Threat intelligence computing for efficient cyber threat hunting

October 17 2018, by Xiaokui Shu

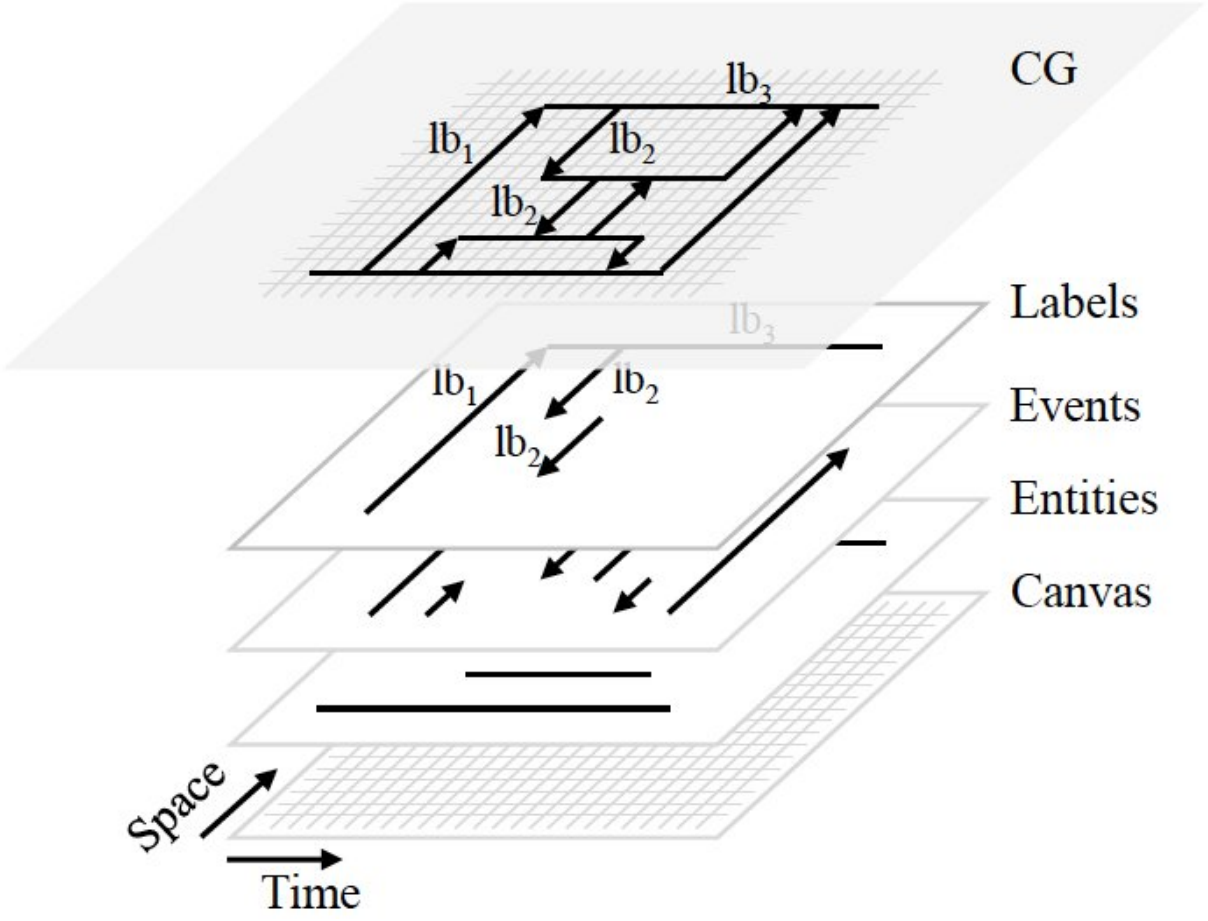

Figure 1. Composition of the computation graph (CG). It comprises a stack of four layers. Canvas determines the scope and granularity of analytics. Entities and Events are created from logs and traces. Labels express knowledge about the entities/events as well as security domain knowledge such as alerts and analytical results. Credit: IBM

Threat discovery in cybersecurity is like scientific discovery: both start from observations, such as an anomalous cyber activity or an interesting fact; both require hypothesis conception, such as what malicious intent is behind the activity or what causes the fact; both develop hypotheses regarding additional observations and finally validate them. Our team at the IBM Thomas J. Watson Research Center recently developed a cyber reasoning paradigm named threat intelligence computing (TIC) to formalize and facilitate the threat discovery process. The paradigm makes it easy and intuitive for security analysts to observe cyber facts and digest data, create and use threat intelligence, and perform human-machine co-development.

Our paper "Threat Intelligence Computing" presents a concrete realization of TIC through the design and implementation of a domain-specific language τ-calculus with a specialized graph database and peripheral systems. We'll present it at the 25th ACM Conference on Computer and Communications Security (ACM CCS 2018). The paradigm leverages the concept of graph computation and provides security analysts with a well-defined interface for observation and hypothesis validation in the cyber world. TIC encapsulates cyber domain knowledge into a threat intelligence representation named composable graph pattern. Using a τ-calculus console, security analysts can create, load, and match patterns to conduct steps in the threat discovery procedure.

The need for threat intelligence computing

Attackers move fast, and we defenders have to move faster. Within an hour of discovering a vulnerable service in an enterprise, an attacker could use that service as a lateral movement channel across hosts. Even worse, the attacker may leverage legitimate channels for malicious purposes such as leaking a commercial secret through an FTP file-exchange server.

Today's security systems themselves are not able to move as fast as attackers. They are capable of detecting partial steps of attacks or well-known attack strategies, but when attackers move beyond the domain knowledge embedded in the algorithms or training datasets, existing rule-based and machine learning systems both fail to move with attackers. What can potentially move fast are security analysts, or threat hunters, who can conceive creative threat hypotheses, gather data to verify hypotheses, and observe additional data to evolve hypotheses.

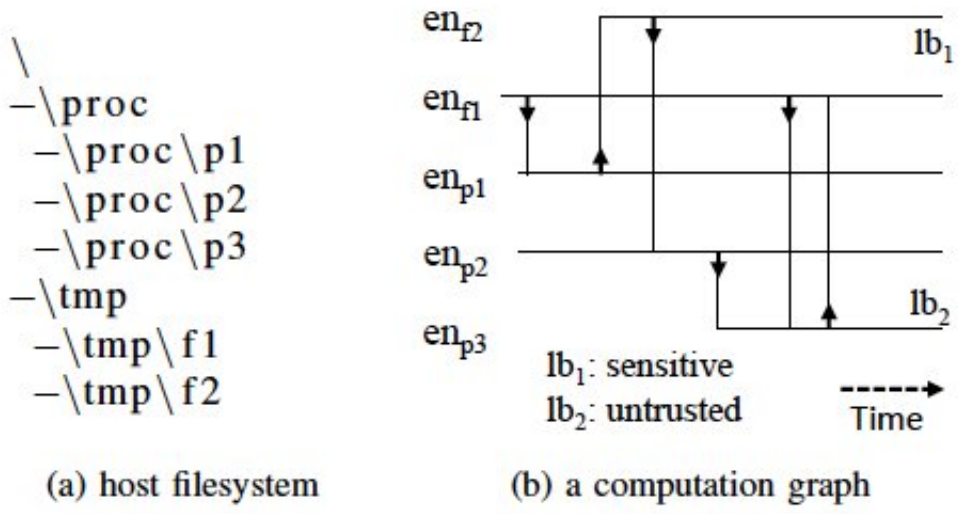

Figure 2. A concrete example of a computation graph on the host level. System activities are logged via syscall monitoring and program instrumentation. Entities in this computation graph consist of subjects (e.g., processes and threads) and objects (e.g., files, pipes, and network sockets). Security data is embedded in labels: lb_1: sensitive indicates that en_{f2} contains sensitive information, and lb2: untrusted indicates that en_{p3} is not certified by the company. Credit: IBM

Unfortunately, their potential is hindered by today's security methodology and support systems. The average enterprise uses more

than a dozen security products to secure its network. It takes a while for an analyst to understand a particular alert from any of the products. It takes some time for her to correlate the alert with related logs in other products for obtaining the wider picture. It takes even longer to go back or forward through time with multiple data sources to understand how the attack happened, what phase it is in, and what impact the suspicious activity has made or will make. What's worse, in case of a customized campaign, an attacker may take advantage of an uncommon service in an enterprise, which may be ignored by an off-the-shelf backtracking system used by a security analyst. If the analyst wants to verify a hypothesis with backtracking on the uncommon service, she has to ask the security vendor to add a feature temporarily, which can hardly be fast.

Ponemon Institute's 2017 Cost of a Data Breach Study shows it takes an average of 206 days to detect a data breach, and it hopes organizations will complete the task within 100 days. We aim even higher. We would like to facilitate steps in the procedure of threat discovery or threat hunting from days to hours, so that defenders can move fast and act even before attackers fully complete their campaigns. A commercial secret could be secured inside the company, or a malicious bank account could be locked, if an analyst finishes reasoning even a minute before the commitment of a critical action in an attack campaign.

To unleash the potential of defenders, we need to make it easy and intuitive for them to observe cyber facts and digest data, fulfill steps in threat discovery, create and use threat intelligence, and perform human-machine co-development. This is why we created TIC, a well-designed cyber reasoning paradigm with all these features to help analysts race against attackers.

The magic of threat intelligence computing

TIC enables agile cyber reasoning development for steps in threat discovery:

```
Taucalculus> pattern patSendmail () { -s s{"cmdline": contain "/var/log/sendmail"}}
[Info] graph pattern patSendmail learned.
Taucalculus> sdml = patS
patSendmail       patSuspiciousIPs
Taucalculus> sdml = patSendmail ()
[Info] solving unary constraint {property} on "s"
[Info] "s" changed, start propagation for it
[Info] initial propagation queue: []
[Info] end propagation for "s"
2 entities, 0 events.
Taucalculus> sdml2 = backtraversal1 (sdml)
[Info] graph pattern backtraversal1 learned.
[Info] solving unary constraint {ingraph} on "x"
[Info] "x" changed, start propagation for it
[Info] initial propagation queue: []
[Info] end propagation for "x"
[Info] solving binary constraint {reachability} on "y" and "x"
[Info] DB query: Joinpoints -> Events; BFS count down: 1
[Info] DB query results: 13 Joinpoints, 78 Events.
[Info] "y" changed, start propagation for it
[Info] initial propagation queue: []
[Info] end propagation for "y"
13 entities, 78 events.
Taucalculus>
```

Figure 3. Screenshot of the τ-calculus console. Users or threat hunters create/load/execute graph patterns in the console to perform observation and threat hypothesis verification. Credit: IBM

- Observation (e.g., did a specific process talk to the network via a particular protocol, and what were the network activities?)
- Threat hypotheses validation (e.g., did this process perform DLL injection or send out a specific file yesterday?)

We do this by transforming a threat discovery problem into a graph computation problem and solving it within the new TIC paradigm. In TIC, all security logs, traces, and alerts are stored in a temporal graph or

computation graph (CG). The CG records the history of monitored systems, including benign activities and malicious ones, as interconnected entities and events. Threat discovery in TIC then becomes a graph computation problem to identify a subgraph of CG that describes a threat or an attack, with the help of alerts and security domain knowledge stored as element labels.

The simple yet powerful way that TIC fulfills steps in threat discovery is by composable graph pattern matching. First, an analyst can compose graph patterns describing suspicious behaviors like DLL injection or port scan. Matching these patterns is no different from executing traditional intrusion detection systems. Second, an analyst can compose graph patterns to explore CG and make observations before creating or adjusting threat hypotheses. A graph pattern with two entities connected by an event plus some specified attributes can easily answer the question of what network activity a specific process had. Third, a graph pattern can be created with other graph patterns like LEGO. An analyst can write a backtracking pattern with a traversal constraint as a separate pattern, which behaves like a call-back function and is specified at runtime as an argument of the backtracking pattern. The composable pattern design allows one to codify abstract knowledge into patterns and conduct higher-order reasoning.

Testing threat intelligence computing

A paradigm describes a world of computation with basic rules. It only shines when realized with a concrete programming environment. We accomplish this in a new graph computation platform named τ-calculus. The platform provides a similar level of interactive programming experience compared to Metasploit, the most popular platform for conducting ethical hacking with moving attack strategies. While penetration testers can compose and test exploiting procedures by stitching different components in the Metasploit console, security

analysts now can create, load, and match graph patterns in the τ-calculus console to conduct creative threat discovery. To support intuitive observation, we developed τ-browser, a visualization tool for τ-calculus to interactively examine subgraphs and results of matched patterns.

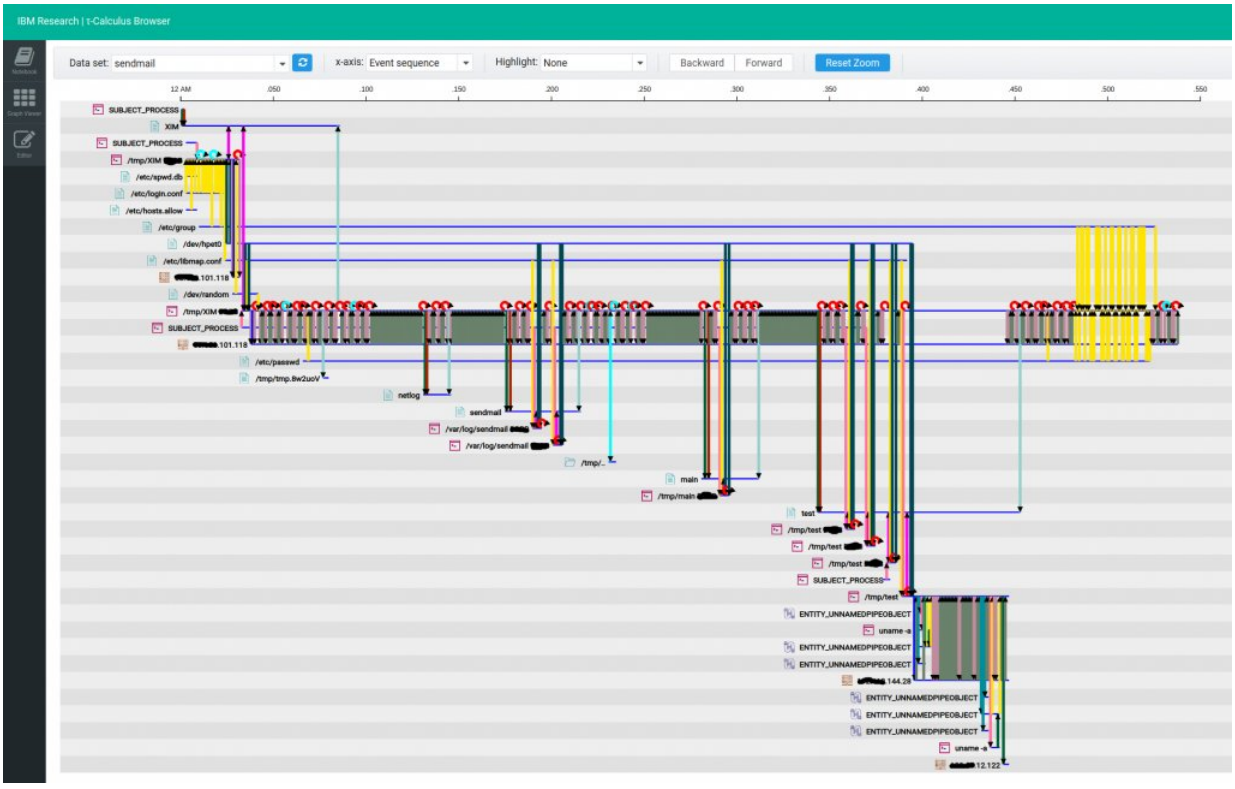

Figure 4. A red team attack campaign discovered and visualized in τ-browser. Users or threat hunters leverage the interactive visualization to inspect subgraphs matched from graph patterns and plan for new patterns. Credit: IBM

We tested the practicality and utility of the paradigm in a two-week red-teaming event covering stealthy attack campaigns on multiple systems ranging from servers to mobile devices. The campaigns were deliberately planned with in-house malware development to bypass traditional

detection systems such as antivirus protection. Stealthy tactics like in-memory attacks were employed with freshly released DoublePulsar/EternalBlue vulnerabilities to test threat discovery against 0-day attacks. During the two-week online detection, about a billion records were produced, streamed, and analyzed through our system.

Dozens of threat-hunting tasks were dynamically planned and programmed, and attack campaigns with various malicious intents were discovered in a timely manner including the attack with DoublePulsar.

Beyond the basics of threat intelligence computing

Besides the agile cyber reasoning we demonstrated in the red-teaming event, the TIC paradigm opens the door to a new world for solving complicated cybersecurity problems. It fundamentally addresses the heterogeneous data headache and encodes all types of logs, traces, provenance information, and even security alerts and knowledge into the temporal graph CG. One can build a network-level CG from NetFlow data, build a host-level CG from system logs and traces, and fold the second into the first with zoom in/out support. TIC lifts threat intelligence onto a new level: a pattern in TIC provides actionable knowledge about threats with context and potential countermeasures. Generating, sharing, and consuming threat intelligence becomes nature in TIC. Moreover, TIC interprets cyber reasoning into explicit graph computation steps. Traditionally hard-to-model opaque human knowledge can now be recorded to train analysts or AI systems.

Big data, threat hunting, agile development, knowledge extraction, and AI potential: TIC brings all these components together into an efficient threat discovery paradigm against stealthy attacks and persistent threats. It offers fine-grained, well-organized security data; protection of cyber assets beyond security products with static knowledge in their algorithms and training datasets; and dynamically formed, quickly evolving

detection strategies as effective countermeasures against customized attack. For more inspiration and realization guidance on TIC, look for our ACM CCS 2018 paper, "Threat Intelligence Computing," in the ACM Digital Library shortly after the conference.

More information: Xiaokui Shu et al. Threat Intelligence Computing, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security - CCS '18* (2018). DOI: 10.1145/3243734.3243829

Provided by IBM