

ProblemChild: Discovering Anomalous Patterns based on Parent-Child Process Relationships

Bobby Filar

Elastic `filar@elastic.co`

\AndDavid French

Elastic `david.french@elastic.co`

Abstract

It is becoming more common that adversary attacks consist of more than a standalone executable or script. Often, evidence of an attack includes conspicuous process heritage that may be ignored by traditional static machine learning models. Advanced attacker techniques, like “living off the land” that appear normal in isolation become more suspicious when observed in a parent-child context. The context derived from parent-child process chains can help identify and group malware families, as well as discover novel attacker techniques. Adversaries chain these techniques to achieve persistence, bypass defenses, and execute actions. Traditional heuristic-based detections often generate noise or disparate events that belong to what constitutes a single attack. ProblemChild is a graph-based framework designed to address these issues. ProblemChild applies a supervised learning classifier to derive a weighted graph used to identify communities of seemingly disparate events into larger attack sequences. ProblemChild applies conditional probability to automatically rank anomalous communities as well as suppress commonly occurring parent-child chains. In combination, this framework can be used by analysts to aid in the crafting or tuning of detectors and reduce false-positives over time. We evaluate ProblemChild against the 2018 MITRE ATT&CK™ emulation of APT3 attack to demonstrate its promise in identifying anomalous parent-child process chains.

Keywords Cyber Attack Detection, Graph Analysis, Machine Learning

1 Introduction

To meet the evolving threat of adversary techniques, Traditional Anti-Virus (AV) leveraged signatures to capture known malicious binaries. Over time this created an arms race between attacker and defender that saw most AVs augment signatures with machine learning in hopes of generalizing previously observed malware against the unknown and Next-Generation Anti-Virus (NGAV) was born. The use of ML to detect and prevent malicious binaries has seen a meteoric rise within the security industry and has become table stakes for endpoint protection platforms. The relative success of this approach at stopping novel attacks with “zero-day” malware infections becoming less commonplace has caused sophisticated adversaries to shift their tactics and techniques.

In June 2017 a new ransomware variant called Petya[1] was first observed in the wild as it hit a variety of organizations worldwide (primarily focused on Ukrainian companies). Petya leverages an SMB vulnerability made popular by the ransomware family WannaCry. Unlike WannaCry, Petya leveraged a series of system-level commands to gain a foothold and establish persistence. These commands provide a “benign” capability to perform credential dumping, execution of itself, setting a scheduled task, and wiping logs. Petya performed all of these tasks using the tools provided by the operating system like Windows Management Instrumentation (WMI) command-line tool *wmic.exe* and task scheduler tool *schtasks.exe*. Tactics like these are called “Living off the land”[2] and represent the next great challenge in securing the endpoint as attackers rely heavily on dual-use software to execute their attacks and “hide in plain sight” on the operating system.

We present ProblemChild a graph-based framework designed to address the concerns listed above. ProblemChild ingests process Event Tracing for Windows data (ETW) and exposes an intuitive interface to address technical challenges in the detection of living off the land techniques, specifically anomalous or rare parent-child relationships. Our four main contributions are as follows:

- Training a machine learning model on a large set of malicious and benign event data targeting features from process creation events.
- Providing an anomaly score that accounts for the prevalence of a parent-child process pair within the local environment against the output of the ML model. This score becomes the edge weight between a parent and child node in a directed graph.
- We apply community detection to segment the weighted graph into smaller “chains” of processes.
- Finally, we apply a threshold against each community to determine if it’s overall score is anomalous and return a ranked list of potentially malicious communities.

We demonstrate that the proposed approach can reduce a large set of process-related events to a manageable list of rank-ordered rare parent-child process chains, suppressing commonly occurring, but previously unobserved activity. Ranking or prioritizing of events has been shown to help reduce noise in an environment[3].

2 Background

2.1 Living off the Land Binaries

Living off the Land Binaries [5] are Microsoft-signed binaries that come pre-installed on the operating system. These pieces of software have alternative or unexpected features outside of their core functionality. For example, the binary `schtasks.exe` allows an administrator to create, delete, query, change, run, and end scheduled tasks on a local or remote computer [4]. However, this binary may also be leveraged by an attacker to bypass User Account Control (UAC) and escalate privileges (e.g., as in Fig. 1). The use of these binaries reduces the number of new files dropped to disk during an attack and lessens the chance of being detected by security software.

```
reg add HKCU\Environment /v windir /d \
    "cmd /K reg delete hkcu\Environment /v windir /f && REM "

schtasks /Run /TN \Microsoft\Windows\DiskCleanup\ SilentCleanup /I
```

Figure 1: Using `schtasks.exe` for disk cleanup UAC bypass is an example of a living-off-the-land technique.

Attackers may also chain these binaries together to perform more sophisticated actions that mimic traditional adversary attack sequences [2].

- **Incursion:** Initial access vector either exploiting a Remote Code Execution (RCE) vulnerability or targeted individuals using a spear-phishing attack.
- **Persistence:** Post-compromise actions to allow the attacker to maintain a presence on the system.
- **Payload:** Employs dual-use tool (e.g. `psexec` or `mimikatz`) or fileless capability to execute rest of attack

The use of these binaries makes the discovery of an attack much more difficult as adversary behavior is mixed in with traditional benign operating system activity. Historically, APT groups and other threat actors were identifiable based on the custom binaries they left behind. By using benign OS binaries attribution becomes that much more difficult. After all, these binaries will not be detected by AV engines and there is no exploit taking place. For security vendors, this means exploring alternative data streams like ETW and attempting to derive anomalous or suspicious event sequences based on file, process, DNS, or security-related events.

| Initial Access | Execution | Persistence |
|----------------------|-------------------|-------------------|
| Privilege Escalation | Defensive Evasion | Credential Access |

| | | |
|-------------------|------------------|------------|
| Discovery | Lateral Movement | Collection |
| Command & Control | Exfiltration | Impact |

Figure 2: MITRE ATT&CK Enterprise Tactics

2.2 MITRE ATT&CK™ Framework

The industry has responded to the living off the land threat by rallying around frameworks like MITRE ATT&CK. The MITRE ATT&CK Framework is a knowledge base that seeks to describe adversary behavior by providing a standardized taxonomy of attack and defense security techniques (e.g. See Fig. 2). At the core of the ATT&CK framework is matrix/ontology that provides a structured visualization of the how and the why of an attack:

- **Tactic:** presentation of an adversary’s objective or reason for taking an action.
- **Technique:** tactical objective of taking an action
- **Group:** Known adversaries that have carried out a specified technique
- **Software:** The binary used to execute a specified technique

Mass adoption of the ATT&CK framework is in large part due to the attention given to parent-child relationships as a detection methodology for living off the land techniques. This methodology is further enhanced by providing conditionals based on metadata associated with the parent-child process chain (e.g. observing a MS Office process spawn powershell.exe with base64 encoded arguments).

2.3 Detection Engineering

The current solution to combat these techniques revolves around developing a rule or heuristic to act as a detector for a given technique. Threat researchers have moved to simple, schema independent query languages to craft real-time detectors against streaming data.[23] Language like EQL allows researchers to craft expressive rule-based logic for detecting a specific ATT&CK technique. Fig. 3 shows an example rule to detect scriptable child processes of Office products and a Spearphishing Attachment (T1193) technique[7].

```
process where
  parent_process_name in ("winword.exe","excel.exe","powerpnt.exe")
  and process_name in ("powershell.exe","cscript.exe", "wscript.exe","cmd.exe")
```

Figure 3: Example EQL rule for (T1193)

This approach is adequate in practice, but does have several limiting factors: Rules can be prone to false positives because they are based on inherently benign software, 2) That noise can contribute to alert deluge common in security software, and 3) The generation of rules is a manual process that requires domain expertise and the ability to learn from or across customer environments to tune detectors over time.

3 Related Work

The use of system-level entities, like parent-child process chains, have historically been used to by security tools to model APT kill chains [3], establish provenance graphs [8] [9], or construct dependency graphs for root cause detection [10].

Additionally, there has been research on correlating alert data to reduce alert noise, such as alert management [3] and improving the quality and efficacy of alerts [11]. However, applying these approaches

in a real-world operation setting can be difficult. Vendors must account for industry, customer, administrator, and user behavior when rolling out a solution. Without careful consideration of environmental factors (e.g. *what is normal in organization X?*) these detectors may become overly sensitive to a specific attack signature and thus be prone to false-positives.

When applying detectors based on human-derived logic, customization, in any form, requires a monitoring period and an ability to tune a detector. If these detectors are too specific, some attacks will be captured, but they will fail to generalize to previously unseen or novel variants. An iterative tuning approach suggested by [12] has proven to be a valuable method for developing powerful detectors.

However, the approach above is only made possible by having detection content to write rules against. Frameworks like Atomic Red Team[14], RTA [15], and Calder [24] provide a library of scripts that replicate living off the land techniques described in the MITRE ATT&CK matrix. These frameworks can be executed by technique ID (e.g. T1088 - Bypass User Account Control) or, in the case of Atomic Red Team, chained together to replicate real-world adversaries. Additionally, there has been a fair amount of work by these researchers to provide a blue team (*defensive*) counterpart to their red team ATT&CK frameworks. AtomicBlue, MITRE CAR, Windows Defender ATP Queries have all released repositories of detectors or heuristics that map to ATT&CK techniques in hopes of fostering a community of sharing. While these frameworks are invaluable to threat researchers writing detectors they may also be leveraged as a labeled dataset for machine learning applications.

4 Approach

ProblemChild relies on system-level data, primarily process events, extracted from Microsoft ETW data to identify anomalous parent-child process chains and hunt for living off the land attacks. In this study, we restrict our attention to Windows events, but the approach may similarly be applied to Linux (using *auditd*). We import the data in bulk using Sysmon[21], an open-source collection tool.

4.1 Data Ingest and Normalization

Data transformation into a numeric representation serves two functions: 1) it helps reduce resource overhead and complexity in storing supplemental data leaving primary data (e.g. *process name*) intact for graphing operations and 2) allows the ProblemChild analytic engine to learn broader details of parent-child relationships, in the scope of an attack, which avoids just learning signatures.

Since ProblemChild is a graph-based analytic framework extracted metadata from each process creation event is stored in the following format and stored in graph format:

- **Node** - object or entity being modeled (e.g. *process name*)
- **Edge** - action taken by an object (e.g. *process create, fork, terminate*)
- **Metadata** - properties or artifacts that describe a node or edge (e.g. *process ID, command line arguments, timestamps*)

The graph is a directed acyclic graph as employed in other works [24, 17, 18].

4.2 Detecting Anomalous Events

Having formed a graph, a series of statistical methods are applied to draw out anomalous parent-child chains and attempt to connect disparate detections based on community detection. Community detection seeks to segment a graph structure based on the relative edge weights between nodes. Often this is represented as the number of “links” between two nodes, but for our use-case, we choose a technique similar to [17] to provide weight assignments for each edge via supervised learning. Specifically, we use *XGBoost*[26], an implementation of a gradient boosted trees model. Unlike previous research, ProblemChild focuses on learning from past malicious and benign process chains to determine a weight correlated with the maliciousness of a given parent-child pair. This becomes a supervised learning problem, that targets the following features to generate a weight for a given edge between (u , v):

- Δt between process creation and termination
- one-hot encoding of $u.child$, $v.parent$
(~100 windows processes + 1 *UNK* slot)

- Process signature info
- Process elevation info
- Process integrity info
- Is process running as system
- Parent-child User mismatch
- Entropy of process name
- Entropy of command line
- TF-IDF (n-grams) of the command line arguments

These data points provide a simple feature vector that can be passed to a *XGBoost* model that has been trained on labeled malicious and benign edges derived from event data. Instead of predicting a binary label, we choose to instead predict the class probability between 0.0-1.0 for a malicious label and use that number as the weight for a given edge.

We then run Louvain community detection on the weighted graph [19] to segment the graph. Community detection should aid in the identification of rare process chains (e.g. *attack sequences*) and provide a structure for "grouped" attack techniques. The resulting dictionary consists of node assignments (e.g. *unique processes*) to a given community.

4.3 Prevalence Engine

When performing a post-mortem on event data, or in hunting within an environment, it may be useful to understand how prevalent a process, parent-child chain, or a process-command line is within a local or target environment. ProblemChild constructs a service at run-time to apply a series of probability calculations to yield a "weight" to multiply against the output of the *XGBoost* model.

First, ProblemChild provides a query interface to get the prevalence of a process using the following method:

$$\Pr(\text{process_count} < \text{PERCENTILE})$$

The score will range between 0 and 99 which represents the percentile of the number of times the process has been seen in the wild. The process prevalence score allows ProblemChild to state *"I've seen this process more than x% of other processes"*. Likewise, detecting the prevalence of a Parent-Child Process chain requires a similar calculation, except we perform conditional probability first, instead of raw counts of a parent-child relationship. So, we let:

$$P(\text{child} \mid \text{parent}) = P(\text{child}, \text{parent}) / P(\text{parent})$$

Since the output of this equation can range widely, we then apply the same percentile technique as above to ensure 0-99 score. Thus, the parent process prevalence score becomes:

$$\Pr(P(\text{child} \mid \text{parent}) < \text{PERCENTILE})$$

The analyst can then determine *"From this parent, I've seen this child more than X%of other child processes"*. This interface to the parent-child prevalence score is accessible via a dictionary lookup similar to `prevalence[child][parent]`.

```
def find_bad_communities(G, threshold):
    bad_communities = [ ]
    communities = community_detection(G, weight=weight)

    for community in communities:
        for node in community:
            prevalence_score = prevalence[node.process_name][node.pprocess_name]
            node['anomalous_score'] = node['weight'] * (1- prevalence_score)
        ψ
    if max([node['anomalous_score'] for node in community]) >= threshold:
        bad_communities.append(community)
```

```
# return most malicious communities first
return sorted(bad_communities, reverse=True)
```

Figure 4: Python code for ranking communities by maliciousness

4.4 Finding Bad Communities

We sought to maximize the malicious_score (*global ML view*) of a given parent-child chain by combining it with its prevalence_score (*local statistical view*) to generate an anomalous_score. We performed this calculation for every node in the graph by iterating through each community (see code block). We then perform a max() across all the anomaly scores for nodes in a given community, if the returned value is \geq the specified threshold the community is deemed malicious and set aside for user review. Upon completing this action for each community in the graph the list of malicious communities is rank-ordered to return the highest scored communities first and suppressing commonly occurring process chains. We hope this will drastically reduce the amount of data and threat researchers have to work with and limit the creation of noisy detectors.

5 Experiments

Prior to the primary experiment, ProblemChild was trained on multiple datasets (both malicious and benign) to establish a normal, global baseline. As mentioned in the previous section, the classifier was used to generate edge weights. The prevalence service was comprised of data from the experiment dataset, representing a local view of rare/common processes.

5.1 Datasets

The ProblemChild model was trained on a combination of real-world and simulated benign and malicious data. Benign data consisted of 3 days of internal Endgame ETW process data. The sources of this data were a mix of user workstations and servers to replicate a small organization. Then we generated malicious data by detonating all the ATT&CK techniques available via the Endgame RTA framework, as well as launching macro and binary-based malware from advanced adversaries like FIN7 and Emotet. We used this combined dataset to train the XGBoost model mentioned in the Approach Section.

Similarly, our evaluation data was comprised of 2 additional days of internal Endgame ETW process data (benign) and Atomic Red Team attack data, plus APT3 data that simulates the 2018 MITRE ATT&CK evaluation (malicious). All data was collected using Windows SysInternals Sysmon[21].

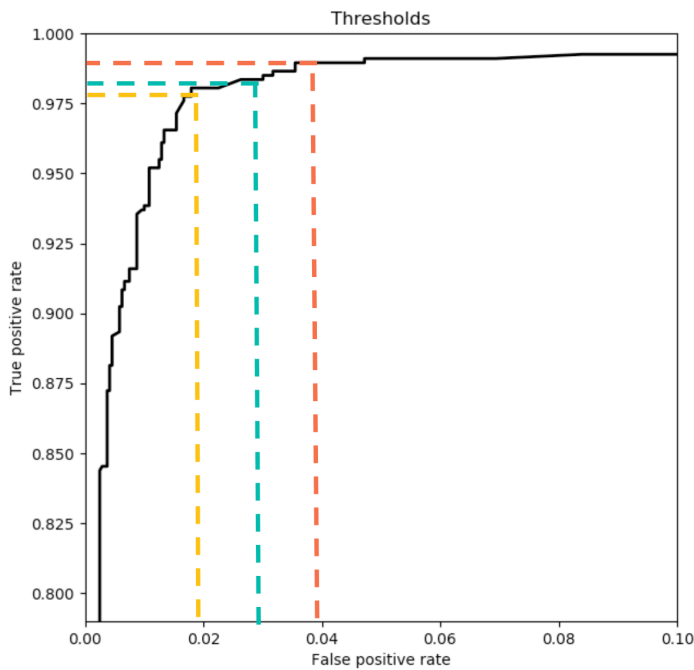


Figure 5: ROC curve w/ target TP/FP Rates

5.2 Determining a Threshold via Leave-one-out (LOO) Retraining

We performed LOO retraining as a method to determine an ideal threshold of malicious communities. The process consisted of retraining a classifier on all but one subset of the training data and then evaluating the holdout to produce a false-positive rate (FPR) and false-negative rate (FNR) for a given model. For example, we trained the ProblemChild classifier on all but one day of benign data from machine A. We then gathered performance metrics by evaluating the model against the held-out data and moved on to repeat the process against data from machine B. Upon completion of this process we determined the ideal threshold to maintain an FPR < 3% was 0.38 (Fig. 5).

The primary cause of false-positives was due to an admin machine that leveraged PowerShell scripting for various tasks (e.g. pushing out updates, automation). Likewise, false-negatives were largely attributed to attack chains that had a single parent-child process event followed by file, registry, and network events.

5.3 Setup

The experiment was presented as a blue team post-mortem exercise. In this case, the red team executed a series of sophisticated attacks that melded together multiple techniques from the ATT&CK matrix. We decided to use event data from the 2018 MITRE ATT&CK Evaluation provided by Cyb3rWard0g’s Mordor project [25]. The ATT&CK Evaluation sought to emulate APT3 activity using FOSS/COTS tools like PSEmpire and CobaltStrike. These tools allow living off the land techniques to be chained to perform Execution, Persistence, or Defense Evasion tasks.

5.4 Results – 2018 MITRE ATT&CK Evaluation (APT3)

APT3 group represents a Chinese-based threat actor associated with China’s Ministry of State Security[20]. The primary target(s) of APT3 campaign had been the US, but in mid-2015 the group shifted focus to political organizations in Hong Kong. APT3 was emulated for the 2018 ATT&CK Evaluation because of its robust, and diverse, post-exploitation tradecraft which relies heavily on living off the land techniques. While this experiment was an ambitious test of ProblemChild’s capabilities there are inherent shortcomings with the setup. First, there is limited user noise in the environment. Second, since the machines were not actively used during the exercise, the adversary activity is unrealistically loud. We still believe it is a valid test of our framework due to the sophistication and diversity of techniques employed.

Overall, ProblemChild was able to recognize groups of ATT&CK techniques executed by the (PowerShell) Empire Project [13]. An example can be seen in below:

```
node8: {'score': 0.7542
  chain: [
    ('cmd.exe', 'C:\\Windows\\system32\\cmd.exe /C ipconfig /all & arp -a & echo
    USERDOMAIN%\\%USERNAME% & tasklist /v & sc query & net start & systeminfo &
    net config workstation'),          (1)
    ('ipconfig.exe', 'ipconfig /all'), (2)
    ('arp.exe', 'arp -a '),            (2)
    ('tasklist.exe', 'tasklist /v'),   (3)
    ('sc.exe', 'sc query '),           (4)
    ('systeminfo.exe', 'systeminfo')   (5)
  ]}
```

This process chain above contains the following techniques:

1. System Owner/User Discovery (*T1033*)
2. System Network Configuration Discovery (*T1016*)
3. Process Discovery (*T1057*)
4. System Service Discovery (*T1007*)
5. System Information Discovery (*T1082*)

Likewise, ProblemChild wasable to identify sequences of Multi-step Persistence, Execution, Exfiltration attempts and lateral movement. However, ProblemChild struggled against sequences where process events

were not the primary trigger. ATT&CK techniques like Multi-band Communication (*T1026*), Remote Desktop Protocol (*T1076*), and Create Account (*T1136*) were completely missed by the platform, highlighting the importance of incorporating additional events types in future research.

Finally, we were able to demonstrate the value of the Prevalence Engine in suppressing FPs that arose during this scenario and reducing the overall number of process creation events from 50K down to 40 "malicious" communities consisting of ~4-6 events in each.

6 Future Work

While the overall performance of ProblemChild demonstrated value in the reduction of process events by identifying anomalous parent-child process chains, a known short-coming is the lack of diverse, realistic data. More work is required to execute, capture, and label malicious binaries. Additionally, we must significantly increase the amount of real-world benign data. Data from admins, developers, and normal enterprise users. That data coupled with the inclusion of other event types (e.g. DNS, registry) should greatly improve our efficacy in delineating good/bad process behavior. Feature extractors for each new event type will require input from threat researchers to determine valuable metadata to target for featurization and inclusion into the model.

Finally, we must continue to better understand the overall performance of ProblemChild in scenarios similar to the APT3 evaluation. We have limited data on the classification performance in this experiment due to the lack of a complete labeled dataset.

7 Conclusion

We have presented ProblemChild, a graph-based analysis framework for automatically uncovering anomalous process-level events. The application of local prevalence increases the ability of ProblemChild to hone in on rare parent-child chains and connect them to larger attack patterns. While detectors based on heuristics perform well in identifying a singular living off the land technique, we believe there is an opportunity for machine learning to help reduce and rank the event data threat researchers based these detectors on, limiting a deluge of FPs.

References

[1] Symantec Response. Petya ransomware outbreak: Here’s what you need to know. 2018.
<https://www.symantec.com/blogs/threat-intelligence/petya-ransomware-wiper>

[2] Wueest, Candid. Living off the land and fileless attack techniques. *An ISTR Special Report*. 2017.

[3] Milajerdi, Sadegh M and Gjomemo, Rigel and Eshete, Birhanu and Sekar, R and Venkatakrisnan, VN.
HOLMES: real-time APT detection through correlation of suspicious information flows, *arXiv preprint arXiv:1810.01594*, 2018.

[4] Whims, Steve. Schtasks.exe - Windows applications.
<https://docs.microsoft.com/en-us/windows/desktop/taskschd/schtasks>

[5] Moe, Oddvar. Github LOLBAS Project. <https://lolbas-project.github.io/>

[6] CISA: Publicly Available Tools Seen in Cyber Incidents Worldwide
<https://www.us-cert.gov/ncas/alerts/AA18-284A>

[7] MITRE ATT&CK T1192 Spearphishing Attachment.
<https://attack.mitre.org/techniques/T1193/>

[8] Gehani, Ashish and Tariq, Dawood. SPADE: support for provenance auditing in distributed environments. *Proceedings of the 13th International Middleware Conference*. 101–120. 2012.

[9] Ma, Shiqing and Zhang, Xiangyu and Xu, Dongyan. Protracer: Towards Practical Provenance Tracing by Alternating Between Logging and Tainting. *NDSS*. 2016.

[10] Milajerdi, Sadegh M and Eshete, Birhanu and Gjomemo, Rigel and Venkatakrishnan, Venkat N. ProPatrol: Attack Investigation via Extracted High-Level Tasks. *International Conference on Information Systems Security*. 107–126. 2018.

[11] Hassan, Wajih Ul and Guo, Shengjian and Li, Ding and Chen, Zhengzhang and Jee, Kangkook and Li, Zhichun and Bates, Adam. NoDoze: Combatting Threat Alert Fatigue with Automated Provenance Triage. *NDSS*. 2019.

[12] Levan, Keshia. Driving Efficacy Through Detector Tuning: Deep Dive Into Detection Engineering.
<https://redcanary.com/blog/tuning-detectors/> Red Canary. 2019.

[13] Github EmpireProject/Empire <https://github.com/EmpireProject/Empire> Oct 2019

[14] Github redcanaryco/atomic-red-team
<https://github.com/redcanaryco/atomic-red-team/> Oct 2019.

[15] Github endgameinc/RTA, <https://github.com/endgameinc/RTA/> Aug 2018.

[16] Github mitre/cascade-server. <https://github.com/mitre/cascade-server> Nov 2018.

[17] Pei, Kexin and Gu, Zhongshu and Saltaformaggio, Brendan and Ma, Shiqing and Wang, Fei and Zhang, Zhiwei and Si, Luo and Zhang, Xiangyu and Xu, Dongyan Hercule: Attack story reconstruction via community discovery on correlated log graph. *Proceedings of the 32Nd Annual Conference on Computer Security Applications*. 583–595. ACM, 2016.

[18] Shu, Xiaokui and Yao, Danfeng and Ramakrishnan, Naren. Unearthing stealthy program attacks buried in extremely long execution paths. *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM. 401–413. 2015.

[19] Blondel, Vincent D and Guillaume, Jean-Loup and Lambiotte, Renaud and Lefebvre, Etienne. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*. 10.2008. *IOP Publishing*. 2018.

[20] Recorded Future Research Concludes (Chinese) Ministry of State Security Behind (APT3)
<https://www.recordedfuture.com/chinese-mss-behind-apt3/> May 2017

[21] Russinovich, Mark. Sysinternals Suite. Microsoft TechNet 2009.

[22] grugq. FIST! FIST! FIST! Its all in the wrist: Remote Exec. Phrack Magazine
<http://phrack.org/issues/62/8.html>

[23] Github endgameinc/eqllib <https://github.com/endgameinc/eqllib> Sep 2019

[24] Github mitre/caldera <https://github.com/mitre/caldera> Sep 2019.

[25] Github hunters-forge/mordor <https://mordordatasets.com> August 2020.


[26] Chen, Tianqi and Guestrin, Carlos Xgboost: A scalable tree boosting system. *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM 785–794. 2016.

[27] George Kour and Raid Saabne.
Real-time segmentation of on-line handwritten arabic script.
In *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*, pages 417–422. IEEE, 2014.

[28] George Kour and Raid Saabne.
Fast classification of handwritten on-line arabic characters.

[29] In *Soft Computing and Pattern Recognition (SoCPaR), 2014 6th International Conference of*, pages 312–318. IEEE, 2014.

Guy Hadash, Einat Kermany, Boaz Carmeli, Ofer Lavi, George Kour, and Alon Jacovi. Estimate and replace: A novel approach to integrating deep neural networks with existing applications.
arXiv preprint arXiv:1804.09028, 2018.

 Feeling lucky?

Conversion report (E)

Report an issue

[View original on arXiv](#)

