H. A. T. E.

Honeypot - based

Analysis

Threat

Engine

What is **H.A.T.E.**?

General description

- A modular, distributed engine to perform **live** threat analysis
- You can use it as an advanced high-interaction honeypot or as a dynamic analysis engine system
- We target Linux threat analysis and we are porting it to handle Windows systems too

H.A.T.E. requirements

- KVM
- Hardware requirements depend on how massively the core is deployed
- The core runs on Linux systems
- The system’s modularity allows, if required, to easily add an API to support particular clouds (OpenStack, VMWare, etc.)

H.A.T.E. supported sandboxes

- Actually, Linux
- WIP: Windows

H.A.T.E. logging capabilities

- Real time monitoring of the entire environment
- Persistent data logging accessible via an API
- Recovery of malicious infected code
- WIP: the possibility to generate a detailed report about the malware behaviour:
  - Function/system calls logging
  - Network activity
  - File system history (e.g. track read/write/delete)
  - Record user activity (e.g. on Linux, command line recording)

Why do you need **H.A.T.E.**?

- It is a perfect solution for the **prevention and deep analysis** of perimetral network attacks
- We executed the software in a big network (almost 40k allocated IPs) and we observed a lot of threats, mostly malware agents targeting Linux based machines/IoT devices
- Solution? Observe the malware's behaviour into sophisticated virtualized/containerized environments to explore their attack surface

What does **H.A.T.E.** gather?

- H.A.T.E. deploys various agents into and outside the sandboxed environments
- Each agent has a purpose:
  - Network monitoring
  - File system monitoring
  - Process dynamic binary instrumentation
  - Human interaction logging
  - Signature generation
- **Eventually, using the filesystem probe, malicious binaries injected in the sandbox can be recovered.**

How is **H.A.T.E.** done?

- The project is highly modular
- We dispatch the work to various units
- This process of connecting units forms a kind of pipeline, and the user can even interact in the middle of it
- This makes the project highly configurable
- E.g. it is possible to inject malicious binaries disabling the honeypot functionalities

How does **H.A.T.E.** work?

The Pipeline

1. We expose some services (SSH, TelNet, etc.)
2. Each time an attacker connects to one of them, we build up at runtime a sandboxed environment (containerized/virtualized)
3. Based on H.A.T.E. configuration for that service, we inject the desired probes into the sandbox
4. All the traffic from the attacker's IP is redirected to the sandbox
5. H.A.T.E. gathers all the information from the probes and generates reports and signatures (using third party AV APIs) for the identified threat
6. Then, the polluted environment gets destroyed

Prototypal **experiments**

- A prototype has been tested by distributing it inside a /16 network (i.e. a campus network)
- Thousands of attacks targeting various protocols (SSH, FTP, TelNet, etc.) have been detected
- The prototype captured ~**30** infected binaries in a few days of activity
- Those malware are often written targeting multiple architectures (x86/64, ARM, MIPS, etc.)

Related work

- We published a paper which describes a more “ancient” prototype called H.E.R.E.Sy. *Honeypot-powered Malware Reverse Engineering, Proc. of 3rd IEEE International Symposium on Software Defined Systems (SDS-2016).*