Using ILP to Analyse Ransomware Attacks

Oliver Ray[1], Samuel Hicks[1], and Steve Moyle[2]

[1] Department of Computer Science, University of Bristol
oliver.ray@bristol.ac.uk, sam.hicks.2014@my.bristol.ac.uk
[2] Acute Intelligence
katarapkokid@hotmail.com

Abstract. The detection and prevention of computer security breaches is made very difficult by the continual development of sophisticated and innovative attacks. When a new threat is discovered, defenders must work hard and fast to understand its behaviour and develop effective counter-measures to avoid potentially serious consequences. The Cyber Threat Alliance estimates that, last year alone, one particularly potent attack known as CryptoWall-3 cheated victims out of *several hundred million dollars* worldwide by forcing them to pay ransoms to recover files it secretly encrypted on their machines [6]. At the time of writing, an improved attack called CryptoWall-4 is now grabbing headlines and wreaking havoc around the globe. This paper proposes an ILP method to assist experts in detecting and analysing cyber-attacks using data logs acquired by an eaves-dropping device placed in the host computer's network to monitor various aspects of its communications traffic. Using CryptoWall-4 as a proof-of-principle case study, we show how ILP can be used to interactively learn rules describing malware behaviour that are comparable to those hand-crafted by a human expert.

Key words: Cyber-Attack, Ransomware, CryptoWall, ILP

1 Introduction

Defending computer systems from attack is a daunting task for even the most expert humans. Defenders must attempt to prevent all attacks while attackers only need to get lucky once. The asymmetries and volumes of data are against the defenders: nearly all computer interactions are benign. Traditional classification machine learning approaches with such skewed classes overgeneralise and produce models amounting to "all interactions are benign". Furthermore, the asymmetries in the costs of making errors means that machine-learned models predicting false-positive attacks rapidly lose the trust of the defenders.

Cybercrime is increasingly industrialised, with professional software engineering teams building the technology to exploit abundant weaknesses in computer security. New attacks are being released all the time – some relying on vulnerabilities in computer systems, others simply preying on the computer user.

Ransomware is a form of phishing attack where a 'victim' is tricked into installing malware that encrypts files that the user has permission to write to (including networked file shares).

Malware defences come in two main forms: *Endpoint security* software that runs on the computer being defended; and *Network security* systems that eavesdrop on the network interactions between computer systems. Endpoint security programs can make detailed local observations about individual user behaviour, including operating system calls. While network security devices do not observe details of processes on the endpoints, they do observe the interaction of many endpoints simultaneously – often hundreds of thousands of connections between source and destination computer systems. These network traffic sniffer devices are known as *N*etwork *I*ntrusion *D*etection *S*ystems (NIDS).

Today, nearly all malware is transferred via the network (e.g. as email attachments or 'accidental' web downloads). The full network interactions of such can be observed by a NIDS. In addition to NIDS log files, defenders have at their disposal a plethora of syndicated threat intelligence information (e.g. known malware internet domains), human expertise, and support communities (e.g. online forums). We do not seek to replace the defender with a machine – we wish to have them join forces in a way that amplifies their power. The objective of this work is primarily to assist a human defender in *analysing* an attack rather than to automatically induce detection signatures of attacks.

The paper is organised as follows. Section 2 introduces ransomware and the infamous CryptoWall-4 example. Section 3 outlines an ongoing experiment to induce the understanding of the workings of the ransomware. Finally, our preliminary findings and plans for future work are described in section 4.

2 Ransomware and CryptoWall

Ransomware, has been defined as "... a cryptovirology attack carried out using covertly installed malware that encrypts the victim's files and then requests a ransom payment in return for the decryption key that is needed to recover the encrypted files. Thus, ransomware is an access-denial type of attack that prevents legitimate users from accessing files since it is intractable to decrypt the files without the decryption key..." [5].

In late 2015 a particular embodiment of ransomware came to the attention of cyber-defenders. The original unfolding of how the behaviour of CryptoWall-4 was worked out is described in detail on an online forum [3]. This section merely outlines the basic concepts required to understand its operation.

Attack Outline: The CryptoWall attack works as follows.

1. An unsuspecting victim is phished. This has many forms but often occurs via an attachment to an email that the user opens. Typically this contains a URI to a piece of (sometimes obfuscated) JavaScript code known as a **Dropper**.
2. The Dropper (executing in the victim's browser, with the victim's privileges) contacts one or more remote computers hosting the CryptoWall malware executable file. Multiple malware servers are tried as the attacker does not want a single point of failure (see listing 1). Sometimes the names/IP addresses of

```
# VictimIP / Port  AttackerIP / Port  HostNameContacted  Resource
192.168.122.163 49184 103.21.59.9 80 shrisaisales.in /ZUQce4.php?m=egw08th5kll
192.168.122.163 49185 173.237.136.250 80 myshop.lk /6872VF.php?m=egw08th5kll
192.168.122.163 49186 195.208.1.122 80 frc conf.com /o51qYV.php?w=egw08th5kll
192.168.122.163 49187 103.21.59.9 80 shrisaisales.in /ZUQce4.php?f=okm0ua6s71c58
192.168.122.163 49188 173.237.136.250 80 myshop.lk /6872VF.php?x=okm0ua6s71c58
192.168.122.163 49189 195.208.1.122 80 frc conf.com /o51qYV.php?u=okm0ua6s71c58
192.168.122.163 49190 103.21.59.9 80 shrisaisales.in /ZUQce4.php?r=5jjh2t0np4
192.168.122.163 49191 173.237.136.250 80 myshop.lk /6872VF.php?r=5jjh2t0np4
192.168.122.163 49192 195.208.1.122 80 frc conf.com /o51qYV.php?y=5jjh2t0np4
```

Listing 1: Excerpt of consecutive HTTP log entries showing the CryptoWall-4 Dropper 'phoning home' to three locations to retrieve the ransomware code.

malware servers are known to the cyber-defence community, but alternative sites are of course constantly popping up.
3. The malware is immediately installed, executed, and sets to work encrypting many or all files the victim has permission to write to.
4. In a short space of time, the malware has encrypted the victim's files and opens the default browser in the victim's session and retrieves a ransom note web page from malware servers. This note includes payment links to for the victim to send BitCoin (or other digital currency).

The eaves-dropping NIDS is able to observe steps 2 and 4 occurring, but not step 3 as the victim's file encryption happens locally on the infected endpoint.

Capturing NIDS logs: Security defenders analyse the behaviour of malware by simulating the attack from within a *sandbox* computer system and recording the behaviour of the malware. In this way, a NIDS[1] was used to produce logs from the CryptoWall-4 malware. The logs of interest were: `conn.log` summarising connections between source and host; `dns.log` detailing DNS queries; `http.log` detailing HTTP requests/responses; and `files.log` detailing file transfers between source and host machines. We transformed these logs into ground Prolog facts for use in reasoning about the attack (see section 3).

Other Domain Knowledge: External information known as *threat intelligence* adds information. For this attack, we are informed that `shrisaisales.in`, `myshop.lk`, `thegingod.com`, `frcpr.com`, and `adrive62.com` (amongst others) are known hosts for malware software downloads.

3 Experiment using ILP to help understand Ransomware

This section describes a controlled experiment to learn logical rules that help us understand the behaviour of ransomware using the ILP system `ALEPH` [4]. The motivation is to test the hypothesis that *ILP can be used to recover reasonable*

[1] The open source NIDS Bro https://www.bro.org/

rules explaining how ransomware works. We choose to use `ALEPH` because it has an *incremental learning* mode which suits the exploratory nature of understanding attacks in conjunction with a human expert defender.

Raw log data detailing DNS and HTTP requests was obtained from a sandbox computer that was deliberately allowed to become infected by CryptoWall-4. Records were converted to a datalog representation (for which illustrative examples are shown below for the predicates `dns/8` and `http/17`).

```
dns(date(2015,11,5,13,4,44,740), 'CZQ4Zw32jddFeiK8z2',
    ipv4(192,168,122,163), 'shrisaisales.in', 'C_INTERNET',
    'A', 'NOERROR', vector(ipv4(103,21,59,9))).
⋮
http(date(2015,11,5,13,4,45,7), 'CiT3vV2lnFWQC0NIAl',
    ipv4(192,168,122,163), ipv4(103,21,59,9), 'POST',
    'shrisaisales.in', '/ZUQce4.php', 'egw08th5kll', unset,
    'Mozilla/4.0...', 115, 0, 200, vector('F7Ze8e1xZMQcH7MbM8'),
    vector('text/plain'), unset, unset).
```

For convenience, time-stamps are written as `date/7` terms (in a year, month, day, hour, minute, second, millisecond format) with an associated predicate `after/2` to determine if a first given date term is strictly later than a second. Other background predicates include `http_category/2` to determine the HTTP request return code as `success`, `redirection`, `server_error`, etc. Other projection predicates are also included to select particular fields out of raw log records:

```
http_domain_name_parameter(Machine,Domain,Name,Param):-
  http(_,_,Machine,_,_,Domain,Name,Param,_,_,_,_,_,_,_,_,_).
⋮
successful_dns(Time,UID,Machine,Domain,IP):-
  dns(Time,UID,Machine,Domain,'C_INTERNET','A','NOERROR',vector(IP)).
```

Studying the logs by hand showed the malware makes several HTTP requests such that the same parameter is sent to different domains, and different parameters are sent to the same **malware domain**. The malware interacts with 3 malware domains and 2 pay sites. So we used these facts as positive and negative examples, respectively, along with following settings to learn a characterisation of a malware domain in terms of HTTP interactions:

```
malware_domain('shrisaisales.in').
malware_domain('myshop.lk').
malware_domain('frc-conf.com').
not(malware_domain(('3wzn5p2yiumh7akj.partnersinvestpayto.com'))).
not(malware_domain(('3wzn5p2yiumh7akj.marketcryptopartners.com'))).

:-set(clauselength,10). :-set(depth,1000). :-set(i,3).
:-mode(1, malware_domain(+domain)). % modeh
:-mode(*, http_domain_name_parameter(-machine,+domain,-name,-parameter)).
:-mode(*, http_domain_name_parameter(+machine,-domain,-name,+parameter)).
:-mode(*, +name \= +name). :-mode(*, +parameter \= +parameter).
:-determination(malware_domain/1, http_domain_name_parameter/4).
:-determination(malware_domain/1, \= /2).
```

With these settings[3] ALEPH's `induce_incremental` was used to learn the following hypothesis, which correctly explains the way the dropper interacts with its potential malware domain servers:

```
malware_domain(Domain):-
  http_domain_name_parameter(Machine,Domain,Name1,Param1),
  http_domain_name_parameter(Machine,Domain,Name2,Param1),
  http_domain_name_parameter(Machine,Domain,Name1,Param2),
  Name1 \= Name2, Param1 \= Param2.
```

After adding this hypothesis to our theory, we continued using ALEPH interactively with the following *existing* mode declarations (where we have omitted the determinations to save space)[4] to learn a definition of a **malware fetch**:

```
:-mode(1, malware_fetch(+time,+machine,+domain)). % modeh
:-mode(*, successful_dns(-time,-uid,+machine,+domain,-ip)).
:-mode(*, malware_domain(+domain)).
:-mode(*, http(+time,-http_id,+machine,-ip,#http_command,
           +domain,-uri_name,-uri_parameter,-referer,
           -user_agent,-size,-size,-response_code,
           -malware_request_uid,#mime_type,
           -malware_response_uid,#mime_type)).
:-mode(*, after(+time,+time)).
:-mode(*, http_category(+response_code,#category)).
:-mode(*, +ip=+ip).
:-mode(*, gt1000(+size)).
```

Given a single positive example that we obtained by hand from the logs, ALEPH constructs the following Bottom Clause [7]:

```
malware_fetch(A,B,C):-
   successful_dns(D,E,B,C,F), malware_domain(C),
   http(A,G,B,F,'POST',C,H,I,J,K,L,M,N,O,
     vector('text/plain'),P,vector('text/plain')),
   after(A,D), http_category(N,success), gt1000(M).
```

Using further examples and constraints created by a systematic Skolemisation-based process that we have developed in order to allow the rebuttal of overgeneral hypotheses through exploratory interactions with the logs, ALEPH learns the rule shown below – which was found to subsume one crafted by a human expert, and which correctly states that a malware fetch involves the return of a large file from an HTTP request to a malware domain:

```
malware_fetch(A,B,C):-
   malware_domain(C), http(A,D,B,E,'POST',C,F,G,H,I,J,K,L,M,
   vector('text/plain'),N,vector('text/plain')), gt1000(K).
```

[3] Refer to the ALEPH manual [4] for an explanation of the notation used.
[4] Note that predicate `gt1000/1` is true if its argument is greater than 1000.

4 Conclusion and Future Work

Encoding a computer security domain has previously been successful for an end point setting. In [2] ILP was applied to learning to detect buffer overflow attack construction strategies [1]. Our current work differs in that, firstly, it focuses on information that is observed on the computer *network* and, secondly, it aims to help humans *understand* specific attacks rather than *detect* them.

Although we have achieved a working proof-of-principle reconstruction of known rules, we are working on several extensions of this study. In particular, we have acquired several months' worth of network data from a small business network and we aim to incorporate data from those much more extensive logs into our learning process. We are also planning to experiment with using event calculi to reason about more complex temporal cyber-attack signatures.

Our work has already revealed a difficulty in designing ILP systems that co-operate with humans in this task. In a sense there is a trade-off between human-readability (where large arity predicates are preferred to allow humans to see patterns more easily in the data) vs. machine-learnability (where small arity predicates are preferred to avoid a proliferation of somewhat arbitrary place-markers in the language bias). We will need to investigate automated ways of setting the mode-declarations to work effectively with our larger logs.

We are in the process of building our interactive rebuttal methodology into `ALEPH`'s incremental learning menu because we believe it has some advantages over the meta-constraints that `ALEPH` currently uses to eliminate "overgeneral" hypotheses. In this way we hope to provide a more powerful and user-friendly tool to help humans carry out interactive ILP on data-rich domains.

Acknowledgements

This work is supported by the EPSRC Summer Bursary Scheme. We also thank Paul Byrne for providing the detailed ransomware logs.

References

1. "Aleph One". *Smashing The Stack For Fun And Profit*. Phrack 49, 1996.
2. S. Moyle and J. Heasman. *Machine Learning to Detect Intrusion Strategies*. KES 2003, LNCS 2773:371-378, 2003
3. BleepingComputer.com. *CryptoWall 4.0: Help_Your_Files Ransomware Support Topic*. `http://www.bleepingcomputer.com/forums/t/595215/cryptowall-40-help-your-files-ransomware-support-topic/`, November 2015.
4. A. Srinivasan. *The Aleph Manual*. `http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/`, 2007.
5. T. Simonite. *Holding Data Hostage: The Perfect Internet Crime? Ransomware (Scareware)*. MIT Technology Review. February 2015.
6. Cyber Threat Alliance. *Lucrative Ransomware Attacks: Analysis of the CryptoWall Version 3 Threat*. `http://cyberthreatalliance.org/cryptowall-report.pdf`, October 2015.
7. S. Muggleton. *Inverse Entailment and Progol*, New Generation Computing 13(3-4):245–286, 1995.