DACA: Automated Attack Scenarios and Dataset Generation

Frank Korving and Risto Vaarandi
Centre for Digital Forensics and Cyber Security, Tallinn University of Technology, Tallinn, Estonia

korving.f@gmail.com
risto.vaarandi@taltech.ee

Abstract: Computer networks and systems are under an ever-increasing risk of being attacked and abused. High-quality datasets can assist with in-depth analysis of attack scenarios, improve detection rules, and help educate analysts. However, existing solutions for creating such datasets suffer from a number of drawbacks. First, several solutions are not open source with publicly released implementations or are not vendor neutral. Second, some existing solutions neglect the complexity and variance of specific attack techniques when creating datasets or neglect certain attack types. Third, existing solutions are not fully automating the entire data collection pipeline. This paper presents and discusses the *Dataset Creation and Acquisition Engine* (DACA), a configurable dataset generation testbed, built around commonly used *Infrastructure-as-Code* (IaC) and *DevOps* tooling which can be used to create varied, reproducible datasets in a highly automated fashion. DACA acts as a versatile wrapper around existing virtualization technologies and can be used by blue as well as red teamers alike to run attack scenarios and generate datasets. These in turn can be used for tuning detection rules, for educational purposes or pushed into data processing pipelines for further analysis. To show DACA's effectiveness, DACA is used to create two extensive datasets examining covert DNS Tunnelling activity on which a detailed analysis is performed.

Keywords: security dataset, testbed, DevOps, detection engineering

1. Introduction

DevOps tooling and practices have been widely adopted by both development and IT-operational teams to automate the building, testing and deployment of software and the infrastructure it runs on. One of the effects of this adoption is an increase in trust of the developed solutions since existing *Quality Assurance* (QA) processes can be transformed into more rigid ones by automating and enforcing otherwise labour-intensive steps like regression testing.

Red teams are using these same tools in increasingly creative ways to systematically enforce QA practices like uniquely obfuscating exploit code to avoid being signatured or to test exploit-viability and defence evasion techniques under varying conditions before deployment.

Blue teams can utilize these tools as well by integrating static and dynamic analysers into development pipelines or Vulnerability Management programs. They can also be used to build easily reproducible lab environments which facilitate adversary emulation scenarios and analysis of produced attack artefacts. Security datasets exist that can help with *Intrusion Detection System* (IDS) / *Intrusion Prevention System* (IPS) rule tuning and validation as well as provide valuable practice data for Detection engineers. This has inspired the author to explore the combination of these two topics: automated creation of security datasets using DevOps tooling.

1.1 Motivation

Security Datasets exist in many data types (e.g. Network vs. Flow vs. Log) and data formats (e.g. PCAP vs. EVTX vs. flat file), some are open-sourced, others only available on request, some are artificial, others are captures of a real-world infrastructure setup, some target *Machine Learning* (ML) use-cases while yet others target IDS/IPS use-cases. Tools exist which can manipulate, augment or help create such datasets and can be used to help validate detection capabilities (Brauckhoff et al, 2008; Cordero et al, 2015; Vasilomanolakis et al, 2016).

The creation of such datasets is often a very time-consuming task, however not commonly automated. When tooling *is* created to auto-generate such data it is either not published or shared-on-request, making validation and adjustments very hard.

Unfortunately, published datasets have some other commonly encountered deficiencies like: poor data quality, unclear methodology, are unmaintained or unmaintainable and are often unreproducible (Kenyon et al, 2020).

To illustrate why having high-quality datasets can be important, one can look at a project like Sigma, which aims to provide a vendor-neutral detection format for *Security Information and Event Management* (SIEM) solutions. Of the over 2300 published detection rules that come with the project, only 94 were marked as *stable* as of this writing and were tested with available attack data.

This makes it worth while to investigate whether the creation of a more generic, open-source, automated framework to run adversarial / attack scenarios and extract valuable data is feasible, adjustable and can contribute meaningful datasets to the security community.

1.2 Novelty and contributions

Setting up testbeds using IaC for the use of Security research or dataset creation in particular is not a novel idea on its own. However existing solutions:

- Are not always vendor neutral. Either in the format of the datasets they create, or the technology that is leveraged to create the datasets.
- Are not always open-source or are underdeveloped. Developed tools might be mentioned in a paper, but not publicly released.
- Neglect complexity and/or variance of specific attack techniques when creating datasets.
- Don't fully automate the entire setup, attack, data collection pipeline.

Therefore this research aims to create an open-source, adjustable tool called DACA and/or set of configurations that can run end-to-end automated attack scenarios and extract security datasets from the systems under analysis.

The main considerations that went into the design of DACA are:

- *Modularity* – Many different commonly used provisioning and orchestration tools exist to setup IT infrastructure. The tool should allow for an abstraction layer so that new platforms can be supported with minimal effort or code duplication.
- *Flexibility* – The tool should allow for a wide variety of scenarios and attacks to be carried out where ideally a researcher's imagination is the limiting factor.
- *Functionality* – The tool should be able to create reproducible, useful datasets in a highly automated fashion inspired by *Continuous Integration / Continuous Deployment* (CI/CD) parallelization techniques.

The main contributions of this paper are:

- A publicly available toolkit that creates security datasets for attack scenarios.
- Publicly available security datasets created with the toolkit.
- *Intrusion Detection System* (IDS) rules that have been developed with the help of the datasets.

1.3 Related works

This section covers various testbeds, tools that either generate or manipulate datasets as well as relevant datasets themselves. A more comprehensive overview of these tools and datasets can be found in comprehensive dataset studies of the thesis this work was originally based on (Korving, 2022).

1.3.1 Testbeds

Various testbeds have been created that can facilitate research into information systems or security based research in particular:

- Cloudlab, which provides a cloud-like platform. It can be seen as an extension of Emulab (Hibler et al, 2008), another testbed focused on providing a shared environment for researchers (Duplyakin et al, 2019).
- The DETER Project which is a US national Cyber Security experimentation testbed providing infrastructure and tools to security researchers and educators (Benzel et al, 2011).

The following set of testbeds are not represented in the literature, but do come close to this research's design goals. They are publicly available, create production-like environments with security tooling and are adjustable. However, there is a lack of attack automation and/or automatic data extraction in these testbeds which is a significant drawback.

- Splunk's Attack-Range spins up small lab environments with proprietary detection and analysis solutions, which allows for attack simulation and immediate data analysis for building IDS rules.
- DetectionLab is a set of scripts that allow the automated construction of an Active Directory lab environment with monitoring tooling installed.

One notable domain specific example is an *Industrial Control System* (ICS) targeted testbed called HAI. While the technology stack is not reusable for this particular research, it comes very close to the author's research design and research goals of automating an otherwise very manual attack / dataset extraction process. (Hyeok-Ki et al, 2019)

1.3.1 Datasets

There are multiple relevant public (static) datasets available as well as projects that aim to become security dataset exchanges. Two of the most related to this work are:

- **Security Datasets** (formerly *Mordor*) which is an open-source initiative to collect multi-platform datasets to improve detection capabilities.
- **Canadian Institute for Cybersecurity** associated with the University of New Brunswick produces and publishes many types of security datasets targeting anything from IoT environments to Android malware and *DNS-over-HTTPS* (DoH) (Mohammadreza et al, 2020).

2. Design and Development

DACA is a *Command-line Interface* (CLI)-based tool written in the Python programming language that comes with its own configuration specification to define attack scenarios and acts as a wrapper around existing DevOps tools. For the code that executes scenarios an Object Oriented Programming approach was taken. This allows the tool to be extended to new platforms by implementing an interface (i.e. Controller interface) into new execution classes (e.g. a VagrantController or DockerController class). This avoids having to touch the existing code base and therefore works towards modularity and extensibility design goals of the tool. An orchestration class called ScenarioRunner delegates actual execution to the relevant Controller code.

2.1 Language and Dependencies

DACA was written and tested in Python (v3.8+), dependency management for used libraries is performed using Pipenv. The initial implementation comes only with Vagrant support as a scenario provisioner and therefore needs to be installed together with one of their supported hypervisors (e.g. Virtualbox).

2.2 Execution Phases

When a scenario file is passed to DACA it will go through multiple distinct phases in its execution. An overview of these execution phases can be seen in Figure 1.

1. **Definition** – Scenarios are defined in YAML-based configuration files and can utilize Jinja functionality like variables and filters. These are then transformed into python dictionaries.
2. **Validation** – The python dictionary is validated against a defined schema using Cerberus.
3. **Compilation** – Depending on which backend technology is targeted (e.g. Vagrant) the dictionary is compiled into the appropriate configuration file format describing things like machine type and networking.
4. **Setup** – A provisioning step for the defined machine. Here required services are installed which can be done through inline commands, script files or Ansible playbooks.
5. **Run** – A script or series of commands that should be executed on machine startup. For example the start of a data capture or the execution of an attack-sequence.
6. **Data Collection** – In this step any produced artefacts are collected from the machines in the testbed (e.g. log files and network capture files). Some collection methods require streaming data (e.g. Elasticsearch and Kafka output) and live more in the Setup / Run phases.

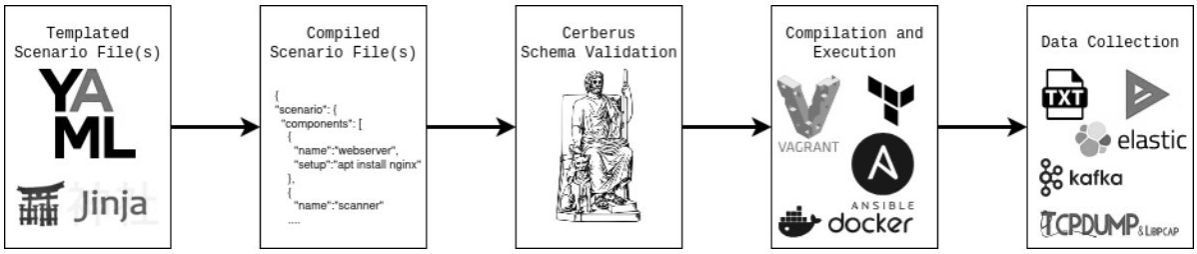

Figure 1: DACA – Step by step execution overview from scenario file to data collection

2.3 Configuration File

DACA supports the definition and execution of attack scenarios through a configuration file. This was implemented by creating a YAML-based configuration language that makes use of a powerful templating engine called Jinja and was used to fulfil flexibility requirements. This solution was chosen in favour of other templating engines because it directly targets python projects and it has proven its value for configuration management tasks by being the primary templating engine supporting some of the most widely used configuration management tools at this time, Ansible and Salt.

To make sure misconfigurations are not allowed to proceed to a Scenario's execution, a schema for the configuration file format was created. This schema definition and validation was performed using a python library called Cerberus. A YAML file is read, transformed into a python dictionary where its keys/values are compared against a schema where attributes like data types, valid values and presence are checked. See the original thesis this paper was based on for the actual DACA configuration file schema (Korving et al, 2022). Cerberus allows to check for slightly more complex, logical errors within the configuration file as well, for example by expressing dependencies between fields or writing custom validation functions. These are not currently used, but could become especially useful when more platforms are supported.

Configuration files have a special section with defined variables and a set of possible values. This section is stripped away in the scenario compilation step after schema validation and substituted into the other sections of the configuration file. This allows a single configuration file to produce multiple execution scenarios. See also Figure 2.

When multiple variables are defined a Cartesian Product is calculated between them, producing a set of variable groups. For example a popular WordPress scanner called "*WPscan*" can enumerate various WordPress objects (e.g. "*u*"-flag indicates user objects) and add Web Application Firewall (WAF) evasion flags. A configuration example for this functionality can be seen in Figure 3 and the resulting set of commands in Table 1. This allows for extensive profiling and possibility of finding unexpected edge cases.

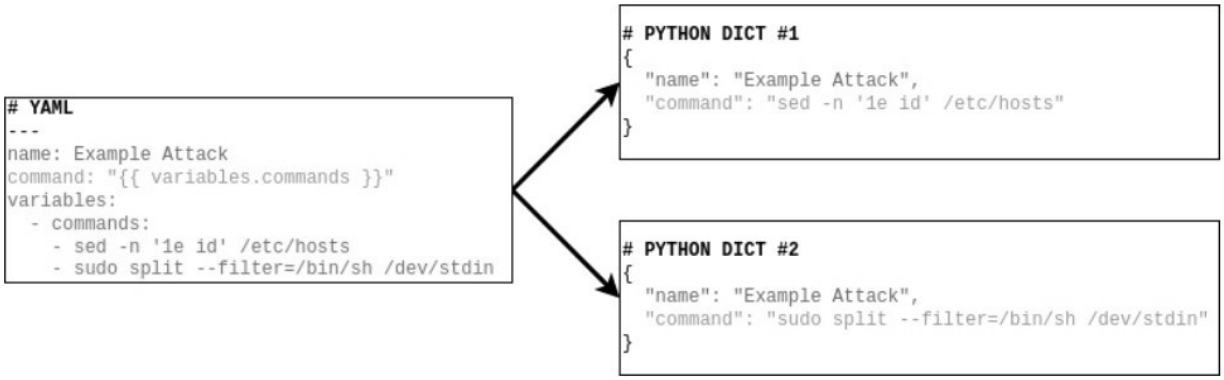

Figure 2: DACA – Single configuration resulting into multiple scenarios using Jinja variables.

```
# YAML
---
name: Example WordPress Attack
command: wpscan --url example.com "{{ variables.flags }}" -e "{{ variables.wp_objects }}"
variables:
  - wp_objects:
    - "u"
    - "cb"
    - "vp"
  - flags:
    - "--stealthy"
    - "--random-user-agent"
```

Figure 3: DACA – WordPress attack variation using Jinja variables.

Table 1: WordPress attack variations

	--stealthy	**--random-user-agent**
u	wpscan –url example.com –stealthy -e u	wpscan –url example.com –random-user-agent -e u
cb	cb wpscan –url example.com –stealthy -e cb	wpscan –url example.com –random-user-agent -e cb
vp	vp wpscan –url example.com –stealthy -e vp	wpscan –url example.com –random-user-agent -e vp

2.4 Compilation

DACA is a wrapper around existing IaC/virtualization technologies which means scenarios need to be translated into a configuration language the target technology understands. This is accomplished using the same Jinja templating engine used in the DACA configuration language itself. The final compiled configuration file is one of the outputs of the tool which make the generated datasets reproducible.

Figure 4 shows a small section of the master Vagrantfile template, its use of variables and basic control structures. Figure 5 shows an example scenario file and the final result. The following 3 variables are used in the template and need to be defined through the DACA scenario file:

- **hostname** – Name of the component.
- **setup['type']** – This variable controls how the value stored in *setup['val']* is interpreted. '*shell*' forces inline command execution while '*script*' and '*ansible*' expect a filename for a shell script or Ansible playbook, which is copied to the target machine and then executed during runtime.
- **setup['val']** – The inline commands, shell script name or Ansible playbook filename which will be executed.

```
1  # Provision VM
2  {%- if setup['type'] == 'ansible' %}
3  {{ hostname }}.vm.provision "ansible_local" do |a|
4      a.install = true,
5      a.install_mode = "default",
6      a.playbook = '{{ setup['val'] }}',
7      a.become_user = "root",
8      a.become = true
9  end
10 {% elif setup['type'] == 'shell' %}
11 {{ hostname }}.vm.provision "shell", inline: '{{ setup['val'] }}', privileged: true
12 {% elif setup['type'] == 'script' %}
13 {{ hostname }}.vm.provision "shell", path: '{{ setup['val'] }}', privileged: true
14 {% endif %}
```

Figure 4: DACA – Templated provisioning section in Vagrantfile.

```
# YAML
---
name: bind9
description: Example BIND9 DNS Server Provisioning
provisioner: vagrant
setup:
  type: ansible
  val:  bind9_playbook
```

→

```
# VAGRANTFILE
bind9.vm.provision "ansible_local" do |a|
    a.install = true,
    a.install_mode = "default",
    a.playbook = 'bind9_playbook',
    a.become_user = "root",
    a.become = true
end
```

Figure 5: DACA – Scenario to Vagrantfile compilation

2.5 Data Collection

DACA allows for multiple types of data to be collected by specifying an "*artefacts_to_collect*"-section under a defined component in a scenario. The currently supported outputs include:

- **Files** – A list of files can be specified which will be collected when the scenario run-through has ended.

- **PCAP** – This expects a tcpdump command which is run as a background task on startup and cleanly stopped at shutdown. Any produced PCAP files need to be specified in the "*files*"-section for them to be collected.
- **Filebeat** – A list of files can be specified to be monitored using Filebeat. Output will be directed to "*/tmp/filebeat.json*" and can be more useful than simple flat files since a lot of metadata will be attached that can be retroactively ingested into Elasticsearch for easy searching. Specifying a list of files under the "*Filebeat*", the "*Elasticsearch*" or the "*Kafka*"-section triggers a dedicated Ansible playbook to be run for installation and configuration of Filebeat. Since a Filebeat instance only supports a single output, these three options are mutually exclusive.
- **Elasticsearch** – Expects an Elasticsearch HTTP endpoint. All files defined in the "*files*"-section (except for PCAP and shell recording files) are monitored and pushed to the defined endpoint.
- **Kafka** – Expects the address of a Kafka listener. Pushes data into a dedicated topic per monitored logfile. This output can be useful when live data streams or additional post processing is wanted (e.g. data normalization).
- **Asciinema** – Asciinema is a terminal session recording utility which allows for attacker-perspective recording and playback. This stanza requires the name of an output file. It will cause the shell commands that are part of the "*run*"-section (and their outputs) to be recorded.

3. Validation

3.1 Implementation Comparison

As mentioned in Section 1.3.1, multiple other testbeds exist. A limited comparison between these available testbeds can be seen in Table 2, highlighting some of the most important qualities in relation to this work.

Aspects like scalability, usability and flexibility are important but are left out since they are subjective or hard to quantify without performing an extensive comparative analysis. While DACA has only limited platform support as of this writing, its focus on automation makes it stand out when compared to the other identified testbeds.

Table 2: Implementation Comparison.

Tool	Publicly Available	Proprietary Data	Supported Platforms	Automated
DACA	Yes	No	Linux, Partial Windows through Vagrant	Fully, Partial
Splunk Attack Range	Yes	Yes	Linux, Windows, AWS, Azure through Vagrant and Terraform	Partial
DETER	No	No	Linux, Windows, Containers through QEMU and Openvz	Partial
Cloudlab	No	No	Linux, Windows through OpenStack	Partial
Detectionlab	Yes	No	Linux, Windows, MacOS, AWS, Azure and more through Terraform	Partial

3.2 DNS Tunnelling Scenario

The following section discusses the run-through of one of the implemented DNS tunnelling scenarios which functions as a test and validation of the developed tool. In total 136 attack variations were performed each resulting in their own sub dataset. 116 were gathered in fully automated mode (file transfer) and 20 of these were performed in DACA's interactive mode (C2). The evaluated DNS servers were: PDNS, Bind9, DNSmasq and CoreDNS. The evaluated tunnelling tools were: Iodine (file transfer only), dns2tcp and dnscat. Suricata was used for additional data analysis.

To design a scenario which emulates a DNS Tunnelling attack it needs to be boiled down to the minimal needed components while still being able to generate the data samples one wants to obtain. The three needed components are: a "compromised" client, a victim DNS resolver and a malicious authoritative DNS server. These components are all deployed within the same network, where requests for a faux domain (*example.attack*) are directly forwarded from the provisioned victim DNS servers to our provisioned attacker authoritative DNS server. Figure 6 shows a step-by-step execution overview of these attacks:

1. The scenario gets rendered into multiple instances, each of which gets compiled into a valid Vagrantfile. Attacks vary by the used tunnelling tool (e.g. iodine vs. dnscat2), data encoding (base32 vs. base64), record type (e.g. TXT vs. NULL) or victim DNS server (e.g. Bind9 vs. CoreDNS).
2. The wrapper-class around the vagrant CLI-utility brings all the defined Virtual Machines (VM) up.

3. The provisioning sections are executed which install the DNS servers and setup zone forwarding / logging configurations. It also prepares the attack server and client.
4. Run-time triggers are defined to start DNS packet captures, start the DNS servers and initiate the DNS tunnel.
5. Once the main attacker's run-time command exits, the VMs are shutdown which triggers the shutdown actions. This includes stopping the packet capture, collecting files and performing artefact clean-up tasks.
6. Data is collected and stored in a dedicated path, along with metadata before moving on to the next execution.

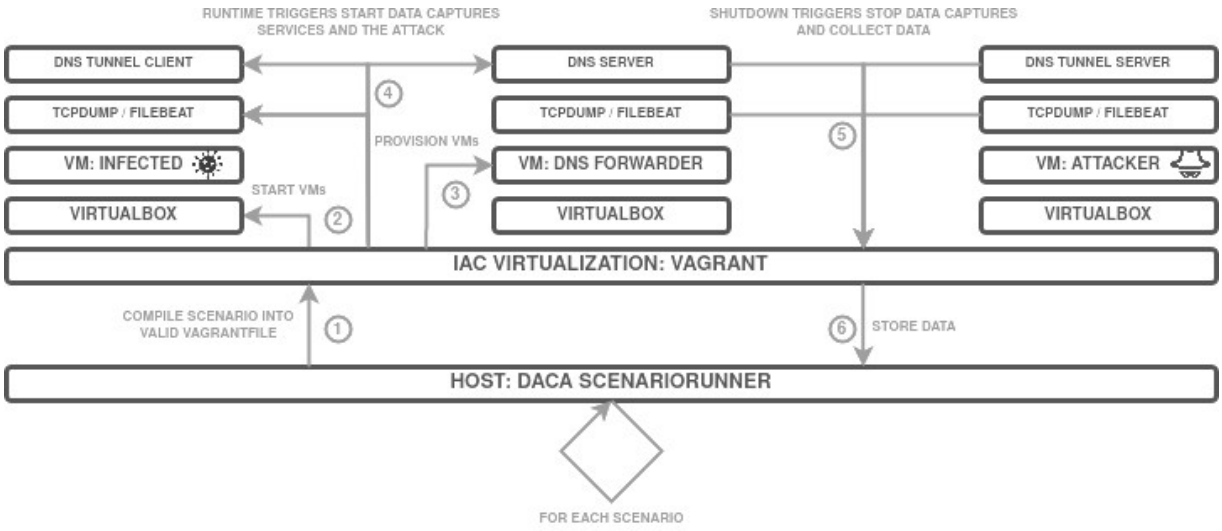

Figure 6: DACA – ScenarioRunner / VagrantController Scenario execution.

3.3 Data Analysis

This section provides a short analysis of data collected during scenario executions, for more details please see the original thesis this paper was based on (Korving et al, 2022). The analysis of collected data allows to assess how tested solutions are responding to attacks and what evidence they produce about the attacks. Furthermore, the evidence that has been identified during data analysis allows human experts to write signatures for detecting these attacks in production environments.

To give an idea of the volume of log data that gets generated when running six exploratory C2 commands ("*whoami*", "*w*", "*pwd*", "*uname -a*", "*ip a*", "*env*"), on average approximately 250 log lines were created for the tunnel establishment, control messages and commands/responses. There is a small but noticeable difference between the two DNS tunnelling tools used: dnscat2 and dns2tcp. It also reveals the fact that DNS servers can produce wildly different amounts of log volumes when query logging has been enabled. PowerDNS needs mentioning since it produced an order of magnitude more data than any of the other tested DNS resolvers although it bears mentioning that the solution's documentation warns users of this fact.

An additional volumetric difference between the DNS Servers was that Dnsmasq had more than double the amount of log lines than the other two remaining DNS servers under evaluation: BIND9 and CoreDNS. While all tested DNS servers log incoming queries, only two make attempts at logging responses, these latter two do not, which explains the difference in volume. The lack of response logging diminishes forensic usefulness since C2 commands and/or any dropped payloads are contained within the responses.

When looking at the produced data itself we can make some other observations, starting with how DNS servers handle writing logs to disk when special or non-printable characters are present in the request/response. These characters mostly occur due to query compression to increase link bandwidth but other mentionable encoding issues occur as well. We would expect similar characters to occur when for example other injection attacks are being performed (Philipp et al, 2021). Octal escape of the raw byte seems to be the standard way to approach this problem (i.e. *000* to *255*) and is adopted by CoreDNS, PowerDNS and Bind9 while Suricata converts special characters to Hex with double backslashes (i.e. "*xee*" or "*xc8*"). This escaping of non-ASCII characters is prudent to avoid injection attacks. Dnsmasq's approach was found to be somewhat inconsistent. No user queries are logged when even a single non-ASCII character is found. However dnsmasq seems to allow logging the

original query in the response message when all characters fall within the wider ISO 8859-1 character set. The response will be printed character by character until a prohibited character is encountered and seems to abandon response logging altogether (even without the 'unprintable' message) when a non supported byte or record type is encountered. See also Figure 7.

```
1 | Mar 31 07:16:42 dnsmasq[1586]: query[TXT] <name unprintable> from 192.168.0.30
2 | Mar 31 07:16:42 dnsmasq[1586]: forwarded <name unprintable> to 192.168.0.20
3 | Mar 31 07:16:42 dnsmasq[1586]: reply rayadíÙBûüý½« REST OMITTED
          »ü.example.attack is r << REMNANT RESPONSE DATA IS NOT PRINTED >>
```

Figure 7: Special characters in log lines.

One basic way to creating signatures for these tools is to find cleartext markers or control messages used by the tool when establishing the tunnel. Figure 8 shows some example ones used by dns2tcp and Figure 9 shows some for iodine.

```
1 | XXXXX.=auth.example.attack
2 | XXXXX.=connect.example.attack
3 | XXXXX.0k.example.attack
4 | XXXXX0.k.example.attack
```

Figure 8: Dns2tcp control messages.

```
1 | XXXXaAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ.example.attack
2 | XXXXaA-Aaahhh-Drink-mal-ein-J\228germeister-.example.attack
3 | XXXXaA-La-fl\251te-na\239ve-fran\231aise-est-retir\233-\224-Cr\232te.example.attack
```

Figure 9: Iodine control messages.

As part of this analysis we've seen a lack of response logging in BIND9 and CoreDNS, excessive debug logging in PowerDNS and inconsistent handling of non-printable characters in Dnsmasq. Some straightforward way to profile these tools was discussed. In addition query logging can have significant overhead to the DNS server and induce query response latency. If one wants to reliably detect the described or similar techniques some other solutions exist that could be considered:

- Dnstap is natively supported by BIND9, CoreDNS and PowerDNS (amongst others) and can be used to avoid IO performance penalties by using a binary output format as well as gain response logging. Data forwarding might be a problem with this solution making centralized analysis harder.
- Tcpdump or DNSCAP can collect DNS traffic passively which has potentially similar data collection issues as Dnstap but would give the full exchange.
- An IDS like Suricata or network analysis tool like Arkime can ingest network data through network TAPs and allows for request/response DNS logging and analysis.

3.4 Evaluation Discussions

3.4.1 Limitations

At the moment the tool only supports the design of local VM based scenarios, which means cloud-native attack scenarios are not yet supported nor can the cloud be utilized to scale out scenarios or parallelize the scenario execution phase. This also means it's not yet possible to design scenarios built onto other virtualization platforms like Docker. Adding support for Docker would not add the ability for new types of scenarios to be developed, in fact it's not suitable when analysing a class of attack scenarios where privileged access is needed to the underlying host (e.g. when installing/reconfiguring an EDR or HIDS solution). However for many situations it would greatly shorten the execution time and would allow more parallelization which in turn allows for a shorter scenario design-development-test lifecycle.

Other limitations of the tool exist that are related to networking and Windows platform support. This is addressed more in the original work (Korving et al, 2022).

3.4.1 Availability information

The code of DACA has been published under a permissive open source license and can be found at the following link: https://github.com/Korving-F/DACA/

Two example DNS tunnelling datasets described in this paper were similarly released and can be found here:

- https://github.com/Korving-F/dns-tunnel-dataset
- https://github.com/Korving-F/doh-tunnel-dataset

4. Future Work

The presented work can be extended, matured and improved upon. The following list describes some ideas on how to do so. Extending to the cloud and other virtualization platforms were already mentioned, so were some limitations that can be addressed.

- Windows Domain – A common requirement for production-like scenarios would likely be to have all participating machines be domain-joined. This requires specialized knowledge on the part of a scenario designer, not relevant to the scenario design at hand and would add a lot of boilerplate code to many scenarios. This setup should be templated for simple deployments and be enabled through a flag in the scenario configuration file (i.e. "*create_domain: true*").
- Built-in Analytics – Live data analysis is dependent on externally setup Kafka and Elasticsearch clusters and their network being reachable. It would be a nice thing to also include these solutions, as well as a data normalization pipeline from within the testbed itself.
- Another challenging research topic is the creation of benign datasets. DACA supports the creation of such datasets, but requires extensive time and energy to be invested into the scenario design.
- Since one of the main value propositions for DACA is automation, scenarios built around automatable C2 (e.g. Mythic) and Adversary Emulation tools (e.g. Atomic Red Team) is something that the author would like to see implemented.

References

Benzel, T. (2011) *The Science of Cyber Security Experimentation: The DETER Project*, Proceedings of the 27th Annual Computer Security Applications Conference, pp 137–148.

Brauckhoff, D. and Wagner, A. and May, M. (2008) *FLAME: A Flow-Level Anomaly Modeling Engine,* USENIX CSET '08

Cordero, C. and Vasilomanolakis, E. and Milanov, N. et al. (2015) *ID2T: A DIY dataset creation toolkit for Intrusion Detection Systems*, 2015 IEEE Conference on Communications and Network Security (CNS)

Duplyakin, D. and Ricci, R. and Maricq, A. et al. (2019) *The Design and Operation of Cloudlab*, USENIX ATC '19.

Hibler, M. and Ricci, R. and Stoller, L. et al. (2008) *Large-Scale Virtualization in the Emulab Network Testbed*, USENIX 2008 Annual Technical Conference

Hyeok-Ki, S. and Woomyo, L. and Jeong-Han, Y. et al. (2019) *Implementation of Programmable CPS Testbed for Anomaly Detection*, 12th USENIX Workshop on Cyber Security Experimentation and Test (CSET 19),

Kenyon, A. and Deka, L. and Elizondo, D. (2020) *Are public intrusion datasets fit for purpose characterising the state of the art in intrusion event datasets*, Computers & Security, Vol. 99.

Korving, F. (2022) *DACA: Automated attack scenarios and dataset generation*, Master thesis, Tallinn University of Technology, Department of Software Science.

Mohammadreza, M. and Logan, D. and Gurdip, K. and Arash, H. (2020) *Detection of DoH Tunnels using Time-series Classification of Encrypted Traffic*, The 5th IEEE Cyber Science and Technology Congress, Calgary, Canada, August 2020.

Philipp, J. and Haya, S. (2021) *Injection Attacks Reloaded: Tunnelling Malicious Payloads over DNS*, 30th USENIX Security Symposium (USENIX Security 21).

Vasilomanolakis, E. and Cordero, C. and Milanov, N. (2016) *Towards the creation of synthetic, yet realistic, intrusion detection datasets*, 2016 IEEE/IFIP Network Operations and Management Symposium