Appendix

I. Attack-Conditions

The provenance queries are constructed from two queries (prerequisite query and the main query). Here, we are consolidating the conditions for these two building blocks queries for different attack behaviors.

Initial Compromise.

Domain Hijaking (T1584-001).
The prerequisite query:

$$N_1 \in \{BrowserProcesses\} \ \wedge \ R_I = fork \wedge \\ N_2 \in \{RemoteAccessProcesses\}$$

The main query:

$$N_3 \in \{N_2\} \ \wedge R_M = connect \wedge \\ N_4.ip \notin \{TrustedIPAddresses\}$$

where *RemoteAccessProcesses* is the list of processes used for remote access (e.g., SSHD).

Exploit Public-Facing Applications (T1190).
The prerequisite query:

$$N_1 \in \{PublicFacingProcesses\} \ \wedge R_I = accept \wedge \\ N_2.ip \notin \{TrustedIPAddresses\}$$

The main query:

$$N_3 \in \{PublicFacingProcesses\} \ \wedge \ R_M = connect \wedge \\ N_4.ip \notin \{TrustedIPAddresses\} \ \wedge N_4 \notin \{N_2\}$$

where N_2 and N_4 represent sockets (IP Address and Port).

Non Standard Port (T1571).
The prerequisite query: None
The main query:

$$N_4.ip \notin \{TrustedIPAddresses\} \wedge \\ Pair(Process \ N_3 \ and \ Port \ N_4.port) \notin \\ \{ServicePortList\} \ \wedge \ R_M = connect$$

Establish Foothold.
The prerequisite query:

$$N_1 \in \{CompromisedProcesses\} \wedge \\ R_I \in \{fork, execute\} \wedge \\ length \leq SelectedLength$$

CompromisedProcesses is a set of all processes tagged as compromised from the Initial Compromise stage.
The main query:

$$N_3 \in \{N_2\} \ \wedge \ R_M \in \{fork, execute\} \wedge \\ N_4 \in \{CommandLineUtilities\}$$

Escalate Privileges.

Super User Privilege.
The prerequisite query:

$$N_1 \in \{CompromisedProcesses\} \wedge \\ R_I \in \{fork, execute\} \wedge \\ length \leq SelectedLength$$

The main query:

$$N_3 \in \{N_2\} \ \wedge \ R_M \in \{ChangePrincipal\} \wedge \\ N_4.uid \in \{SuperUsers\}$$

ChangePrincipal is the set of syscalls that change the owner user. Examples of those syscalls are `chown, fchown,` and `lchown` syscalls. $N_4.uid$ is the real user ID of the affected process (N_4).

Super User Utilities.
The prerequisite query:

$$N_1 \in \{CompromisedProcesses\} \wedge \\ R_I \in \{fork, execute\} \wedge \\ length \leq SelectedLength$$

The main query:

$$N_3 \in \{N_2\} \ \wedge \ R_M \in \{fork, execute\} \wedge \\ N_4 \in \{SuperUserUtilities\}$$

Scheduled Tasks.
The prerequisite query:

$$N_1 \in \{cron.d\} \wedge \\ R_I \in \{fork, execute\} \wedge \\ length \leq SelectedLength$$

The main query:

$$N_3 \in \{N_2\} \ \wedge \ R_M \in \{chown\} \wedge \\ N_4.uid \in \{SuperUsers\}$$

Credential Dump.
The prerequisite query:

$$R_I \in \{fork, execute\} \ \wedge \ N_1 \in \\ (\{CompromisedProcesses\} \ \vee \ \{SuperUserPrivilege\}) \\ \wedge \ length \leq SelectedLength$$

SuperUserPrivilege is a set of all processes tagged as compromised from *Super User Privilege* technique in Escalate Privileges stage.
The main query:

$$N_3 \in \{N_2\} \ \wedge \ N_4.path \ contains \ \text{“}procdump\text{”} \wedge \\ (N_3.euid \in \{SuperUsers\} \ \vee N_3.uid \in \{SuperUsers\}) \wedge \\ R_M = execute$$

$N_3.uid$ and $N_3.euid$ are the real and effective user IDs for the process N_3 respectively.

Valid Domain Accounts.

The prerequisite query:

$$N_1.uid \notin \{DomainUsers\} \wedge$$
$$N_2 \in \{ScriptingProcesses\} \wedge$$
$$R_I \in \{fork, execute\} \wedge$$
$$length \leq SelectedLength$$

The main query:

$$N_3 \in \{N_2\} \ \wedge \ R_M = connect \wedge$$
$$N_3.uid \in \{DomainUsers\} \wedge$$
$$N_4 \in \{InternalIPAddresses\} \wedge$$
$$N_4.ip \in \{DomainIPAddresses\}$$

DomainUsers is the list of users who are authorized to access the domain controller. *ScriptingProcesses* examples include Python and Powershell. *DomainIPAddresses* are IP addresses of the domain controllers. *uid* is the user id for the corresponding process.

Internal Reconnaissance.

Sensitive Access.

The prerequisite query:

$$N_1 \in \{CompromisedProcesses\} \wedge$$
$$R_I \in \{fork, execute\} \wedge$$
$$length \leq SelectedLength$$

The prerequisite query:

$$N_3 \in \{N_2\} \ \wedge \ R_M \in \{open, read\} \ \wedge \ N_4 \in$$
$$(\{SensitivePaths\} \vee \{SystemCriticalPaths\})$$

Recon Command.

The prerequisite query:

$$N_1 \in \{CompromisedProcesses\} \wedge$$
$$R_I \in \{fork, execute\} \wedge$$
$$length \leq SelectedLength$$

The main query:

$$N_3 \in \{N_2\} \ \wedge \ R_M = execute \wedge$$
$$N_4 \in \{SensitiveCommands\}$$

Port Scan.

The prerequisite query:

$$N_1 \in \{CompromisedProcesses\} \wedge$$
$$R_I \in \{fork, execute\} \wedge$$
$$length \leq SelectedLength$$

The main query:

$$N_3 \in \{N_2\} \ \wedge \ R_M = send \wedge$$
$$N_4 \in \{InternalIPAddresses\} \wedge$$
$$N_4.port \in \{WellKnownPorts\} \wedge$$
$$count(N_4.port) \geq PortCountThres$$

Based on the enterprise settings, the analyst selects the number of ports ($PortCountThres$) at which the provenance query should generate an alert.

Lateral Movement.

The prerequisite query:

$$(N_1 \in \{CompromisedProcesses\} \vee$$
$$N_1 \in \{InternalReconProcesses\}) \wedge$$
$$R_I \in \{fork, execute\} \wedge$$
$$length \leq SelectedLength$$

The main query:

$$N_3 \in \{N_2\} \ \wedge \ R_M = connect \wedge$$
$$N_4 \in \{InternalIPAddresses\}$$

InternalReconProcesses is a set of all processes engaged in Internal Reconnaissance activities.

Complete Mission.

Exfiltration Over C2 Channel.

The prerequisite query:

$$(N_1 \in \{CompromisedProcesses\} \vee$$
$$N_1 \in \{InternalReconProcesses\}) \wedge$$
$$R_I \in \{fork, execute\} \wedge$$
$$length \leq SelectedLength$$

The main query:

$$N_3 \in \{N_2\} \wedge N_4 \notin \{TrustedIPAddresses\} \wedge$$
$$R_M = send$$

`send` is the family of syscalls that includes syscalls used to send data over the network including sendmsg, sendto, sendfile, etc.

Exfiltration by Bypassing Defense Controls.

The prerequisite query:

$$N_1 \in \{InternalReconProcesses\} \wedge$$
$$\in \{EscalatePrivilegeProcesses\} \wedge$$
$$R_I \in \{fork, execute\}$$
$$\wedge \ length \leq SelectedLength$$

The main query:

$$N_3 \in \{N_2\} \ \wedge N_3.uid \in \{SuperUsers\} \wedge$$
$$N_4 \notin \{TrustedIPAddresses\} \ \wedge R_M = send$$

EscalatePrivilegeProcesses is a set of all processes with super user privileges detected in Escalate Privileges stage.

Destroy System.

The prerequisite query:

$$(N_1 \in \{CompromisedProcesses\} \vee$$
$$N_1 \in \{InternalReconProcesses\}) \wedge$$
$$R_I \in \{fork, execute\} \wedge$$
$$length \leq SelectedLength$$

The main query:

$$N_3 \ \in \{N_2\} \wedge$$
$$(N_4 \ \in \{SensitivePaths\} \vee$$
$$\in \{SystemFiles\}) \wedge$$
$$R_M \ \in \{write, unlink\}$$

SystemFiles can be generic per the operating system and can be based on the CTI report. *SensitivePaths* is customised per every enterprise. Here, enterprises define paths to sensitive files and directories. This includes user drives, internal, confidential, and secret shares. Any suspicious operation on those files will be flagged and an alert will be generated.

Cleanup Tracks.

File Deletion.

The prerequisite query:

$$N_1 \in (\{CompromisedProcesses\} \vee \in \{EscalatePrivilegeProcesses\}) \wedge R_I \in \{fork, execute\} \wedge length \leq SelectedLength$$

The main query:

$$N_3 \in \{N_2\} \wedge N_4 \notin \{LogFilesPaths\} \wedge R_M = unlink$$

Remove Log files.

The prerequisite query:

$$N_1 \in (\{CompromisedProcesses\} \vee \in \{EscalatePrivilegeProcesses\}) \wedge R_I \in \{fork, execute\} \wedge length \leq SelectedLength$$

The main query:

$$N_3 \in \{N_2\} \wedge N_4 \in \{LogFilesPaths\} \wedge R_M = unlink$$

Clear Log commands.

The prerequisite query:

$$N_1 \in (\{CompromisedProcesses\} \vee \in \{EscalatePrivilegeProcesses\}) \wedge R_I \in \{fork, execute\} \wedge length \leq SelectedLength$$

The main query:

$$N_3 \in \{N_2\} \wedge R_M = write \wedge N_4 \in \{LogFilesPaths\}$$

II. Normalizing Logs to Canonical Form

The audit log contains an immense number of raw low level system events that do not readily reveal the causal relationship between system entities. Hence, it cannot be coupled with the high level artifacts in CTI reports for the reasons discussed below. We also discuss our workarounds for generating the normalized event log form.

A. Asymmetric System Call Arguments

Every system call (syscall) has its kind of arguments, which are not symmetrically structured across the audit log. Listing 1 shows examples of system call records from the event log for Ubuntu 14.04 (64 bit). For instance, listing 1(a) shows an event log record for `execve` syscall ($syscall = 59$) with the syscall arguments are distributed in `a0` (/bin/bash), `a1` (/usr/share/gh0st) and `a2` fields. On the other hand, it is clear that `fork` syscall ($syscall = 56$) allocates its main argument, the process ID (PID) of the new forked sub-process, in the `exit` field (see Listing 1(b)). The `chown` syscall family attributes the owner (user) id and group id in fields `a2` and `a3`, respectively. We can see in Listing 1(g) that the owner id and group id are both 0, which refers that the owner is now root. We need to fix this to represent system entities and the relationship between them in symmetric structure to map them to the high level descriptions in the CTI reports.

B. System Entities Unique Identifiers

We also observe that no explicit unique identifier is provided for the system entities in the event log record. Even if the PID can uniquely identify a running process at the runtime, the same PID can be later reused by another process after the earlier process terminates. Hence, the PID or the PPID cannot be used to uniquly identify a process and its parent across the system runtime.

In addition, for particular syscalls, such as `execve`, the executed process inherits the PID of the origin (source) process. So technically, the origin process is not clearly identified in the `execve` syscall event. We run a backward tracing [8] on all previously run processes to pinpoint the recent process with the same PID before the new process is started.

C. System Call Dependencies

For specific syscalls, event records are dependent on previous calls (e.g., `read` and `write` depend on `open`). As in Listings 1(d) and 1(e), the process `scp` reads and writes an object which is not included in the corresponding log record. A backward scan for the event logs shows that the `open` syscall record (see Listing 1(f)) contains the object to be read or written to, with its identifier (inode). We complement `read` and `write` syscall records with that missing information in our normalized records.

D. Data Format/Encoding

The audit log presents syscall arguments in different formats as evident in Listing 1, including: decimal values for `pid, ppid` in all syscalls, hex values for `a1, a2, a3` in `chmod` (see Listing 1(h)), and string values for `exe`. Besides, the same attribute can be in different format based on the syscall; for example `a2, a3` are represented in decimal values for the `chown` syscall family, but in hex for `chmod`. Once the data format representation is determined, it is important to understand the encoding used with every attribute before being able to reveal its value. As an example, `a2, a3` for the `chmod` syscall family are represented in hex and need to be converted into octal format to reveal the permissions granted for the object. Other attributes in the event logs contain encoded packed values. For example, in `connect` (see Listing 1(c)) and `accept` syscalls, the `saddr` attribute which includes the socket information (IP Address and Port) is encoded in hex packed format which needs to be processed to unpack it and retrieve the corresponding IP Address and Port.

To overcome the above challenges, we provide a canonical representation of the audit log with the following features:

- Compact format: we combine records that have the same syscall ID (indicating that they belong to the same operation), in one record in the compact form. For example, `execve` spwans at least the following record types in the audit log: `SYSCALL`, `EXECVE`, `CMD`, `PATH`, and `PROCTITLE`. We correlate all these events and include all the important information in one compact record.
- Unique identifier representation: we create a universal unique identifier (UUID) for every entity (e.g., processes,

Listing 1 The audit log - system call record examples.

(a) Execve Syscall

```
type=SYSCALL msg=audit(1618071287.244:687514):syscall
=59 success=yes exit=0 a0=1579088 a1=1532a88
a2=163c008 a3=598 ppid=2077 pid=2083 exe="/bin/bash"
type=EXECVE msg=audit(1618071287.244:687514):
a0="/bin/bash" a1="/usr/share/gh0st" a2="start"
type=PATH msg=audit(1618071287.244:687514): item=0
name="/usr/share/gh0st" inode=792704
```

(b) Fork Syscall

```
type=SYSCALL msg=audit(1618072037.116:827848):syscall
=56 success=yes exit=7177 a0=1200011 a1=0 a2=0
a3=7fbec78bd9d0 ppid=7162 pid=7176 exe="/usr/bin/scp"
```

(c) Connect Syscall

```
type=SYSCALL msg=audit(1618071198.080:665224):
syscall=42 success=yes exit=0 a0=aa
a1=7fe95d5b17b0 a2=10 a3=0 items=0 ppid=1407
pid=3461 exe="/usr/lib/firefox/firefox"
type=SOCKADDR msg=audit(1618071198.080:665224):
saddr=020000000DE1BD3D0000000000000000
```

(d) Read Syscall

```
type=SYSCALL msg=audit(1618072049.092:828551):
syscall=0 success=yes exit=221 ppid=7162 pid=7176
exe="/usr/bin/scp"
```

(e) Write Syscall

```
type=SYSCALL msg=audit(1618072049.092:828550):
syscall=1 success=yes exit=80 a0=1 a1=7fff327bd760
a2=50 a3=1 items=0 ppid=7162 pid=7176
exe="/usr/bin/scp"
```

(f) Open Syscall

```
type=SYSCALL msg=audit(1618072049.092:828543):
syscall=2 success=yes exit=3 a1=800 a2=0 a3=8
items=1 ppid=7162 pid=7176 exe="/usr/bin/scp"
type=CWD msg=audit(1618072049.092:828543):
cwd="/home/ubuntu"
type=PATH msg=audit(1618072049.092:828543):
item=0 name="/etc/hosts" inode=2359455
```

(g) Fchownat Syscall

```
type=SYSCALL msg=audit(1617660032.907:154792795):
syscall=260 success=yes exit=0 a0=ffffff9c
a1=1da5cb0 ppid=114171 pid=114172 exe="/bin/chown"
type=PATH msg=audit(1617660032.907:154792795):
name="priv_key.txt" inode=2495043
```

(h) Fchmodat Syscall

```
type=SYSCALL msg=audit(1617660053.219:154794519):
syscall=268 success=yes exit=0 a0=ffffffffffffff9c
a1=11570f0 a2=1ff a3=3c0 ppid=114176 pid=114177
exe="/bin/chmod"
type=CWD msg=audit(1617660053.219:154794519):
cwd="/home/ubuntu"
type=PATH msg=audit(1617660053.219:154794519):
name="gh0st.sh" inode=2495042
```

files, and sockets) in the system to uniquely identify the entity across the runtime. This requires analysis of different syscalls and their arguments and attributes.

- Symmetrical structure: we symmetrically structure every record in the new canonical form across different syscalls. The new record, in our proposal, contains 9 fields which capture the needed information for preserving causality among system entities during the system runtime: Timestamp, ID, SubjectUUID, SubjectProcess, Action, ObjectUUID, ObjectName, ActionDetails, and Hostname.

Our devised canonical log representation provides all the important information needed in attack forensics and real-time attack artifacts detection. For further reducing the log size while preserving the causality relationship between system entities, we applied a causality preserved reduction technique as in [10]. The core idea is to merge excessive events between the same pair of entities. Every type of event between a pair of entities is maintained in a stack. Every time the same type of event is occurred between the same pair of entities, the event is checked if it can be aggregated with the events in the stack. The aggregation is done if the two events (the new occurred event and the event in the stack) have the same backward and forward trackability. Any two events between two nodes are said to have same backward trakability if no other incoming events to the first node has occured between the end time of the two events in question. On the other hand, any two events between two nodes are said to have same forward trakability if no other outgoing events from the second node has occured between the start time of the two events in question. If both conditions match, the two events are said to have the same causal dependency and are then merged together. This helps in reducing the intense bursts of semantically similar events which are produced by system daemons and other several applications [10]. This is the last step of log processing done by the LogCore engine before building the whole system provenance graph. A significant point about our provenance graph is that because it is a highly compact version of the audit log, it requires less memory which facilitates real-time ingestion of events and generation of the graph over a long period of time. On this provenance graph, we apply the generated attack behavior queries to pinpoint attack behaviors as in the CTI reports.

III. Log Consumption Time

Figure 1 shows the log size on disk and the log consumption time taken by APTHunter and S-HOLMES. The columns show the uncompressed log size on disk, while the brown and green lines show the consumption time for S-HOLMES and APTHunter respectively.

IV. DARPA Datasets

Table I consolidates information about different attack scenarios in DARPA datasets used in our evaluation.

V. Predicated Hosts Supported by APTHunter

Figure 2 shows the number of hosts can be supported by APTHunter for different memory sizes. The time taken by the operating system and the other needed processes (e.g., Neo4j engine) are considered.

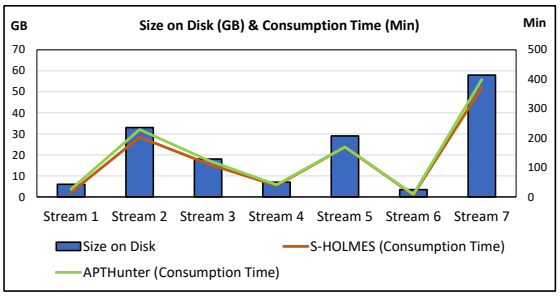

Figure 1. Log size on disk (in GB) and consumption time (from the moment the log is generated all the way to generating the provenance graph) (in Minutes).

Table I
DATASETS. STREAMS 1 TO 5 ARE FROM DARPA ENGAGEMENT 3, STREAMS 6 AND 7 ARE FROM DARPA ENGAGEMENT 5. STREAMS 5 CONTAINS TWO INDEPENDENT ATTACK VECTORS OCCURRING ON THE SAME HOST.

Stream No.	Duration	Platform	Scenario No.	Initial Access Technique	Attack Surface
1	0d1h17m	Ubuntu 14.04 (64bit)	1	Drive-by Download	Firefox 42.0
2	2d5h8m	Ubuntu 12.04 (64bit)	2	Drive-by Download	Firefox 42.0 / Trojan / RAT
3	1d7h25m	Ubuntu 12.04 (64bit)	3	Drive-by Download	Firefox 42.0 / Trojan / RAT
4	2d5h17m	FreeBSD 11.0 (64bit)	4	Web Shell	Web Shell / Nginx backdoor / RAT
5	8d7h15m	FreeBSD 11.0 (64bit)	5	Web shell	Nginx backdoor / sudo
			6	Public-Facing	Nginx backdoor
6	0d0h36	Ubuntu 14.04 (64bit)	7	Drive-by Download	Firefox 42.0 / sudo /sshd / Process Inject
7	1d22h58m	Ubuntu 12.04 (64bit)	8	Drive-by Download	Firefox 42.0 / sudo /sshd / Process Inject

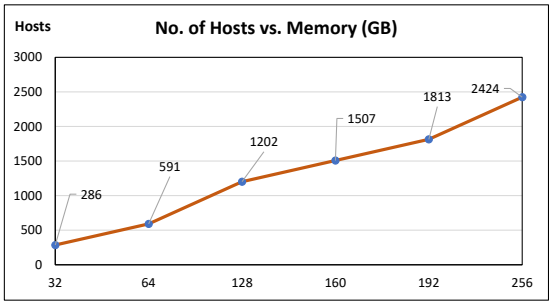

Figure 2. Number of hosts can be supported by APTHunter in one APTHunter's dedicated server versus different server memory sizes (in GB).

VI. S-HOLMES THREAT AND BENIGN SCORES

Table II and Table III show the measurements of S-HOLMES in terms of threat and benign scores calculated as per every attack stage for DARPA attack scenarios and the two public APT attacks respectively. As S-HOLMES follows the original HOLMES design, threat and benign scores do not change when there is no adversarial activities per the corresponding attack stage.

Table II
S-HOLMES EVALUATION ON DARPA ATTACK SCENARIOS. THREAT AND BENIGN SCORES ARE CALCULATED PER EVERY ATTACK STAGE (S1, S2, ..., S7).

Scenario No.	Measurements	S-HOLMES						
		S1	S2	S3	S4	S5	S6	S7
1	Threat Score	13	108	108	1987	1987	55342	1163881
	Benign Score	13	108	108	1328	1328	1328	1328
2	Threat Score	13	108	108	1988	1988	55379	55379
	Benign Score	13	108	108	1331	1331	1331	1331
3	Threat Score	13	108	108	1987	1987	55342	1163881
	Benign Score	13	29	298	298	298	298	298
4	Threat Score	2	5	74	903	903	25153	25153
	Benign Score	7	62	638	638	638	638	638
5	Threat Score	13	29	432	7946	7946	221381	4648997
	Benign Score	7	60	897	16504	16504	16504	16504
6	Threat Score	7	17	247	4530	4530	126199	2653973
	Benign Score	7	60	897	16504	16504	16504	16504
7	Threat Score	13	29	297	5466	96092	96092	96092
	Benign Score	13	108	1110	13650	13650	13650	13650
8	Threat Score	13	108	1110	20414	20414	568743	568743
	Benign Score	13	13	108	1615	19844	19844	19844

Table III
S-HOLMES EVALUATION ON PUBLIC ATTACKS.

APTs	CTI Report	Report Year	Measurements	S-HOLMES						
				S1	S2	S3	S4	S5	S6	S7
APT41	FireEye [5]	2019	Threat Score	2	18	18	226	226	9002	308730
			Benign Score	2	18	18	338	338	338	338
APT35	Darktrace [3], FireEye [7]	2021 2019	Threat Score	7	62	62	1133	1133	45089	948268
			Benign Score	10	85	85	1039	1039	1039	1039

VII. SINGLE PROCESS APPLICATION EXAMPLE

Listing 2 shows event log record example for a single process application (scp) which handles tasks sequentially (i.e., does not spawn new process to handle new tasks).

Listing 2 Single process application example

(a) Scp Process Execve SYSCALL

```
type=SYSCALL TS=1618072037.116 ID=827810
syscall=execve exit=0 ppid=7162 pid=7176
exe="/usr/bin/scp"
type=EXECVE a0="scp" a1="/etc/hosts"
a2="x10@192.168.8.134:~/victim_data/hosts"
```

(b) Close SYSCALL

```
type=SYSCALL TS=1618072037.116 ID=827812
syscall=close ppid=7162 pid=7176
exe="/usr/bin/scp"
type=PROCTITLE TS=1618072037.116 ID=827812
proctitle="bash"
```

VIII. THREAT INTELLIGENCE INFORMATION

Threat intelligence information is available in public and private threat intelligence feeds (e.g., AlienVault [2], Abuse.ch [1], EclecticIQ [4]), as well as from threat intelligence reports by industry (e.g., FireEye [6], Red Canary [9]). The attack artifacts are described in structured and semi-structured formats including, OpenIOC,[1] Structured Threat Information eXpression (STIX),[2] Cyber Observable eXpression (CybOX),[3] YARA,[4] etc. Given the rapid increase in the attack volume and sophistication, the attack artifacts are often descried in unstructured text (as in the CTI reports by industry).

[1]https://github.com/mandiant/OpenIOC_1.1
[2]https://stixproject.github.io/
[3]https://cyboxproject.github.io/
[4]http://virustotal.github.io/yara/

REFERENCES

[1] ABUSE, "Fighting malware and botnets," https://abuse.ch/.

[2] AlienVault, "AlienVault Open Threat Exchange," https://otx.alienvault.com/browse/global.

[3] Darktrace, "APT35 'Charming Kitten' discovered in a pre-infected environment," https://www.darktrace.com/en/blog/apt-35-charming-kitten-discovered-in-a-pre-infected-environment/, 2021.

[4] EclecticIQ, "Intelligence at the core," https://www.eclecticiq.com/.

[5] FireEye, "Special Report: Double Dragon APT41, a dual espionage and cyber crime operation," https://content.fireeye.com/apt-41/rpt-apt41, 2019.

[6] ——, "Threat Intelligence Reports by Industry," https://www.fireeye.com/current-threats/reports-by-industry.html, 2021.

[7] FireEye-Mandiant, "M-Trends 2018 report," https://www.fireeye.com/content/dam/collateral/en/mtrends-2018.pdf, 2018.

[8] N. Michael, J. Mink, J. Liu, S. Gaur, W. U. Hassan, and A. Bates, "On the forensic validity of approximated audit logs," in *Annual Computer Security Applications Conference*, 2020, pp. 189–202.

[9] Red Canary, "Red Canary 2021 Threat Detection Report," https://redcanary.com/threat-detection-report/.

[10] Z. Xu, Z. Wu, Z. Li, K. Jee, J. Rhee, X. Xiao, F. Xu, H. Wang, and G. Jiang, "High fidelity data reduction for big data security dependency analyses," in *CCS*, 2016.