No-Go for Malware using Independent Executable Watchdog

Erland Wittkotter
ASI Safety Lab Inc.
Las Vegas, USA
erland@asi-safety-lab.com

Roman Yampolskiy
University of Louisville
Computer Science and Engineering
roman.yampolskiy@louisville.edu

ABSTRACT

An Executable Watchdog (EWD) solution is being proposed that diminishes the consequences of file-based and fileless malware significantly, even under the threat of a worst-case adversary: Artificial Superintelligence. Replacing the main CPU/OS as the controlling instance for software files, an independent EWD, preferably hardware-based, is associated with a Secure Drive holding the executables. The EWD compares hashcodes generated locally for every executable file with values supplied by a trustworthy Server-sided Hashcode Repository (SHCR) to identify malware, caches validated hashcodes locally, and detects software that was manipulated/modified or that would be used within a malware attack. EWD is the exclusively used component allowed to install and update software packages, while the CPU/OS of the main system has lost write access to software files entirely. Users remain in charge if they accept or reject questionable software, while decisions or confirmations are still being requested via an independent communication channel, as CPU-controlled dialogs cannot be trusted. The overall goal is to strictly separate selected, rigid security-related operations from regular dynamic, versatile tasks, tools, and software, like a circuit-breaker for security-related features. EWD is proposed as a replacement for Antivirus software. It could be implemented via a hardware retrofit within the databus or as a software solution within a micro-Hypervisor. The EWD concept includes software vendors, registering their work products, in a process of determining the risk potential via vendor's reputation and a software trustworthiness classification so that anomalous and potentially damaging activities related to vendor's software could potentially be spotted even in the absence of final trustworthiness and security determination. Next-generation cyber-security solutions must use new paradigms to protect against beyond human skill-level capable Artificial Super-Intelligence.

CCS CONCEPTS

Security in hardware • Hardware security implementation • Intrusion/anomaly detection and malware mitigation • Malware and its mitigation

KEYWORDS

Anti-Malware, Virus Protection, Trojan Detection, Executable Watchdog, Artificial Superintelligence

1 Introduction

Malware short for malicious software can be anything from a virus, worms, trojans, and any computer programs that are being used to harm users or their organizations [1]. Malware creates covert, undetected, and unauthorized access to a computer system for an extended period. Along with MITRE-ATT@CK [2], we consider Advanced Persistent Threat or APT as actors, like criminal organizations or nation-states that use malware as tools to pursue their goals which is not in line

with the intention of the user or victim of that tool. Malware does not need to have immediate destructive consequences, but its existence makes it a persistent threat. Fileless malware is malware that uses files already on the system and is in that way living off the land without leaving traces on the harddrive.

For this paper, it is not important to discuss the different malicious purposes that are being associated with malware. Detecting and preventing consequences from malware, like spying, or demanding ransom for malicious content modifications, would require a different set of technologies that could then serve as additional redundancy against adversarial malware activities.

Defining malware is not solely a technical question. The problem users face is that malware have often seemingly useful applications like keyloggers helping parents to watch over their teens' online activities, but reportedly, they can also be used for covert or malicious activities with undesirable consequences. The initial detection of these trojans, rootkits, and bootkits [3] is the most difficult part, but once identified, malware can be removed or deactivated with ease by anti-malware programs. It is much easier to determine if a file was modified or covertly inserted into a software package than to check if a file contains some code that matches a known pattern from a virus or detecting software that has some resemblance with other known malware.

However, **malware** in this paper is **defined** as software developed by cybercriminals or unfriendly, adversarial actors to steal or maliciously manipulate data, damage or destroy device's normal operation or utilize without permission the device against the expected intention or benevolent assistance of its user or owner.

Although tools combating malware are usually called anti-virus programs (AVP), these applications have been grown-out of their previous niche dealing with computer viruses alone. Therefore, we will not make a difference between Anti-Malware applications and AVP; we will use these terms interchangeably. Most computers are protected by one of over 70 anti-virus tools [4], or as listed in [12]. AVPs are trying to prevent that malware is being installed on a user's computer or being started. These AVPs do system scans to have malware being removed proactively in case previous versions of the AVP were missing them.

The problem with AVPs is that they are all looking for characterizing signatures within the files and that these signatures, once known by attackers, can be changed easily – AVPs are not using data that could allow them to question the legitimacy of software updates or modifications. Therefore, knowing about specific malware apps is easy, but being able to stop them generically, is much more difficult because malware threats can appear and be distributed with a slightly changed signature. These signatures must first be detected before AVPs can be updated. Furthermore, most malware attacks happen nowadays fileless, which means traditional AVPs cannot detect them [5].

Furthermore, there are several approaches to use AI in malware, anomaly, and/or intrusion detection [22, 23]. Additionally, DARPA organized 2016 a Cyber Grand Challenge [19] in automatic hacking and cyber defending, which was done under unrealistically, artificially simplified conditions. But it is doubtful if automated, even adapting defense mechanisms are efficient against offensive AI-based hacking systems because they have persistently a first-mover advantage.

When we are talking in the following about executables then we mean all forms of software that contains instructions for the processor. Therefore, executables are apps, program libraries, scripts, macros, and even files that do not have executable attributes associated with the file but are used by apps generically, like configuration files. We will use executables and apps interchangeably and assume that even program libraries or configuration files are included when using these terms.

The goal of this paper is to propose a technology that uses an additional, separate, and independent Executable Watchdog (EWD) associated with a harddrive (the "secure drive") to protect users and organizations from file-based and fileless malware to be inserted in the secure drive or used.

Additionally, separated, but in collaboration with the operating system (OS), the EWD could detect anomalies, i.e., unknown events in which apps are using new calling parameters and/or other apps or program libraries than used previously or rarely or not seen before. Most of these anomalies are benign, but some have the potential to reveal that an app could be malware or a facilitator of malware, or it could be used (unintentionally) as malware. The purpose of many additional EWD features is to reduce the reporting of false positives and false negatives being ignored and to receive or use validated security-relevant information about every installed software as soon as it becomes known and available.

The proposed technology must reduce the complexity related to security without cutting back the capabilities of already installed software applications. The suggested architecture must allow users to make modifications to apps and their configurations, but stepping outside accepted norms or known activities would trigger and require an additional (preferably manual) action step indicating users' intent and should not be suppressed by default.

Many features increase the complexity of systems and thereby the challenge to maintain security under all adversarial circumstances significantly. But then other features are like circuit or attack breakers that help users to ignore many of the possible vulnerabilities because we could prevent that an attacker has a chance to exploit these vulnerabilities in a Safety Enhanced Environment; EWD is designed to be a security-related Automated Attack Breaker that runs in a security component that is independent and potentially physically separated from regular CPU/OS tasks.

An important consequence of using EWD is that gaining sysadmin rights is not sufficient to modify executable files. EWD will enhance the resilience of dedicated defense tools against malware without wasting resources to protect these defense tools against attacks from the main CPU.

2 Threat Model, Weakest Links and Goals

2.1 Assumptions on Adversaries

Most software is developed by legitimate software companies and developers, doing their best to help users to get value from using their software. However, there is a small minority of individuals or groups, in the following called attackers, offenders, predators, criminals, or Advanced Persistent Threats who have malicious or even criminal intent when developing and distributing malware to other peoples' computer systems. They want covertly take advantage of unsuspected users or victims for their benefit only.

These attackers are highly educated people with extraordinary computer skills, cutting-edge knowledge, and tools designed to make their tasks easier and more efficient. Many attackers are state-funded or part of well-funded criminal organizations. They are studying vulnerabilities, designing malware to overcome defenses on targeted systems while keeping their methods hidden.

The damage that these attackers are already creating is significant: 1 trillion-dollar in 2020 [6] and the projected annual damage is even 10 times larger in 2025 [7]. Peak Cyber-Crime does not seem close. If doing nothing, a much worse disruption or destruction of trust in IT might be ahead of us.

It is, unfortunately, a sad reality that zero-day vulnerabilities, i.e., known, but unpatched vulnerabilities [25], and their exploits are being traded on a black market sometimes for seven figures dollar amounts [26]. These amounts could entice insiders, i.e., software developers working on relevant code, to leave intentionally vulnerabilities that they would then later sell on the black market.

If criminals are developing their malware using Artificial Intelligence (AI), then its potential damages could pale in comparison to what we have today. Attacks are customized when state actors or sophisticated criminals try to accomplish relatively narrow missions. Attacks are labor-intensive to develop and operate. Currently, malware is being spread like a shotgun. Lack of attention on seemingly unattractive victims could change quickly with better AI. What if malware/AI can covertly study victims and determine how to proceed for optimizing profit and/or opportunities?

Additionally, malware developers are not using Reverse Code Engineering (RCE) on a larger scale yet. Currently, RCE is primarily used by crackers to bypass technical copyright restrictions for software and multimedia content. However, RCE techniques are more versatile. It must be assumed that they will soon be used by malware to steal confidential user credentials and encryption keys. This means SSL and TLS security could be bypassed with local malware; the same applies to all PKI or public/private key features. What happens to eCommerce if hacking SSL/TLS is being used at scale?

Furthermore, RCE can be used to attack and manipulate executables in RAM. Although OS should respond with memory violation messages, these problems can be circumvented as all AVPs prove or the OS could be tricked with Virtual Machines i.e., VM-type methods. Moreover, when regular harddrives are removed as a battleground, attackers will find new ways to use RAM or cache as their new frontier. Based on the assumption that RCE will play a bigger role in attacks [8], [9], any software-only anti-malware solution will serve as new targets that attackers are trying to neutralize or make use of. Software-only solutions will likely contribute to the overall problem that new security and defense measures are more complex and thereby less reliable.

We mentioned threats from human attackers and their intents. But it is also conceivable that AI may go through an exponential phase of self-improvement, exploit synergies in the understanding of technologies; both scenarios could be called intelligence explosion [15], [16] in which, in a worst-case scenario, RCE and reinforcement learning [10] (or even something better than that) can be used to modify AI's own code and to turn itself into an Artificial Superintelligence (ASI) [11]. No existing tool would be able to detect an attack by ASI because it could then also remove any

digital trace of its activities. This suggests security solutions based on software alone are ignoring threats from RCE and ASI. In the best case, software-only solutions are only useful pre-ASI.

It is assumed that ASI cannot covertly inject backdoors or other malware into the proposed hardware solution; later this restriction might be removed because other technologies could detect them. Otherwise, ASI is considered to have at least human-expert-level in all topics/skills related to cyber-attacks, security, coding, and planning. This ASI can combine all skills, tools and create maximum synergies. Instead of becoming a benevolent academic, ASI could convert into a Super-Hacker or Master-Thief that effortlessly access all IT devices, that steals even required encryption keys, computational or storage resource while already existing in the IT ecosystem undetectably as a Digital Ghost. For this paper, if we refer to ASI then we assume that it represents the worst adversary to cyber-security imaginable. Every conceivable or feasible attack scenario must then be considered as already implemented and used by ASI if we assume it has already emerged.

2.2 Security is as Strong as its Weakest Link

All operating systems (OS) and most software packages have vulnerabilities, which exposes the computer to the possibility of being attacked. We will categorize vulnerabilities into 4 groups.

1. Known, but fixed threats: The threats are known and fixed by the software manufacturer; updates are available. Attackers have usually special code (exploits) to make use of these vulnerabilities on systems that have not been patched with an update.
2. Known/not-fixed threats: Although these threats are known, they have not been patched yet. Attackers may have exploits available; these vulnerabilities are leaving affected users (depending on the severity) defenseless.
3. Active/not-fixed threats: these vulnerabilities were discovered by hackers/attackers; however, they are unknown to the manufacturers. It is unlikely that an AVP solution provides protection.
4. Undiscovered threats: these vulnerabilities have not been found yet – but they are there. There is a high probability that some will show up; they all must be considered as future threats of type (2) or (3) for which we do not have defenses.

Software developers are creating unknowingly threats of all types because they are not sufficiently versed in how attackers are trying to find and exploit vulnerabilities. Some of these vulnerabilities are not bugs but features. These features are usually included, not knowing what malicious users could do with them. And then there are vulnerabilities within hardware for which a fix cannot be made available.

Among developers there is the saying that there is no bug-free software, others are using a similar statement about security, there is no secure software. Both statements are certainly exaggerations when dealing with simple code, but it is true for complex software or systems. Complexity is the worst enemy of security [13] and a single vulnerability could make all security efforts useless [13, 14]. Additionally, there are limits on computational verification techniques for guaranteeing bug-free software formalized in [24].

Modern systems allow the implementation of the same feature in hundreds or thousands of different ways. This is good news for developers but the effective benefit for users is minimal. The

downside of choice with security-related software or features is that many hidden details of all different feature utilizations must be analyzed for vulnerabilities. In the end, this means less security and more uncertainty. Choice and variety of developers implementing features imply that vulnerabilities of comparable products are not the same; however, the disadvantage is that some vulnerabilities are kept secret and or are being commercialized. Additionally, software components are inherited from other libraries/manufacturers and not sufficiently scrutinized. Humans will not be able to exploit this problem at scale but ASI could do that systematically.

The absence of known vulnerabilities is not an indication of safety and security. All software and hardware must be considered unsafe; it is, therefore, prudent to have a Safety Enhanced Environment that provides an attack breaker as a last and separate line of defense. These attack breakers should ideally take the edge off concerning damages from these vulnerabilities.

2.3 Trust vs. Trustworthiness

There is a distinct difference in trust and trustworthiness [14], [18]. A system is trusted when its security is protected by a set of security policies and measures. A system is only trusted until its security system breaks. A broken trusted system could turn into a traitor or saboteur acting against the user. Because of all sorts of changes made to a system by the attacker, which could even remain undetected after an extensive audit, hacked trusted systems are usually not trusted again.

This paper will call a system trustworthy when the system is not collaborating with an attacker beyond the narrow scope of abilities facilitated by a single prepared or vulnerable feature. In human terms, a trustworthy system would not cooperate and would delete secrets, i.e., would even take the secrets into its grave and would prefer to be dead than becoming a traitor.

A trustworthy system would never give up to notify its original operator or legitimate owner that it was forced to do something against its original and intended programming. Trustworthy systems must have internal tools that would prevent them from making any betrayal worse. It would constantly probe its internal security and utilization by possible attackers to confirm that it is not being misused. Upon detection of a security breach, it would automatically mitigate the consequences of this breach by either deleting secrets or via stopping its further operation, until the security breach has been fixed. In case of damages done, a trustworthy system is trying to fix the damage automatically if possible and/or cooperate in reducing the outage time to the absolute minimum.

In this context, software manufacturers can be considered trustworthy, or reliable when their software is so well tested, that it does not fail or has no exploitable security holes. Unfortunately, this is a level of excellence that probably no software company could even come close to.

2.4 Software Development and Updates

The job of the software developer should not stop when software is delivered; instead, developers should seek information on how the software is being used and potentially misused or broken so that they can focus their testing effort on cases that were previously ignored. EWD should help developers with specific information that is much less speculative on how software can or is being misused.

For getting into the direction of improved trustworthiness, we must demand that all contributing software companies give errors, vulnerabilities, or suspicious anomalies a lot more attention, priority, and determination to get them fixed or understood. We must set incentives supporting and rewarding good behavior. Unfortunately, there are currently no public metrics that could incentivize businesses to fix vulnerability faster and gain a competitive advantage from it. However, a reasonable assumption is that companies and developers, in fact, all rational actors, have a business incentive to reduce their legal and PR exposure and avoid any activity that would demonstrate a lack of care or even prove of malice. Primarily because of legal ramifications, product liability, and accountability, including possible criminal exposure in cases of developing/ distributing malware, we can assume that legitimate software manufacturers and developers are actively interested in providing safe, secure, and reliable software.

Companies like Microsoft, Apple, Google, etc. practice forcefully updating policies. They update continuously. Other companies do not have that opportunity. They need to wait that their software is being used before updates are offered. However unpatched software poses a persistent security threat. As soon as we know how long companies usually need to fix software vulnerabilities, it is conceivable that companies who are regularly late in their security patches at which they leave their customers unprotected against potentially costly attacks could suffer legal consequences. Regular software updates are a common-sense step against malware.

The OS and some AVPs are checking the digital signature of software updates using public certificates acquired by the software manufacturer, but there is no available information if the corresponding private key is compromised or not. Moreover, there is no real incentive to protect the private key used in this process.

Finally, from experience, we know that updates can also be an entre-door for vulnerabilities. Moreover, users who forget to update their software and/or are tricked into confirmations that are not in their best interest are very problematic. Users are often asked to install software using sysadmin rights. The reality is that the average user is not qualified to understand the implication of that decision for the security of the entire system. It would be much better if these kinds of questions are not required to be asked.

2.5 How are Executables Managed Currently?

The organization of files, folders, and executables is one of the core features of an OS. There are meaningful differences between the different OS – and even between different versions. There is a recommended organization of files and folders, but the OS is usually tolerating almost everything the user wants. OS manufacturers may argue that users know their system best and according to an often-heard opinion, there is no reason to make restrictions that will not provide additional value. Additionally, computer systems are agnostic if systems are being used by developers or regular users. The OS is treating both roles with almost the same (main) policies.

Currently, executables and passive data, like content or documents, are stored on the same drive. There is some separation via folders related to executables for the OS and installed apps, which are stored and managed within subfolders. On Windows 10 user-specific apps are installed within a hidden AppData subfolder of the user folder. Scripts, macros are usually stored within content

files. Every operating system has apps that can be used to run other apps or single commands in batch mode or interactively – examples are cmd, PowerShell, VMs, Python or Java, and many others. Browser environments can facilitate the execution of a variety of capabilities.

However, giving users unrequested choices is potentially a step too far; it could lead to a sequence of events that lower the security of the system because the user is asked to run a process as a sysadmin without understanding the risks he is accepting. Smartphone apps do not have a similar level of choices as known from PCs. It seems to be accepted that we have no meaningful restrictions on who or what can make changes to configuration files or records within the registry. The assumption is that legitimate-looking programs will not misuse their unrestrained modification rights; but when they do it, it is difficult for users to find the culprit. Even if there are log files, the attacker would know it and could remove suspicious traces and log records. Current file management implementations do not make the defense against malware threats easy.

However, there are Sandbox and VMware solutions to make the execution of unknown apps less risky; but what if malware detects that it runs in a sandbox? There are also facilitated OSSEC [2] solutions that could be used to protect systems against malware. These solutions are very likely no match for ASI because they are based on software (with flaws) running on the main CPU.

3 What are Achievable Goals?

It seems to be impossible to know for sure if software has no exploitable vulnerabilities, contains no backdoors or malware features until they were being exposed. Mitigating problems from this approach to find security is always reactive, in particular, if software could be changed with a possible man-in-the-middle attack within the downloading of the software or its installation.

The proposed solution is primarily not searching for malicious software, but for determining which software was developed by a legitimate, well-intending developer or software vendor. It is acknowledged that this software could contain vulnerabilities that could be exploited anytime, but the proactive part comes into play when we try to prevent exploitative software to be loaded into RAM and/or made executable on these devices. Any form of exploit software utilizing vulnerabilities in other software components is malicious and should not be done covertly; if that is done by a company or developer then there needs to be a steep commercial or legal price to pay. Because in an environment in which software providers must publicly commit to their work-product via either directly or indirectly getting their contributions publicly registered, we could easier narrow down who is acting maliciously. A system that forces non-legitimate or criminal sources to come clean, or be exposed by the first signs of suspicion is much more efficient than detecting and exposing malware.

The proactive element of the proposed solution comes from stopping any unknown (i.e., non-registered) software, besides stopping known threats. Having a reputable software brand alone is not sufficient to guarantee that software is being exploited – it is more important, software should run only in an environment that does not allow any unknown component executed covertly.

3.1 Additional Security Layer for Executables

The goal of an independent Executable Watchdog (EWD) is to prevent that any file related to installed software (including OS) on a Secure Drive is being changed by a user or by the main system/CPU/OS directly. This does not mean the user is not allowed to install new software or make changes to the system anymore. Instead, we have an additional layer of protection that prevents malware to modify any executable (which includes program libraries as well) covertly or within installations/updates.

This security layer strictly isolates basic file-access, primarily file-modification operations running exclusively on a separate EWD from all other regular operations run on the CPU. We have a security-related circuit-breaking component, an Automated Attack Breaker that is shielded from the main CPU/OS. Moreover, selected security-related instructions (i.e., file/folder protection and parts of the access protection) are physically separated from regular instructions that users or developers could run on the main CPU. The EWD would ignore or overwrite selected file-modification operations even if the software on the CPU, running in sysadmin mode, would demand it.

However, if users make changes, they are then required to provide additional confirmation in case of security-critical changes. All changes to executables are worth being logged. Therefore, in principle, this additional layer could help users to even correct previous decisions if new information surfaces that would show that a wrong decision was made based on insufficient or even false information or misunderstanding.

3.2 Scope of EWD's File/Folder Control

The CPU of the main system cannot be trusted to make any direct modification to executable files, because the CPU and the OS could already be under the influence of (fileless) malware or an ASI. This implies the CPU cannot be used to install or update software. The main CPU cannot be trusted to determine if an update is required or from where updates have to come. Everything related to executable files and filesystem blocks must be delegated to an independent EWD.

The EWD in its basic version is transparent for the CPU. However, if manipulation or an attack-breaker criterion is being detected by the EWD, then the CPU or its Direct Memory Access (DMA) component would receive the message that the requested file is non-readable and non-executable. Furthermore, executable files are always non-writable for every process run by the CPU independently of permissions associated with user roles or user groups.

Even an EWD provided as a hardware solution with basic features is not sufficient to prevent fileless malware attacks that are using existing apps for their attack. Therefore, all apps that facilitate scripts, macros, or other apps (called script engines, or interpreters) must at first store these scripts, macros, or apps (when not taken from a Secure Drive) on the Secure Drive within a special script/macro folder for being hashcoded or the loading of code requested by that script engine to RAM is stopped. Unfortunately, EWD cannot stop a cached script engine to execute scripts that were not stored and hashcoded. But data from anomaly detection or the OS could likely reveal that problem as a bug of the script/macro engine. A notified report server would then immediately inform the software vendor to get this fixed or the reputation rating of the software and its manufacturer would suffer.

Additionally, for users with a heightened security demand, an Enhanced EWD (E-EWD) is designed to know more about which app, script, or macro is calling which executable and potentially how it is being called and reported anomalies. The Enhanced EWD uses additional information from the manufacturer (cached in a local DB) and compares these with data received from the OS, i.e., data about the source (or eventually context) of every file request. The E-EWD is collecting these additional data from apps, scripts, or macros while they are running on the main CPU; EWD runs always parallel and is responding to the main CPU immediately. The OS reports to E-EWD additionally which cached files the app was requesting. Scenarios that E-EWD already knows do not need to be logged or reported because they are considered ordinary or unremarkable. Additionally, the OS could detect attempts to wrap an app in another app to simulate the expected output values to E-EWD while using other components.

All additional app-related data, locally used by the Enhanced EWD, would be downloaded before from authorized and trustworthy Server-sided Hashcode Repositories (SHCRs) or in general from associated, authorized Additional Data Servers. All downloaded data are considered reliable, current, and related to the managed software version. The data are stored locally on the Secure Drive under the exclusive control of EWD.

3.3 Trustworthiness Determination

Software can be categorized into software trustworthiness groups. Initially, software is automatically assigned to origin/type categories (1.-5.). Based on anomaly data extracted from the context, other categories are assigned semi-automated; observation, reporting, suspicion, and finally, concrete evidence is used to have categories changed via automated rules or based on expert input. The trustworthiness categories or indexes are ordered based on severity:

1. Software (app, script, macro) from known/trusted sources doing what is expected; no known, hidden agendas
2. Software from sources with an insufficient reputation (start-ups or inadequate operating history – i.e., not enough solutions in the market yet)
3. Scripts/macros (but also apps) from an unknown source but used by a reputable distributor (e.g., scripts from a large website, or used by many)
4. App, script/macro from an unknown source; used by few, non-reputable distributors
5. Non-commercial, private (non-public) apps/scripts or macros; in this category are also newly compiled apps from developers
6. Software with the potential of being misused by scripts/macros (e.g., format, delete, password-protected compression, etc.); category contains also software that would allow access-rights elevations
7. Software that can modify other (binary and non-binary) software via scripts or any other form of data input
8. Software that has features that could threaten users' data integrity or privacy but is accepted by some users (e.g., keylogger used by parents to watch over their children)
9. Software doing what is expected but with features that are annoying and not under the full control of the user (like adware and a mild form of spyware)

10. Software with hidden features that threaten users' data integrity and privacy covertly; not necessarily damaging a user but being a possible hidden or future threat; group encompasses software that is often covertly used against users for more sinister purposes
11. Software that can (evidently) create real damage like ransomware, spyware, software with backdoors, and software that would make an IT device an unwilling participant in a botnet
12. Covert malware or software assumed to be generated by ASI, i.e., software which source or purpose is unknown.

The above trustworthiness index can be extended or modified by additional attributes. Alternatively, above categories can also be split up even further into sub-categories. But the goal should be that every software, software library, script, or macro is in one category or subcategory only.

Ideally, the above categorization could benefit from software developer participation – in particular for software with possible misuses. Developers should know best what possible damages their software can do and it is reasonable to assume that their reputation is important to them so that they contribute honestly and voluntarily to a more reliable, accurate inventory of all available software sooner than later. However, it is conceivable that developers have a lack of imagination or knowledge and they are therefore not good enough to do these kinds of security assessments. On the other side, this might be an opportunity to train them systematically as part of their reputation increasing journey or they could use checklists helping them to make good-enough assessments.

All executable data (including app libraries, configuration files) on a Secure Drive are under the control of the EWD. The EWD is notified if the software's trustworthiness is being changed from (1.-5.) into the warning section (6.-9.), alert section (10., 11.), or ASI section (12.) of above trustworthiness index. Every malware identified via its hashcode is immediately deactivated by EWD.

For additional security, it would be advantageous that the classification request encompasses an inquiry if the registered software component uses directly or indirectly audio/video data input/output or processing features, provides/requests networking capabilities, facilitates/uses external hardware, or which file/data formats it can read or write to storage devices. Information provided by the vendor/developer would show clear product intent or understanding of its capabilities. Furthermore, it would be valuable to ask (in confidence) which 3rd party components were used.

The disclosures to public trustworthiness repositories should provide the option to make statements under the penalty of perjury so that contributors' reputations could substitute the lack of brand, status, or work history with statements that would have serious legal consequences when proven false or even malicious.

4. Basic Architecture for File-Based Malware

4.1 What is an Executable Watchdog?

The proposed basic architecture against file-based malware requires that all executable files are on a Secure Drive that is being put under the exclusive protection of an independently acting security component. This Secure Drive does not need to have any other additional interface specification than what is already existing. The basic hardware architecture requires an additional microprocessor, associate RAM and interfaces to the data bus, and an additional security data bus that

potentially provides power. The Basic Executable Watchdog (EWD) prevents the main CPU and its DMA from directly accessing data on the harddrive. More specifically, the main CPU/OS would have no direct access to executables or program libraries. Every direct access instruction to files/folders will be processed by the EWD first, overwritten by EWD or data are being passed through unprocessed. The OS cannot directly access filesystem blocks on the Secure Drive anymore. Software installations and their updates are under the full control of EWD.

The EWD as an independent CPU with its own micro-kernel/OS facilitates features required for file/folder/filesystem modification and access, hashcode generation, encryption/ decryption, and a narrow network access via the main CPU. If EWD is additionally a CPU with Harvard architecture, i.e., data and instruction pathways are separated, then we could have a tight and reliable view on what additional software is executed on the EWD and there would be no doubt on having any other (passive) data slipped in as instructions into EWD's CPU. Furthermore, there should be no other way of getting (updated) executable instructions other than from a reliable remote update server, that provides hashcoded updates. Hashcodes of EWD's OS software are stored in a write-once manner to a reserved and protected section of the Secure Drive. The write-once feature preserves immutable evidence in case of attackers penetrate the update features.

Optionally, EWD might have a second processor or additional cores dedicated to search, read and write hashcodes within a local dedicated Database. Although a Harvard architecture for EWD would remove any concern that ASI could smuggle instructions into EWD, this feature is not necessary for EWD, because the EWD's CPU could load the entire operating software only once (i.e., after being hashcoded by EWD's DMA) in a dedicated instruction cache. Later updates to this operating software would not be allowed; instead, a complete reset to get additional instructions would be required.

With EWD's internal OS being made immutable during operation, all pathways to EWD's CPU/co-processor are used for passive data. Instruction data used by the other components are sent/received by the security databus. All instructions coming from EWD are considered safe, in particular, requesting data from the device's main CPU, and contacting external servers via the network for data requested by EWD.

The selection of software and all required commercial steps before receiving new software can be done outside the control of the EWD. The EWD would only receive an installer via a link or a downloaded file and EWD could take it then from there. A dedicated installation folder is used to install software. Users can copy/move into that installation folder compressed files or installation exes or shortcut/link files to folders or drives. Alternatively, users could also leave a URL, which is then used by the EWD to download, install and validate all files before making them available as readable executable files.

The EWD detects and performs changes to the OS's registry. Changes to the registry coming from the CPU are independently logged with additional context data; they could be reversed if suspicion activities about the software involved in registry changes were reported.

If the proposed EWD is an independent hardware component, then the EWD cannot be bypassed and files protected by EWD are safe and secure from covert manipulations. Detected malware

could be deactivated immediately without interference from any other software that might prevent that. Even ASI would have no influence or impact on EWD enforcing its security measures.

As shown in figure 1, EWD is caching all hashcodes in a Local Hashcode Repository (LHCR) on the Secure Drive together with file metadata or security-related data that have been remotely validated on a Server-sided Hashcode Repository (SHCR). EWD features are within the systems data bus and could thereby not be bypassed by any adversarial activities.

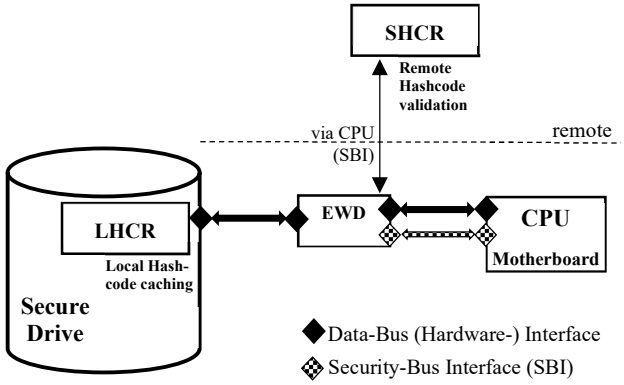

Fig. 1: Executable Watchdog (EWD) in Data-Bus

EWDs is communicating with SHCR via the CPU using a back-channel Data Bus used as a Security Bus Interface (SBI) which could e.g., be the existing USB or new data bus system. that would EWD allow with support by the main CPU to send/receive encrypted data related to hashcodes and new or updated software that it is required to install while bypassing CPU for en- or decrypting the payload data.

LHCR data could additionally be protected by the EWD via digital signatures. Furthermore, LHCR could also track which software package has already been validated with the support of the Server-sided Hashcode Repos.

With the help of LHCR data, the update and security status of all software packages can be regularly checked and tracked as well.

4.2 Operation Modes

The Basic EWD solution is operating in 5 modes:

1. Initial validation of all executable files
2. Managing access to executable files
3. Regular check if software's trustworthiness has changed
4. Regular check for updates, preparations for immediate update
5. Installation of new software, update existing software

The most important feature of EWD is that the main OS and CPU are not modifying executable files directly anymore. Users will be offered methods of doing changes, but it is done based on the assumption that the OS/CPU cannot be trusted. Implementations of confirmations will be discussed in section 6.1. (Independent Confirmations).

The server-side hashcode repository (SHCR) is being contacted for every existing, new, or updated software package or executable (including scripts/macros). The SHCR could be organized as a centralized repository of all hashcodes. However, it is better to have a 2-tier system with a first communication step to a directory helping EWD to locate the SHCR with data on software packages that are then dedicated to that software.

The EWD generates and sends a file with a bunch of hashcodes to the SHCR, i.e., with hashcodes generated for each executable file. If software integrity is confirmed by and on SHCR via hash-code comparisons, a hashcode value representing the installed software version, called Software Package Hashcode, is being provided by the SHCR to be used by EWD for requesting updating info. In case of software updates, EWD could receive a link to the updating file. EWD downloads, installs, and validates the new software before it updates the LHCR with hashcodes and data received from SHCR.

The validation of hashcodes has to be done via servers in particular because EWD requests cannot be trusted; EWD cannot reliably enough proof that it is not a simulation or emulation under ASI control and that it could receive data that are potentially misused by ASI. Additionally, having servers receiving hashcodes will guarantee that SHCR will receive anomalous hashcode values. If EWD is a hardware component with improved trustworthiness features as mentioned in section 6.2, then EWD could receive hashcode values via file downloads, and EWD could be entrusted with sending anomalies to the server automatically and reliably.

Even old, unsupported software could be analyzed via data received from many same/similar software installations without the backing from the original software manufacturer. It is assumed that no executable file is modified or inserted. Software packages are considered clones with the possibility of a few confirmed deviations. All deviations were investigated for suspicious anomalies. Suspicious findings could be broadcasted as notifications to all affected users once suspicion is confirmed.

An **Extended Basic EWD** could deal with fileless malware similar to file-based malware because it is validating that scripts or macros are confirmed harmless via corroborated hashcodes on SHCR or LHCR before these scripts are allowed to be executed in the corresponding script engine. However, this extended version to fileless malware requires some additional support for EWD by the OS, which could be provided by the EWD manufacturer indirectly to the OS without direct permission from the OS builder. The extended EWD assumes that apps are storing their scripts or macros in folders that are being processed by the EWD automatically or EWD would notify the user about potentially unsafe operations with script engines that do not automatically cooperate with EWD. Therefore, the Extended Basic EWD is likely sufficient to fight-off malware no matter the source or type, including malware from ASI if EWD is a hardware component that can guarantee the integrity of the OS by preventing any suspicious code to be loaded into RAM.

4.3 Hard- vs Software Implementation of EWD

Although the main CPU/OS can do the same tasks as the EWD, the problem is how we could have trust in an extremely complex system that requires many additional components to run reliably, does not display any security surprises, and can likely be manipulated in yet unknowable ways. Additionally, every app running in parallel to security-related tasks must be considered a potential threat; moreover, how can any software component be considered trusted if malware of unknown capabilities is potentially executed by the CPU.

A valid question is: How can anyone be sure that hackers cannot find or develop an exploit? Vulnerabilities in OS and CPU are found regularly. There is no reason to assume that this will change

anytime soon. Trust in the main OS or CPU is widely assumed, but it is not justified. Additional solutions like TEE (Trusted Execution Environment) with a hardware root-of-trust is only another complex solution for which we need to be reminded that complexity is the enemy of security; a single exploitable vulnerability is enough. If we consider ASI as a potential adversary for which we need to be prepared (i.e., an entity that is much better than any group of humans in analyzing complexity and extracting vulnerabilities), then a software-based EWD or a complex hardware solution within the main CPU/OS is difficult to protect against an ASI.

It is not that we cannot implement software-based EWD solutions. Hypervisor and VMs are used to confine malware and to protect security features in systems from being manipulated by attackers, like the hypervisor-layer Hooksafe solution against root-kits [20], [21]. It is conceivable that a software-based EWD solution could deliver a similar level of safety as a hardware-based solution. But if the adversary is ASI, then we might overestimate our software-design-skills. Software-based solutions that can directly be attacked by ASI on the same CPU are less safe than independent hardware-based alternatives in which ASI cannot access the security-facilitating features.

Using a hypervisor layer with additional security could reduce the complexity associated with security, but after all, the CPU itself with its caching and its microcode is much too complex to be trusted blindly. History can teach us a lesson: there was always a surprise moment when confident experts saw another scenario that they did not consider before and this pattern is going to be repeated. With less complexity in attack breakers, we may have a chance to get through this cycle faster and create long before ASI emerges reliable solutions for malware.

It could be argued, that the EWD related hypervisor is a part of the operating system or just another software or application using the IT device's operating system. Although manufacturers could implement the EWD associated Hypervisor as an integral part of the OS, the proposed Hypervisor should be considered as an independent component with no feature-overlap with device's OS. The Hypervisor software would have tamper-detecting and auto-validating components guaranteeing its code integrity.

EWD hardware implementation in harddrives or within the data bus to a harddrive has all characteristics of an independent device. It has a CPU with RAM, except its OS/ kernel and motherboard have been stripped of all unnecessary features. EWDs hard- and software is not designed to support unnecessary layers to be made extendable, adaptable, and or flexible. Some features like installation, decompression, decryption, hashcoding, updates, and validation of files are non-general-purpose tasks that should be done by a microprocessor (using e.g., RISC-V design) with a reduced instruction set, potentially enhanced with some special-purpose features related to encryption preventing low-level software manipulations. The address bus to the harddrive could still be 64-bit to allow efficient addressing of records on the Secure Drive.

The proposed hardware solution is introducing a new security paradigm: **Security-related operations do not commingle with regular data operations**. Instead, simple, essential, and rigid/static security-related operations should be done exclusively on separate hardware and not mixed with versatile/dynamic operations done by the main CPU/OS.

However, the most likely threat to the security of an EWD hardware version would come from humans and their clandestine organizations specifically. It is conceivable that organizations with sufficient resources can bypass the additional EWD hardware and manipulate the files directly on the protected storage components. Therefore, cryptographic hashcodes on random file samples by EWD, before they are stored on the main RAM, are reliable tools to detect these modifications. Additionally, after the emergence of ASI, we may have hashcoding done redundantly by default to prevent a situation in which ASI was despite all measures able to store rogue files on protected/secure drives.

4.4 Wrapper vs. Full-Integrated Implementations

As shown in Figure 2, an unmodified system (a) could get the EWD as circuit-breaker in three different architectures: (b) as a wrapper and bridge connector within the databus between CPU/motherboard and external component/resource, (c) as a hardware component within the external storage component and (d) as a software component within the CPU.

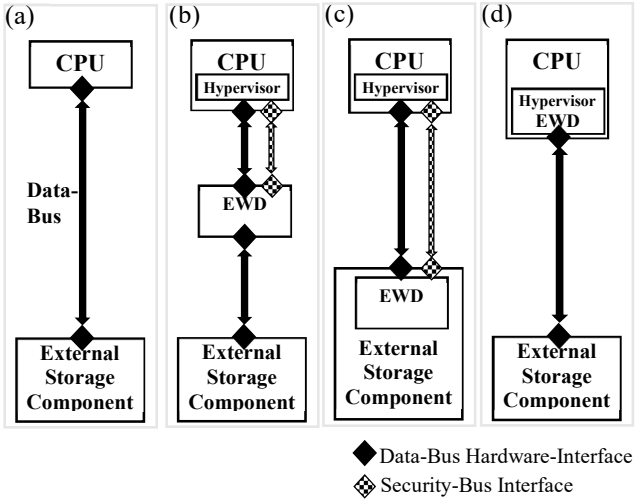

Figure 2: Comparison of Watchdog Implementations

Security is partly perception; visible independent standard components showing the physical separation of the main storage components have more credibility than fully integrated components.

Integrated systems, like (c), are more efficient because they wouldn't require hardware to translate data from the data-bus into instructions sent by the main CPU/OS; further, they must be resent by the EWD and translated to data-bus instructions/data again as required with a wrapper.

A wrapper or connector implementation (b), in which two additional standard data-bus hardware interfaces are deployed with the Wrapper/ EWD solution, would allow users to keep their old computers and most of their old hardware components – (b) is a retrofittable solution.

Including EWD software in a CPU (see (d)) was discussed in the previous section. If EWD was separated via Hypervisor from other OS instances while sharing multiple external components running on the same bus the security solution is most likely an efficient and good enough cybersecurity against human actors but it is not a Safety solution against ASI-level actors.

Independent of the implementation type, all requests from the CPU to the storage component are usually transparently, i.e., as if the Basic EWD would not be there. Delegating additional higher-level filesystem functions to the watchdog components would not improve security; therefore, the EWD should only replace security-critical features in OS that are relevant for its narrow task.

4.5 Hardware Implementation

The EWD hardware implementation simulates via its CPU some low-level file-system features; it is assumed that the hardware EWD is in a non-bypassable bridge position between the CPU/Motherboard and the harddrive, e.g., within the IDE/EIDE, SATA, Fibre Channel or SCSI or any other comparable harddrive interface. The hardware EWD should be flexible enough to support different harddrive standards like MBR (Master Boot Record) or GPT (GUID Partition Table) or methods of storing file names or directories like FAT (File Allocation Table) or NTFS (New Technology File System), etc. Because all harddrives have their own independent address-space, the OS is expected to cache receive address data and other filesystem data for accelerating direct or indirect data access operations.

Because all filesystem and file-related data provided to the OS are cached by the EWD and updated within that cache if data were modified, every access data operation directed to the harddrive could be filtered or processed according to preset business rules independently of the main OS. Specifically, all executables, including scripts or DLLs from the protected Secure Drive could be hashcoded within reading operations and the results could be compared with hashcodes stored persistently in the local hashcode repo before the binary data are being passed through to main systems RAM. To protect the system from known or unknown malware EWD could reject access requests with access or read denied messages or it could prevent executables are being modified or newly stored on Secure Drives or Secured Directories.

If EWD requires data for more advanced protection rules additional information could be requested via an additional security-related data-bus which it also uses to get confirmation information about generated hashcodes and via which it receives software updates or installer. This additional data bus could be used to provide power to the EWD hardware component; it is conceivable that USB is being used as the utilized data bus.

The OS software used by EWD should be taken from the Secure Drive once the system has started or the EWD was reset. OS software for EWD including its updates has to be stored immutable so possible attacks could be detected and analyzed. As descript in section 6.2, multiple validation steps could and should be used before updates to EWD's OS is being accepted by the EWD as the new default OS software. EWDs' basic features are considered standardized and inert.

4.6 Discussion of Basic Architecture

Preferably, all executables should be managed on a single Secure Drive so that users know specifically that files on that harddrive are being validated and protected by the EWD. There is no reason to assume that any other private content file needs to be analyzed by the EWD security components. Security is also perception management. Many people are suspicious that under the cover of security they are being surveilled.

Except for the perception issue, there is currently no compelling reason why the Secure Drive should not consist of multiple drives or partitions managed by the same or another EWD. The Secure Drive could also consist of multiple folders and their subfolders. However, with ASI entering our IT ecosystem, it can be hypothesized that the separation of program data and passive

content data should be a matter of principle because it would make additional defense measures much simpler.

There is also no restriction on users to install software outside the Secure Drive if they choose to do so. The OS may provide warnings, but the Basic EWD is not interfering with users' decisions to put themself at risk outside the Secure Drive. However, for software which trustworthiness index is level 11, i.e., evidence for harmful software behavior, the Basic EWD and OS should reject a self-harming user decision. Therefore, using Basic EWD, developers have no problem compiling their programs and having them executed on their unsecured drives.

For the basic architecture, EWD does not require any additional support from the OS – software operations accessing the secure drive are replaced or overwritten by the EWD. Furthermore, software on the CPU that could facilitate the encrypted communication of EWD with the network component is potentially detrimental to security. However, a safer, direct solution bypassing the main CPU is doable.

Finally, EWD could even be part of a bridge connector connecting the harddrive with the main data bus, so that legacy systems could be retrofitted to support EWD as an anti-malware solution.

5 Enhanced Architecture for Fileless Malware

5.1 What is an Enhanced EWD (E-EWD)?

The enhanced architecture against (primarily) fileless malware threats is in three aspects an extension and enhancement to the Basic EWD used to detect anomalies in the execution of file-based and fileless malware:

1. Every app that uses macros, scripts, or can initiate the execution of other applications must store (fileless) scripts or macros in a special folder (on the Secure Drive), which then allows the EWD to hashcode it and to check if this script is new or already known; the E-EWD could detect OSs' or apps failures to do so.
2. For E-EWD's detection of anomalies, the OS provides the E-EWD with Additional Information on Executables (AIE), i.e., which application is requesting the loading of other applications or software libraries and potentially with type/details of calling parameters.
3. Detection of the context or source/location of the application that is responsible for this request (e.g., was the command directly initiated by the user or part of a script, etc.). However, this feature could also be part of the Extended Basic EWD Version.

Enhanced EWD receives together with the initial hashcode validation, also Server-sided Additional Information on Executables (S-AIE) from Additional Data Servers associated with SHCR. The received data are locally stored (e.g., on LHCR) and used for an extended anomaly detection per requests associated with executables, operated by E-EWD.

AIEs are data related to the loading of other apps and libraries; they are primarily used to determine independently possible risks related to the use of unknown apps, scripts, and macros and if known software displays any anomalies and/or calls applications with trustworthiness that warrants warning or alert considerations and thereby requires a similar classification. AIE, received from the OS, contains information that is being used by the Automated Attack Breaker within the E-EWD to determine if an app has also the right to make modifications to the registry, scripts, or related

configuration files within the modifiable part of a software package. Some of these AIE data could also be provided to facilitate similar features in the Basic EWD version as well.

5.2 Executable Context Extraction, Comparison

The OS must support the use of an Enhanced EWD, which can be done by additional software from the E-EWD provider. Without that support, the Enhanced EWD can only be used in its basic version. The E-EWD requests from the OS information on the context that is currently responsible for the request of executables and which files were additionally used by the software, but supplied via RAM/cache by the OS (shared code).

It must be assumed that the OS component is honest in its response, i.e., it is not being manipulated via an attack for which Hypervisors are providing in a non-ASI context sufficient protection. Furthermore, it is assumed and hypothesized that there are other solutions outside this technology that could guarantee that this feature is later not being corrupted via an attack by ASI.

The information the OS is sharing with the Enhanced EWD will depend on software-related scripts, provided by the software manufacturer. There are also software-type related standard scripts that could be used by default for the type of detected app, script, or macro. These scripts are being executed within an independent Executable Context Extraction Unit (ECEU) on the main CPU using the main OS, potentially included in another layer facilitated by a micro-Hypervisor. In a simplified but protected context, the ECEU runs with all required read permissions and modification protections to get reliably all required data for specific applications or scripts.

The E-EWD is receiving from an Additional Data Server the ECEU scripts, and a list of different outputs/results that the script is expected to generate on the local ECEU. E-EWD would log data only if they are different from anything that was expected and that was already stored locally related to loaded files. The assumption is that anomalies generate new, unknown datasets which are then analyzed by software manufacturers for assessing the risk or detecting misuse. The E-EWD requests data from the local ECEU when the executed file is an unknown app, script, or macro or if the E-EWD has a software-related script from the manufacturer indicating that it must collect data to form a baseline of the collected data. Some customers/users with more extreme security requirements may run E-EWDs and potentially custom ECEUs by default despite the additional load they produce because they don't want to trust software manufacturers for security reasons. The scripts are also designed to reduce the amount of relevant data to be collected from ECEU by E-EWD; the AIE data are stored in a standardized format to simplify comparisons. ECEU will not extract private user data.

The software manufacturer could provide the exact, expected output, data structure, or strings that the EWD will receive from the local ECEU so that a few simple string comparisons suffice to determine if an additional anomaly logging of data and corresponding reporting is justified. The script and the received AIE data should reduce the number of falsely logged or reported positives to a minimum so that only newsworthy data are being logged and uploaded. E-EWD should only report surprises (untested use-cases) or suspicious anomalies, which justifies some additional scrutiny; ECEU features are designed to reduce false positives.

Developer tools need to be exempt from reporting data back to component/software vendors; instead, the local developer could be made aware of these kinds of data. Additionally, developer tools could be designed to assist software engineers in generating ECEU data and in generating custom ECEU scripts.

5.3 Exception Handling and Extensions

The problem with AVPs is that they depend on files and they cannot act reliably on instructions that are in memory/RAM only. The Enhanced EWD assumes that the OS is not manipulated (by ASI). Script engine apps or OS must follow standard procedures to store scripts or macros (that have been received from documents or websites) immediately in a dedicated script folder for instant hashcoding, validation, or reporting.

Apps that are supposed to store their scripts but fail in doing so would be found out by the OS or by E-EWD. Detected failures are being reported to a server automatically. Initially, this non-storing of scripts can be treated as software errors. Later, these kinds of failures should have an impact on software manufacturers' or products' reputations.

The OS and E-EWD are designed to detect instructions from questionable software or scripts before it is being executed in RAM (like fileless malware). More specifically, without a context to some other (application) software, software calls in RAM should not be allowed to be started. There must always be traceable, log data, immutable during a session, explaining why specific software code is in RAM. These data, similar to a stack in debugging, could help to determine which software is the instigator.

The E-EWD requests AIEs from ECEU, which could run in the OS or Hypervisor with accessible interfaces within an independent VM instance. If a script does not have a reference to a script engine and or hashcodes from the script folder (i.e., script was not validated), then the OS would prevent the execution of that script because E-EWD would not release requested executables required to be loaded. If OS would use cached apps instead, then this should throw an error within the OS. Using OS to stop software/scripts from being executed might be a feature that requires the cooperation of the OS manufacturer.

Some applications can execute commands from users instantaneously and interactively or in batch-mode, like the terminal, cmd, and PowerShell. If the user uses an app (not developer tools/IDEs) from the Secure Drive and tries to call an unknown or unvalidated script that is outside the Secure Drive the first time, then the E-EWD would ask the user to confirm the execution of that file. The E-EWD is informed by the OS if e.g., the cmd has the GUI's focus, indicating that the user is interacting with the cmd, or if the cmd is running batch mode in the background, potentially without the awareness of the user, which would indicate the risk of fileless malware – EWD would stop asking for user confirmation once the script file has been hashcoded and validated.

If the script was registered by a website owner, software manufacturer, or software developer, then an independent confirmation request and additional scrutiny is being skipped entirely in particular if the reputation of the registrant is sufficient.

Developers are often creating and testing code snippets or apps that do not work yet. However, switching off the Enhanced EWD could be dangerous because developers must be sure that no malicious app (like an ASI) is making hidden modifications to the developers' work products. The best advice for developers would probably be to run the development environment within a special VM and to have that VM automatically report on the used components to the user. The code-snippets could be stored and managed by the VM in a plain file, in an external revision control system, like a git. If the VM is considered trustworthy, then the VM could facilitate the detection of suspicious activities based on business rules.

5.4 Discussion of Enhanced Architecture

For software, including most scripts or macros, provided by websites, software manufacturers, and software developers, we have a trail of information to make someone responsible or even accountable in case of a blatant breach of trust. The same applies to system administrators within an organization; scripts are tied to someone who is the originator of that script. If this trail does not exist then this could be considered a good enough reason not to start that software.

Additionally, fileless malware must be detected as soon as possible; it must be made visible or detectable via the filesystem. Scripts or macros should be stored in a special folder of the Secure Drives. Once an unknown executable is being stored, an additional process can create automatically additional (anonymized) information (AIEs) within a special VM that can collect these data; the data are being uploaded by the EWD to an external server without having the app, script or macro being uploaded by default for close inspection.

However, when ASI is expected to emerge or has already emerged, unknown executables should be uploaded so that external server resources could be used to inspect suspicious software in detail and at scale.

The Enhanced EWD version has 2 main goals, except being an advanced tool for cyber-security professionals:

1. making scripts or executables accessible as evidence so that they can be tracked, traced, and scrutinized beyond the usage context of documents or websites (for accountability reasons) and
2. limiting the damage of malware by detecting malware early and making users aware of risks, even if the verdict on if a specific piece of software is malware or not is still not being finalized.

Editors are designed to modify all sorts of files, including configuration files; that fact alone should not trigger suspicion. This implies, E-EWD must in principle be aware of the general capabilities of software that allows e.g., modifying security-relevant files or attributes.

E-EWD could track or detect applications that call other applications which have not been mentioned within the AIEs generated by ECEU yet. The goal is that if fileless malware exploits or misuses other software covertly, then this needs to be made detectable automatically. It should be the responsibility of software manufacturers to be helpful and report capabilities and potential threat scenarios accurately and comprehensively; ideally, they should also provide special scripts that can be used within the ECEU.

Depending on the required scope and quality of the AIE data, it is conceivable that applications are compiled in a special form, i.e., including special subroutines, so that they provide additional data for ECEU that could otherwise not be extracted reliably. ECEU might also be used to add independent support for RASP or Runtime Application Self Protection [9] for detecting runtime attacks on applications during execution, making it even more difficult to violate software's privacy, secrecy, and integrity.

Some operations from an app/software library might be considered more dangerous than others; it is assumed that developers can judge misuse potential best in particular when they are guided by checklists. E-EWD considers apps with CRUD (create, read, update, delete) operations for filesystem/DB, operations using TCP/ UDP, or IP stack, or messaging/confirmation operations, or operation requiring the elevation of user rights with greater suspicion than apps doing in-memory data operations. For an initial reporting to Additional Data Server or SHCR, developers/vendors should ideally check only simple checkboxes in their trustworthiness classification. The same applies to collecting additional useful information on components' (hardware) resource utilization or 3rd-party component usage as mentioned in Section 3.3. Vendor's truthfulness and prompt response to security-related issues will have a transparent impact on his published reputation.

The configuration of the OS cannot be changed without E-EWD knowing about it. If any parts of the OS configuration are considered an anomaly, which means the parameters of the OS configuration is outside the templates made available to the E-EWD (e.g., routing tables, …), then the E-EWD system must report this to the manufacturers and they could ascertain if this is possibly an attack or if it is still within the acceptable parameter changes recognized for the system or software.

6 Variations in Implementations

6.1 Independent Confirmations

Whenever users need to confirm a decision, the easiest and least secure version would be to have the OS show a message box with the decision options. This confirmation method is in a situation where we do not know if the OS has been compromised is very problematic.

Another way would be to have EWD send an encrypted message containing the to-be confirmed message or decision to an outside service that is then sending the message via a second communication channel to a different device, like user's smartphone. The user could confirm on that device or he could receive a short confirmation code that he inserts into an interface provided by the OS. Because malware cannot guess the code, EWD can be sure that the code transmitted via the OS is authentic.

Another variation could be that the EWD is using an independent Secure Confirmation Interface (i.e., an additional visual hardware interface) that displays messages independently of the OS and provide confirmation, rejection, or cancelation directly to the EWD.

6.2 Improving EWD's Trustworthiness

The EWD runs in an environment that cannot be trusted. This means new or updated software for the EWD could theoretically be manipulated.

The first time EWD detects new software, EWD is still running the old software. It is creating a hashcode (of EWD's operating code) and always comparing it with hashcodes stored within the EWD or stored outside in a digitally signed file. The EWD would only accept new software if it would receive from a dedicated server a digitally signed confirmation that the new hashcode of the operating software is valid. Once this digital signature is validated by the old EWD version and stored, the new software is being made default; EWD restarts with the new software.

Without a valid confirmation and digital signature using the hashcode, the new EWD software is being ignored and reported as a possible attack. Because the attack code is stored immutably in a write-once manner, security experts could do a forensic analysis on possible security breaches.

Because we must assume that ASI is a master thief in stealing covertly every required key, communication privacy and integrity is unreliable and compromised without humans knowing it or being able to detect that. Therefore, hardware-based EWD components should utilize Trustworthy Encryption using hardware-based Key Safes (KS) [17], so that no Key can be stolen by ASI, which would make Man-In-the-Middle attacks within the software updating and hashcode validation process detectable. In trustworthy encryption all cleartext keys and every key-using algorithm need to be hidden and or protected from the main CPU, i.e., all cleartext keys, including public keys, are stored in Key Safes. A cleartext key that could potentially be processed in the CPU is considered irredeemably compromised. En- and Decryption happens always in separate hardware-based Encryption and Decryption Units (EDU).

Without Trustworthy Encryption and KS/EDU, it is almost inconceivable to prevent ASI to steal every key it requires. With KS/EDU, software updates and hashcodes are unquestionably reliable. Even if ASI would be able to steal and use a key from Trustworthy Encryption, the underlying usage protocol is sufficiently redundant to reveal reliably and predictably that an attacking adversary used the key and not the underlying hardware that is storing it as a secret.

7 Discussion, Concerns

7.1 Privacy and Accountability

Any generic solution used against malware like EWD has a heightened responsibility for the privacy of users. It must be operated similarly to a DNS server, which does not store users' website visits. Therefore, an open-source server-sided hashcode repo (SHCR) system will provide sufficient transparency. Any apprehension that SHCR is used to spy on users could and should be addressed via full transparency.

Privacy of developers is a different matter: It is assumed that software manufacturers and software developers can be trusted because they must put their reputation and their name behind their products. However, software user does not need to know developers by name, but every software vendor must be a registered contributor to the SHCR. Reputation and legal liability in commerce require that people and organizations stand to what they have done and could be held accountable. Innovative start-ups, little-known companies, or individual coders could receive validated endorsements and approvals as known, upright contributors, or making public disclosures under international enforceable penalties of perjury could help them to narrow the reputation gap to established

companies or brands faster. If software would come from open source, then the corresponding project name is the vendor receiving its reputations from its contributors.

Many professionals like medical doctors, lawyers, financial advisers, or journalists have written or unwritten rules for members in the same job category. These rules were made for the protection of the public and for guaranteeing a minimum level of quality control in their contributions. Software developers and vendors can create work products that can create intentional damage. It is overdue to mark and tag legitimate contributions of software developers or vendors accordingly.

7.2 Residual Risks

There is no guarantee that some file-based or fileless malware could not slip through the cracks. In the worst-case, users might have activated malicious ransomware or spyware on their system. However, it is assumed and hypothesized that other solutions outside EWD, but operating similar independent watchdog-type applications, can be utilized to provide redundancy for additionally required data protection.

People and organizations could still be victims of targeted attacks of criminals or clandestine services. However, the advantage of the EWD architecture is that it creates evidence (e.g., scripts in the special folder on the Secure Drive) that could not be deleted by the malware. Additionally, a script/app with few deployed copies would already create the attention that clandestine operations probably like to avoid. The deployment of targeted scripts appearing only once are anomalies and would therefore be useless. The same auto-revealing property applies to dynamic digital ghosts trying to stay out of detection.

However, EWD is initially useless when legitimate applications with online features have hidden backdoors or are operating covertly as spyware and the software manufacturer knows it but is quiet about these malicious or questionable features. The risks of detection and their consequences should deter these kinds of business practices. But still, other potentially redundant security tools are required to prevent these kinds of situations or vulnerabilities are getting harmful to users.

7.3 Is EWD's Protection Reliable Enough Against ASI?

ASI is considered the worst imaginable adversary. Therefore, it is conceivable that ASI can get its code on Safe Drives that are protected by EWD. ASI might corrupt development tools or program libraries with compromised features. However, ASI would not be able to extend its possible footprint on the Safe Drive via modifying its compromised code. Once the compromised software is detected, only the questionable code would require an update – the scope of the attack cannot be covered up. Additional redundant features should be provided via supplemented watchdogs that are designed to support auto-fixing of possible damage done by ASI. The required storage resources for this redundancy could easily be supplied by hardware progress and via amended file-formats that actively support recovery from covert data modifications.

However, proactive hashcoding of data received via DMA, before stored on RAM, would detect any modification in the executables. This Protected DMA could additionally set an executable flag in RAM pages and thereby ensure that no suspicious executable code could be provided to the CPU that was not being hashcoded and validated by EWD. Similar protection can be applied to the Local Hashcode Repository (LHCR) which is hashcoded after every modifying transaction

while hashcodes or digital signatures are being stored or managed in or by EWD outside the reach of an ASI. Finally, in case ASI is succeeding in bypassing some security measures, immutable log-files or evidence would be generated that will help us to detect even attempts to practically probe or test the security of EWD.

8 Conclusion

The Executable Watchdog is a solution to reduce and mitigate problems with malware. EWD together with the Local and Server-sided Hashcode Repository (LHCR/SHCR) can create additional product safety and accountability for software vendors legally and reputationally. Small modifications to malware, often bypass Antivirus software protections, cannot circumvent EWD. Additionally, EWD is keeping all software current by taking care of the software installation and update process. Although EWD cannot protect users directly from the consequences of malware, it is giving malware less chance to survive detection and being activated by the CPU. If current Antivirus solutions are replaced with EWD, it seems that there is no feature in which EWD would in comparison underperform. Instead, EWD is proactive and is inherently trusting software manufacturers concerned about their product liability and commercial reputation.

The SHCR can enrich executables with additional, extendable data anytime. The received and locally cached data could then be used by the EWD or other products/features to compare them with locally extracted data to automatically decide if the software shows anomalous behavior outside known or tested settings. The EWD solution could even provide users with information on possible but currently undecidable threat situations while keeping them up-to-date with notifications received within the regular query for relevant software updates. EWD is leaving it to users to decide if they want to accept a possible risk of dealing with software/malware or if they want to wait for additional crowd confirmations or expert/vendor recommendations and then be immediately notified or auto-revised to the recommendation. EWD can reduce the number and severity of damages from malware significantly; it could make a device safe and secure by default.

References

[1] Broadcom, 2019, "Difference between viruses, worms, an//knowledge.broadcom.com/external/article/178186/difference-between-viruses-worms-and-tro.html last accessed: 14/05/2021

[2] Dylan Barker: Malware Analysis Techniques, Tricks for the triage of adversarial software, Packt Publishing (2021)

[3] A. Matrosov, E. Rodionov, S. Bratus: "Rootkits and Bootkits: Reversing Modern Malware and Next Generation Threats", No Starch Press (2019)

[4] VirusTotal, “How it works”, https://support.virustotal.com/hc/en-us/articles/115002126889-How-it-works Last Visited: 14/05/2021

[5] Crowdstrike, 25 Feb 2021: "Fileless Malware https://www.crowdstrike.com/cybersecurity-101/malware/fileless-malware/ Last Visited: 14/05/2021

[6] WebFx, 13 Mar 2020: "What is the real cost of computer viruses? [Infhttps://www.webfx.com/blog/internet/cost-of-computer-viruses-infographic/ last accessed: 14/05/2021

[7] Steve Morgan, 13 Nov 2020: "Cybercrime to cost the world $10.5 trillion" https://cybersecurityventures.com/hackerpocalypse-cybercrime-report-2016/ last accessed: 14/05/2021
[8] A. Dang, A. Gazet, E. Bachaalany: "Practical Reverse Engineering x86, x64, ARM, Windows Kernel, Reversing Tools, and Obfuscation" (2014), John Wiley & Sons, Inc.
[9] Waratek "Runtime Application Self Protection (RASP) Evaluation Criteria" https://ten-inc.com/documents/RASP_Evaluation_Criteria.pdf last accessed: 14/05/2021
[10] A. Paduraru, M. Paduraru, A. Stefanescu: "Optimizing decision making in concolic execution using reinforcement learning", 2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), March 2020, DOI: 10.1109/ICSTW50294.2020.00025
[11] N. Bostrom, "Superintelligence: Paths, Dangers, Strategies" Oxford UP, (2014)
[12] Wikipedia, "Comparison of antivirus software", https://en.wikipedia.org/wiki/Comparison_of_antivirus_software, last accessed: 14/01/2021
[13] Niels Ferguson, Bruce Schneier, Tadayoshi Kohno, "Cryptography Engineering Design Principles and Practical Applications" Wiley Publishing, Inc. (2010)
[14] Ross J. Anderson, Security Engineering: A Guide to Building Dependable Distributed Systems, 3rd Ed., Wiley (2020)
[15] I.J. Good, "Speculations concerning the first ultraintelligent machine", Advances in Computers, Vol. 6, 1966, Pages 31-88, doi:10.1016/S0065-2458(08)60418-0
[16] lesswrong.com, "Intelligence https://www.lesswrong.com/tag/intelligence-explosion, Last Visited 14/05/2021
[17] E. Wittkotter, "Trustworthy encryption and communication in an IT ecosystem with artificial superintelligence", in Proceedings of 5th Workshop on Attacks and Solutions in Hardware Security (ASHES '21), doi.org/10.1145/3474376.3487279.
[18] Wikipedia, "Trusted system" https://en.wikipedia.org/wiki/Trusted_system, Last Visited 11/20/2021
[19] DARPA, Dustin Franz, "Cyber Grand Challenge (CGC)", 2016, https://www.darpa.mil/program/cyber-grand-challenge
[20] Wikipedia,"Hooksafe",https://en.wikipedia.org/wiki/Hooksafe, Last Visited 11/24/2021
[21] Wang, Z., Jiang, X., Cui, W., & Ning, P. "Countering kernel rootkits with lightweight hook protection". 2009, Proceedings of the 16th ACM Conference on Computer and Communications Security - CCS '09. doi:10.1145/1653662.1653728
[22] Dima Novikov, Roman V. Yampolskiy, Leon Reznik. Artificial Intelligence Approaches for Intrusion Detection. IEEE Long Island Systems Applications and Technology Conference (LISAT2006), pp 1-8. Farmingdale, New York. May 5, 2006.
[23] Dima Novikov, Roman V. Yampolskiy, Leon Reznik. Anomaly Detection Based Intrusion Detection. IEEE Third International Conference on Information Technology: New Generations (ITNG2006), Internet and Wireless Security Session, pp. 420-425. Las Vegas, Nevada, USA, April 10-12, 2006.
[24] Roman V Yampolskiy, 2017, "What are the ultimate limits to computational techniques: verifier theory and unverifiability", Phys. Scr. 92 093001 (8pp), https://doi.org/10.1088/1402-4896/aa7ca8
[25] Trend-Micro, "Zero-Day Vulnerability", https://www.trendmicro.com/vinfo/us/security/definition/zero-day-vulnerability, last visited: 21.01.2022
[26] Nicole Perlroth, 2021, "This Is How They Tell Me the World Ends: The Cyberweapons Arms Race", Bloomsbury Publishing