Evaluating Low Interaction Honeypots and On their Use against Advanced Persistent Threats

ABSTRACT

In this paper we evaluate several Low Interaction Honeypots (LIHs) according to several usability and performance criteria. Furthermore we argue on the utilization of LIHs that could indicate early signs of jeopardy from Advanced Persistent Threats (APT).

Keywords

Low-interaction honeypots; Usability, Performance, APTs

1. INTRODUCTION

A big challenge for information systems is analyzing malicious code behavior [10]. As the dependence of our society on critical infrastructures in the form of Industrial Control Systems (ICS) increases, a whole new underground economy flourishes [25]. As a result, malicious code is becoming more and more sophisticated. For example, Advanced Persistent Threats (APT) such as Stuxnet, Duqu, Flame, and Gauss are considered the cutting edge of malware [18] and, as such, they present exceptionally high complexity, are extremely targeted and have advanced stealth capabilities [27].

Honeypots are information systems, either physical machines, virtual machines or software emulated services [5] whose value derives from unauthorised or illicit use by adversaries [6]. Their passive, deceptive nature's main target is to gather information about the attacks they receive [4]. They can be categorized in various ways. Depending on the deployment motive, into Research and Production Honeypots [9]; regarding the level of interaction provided to the attacker, into High Interaction and *Low Interaction Honeypots* [11]; considering the hardware deployment type, into Physical and Virtual Honeypots [7]; finally, regarding the role of the honeypot, into Server Side and Client Side Honeypots [16].

Our Contribution. A recent evaluation, conducted by ENISA [24], provided some insight regarding the best honeypot software solution, suitable for usage by a CERT team. In this paper we extent that study by assessing state-of-the-art honeypot software solutions in terms of a set of usability and performance criteria, according to the latest SSI standards [29]. In addition, we argue on the utilization of LIHs that could indicate early signs of jeopardy from APTs.

2. LIH SOFTWARE EVALUATION

2.1 Criteria

First we will review the criteria which will be taken into account for our evaluation [20]:

1. Usability. The main factors that are considered regarding the usability of honeypots are:

- **Understandability.** This factor is defined by the level of difficulty a user faces in order to: understand the purpose of a particular honeypot, but also its basic and advanced functions; establish its design rationale and the availability of an architectural overview; get an initial set of case studies.
- **Documentation.** The factors taken into consideration for this evaluation are: Availability of documentation on the project site; the quality of the available documentation, its completeness, accuracy and clarity; assumptions that are made about the background expertise of the reader; the task-orientation of the documentation; whether it gives examples of what the user can see at each step.
- **Buildability/Installability** Important factors here are: Difficulty of meeting the prerequisites for building/installing the software on a target platform; availability of instructions at the project's site for building/installing the software; existence of an automated build(*e.g.*, Make); documentation of all mandatory third-party dependencies; use of dependency management to automatically download dependencies; provision of tests to verify the success of the build/installation; availability, for all source and binary distributions, of a README.TXT with project name, web site, how/where to get help, version, date, licence etc; availability of an installer and of an automatic uninstaller.
- **Learnability** The aspects considered are: straightforwardness of achieving basic functionality; availability of a getting-started guide; instructions supporting all use cases; availability of API documentation for user-developers.

2. Performance. Performance in a LIH is measured in terms of the following factors: Number of collected malware samples in a given deployment period; the time the system continues to function properly; number of concurrent sessions the honeypot can handle; quality of collected metadata; number of available emulated services.

2.2 Dionaea

Dionaea (http://dionaea.carnivore.it) is one of the most complete and highly productive honeypots used in this study, mainly because of the advanced emulation of the vulnerable services it mimics. Its evaluation is depicted in Fig. 1.

Figure 1: Dionaea honeypot evaluation

Understandability *(Fair).* Dionaea's functionality is well described in the project's site but could be more analytic concerning its advanced functions. The design rationale is available but case studies are not present.

Documentation *(Poor).* The documentation at Dionaea's website and blog do not seem to have been updated for long period of time.

Buildability *(Poor).* Dionaea depends on numerous software prerequisites, all of which are third party and need to be built separately. An automated build is provided and all dependencies were available at the time of writing.

Installability *(Fair).* By following the instructions on the projects website, installation was successful with no problems. The third party dependencies were available and a test to verify installation success was provided. An installer is not available, neither an automatic uninstaller.

Learnability *(Poor).* Dionaea has no graphical interface and long commands are required for the system to start functioning. Configuration is available through one configuration file that is not easily understood by the average user. There are no use cases provided, nor API documentation for developers.

Malware Samples Quality *(Excellent).* The quality of samples is exceptional and are maintained in a specific folder for further analysis named after the hash they produce, for ease of manipulation.

Malware Samples Quantity *(Excellent).* Dionaea is able to collect vast amounts of malware samples, mainly because of the number of services it can emulate (see below).

Uptime *(Excellent).* The system remained functional during the three days it was tested and no malicious communication seemed to disrupt its working state.

Concurrent Sessions *(Excellent).* Dionaea is able to listen on numerous network interfaces and IP addresses.

Metadata Quality *(Fair).* All information Dionaea collects are stored in a logfile and in SQlite database, allowing easy manipulation of the data. Third party tools such as P0f can be integrated with Dionaea in order to provide additional information about the attacks.

Emulated Services *(Excellent).* The vulnerable services it emulates are numerous, *i.e.*, *SMB, HTTP, FTP, TFTP, MSSQL, MySQL, SIP*, and their emulation is quite realistic.

2.3 Honeyd

Honeyd [8] is a framework to instrument multiple, unallocated Internet addresses on an existing network with virtual honeypots and corresponding network services. For each IP address, we can tell Honeyd how we want the simulated computer to behave. Honeyd is not designed to collect malware samples but to collect statistics about malware attacks. The project has not been updated since 2007 but still remains functional and provides invaluable services to the research community. Its evaluation is depicted in Fig. 2.

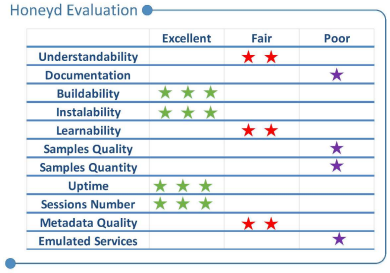

Figure 2: Honeyd evaluation

Understandability *(Fair).* Honeyd's purpose of the tool is clear and basic functionality is well defined. Advanced functions are not explained thoroughly, however numerous configurations possibilities are offered. The website provides thorough functionality and design rationale.

Documentation *(Poor).* Honeyd's documentation provided in the project's website is introductory and thus incomplete. Note however that software's man-pages are precise and clear, portray accurately the different commands Honeyd can process but are not enough. A commercial book was also written by the developer [12].

Buildability *(Excellent).* Honeyd's prerequisites are not difficult to meet and the different releases are available for downloading in the project's website.

Installability *(Excellent).* In most cases a simple command is sufficient for downloading and installing the honeypot without further interaction from the user.

Learnability *(Fair).* Its basic functionality is easily under-

stood but given the fact that Honeyd could emulate entire networks with blocks of IP's and network devices, advanced functionality is fairly trivial.

Malware Samples Quality and Quantity *(Poor)*. Honeyd is not designed to collect malware samples.

Uptime *(Excellent)*. Honeyd is a very reliable honeypot and can be functioning unattained for long periods of time with the basic configuration.

Concurrent Sessions *(Excellent)*. With the default configuration Honeyd can handle with no problem up to 100 IP addresses.

Metadata Quality *(Fair)*. Honeyd gathers only basic information about the incoming connections and stores them in a log file.

Emulated Services *(Poor)*. Emulation of services listening on the ports Honeyd monitors depends on python scripts, the most basic of which come with the honeypot. The level of interaction these scripts provide is basic and most advanced emulation can be achieved but must be implemented by the user.

2.4 Amun

Amun is a LIH available under the GNU Public Licence. It is developed for the Unix platform and was available, at the time of writing, for download. Amun, like Dionaea, emulates a wide range of vulnerabilities in order to acquire samples from autonomous spreading malware. Amun is written in Python and has a modular design [14]. Its evaluation is depicted in Fig. 3.

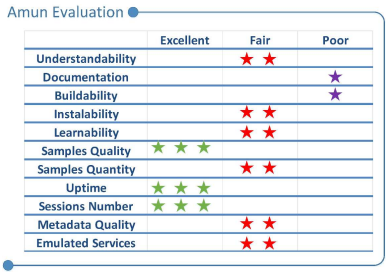

Figure 3: Amun evaluation

Understandability *(Fair)*. Amun's design rationale and architectural overview is available on-line at the time of writing [14] but case studies are not available.

Documentation *(Poor)*. The only available documentation is a text file with instructions on how to run the honeypot that is contained in Amun's .tar file.

Buildability *(Poor)*. No instructions are available for building the honeypot in the project's site. There is no dependency management provided even though the dependencies are available at the time of writing. Furthermore, no tests are provided for verifying the success of the build.

Installability *(Fair)*. The only prerequisites are the Python programming language and Python Psyco, a library currently available on-line, but unmaintained since March 2012.

Learnability *(Fair)*. Amun's basic functionality can be achieved only by starting the server.py file and waiting for the outbound communication. More advanced functionality can be achieved through the configuration file.

Malware Samples Quality *(Excellent)*. The quality of malware samples Amun captures is high. Additionally to the binaries it collects during interaction, Md5 sums are generated.

Malware Samples Quantity *(Fair)*. The modular design of Amun does not allow the collection of zero-day exploits limiting the quantity of threats detected.

Uptime *(Excellent)*. Amun was running uninterrupted for long periods of time during evaluation.

Concurrent Sessions *(Excellent)*. A single installation of Amun can easily handle over 100 IP's. Several Amun servers can run simultaneously on the same host, limited only by the fact that one server can bound to a specific port.

Metadata Quality *(Fair)*. According to the level of interaction and the type of the emulated services Amun generates exploit details, shellcode analysis, HTTP headers, Md5 sums etc.

Emulated Services *(Fair)*. The emulated services quality depends on the module that provides the emulation. The basic modules that come with the honeypot provide an fair emulation. Furthermore, Amun provides the user with the ability to create new modules in XML format, not requiring Python programming experience in return.

2.5 Conpot

Conpot is an open source, server-side LIH honeypot designed to be easy to deploy, modify and extend [28]. It provides a number of protocols that are capable of emulating Industrial Control Environments. Its evaluation is depicted in Fig. 4.

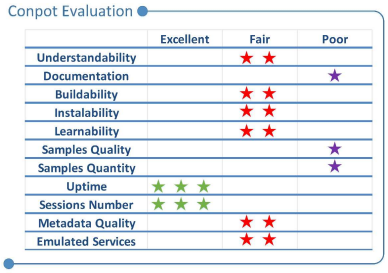

Figure 4: Conpot evaluation

Understandability *(Fair)*. Conpot's purpose is clear and its basic functions are easily understood whereas the advanced functionality demands specialized background in Industrial Control Systems, as well as knowledge of the specific hardware and software used in these systems.

Documentation *(Poor)*. The project's website only con-

tains a high level description of Conpot's functionality and only a very basic information about its installation and usage.

Buildability *(Fair).* The tool is written in Python and presents a modular design. Various third party dependencies are required in order to build Conpot but are documented in the project's website and no problems should be faced if the directions are strictly followed.

Installability *(Fair).* Conpot is available in PyPi and can be easily installed with a single command. Installation instructions are available on the project's website; although third party dependencies are not provided, all were available on-line at the time of writing.

Learnability *(Fair).* It is easy to achieve basic functionality based on the default configuration file. Advanced functionality on the other hand is considerably much harder given the fact that advanced background in Industrial Control System's hardware, software and connectivity is required.

Malware Samples Quality and Quantity *(Poor).* Conpot is not yet capable of providing malware collecting capabilities, but instead to provide the emulation of an ICS system gathering information about automated threats and statistic information.

Uptime *(Excellent).* Even though the experimental setup was scanned several times by adversaries during testing time, the system did not crash or halt functioning.

Concurrent Sessions *(Excellent).* Conpot provides the ability to deploy several configurations with multiple slave systems adding to the system's deception potential.

Metadata Quality *(Fair).* Even though the most critical information such as timestamps, IP's and instructions to the emulated PLC's are logged, a database integration would be a major improvement.

Emulated Services *(Fair).* Even though complex installations can be achieved, the Web-HMI Conpot provides is very basic and when the emulated S7 PLC is probed by specific scanners it can be easily distinguished from a production system.

2.6 Valhala

Valhala is a free LIH designed for the Windows operating system. Even though it is not able to collect malware samples, Valhala can emulate several services and provide important statistical information regarding: *http, ftp, tftp, finger, pop3, smtp, echo, daytime, telnet, port forwarding.* Its evaluation is depicted in Fig. 5.

Understandability *(Excellent).* The purpose of Valhala is clear and both basic and advanced functionality is easily achieved through a friendly graphical user interface. A high level description of its functionality is available on the website.

Documentation *(Poor).* No documentation is available on the project's website or in the software.

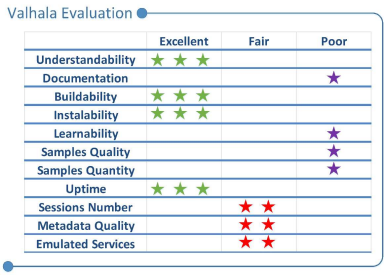

Figure 5: Valhala evaluation

Buildability/Installability *(Excellent).* The program is available in a precompiled executable file and does not require any form of complicate build/installation actions.

Learnability *(Excellent).* The program is self explanatory and minimum interaction is enough to familiarize with the software.

Malware Samples Quality/Quantity *(Poor).* Valhala is not designed to collect malware samples.

Uptime *(Excellent).* During the evaluation period no interaction caused disruptions in Valhala's operation.

Concurrent Sessions *(Fair).* Although Valhala can listen on many ports emulating a sufficient amount of services it can only use one IP address limiting its research potentials.

Metadata Quality *(Fair).* Time, source IP, service and related information are captured and stored in log files. Valhala also supports automated e-mailing for the logfiles which could prove useful for remote administration.

Emulated Services *(Fair).* A consistent emulation of the supported services is provided to the adversary constituting Valhala a competent tool for gathering important information about malware activity.

3. HONEYPOTS AGAINST APTS

Traditional security mechanisms that focus on perimeter security are inadequate to encounter APT's and, in addition, they provide minimal to zero insight of the attack. Honeypots produce no false positives or negatives, a characteristic of invaluable importance in the fight against APT's.

3.1 Stuxnet

Stuxnet's ultimate goal was to alter an ICS functionality (*e.g.*, the control of IR-1 centrifuges used by the Iranian nuclear reactors [21]) by reprogramming its Programmable Logic Controllers and hide the changes from the system operator. It used zero-day exploits, Windows-specific and PLC-specific rootkits, antivirus evasion, process injection and hooking, network infection routines, peer-to-peer updates and a C&C interface [17].

The installation of an LIH such as Conpot, or an actual PLC with no production functionality could indicate the attack factor and the threat could have been eliminated much earlier. A hypothetical architecture is portrayed in Fig. 6. A

honeypot such as Conpot acting as an actual PLC would have been probed by the infected with Stuxnet EWS (Engineering Work Station), indicating the attack at an early stage.

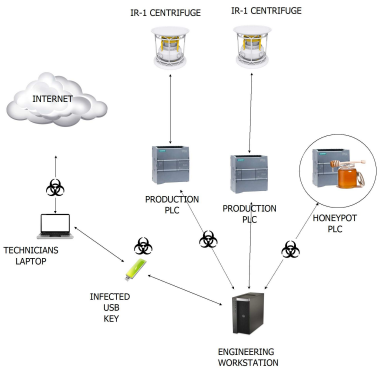

Figure 6: Hypothetical architecture against Stuxnet

3.2 Duqu

Duqu's main purpose is to gather information and assets from industrial infrastructure and system manufacturers organizations in order to attack other third parties [2, 22]. It seeks specific information such as design documents that could help the attackers onset various industrial control system facilities [19]. Its tactics include a keylogger and communication with a C&C server.

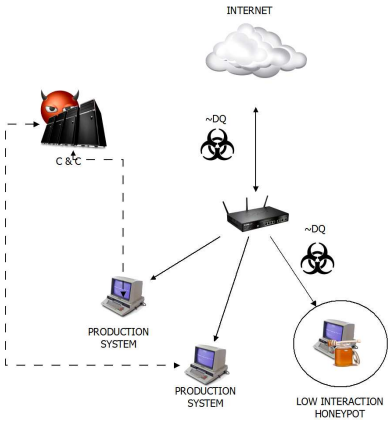

Figure 7: Hypothetical architecture against Duqu

After the establishment of communication with the server a jpeg image is being transferred that is followed by a binary file [19]. A typical LIH could easily realize the attack (Fig. 7). Even though Duqu is triggered by downloading an infected e-mail attachment, it further infects systems connected to the network in an attempt to mitigate the risk of detection. This exact behaviour would have been detected by the LIH, gathering the sample and indicating the sign of compromise.

3.3 Flame

Also known as sKyWIper, Viper or Wiper [23] Flame targets Windows-based PCs. It collects information in multiple ways such as key-logging, taking screen-shots, enabling microphone and camera, gathering files from the infected system [3]. It may also enable a Bluetooth receiver -if available on the target machine- and collects information about nearby devices or broadcasts information regarding the compromised system [1].

Since Flame uses Bluetooth services to collect information about the compromised system, that unique characteristic could be used against it with the aid of Bluetooth honeypot technology [13, 15] (Fig. 8). The infected system would eventually broadcast bluetooth information that could possibly be captured by a bluetooth specific honeypot leading to an early indication of the compromise.

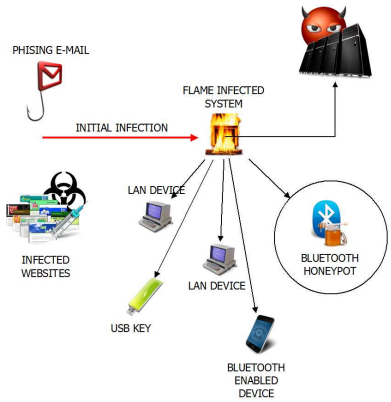

Figure 8: Hypothetical architecture against Flame

3.4 Gauss

Gauss shares a lot of similarities with Flame, mainly regarding the code base and the C&C communication system. It has been actively distributed in the Middle East [26].

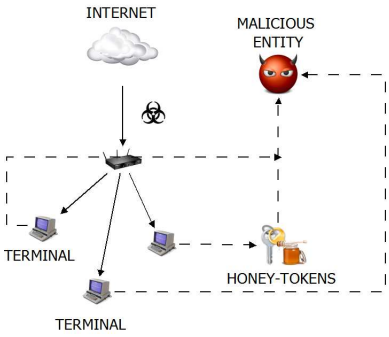

Figure 9: Hypothetical architecture against Gauss

Gauss, similarly to Flame and Duqu, focuses on exfiltrating information designed with emphasis on the maximization of the collected information. This particular characteristic could be its Achilles' heel if provided with honey-tokens, as depicted in Fig. 9. For example, Gauss attempts to steal

information regarding banking information. Specific honeytokens such as fake credit card number and PIN information could easily indicate the attack if the tokens are used in future time.

4. CONCLUSIONS

The honeypot software evaluation that was carried out in this study indicates that even though the level of honeypot technology -in terms of productivity- is considerably transcendent most of the evaluated honeypots do not fully comply with Software Sustainability Institutes software evaluation standards. On the other hand, in this study we argue that traditional perimeter security approach is inadequate against future generation malware and Advanced Persistent Threats. The more persistent and targeted the threat is the more risk must be inevitably undertaken by the defendant in order to champion her assets. Honeypots could prove of invaluable importance in the fight against future attacks deriving both from APTs and human adversaries equally.

5. REFERENCES

[1] Lazar, Marian. Computer viruses as espionage instruments. the case of flame virus. page, 494.

[2] Protected, Wifi. More links between duqu, stuxnet and other malware.

[3] Fiaidhi, Jinan and Gelogo, Yvette E. Scada cyber attacks and security vulnerabilities: Review.

[4] Skoudis, Ed and others. *Counter Hack*. Prentice Hall PTR Upper Saddle River, New Jersey, 2002.

[5] Zhang, Feng and Zhou, Shijie and Qin, Zhiguang and Liu, Jinde. Honeypot: a supplemented active defense system for network security. pages, 231–235. IEEE, 2003.

[6] Spitzner, Lance. volume, 1. Addison-Wesley Reading, 2003.

[7] Jiang, Xuxian and Xu, Dongyan. Collapsar: A vm-based architecture for network attack detention center. pages, 15–28, 2004.

[8] Provos, Niels. A virtual honeypot framework. volume, 173, 2004.

[9] Sadasivam, Karthik and Samudrala, Banuprasad and Yang, T Andrew. Design of network security projects using honeypots. *Journal of Computing Sciences in Colleges*, 20(4):282–293, 2005.

[10] Moser, Andreas and Kruegel, Christopher and Kirda, Engin. Exploring multiple execution paths for malware analysis. pages, 231–245. IEEE, 2007.

[11] Mokube, Iyatiti and Adams, Michele. Honeypots: concepts, approaches, and challenges. pages, 321–326. ACM, 2007.

[12] Provos, Niels and Holz, Thorsten. *Virtual honeypots: from botnet tracking to intrusion detection*. Pearson Education, 2007.

[13] OConnor, Terrence and Reeves, Douglas. Bluetooth network-based misuse detection. pages, 377–391. IEEE, 2008.

[14] Göbel, Jan Gerrit. Amun: A python honeypot. 2009.

[15] Galante, Antonio and Kokos, Ary and Zanero, Stefano. Bluebat: Towards practical bluetooth honeypots. pages, 1–6. IEEE, 2009.

[16] Alosefer, Yaser and Rana, Omer. Honeyware: a web-based low interaction client honeypot. pages, 410–417. IEEE, 2010.

[17] Schneier, Bruce. The story behind the stuxnet virus. *Forbes. com*, 2010.

[18] Tankard, Colin. Advanced persistent threats and how to monitor and deter them. *Network security*, 2011(8):16–19, 2011.

[19] Bencsáth, Boldizsár and Pék, Gábor and Buttyán, Levente and Félegyházi, Márk. Duqu: A stuxnet-like malware found in the wild. *CrySyS Lab Technical Report*, 14, 2011.

[20] Jackson, Crouch, Baxter. Software evaluation: Criteria-based assessment, 2011.

[21] Albright, David and Brannan, Paul and Walrond, Christina. Stuxnet malware and natanz: Update of isis december 22, 2010 report. *Institute for Science and International Security ISIS Reports*, 2011.

[22] Bencsáth, Boldizsár and Pék, Gábor and Buttyán, Levente and Félegyházi, Márk. The cousins of stuxnet: Duqu, flame, and gauss. *Future Internet*, 4(4):971–1003, 2012.

[23] Walter, Jim. Flame attacks: Briefing and indicators of compromise. *McAfee Labs, May*, 2012.

[24] Grudziecki, Jacewicz, Juszczyk, Kijewski, Pawlinski. Proactive detection of security incidents, honeypots, 2012.

[25] Sood, Aditya K and Bansal, Rohit and Enbody, Richard J. Cybercrime: Dissecting the state of underground enterprise. *Ieee internet computing*, 17(1), 2013.

[26] Kushner, David. The real story of stuxnet. *Spectrum, IEEE*, 50(3):48–53, 2013.

[27] Virvilis, Nikos and Gritzalis, Dimitris and Apostolopoulos, Theodoros. Trusted computing vs. advanced persistent threats: Can a defender win this game? pages, 396–403. IEEE, 2013.

[28] Lukas 'glaslos' Rist. Conpot ics/scada honeypot, May 2014.

[29] University of Edinburgh on behalf of the Software Sustainability Insitute. The software sustainability institute, May 2014.