrated* if both comparison expressions *c8* and *c9* are evaluated as true.

Table 1 lists ten representative comparison expressions used in the model for matching possible threat evidence collected from logs. The types of evidence include file, directory, process, Windows registry, IP address, and network traffic.

Table 1. Ten comparison expressions used in the FED-FSM model

ID	Comparison Expressions	Matches
c1	file: name = 'myPoisonIvy_HK.exe'	the name of PIVY Sever
c2	file: name MATCHES 'BG.bat.lnk'	a created file
c3	Directory: path LIKE 'c:\\Windows\\%\\dfed'	a created folder
c4	Process: name = 'myPoisonIvy_HK'	a running process
c5	win-registry-key: key = '^HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run\\myPoisonIvy_autorun'	a created registry key for auto run
c6	ipv4-addr: value='10.0.2.12'	an IP address
c7	ipv4-addr: value='10.0.2.12/24'	a subset of IP address
c8	network-traffic:dst_ref.value = '10.0.2.12'	network traffic to IP '10.0.2.12'
c9	network-traffic:dst_port = '3460'	network traffic to port '3460'

Note that (1) the nine observable objects are a subset of 102 objects described on the PIVY report [10] and Trend Micro [23]. (2) The empirical study uses lightweight command-line tools (CLT) to collect observable data. These tools include netstat, Windows Management Interface Command (wmic), PowerShell, Logparser, and Sysinternals Utilities. For demonstration, Figure 12 shows the observable evidence detected on the Windows registry, which indicates PIVY has created a registry key for autorun, which maintains the persistence of threats.



OptimalLayout
policies
Reliability
Run
RunOnce

Name	Type	Data
(Default)	REG_SZ	(value not set)
myPoison_autorun	REG_SZ	C:\Documents and Settings\Frank\Desktop\PoisonIvy_HK\myPosionIvy_HK.exe
VBoxTray	REG_SZ	C:\WINDOWS\system32\VBoxTray.exe

Figure 12. Observable evidence showing on Windows registry

C. Framework Deployment Diagram

The deployment diagram for detecting APTs is shown in Figure 13, which describes the system components after implementation. To facilitate the discus, we have added two teams in the diagram. A red team (red icon on the figure) is an independent group that challenges an organization to improve its effectiveness by assuming an adversarial role or point of view. The red team will: (1) Simulate APTs. Set up a controlled environment, e.g., using virtual machines, to simulate attacking scenarios, e.g., APT uses Poison Ivy. (2) Set up a Trusted Automated Exchange of Intelligence Information (TAXII) server [9]. It stores PIVY data models in a local repository [24]. (3) Serialize and de-serialize STIX JSON content using a TAXII client and Python APIs [25]. A blue team (black icon on the figure) is a group of individuals who perform an analysis of information systems to ensure the security of SUIs. Specifically, the blue team sets up a FED-FSM server to host FED-FSM execution frameworks, an observable data server to collect logs from an SUI and a TAXII/FED-FSM client [26] that executes the TADE and visualizes FED-FSM models. The empirical study adopts a Python framework for developing and running FSM-based workflows on AWS Lambda [27]. The framework provides a means to check a state machine's logic and monitor executions.

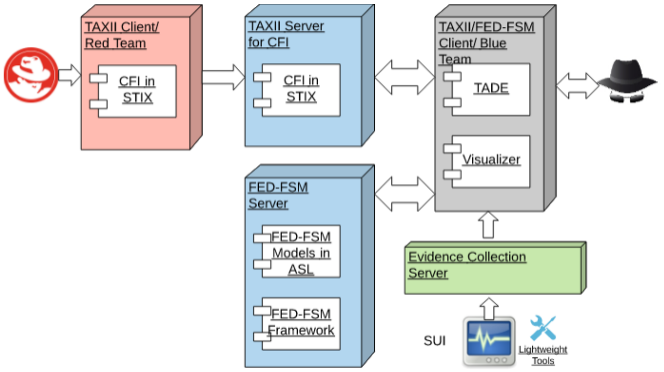


Figure 13. The deployment diagram for detecting APTs

VII. RELATED WORK

There have been many attempts to develop frameworks to systematically detect APTs. Bhatt et al. presented a framework [28] that models multi-stage attacks in a way that both describes the attack methods as well as the anticipated effects of attacks. The foundation to model behaviors is by the combination of the Intrusion Kill-Chain attack model and defense patterns. Haq et al. [29] described a computerized method in which one or more received objects are analyzed by an APT detection center to determine if the objects are APTs. The analysis may include the extraction of features describing and characterizing features of the received objects. The extracted features may be compared with features of known APT malware objects and known non-APT malware objects to determine a classification or probability of the received objects being APT malware. Wan et al. proposed a network gene-based framework [30] to describe the semantic-rich network behavior patterns of network applications. It took advantage of the latest advances in the fields of protocol reverse analysis, cloud computing, and big data processing, with automatic analysis and extraction of network genes, and data stream computing-based network gene real-time processing. Vert et al. [31] applied an advanced state machine engine to the analysis of state variables that can detect the presence of APTs and other malware. The Finite Angular State Velocity Machine (FAST-VM) can model and analyze large amounts of state information over a temporal space. The approach can analyze and model large amounts of data over time. Friedberg et al. applied a kind of black-list approach and only considered actions and behavior that match well-known attack patterns and signatures of malware traces [32]. They proposed an anomaly detection technique that keeps track of system events, their dependencies, and occurrences, and thus, the technique can learn the normal system behavior over time and report all actions that differ from the created system model.

None of the aforementioned APT detection frameworks are practical since they lack the essential characterizations of a framework for automation, including the scalability of architecture [28][31][32], the formalization attacks of APT features [28][29][31], and the diversity of observable objects of SUI [30][32].

VIII. Conclusion

The paper presents a new formal approach that uses FED-FSM to detect APTs. The FED-FSM models are driven by real-world knowledge of adversary tactics and techniques stored in a shared repository. Instead of monitoring APTs directly, the approach infers the APT's status by analyzing the forensic evidence that malicious actors left on digital devices. The approach requires us systematically collecting crime activity logs, extracting evidence from logs, and formalizing digital forensic evidence. Two types of digital forensic evidence are defined in the paper, shared CFE objects, and observed CFE objects. These two objects are the drive force of FED-FSM. A demo program that is implemented in Java can be accessed at [33]. Note that the guided conditions of transitions in FED-FSM are predefined in FED-FSM using patterns. In future work, we are interested in investigating artificial intelligence-based approaches to discover patterns from shared CFE objects automatically and match patterns with observed CFE objects.

Acknowledgment

The work is supported in part by the National Science Foundation 1714261, 2039289, and the Office of Justice Programs 2019-DF-BX-K00.

References

[1] R. S. Hoefelmeyer and T. E. Phillips, “System and method for malicious code detection,” May 9, 2006, US Patent 7,043,757.

[2] P. Szor and P. Ferrie, “Detecting malicious software through process dump scanning,” Jul. 28 2009, US Patent 7,568,233.

[3] S. Ji, “Computer network malicious code scanner method and apparatus,” Aug. 2001, US Patent 6,272,641.

[4] A. Tuor, S. Kaplan, B. Hutchinson, N. Nichols, and S. Robinson, “Deep learning for unsupervised insider threat detection in structured cybersecurity data streams,” in *Workshops at the Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[5] C.-H. Hsieh, C.-M. Lai, C.-H. Mao, T.-C. Kao, and K.-C. Lee, “Ad2: Anomaly detection on active directory log data for insider threat monitoring,” in *2015 International Carnahan Conference on Security Technology (ICCST)*. IEEE, 2015, pp. 287–292.

[6] B. E. Strom, J. A. Battaglia, M. S. Kemmerer, W. Kupersanin, D. P. Miller, C. Wampler, S. M. Whitley, and R. D. Wolf, “Finding cyber threats with att&ck-based analytics,” Technical Report MTR170202, MITRE, Tech. Rep., 2017.

[7] B. E. Strom, A. Applebaum, D. P. Miller, K. C. Nickels, A. G. Pennington, and C. B. Thomas, “Mitre att&ck: Design and philosophy,” Tech. Rep., 2018.

[8] “Sharing threat intelligence just got a lot easier,” https://oasis-open.github.io/cti-documentation/, Dec. 2019.

[9] “Oasis cyber threat intelligence technical committee,” https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=cti, Dec. 2019.

[10] N. M. Bennett, James T. and N. Villeneuve, “Poison ivy: Assessing damage and extracting intelligence,” FireEye Threat Research Blog, Tech. Rep., 2013.

[11] D. J. Ryan and G. Shpantzer, “Legal aspects of digital forensics,” in *Proceedings: Forensics Workshop*, 2002.

[12] K. Nance and D. J. Ryan, “Legal aspects of digital forensics: a research agenda,” in *2011 44th Hawaii International Conference on System Sciences*. IEEE, 2011, pp. 1–6.

[13] R. Fulton and R. Vandermolen, *Airborne Electronic Hardware Design Assurance: A Practitioner’s Guide to RTCA/DO-254*. CRC Press, 2017.

[14] V. Foundation, “Volatility framework - volatile memory extraction utility framework,” Web: https://github.com/volatilityfoundation/volatility, May 2020.

[15] K.-T. Cheng and A. S. Krishnakumar, “Automatic functional test generation using the extended finite state machine model,” in *The 30th ACM/IEEE Design Automation Conference*, 1993.

[16] P. C. Hershey, D. B. Johnson, A. V. Le, S. M. Matyas, J. G. Waclawsky, and J. D. Wilkins., “Network security system and method using a parallel finite state machine adaptive active monitor and responder.” U.S. Patent 5,414,833, May 1995.

[17] D. Xu, W. Xu, and M. Tu, “Automated generation of integration test sequences from logical contracts,” in *The 38th International Computer Software and Applications Conference Workshops (COMPSACW)*. Sweden: IEEE, Jul. 2014, pp. 632–637.

[18] D. Xu, W. Xu, M. Kent, L. Thomas, and L. Wang, “An automated test generation technique for software quality assurance,” *IEEE Transactions on Reliability*, vol. 64, no. 1, pp. 247–268, 2015.

[19] Anomali, “Anomali,” Web, Jun. 2020, https://www.anomali.com/resources/what-are-stix-taxii.

[20] Amazon, “Amazon states language,” https://docs.aws.amazon.com/step-functions/latest/dg/concepts-amazon-states-language.html, Dec. 2019.

[21] FireEye, “Red team operations (RTO) - FireEye,” https://www.fireeye.com/content/dam/fireeye-www/services/pdfs/pf/ms/ds-red-team-operations.pdf, Dec. 2019.

[22] “Poison ivy lab,” https://github.com/frankwxu/Ubalt/tree/master/EthicalHacking/Labs/PosionIvy, Jan. 2020.

[23] TrendMicro, “Poisonivy,” https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/poisonivy, Jan. 2020.

[24] “Oasis TC open repository: Taxii 2 client library written in python,” https://github.com/oasis-open/cti-taxii-client, Dec. 2019.

[25] “Oasis TC open repository: Python APIs for STIX 2,” https://github.com/oasis-open/cti-python-stix2, Dec. 2019.

[26] “Oasis TC open repository: Taxii 2 server library written in python,” https://github.com/oasis-open/cti-taxii-server, Dec. 2019.

[27] “A python framework for developing finite-state machine-based workflows on AWS lambda.” https://github.com/Workiva/aws-lambda-fsm-workflows, Dec. 2019.

[28] P. Bhatt, E. T. Yano, and P. M. Gustavsson, “Towards a framework to detect multi-stage advanced persistent threats attacks,” in *8th IEEE International Symposium on Service-Oriented System Engineering, SOSE 2014, Oxford, United Kingdom, April 7-11, 2014*. IEEE Computer Society, 2014, pp. 390–395.

[29] T. Haq, J. Zhai, and V. K. Pidathala, “Advanced persistent threat (apt) detection center,” Apr. 18 2017, US Patent 9,628,507.

[30] Y. Wang, Y. Wang, J. Liu, and Z. Huang, “A network gene-based framework for detecting advanced persistent threats,” in *2014 Ninth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*. IEEE, 2014, pp. 97–102.

[31] G. Vert, A. L. Claesson-Vert, J. Roberts, and E. Bott, “A technology for detection of advanced persistent threat in networks and systems using a finite angular state velocity machine and vector mathematics,” in *Computer and Network Security Essentials*. Springer, 2018, pp. 41–64.

[32] I. Friedberg, F. Skopik, G. Settanni, and R. Fiedler, “Combating advanced persistent threats: From network event correlation to incident detection,” *Computers & Security*, vol. 48, pp. 35–57, 2015.

[33] W. Xu, “A FED-FSM implemented in squirrel framework state machine,” Web, Jun. 2020, https://github.com/frankwxu/Ubalt/tree/master/Research/APT_FSM.