SGNET: Implementation Insights

Corrado Leita, Marc Dacier
Institut Eurecom
2229 Route des Cretes
Sophia Antipolis, France
Email: {leita,dacier}@eurecom.fr

***Abstract*—We present in this paper SGNET, a distributed framework to collect information on Internet attacks, with special attention to self-propagating malware and code injections. This framework is the result of our latest research work on the so-called ScriptGen technology. It is characterized by several unique characteristics that may allow it to provide in the future an extremely interesting perspective on the Internet attacks. In order to make it possible, we need to spread its observation points as much as possible to obtain a complete view on the different blocks of the IP space. We present here an overview of the characteristics of its design with special focus on the possibility to expand it and improve it with additional functional blocks. The SGNET is in fact an open initiative, integrating together tools produced by different research teams such as Argos (VU Amsterdam), Nepenthes, Anubis (TU Wien) and VirusTotal (Hispasec Sistemas). Everybody is welcome and encouraged to participate to this initiative, by hosting observation points and/or by extending this framework with additional modules.**

I. Introduction

The Internet is populated by different varieties of network activities aiming at scanning the IP space and to take control of vulnerable systems for criminal purposes. The existence of botnets, self-propagating worms, trojans, and different kinds of scanning activities is well known. But there is no precise and exhaustive information about the quantity and the variety of these activities in the Internet. Many different projects aim at addressing this deficiency, such as DShield [1], the Internet Motion Sensor [2], the CAIDA project [3] or the Leurré.com project [4]. All these projects take advantage of different techniques and offer different perspectives over the Internet network attacks. The efforts of collecting exhaustive information on Internet attacks normally need to cope with two major problems. First, the need to characterize the different segments of the IP space; second, the challenge of retrieving a significant characterization of the observed attacks.

Previous research [5], [6] showed how different network blocks observe different varieties and quantities of attacks. In order to obtain an exhaustive picture of the network attacks spreading over the Internet it is thus necessary to achieve a global observation perspective. This is only achievable by installing a large number of observation points in different locations of the IP space.

Honeypots [7] proved to be a valuable mean to collect information about network attacks. In order to retrieve sufficient information about the observed network activities, honeypots need to provide a high level of interaction with the attacking sources. They need to drive the attacker into revealing his intent by responding correctly to them: we showed in [8] how incorrect replies to client requests may lead the attacking client to abandon the conversation. Increasing the level of interaction of a honeypot normally leads also to a sensible increase in its resource and maintenance cost. This cost becomes unacceptable when coupled with the previously identified need of maximizing the coverage of the IP space.

We have conceived a novel honeypot deployment called SGNET. SGNET addresses the previous problems taking advantage of a novel technology, ScriptGen [8], [9]. ScriptGen allows to automatically generate approximations of the protocol behavior under the form of FSMs, and take advantage of them to achieve very high levels of interaction with the clients at a very low cost. Whenever the network interaction falls outside the FSM knowledge (newly observed activity), ScriptGen allows to take advantage of a real host to continue the network interaction with the attacker. In that case, the honeypot mainly acts as a proxy for the real host relying to it all the received client requests and sending back the generated responses to the attacking client. This allows to build samples of network conversation for the new activity that are then used to refine the current FSM knowledge (refer to [9] for an in-depth explanation of the proxying algorithm).

The motivations and the architecture of the SGNET are extensively detailed and validated in [10]. In [10] we extend our previous work on ScriptGen by embedding in the learning phase information about code injection attacks and by emulating the shellcode behavior, ultimately downloading malware samples. We also take advantage of an experimental deployment to validate the approach. This initial experiment has evolved into an open deployment available to anybody willing to join by hosting one of its honeypot sensors. We provide here an overview of the design choices underneath its implementation and the possibility of expansion of its functional modules. We also provide information on the requirements to join SGNET and on the consequent advantages. A performance evaluation of the tool is left out of this work for two main reasons. On the one hand, the few sensors currently deployed do not generate yet any sensible load on the system and it would thus be difficult to make assessments in this direction. On the other hand, thanks to the scalability properties of the system, we do not consider load as a critical factor.

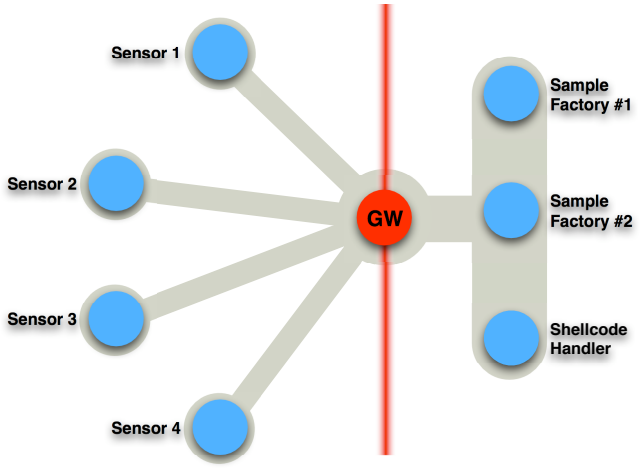

Fig. 1. SGNET architecture

II. SGNET IMPLEMENTATION

Figure 1 shows the SGNET architecture as defined in [10]. SGNET is a distributed infrastructure, composed of different components communicating through a TCP-based HTTP-like protocol called Peiros. The Peiros protocol defines two types of entities: clients and service providers. The service providers offer different services to the requesting clients. The different entities currently implemented in the SGNET architecture are described in the following sections.

A. Sensors

The sensors are the observation points of the SGNET architecture. The SGNET sensors are small daemons written in python with small resource requirements that run over remote machines hosted by different partners (see Section IV) in different locations of the IP space. The sensors are responsible for all the network interaction with the attacking clients. The interaction is normally driven by the knowledge acquired by ScriptGen and represented under the form of FSMs associated to each TCP/UDP port. During this operation, the sensor daemon interacts with clients using the normal socket API and generating the answers taking advantage of the algorithm defined in [8] and [9].

The normal operation of the sensors (normally called FSM-driven operation) can be interrupted by the occurrence of one of the two following conditions:

- The sensor faces a 0-day attack, that is a network attack not belonging to the current FSM knowledge. In this occasion the sensor is not able to handle autonomously the activity, and relies on an external entity used as an oracle (*sample factory*).
- The sensor identifies an FSM state marked as final stage of a code injection attack. In this case, the sensor relies on an external entity (*shellcode handler*) able to unpack and understand the injected shellcode and provide information about the required network interaction to emulate its behavior.

In both cases the sensor network interaction is driven by external entities acting as "oracles". The sensor behaves as a proxy relaying all the packets sent by the attacker to the peiros entity and viceversa. When acting as a proxy, the sensor fetches from the network the RAW IP packets (comprising the IP and the TCP headers) and delivers them to the remote entity over the Peiros protocol. The packets generated by the remote entity are transferred back in the same way and injected on the network by the sensor. The transition from FSM-driven mode to RAW proxying is done by the sensors on a per-source basis taking advantage of the Netfilter *ipqueue* libraries [11]. That is, at a given moment a sensor is able to handle a source A in FSM-driven mode, thus relying on the normal Socket API, and a source B in RAW proxying mode. To accomplish this, the sensor inspects the *iptables* incoming packet queue and removes from the queue all the packets having B as a source address. This prevents the kernel of the sensor host from receiving packets belonging to a session handled by another entity, and allows to independently handle different states for multiple attacking sources hitting the sensor at the same time.

An important aspect of the transition to RAW proxying mode is the necessity to handle the replay of already established TCP sessions. A rather common scenario in the sensor operation is the case in which a 0-day activity is detected in the middle of an established TCP session. For instance, a TCP session starts a normal login attempt with the FSM-driven FTP server and then continues performing an exploit falling out of the FSM knowledge. In this case, the sensor handles autonomously the first part of the TCP session (the FTP login) and only in a second time (the new exploit) it requires the intervention of a sample factory to carry on the conversation. In order to retrieve information about the exploit interaction, the sensor needs to initialize the sample factory by replaying the first part of the TCP session, that is the login attempt. This operation is non-trivial. For instance, the initial sequence number chosen by the sensor host in the TCP 3-way-handshake in the FSM-driven operation will be different from that chosen by the sample factory in the successive replay. The same will happen with the TCP timestamps when present. In order to correctly reproduce the network conversation the sensor needs thus to re-write the sequence numbers and the timestamps of the packets in order to deliver packets with correct TCP header information.

B. Service providers

In the previous section we introduced two main types of Peiros service providers: sample factories and shellcode handlers.

The sample factories are in charge of providing information to the sensor about activities which do not fall yet into the FSM knowledge. The sample factory mainly offers to the sensor a real OS implementation taking advantage of virtualization techniques. By proxying traffic towards these hosts, the sensors generate new samples of protocol interaction. These samples

are used by the ScriptGen algorithm to *learn* new activities incrementally refining the FSM knowledge.

The sample factories are implemented using a custom version of the Argos high interaction honeypots [12]. Argos is a virtualization system that takes advantage of memory tainting to retrieve information about the presence and the nature of code injection attacks. We modified Argos enabling its interaction with a daemon implementing the Peiros interface. When receiving a client request, the daemon instantiates a new Argos process with the required parameters (VM profile, IP address, gateway IP, DNS address,...). This is done using two different approaches. On the one hand, we modified an existing DHCP daemon in order to dynamically configure the guest OS when loading its memory snapshot. The DHCP-based network configuration proved to be unfeasible on some OSs (we experienced the problem on Microsoft Windows 2000). When assigned an IP address, these OSs start to advertise their new address for a period of 30 seconds during which all the network daemons refuse to answer to incoming connections. Such a delay is unacceptable, and required the development of an alternate strategy. When the DHCP-based method fails, we save different memory snapshots of the virtual host for each required network configuration. Being a memory snapshot of the size of approximately 50MB, we consider the space requirement accepatble even for an order of magnitude of thousands of different network configurations.

The shellcode handlers provide information about the necessary network interaction to emulate the behavior of an injected shellcode. To do so, the shellcode handler takes advantage of the *nepenthes* [13] framework. We implemented a nepenthes plugin, called *module-peiros*, to override the standard nepenthes vulnerability modules and directly feed nepenthes with shellcode samples. Module-peiros also takes care of encapsulating and decapsulating the packets proxied over the Peiros protocol.

The shellcode handlers and the sample factories are just two examples of implementation of Peiros service providers. The Peiros protocol has been designed to be as independent as possible from the practical implementation of the service providers. The Peiros protocol defines 4 capabilities:

- **C1**: capability to provide samples of protocol interaction.
- **C2**: capability to provide information about the position of a shellcode.
- **C3**: capability to provide information about the behavior of a shellcode.
- **C4**: capability to provide updates to the FSM knowledge.

It is clear from the previous description that the SGNET sample factories provide capabilities C1+C2, while the shellcode handlers provide the capability C3. The capability C4 is assigned in the current implementation to the gateway entity, extensively described in the next session.

C. Gateway

The SGNET gateway can be considered as a super-entity acting both as a Peiros client and as a Peiros service provider. The gateway acts as an aggregator and load balancer for the other service providers: it maintains a list of all the active service providers assigned to it and of their capabilities and manages their service load. From the point of view of the clients, the gateway is nothing else than a service provider having as capabilities the union of the capabilities of the entities under its management.

In the current SGNET architecture (Figure 1), the gateway is an interface between the private farm of sample factories and shellcode handlers (centrally maintained in Eurecom) and the outside clients (the sensors deployed along the IP space). It supervises sample factories with capabilities C1+C2 and shellcode handlers with capabilities C3. So, from the point of view of the clients, the gateway is a service provider offering the capabilities C1+C2+C3. The whole architecture is flexible to extensions of the initial setup. For instance, it would be possible to deploy several installations of service providers in different locations of the IP space and instruct the sensors to choose the Peiros endpoint offering the best service in terms, for instance, of network latency. This flexibility also allows the system to scale, and cope with the growth of the deployment that will probably impact the system load.

The gateway architecture is based on the publish/subscriber design pattern: the network interaction generates a flow of events, allowing to easily generate plugins subscribing to one or more of these events. The gateway is thus easily extensible to provide new features, such as sensor status logging (reacting to misbehavior of one or more of the sensors).

The capability C4 is implemented taking advantage of this architecture. The ScriptGen refinement engine is a gateway plugin that reacts to the observation of a new sample produced by the sample factories. The plugin tracks all the packets exchanged between a sensor and the sample factory and takes advantage of the information to refine the FSM knowledge. The configuration of all the sensors of the architecture is centralized for ease of maintenance. All the sensors subscribe on startup to the C4 service (provided by the gateway plugin) and receive the FSM knowledge corresponding to the profile assigned to them by the configuration.

III. Accessing SGNET data

The main purpose of the SGNET deployment is to collect information about the observed Internet attacks. This information is stored under a variety of logs:

- Sensor logs. Information about the network interaction (traversal of FSM, presence of code injections,...)
- Sensor dumps. Complete tcpdump trace of the network interaction of the attackers with the sensors.
- Gateway logs. Information about the status of the sensors (activity status, configuration) and about the FSM refinement.
- Nepenthes logs. Information about the nature of the shellcodes, on their network behavior, and downloaded malware samples.

This sparse information is automatically collected and aggregated, and stored in a relational database extending the

original Leurré.com [4] schema. The database collects information ranging from the whole content of the observed packets to different levels of aggregation, easing the analysis of the collected data. We take advantage of a number of external tools and information sources to enrich as much as possible the collected information. We recall here for brevity two of the most interesting ones:

- Malware behavior. We automatically submit the malware samples to the Anubis sandbox [14] and store the retrieved behavioral information in the database.
- Malware AV signatures. We submit the malware samples to VirusTotal (re-submitting on daily basis the samples considered "most interesting") to retrieve statistics about the detection rate and thus have an estimate on the "dangerousness" of a malware sample.

The log parsing scripts are easily extensible to include additional sources of information, or to run daily checks and generate alerts for interesting events.

IV. Contributing to the SGNET

It should be clear that SGNET is an open architecture in many ways. The whole SGNET core has been conceived to make its expansion as easy and straightforward as possible by the simple addition of new plugins. We already integrate the information provided by many different tools, developed by us and by other research teams. There is a lot of space for expansion of the collected information through the integration with other tools. We strongly encourage any research team working in the domain to contribute to this framework and improve the richness of the collected information.

In order to try to expand as much as possible our view over the Internet, we ask to anybody willing to contribute and access our complete dataset to install an SGNET sensor. The sensor deployment procedure has been made as easy and straightforward as possible, and requires a minimum hardware cost. We have prepared a custom Linux distribution derived from the Fedora Core 7 distribution. The installation and configuration of the sensor host is completely automated and requires a minimum configuration effort by the partner. As we explained in this paper, the sensor daemon has very small resource requirements: it can run on an old CPU (Pentium II) with just 256MB of RAM and 1.5GB of free hard drive space. For consistency among all the sensor platforms, we require all the partners to provide 4 routable IPs to be assigned to the sensor installation. There is no maintenance cost for the partner: other than hardware interventions, we take care of the installed sensors remotely, fetching the logs on regular basis, installing security updates, and taking care of the integrity of the system using tools such as *tripwire*.

On the other hand, the benefits for the partners are conspicuous. We provide complete access to the underlying database, both through SSH login and through a PHP web interface. The identity of the partners participating to the deployment is protected by a Non-Disclosure Agreement that every partner needs to sign. Except for the limited requirements of the NDA, all the partners are welcome to take advantage of the vast amount of data to carry on their research or validate the observations made with other sources of data.

V. Conclusions

This paper presented SGNET, a tool that results from the integration of several software tools aiming at observing and collecting information about Internet attacks. The SGNET deployment was designed to ease its extension and the integration with different sources of information. The amount and the quality of the collected data is potentially extremely significant, but requires a widespread deployment of sensors along the IP space. We thus strongly invite any research or industrial entity to take part to the deployment and actively contribute to it.

Acknowledgment

The authors would like to thank Georg Wicherski for his valuable contribution to the ideas and the implementation of the architecture. This work was supported by the Resist Network of Excellence (contract number 026764) and by the RNTR ACES project (contract number ANR05RNRT00103).

References

[1] DShield, "Distributed Intrusion Detection System, www.dshield.org," 2007.

[2] M. Bailey, E. Cooke, F. Jahanian, J. Nazario, and D. Watson, "The internet motion sensor: A distributed blackhole monitoring system," in *12th Annual Network and Distributed System Security Symposium (NDSS)*, (San Diego), February 2005.

[3] Caida Project, "The UCSD Network Telescope, www.caida.org," 2007.

[4] M. Dacier, F. Pouget, and H. Debar, "Leurre.com: On the advantages of deploying a large scale distributed honeypot platform," in *Proceedings of the E-Crime and Computer Conference 2005 (ECCE'05)*, (Monaco), March 2005.

[5] M. Dacier, F. Pouget, and H. Debar, "Honeypots, a practical mean to validate malicious fault assumptions," in *Proceedings of the 10th Pacific Ream Dependable Computing Conference (PRDC04)*, (Tahiti), February 2004.

[6] E. Cooke, M. Bailey, Z. M. Mao, D. Watson, F. Jahanian, and D. McPherson, "Toward understanding distributed blackhole placement," in *WORM '04: Proceedings of the 2004 ACM workshop on Rapid malcode*, (New York, NY, USA), pp. 54–64, ACM Press, 2004.

[7] L. Spitzner, *Honeypots: Tracking Hackers*. Boston: Addison-Welsey, 2002.

[8] C. Leita, K. Mermoud, and M. Dacier, "Scriptgen: an automated script generation tool for honeyd," in *Proceedings of the 21st Annual Computer Security Applications Conference*, December 2005.

[9] C. Leita, M. Dacier, and F. Massicotte, "Automatic handling of protocol dependencies and reaction to 0-day attacks with ScriptGen based honeypots," in *RAID 2006, 9th International Symposium on Recent Advances in Intrusion Detection, September 20-22, 2006, Hamburg, Germany - Also published as Lecture Notes in Computer Science Volume 4219/2006*, Sep 2006.

[10] C. Leita and M. Dacier, "Sgnet: a worldwide deployable framework to support the analysis of malware threat models," in *Proceedings of the 7th European Dependable Computing Conference (EDCC 2008)*, May 2008.

[11] H. Welte, "The Netfilter framework in Linux 2.4," *Proceedings of Linux Kongress*, 2000.

[12] G. Portokalidis, A. Slowinska, and H. Bos, "Argos: an emulator for fingerprinting zero-day attacks," *Proc. ACM SIGOPS EUROSYS*, 2006.

[13] P. Baecher, M. Koetter, T. Holz, M. Dornseif, and F. Freiling, "The Nepenthes Platform: An Efficient Approach to Collect Malware," *Proceedings of the 9th International Symposium on Recent Advances in Intrusion Detection (RAID)*, September 2006.

[14] U. Bayer, C. Kruegel, and E. Kirda, *TTAnalyze: A Tool for Analyzing Malware*. PhD thesis, Master's Thesis, Technical University of Vienna, 2005.