Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: May 7, 2020

J. Fabini
TU Wien

November 4, 2019

Communication Network Perspective on Malware Lifecycle
draft-fabini-smart-malware-lifecycle-00

Abstract

Today's systems, networks, and protocols are complex and include unknown vulnerabilities that adversaries can exploit. The large-scale deployment of network security protocols establishes an additional threat by implementing a substrate for hidden communications like covert or subliminal channels. The resulting ecosystem builds a convenient platform for malicious, automated software (malware) to infiltrate critical infrastructures, to gradually infect large parts of the system and to coordinate distributed malware operation.

Based on the observation that malware depends on network communications to discover, propagate, coordinate, and unleash its functionality, this memo recommends methods to identify potential interfaces and interactions between malware and protocols. It proposes a generic malware lifecycle model that defines a set of generic malware states and possible transitions between these states. Coordinated activities of distributed malware can be mapped to state transitions in malware instances, supporting the identification of (potentially hidden) network communication as a trigger for actions and hints on protocols that enabled the communication. Eventually, the proposed model aims at supporting the identification of architectures, protocols, interfaces, and points in time that a) either inhibit hidden malware communication or b) allow for optimized detection of anomalies as main prerequisite for timely countermeasures.

While earlier work focused on protecting single hosts from compromise, this memo adopts a holistic view and considers the health of the overall networked system to be of highest priority. Presuming vulnerable systems, we stress that components or subsystems must be disconnected on suspected infection in an attempt to continue (even partial) operation of the overall (non-infected) system after the disconnect. Containment - the isolation of an infected subsystem - becomes an essential security feature in the context of critical infrastructures that influences on deployed protocols, interfaces and architectures.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

A central guideline of the IETF security area's activity focus is summarized in RFC 3552 [RFC3552]: "Protecting against an attack when one of the end-systems has been compromised is extraordinarily difficult". This statement is still valid today but must be seen in a historical context: in times of monolithic systems, the main goal of security is (or was) to protect one's own networked end system (PC, server) against compromise. This implies a worst case scenario and "game over" in case of a system compromise. In a distributed context, one single compromised system can be fatal whenever relying on a chain of trust, which is a common security policy within closed (corporate or enterprise) networks.

However, architectures and protocols have evolved. Emerging critical infrastructures consist of ensembles of hundreds, thousands or tens of thousands of identical networked systems like for instance smart meters or other Internet of Things (IoT) devices. These systems all run identical software and identical firmware on top of identical hardware, all of them being potentially subject to identical vulnerabilities. Likewise, most personal computers that are connected to the Internet run one of a few operating system alternatives, including Microsoft Windows, Apple MacOS, or various Linux distributions. Portable software and common Application Programming Interfaces (APIs) increase the likelihood that one vulnerability affects multiple platforms.

When viewing a system as a complex set of components and relations (Rechtin [CBCS]), there are cases when vital system functions can be performed even in the case when some subsystems (components or links) have been compromised. Therefore, today's security concepts and

research must support (a) identification and (b) containment, i.e., isolation, of compromised subsystems at an architectural and protocol level. It is important to note that these requirements *extend* (and by no means contradict) the requirements stated in RFC 3552 [RFC3552] with respect to the importance of protecting systems against compromise.

On this purpose, this memo proposes to enlarge the scope of systems security, starting from two main prerequisites, namely that (a) any system (single end node, component, link) is vulnerable and (b) malware must communicate to propagate, to discover and to coordinate its distributed instances. Section 2 proposes a generic malware lifecycle model consisting of malware states and transitions. This generic state diagram is subsequently mapped to existing malware implementations to infer on malware communication needs, as well as on potential interfaces and protocols that malware may use for discovery, infection, propagation, and control through available network paths. By monitoring these interfaces, systems can detect patterns of - potentially hidden - communications as an anomalous component of the network traffic. Subsequent analysis of available architectures, interfaces, and protocols can help in identifying anomalous communications and stopping it in order to prevent malware from propagation and execution.

We consider the identification of systematic and design shortcomings of architectures and protocols with respect to hidden communications to be an essential component of the security-by-design concept. A first step is the definition of metrics and methods that can assess the degree to which protocols under investigation support -- or prevent -- hidden communications. The ability to evaluate protocols and choose the ones that are proven to be covert-channel free enables system architects to close existing gaps for hidden malware communication.

Todo: the terms used in this memo should be eventually aligned to [I-D.mcfadden-smart-endpoint-taxonomy-for-cless].

1.1. Requirements Language

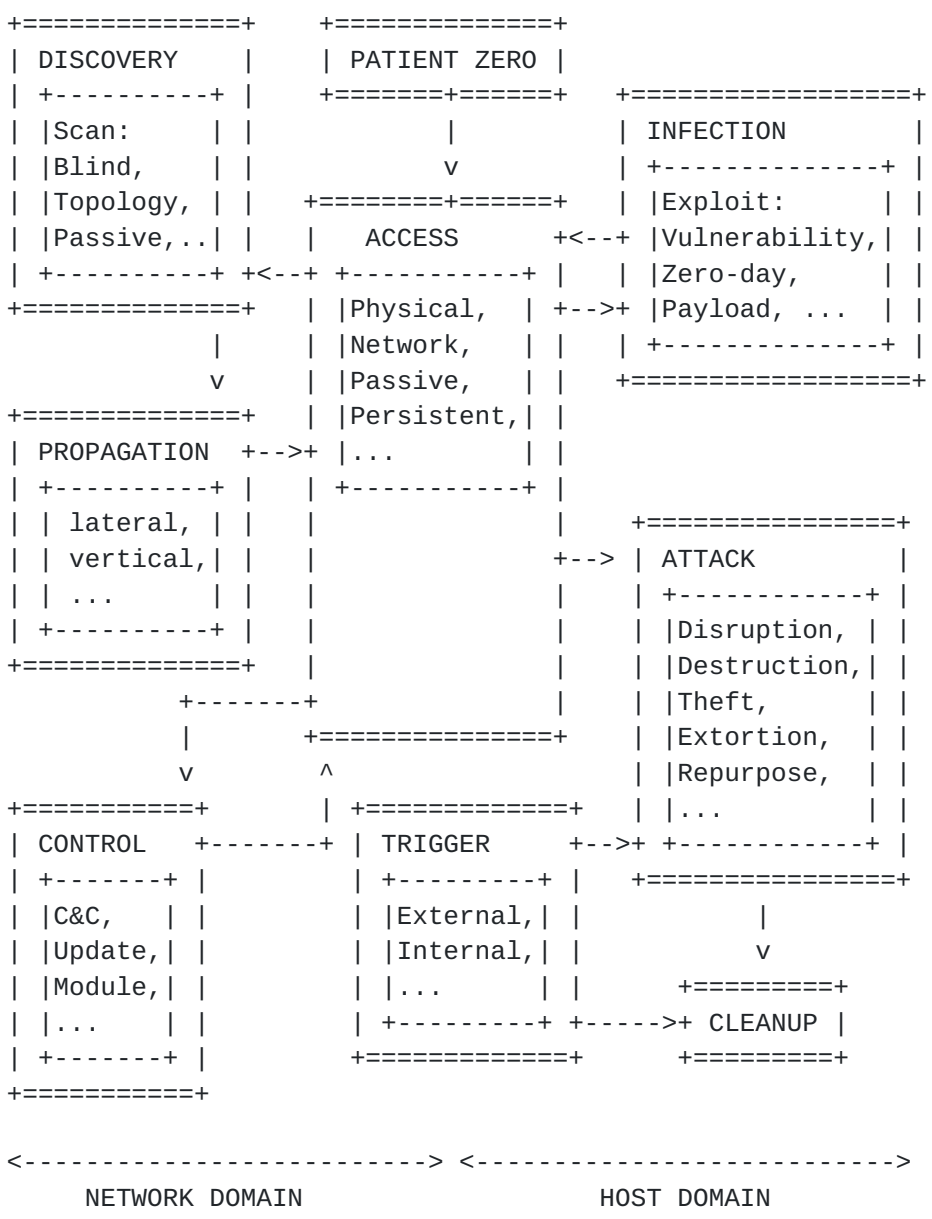
The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. Generic Malware Lifecycle

The state diagram depicted in Figure 1 illustrates a generic malware lifecycle model. A graphical representation of the diagram along with a detailed description can be found in the original publication

[GML] or its pre-published version at https://publik.tuwien.ac.at/files/publik_261089.pdf.

Generic Stages of Malware Lifecycle.



An extended graphical representation of this diagram along with detailed descriptions can be found in [GML].

Figure 1

Malware activity in Figure 1 revolves around the concept of access to abstract resources. Essential from an defender's monitoring perspective is that, depending on their implementation and target,

malware variants differ substantially in their use of communication networks. Common to many recent malware is that it encrypts communication, attempts to obfuscate it as legitimate traffic, and/or uses hidden communication channels to stay unobserved. Aggressiveness and "noise" that malware generates while propagating, infecting and attacking differs substantially between malware types.

This is why this memo focuses on evaluating protocols, interfaces and architectures with respect to their ability to inhibit or support hidden communications. The proposed generic lifecycle model can identify the malware's need for communication to trigger state changes. (Internal: provide hints to anomaly detection systems? estimated amount of data as an order of magnitude: transferring a malware update or additional malware modules requires more data transfer than a single command.)

The following subsections discuss briefly the generic stages of malware lifecycle in line with [GML].

2.1. Access

Starting point of malware operation is the so-called patient zero, denoting a device or method that triggers the initial infection within the system under observation. Examples for access options include, but are not limited to physical access (e.g., through a compromised USB stick inserted into a computer, through hard drive replacement or through starting from a temporary boot device), network access (e.g., as part of existing connections, or through a hidden communication channel), application access (e.g., by sending a legitimate email with compromised payload), or persistent access (for instance an intentional or unintentional backdoor that is installed by firmware or BIOS).

The patient zero may depend on qualified (human) support to bypass existing security barriers and gain access to the system. This may be, e.g., a staff member plugging a compromised USB stick into a computer to infiltrate an air-gapped system, or an employee of the computer manufacturer who adds a backdoor to the computer BIOS, firmware or software. Once it has gained access to the target system, the malware can start its operation.

Todo: Extend, discuss options.

2.2. Infection

Having gained (temporary) access to the system, malware depends on system vulnerabilities to support its attempt to infect the system and install itself persistently. Examples include the exploit of

backdoors, zero-day vulnerabilities or execution of a malicious email attachment.

Todo: Extend, discuss options.

2.3. Discovery

Once the local system is infected, malware has several options. The most common malware strategy is to first discover new potential victims that are reachable via the communication network. Alternatives for discovery differ in terms of communication verbosity and range from blind scans to passive monitoring of incoming network connections and many variants in between. Blind scans are the most aggressive but also the most verbose variant of discovery, malware actively scanning ranges of IPv4 or IPv6 addresses like, e.g., the current subnet or all IPv4 addresses. In typical networks monitoring devices can easily detect these blind scans because of the high volume of additional illegitimate traffic. Adding some more intelligence to the discovery process results in targeted scans to decrease the amount of traffic that is needed for probing. Examples include the support for distributed scan lists that record already scanned (and infected) devices, or a prioritization of the scan process to prefer system-critical devices like, e.g., the standard gateway. Most stealthy and most difficult to detect is malware that monitors passively its local network interfaces on incoming and outgoing traffic to infer on the network topology and potential targets. However, this stealthiness comes at the cost of reduced malware propagation speed and is typical for complex attack patterns.

It is worth mentioning that the supported IP address version has substantial impact on the discovery strategy that malware may use or prefer. Whenever targeting IPv4 addresses, distributed malware can scan the entire Internet within reasonable time. The large address space of IPv6 and the resulting sparse population of subnets will likely result in malware to prefer targeted active scans or passive scanning for the discovery process.

Todo: Extend, discuss options.

2.4. Propagation

Following the discovery of a potential victim, malware attempts to propagate over existing communication channels to gain access to these victims and install new instances of itself in the network.

Todo: Extend, discuss options.

2.5. Control

All presented malware activities or state changes happen either autonomously, which is typical for early malware variants, or guided by some command & control infrastructures that recent malware variants prefer to allow for later malware modification and coordinated attacks. Examples of the latter variant include malware that supports remotely controlled updates, loading of new modules and distributed C&C structures. Such functionality facilitates the update of encryption keys, communication patterns and functionality, as well as the support for new communication protocols. Eventually, this functionality enables offerings business models of "malware as a service": botnet owners may operate infrastructures of compromised devices that customers can rent and use to execute their custom-tailored malicious code.

Todo: Extend, discuss options.

2.6. Trigger

Triggers are essential for supporting the coordination of functionality in distributed malware instances, typical example being the launch of a coordinated DDoS attack. Explicit control communication (command) is one option for an external trigger, other less suspicious options include the setting of conditions that distributed malware instances can observe. Examples include timers (some malware variants implementing explicit time synchronization with dedicated time servers for improved accuracy) but also availability of specific servers at specific domain names, etc.

Internal triggers are typically hard-coded into the malware or its modules and support it in targeting and focusing its attacks. These triggers can, for instance, control malware to launch its attacks on specific hardware- or software systems only, or can limit its actions to specific IP address ranges and/or DNS domains.

Todo: Extend, discuss options.

2.7. Attack

Once successfully propagated, malware can start its damaging functionality that ranges from destruction and disruption to theft or extortion.

Todo: Extend, discuss options.

2.8. Cleanup

Recent malware variants focusing on stealthy operation include hidden communication and cleanup functionality to remove themselve from infected systems. The cleanup starts either on completing the attack or on external triggers after accomplishing their goal.

Todo: Extend, discuss options.

3. Mapping the Lifecycle Model to Real Malware

This section maps the known behavior of well-studied, prototypical malware variants to the Malware Lifecycle Model. Eventually, this mapping aims at identifying malware communication needs and behavioral patterns that automated processes can use to discover unknown malware.

Central observation with respect to the Malware Lifecycle Model's applicability is that malware has huge incentives to communicate, and that monitoring devices can detect this communication as anomaly. In particular, network communication is a key component for malware to unleash its full destructive potential. Infecting systems remotely and automating and coordinating their distributed activities using network communications brings huge benefits to malware authors. Most notably, being physically located in distinct geographical, jurisdictional, and/or legislational regions supports networked operations while minimizing the risk of being prosecuted for the results of these actions.

Bridging the air gap to an isolated system is conditioned by physical access to the system. Options include access to the system or to parts of it, either during the manufacturing process (e.g., by compromising a computer's BIOS and adding a backdoor) or later on, during installation or operation (e.g., by inserting a compromised USB drive into the system). From a malware author's perspective, the physical access alternative has severe drawbacks. First, the need for physical access may leave traces that help in identifying the originator. Second, the lack of updates and coordination: malware must be fully functional at the time of first infection, updates for it depending on recurring physical access to the system. However, even in the case of air-gapped systems malware may subsequently attempt to discover and infect locally connected systems (as exhibited for instance by Stuxnet). These communication attempts may be monitored and detected.

Summarizing, the main incentives for malware to communicate include the following:

- Network-based malware coordination and control: the closer coordinated distributed malware instances can act, the higher the potential severity of their aggregated actions (for instance in the case of DDoS attacks). Malware may use coordination to reduce network traffic, too (for example by maintaining scan lists when scanning for new victims).

- Network-based update: the complexity and sophistication of today's malware increases the effort for its programming. This drives the trend for modular malware that can install a minimum persistent foothold, update itself and can load novel functionality on demand as additional modules. Malware update can support malware authors by protecting their assets in the case of malware identification and/or takeover attempts by competing organizations. In such cases, malware updates can support in the modifications of keys, change of encryption algorithms, use of novel obfuscation methods, etc.

- Network-based discovery of potential infection targets and propagation: Scanning for infection candidates and propagation range among the two most verbose activities of today's malware. Worth noting is that specific malware functionality is typically related to malware size, i.e., data volume that the malware must transfer. Depending on the implementation, malware can decide to transfer its entire body at propagation time or install a tiny foothold during propagation that subsequently loads the required modules. The data pattern that monitoring devices can identify differs for these two alternatives: the self-carried malware will be visible in monitoring logs only once, when transferring a large amount of data. The modular variant consists of several smaller data transfers.

- tbc...

The remainder of this section presents prototypical case studies of existing malware variants, the mapping of their behavior to malware lifecycle model stages and how the lifecycle model can support in their detection. Tables 1-4 of [GML] compare and discuss features and peculiarities of various malware variants in more detail. Future versions of this draft are planned to structure and extend the malware communication aspects that these tables summarize, eventually building the base for a generic malware detection framework.

3.1. Case study: Stuxnet

Stuxnet [Stuxnet] is a computer worm that was reported for the first time in June 2010. The effort associated with the design and implementation of Stuxnet was substantial, pointing to nation states

or intelligence services as authors. This speculation is backed by Stuxnet's stealthy behavior and targeted attack against Siemens Simatic S7 Supervisory Control and Data Acquisition (SCADA) Industrial Control Systems (ICS), eventually aimed at causing physical damage. The following subsections map the known Stuxnet behavioral and communication patterns to the generic Malware Lifecycle model stages and transitionsl.

3.1.1. Access

The initial Stuxnet access (patient zero) targets air-gapped systems. It uses the autostart functionality of Microsoft Windows 32 bit operating system variants on inserting a USB stick. As soon as the first system has been infected, Stuxnet attempts to discover and access other computers within the same LAN. Whereas the initial infection (USB drive autostart) can not be captured by the Lifecycle Model, the access to other computers within the LAN can be monitored within the network traffic.

3.1.2. Infection

Stuxnet exploits several zero-day vulnerabilities that were unknown by the time of its release and allowed for privilege escalation on several Microsoft Windows 32 bit operating system variants. In addition, Stuxnet made use of two stolen certificates to sign its drivers. The infection includes installation of dedicated RPC servers and -clients and peer-to-peer clients for communication with other infected Stuxnet instances within the same LAN, as well as infection of connected network shares. Local infection and installation is not in the scope of the Lifecycle Model, whereas the infection of network shares may lead to unexpected network traffic and monitored network anomalies.

3.1.3. Discovery

Following an initial infection, Stuxnet scans the local network for potential, previously uninfected targets. Stuxnet also uses specific domains to probe for Internet connectivity. All of these network scan operations are typical and can be monitored and detected.

3.1.4. Propagation

Whenever Stuxnet identifies uninfected targets in the local network with Siemens Step7 software installed, it propagates and attempts to infect these PCs. Otherwise it enters a dormant mode.

3.1.5. Control

Stuxnet instances within the same network use peer-to-peer RPC calls and encryption to update each other. This method allows one single USB drive infection to update distributed Stuxnet instances in air-gapped systems. Whenever Internet access is available, Stuxnet contacts command and control servers using encrypted communication to receive updates, additional features, and instructions. These update RPC calls and the traffic to command and control servers can be identified by network monitoring systems as anomalies.

3.1.6. Trigger

Stuxnet installation is conditioned by software (Siemens Step7) software to be installed on, and/or Siemens PLCs being connected to the Windows-based system. The Lifecycle Model can not capture these triggers as they are proprietary to the malware and do not involve network communication.

3.1.7. Attack

Once installed on a system that controls a PLC, Stuxnet acts as a man-in-the-middle. Faulty commands, aimed to cause physical damage, are sent to the PLC, and forged PLC response codes are forwarded by Stuxnet back to the controlling application to pretend correct operation. Complex (cross-layer) monitoring systems, featuring sensors inside the PLC, could identify the mismatch between the commands sent by the controlling application and the commands received by the PLC. Likewise, system log correlation with network traffic data could reveal anomalous behavior.

3.1.8. Cleanup

Stuxnet stores several encrypted copies of itself on infected systems. Whereas cleanup on the host system should be feasible, Stuxnet can not delete the malicious code that has been sent to the PLC. Therefore, Stuxnet will leave traces that may identify its presence.

3.1.9. Discussion: Stuxnet

An analysis of Stuxnet reveals communication patterns that can be matched to specific stages of the Malware Lifecycle Model. Depending on the specific network architecture and on the type of systems connected to the network under observation, these communications may appear more or less anomalous. In Internet of Things (IoT) networks where automated machine-to-machine communications predominate, the type of communication originated by Stuxnet will be highly visible.

The more human-triggered network communications are present in the observed traffic, the more difficult the anomaly detection becomes,

However, a word of warning is due: Stuxnet incorporates technology that was state-of-the-art more than ten years ago. Evolutions of Stuxnet like Duqu and Duqu2, but also recent malware variants like Gauss, BlackEnergy3, AdWind or Locky show that multi-layer obfuscation and encryption will become the standard for advanced malware. Moreover, malware like, e.g., Regin passively monitors the actual network traffic to select the least suspicious communication protocol as VPN tunnel for its command and control traffic. Therefore, network monitoring and subsequent anomaly detection systems will be challenged to identify anomalies in encrypted and obfuscated network traffic.

4. Future work

This draft aims at defining the basic framework that advanced anomaly detection methods will build upon. Plans and ongoing work include the definition of metrics and methodologies to rate malware communications, protocols, and interfaces to applications. As an example a malware's adopted scanning strategy is commonly related to its propagation speed. On one hand, aggressive probing by a malware discovers a higher number of potential victims within a shorter time, increasing the malware's speed and likelihood of propagation. The cost of this propagation speed is an increased scanning traffic that results in malware activity being detectable through network monitoring. On the other hand, passive listening malware may spend long periods of time unobserved in a system, monitoring and learning its environment while waiting for activation through potentially hidden communication channels. Discovery of such dormant persistent threats depends, therefore, on detection of highly sporadic, hidden activation signals in almost real-time.

5. Acknowledgements

Thanks to Kirsty P., Sage B., and Tanja Zseby for their comments that helped substantially in scoping, structuring and wording the initial version of this draft.

6. IANA Considerations

This memo includes no request to IANA.

All drafts are required to have an IANA considerations section (see the update of RFC 2434 [I-D.narten-iana-considerations-rfc2434bis] for a guide). If the draft does not require IANA to do anything, the section contains an explicit statement that this is the case (as

above). If there are no requirements for IANA, the section will be removed during conversion into an RFC by the RFC Editor.

7. Security Considerations

All drafts are required to have a security considerations section. See RFC 3552 [RFC3552] for a guide.

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <https://www.rfc-editor.org/info/rfc3552>.

8.2. Informative References

[CBCS] Rechtin, E., "Systems Architecting: Creating and Building Complex Systems", Prentice Hall ISBN-13: 978-0138803452, 1991, 352 pages, 1991.

[GML] Eder-Neuhauser, P., Zseby, T., Fabini, J., and G. Vormayr, "Cyber Attack Models for Smart Grid Environments", Elsevier Sustainable Energy, Grids and Networks Volume 12, 2017, pp 10-29, December 2017.

Pre-published version available for download at https://publik.tuwien.ac.at/files/publik_261089.pdf

[I-D.mcfadden-smart-endpoint-taxonomy-for-cless] McFadden, M., "Endpoint Taxonomy for CLESS", draft-mcfadden-smart-endpoint-taxonomy-for-cless-00 (work in progress), July 2019.

[I-D.narten-iana-considerations-rfc2434bis] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", draft-narten-iana-considerations-rfc2434bis-09 (work in progress), March 2008.

[Stuxnet] Falliere, N., O Murchu, L., and E. Chien, "W32.Stuxnet Dossier", February 2011.

URL: https://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf

Author's Address

Joachim Fabini
TU Wien
Gusshausstrasse 25/E389
Vienna 1040
AT

Phone: +43 1 58801 38813
Email: Joachim.Fabini@tuwien.ac.at