

---

# Classifying Threats

Clustering malware with ANUBIS and SGNET

Paolo Milani Comparetti

[pmilani@seclab.tuwien.ac.at](mailto:pmilani@seclab.tuwien.ac.at)

Corrado Leita

[corrado\\_leita@symantec.com](mailto: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 analyzed >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 Allapple 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

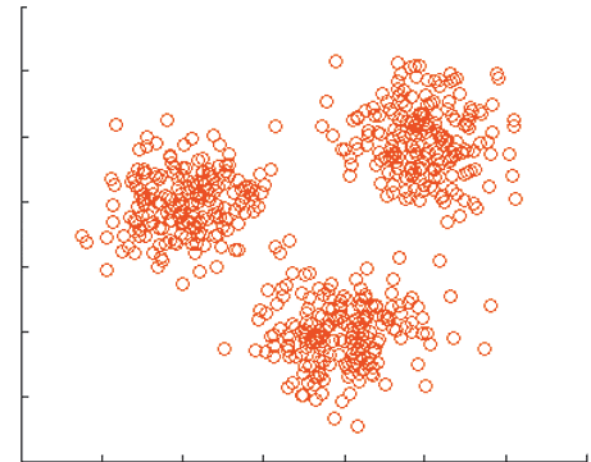
---

- 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

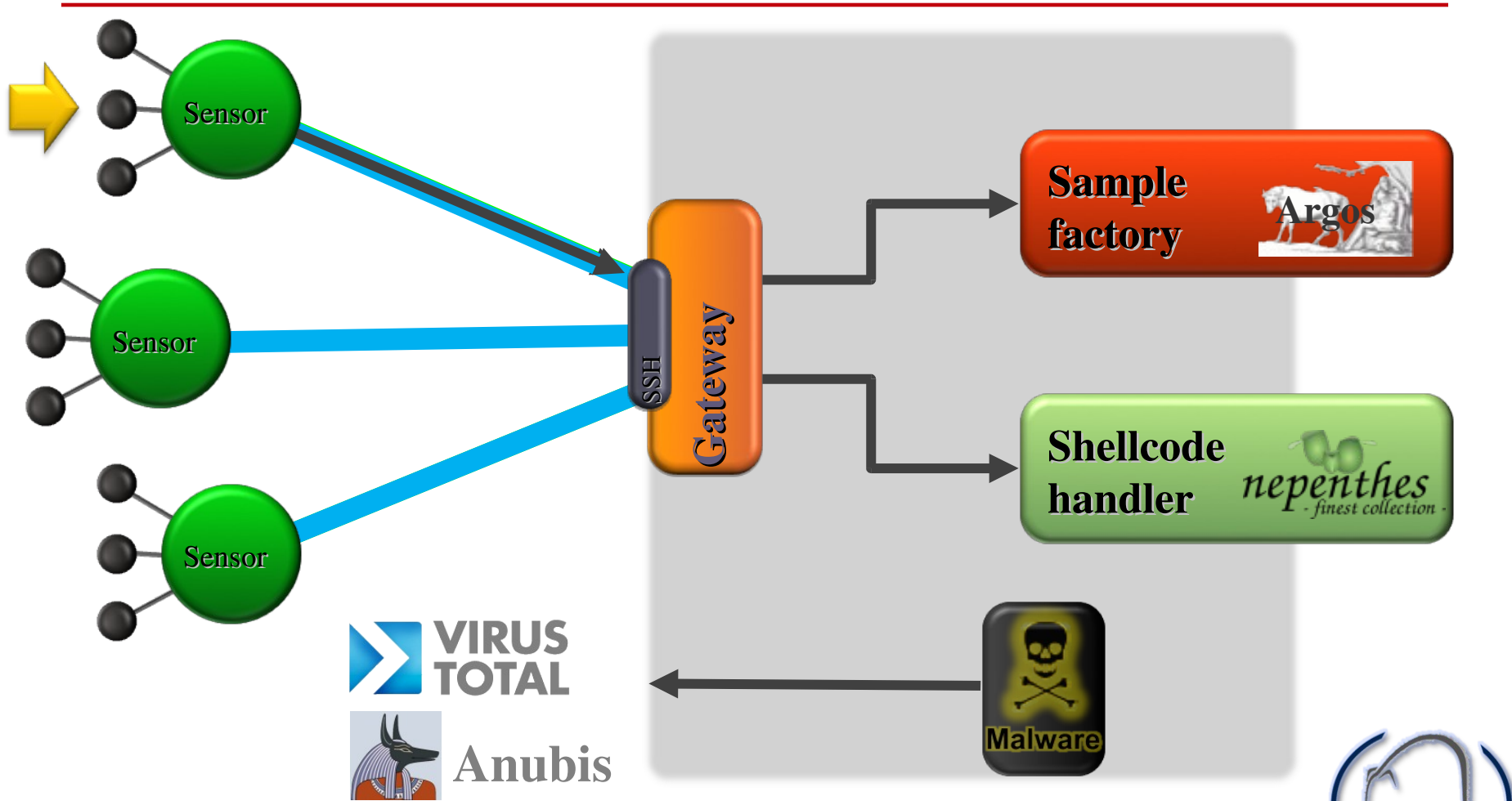
---

- 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

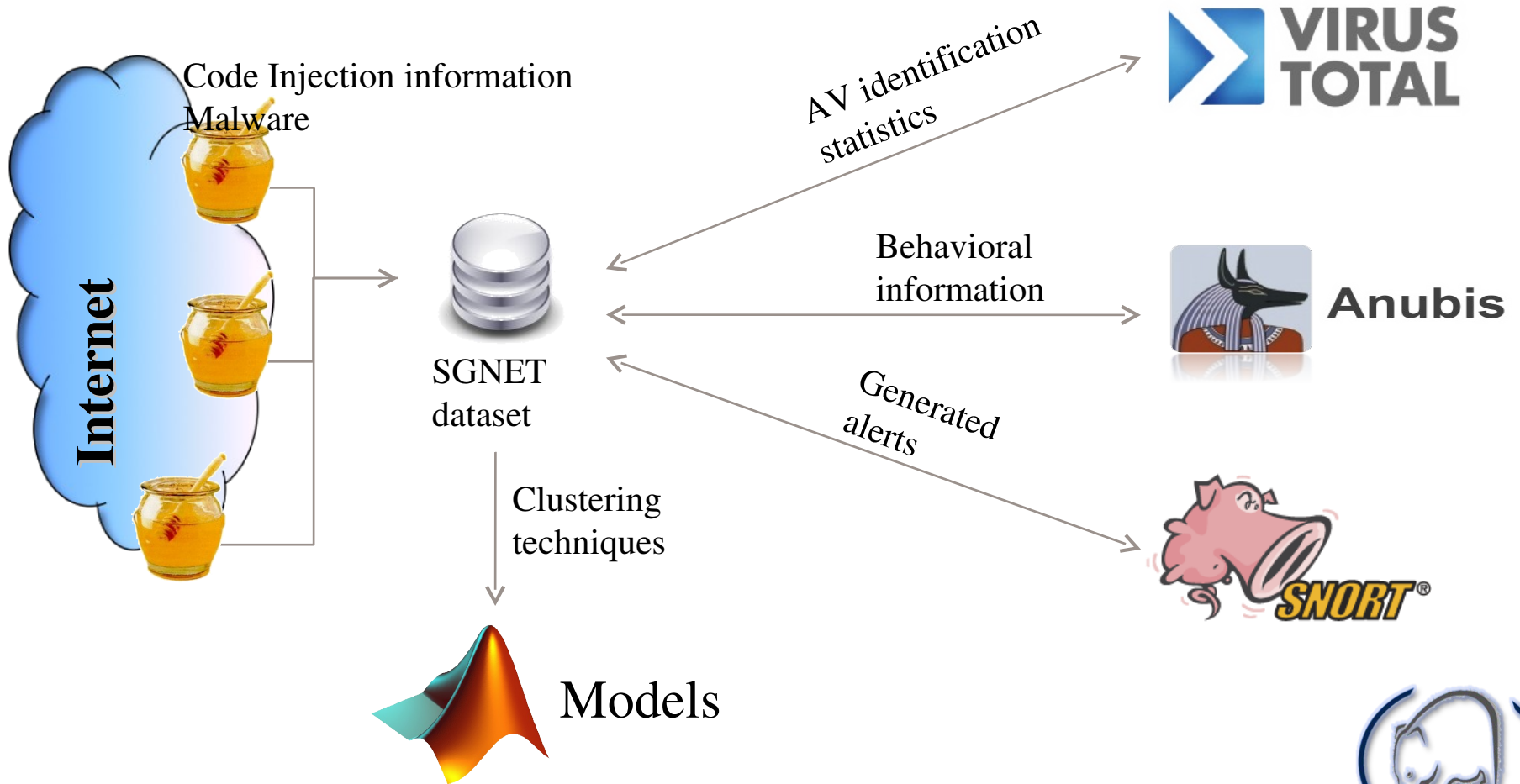
---

# SGNET EPM Clustering

# SGNET Operation



# SGNET data enrichment framework

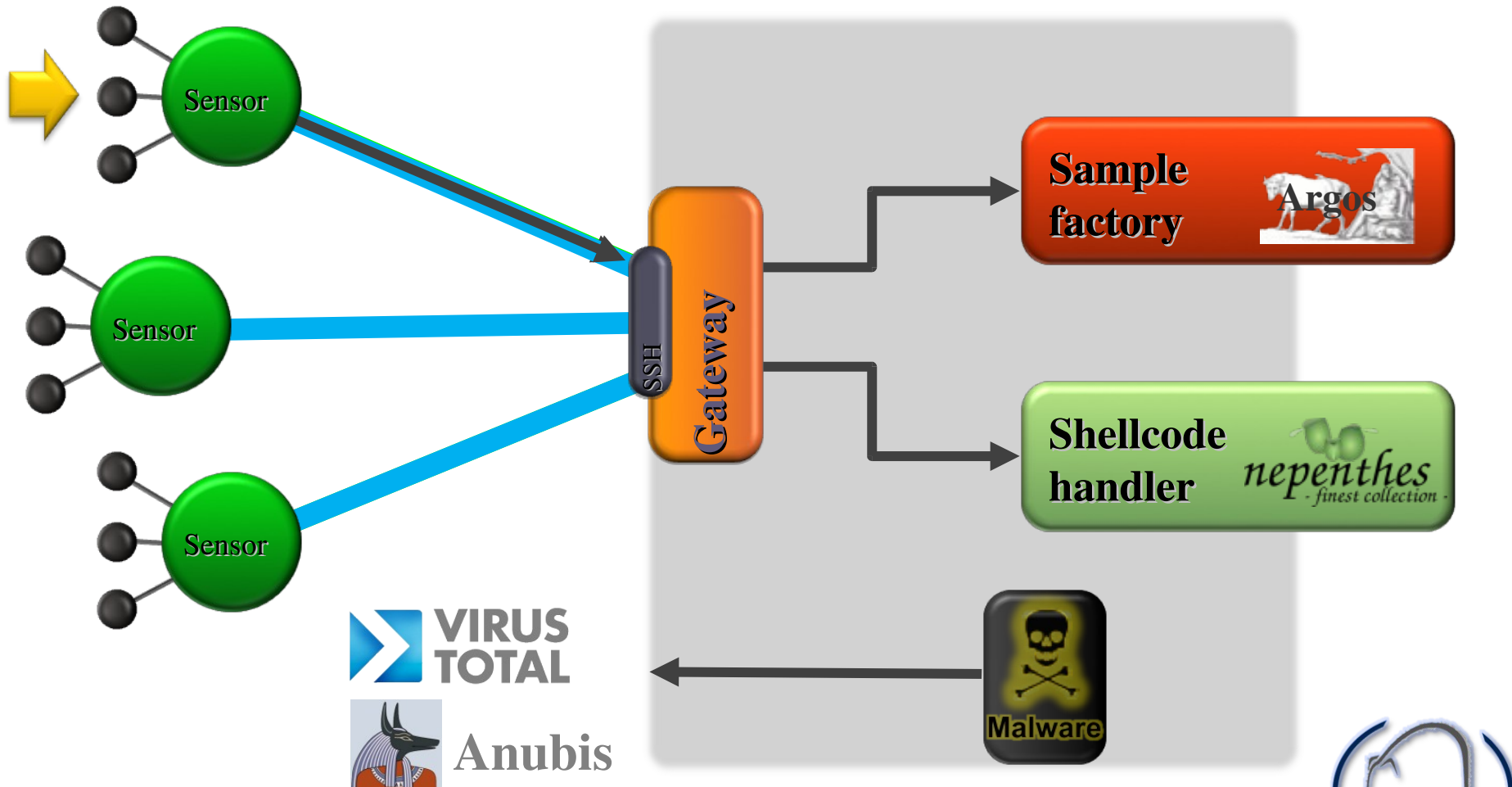


# SGNET and Code Injections

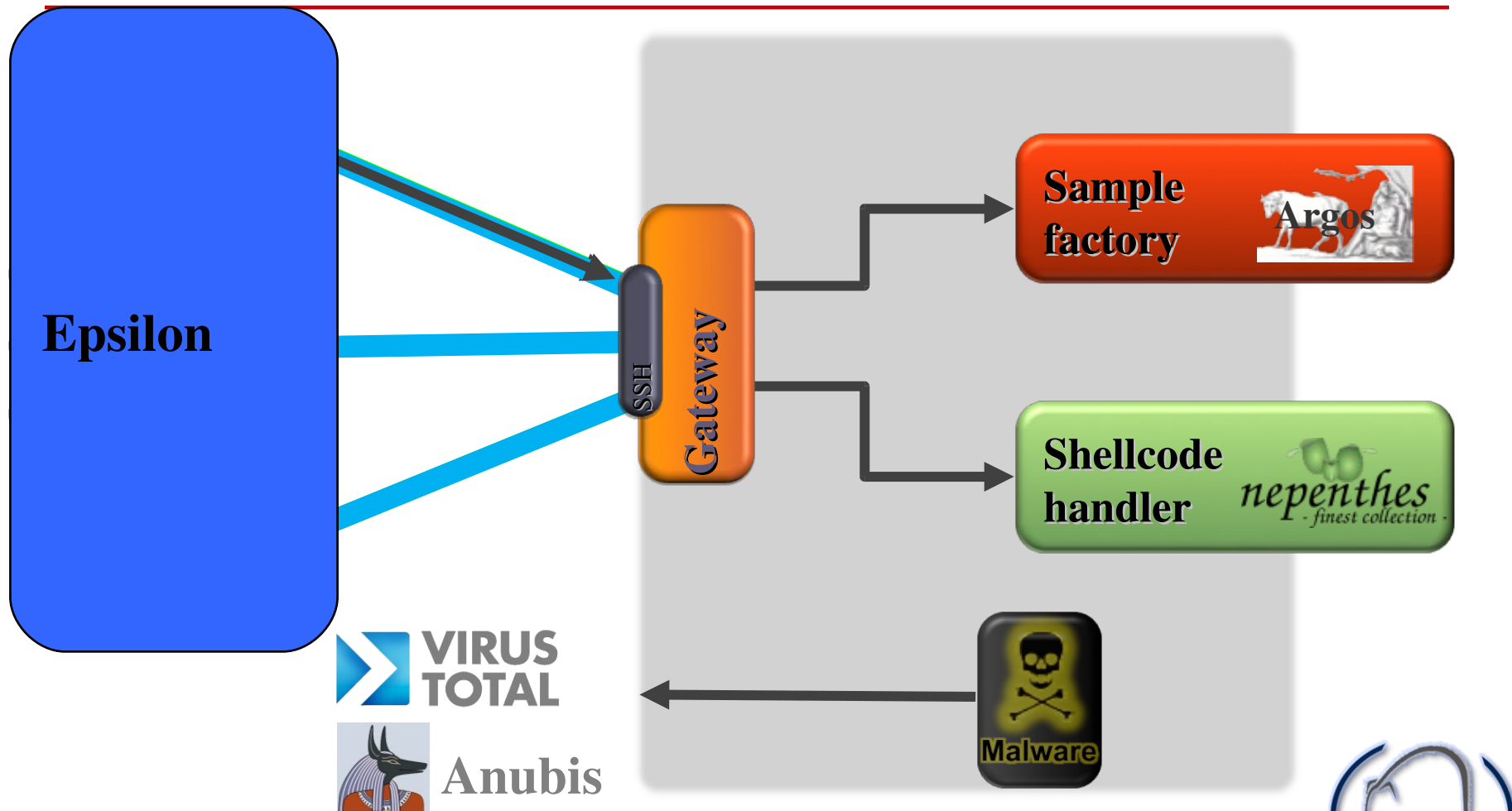
---

- 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

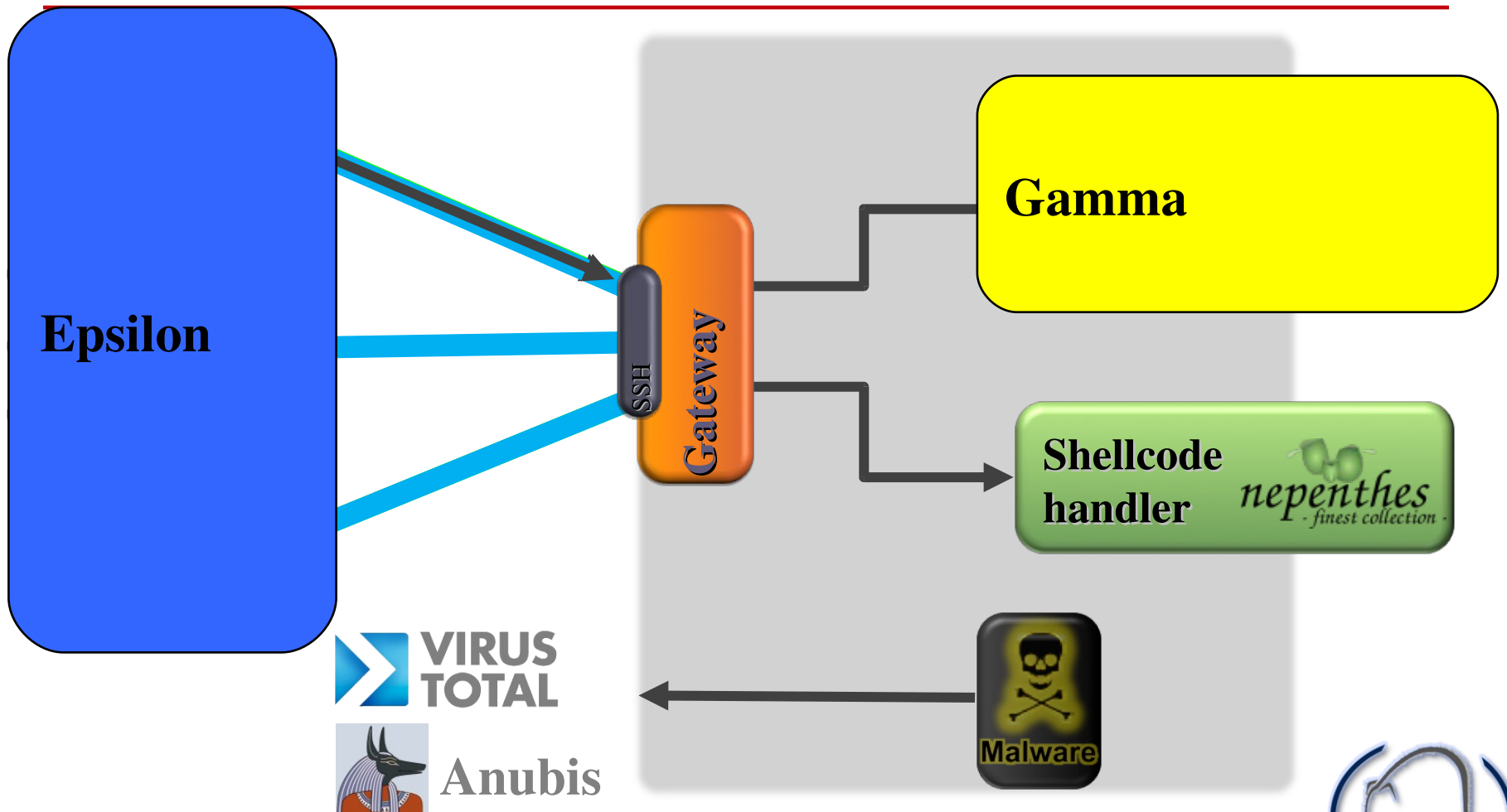
# SGNET and Code Injections



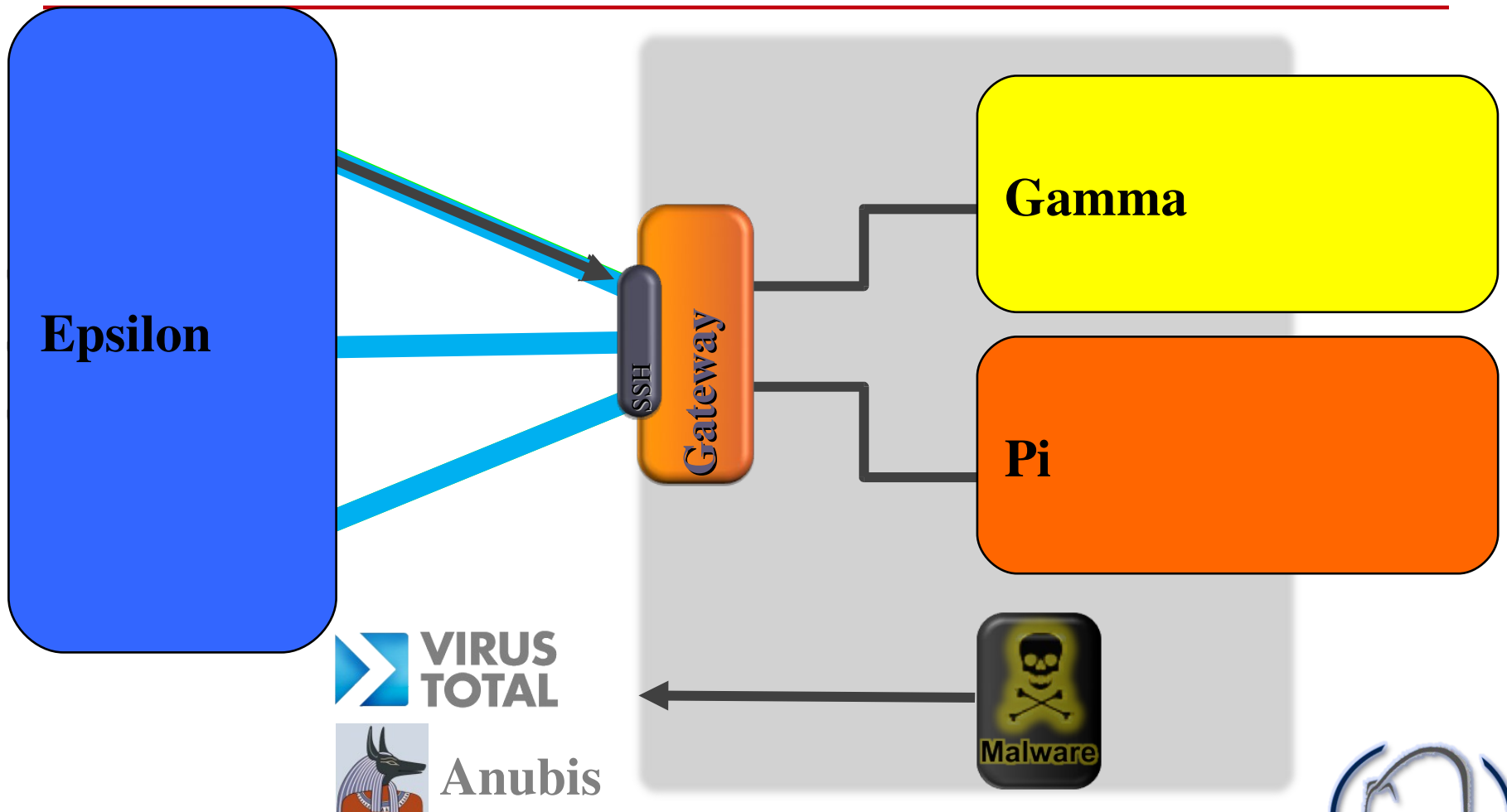
# SGNET and Code Injections



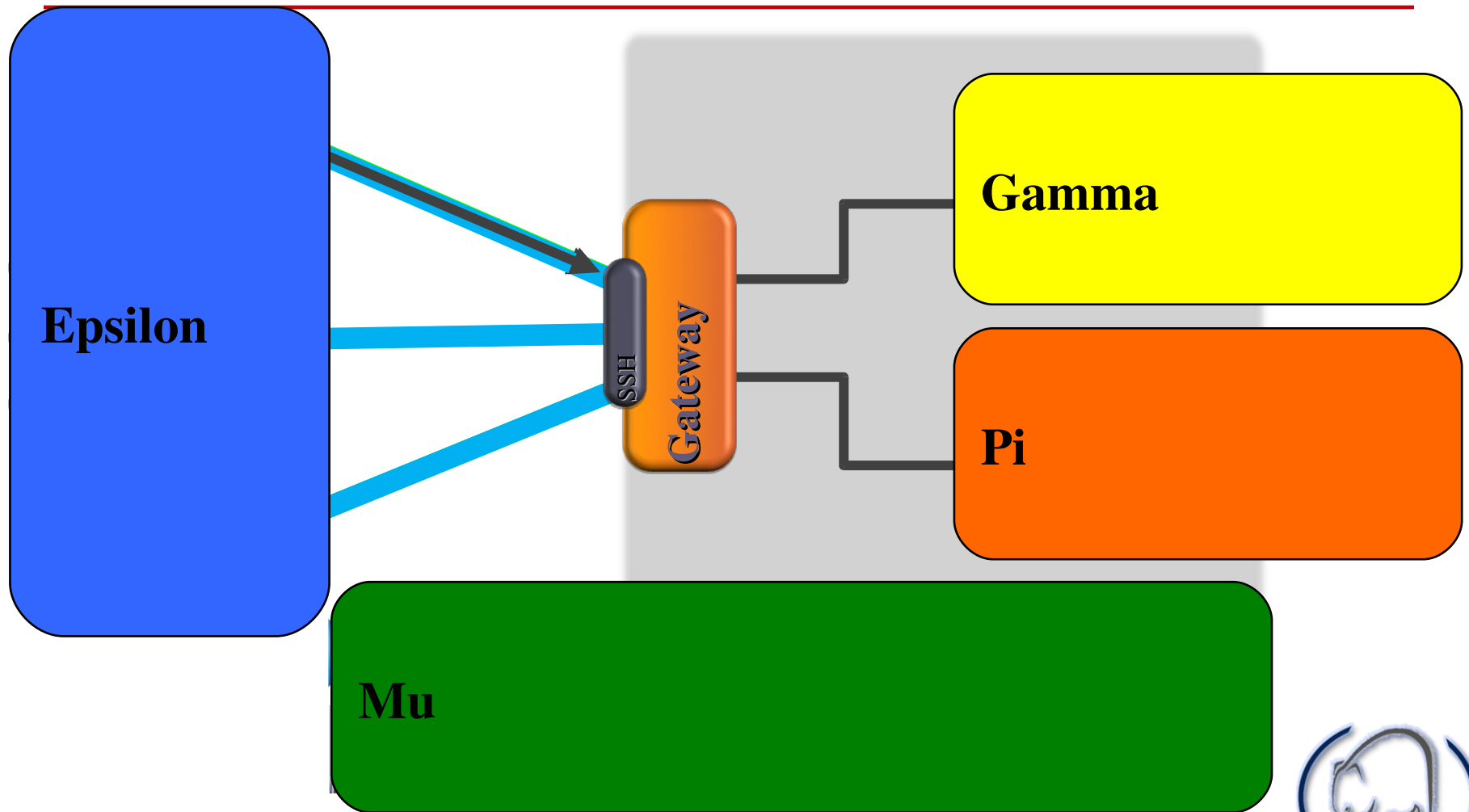
# SGNET and Code Injections



# 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
XXXXXX	32678	3	data
YYYYYY	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

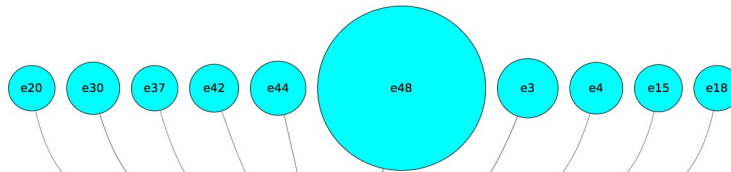
# 2 Complementary Approaches

---

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

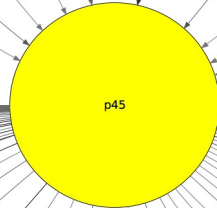
# Example 1: dynamic “wins”!

Epsilon

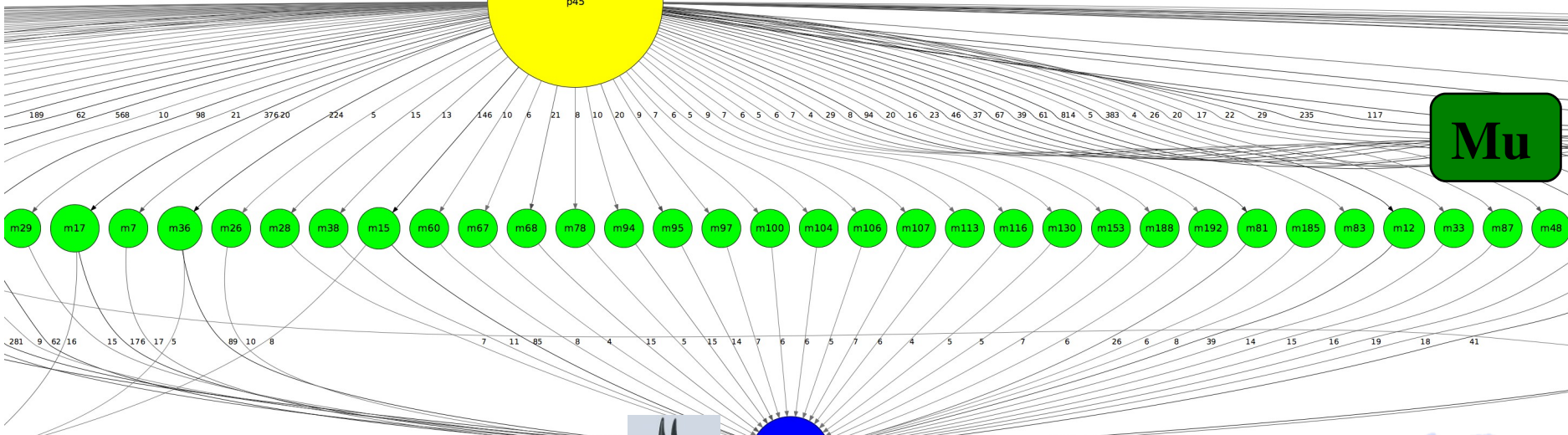


Rahack (aka Allapple) propagation

Pi

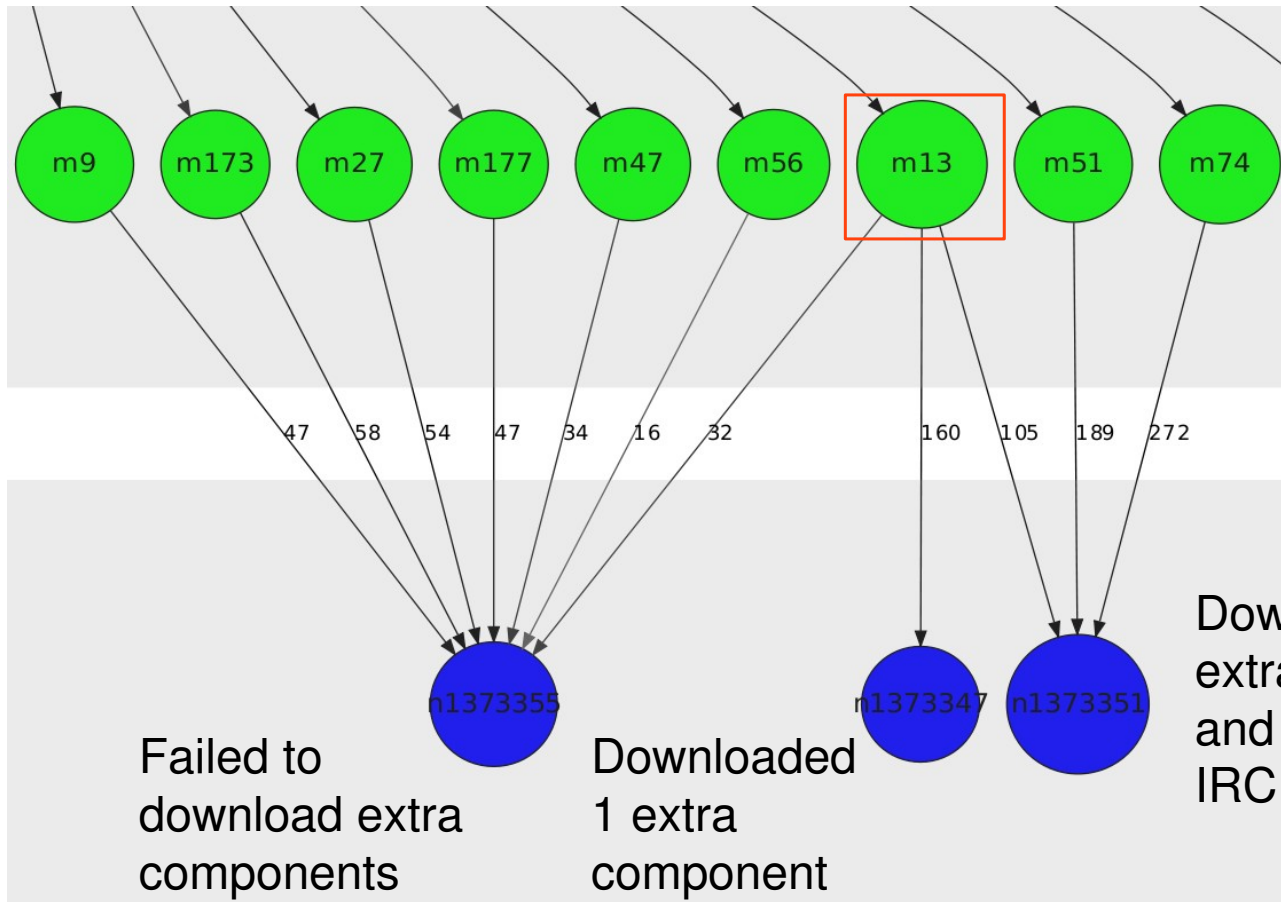


Mu



# Example 2: static “wins”

**Mu**



# 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 existence
    - Distribution of the sources
- Other malware types sharing the same E,P tuple exist, but have different “M”-signatures and context!

# References

---

- TTAalyze (precursor to Anubis):
  - U. Bayer, C. Kruegel, and E. Kirda, “TTAalyze: 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?