Nomadic Honeypots: A Novel Concept for Smartphone Honeypots

Steffen Liebergeld[1], Matthias Lange[1], and Collin Mulliner[2]

[1]Security in Telecommunications, Technische Universität Berlin, {steffen,mlange}@sec.t-labs.tu-berlin.de
[2]Northeastern University, crm@ccs.neu.edu

***Abstract*—Intelligence on mobile threats is a valuable asset. Honeypots showed to provide a good resource to gain threat intelligence in other areas. Unfortunately, current malware largely relies on social engineering to infect smartphones. Recently, attacks against smartphones have shifted towards local communication interfaces. These trends make traditional honeypot concepts unsuitable. We propose a novel concept called *nomadic honeypot* that provides an infrastructure to enable mobile network operators to collect threat intelligence directly on smartphones. We present a practical design that confines the mobile operating system in a virtual machine. Through virtualization all communication interfaces can be monitored. The actual monitoring is carried out by a second virtual machine running on the same device. This machine hosts sensors and provides a secure backchannel for the operator. Our nomadic honeypot is meant to be used by people. Thus it has the capability to catch malware that is distributed through app stores as well as future threats that attack the smartphone using local communication such as NFC, Bluetooth, and QR codes. We implemented a prototype that runs on an off the shelf smartphone.**

***Keywords*-smartphone, honeypot, malware, worms, threat intelligence, system virtualization**

With rising popularity, smartphones become increasingly attractive for malware. In fact, smartphone security–or the lack thereof–has reached a level of publicity, where customers are very conscious about it. With security becoming a selling argument, being able to warn and protect customers becomes a valuable asset for cellular operators.

Zhou et al. [12] found 93% of malware samples to employ C&C channels, which makes them bots. Such botnets can be very harmful to the core cellular network [11]. Therefore it is in the interest of the operators to know about such threats in order to be able to enact countermeasures.

In IP networks *Honeypots* have been successfully used to collect threat information. However, the classical passive IP-based Honeypot does not fit the infection vectors on smartphones. Most malware today is being installed by the user himself, for example when he installs an infected App from a black market. We also observed the first infections through malicious Quick Response (QR) codes [1].

We observe that the smartphone and its user form a "very high interaction" honeypot. The key insight is that the user is a part of the honeypot. The user involuntarily increases the honeypot's visibility, when he installs malware, scans malicious QR codes, and interacts with malicious NFC devices and RFID tags.

With this insight, we determined that the best place to collect intelligence on current mobile threats is the device itself. To this end, we introduce the concept of *nomadic honeypots*.

Today about 37% [12] of Android malware contains root exploits to elevate its privileges. If it succeeds all security measures of the operating system (OS) become useless. Thus we cannot host our solution in the mobile OS itself. Instead we divide the device into two logical partitions. We move the entire mobile OS into its own partition and remove its direct access to the device's communication hardware. In a second partition we host the nomadic honeypot infrastructure. It has four obligations: First it controls the communication interfaces and mediates all communication of the mobile OS. Second it hosts a wide range of *sensors* to collect and filter events on the communication interfaces. Third it implements facilities for snapshots and logging of the mobile OS. Fourth it establishes a secure backchannel to communicate with the operator.

To show how nomadic honeypots can be constructed in practice, we present a design that is based on a modern microkernel. We implement the partitions with virtual machines (VMs). We implemented a prototype that runs on an off the shelf Samsung Galaxy S2 smartphone.

Our contributions are:

- **Concept of nomadic honeypots** We introduce nomadic honeypots as an infrastructure to collect information on threats directly on mobile devices.
- **Practical design** We present a practical design of our nomadic honeypot. We employ virtualization to confine the mobile OS and remove its direct access to communication hardware. We mediate all communication in a separate VM, where we deploy sensors which collect information, and a secure backchannel.

This paper is structured as follows. We introduce the concept of nomadic honeypots in Section I. Then we show how a nomadic honeypot can be constructed in practice in Section II. We show our prototype in Section III, and present ideas for sensors in Section IV. Sections V and VI discuss the

ethical implications of nomadic honeypots and how operators can deploy them. We conclude in Section VII.

I. Concept of a Nomadic Honeypot

The nomadic honeypot is deployed directly on a smartphone. In our concept the user plays a key role, as he is responsible for the visibility of our honeypot: He moves the honeypot into interesting areas, scans malicious QR codes and installs malicious applications. Ideally the nomadic honeypot is the primary smartphone of the user that he uses on a daily basis. We discuss an idea on how operators can make people use nomadic honeypots in Section VI. Conceptually the nomadic honeypot requires that the smartphone is logically divided into two isolated partitions.

The main partition hosts the mobile OS, but has no direct access to the device's communication hardware. Malware often includes checks if it is being run in an unusual environment such as an emulator, and turns off its malicious payload to escape detection. Therefore it is vital that the mobile OS is modified as little as possible.

The second partition hosts the infrastructure for our nomadic honeypot. It has four obligations: First, it mediates all communication of the mobile OS. Second, it hosts infrastructure for data collection (sensors). Third, it implements facilities for snapshots and logging. And fourth, it provides a secure backchannel for the operator.

Mediating all communication serves two purposes. First, it allows for powerful sensors that can monitor the data stream directly and no communication goes unnoticed. Second, it allows us to confine malware and stop it from spreading.

Strict isolation between the partitions ensures that even a subverted mobile OS cannot tamper with the nomadic honeypot's infrastructure. Therefore the operator can trust in the information that is being collected by the nomadic honeypot. Cryptographic keys that are needed to establish the backchannel remain confidential, and an attacker cannot use them to connect to the operator.

The operator can use the collected data to gain intelligence on mobile threats. He can request snapshots of the mobile OS's file system to do an offline forensic analysis of attacks. Thereby he can gain thorough insight on the nature of the threats and use his findings to protect his customers. An illustration of nomadic honeypots in action is given in Figure 1.

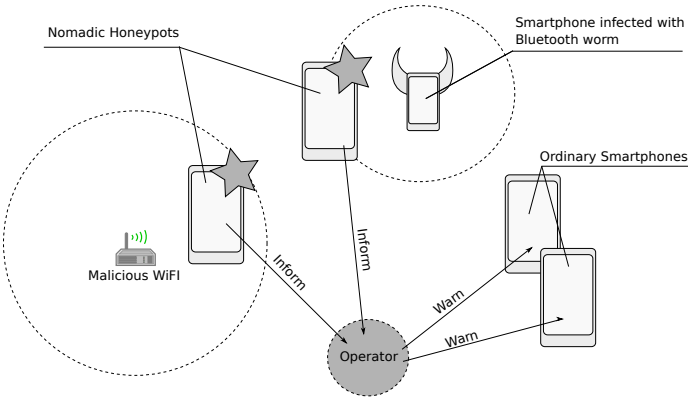

Figure 1. Illustration of nomadic honeypots at work. In this picture one nomadic honeypot is connected to a malicious WiFi hotspot. Another one is within reach of a smartphone that has been infected with a Bluetooth worm. Both honeypots collect data about the threats and send this information to the operator. The operator can use this information to protect his customers, for example by updating blacklists of WiFi hotspots.

II. Design of a Practical Nomadic Honeypot

In this section we show how a nomadic honeypot can be constructed for today's smartphone hardware. The most prominent question is how to partition the device. ARM TrustZone [10] implements partitioning in hardware. We want to be able to deploy our nomadic honeypot to all smartphones. Therefore TrustZone is not an option because it is not implemented in all smartphones. Even if it is implemented, it is usually not available because the OEM already deployed a secure monitor that cannot be replaced. Instead we opt to do virtualization on a microkernel. As shown by Lange et al. [7] virtualization of mobile OSes like Android is possible even on today's smartphones. Mediating hardware access is possible as well. This was shown by Mulliner et al. [8], who used L4Android [2] for their signalling filter. Our design mandates to boot the microkernel with secure boot to ensure its integrity. An illustration of our setup is shown in Figure 2. It consists of the following components:

Microkernel The microkernel is the only component running in the most privileged mode of the CPU. Its purpose is to partition the device's resources, and to ensure isolation. The small code base of the microkernel allows for thorough analysis of the its code. There even is a microkernel that has been formally verified [6]. That is why we put trust in the microkernel's isolation capabilities and assume it cannot be compromised.

Honeypot VM We run the mobile OS in its own VM, which is configured such that it does not have direct access to communication hardware. This allows us to mediate and monitor all of its communication. As an additional benefit, it allows us to contain a potential compromise of the mobile OS as well. For our architecture the mobile OS kernel needs to be modified. First, it needs to be virtualized. Second, its drivers need to be made aware of the indirection of communication. Both modifications are transparent to applications. The question is if malware can detect that it is running in a nomadic honeypot. The fact that the mobile OS has been virtualized can be detected with timing analysis. Timing analysis is only possible if the attacker knows the exact temporal behaviour of the victim's device hardware. If the nomadic honeypot is deployed on a large variety of devices, an attacker would have to know the characteristics of all devices that are being used. If the malware succeeds to subvert the mobile OS kernel, it can determine that the device drivers have been modified for indirection. Even in this case the attacker needs to know the original drivers to tell if they are modified. In summary it is unlikely that malware authors go through the effort needed to detect a nomadic honeypot.

Infrastructure VM We place the nomadic honeypot infrastructure inside a second VM. We call it the *Infrastructure VM*. It runs a lightweight Linux environment. This allows us to leverage the rich programming environment of Linux for sensors, snapshots, logging and the backchannel. The Infrastructure VM is allowed to control the communication hardware directly. It presents the Honeypot VM with virtual equivalents of the communication hardware as well as a virtual file system.

In detail, the Infrastructure VM hosts the following facilities.

Sensors Our system employs sensors to record events of interest that are to be sent to the operator for threat intelligence. A sensor sits in between the mobile OS and its respective communication device. Thus it can tap into the communication to detect events of interest and if needed stop outgoing attacks. Sensors can also leverage the virtualization infrastructure to directly get information from the Honeypot VM's memory. Sensors do filtering and possibly compression of information to keep the amount of data being sent to the network in control. When a sensor detects an event that is typical for an attack, it can instruct other sensors to start collecting more data. This increases the chances for a forensic analyst to have enough data to analyze an attack. There are many potential sensors, specifically any kind of IO device inside a smartphone is a potential point for data collection. We provide more details and ideas on sensors in the next section.

Facilities for snapshots and logging The Infrastructure VM presents the Honeypot VM with a virtual file system. The resident data of the Honeypot VM is entirely hosted by the Infrastructure VM. This enables it to do snapshots, which can help with attack forensics. We envision to have a restore mechanism for the Honeypot VM as well. Our design includes a mechanism that allows us to keep log files for forensic analysis. It may even log events from the virtualization layer, which could provide enough data for an analyst to completely replay the Honeypot VM. We envision that both log information and a snapshot are sent to the operator on demand, so that he can do attack forensics offline.

Secure backchannel The secure backchannel is used for communication with the operator. It directly communicates using the device's communication hardware, as shown in Figure 2. To that end we establish a virtual private network (VPN). The cryptographic key material for the VPN is hosted entirely in the Infrastructure VM and is therefore inaccessible to the mobile OS. The backchannel comprises a ringbuffer to store information during the time when there is no network connection available. As an added value, the operator can use the backchannel to deploy countermeasures to new attacks. For example he can distribute updated blacklists of malicious WiFi hotspots.

III. Prototype

We implemented a prototype of our nomadic honeypot architecture that runs on the Galaxy S2 smartphone. Our prototype is based on the Fiasco.OC microkernel [3]. We choose Android as the mobile OS because it is based on the open source Linux kernel, which can be virtualized on non-virtualizable CPUs such as the current ARM Cortex-A9. In fact we leverage the L4Android project [2] as introduced by Lange et al. [7].

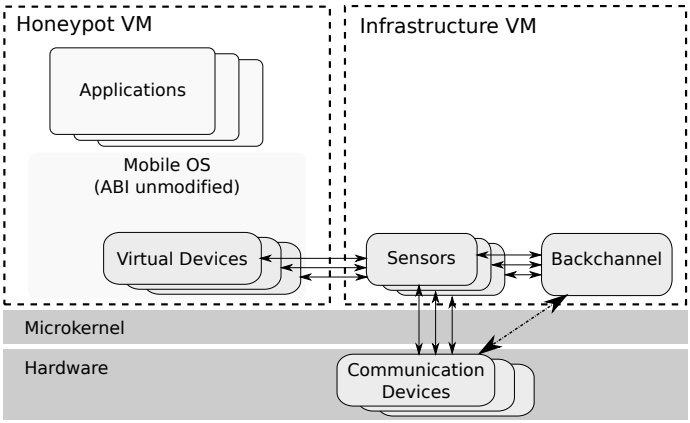

Figure 2. Simplified illustration of a practical nomadic honeypot. It consists of the Honeypot VM running the mobile OS and its applications and an Infrastructure VM. The Infrastructure VM mediates all communication of the mobile OS, and employs sensors to collect threat intelligence. The collected data is sent to the operator via a secure backchannel.

The implementation proved to be very laborious because all involved drivers need to be modified to be interposed. So far we implemented device drivers for the cellular baseband, the touchscreen and buttons, and the accelerometer. In our testing the prototype showed performance and battery runtime that– while being degraded compared to stock Android–is sufficient for everyday use. This makes us confident, that our approach can work in practice.

IV. Sensors

Our nomadic honeypot's purpose is to collect intelligence on the smartphone. Its data collection facilities are the sensors. We envision to employ sensors for the following devices: NFC, Bluetooth, WiFi, input (touchscreen, buttons), GPS and cameras. We will now give some examples on how our infrastructure can help in collecting information about attacks.

App Store A lot of malware is distributed via centralized app stores. Our nomadic honeypot can detect infections by using virtual machine introspection to monitor the mobile OS, e.g. to detect rooting or kernel compromise.

Bluetooth Bluetooth attacks can be caught with Bluetooth honeypots like Bluebat [5] and Bluepot [9]. The problem of these Bluetooth honeypots was that they were not deployed on real mobile devices, and therefore they did not catch many attacks. In our system, these honeypots can be deployed in the Infrastructure VM without much modification. We believe that this setup perfectly matches the attack vector of Bluetooth worms and therefore has best chances of catching attacks.

WiFi Malicious WiFi hotspots do harm, for example with modified DNS entries that direct unsuspecting users to phishing sites. Once the user decides to join a public WiFi hotspot, a

sensor can probe it for malicious behaviour. Possible tests are checks for unusual traceroutes or unexpected DNS resolution of known URLs (e.g. those of banks).

SMS/MMS/Calls In addition to recording and monitoring all incoming SMS and MMS, we can also employ intelligent filters to block malicious premium SMS or SMS that subscribe the user to expensive services before they are sent to the network.

QR Codes QR codes are a widely used form of 2D barcodes. On scanning the QR code, a special application decodes the barcode to reveal its content. Malicious QR codes contain information to execute services such as dial bad telephone numbers, send unintended SMS, execute USSD codes [4] or access malicious URLs. All these malicious intents involve communication, which goes through our Infrastructure VM and its sensors and can therefore be detected.

Location For local attacks such as NFC, Bluetooth and WiFi it is of interest where the attack happens. For this it makes sense to record the phone's current location together with recorded attack events.

V. Discussion: Ethical Issues

Our nomadic honeypot aims at being used by real people on a daily basis. It will necessarily host the user's private information, including the address information and telephone numbers of his peers, his emails, location, browser history and more. In many countries there are strict privacy laws, which have to be maintained. This can be tricky, especially when the data is transmitted to the operator. A possible solution could be certified anonymizing algorithms that can be deployed in the sensors. However, for some information for example location data, anonymization might not be possible at all.

It is in the nature of our honeypot that it is prone to be attacked. If subverted, the device potentially becomes a threat to other devices in that its malware will try to spread itself by infecting them as well. In our architecture the compromised smartphone OS does not have the capability to communicate directly. Instead, the Infrastructure VM mediates all communication, which gives us the ability to stop malware directly on the device, before it can harm other devices.

VI. Discussion: Deployment of Nomadic Honeypots

The nomadic honeypot is of use only when it is being actively used by people on a daily basis. The question is how can operators recruit people to use nomadic honeypots?

The nomadic honeypot has inherent usability drawbacks: It has some computational overhead, which means the devices will not be as fast as they could be and that the battery will not last as long. Further, the honeypot has an inherent privacy issue as discussed in the previous section.

We think the best way to make people use nomadic honeypots is when the operator offers these devices to selected users at a large discount (e.g. hand out these phones for free), or with a data plan with little or no cost. The user would be informed about what drawbacks nomadic honeypots have in terms of usability and privacy compared to the other (more costly) smartphones and data plans. The user must agree to have understood the terms of use.

VII. Conclusion

In this work we introduced a concept for nomadic honeypots. We described a practical design to show how our concept can be implemented in practice, and implemented a prototype. We presented ideas for sensors for the most prominent mobile attack vectors, such as malware, SMS/MMS, Bluetooth, NFC and QR codes. Finally we elaborated on ethical issues and how nomadic honeypots could be deployed.

VIII. Acknowledgements

We would like to thank Adam Lackorzynski, Alexander Warg, Jean Wolter, Christian Ludwig, Michael Voigt, Janis Danisevskis and Jan Nordholz for their help in creating our prototype and Ravishankar Borgaonkar for proof-reading.

This work was partially supported by the EU FP7/20072013 (FP7ICT2011.1.4 Trustworthy ICT), under grant agreement no. 317888 (project NEMESYS).

References

[1] That square QR barcode on the poster? Check it's not a sticker. http://www.theregister.co.uk/2012/12/10/qr_code_sticker_scam/, December 2012.

[2] L4Android: Android on top of L4. http://www.l4android.org, February 2013.

[3] The Fiasco microkernel. http://os.inf.tu-dresden.de/fiasco/, January 2013.

[4] R. Borgaonkar. Dirty use of USSD codes in cellular networks. https://www.troopers.de/wp-content/uploads/2012/12/TROOPERS13-Dirty_use_of_USSD_codes_in_cellular-Ravi_Borgaonkor.pdf, March 2013.

[5] A. Galante, A. Kokos, and S. Zanero. Bluebat: towards practical bluetooth honeypots. In *Proceedings of the 2009 IEEE international conference on Communications*, ICC'09, pages 920–925, Piscataway, NJ, USA, 2009. IEEE Press.

[6] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood. seL4: formal verification of an OS kernel. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, SOSP '09, pages 207–220, New York, NY, USA, 2009. ACM.

[7] M. Lange, S. Liebergeld, A. Lackorzynski, A. Warg, and M. Peter. L4android: a generic operating system framework for secure smartphones. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, SPSM '11, pages 39–50, New York, NY, USA, 2011. ACM.

[8] C. Mulliner, S. Liebergeld, M. Lange, and J.-P. Seifert. Taming Mr Hayes: Mitigating Signaling Based Attacks on Smartphones. In *Proceedings of the IEEE/IFIP 41st International Conference on Dependable Systems Networks (DSN)*, Boston, MA, June 2012.

[9] A. Smith. Bluepot: Bluetooth Honeypot. http://code.google.com/p/bluepot/, February 2013.

[10] T. Alves and D. Felton. TrustZone: Integrated hardware and software security. Technical report, ARM Limited, 2004.

[11] P. Traynor, M. Lin, M. Ongtang, V. Rao, T. Jaeger, P. McDaniel, and T. La Porta. On cellular botnets: measuring the impact of malicious devices on a cellular network core. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 223–234, New York, NY, USA, 2009. ACM.

[12] Y. Zhou and X. Jiang. Dissecting android malware: Characterization and evolution. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 95 –109, may 2012.