The 2nd WOMBAT Workshop

Worldwide Observatory of Malicious Behaviors and Attack Threats

September 22-23, 2009
FP7-ICT-216026-WOMBAT

The 2nd WOMBAT Workshop

Worldwide Observatory of Malicious Behaviors and Attack Threats

September 22-23, 2009
FP7-ICT-216026-WOMBAT

Presentations

- *"Introduction to the WOMBAT datasets"*(M. Dacier).
- *"The WOMBAT WAPI: idea, implementation and use"* (C. Leita).
- *"The SHELIA and HSN client honeypot datasets"* (H. Bos; P. Kijewski)..
- *"Clustering malware with ANUBIS and SGNET and interaction with the WAPI"* (P. M. Comparetti).
- Demo session: *"Using WOMBAT APIs on Real-World Tasks: Investigation of a Banking Fraud"* (P. M. Comparetti, F. Maggi)
- Demo session: *"Using WOMBAT APIs on Real-World Tasks: Monitoring of Own Networks"* (C. Leita, H. Bos)

FOREWORD

This volume collects the presentations and handouts of the 2nd WOMBAT Project Workshop, held on September 22-23, 2009 in St. Malo.

The workshop was organized by the European Project WOMBAT (Worldwide Observatory on Malicious Behavior and Attack Threats), funded by the European Union under the 7th Framework Programme (FP7) for Research of the EU. WOMBAT aims at providing new means for understanding the existing and emerging threats that are targeting the Internet economy and the net citizens. To accomplish this, the WOMBAT project goals deal with gathering security related raw data, enrichment of this input by means of various analysis techniques, and root cause identification and understanding of the phenomena.

This year's workshop focuses on the introduction of early results of the project, and in particular on the Wombat APIs or WAPI, a set of API developed by the project partners to allow integrated access to different attack dataset.

The aim of the workshop was to give participants a first-hand experience on how the WAPIs help the analyst and the researcher in investigating new phenomena.

The demos and presentations were prepared thanks to the collective effort of the project partners: France Telecom, Hispasec, Politecnico di Milano, Technical University of Vienna, Institut Eurecom, FORTH-ICS, Symantec Corporation, Vrije Universiteit Amsterdam, Institute for Infocomm Research, NASK.

The WOMBAT Consortium

Introduction to the WOMBAT datasets

M. Dacier
Senior Director, Symantec Research Labs

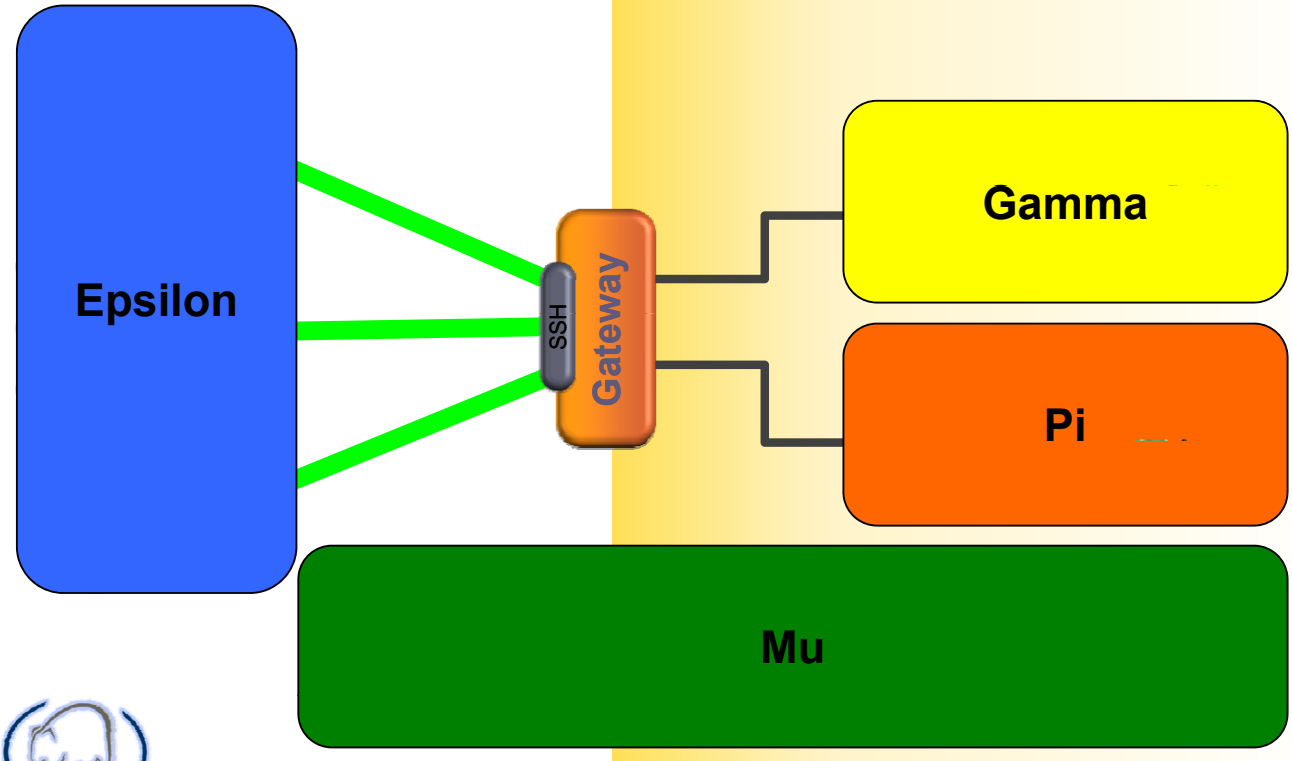
Wapi Enabled Datasets

- Malware Analysis Tools:
  - Anubis
  - Virustotal
- Server side honeypots:
  - SGNET
- Client side honeypots:
  - Honeyspider
  - HARMUR
  - Wepawet
  - Shelia
- Aggregation Platform
  - FORTH
- Others:
  - Bluebat

Malware Analysis Tools

- Anubis:
  - Monitored sandbox
  - Live access to the Internet
  - Public interface for free submission of files
  - Several related publications
  - Large amount of daily submissions
- Virustotal:
  - Reporting of scan results from XX antiviruses
  - Public interface for free submission of files
  - Several related publications
  - Very large amount of daily submissions

SGNET: server side honeypot + analysis tools



Client Side Honeypots

- Honeyspider
  - Crawler
  - Sophisticated set of detection techniques
- Wepawet:
  - Not a crawler
  - Focused analysis of scripts on a given URL.
- Shelia
  - Email + URLs inspection
  - Instrumented high interaction honeyclient
- HARMUR
  - Enrichment framework for Symantec's internal crawler
  - Historical archiving

Others

- FORTH
  - Aggregation platform for caching, efficiency and anonymisation
  - WAPI interface to non wapi-enabled services (e.g. whois)
- Bluebat
  - Bluetooth based honeypot
  - Ongoing effort to assess severity of these infection vectors.

Confidence in a connected world.

WAPI: The WOMBAT API

Corrado Leita, Symantec Research
corrado_leita@symantec.com

Introduction

Motivation

- For the **data provider**
  - Control which content to present to the clients, and how
  - Enrich or modify the dataset without needing to modify all the clients

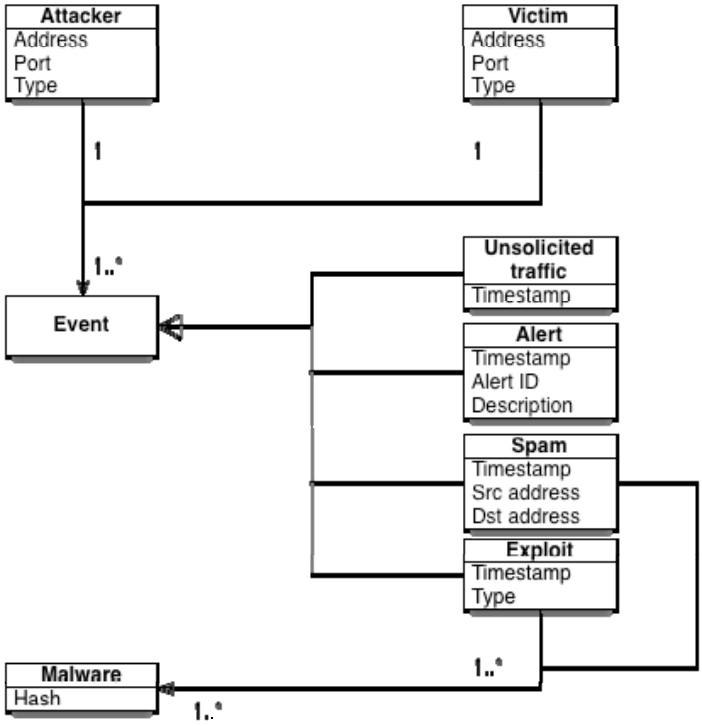
- For the **client**
  - Need for a common “language” to request data from the datasets
    - HTTP submission + XML reports for ANUBIS
    - Email submission + email reports for VirusTotal
    - …
  - Need for programming primitives to easily retrieve information on the fly while performing analysis tasks

WAPI

- What the WAPI **is**:
  - A SOAP-based API to easily allow a client to traverse a hierarchy of objects, characterized by **attributes**, **methods** and **references**.

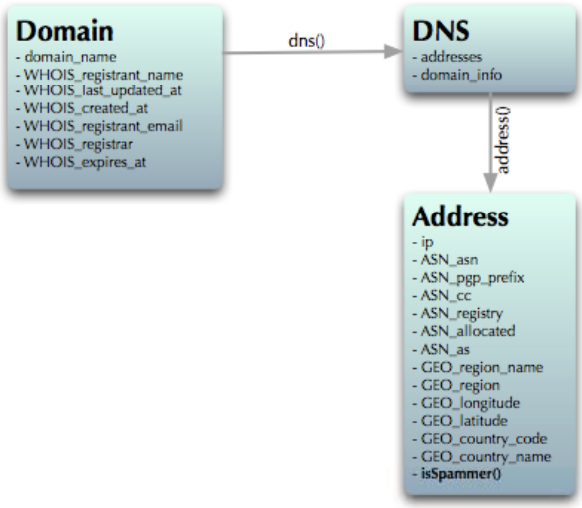
- What the WAPI **is not**:
  - An onthology
  - A detailed specification of how a security related dataset should look like
  - Language-specific. Reference implementation in python, but accessible from any programming language offering a SOAP library (C,C++,Java,PHP,…)

The basic skeleton of a WAPI dataset



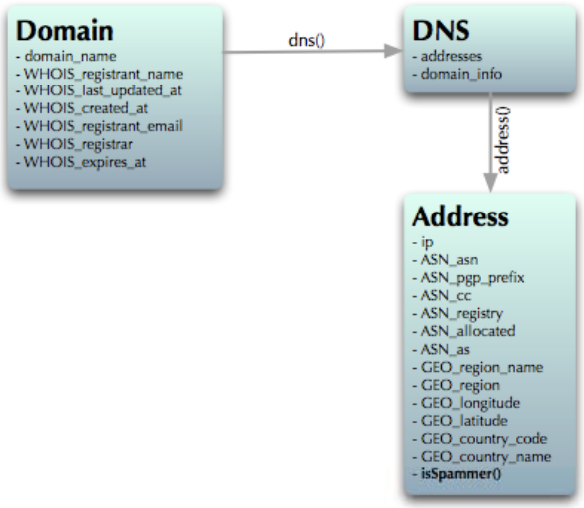
An example

- **Objects:** the dataset "concepts"
- **Object instances:** a specific instance of an object, characterized by each class name and by an identifier.
  - For instance:
    - Address.193.55.112.70
  - But also:
    - Address.143252
- **Attributes:** information on an object instance.
- **Methods:** more expensive calculations on an object
  - Example: check whether a given IP address is blacklisted



An example

- **References:** “special methods” that return lists of object instances.
  - The edges of the graph
  - Allow the user to explore the different objects
  - Example: the list of DNS relations that are known for each domain name



The WAPI in practice

- Set of SOAP calls to query a dataset about its capabilities
  - Which objects are offered by the dataset?
  - What are the methods for a given object?
  - What are the references for a given object?
  - What are the attributes and the attribute values for a specific object instance?
  - Does an object instance exist in the dataset?
  - Call a method
  - Follow a reference
  - Get the documentation for an object

WAPI primitives

- get_objects()
- get_methods(object)
- get_references(object)
- get_attributes(object,identifier)
- exists(object,identifier)
- call_method(object,identifier,method,**kw)
- follow_reference(object,identifier,reference,**kw)

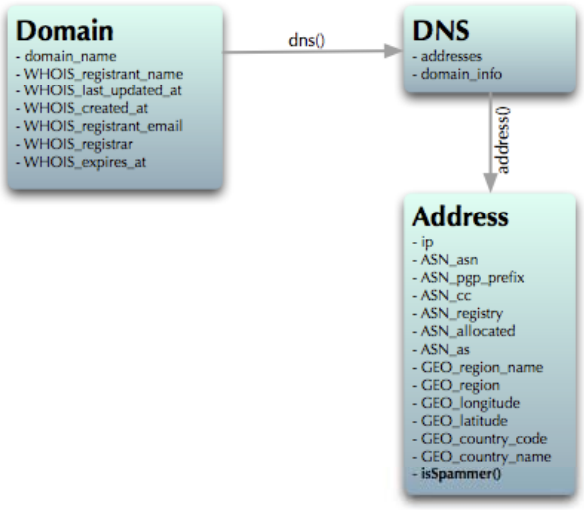
- get_documentation(object)

The Dataset object

- Through references, the client can explore the different WAPI objects as a traversal of an oriented graph
  - Domain name and whois information -> DNS relations -> IP addresses and geolocation
- An object instance cannot often be instantiated directly: the identifer can be an opaque value understandable only to the dataset (e.g. row ID)
- The traversal always starts for a single starting point, the **dataset** object. Only one dataset object instance exists in each WAPI dataset.

The FORTH dataset object

- For instance, in the FORTH dataset the dataset object offers the following references:
  - address(ip_addr) -> Address
  - domain(domain) -> Domain



Using the WAPI

The reference implementation

- Python implementation that takes advantage of the python reflection/introspection features to completely hide the SOAP interaction
  - To the dataset maintainer
  - To the client
- Takes advantage of:
  - SOAPpy (http://pywebsvcs.sourceforge.net/) for the SOAP interaction
  - M2Crypto (http://chandlerproject.org/bin/view/Projects/MeTooCrypto) for SSL encryption and client authentication

Authentication

- How to control who gets access to a WAPI dataset?
- SSL-based authentication
  - Each dataset is associated to a Certification Authority
  - The dataset CA signs the server certificate and each client certificate
  - A dataset allows interaction only to clients providing an SSL certificate signed by the dataset CA
- Current implementation: one single privilege levels
- Multiple privilege levels will be supported in the future

Using the WAPI client

- Python wrappers that offer each WAPI object as a normal python object.
- Object instantiation:
  - Check that the object exists
  - Retrieve the list of methods and references
  - Retrieve the attributes and their values
  - Retrieve the documentation
- Method call:
  - Calls the method over WAPI
- Reference call:
  - Returns a list of wrappers for the returned WAPI objects

Using the WAPI client

```
> f=virustotal.get_file(md5="3228c641929bb40475c44a26bda8531a")[0]
> print f.first_seen
'2009-05-27 15:38:11'
> an=f.get_first_analysis()
> print an.av_positives_report
{'GData': ['Exploit.PDF-JS.Gen', '19', '2009.05.27'], 'AntiVir': ['HEUR/HTML.Malware',
    '7.9.0.168', '2009.05.27'], 'McAfee-GW-Edition': ['Heuristic.HTML.Malware', '6.7.6',
    '2009.05.27'], 'Sophos': ['Troj/PDFJs-AX', '4.42.0', '2009.05.27'], 'ClamAV':
    ['Exploit.PDF-63', '0.94.1', '2009.05.27'], 'Authentium': ['PDF/Obfusc.B!Camelot',
    '5.1.2.4', '2009.05.27'], 'BitDefender': ['Exploit.PDF-JS.Gen', '7.2', '2009.05.27'],
    'Sunbelt': ['Exploit.PDF-JS.Gen (v)', '3.2.1858.2', '2009.05.27'], 'VirusBuster':
    ['JS.Shellcode.AD', '4.6.5.0', '2009.05.26']}
```

How to make a WAPI dataset?

- We have tried to make the task of generating WAPI datasets as easy as possible

- Define the dataset as a collection of python classes following some simple naming conventions
  - Extend wapi.object.WObject
  - Define the type name as a static WTYPE
  - Custom constructor WObject.init()
  - Attributes have prefix W_
  - Methods have prefix WM_
  - References have prefix WR_ and return list of WObjects

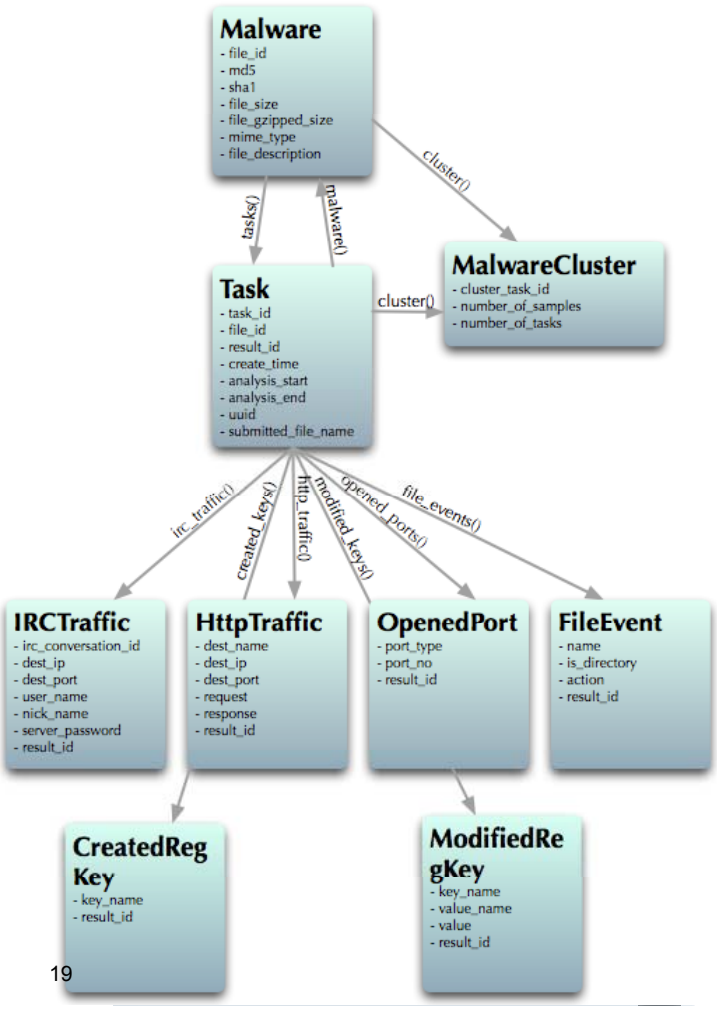
Example

```
class Address(WObject):
    """
    This is the object documentation
    """
    def init(self):
        #query your database to retrieve information
        address,location=query_db(self.W_identifier)

        #define the attributes
        self.W_address=address
        self.W_location=location

    def WM_isspammer(self):
        """
        This is the method documentation
        """
        #query the database to check whether it's a blacklisted IP
        return isblacklisted(self.W_identifier)
```

The WAPI datasets

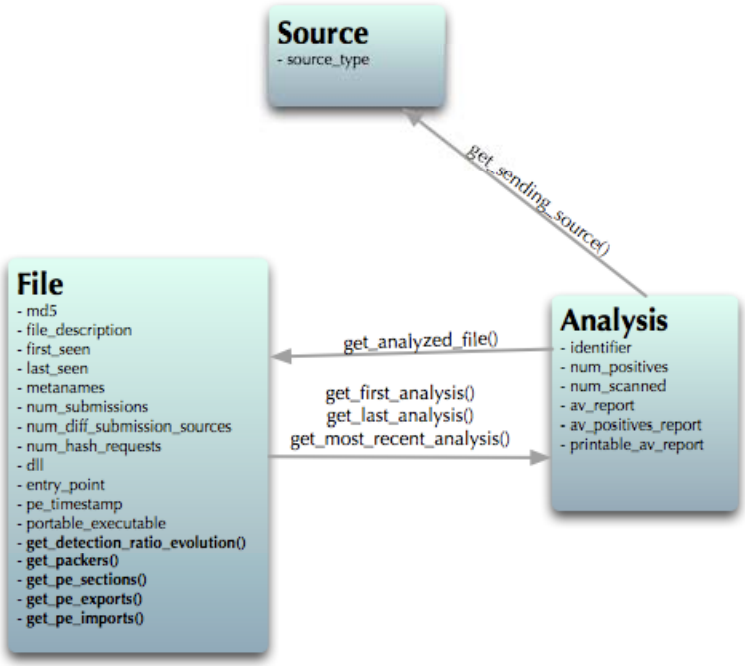
Anubis

- Detailed information on the malware behavioral analysis and the corresponding behavioral clustering
- **Example**: there is a suspicious registry key in my machine. Is any malware known to Anubis performing this action?



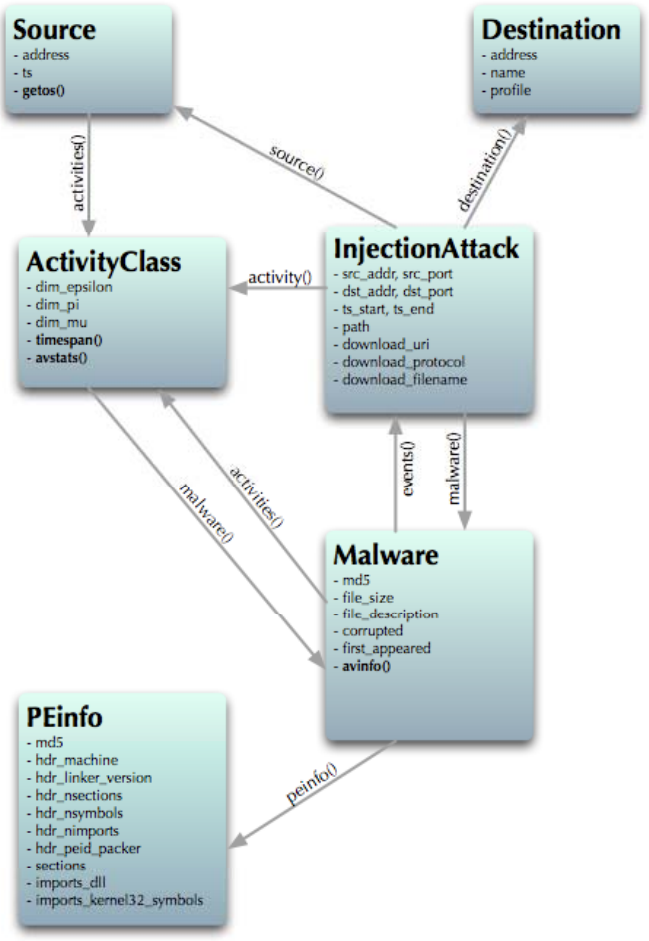
VirusTotal

- AV detection statistics on the malware sample, and much more
- **Example:** is this malware sample recognized by my AV solution? Was it in the past?



SGNET

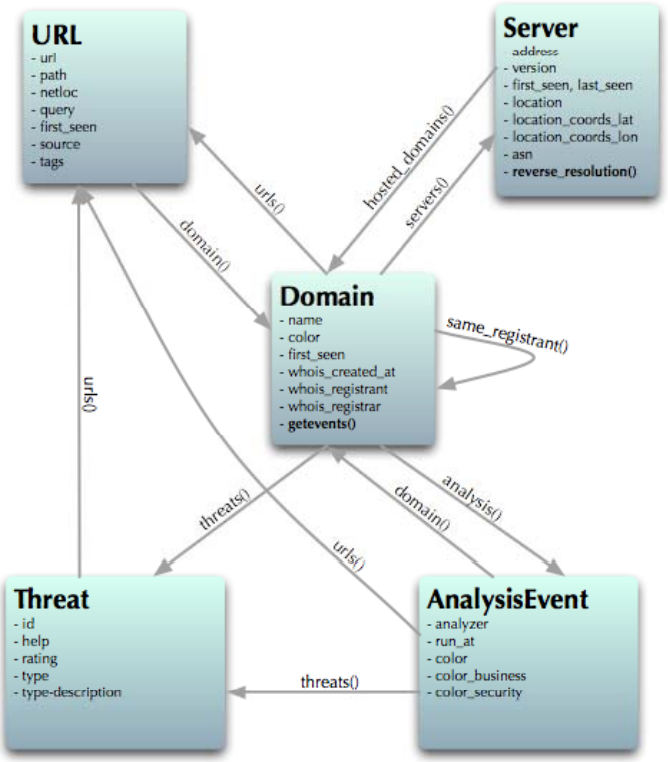
- Allows to explore the characteristics of the code injection attacks observed by the SGNET deployment
- **Example**: have you ever seen this malware? How did it propagate?



HARMUR

- Detailed information on the temporal evolution of malicious websites, on their threats and their location

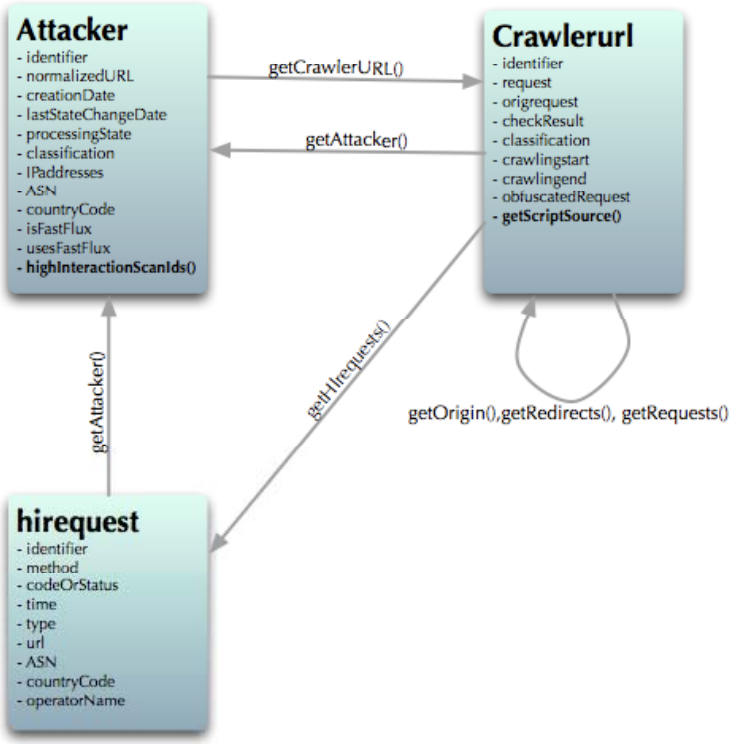
- **Example**: where is this suspicious domain hosted? Is it malicious? Was it moved?



Honeyspider

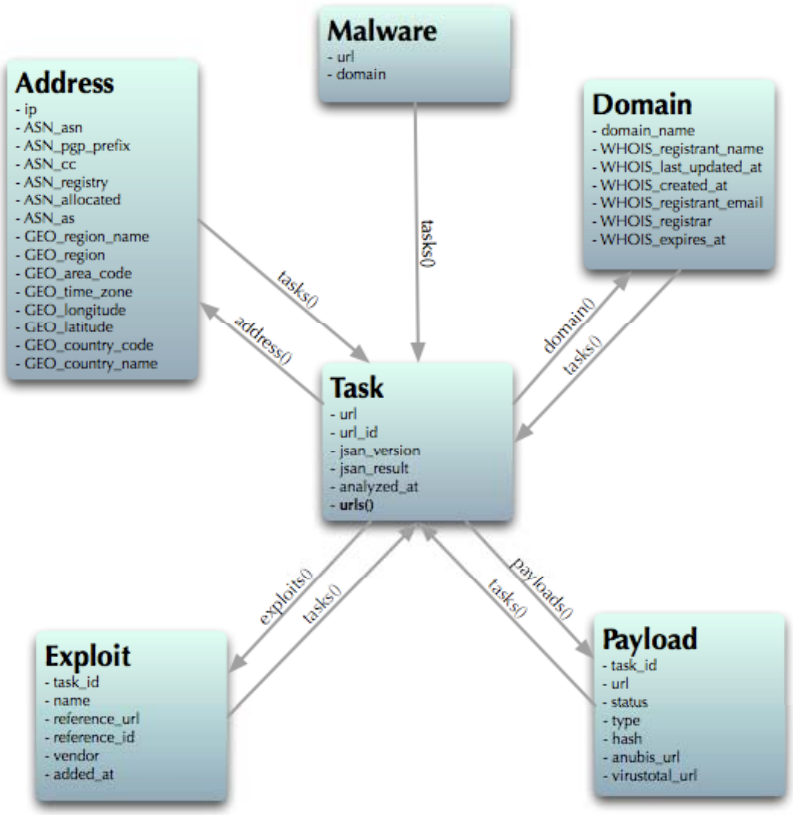
- Allows to retrieve detailed crawling information on each URL submitted to the Honeyspider crawler

- **Example**: what happens in detail when I point my browser towards this specific domain?



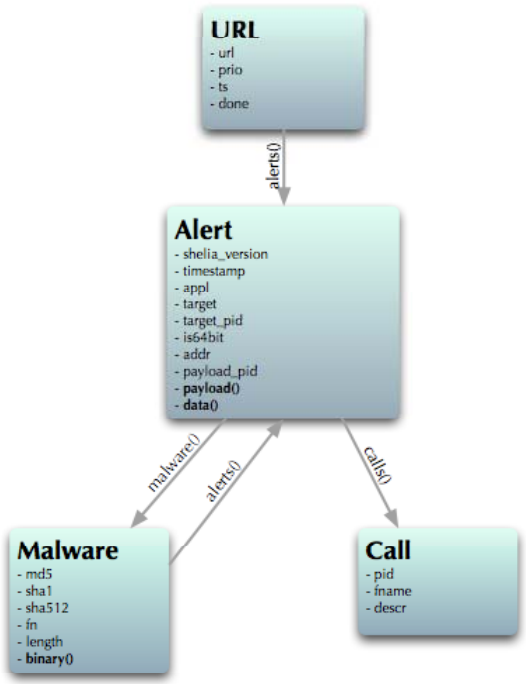
Wepawet

- Detailed exploit information on each analyzed URL
- **Example**: analyze a site and get the exploit characteristics. How many other sites contain this specific exploit?



Shelia

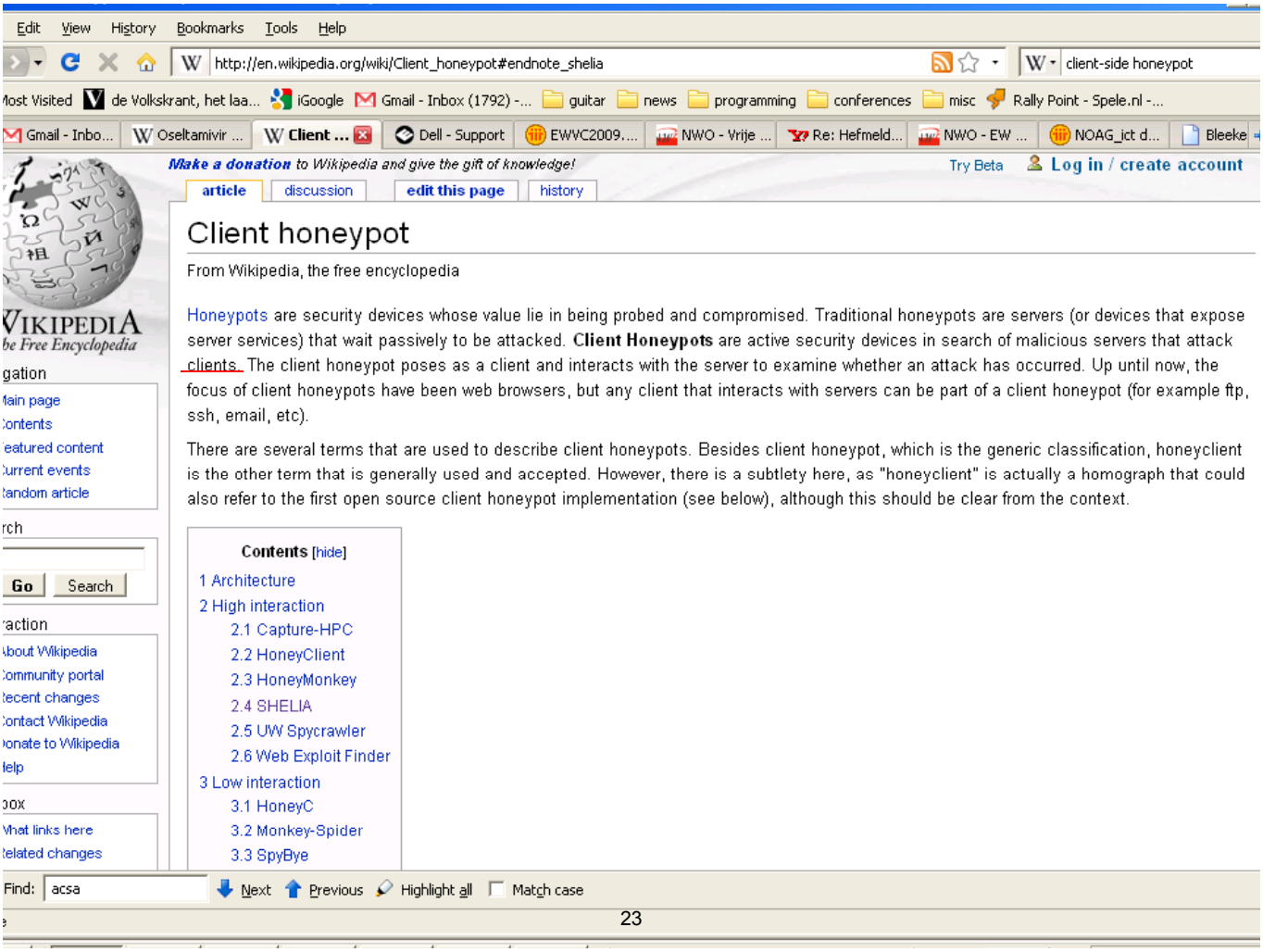
- Allows to retrieve malware samples deriving from browser exploitation as a consequence of visiting a certain URL



Shelia: A Client-Side Honeypot

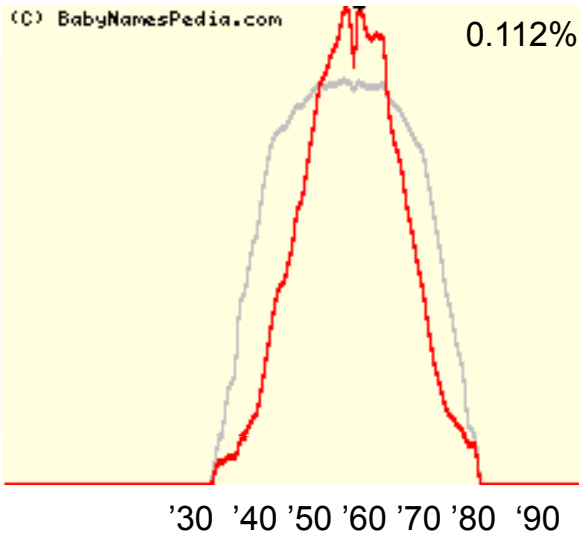
www.cs.vu.nl/~herbertb/misc/shelia

Herbert Bos
Vrije Universiteit Amsterdam



Shelia

- Joan Robert Rocaspana
- Georgios Portokalidis
- Philip Homburg
- Herbert Bos



Latin: "Blind"

"Great, but what's with the name?"

Shelia

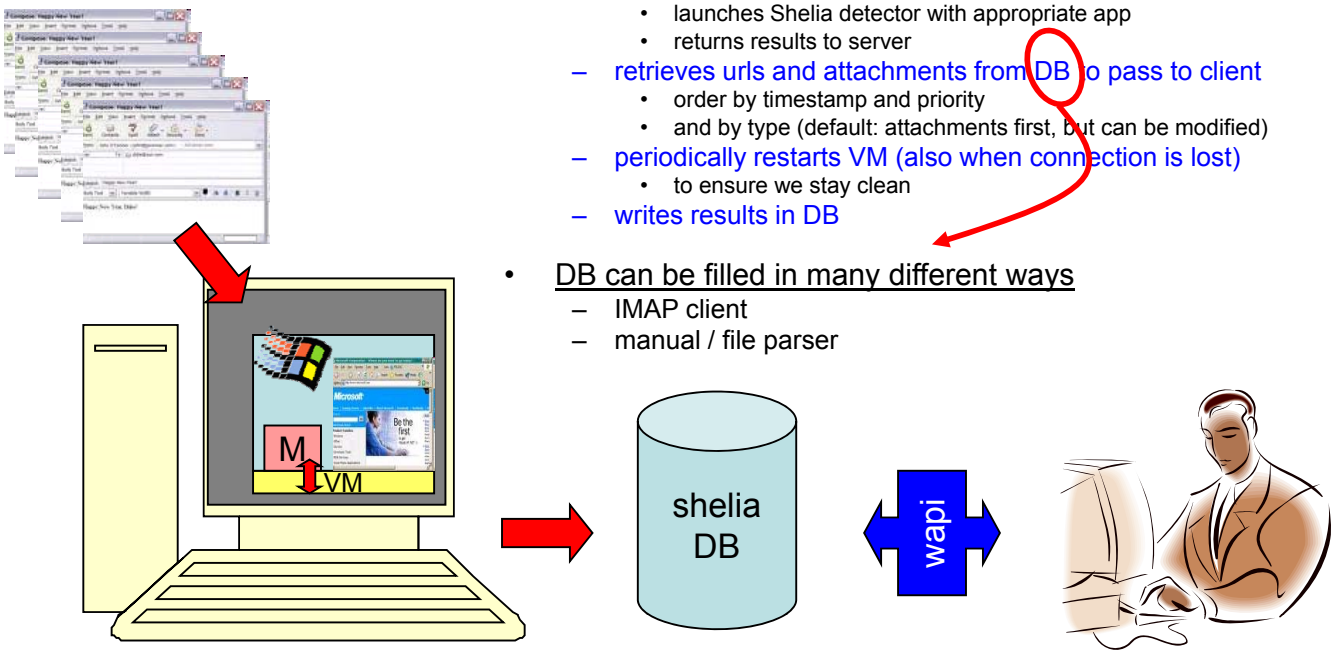
- High-interaction client honeypot
- for Windows
- Goals:
  - no false positives
  - ease of management

3 main phases

- client emulation
  - "blindly follow all links, open all attachments"
- attack detection
  - "did we see any sensitive actions from memory areas where there should not be code?"
- log
  - "If (attack) store as much info as possible"

Big picture

- shelia mgmt server on host
  - starts VM with shelia mgmt client
    - client listens on socket for target objects (URLs and attachments)
    - launches Shelia detector with appropriate app
    - returns results to server
  - retrieves urls and attachments from DB to pass to client
    - order by timestamp and priority
    - and by type (default: attachments first, but can be modified)
  - periodically restarts VM (also when connection is lost)
    - to ensure we stay clean
  - writes results in DB
- DB can be filled in many different ways
  - IMAP client
  - manual / file parser

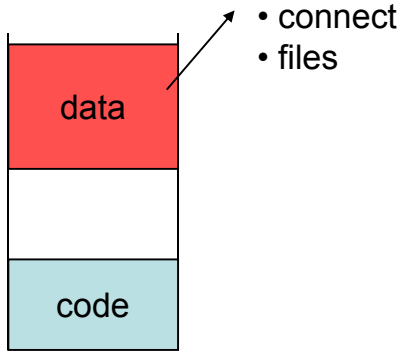


Attack detection

"demo"

Avoid false positives

- we do not treat the system as a black box
  - e.g., we do not look at FS changes
- instead, we explicitly detect badness
  - change the registry
  - open network connections
  - write or execute files



if called from area not supposed to contain code!

Log on alert

- execute payload (shellcode)
- try to download the malware
- log in DB

Alerts
- timestamp
- target application
- target URL/object
- payload (shellcode)

Attachment
- file name
- md5
- data

Malware
- binary
- MD5, sha-1, sha256
- filename of malware on FS

Calls
- function name
- function description
- arguments

URLs
- timestamp entered
- processed

WAPI

WAPI

- let us see how it works

```
coogee: ../src>PYTHONPATH=../../SOAPpy/ python wapi_client.py -c democonf --no-ipython


__          __     _____ _____
\ \        / /\   |  __ \_   _|
 \ \  /\  / /  \  | |__) || |
  \ \/  \/ / /\ \ |  ___/ | |
   \  /\  / ____ \| |    _| |_
    \/  \/_/    \_\_|   |_____|
      The WOMBAT API (version 1.0)

Connecting to the WAPI datasets
-> harmur :  success
-> virustotal :  success
-> wepawet :  success
-> anubis :  success
-> hsn :  success
-> shelia :  success
-> sgnet :  success
-> forth :  success
-> utils :  success
You are connected to 9 WAPI datasets!
Welcome to the wombat wapi client
>>>
```

WAPI

- let us see how it works

```
>>> whelp(shelia)
<wobject 'dataset.shelia'>
 Shelia dataset.
 <attributes>
  identifier:shelia
 <methods>
 <references>
  alert(alert_id) : Returns the WAPI alert object: with a given alert_id (shelia internal key)
  alerts() : Returns all WAPI alert objects in the database:
  alerts_by_target(target) : Returns the WAPI alert objects: that match a URL/filename (SQL) patte
  malware_by_filename(fn) : Returns the WAPI malware objects: that match a filename (SQL) pattern
  malware_by_md5(md5) : Returns the WAPI malware objects: with a given md5
  malware_by_sha1(sha1) : Returns the WAPI malware objects: with a given sha1
  malware_by_sha256(sha256) : Returns the WAPI malware objects: with a given sha256
  urls(url) : Returns the WAPI url objects: that match a URL (SQL) pattern
>>>
```

WAPI

- find alerts caused by target containing ‘http’

```
>>> shelia.alerts_by_target(target="http")
[<shelia.alert object id '2'>, <shelia.alert object id '5'>, <shelia.alert object id '6'>,
<shelia.alert object id '7'>, <shelia.alert object id '8'>, <shelia.alert object id '9'>,
<shelia.alert object id '10'>, <shelia.alert object id '11'>, <shelia.alert object id '15'>,
<shelia.alert object id '17'>, <shelia.alert object id '18'>, <shelia.alert object id '19'>,
<shelia.alert object id '20'>, <shelia.alert object id '21'>, <shelia.alert object id '22'>,
<shelia.alert object id '23'>, <shelia.alert object id '24'>, <shelia.alert object id '25'>,
<shelia.alert object id '26'>, <shelia.alert object id '27'>, <shelia.alert object id '28'>,
<shelia.alert object id '29'>, <shelia.alert object id '30'>, <shelia.alert object id '31'>,
<shelia.alert object id '32'>, <shelia.alert object id '33'>, <shelia.alert object id '34'>,
<shelia.alert object id '35'>, <shelia.alert object id '36'>, <shelia.alert object id '37'>,
<shelia.alert object id '38'>, <shelia.alert object id '39'>, <shelia.alert object id '40'>,
<shelia.alert object id '41'>, <shelia.alert object id '42'>, <shelia.alert object id '43'>,
<shelia.alert object id '44'>, <shelia.alert object id '45'>, <shelia.alert object id '46'>,
<shelia.alert object id '47'>, <shelia.alert object id '48'>, <shelia.alert object id '49'>,
<shelia.alert object id '50'>, <shelia.alert object id '51'>, <shelia.alert object id '52'>,
<shelia.alert object id '53'>, <shelia.alert object id '54'>, <shelia.alert object id '55'>,
<shelia.alert object id '57'>, <shelia.alert object id '62'>, <shelia.alert object id '63'>,
<shelia.alert object id '64'>, <shelia.alert object id '65'>, <shelia.alert object id '66'>]
>>>
```

WAPI

- let us pick one

```
>>> shelia.alerts_by_target(target="http")[8].dump()
<wobject 'alert.15'>
 An alert raised by Shelia
 <attributes>
  addr:202571238
  appl:C:\Program Files\Internet Explorer\iexplore.exe
  identifier:15
  is64bit:0
  payload_pid:1184
  shelia_version:1.2.1
  target:http://azadars.com
  target_pid:1184
  timestamp:2009-08-24 10:51:53
 <methods>
  data()
  payload()
 <references>
  calls() : Returns the WAPI call objects associated with the alert
  malware() : Returns the WAPI malware objects associated with the alert
>>>
```

WAPI

- what about the malware?

```
>>> shelia.alerts_by_target(target="http")[8].malware()
[<shelia.malware object id '1'>]
>>> shelia.alerts_by_target(target="http")[8].malware()[0].dump()
<wobject 'malware.1'>
 Malware object.
 <attributes>
  fn:C:\DOCUME~1\user\LOCALS~1\Temp\update.exe
  identifier:1
  length:28160
  md5:00b23b08657a153fcde4e0891e2484bb
  sha1:522674387e1a8e2d3ab5f7c11ecd9db7e5904dc4
  sha256:b851756487f055bb746cae506e5ffc016f88a07177ab7bfc5b8be7208cbc8156
 <methods>
  binary() : Returns the actual malware
 <references>
  alerts() : Returns the WAPI alerts objects associated with the malware
```

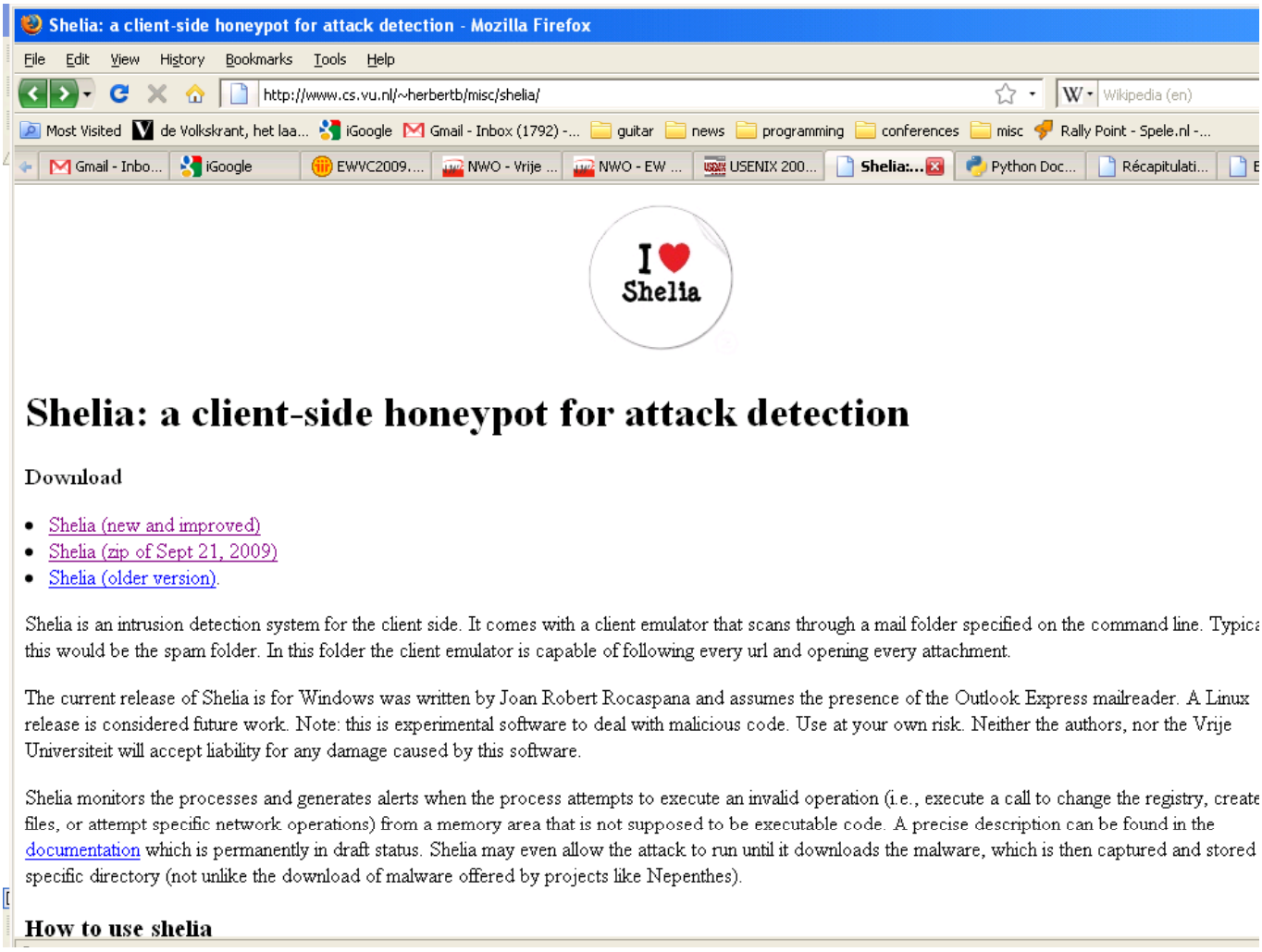
Conclusion

- Shelia works. It is different. It is available.

  Download from:

  www.cs.vu.nl/~herbertb/misc/shelia

- WAPI allows you to combine Shelia data with other sources


Shelia: a client-side honeypot for attack detection - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.cs.vu.nl/~herbertb/misc/shelia/
Wikipedia (en)
Most Visited
de Volkskrant, het laa...
iGoogle
Gmail - Inbox (1792) -...
guitar
news
programming
conferences
misc
Rally Point - Spele.nl -...
Gmail - Inbo...
iGoogle
EWVC2009....
NWO - Vrije ...
NWO - EW ...
USENIX 200...
Shelia:...
Python Doc...
Récapitulati...
I ♥ Shelia
Shelia: a client-side honeypot for attack detection
Download
Shelia (new and improved)
Shelia (zip of Sept 21, 2009)
Shelia (older version).
Shelia is an intrusion detection system for the client side. It comes with a client emulator that scans through a mail folder specified on the command line. Typica
this would be the spam folder. In this folder the client emulator is capable of following every url and opening every attachment.
The current release of Shelia is for Windows was written by Joan Robert Rocaspana and assumes the presence of the Outlook Express mailreader. A Linux
release is considered future work. Note: this is experimental software to deal with malicious code. Use at your own risk. Neither the authors, nor the Vrije
Universiteit will accept liability for any damage caused by this software.
Shelia monitors the processes and generates alerts when the process attempts to execute an invalid operation (i.e., execute a call to change the registry, create
files, or attempt specific network operations) from a memory area that is not supposed to be executable code. A precise description can be found in the
documentation which is permanently in draft status. Shelia may even allow the attack to run until it downloads the malware, which is then captured and stored
specific directory (not unlike the download of malware offered by projects like Nepenthes).
How to use shelia

HoneySpider Network

Overview & WAPI Interface

Piotr Kijewski (NASK/CERT Polska)
Adam Kozakiewicz (NASK)

2nd WOMBAT Workshop, St. Malo, France, 22nd September 2009

Outline

- HSN Introduction
- HSN Architecture
- HSN Low & High Interaction Components
- Mapping WAPI objects to HoneySpider Network
- HSN WAPI Objects
- HSN WAPI Services

The demo uses a limited dataset!
Not everything is available
(URL submission only simulated,
no black/greylists).

HSN Introduction

HoneySpider project – Introduction

> Joint venture between NASK, GOVCERT.NL and SURFnet.

> Development of a complete system, based on low- and high-interaction honeyclient components.

> To detect, identify and describe threats that infect computers through Web browser technology.

Goals

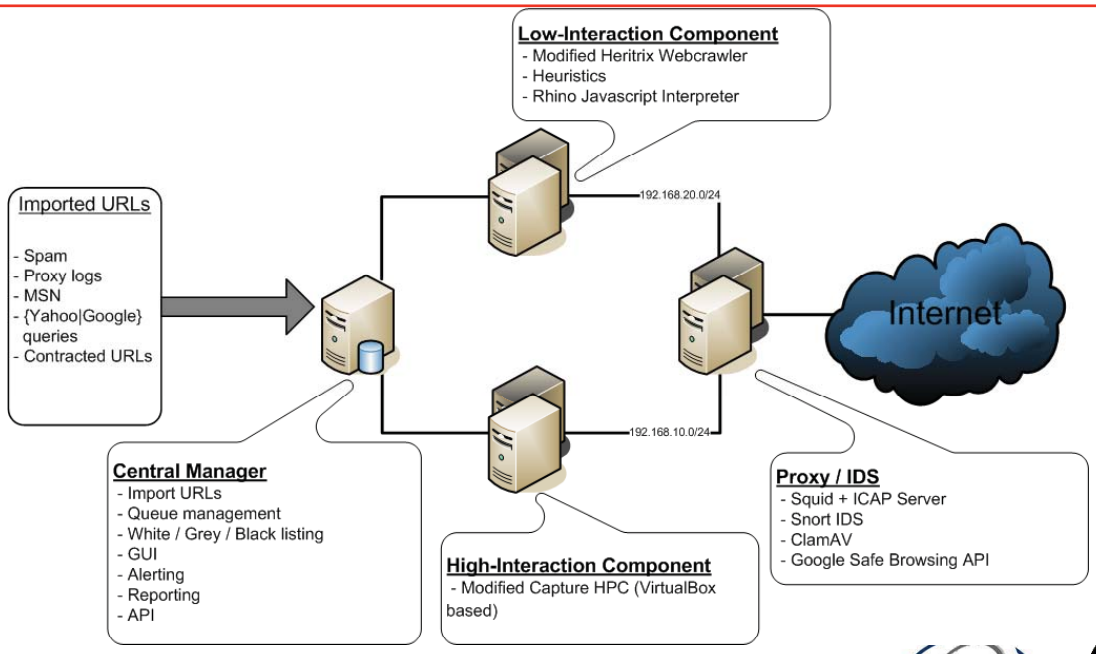
> Build a stable and mature system, capable of processing bulk volume of URL's.

> Detect and identify URL's which serve malicious content.

> Detect, identify and describe threats that infect computers through browser technology, such as:
  – Browser (0)-day exploits
  – Malware offered via drive-by-downloads

HSN Architecture Overview

Architecture



HSN Low Interaction Component

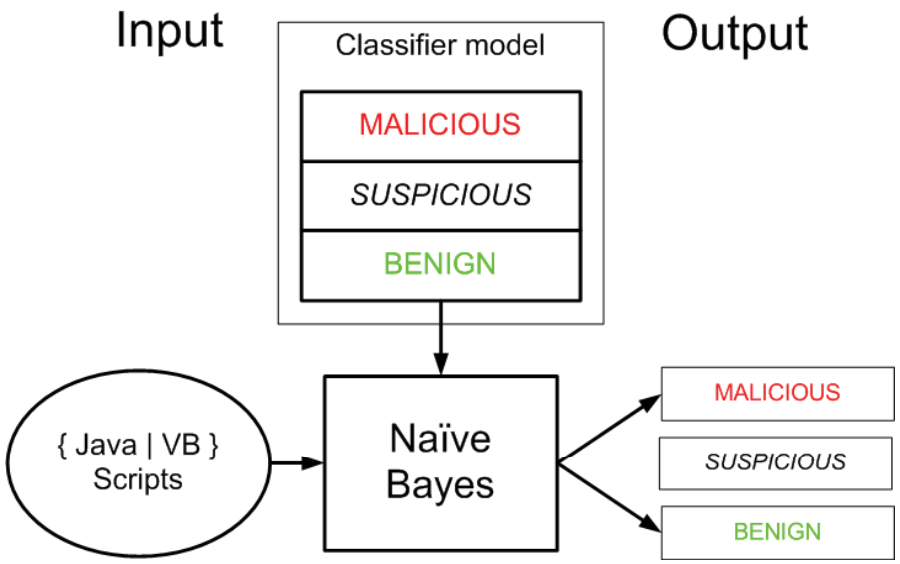
Low interaction component

- Webcrawler (Heritrix)
- Rhino JavaScript interpreter (+ DOM implementation)
- Flash analysis through gnash
- Heuristics (example follows …)
- Fast-flux detection
- Low-Interaction Manager
- Controls & retrieves data from:
  - Webcrawler & Analysers
  - ICAP server (ClamAV/Google SafeBrowsing API)
  - Snort IDS

Example Heuristics

- Approach
  - **Building** classifier models **based on** machine learning **and** data mining-based **techniques for** text classification**.**
- Goal:
  - **Classification of** previously unseen **JavaScript (i.e. assigning them to proper pre-defined categories)**
- Tool of choice:
  - **Weka - Data mining software**
  - **Google n-grams**

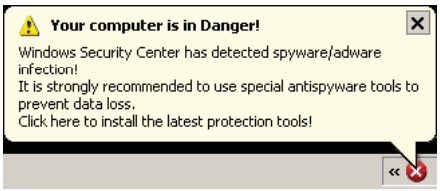
Heuristics - Classifier model



HSN High Interaction Component

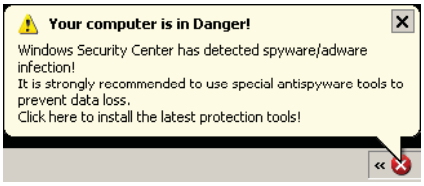
High interaction component (I)

- Based on heavily modified Capture-HPC (VirtualBox)
- Multiple patch levels Microsoft Windows
- IE / Firefox (possibly plugins, like QuickTime & Flash)
- Checks for:
  - Started or terminated processes
  - Filesystem modifications
  - Registry modifications
- High Interaction Manager, controls and retrieves data from:
  - Proxy (Squid)
  - ICAP Server (ClamAV/Google Safebrowsing API)
  - Snort IDS

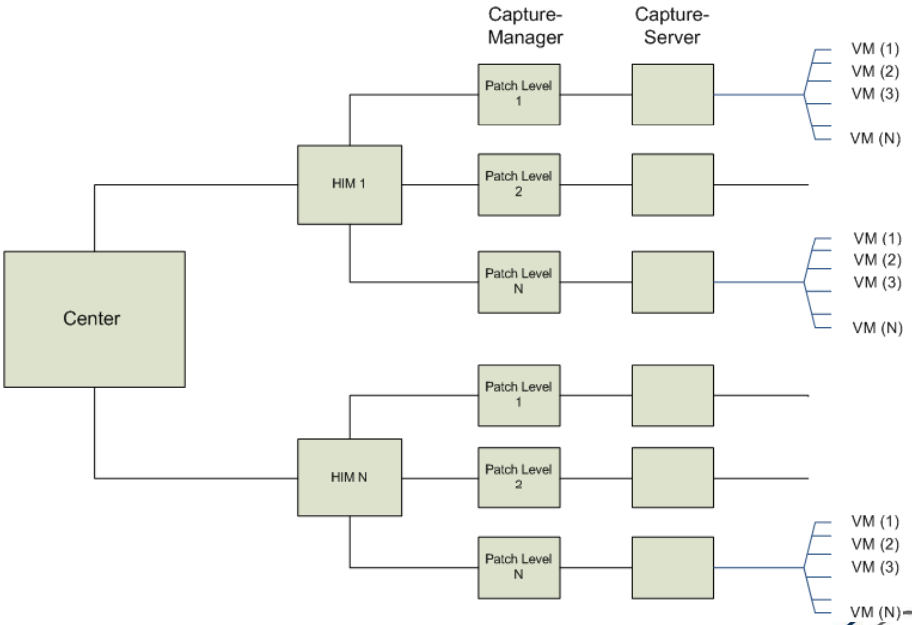


High interaction component (II)

- VMware stalling after thousands of reverts
- Had multiple problems with Capture-HPC server (logging, thread safety issues, lost urls, multiple VM support, others)
- Switched to VirtualBox
  - almost stable ☺
  - vm server and machines ids configured manually
  - client launched from autostart
  - socket communication instead of file
  - stability improvements (thread safety, etc.)
  - logging...



High interaction component (III)



HSN WAPI

Mapping WAPI objects to HSN

- Attacker = malicious URL
- CrawlerURL = URL data in LIM (including requests)
- Hirequest = request in HIM
- DownloadedFile = file details from HIM
- BulkURL = interface to WAPI-submitted URLs
- Dataset object – main gateway

Main HSN WAPI Objects

Attacker

- Generic information about the URL
  - processing state, resolved IP addresses & location info, URL in normalized and original form, fastflux, final classification...
- highInteractionScanIds() – results of high interaction scans
- getHIrequests([scanId]) – HIrequest objects
- getCrawlerURL() – associated CrawlerURL
- getAllDownloadedFiles() - DownloadedFile objects – results of HIM scanning

CrawlerURL

- Detailed information stored in the low interaction crawler
  - LIM processing state, obfuscation of request or content, fastflux, HTTP status, MIME type, script source...
  - depends on urlType (PARENT, REDIRECT, REQUEST)
- getScriptSource() – sources of scripts
- getAttacker() – available if urlType=PARENT
- getOrigin(), getRedirects(), getRequests()

Hirequest, DownloadedFile

- HIRequest
  - Info about requests generated by the high interaction module
    - request, IP, location, size, status, time...
  - getAttacker() - returns to the Attacker object
- DownloadedFile
  - Info about files modified during the scan
    - path+filename, size, MD5, SHA1, type of change (modified or deleted)

BulkURL

- Interface for monitoring URLs entered through WAPI
- processedAttackers() – Attacker objects for the URLs in the batch that reached a final state
- attackers() – Attacker objects for all URLs in the batch that have been imported into HSN

Entry points

HSN Services (Dataset)

> Dataset-level entry points

> searchURL(url/id,[deepSearch])
- gets Attacker objects for a given url or IP address
- if deepSearch is specified, also returns matching CrawlerURLs and Hirequests (much slower!)

> submitBulkURL(urls)
- submits a comma-separated list of URLs for processing and returns a BulkURL object

HSN Services (Dataset)

> getURLList(listType)
  - gets a black- or greylist (lisType=MALICIOUS or SUSPICIOUS)
> listURLsByHost(host)
  - list of URLs for a given host
> get_object(type,id)
  - helper service, useful as a shortcut if the type of the object and its identifier are known

HSN Acknowledgements

- NASK
  - Juliusz Brzostek
  - Krzysztof Fabjański
  - Tomasz Grudziecki
  - Jarosław Jantura
  - Paweł Jacewicz
  - Marcin Koszut
  - Adam Kozakiewicz
  - Tomasz Kruk
  - Marcin Mielniczek
  - Elżbieta Nowicka
  - Cezary Rzewuski
  - Sławomir Suliga
- SURFnet
  - Rogier Spoor
  - Wim Biemolt
  - Kees Trippelvitz
- GOVCERT.NL
  - Carol Overes
  - Michiel Hazen
  - Jeroen van Os
  - Menno Muller

Questions ?

Classifying Threats

Clustering malware with ANUBIS and SGNET

Paolo Milani Comparetti	pmilani@seclab.tuwien.ac.at
Corrado Leita	corrado_leita@symantec.com

Outline

- Motivation
- Anubis malware clustering
- SGNET EPM clustering
- Static vs Dynamic Malware Clustering

Motivation

Motivation

- Internet crime is a large, well-developed economy
  - a lot of $$$
  - a lot of malicious "players"
  - a lot of exploits, malware,.. to analyze/defend against
- Limited amount of resources to analyze all malicious activity
  - anti-virus companies receive thousands of new malware samples every day
  - honeypots gather large amounts of information on network attacks

Gathering data is just the first step

- malware collection
  - Anubis has analyzed >2 million samples (3000 per day)
  - Virustotal has analyed >20 million samples
- server-side honeypots
  - SGNET has observed ~60000 successful code injections
- client-side honeypots
  - Wepawet has analyzed >100k URLs (1700 per day)
  - Shelia and HoneySpiderNetwork are new systems, but they will soon have a lot of data as well

Classifying Threats

- To reduce the human effort required to understand and defend against threats, we need to classify them
- Recognize similar threats
  - automatically generated, polymorphic variants of a malware
  - malware variants generated with human intervention (new "releases")
  - scripted attacks
  - campaigns of attacks against network servers/clients
- Do not need to waste analysis resources on new instances of known threats
  - YAAB (Yet Another Allaple Binary)...

Anubis Malware Clustering

Anubis: Analyzing Unknown Binaries

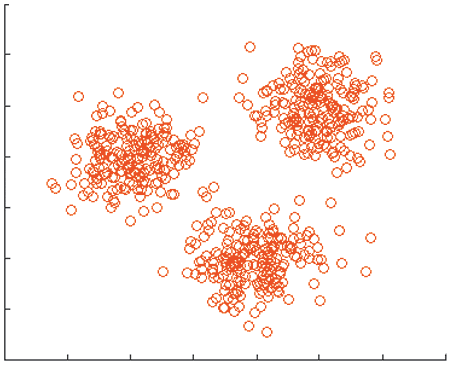
- Anubis automatically analyses submitted samples
  - executes them in an instrumented sandbox environment
- Provides a human-readable report on the malware's behavior
  - what does the program do?
  - network traffic? file-system modification? registry changes?...
- Someone still needs to look at each report!
  - 1000s of reports each day
  - (no, nobody looks at them all)

Malware Clustering

- Recognizing similar malware makes it possible to:
  - discard reports of samples that have been seen before
  - guide an analyst in the selection of those samples that require most attention
  - derive generalized signatures
  - implement removal procedures that work for a whole class of samples

Malware Clustering

- Find a partitioning of a given set of malware samples into subsets so that subsets share some common traits (i.e., find “virus families”)



Anubis Malware Clustering

- Behavioral clustering
- 2 samples are similar if they show similar behavior
  - their anubis reports will be similar

- Scalable
  - cluster all anubis samples
  - most clustering algorithms need to compute the distances between all pairs of points => $O(n^2)$
  - we use Locality Sensitive Hashing, less than $O(n^2)$ distance computations

Anubis Malware Clustering

- To classify malware we need a distance metric
- Each sample is represented as a set of:
  - OS Objects (files, sockets, processes, mutexes...)
  - OS Operations on those objects (read from a file, create a process...)
  - Data-flow dependencies between those objects
- Set:
  - Order, repetition, is irrelevant
  - use standard set distance function (Jaccard Index)

Example Behavior

```
src = NtOpenFile("C:\\sample.exe");
// memory map the target file
dst = NtCreateFile("C:\\Windows\\" + GetTempFilename());
dst_section = NtCreateSection(dst);
char *base = NtMapViewOfSection(dst_section);
while(len < length(src)) {
  *(base+len)=NtReadFile(src, 1); len++; }
```

```
Op|File|C:\sample.exe
  open:1, read:1
Op|File|RANDOM_1
  create:1
Op|Section|RANDOM_1
  open:1, map:1, mem_write: 1
Dep|File|C:\sample.exe -> Section|RANDOM_1
  read – mem_write
```


Example Behavior

```
src = NtOpenFile("C:\\sample.exe");
// memory map the target file
dst = NtCreateFile("C:\\Windows\\" + GetTempFilename());
dst_section = NtCreateSection(dst);
char *base = NtMapViewOfSection(dst_section);
while(len < length(src)) {
  *(base+len)=NtReadFile(src, 1); len++; }
```

```
Op|File|C:\sample.exe
  open:1, read:1
Op|File|RANDOM_1
  create:1
Op|Section|RANDOM_1
  open:1, map:1, mem_write: 1
Dep|File|C:\sample.exe -> Section|RANDOM_1
  read – mem_write
```


Example Behavior

```
src = NtOpenFile("C:\\sample.exe");
// memory map the target file
dst = NtCreateFile("C:\\Windows\\" + GetTempFilename());
dst_section = NtCreateSection(dst);
char *base = NtMapViewOfSection(dst_section);
while(len < length(src)) {
  *(base+len)=NtReadFile(src, 1); len++; }
```

```
Op|File|C:\sample.exe
  open:1, read:1
Op|File|RANDOM_1
  create:1
Op|Section|RANDOM_1
  open:1, map:1, mem_write: 1
Dep|File|C:\sample.exe -> Section|RANDOM_1
  read – mem_write
```


Example Behavior

```
src = NtOpenFile("C:\\sample.exe");
// memory map the target file
dst = NtCreateFile("C:\\Windows\\" + GetTempFilename());
dst_section = NtCreateSection(dst);
char *base = NtMapViewOfSection(dst_section);
while(len < length(src)) {
  *(base+len)=NtReadFile(src, 1); len++; }
```

```
Op|File|C:\sample.exe
  open:1, read:1
Op|File|RANDOM_1
  create:1
Op|Section|RANDOM_1
  open:1, map:1, mem_write: 1
Dep|File|C:\sample.exe -> Section|RANDOM_1
  read – mem_write
```


Example Behavior

```
src = NtOpenFile("C:\\sample.exe");
// memory map the target file
dst = NtCreateFile("C:\\Windows\\" + GetTempFilename());
dst_section = NtCreateSection(dst);
char *base = NtMapViewOfSection(dst_section);
while(len < length(src)) {
  *(base+len)=NtReadFile(src, 1); len++; }
```

```
Op|File|C:\sample.exe
  open:1, read:1
Op|File|RANDOM_1
  create:1
Op|Section|RANDOM_1
  open:1, map:1, mem_write: 1
Dep|File|C:\sample.exe -> Section|RANDOM_1
  read – mem_write
```


Example Behavior

```
src = NtOpenFile("C:\\sample.exe");
// memory map the target file
dst = NtCreateFile("C:\\Windows\\" + GetTempFilename());
dst_section = NtCreateSection(dst);
char *base = NtMapViewOfSection(dst_section);
while(len < length(src)) {
  *(base+len)=NtReadFile(src, 1); len++; }
```

```
Op|File|C:\sample.exe
  open:1, read:1
Op|File|RANDOM_1
  create:1
Op|Section|RANDOM_1
  open:1, map:1, mem_write: 1
Dep|File|C:\sample.exe -> Section|RANDOM_1
  read – mem_write
```


Example Behavior

```
src = NtOpenFile("C:\\sample.exe");
// memory map the target file
dst = NtCreateFile("C:\\Windows\\" + GetTempFilename());
dst_section = NtCreateSection(dst);
char *base = NtMapViewOfSection(dst_section);
while(len < length(src)) {
 *(base+len)=NtReadFile(src, 1); len++; }
```

```
Op|File|C:\sample.exe
  open:1, read:1
Op|File|RANDOM_1
  create:1
Op|Section|RANDOM_1
  open:1, map:1, mem_write: 1
Dep|File|C:\sample.exe -> Section|RANDOM_1
  read – mem_write
```


Example Behavior

```
src = NtOpenFile("C:\\sample.exe");
// memory map the target file
dst = NtCreateFile("C:\\Windows\\" + GetTempFilename());
dst_section = NtCreateSection(dst);
char *base = NtMapViewOfSection(dst_section);
while(len < length(src)) {
 *(base+len)=NtReadFile(src, 1); len++; }
```

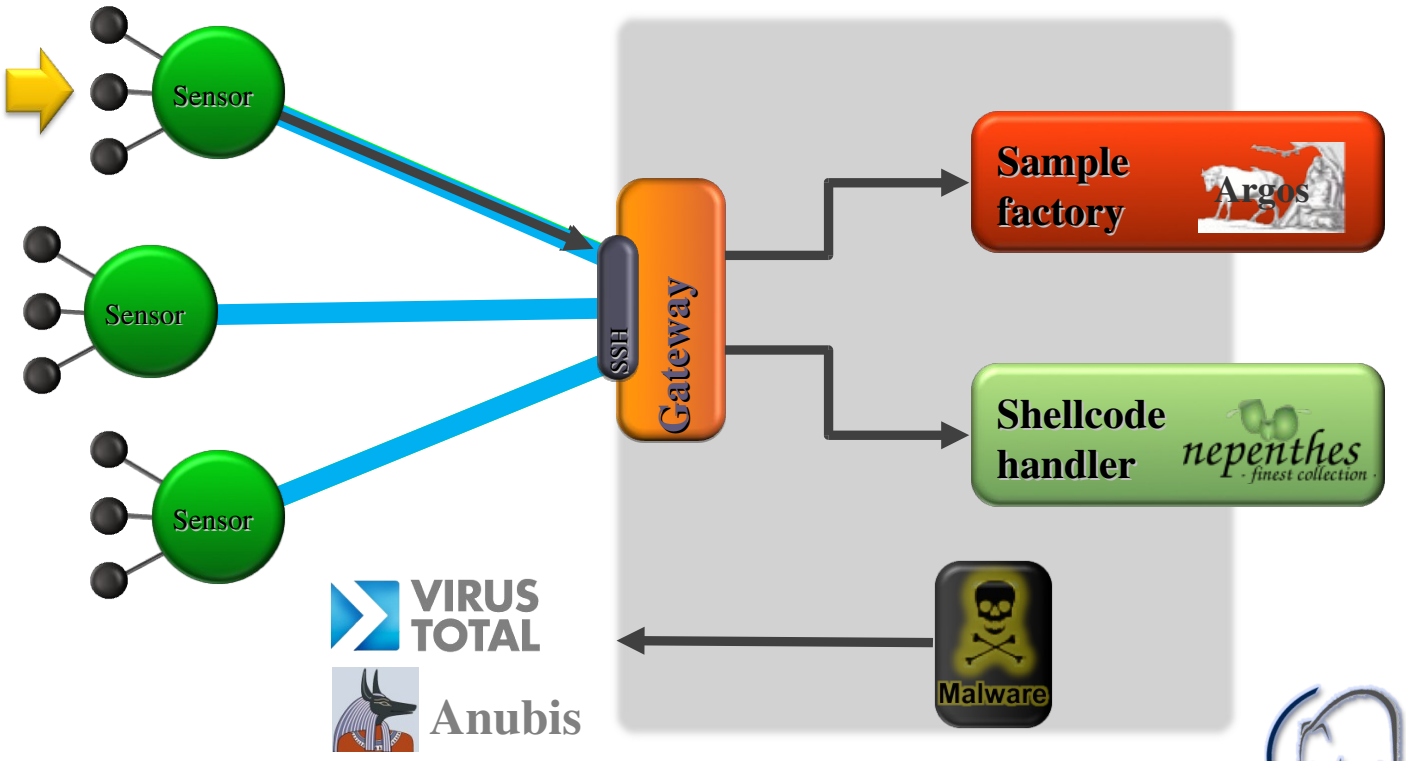
```
Op|File|C:\sample.exe
  open:1, read:1
Op|File|RANDOM_1
  create:1
Op|Section|RANDOM_1
  open:1, map:1, mem_write: 1
Dep|File|C:\sample.exe -> Section|RANDOM_1
  read – mem_write
```


Latest Clustering Results

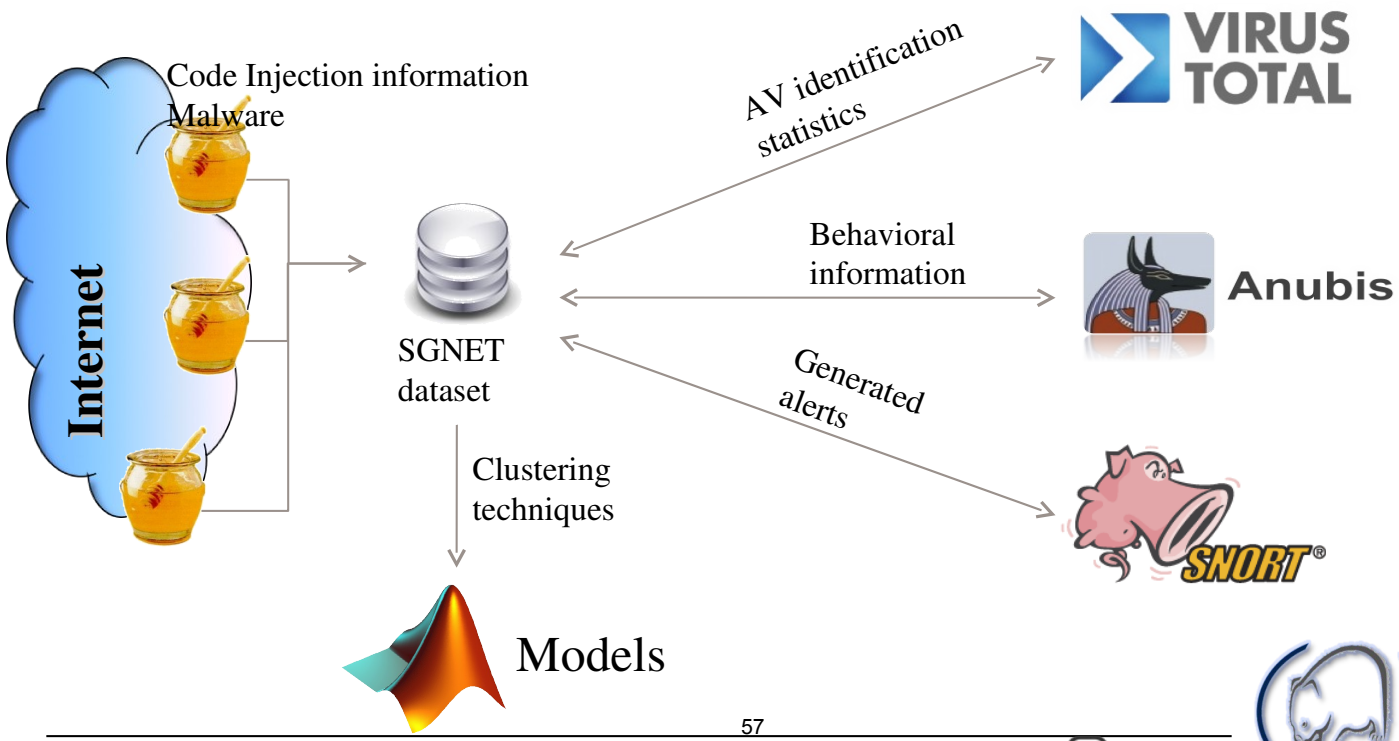
- 998505 anubis tasks
  - all samples analyzed by recent versions of anubis
- 80% similarity required for 2 samples to be in same cluster
  - high confidence! very similar anubis reports
- 91521 clusters
  - 10 times reduction in analysis work
  - largest cluster has 137925 anubis tasks
    - writes rasphone.pbk.. some kind of dialer
- runtime below 8 hours: we run it once a week
  - results available through web interface, WAPI

SGNET EPM Clustering

SGNET Operation

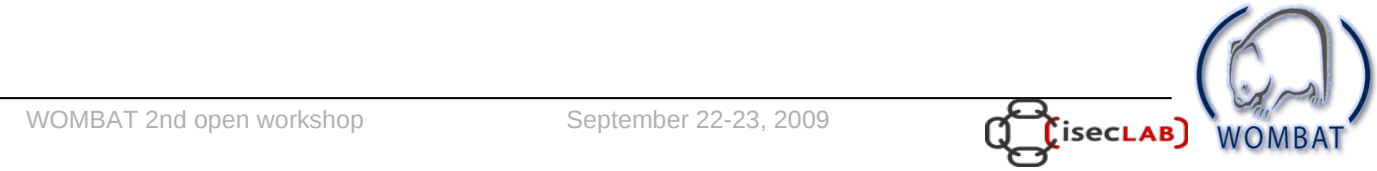


SGNET data enrichment framework

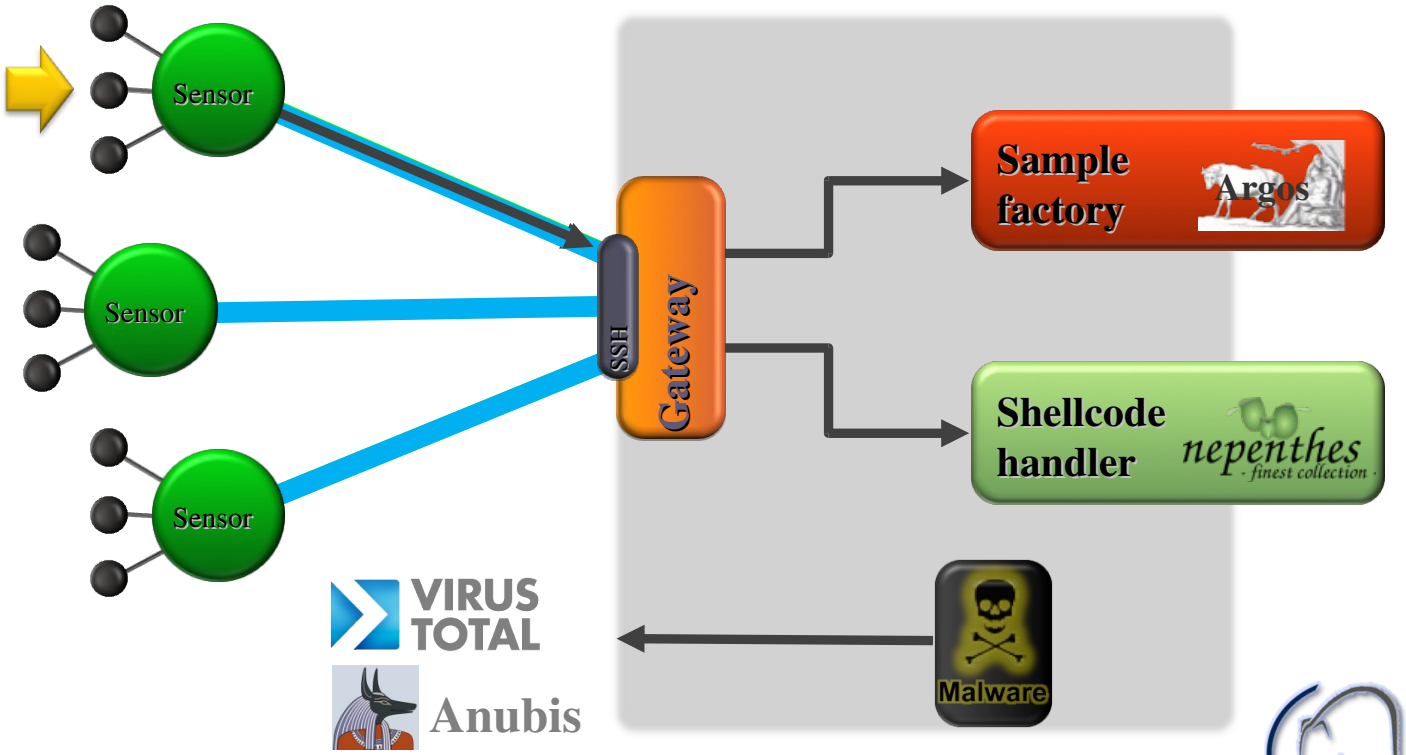


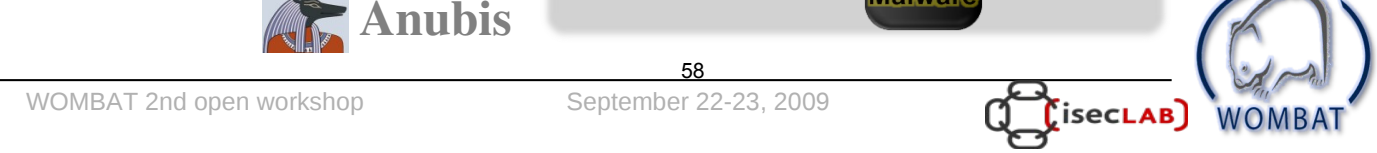
SGNET and Code Injections

- What do we know about the malware we collect?
- EGPM model:
  - Epsilon: the exploit (scriptgen)
  - Gamma: the control flow hijack (argos)
  - Pi: the shellcode (nepenthes)
  - Mu: the malware

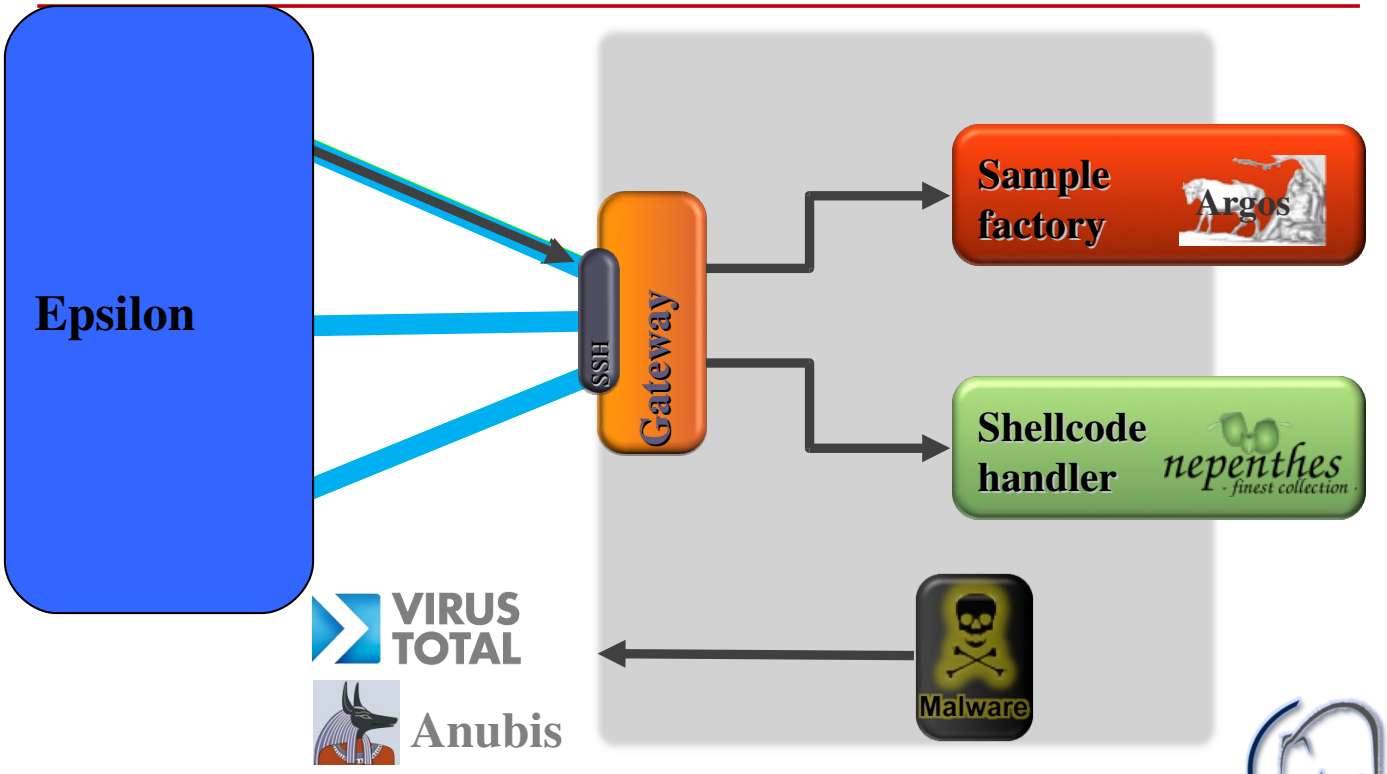


SGNET and Code Injections

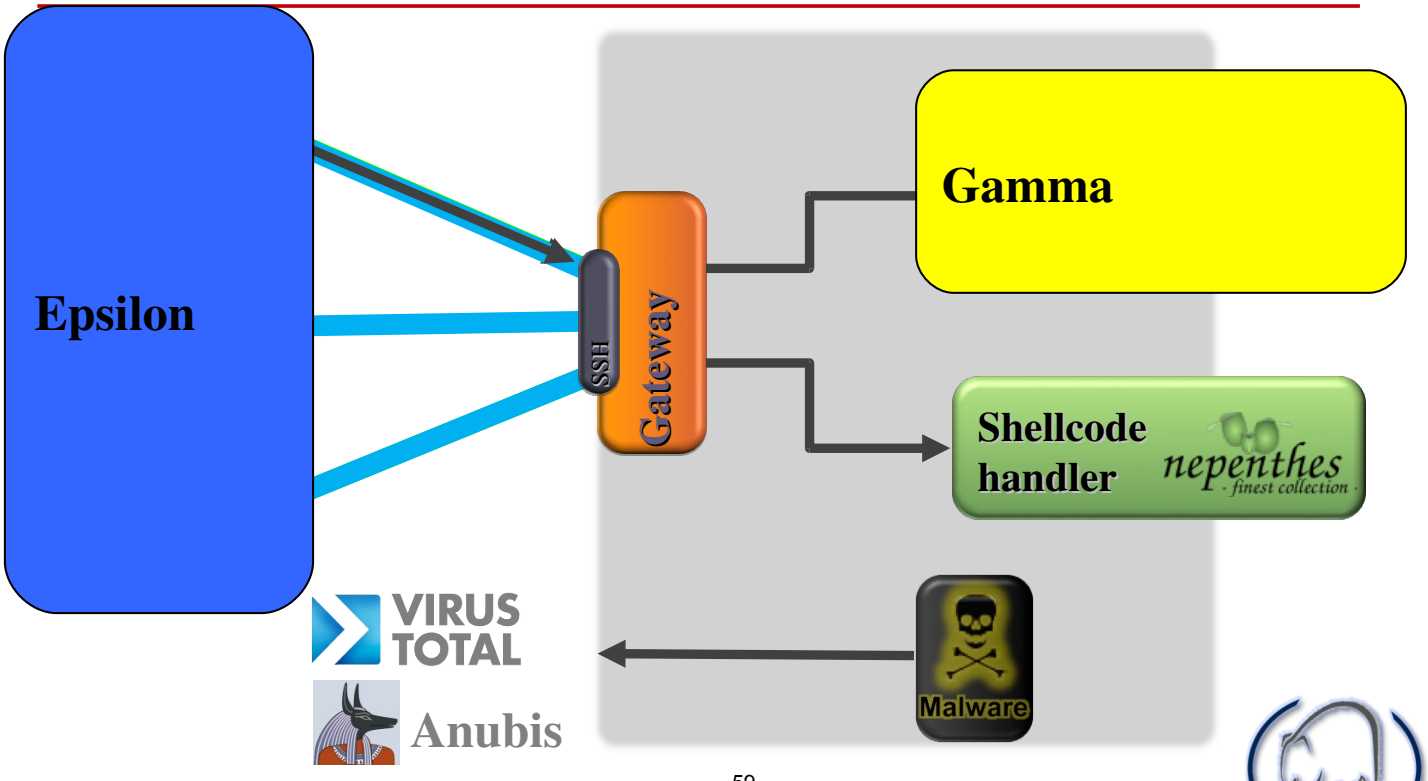




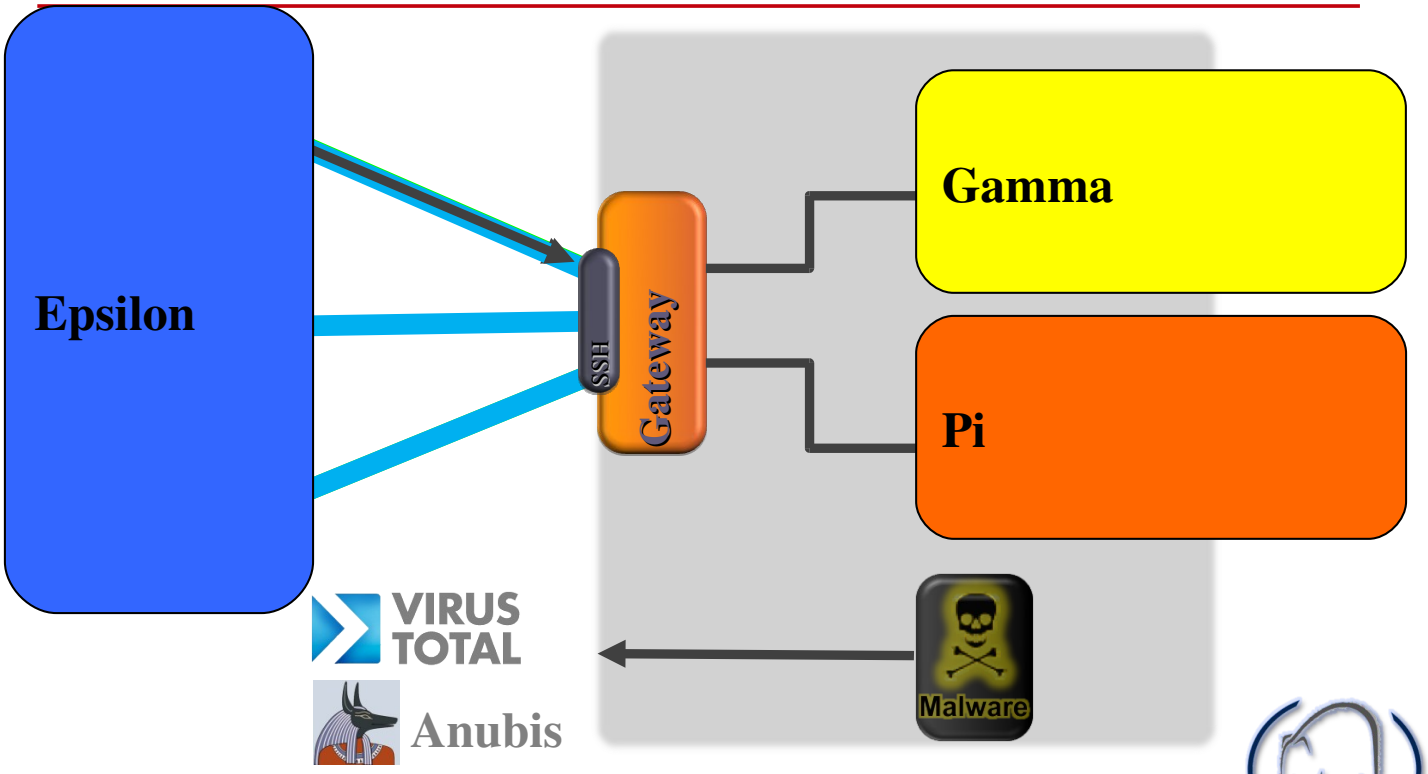
SGNET and Code Injections



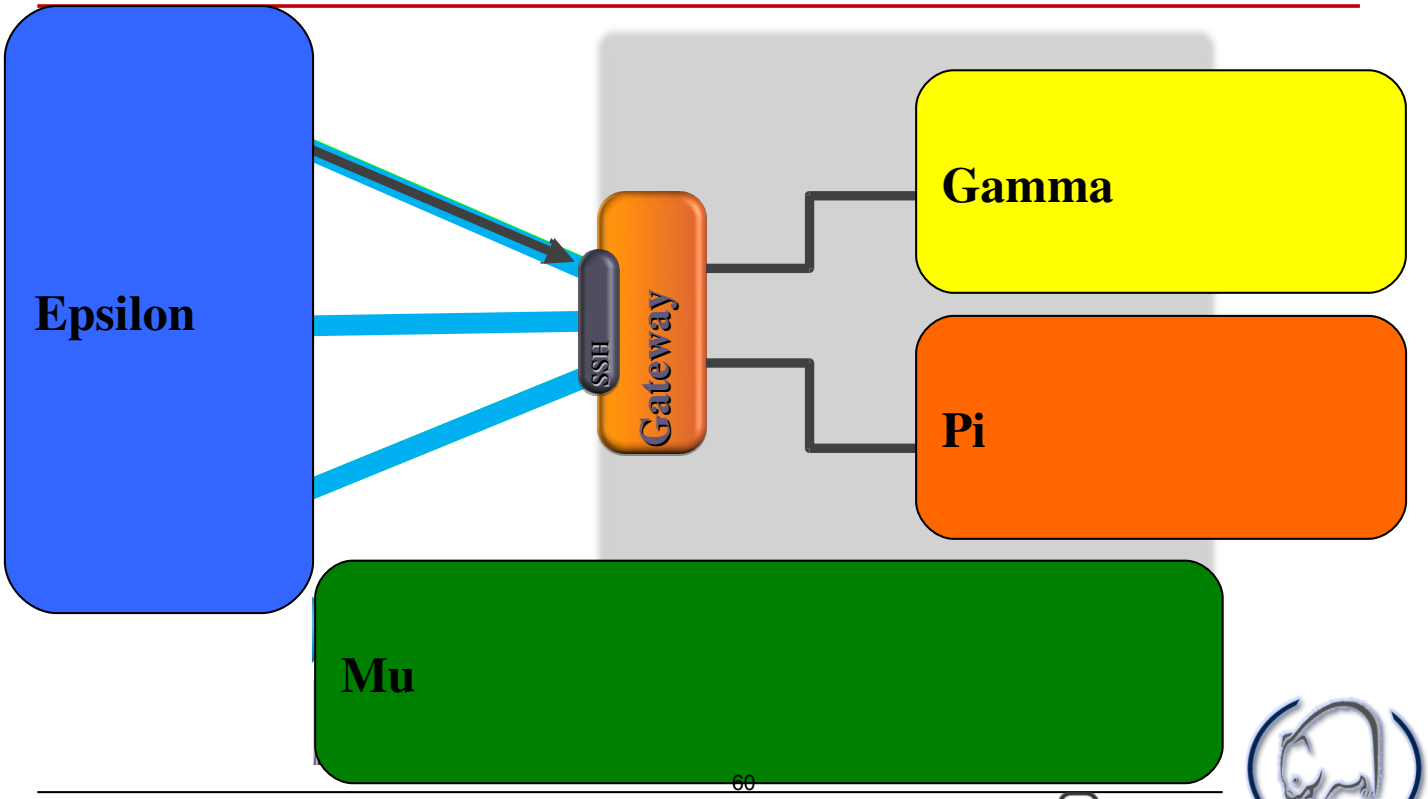
SGNET and Code Injections



SGNET and Code Injections



SGNET and Code Injections



SGNET and Code Injections

- What do we know about the malware we collect?
- The following EGPM dimensions are taken into consideration:
  - Epsilon: the exploit
  - Pi: the shellcode
  - Mu: the malware
- It would be interesting to study the relationships among these “dimensions” of a propagation attempt
- Problem: polymorphism
  - How to understand if two different samples are, or not, the same “mu”?

EPM Clustering

- For each dimension, consider multiple characteristics
  - Discover the “frequent” characteristics
  - basically a pattern-generation process
  - Proved to be sufficient to handle simple forms of polymorphism witnessed by the honeypot deployment

Content	Size	PE Sections	Section names
XXXXX	32678	3	data
YYYYY	32678	3	data
ZZZZZ	32678	3	data
AAAAA	5000	2	data,123packed
BBBBB	5000	2	data,435packed

EPM Dimensions

- **Epsilon**: exploit
  - FSM path: “type” of exploit
  - TCP port: high level view on the type of service
- **Pi**: shellcode
  - Protocol: protocol used for the malware download
  - Filename: name of the malware in the protocol interaction
  - TCP port: port involved in the malware download
  - Download type: PUSH/PULL/Centralized

EPM Dimensions

- Mu: malware
  - Md5
  - Size
  - Type
  - # PE sections
  - # DLL imports
  - PE OS version
  - PE linker version
  - PEiD packer identification
  - PE section names
  - List of importe DLLs
  - Symbols imported from Kernel32.dll

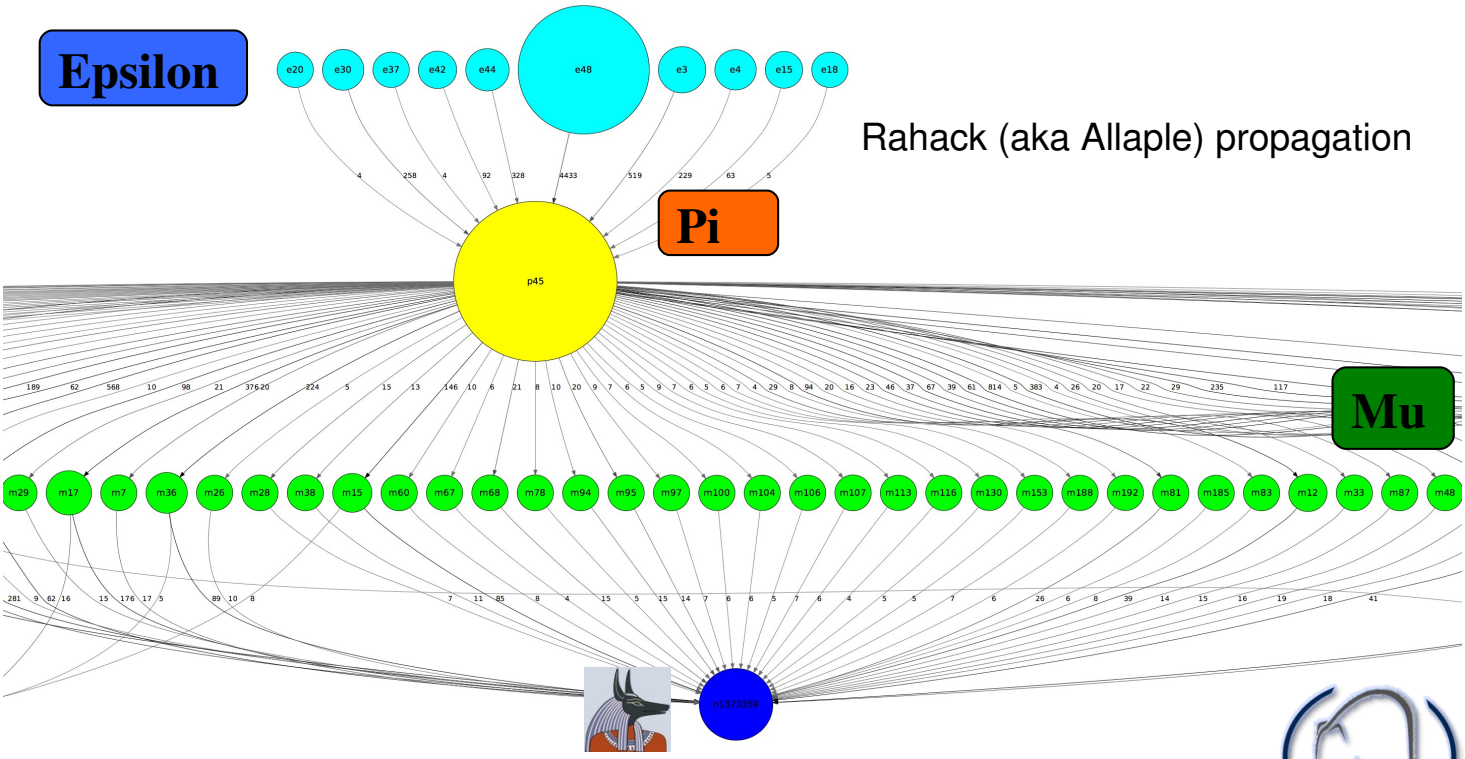
Only static parameters (file properties/PE header information) are taken into consideration

Static vs. Dynamic Malware Clustering

2 Complementary Approaches

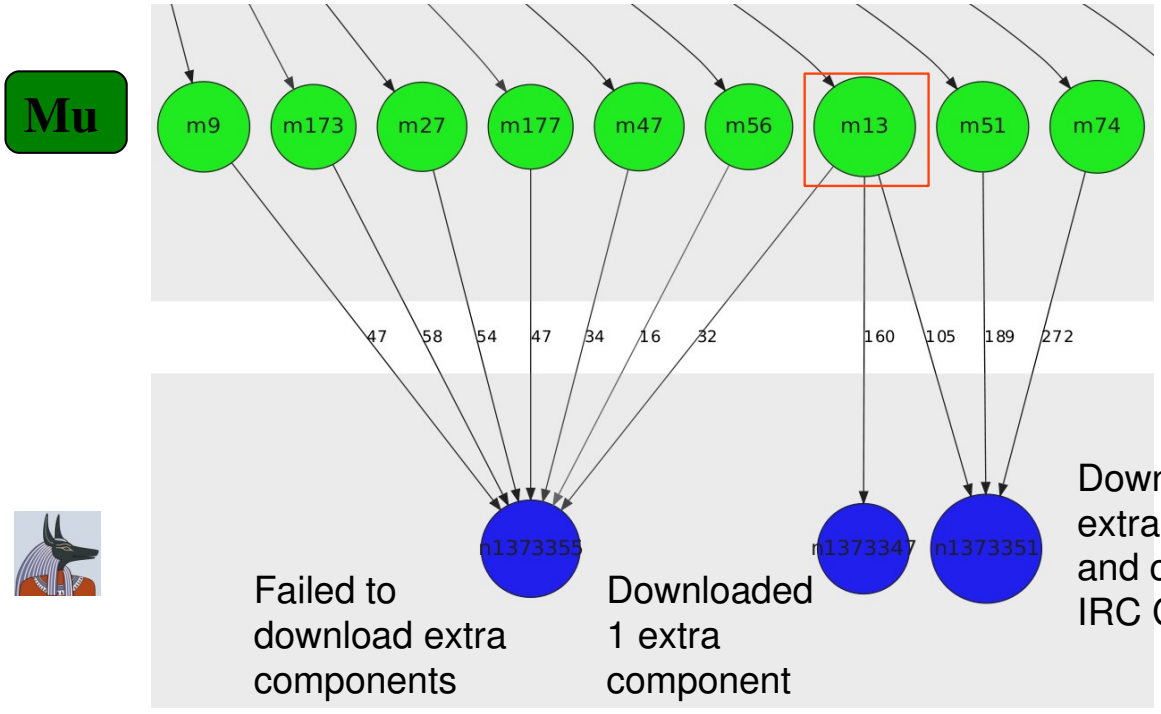
- Behavioral clustering has limitations
  - sandbox detection
  - Offline C&C channels
  - Blacklisting (?)
  - 4 minutes run time
- Static clustering has limitations too!
  - Allaple: lots of different “versions”
    - Different static characteristics (size, linker version, …)
    - Same behavior
  - It’s easy to “attack” by inserting more variability

Example 1: dynamic “wins”!



Rahack (aka Allaple) propagation

Example 2: static “wins”



Failed to download extra components

Downloaded 1 extra component

Downloaded 2 extra components and connected to IRC C&C

Understanding the Malware Ecosystem

- Rahack “M”-clusters
  - Same E,P tuple: they all share the same propagation strategy
  - Similar “M”-signatures: small differences in linker version, or binary size
  - Similar context:
    - Random scanning
    - Continued activity over almost all the SGNET existance
    - Distribution of the sources
- Other malware types sharing the same E,P tuple exist, but have different “M”-signatures and context!

References

- TTAnalyze (precursor to Anubis):
  - U. Bayer, C. Kruegel, and E. Kirda, “TTAnalyze: A Tool for Analyzing Malware” (EICAR 2006)
- Anubis Clustering:
  - U. Bayer, P. Milani Comapretti, C. Hlauscheck, C. Kruegel and E. Kirda: “Scalable, Behavior-Based Malware Clustering” (NDSS 2009)
- SGNET:
  - C. Leita, “SGNET : automated protocol learning for the observation of malicious threats” Ph.D. Thesis
  - C. Leita, M. Dacier, “SGNET: a worldwide deployable framework to support the analysis of malware threat models” (EDCC 2008)

Any Questions?

The 2nd WOMBAT Workshop

Using WOMBAT APIs on Real-World Tasks

September 22-23, 2009

Preliminaries

During this demonstration, you will be shown how to take advantage of the WAPI to query different WOMBAT datasets, namely Anubis, SGNET, HARMUR, Shelia, HSN, WEPAWET, VirusTotal and FORTH.

As explained during the presentation, WAPI takes advantage of SSL to provide confidentiality of the transmitted results and, most importantly, to implement access control. In order to be able to talk with a WAPI server, a client must provide a valid SSL certificate signed by the Certification Authority maintained by each partner providing WAPI services. In the scope of this demo, we have already prepared for you all the required certificates as well as a simple client able to allow interaction with all the available WAPI services.

```
python wapi_client.py -c democonf
```

If available, the provided client will take advantage of the IPython interactive shell (`http://ipython.scipy.org/moin/`) as opposed to the original Python shell, providing better support for autocompletion and results visualization.

To have an idea of the underlying configuration, you can find here the `democonf` configuration file:

```
# WAPI client configuration file. Specifies
# the list of WOMBAT datasets as well as the
# connection options to connect to each of them.

# general format:
# [<dataset_name>]
# url=<scheme>://<hostname or ip>/path/to/wapi/server/
# cert_ca=relative/path/to/ca/certificate.pem
# cert_client=relative/path/to/client/certificate.pem
# namespace=<namespace>
```

```
#NOTE: cert_ca, cert_client and namespace are optional.

[sgnet]
url=https://193.55.112.70/sgnet/
cert_ca=cert/sgnet/cacert.pem
cert_client=cert/sgnet/client.pem
namespace=sgnet.wapi.wombat-project.eu

[wepawet]
url=https://193.55.112.70/wepawet/
cert_ca=cert/wepawet/cacert.pem
cert_client=cert/wepawet/client.pem
namespace=wepawet.wapi.wombat-project.eu

[harmur]
url=https://193.55.112.70/HARMUR/
cert_ca=cert/harmur/cacert.pem
cert_client=cert/harmur/client.pem
namespace=HARMUR.wapi.wombat-project.eu

[anubis]
url=wapi://isis.iseclab.org:8080/anubis/
cert_ca=cert/anubis/cacert.pem
cert_client=cert/anubis/client.pem
namespace=anubis.wapi.wombat-project.eu

[shelia]
url=https://centaur.few.vu.nl:8080/shelia/
cert_ca=cert/shelia/cacert.pem
cert_client=cert/shelia/client.pem
namespace=shelia.wapi.wombat-project.eu

[forth]
url=wapi://139.91.90.201/forth/
cert_ca=cert/forth/cacert.pem
cert_client=cert/forth/client.pem
namespace=forth.wapi.wombat-project.eu

[virustotal]
url=wapi://62.15.230.161/virustotal/
cert_ca=cert/virustotal/cacert.pem
cert_client=cert/virustotal/client.pem

[hsn]
url=https://gror.nask.waw.pl:8888/hsn/
cert_ca=cert/hsn/hsn_ca.pem
cert_client=cert/hsn/hsn_user.pem
namespace=hsn.wapi.wombat-project.eu
```

```
[utils]
anubis_url=https://anubis.iseclab.org/index.php?action=result&task_id=%s&format
    =html
```

1 Investigation of a Banking Fraud

In this scenario, the participants take on the role of CERT responders from a bank. The bank needs to conduct a (forensics) investigation of the machine of a client that has reported a fraud case via electronic banking. The bank up to now has excluded that the fraud was related to phishing or any other physical swindle.

A brief analysis of the infected client does not show any clear evidence of infection, no suspicious BHO is detected and no suspicious registry entries are found in the system. The client affected by the fraud is connected to the Internet through an HTTP proxy, and has agreed to give you the list of the HTTP activity of the infected machine in the last week. After a brief look at such activity, you notice a large amount of HTTP requests towards a suspicious domains. Such requests are performed every 20 minutes approximately, during working hours but also during night and weekends. All the queried URLs are similar to the following one: `http://ijmkkyjves.net/iE=eQBHE8cNe8DRM`

1.1 Malware identification

Can we take advantage of the WAPI to link such suspicious behavior to a specific malware sample?

Q Anubis stores behavioral information for thousands of malware samples during an execution time of two minutes approximately. Is there any malware sample analyzed by Anubis that exposed a similar behavior during its analysis?

A An example solution follows:

```
#let's search for any HTTP conversation targeting the identified
    domain
http = anubis.http_traffic(destination="ijmkkyjves.net")

#let's retrieve the WAPI malware objects associated to this behavior
malware = [h.tasks()[0].malware()[0] for h in http]
```

Q Look more in depth at the MD5 hash of these samples and at their binary size. Is there anything striking about these characteristics?

A Let's extract these simple characteristics from the Anubis malware objects:

```
#what are the basic characteristics of these samples?
stats = set([(m.md5,m.file_size,m.mime_type) for m in malware])
```

Q Try to take advantage of the VirusTotal API to give a name to these samples.

A An example solution follows:

```
md5_hashes = set([m.md5 for m in malware])

for md5 in md5_hashes:
```

```
print "=== %s"%md5
print virustotal.get_file(md5=md5)[0].get_last_analysis()[0].
    av_positives_report
```

1.2 Infection analysis

In the previous step, we have been able to link the suspicious network behavior observed on the infected host with a specific malware sample, called Mebroot. Mebroot received a certain amount of press coverage (see, for instance, `http://www.symantec.com/connect/blogs/bootroot-trojanmebroot-rootkit-your-mbr`). While the low antivirus detection rate might explain why such malware was not detected by the antivirus software installed on the infected machine, we still do not know how the machine got infected in the first place.

We go back to the logs, and we are able to identify the moment in which the anomalous connection attempts started. Assuming that the infection happened approximately at that time, we extract a list of URLs visited by the infected machine in the hour preceding the beginning of the anomaly. The traffic was concerning the following domains:

```
domains = ["google.com",
"facebook.com",
"baidu.cn",
"adobe.com",
"bandwidthplace.com",
"azadars.com"]
```

We therefore hypothesize that one of these domains is the potential cause of the infection. Let's use the WAPI to verify this hypothesis.

Q Check if any of the domains visited by the client are known to the HoneySpider network.

A An example solution follows:

```
for d in domains:
hsnatts = hsn.searchURL(url=d)
if len(hsnatts) == 0:
print "No results for %s"%d
else:
for hsnatt in hsnatts:
print "Result for %s:"%d
print "URL: %s, classified as %s"%(hsnatt.normalizedURL,
hsnatt.classification)
```

Q `azadars.com` is seen by the HoneySpider Network as suspicious. Take advantage of the WAPI to understand what type of threat is associated to it.

A An example solution follows:

```
hsnazadars = hsn.searchURL(url="azadars.com")[1]
hsnazadars.dump();
# Was it scanned with the high interaction component?
print hsnazadars.highInteractionScanIds()

# Ok, and what was the result of low interaction scan?
hsnazalim = hsnazadars.getCrawlerURL()[0]
hsnazalim.dump()

#So, low interaction heuristics show, that this is just suspicious.
    Why?
#Let's have a look at the requests:"
hsnreqs = hsnazalim.getRequests()
for req in hsnreqs:
print "%s request %s"%(req.classification,req.request)

#Why is the suspicious one actually suspicious?
for req in hsnreqs:
if req.classification == "SUSPICIOUS":
req.dump()

#So, it was obfuscated. Were any redirections extracted from the
    scripts?
for redir in hsnazalim.getRedirects():
print "%s redirection to %s"%(redir.classification,redir.request)

#There was one, but heuristics found nothing interesting there."
```

Q Try to perform the same analysis taking advantage of the shelia dataset.

A An example solution follows:

```
sheliaazadars = shelia.alerts_by_target(target="azadars.com")[0]

#is there any malware downloaded as a consequence of the analysis?
sheliamalware = sheliaazadars.malware()[0]

#notice the MD5 and the file length and compare them with
#the malware identified in Anubis. Is there anything interesting?
```

Q Do you think that the temporal evolution might be the reason for which Shelia and HSN provide discording reports? Try to validate this hypothesis taking advantage of the different WAPI datasets.

A First of all, we should look at the timestamp of the analysis tasks for the domain `azadars.com`. Did Shelia and HSN look at the site at the same moment in time?

```
#when did HSN first look at the domain?
```

```
print hsnazadars.creationDate

#when did Shelia analyze the same domain?
print sheliaazadars.timestamp
```

If we look at the wepawet information, we can have a more clear proof that `azadars.com` changed over time.

```
wepawetazadars = wepawet.domain(domain_name="azadars.com")[0]

#let's look at the different analysis tasks
for t in wepawetazadars.tasks():
print "On %s, there were %d exploits"%(t.analyzed_at,len(t.exploits())
    )
```

Q Which vulnerabilities have been exploited?

A The answer can be easily retrieved by looking in more detail the wepawet analysis.

```
#let's pick the oldest analysis task
wepa_ana = wepawetazadars.tasks()[0]

#which exploits were detected?
for expl in wepa_ana.exploits():
expl.dump()
```

1.3 The real culprit

In the previous Section, we have been able to identify a domain, `azadars.com`, that was visited by our victim just before the infection. We have been able to show that, before September, this domain was indeed capable of exploiting clients to install malware and compromise our victim machine. We don't know yet much about `azadars.com`: is it a malicious site set up on purpose to compromise victims redirected to it through phishing campaigns? Let's try to use the WAPI to know more about it.

Q Take advantage of the HARMUR dataset to know more about the `azadars.com` site. Is the site registrant associated to other malicious domains?

A An example solution follows:

```
azadars = harmur.domain(domain="azadars.com")[0]
azadars.dump()

print azadars.same_registrant()
```

Q On what physical server is the site hosted?

A An example solution follows:

```
azadars_srv = azadars.servers()[0]

#let's print all the available information
azadars_srv.dump()
```

We can query HARMUR to know more about other sites hosted on the same physical server.

```
azadars.servers()[0].reverse_resolution()
same_server = [(d.name,d.color) for d in azadars.servers()[0].
    hosted_domains()]
```

Q From WHOIS/DNS information, `azadars.com` looks like a legitimate site hosted by a web hosting service that has been compromised by an exploit toolkit. Look at the exploit information provided by HARMUR and wepawet and try to confirm this hypothesis.

A HARMUR aggregates different threat information feeds to decide whether a certain domain is malicious or not. What can we say about this domain?

```
for threat in azadars.threats():
threat.dump()
```

Wepawet allows us to analyze more in detail the effect of the exploits detected in the previous phase.

```
for payload in wepa_ana.payloads():
print "download %s from %s"%(payload.md5,payload.url)
#compare with Anubis samples
if payload.md5 in md5_hashes:
print "the downloaded sample performs connections to ijmkkyjves.net
    once executed!"
```

The site azadars.com seems to be infected by an exploit toolkit that forces the victim to download from the domain ijmkkyjves.com a malware sample whose MD5 was analyzed by Anubis and that is known to be performing connections to ijmkkyjves.net.

Q "ijmkkyjves" looks like a randomly generated string. Are these two domains part of a bigger picture?

A We can query the HARMUR dataset to find an answer to this question:

```
dl_site = harmur.domain(domain="ijmkkyjves.com")[0]
#all the information available on the server:
dl_site.servers()[0].dump()
#which other domains are hosted on the same server?
dl_site.servers()[0].reverse_resolution()
```

1.4 Conclusions

During this demonstration we have explored the potential of the WAPI to build better pictures on the threats "ecosystem". We have started from a real case scenario, and we have combined information retrieved by Anubis with the analysis performed by different client honeypot technologies. We have seen how each of these datasets is often able to show us only one facet of the truth, and the value of aggregating together these different facets to get a broader view on Internet threats.

2 Monitoring of Own Networks

In this scenario, the participants are in the security staff of an ISP or an enterprise network (or even a CERT for a given country), and they are interested in querying our sources to get information about infected machines in their own network.

`211.108.242.0/24`

The idea is to offer to network administrators useful information that might help them in understanding what type of threat is affecting the different clients in order to clean them or notify them.

2.1 Searching for infections

Q We can start by querying the SGNET dataset to know if any honeypot has observed malicious activity generated by hosts of our own network.

- What exploit events have we found, if any?
- Can we associate to these exploit events the activity of a specific malware type?
- If yes, how many?

A We first retrieve all the known sources and, for each sources, we save the list of events associated with it.

```
ip = "211.108.242.0"
print "We are interested in our network %s/24" % ip

#What does sgnet know about it
sources = sgnet.source(address=ip,network_prefix=24)
print "We have %d sources" % len(sources)
```

Q One host of our network has performed a successful code injection attack against one of the SGNET honeypots, and is therefore likely to be infected. Take advantage of the Anubis and VirusTotal dataset to know more about the nature of this malware. Try to give a "name" to such sample.

A An example solution follows:

```
source = sources[0]
print "Address is %s" % source.address

event = source.events()[0]
print "Here is the exploit event"

event.dump()
event_md5 = malware.md5
print "Uploaded malware with md5 %s\n" % event_md5
```

```
#What does virustotal say about the md5?
print virustotal.get_file(
    md5=event_md5)[0].get_last_analysis()[0].av_positives_report

event_tasks = anubis.malware(md5=event_md5)[0].tasks()
print "\nAnubis has %d tasks for this md5\n" % len(event_tasks)

#Here is one
event_tasks[0].dump()
```

Q We want to take a deeper look at this task to see, say, what type of activity it performed (e.g., files created, remote HTTP connections).

A An example solution follows:

```
#Let's look at this task in more detail
t = event_tasks[0]

#File events
print [x.name for x in t.file_events()]

#Registry keys created
print [x.key_name for x in t.created_keys()]

#HTTP traffic
print t.http_traffic()
print "No real files, and nothing very meaningful in the registry
    either\n"
```

Q Recall that the web interface of Anubis contain many details about capture samples. Can we take a deeper look at it? We can rely on the `open_anubis_report` shortcut method provided by the `Utils` module.

A This can be done by invoking the appropriate method:

```
event_uuid = event_tasks[0].uuid
print "\nAnubis has this as task uuid %s\n" % event_uuid

#Look at the report
Utils.open_anubis_report(event_uuid)

#It has an exception, does not do much
```

2.2 Looking for similar malware samples

From the previous point, we have seen that the specific exploitation behavior used by this malware sample is used by lots of different malware groups. While such

exploitation behavior is mainly famous for being associated to the Rahack/Allaple worm, it seems to be used also by other, less known, malware families.

Q Let's look more in depth at this malware cluster and let's take advantage of Anubis to identify any less visible, but interesting, behavior. To reduce querying time, we can safely cap our requests to the first 30 samples. Also, remember that the `Utils` module provides the handy `flatten_list`, which just does what it says.

A An example solution follows:

```
#Are both samples in same cluster?
print set(Utils.flatten_list([t.cluster() for t in event_tasks]))

cluster = t.cluster()[0]
cluster.dump()

#It's a big cluster. Let's look at a few of the tasks in it
#let's take the first 30, and see if they do http

cluster_tasks = cluster.tasks()[0:30]
httpl = [t.http_traffic() for t in cluster_tasks]
cluster_http = Utils.flatten_list(httpl)
print set([x.dest_name for x in cluster_http])

#..or interesting file stuff

filel = [t.file_events() for t in cluster_tasks]
cluster_files = Utils.flatten_list(filel)
print set([x.name for x in cluster_files])
```

Q It looks like the whole cluster is not doing much. Maybe we can retrieve other similar samples by leveraging the SGNet EPM clustering and rely on Anubis to figure out what type of HTTP traffic these malware are generating. In particular, we are interested in finding malware that attempt to connect to a domain we retain suspicious: `zief.pl`.

A An example solution follows:

```
#Let's use sgnet EPM clustering, to see if similar activities
#have malware with more interesting behavior

activity = event.activity()[0]
print "Here is the activity:\n"
activity.dump()

#Looking up MD5s for this activity
epm = activity
md5s_epm = [m.md5 for m in epm.malware()[0:30]]
print "\nGot %d MD5s\n" % len(md5s_epm)
```

```
#Let's look for anubis tasks
malwares = [anubis.malware(md5=m) for m in md5s_epm]
malwares = filter(lambda m:len(m),malwares)
malwares = [m[0] for m in malwares]
tasksl = [m.tasks() for m in malwares]
tasks = Utils.flatten_list(tasksl)
print "Got %d tasks" % len(tasks)

#Looking up all of their HTTP traffic
httpl = [t.http_traffic() for t in tasks]
http = Utils.flatten_list(httpl)

#Here are all accessed domains
print list(set([x.dest_name for x in http]))

#We are interested in zief.pl
http_zief = filter(lambda h:h.dest_name.endswith("zief.pl"),http)
```

2.3 Looking more in depth at zief.pl

We have started from a single infection of a polymorphic worm and we have been able, taking advantage of SGNET EPM clustering, to identify a larger set of malware samples and infections that are linked to the same malware group. While the specific sample from which we started our analysis did not expose any other behavior than simple worm-like propagation, we have been able to identify in other samples belonging to the same group some suspicious HTTP behavior related to a specific domain, `zief.pl`.

From now on, we want to dig a little bit deeper on the `zief.pl` domain. What is its role in the infection?

Q Take advantage of the HARMUR dataset to get information about the server(s) hosting this domain. Are they located in Poland or somewhere else?

A An example solution follows:

```
#What does harmur say of threats from zief.pl?
domain = harmur.domain(domain="zief.pl")[0]

#Here are urls harmur has on this domain
print [u.url for u in domain.urls()]

#Here is a summary of the threats on this domain
print [(t.type,t.id) for t in domain.threats()]

#Here is one of the bloodhound threat in more detail
domain.threats()[1].dump()

#Let's look up one of the threats on the symantec website
```

```
threat_urls = domain.threats()[1].help.split()
print threat_urls

#Let's look up the servers' geo location information
servers = domain.servers()

#only one of the three servers is located in China!
for srv in zief_servers:
    srv.dump()
    print srv.reverse_resolution()
```

Q Looks like only one of the three servers is actually located in China. Maybe the FORTH dataset can provide further information about `zief.pl`. Let's query it.

A An example solution follows:

```
#Where is zief.pl located?
zief_ips = set([h.dest_ip for h in http_zief])
print "Here are the ips:\n%s\n" % str(zief_ips)
zief_ip = list(zief_ips)[0]

#Let's look up the whois data
forth.address(ip_addr=zief_ip)[0].dump()

#Does this address send spam?
print forth.address(ip_addr=zief_ip)[0].isSpammer()
```

Q `zief.pl` seems to be a download site for additional components retrieved by the malware once executed. Is there any additional threat on this domain? We can take advanatage of the Anubis web interface to retrieve in depth information, on its servers and on any threat known to be associated to it.

A An example solution follows:

```
#Let's see if anubis tasks that contact zief.pl are more interesting
zief_http = filter(lambda x:x.dest_name.endswith("zief.pl"),http)
zief_tasksl = [h.tasks() for h in zief_http]
zief_tasks = Utils.flatten_list(zief_tasksl)
map(lambda t:zief_tasks.extend(t),zief_tasksl)
print "We have %d tasks that contact www.zief.pl\n" % len(zief_tasks
    )

#How many anubis clusters are they in?
print set(Utils.flatten_list([t.cluster() for t in zief_tasks]))

#Here is one such anubis task
zief_tasks[0].dump()
print "MD5 is %s" % zief_tasks[0].malware()[0].md5
```

```
#Look at the report. A lot more going on...
Utils.open_anubis_report(zief_tasks[0].uuid)
```

Q Interestingly, we have found a match into Anubis. It's worth looking up VirusTotal analyses about the malware sample we just found in Anubis. Optionally, the WEPAWET dataset may be useful to find any client-side threat (e.g., JavaScript, PDF, Flash) related to `zief.pl`.

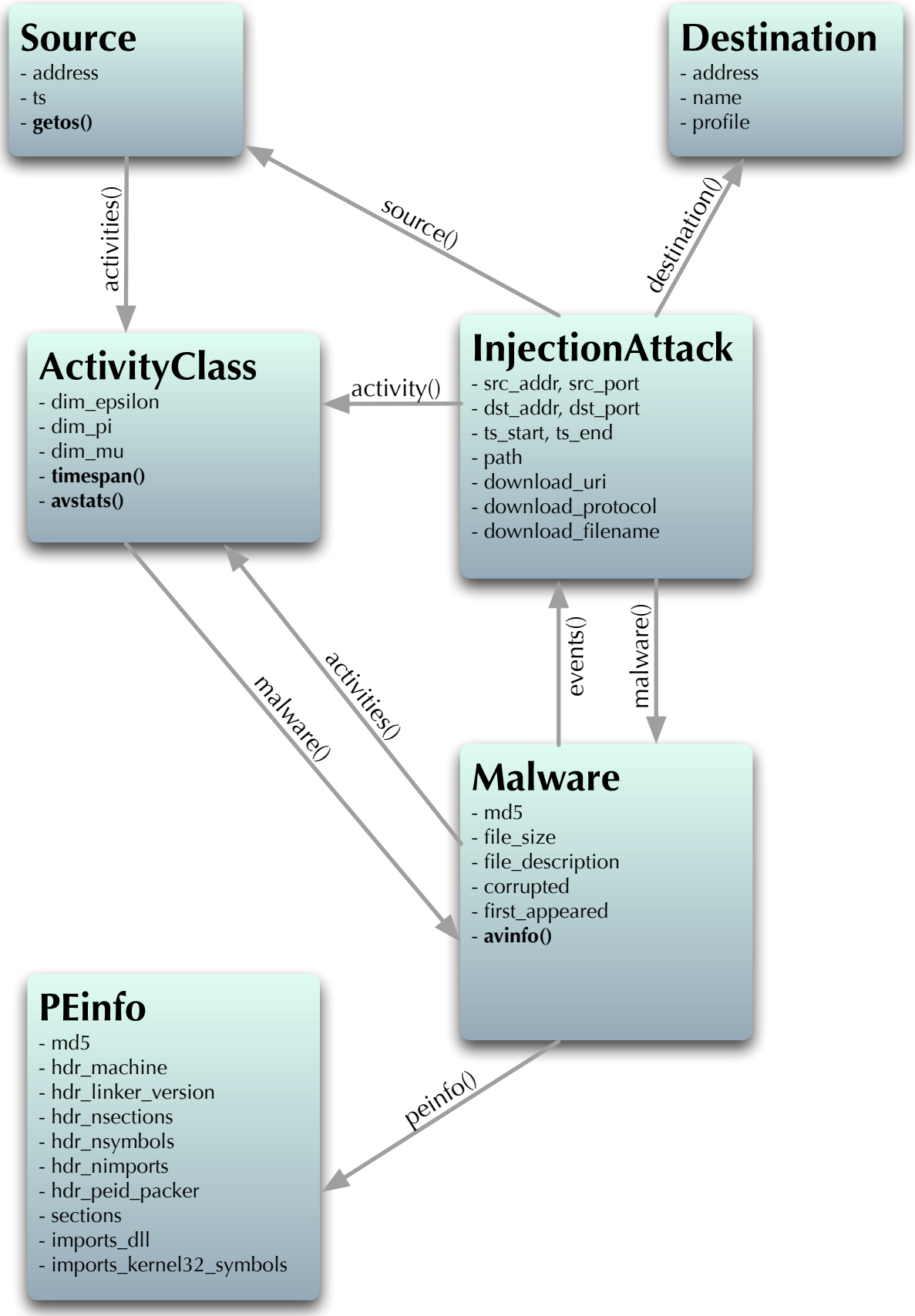
A An example solution follows:

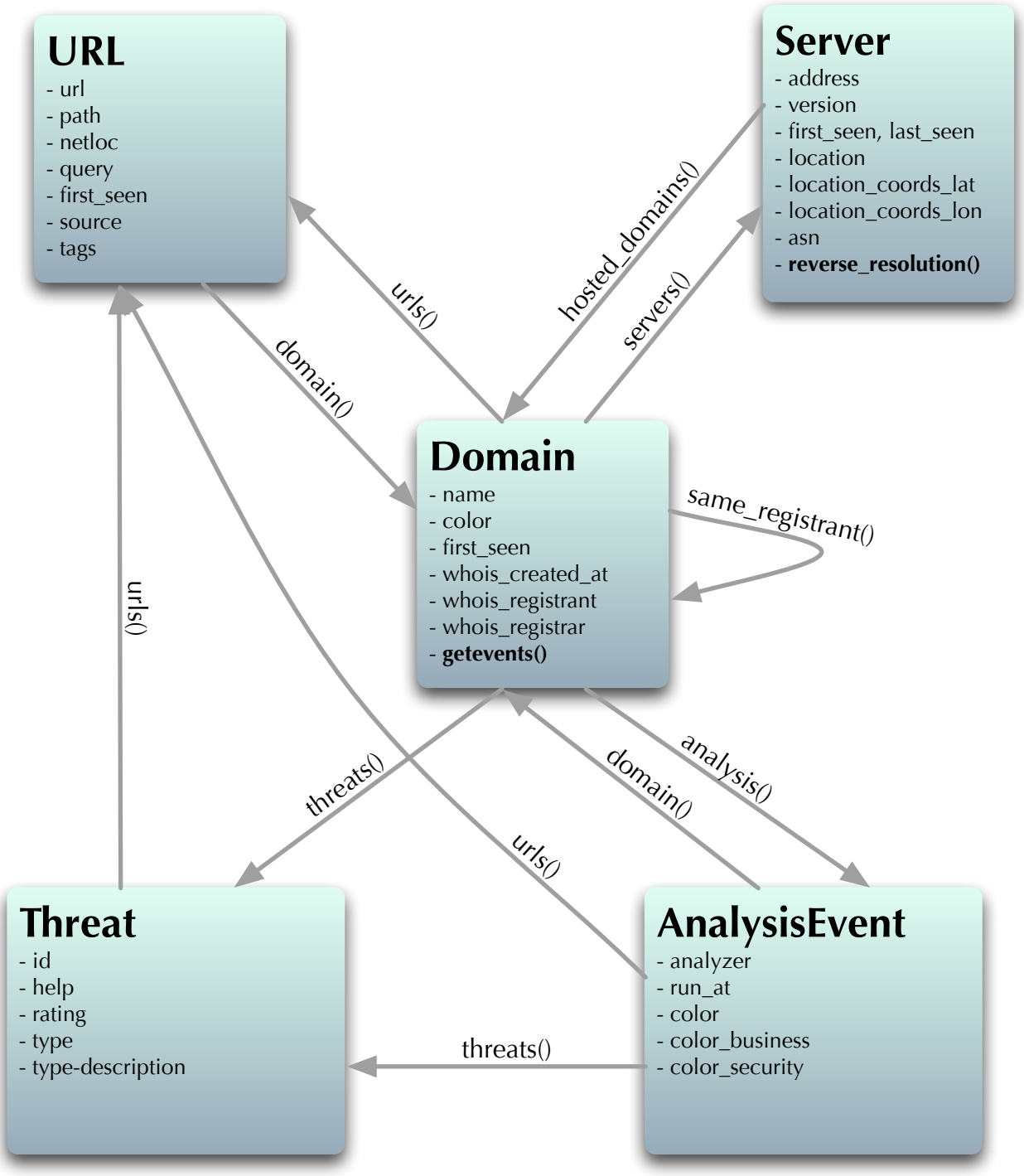
```
#What does VirusTotal say about this malware?
print virustotal.get_file(
md5 = zief_tasks[0].malware()[0].md5)[0].get_last_analysis()[0].
    av_positives_report
```

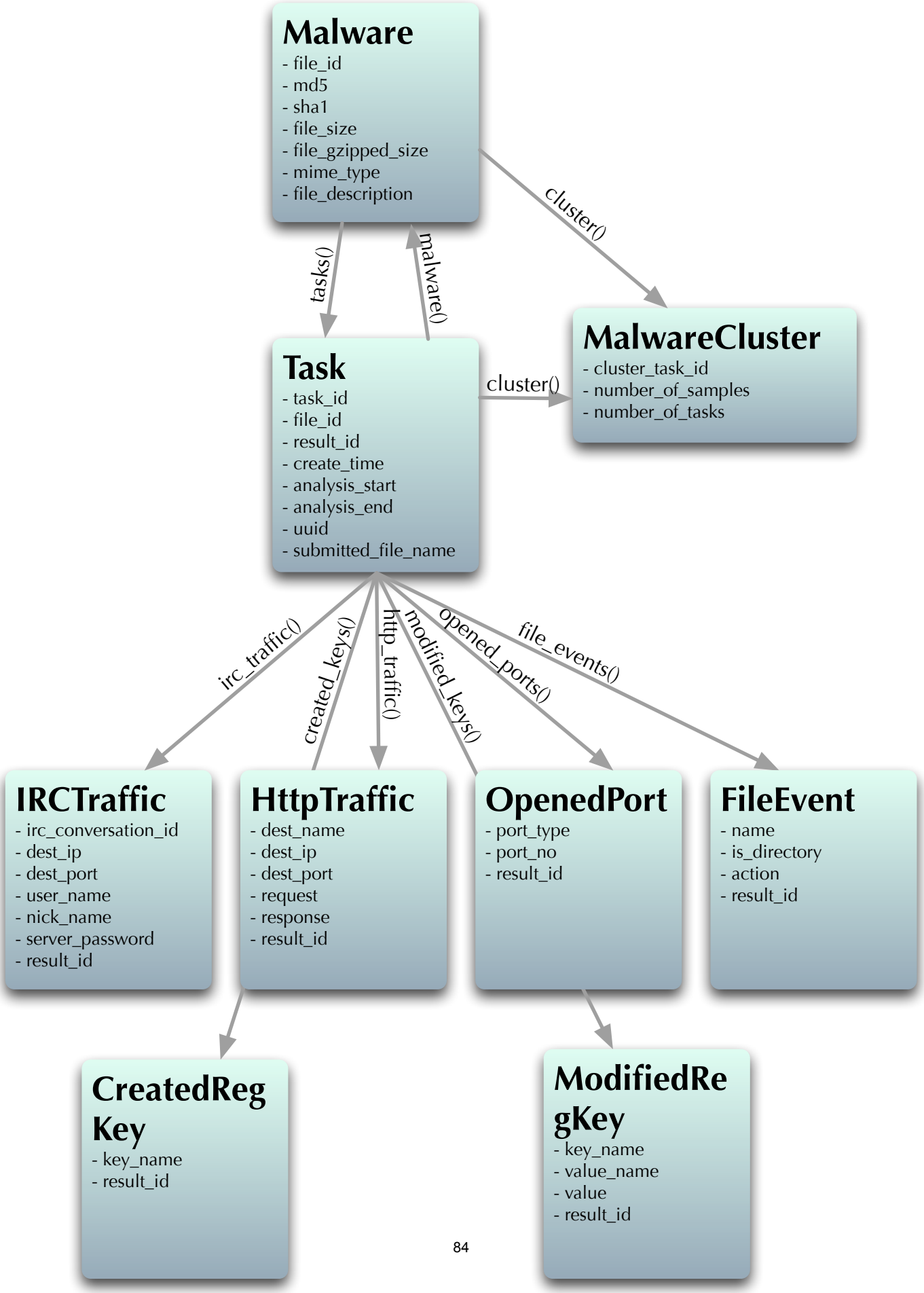
In addition, the WEPAWET report provides direct links to FIRE reports which are worth to inspect as they show the different activities (e.g., C&C, phishing) of the malicious IPs over time.

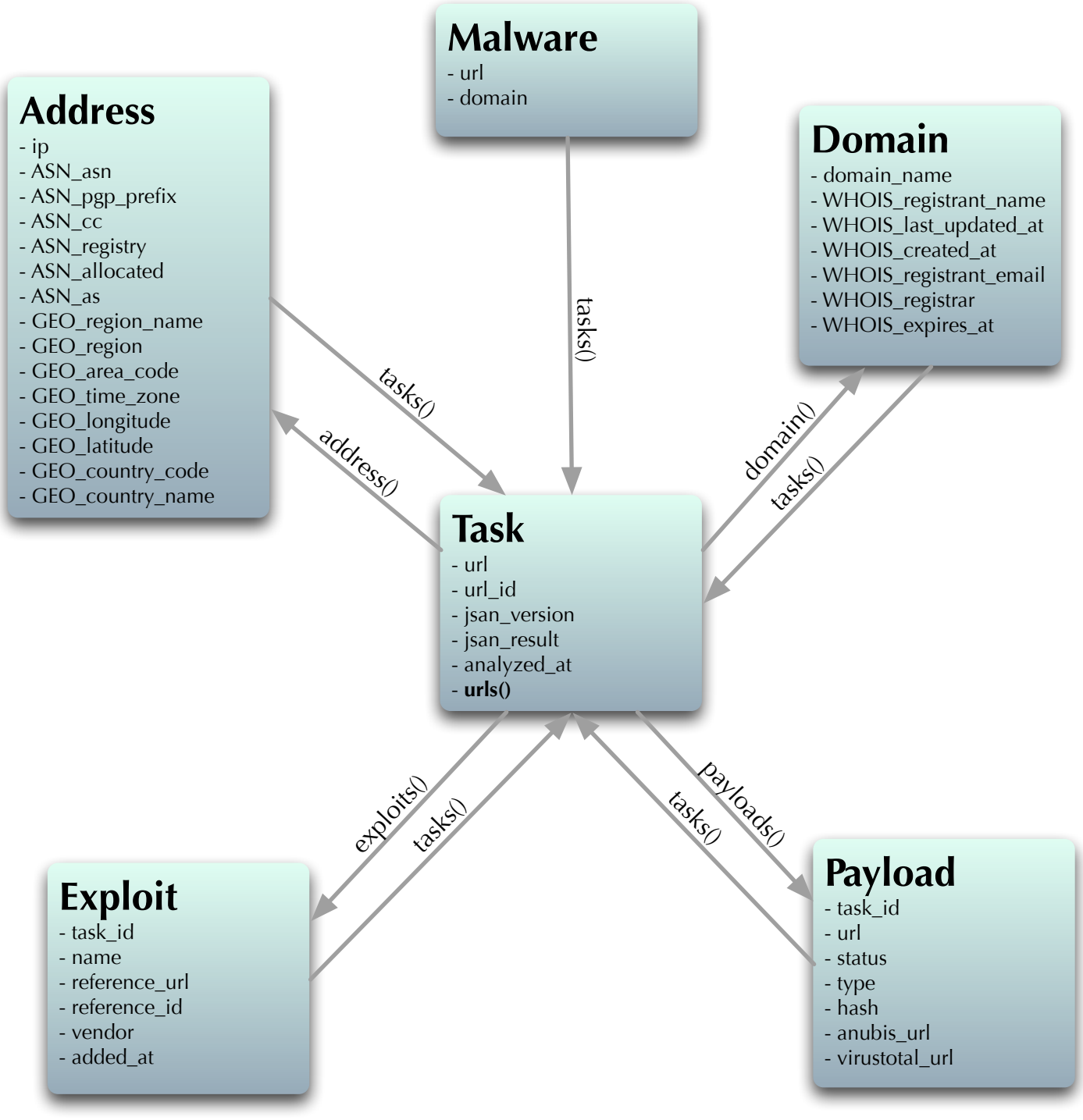
2.4 Conclusions

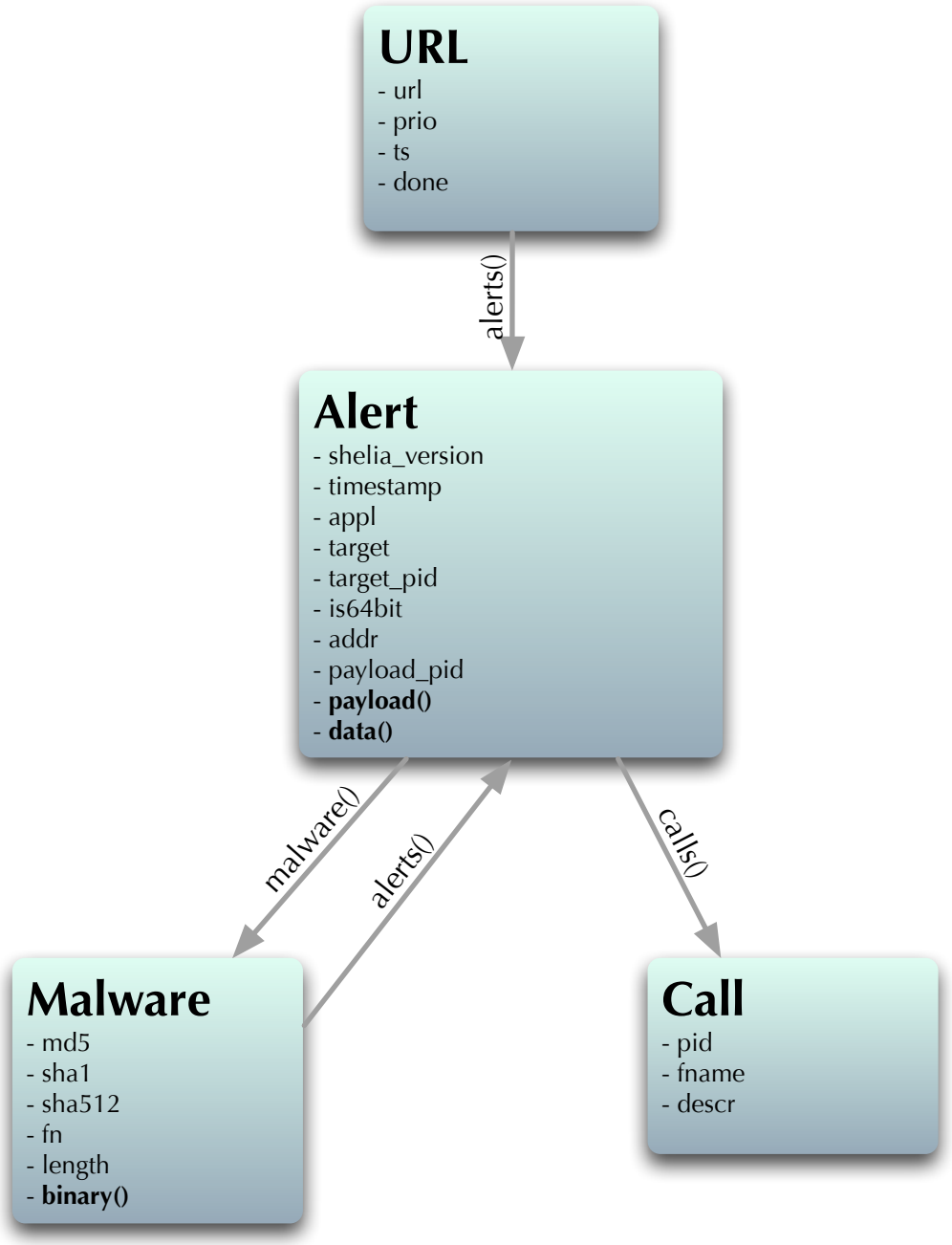
In this scenario, we have shown how it is possible to take advantage of the WAPI to investigate the security status of a certain network. We have shown how to take advantage of the different features offered by the datasets to investigate malware infections and get a better understanding of the underlying processes.

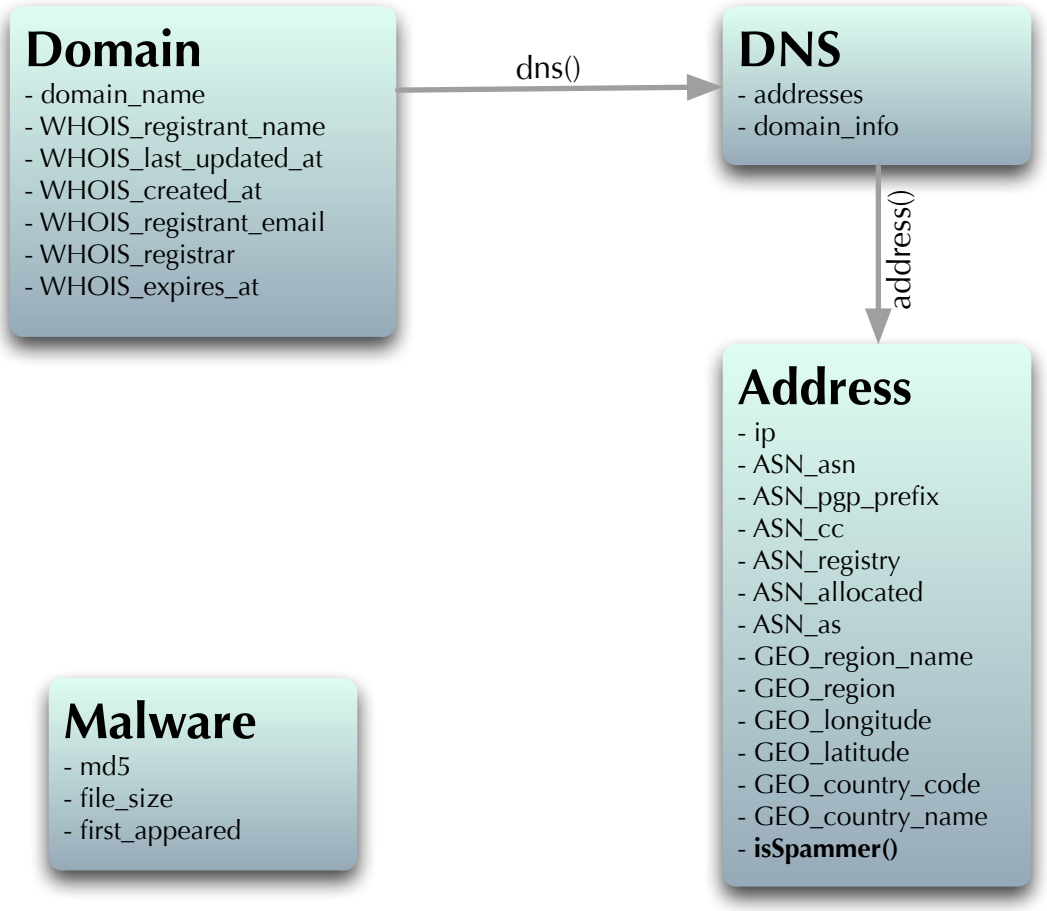

Source
- address
- ts
- getos()
Destination
- address
- name
- profile
activities()
source()
destination()
ActivityClass
- dim_epsilon
- dim_pi
- dim_mu
- timespan()
- avstats()
activity()
InjectionAttack
- src_addr, src_port
- dst_addr, dst_port
- ts_start, ts_end
- path
- download_uri
- download_protocol
- download_filename
malware()
activities()
events()
malware()
Malware
- md5
- file_size
- file_description
- corrupted
- first_appeared
- avinfo()
PEinfo
- md5
- hdr_machine
- hdr_linker_version
- hdr_nsections
- hdr_nsymbols
- hdr_nimports
- hdr_peid_packer
- sections
- imports_dll
- imports_kernel32_symbols
peinfo()

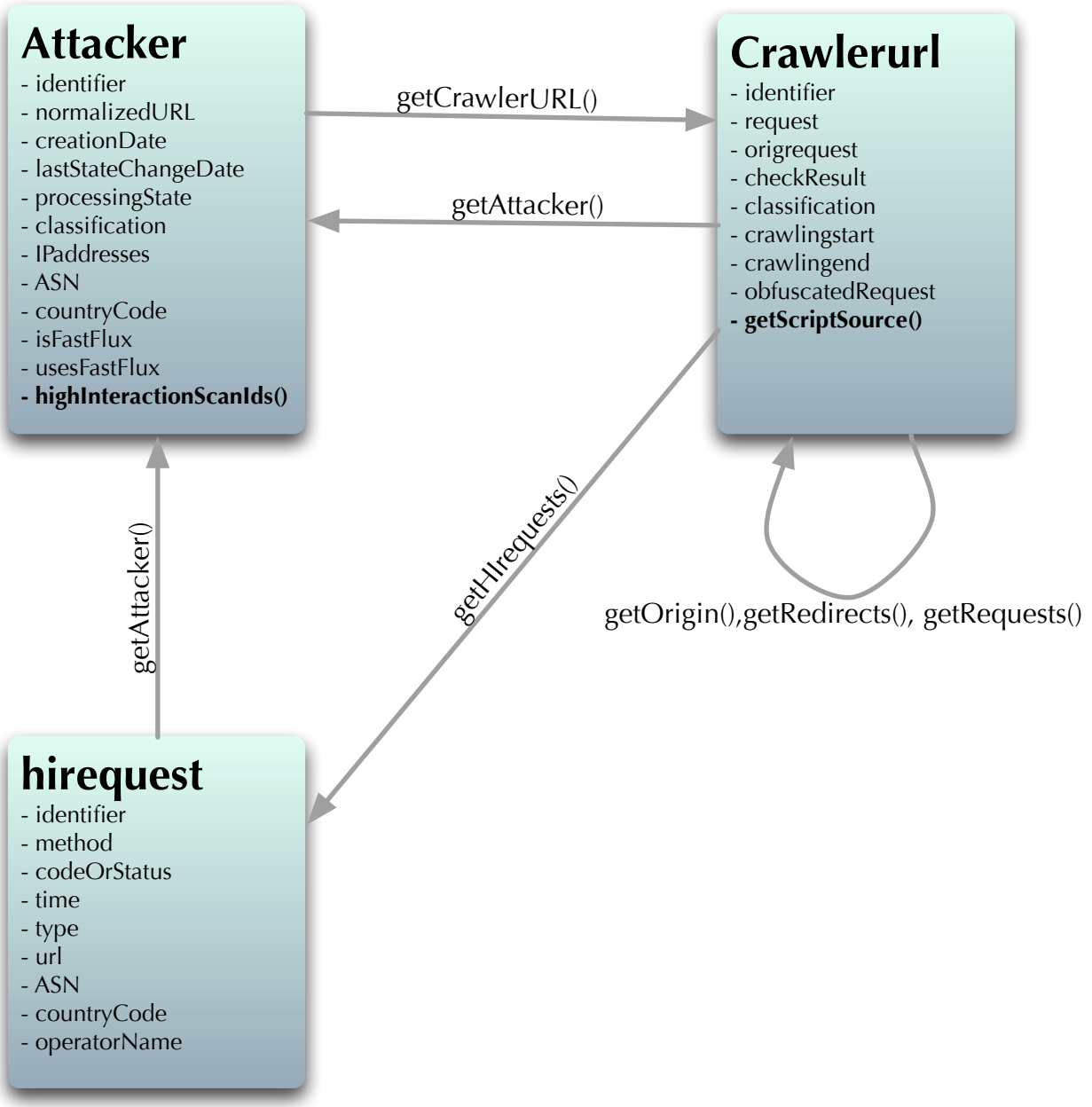

URL
- url
- path
- netloc
- query
- first_seen
- source
- tags
Server
- address
- version
- first_seen, last_seen
- location
- location_coords_lat
- location_coords_lon
- asn
- reverse_resolution()
Domain
- name
- color
- first_seen
- whois_created_at
- whois_registrant
- whois_registrar
- getevents()
Threat
- id
- help
- rating
- type
- type-description
AnalysisEvent
- analyzer
- run_at
- color
- color_business
- color_security
urls()
hosted_domains()
servers()
domain()
same_registrant()
urls()
threats()
domain()
analysis()
urls()
threats()

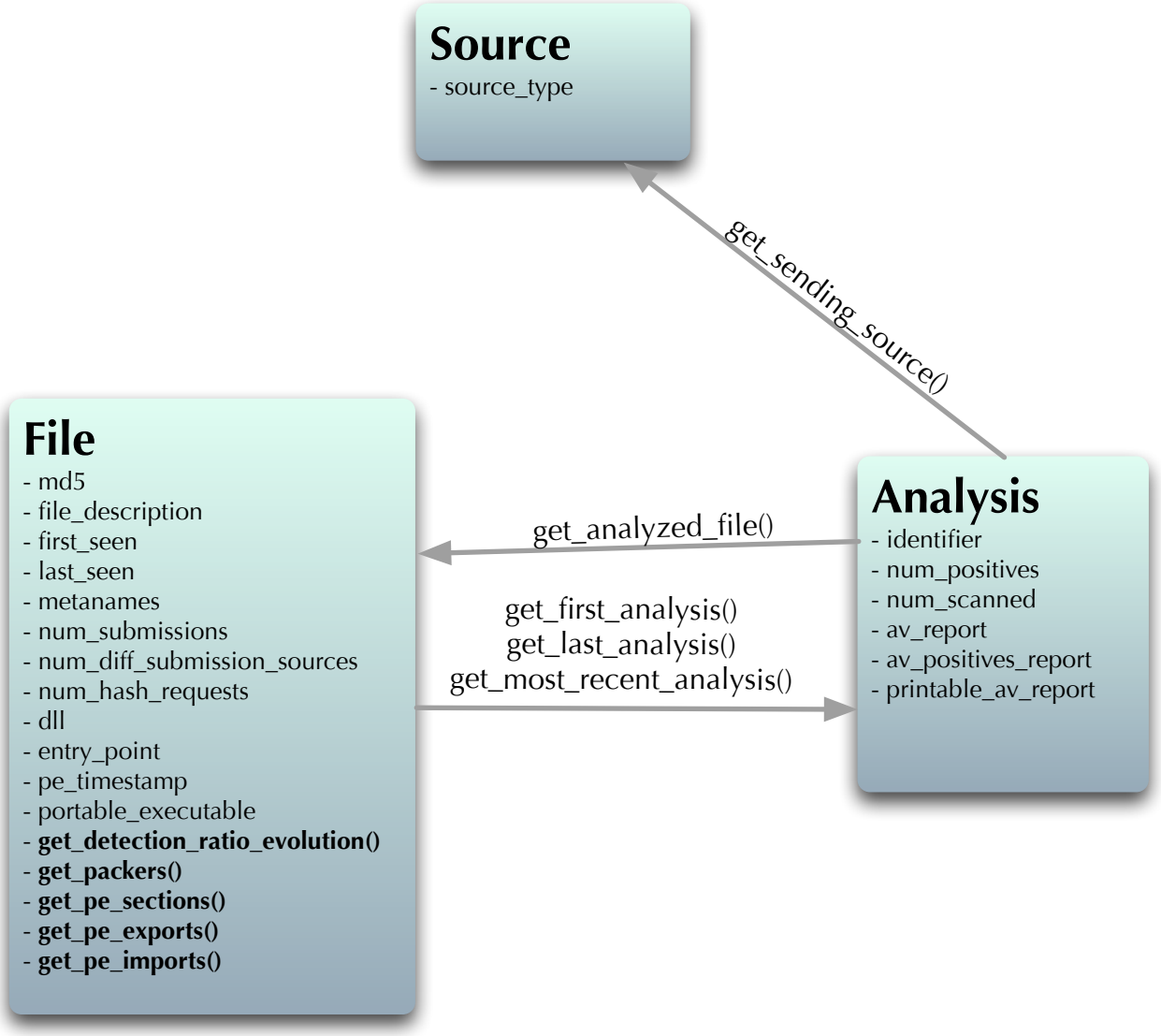

Malware
- file_id
- md5
- sha1
- file_size
- file_gzipped_size
- mime_type
- file_description
cluster()
tasks()
malware()
MalwareCluster
- cluster_task_id
- number_of_samples
- number_of_tasks
Task
- task_id
- file_id
- result_id
- create_time
- analysis_start
- analysis_end
- uuid
- submitted_file_name
cluster()
irc_traffic()
created_keys()
http_traffic()
modified_keys()
opened_ports()
file_events()
IRCTraffic
- irc_conversation_id
- dest_ip
- dest_port
- user_name
- nick_name
- server_password
- result_id
HttpTraffic
- dest_name
- dest_ip
- dest_port
- request
- response
- result_id
OpenedPort
- port_type
- port_no
- result_id
FileEvent
- name
- is_directory
- action
- result_id
CreatedRegKey
- key_name
- result_id
ModifiedRegKey
- key_name
- value_name
- value
- result_id


Malware
- url
- domain
Address
- ip
- ASN_asn
- ASN_pgp_prefix
- ASN_cc
- ASN_registry
- ASN_allocated
- ASN_as
- GEO_region_name
- GEO_region
- GEO_area_code
- GEO_time_zone
- GEO_longitude
- GEO_latitude
- GEO_country_code
- GEO_country_name
Domain
- domain_name
- WHOIS_registrant_name
- WHOIS_last_updated_at
- WHOIS_created_at
- WHOIS_registrant_email
- WHOIS_registrar
- WHOIS_expires_at
tasks()
tasks()
address()
domain()
tasks()
Task
- url
- url_id
- jsan_version
- jsan_result
- analyzed_at
- urls()
exploits()
tasks()
tasks()
payloads()
Exploit
- task_id
- name
- reference_url
- reference_id
- vendor
- added_at
Payload
- task_id
- url
- status
- type
- hash
- anubis_url
- virustotal_url


URL
- url
- prio
- ts
- done
alerts()
Alert
- shelia_version
- timestamp
- appl
- target
- target_pid
- is64bit
- addr
- payload_pid
- payload()
- data()
malware()
alerts()
calls()
Malware
- md5
- sha1
- sha512
- fn
- length
- binary()
Call
- pid
- fname
- descr


Domain
- domain_name
- WHOIS_registrant_name
- WHOIS_last_updated_at
- WHOIS_created_at
- WHOIS_registrant_email
- WHOIS_registrar
- WHOIS_expires_at
dns()
DNS
- addresses
- domain_info
address()
Address
- ip
- ASN_asn
- ASN_pgp_prefix
- ASN_cc
- ASN_registry
- ASN_allocated
- ASN_as
- GEO_region_name
- GEO_region
- GEO_longitude
- GEO_latitude
- GEO_country_code
- GEO_country_name
- isSpammer()
Malware
- md5
- file_size
- first_appeared


Attacker
- identifier
- normalizedURL
- creationDate
- lastStateChangeDate
- processingState
- classification
- IPaddresses
- ASN
- countryCode
- isFastFlux
- usesFastFlux
- highInteractionScanIds()
Crawlerurl
- identifier
- request
- origrequest
- checkResult
- classification
- crawlingstart
- crawlingend
- obfuscatedRequest
- getScriptSource()
hirequest
- identifier
- method
- codeOrStatus
- time
- type
- url
- ASN
- countryCode
- operatorName
getCrawlerURL()
getAttacker()
getAttacker()
getHIrequests()
getOrigin(),getRedirects(), getRequests()


Source
- source_type
get_sending_source()
File
- md5
- file_description
- first_seen
- last_seen
- metanames
- num_submissions
- num_diff_submission_sources
- num_hash_requests
- dll
- entry_point
- pe_timestamp
- portable_executable
- get_detection_ratio_evolution()
- get_packers()
- get_pe_sections()
- get_pe_exports()
- get_pe_imports()
get_analyzed_file()
get_first_analysis()
get_last_analysis()
get_most_recent_analysis()
Analysis
- identifier
- num_positives
- num_scanned
- av_report
- av_positives_report
- printable_av_report