2028 - Hacker-AI and Cyberwar 2.0+
(Securing our Future: Proactive Resilience through Separated Security Measures)
ISBN: 9798375252568


# 13. Development of Hacker-AI Countermeasures

## Low-Level Security Separation (L2S2)

We do not have a lot of time to solve our cybersecurity problems. It is conceivable that Hacker-AI is already available in its main features, or it can be developed quickly - that is the underlying hypothesis of this book. Hacker AI and its capabilities in waging Cyberwar 2.0 is, for many countries, a threat to their national security.

The vulnerabilities are all too obvious. Our IT is extremely complex because of old systems and specialized technologies. Products were often developed without security, privacy, or unauthorized usage in mind; an example is the Internet of Things (IoT).

It seems hopeless and naïve to assume that we can make our IT safe quickly. I have argued throughout this book that we have no choice. Therefore, we need to have a strategy to make progress fast. I believe that putting the mission of "making our IT safe ASAP" has to come before any other consideration. I propose that we pursue the following three overlapping phases in parallel:

1. Developing basic Low-Level Security Separation (L2S2) as a redundant software solution that separates all security-relevant activities (with duplicated security features) from regular activities via quickly installable software security solutions.
2. Providing hardware that supports basic L2S2 software solutions is being made available as a retrofit and as an additional security component for most devices, including many legacy devices.
3. Establishing technology that provides L2S2 support and security by default for all new products.

In short, Phase 1 can use low-level system programming and Hypervisors (super-supervisors) inserted below every operating system so that L2S2 redo and, if necessary, overwrite all security operations done by the OS. The additionally used hashcoding solution will guarantee that only known (safe) executables/apps are loaded and accepted in RAM; blacklisted and unknown codes are immediately rejected. This feature alone will make a huge difference. Additional watchdog features within an enhanced L2S2+ can follow later.

Phase 2 will develop small, independent hardware components, enabling L2S2/L2S2+ within the databus to all storage devices and networking components. In this phase, we should try to have also USB security hardware, like a

security stick, that could check the integrity of the device independently. With this security stick, we could ensure that Phase 1 software is in solid control of the devices, which is later important for smartphones. The developed (watchdog-type) hardware will operate like uncircumventable or non-bypassable bridges between storage/networking components, the main CPU, and its RAM. This hardware will be the circuit breaker we need to control our hardware.

Phase 3 will include Phase 2 watchdog hardware within storage devices and the networking components by default.

The primary goal should be to have Phase 2 retrofit or Phase 3 security by-default technology in as many devices and as fast as possible because only these technologies could help us to be protected against advanced malware from Hacker-AI.

But there are two major obstacles to this goal: (a) it takes too much time to have this hardware developed, tested, manufactured, distributed, and deployed, and (b) hardware retrofits are only possible in servers, desktops, or laptops and not in portable smartphones and many other (smart) devices.

Therefore, Phase 1 goal must be to provide a simple software solution that can be implemented in all multi-tasking systems as soon as possible. This solution would be integrated by the OS providers and distributed as a security fix for all currently updatable/supported OS versions. This also includes all servers, desktops, laptops, and smartphones.

This simple security solution has five tasks/features: (1) local data inventory, (2) protected apps loading, (3) 3rd party software installation/updates, (4) safe solution updates for L2S2, and (5) L2S2 integrity validation.

Now, here are some more details on that list:

### (1) Creating a Local Data Inventory:

The new security software creates and caches immediately all hashcodes of all locally available and installed executables. It checks these hashcodes remotely via a new hashcode validation service if any of these hashcodes are blacklisted. Otherwise, they are being managed in a protected local cache as graylisted. Once all executables are hashcoded and checked, the solution would not allow blacklisted apps or software with an unknown hashcode to be loaded into RAM.

Although the solution generates tens of thousands of hashcodes for potentially hundreds of millions of machines, uploading these data to the servers happens entirely in the background (over days). Users are not bothered; these tasks are done with several optimizations to reduce the amount of data that is being uploaded and validated. This service is multi-tiered for quick up- and downscaling. Additionally, zero-listing is supported (i.e., waiting for better statistics); this status type prevents any delay in using the new security system. It won't have any long-term impact on the system's overall security.

### (2) Protected Loading:

Once the hashcode inventory is locally created/established, only zero- or gray-listed (and whitelisted) software is loaded into RAM. Blacklisted software is rejected. Any new unknown hashcodes must be validated via a remote hashcode validation server to determine if the software is accepted as gray- or whitelisted or if it is on a blacklist. Unknown hashcodes are managed on a yellow-list (implying some associated risk). This yellow-list has replaced the initial zero-list when the entire system started, and insufficient data were unavailable. Validation server's decisions of rejecting or accepting hashcodes are based on whitelisting or statistics and rules heuristically. As a precaution, hashcodes from the yellow-list are rejected by default.

The status of the hashcodes is not static. Grey-listed hashcodes could be white or blacklisted. Even unfair or accidental blacklisting could be reversed. Later, additional data from feature disclosures within the registration for whitelisting are managed locally and updated regularly.

### (3) Support for Safe Updates/Installation of 3rd Party Software:

Outdated (regular) software is a serious security risk. Hackers don't care if the software is not being used anymore. Currently, we often need to wait until users download and install rarely-used software updates. It is much better if the software is not manually updated by users but kept updated automatically. L2S2 will set strong incentives to have software manufacturers use L2S2 update services or inform users about problems with their outdated apps that L2S2 auto-update does not support.

Device's L2S2 instance will regularly request servers for additional data and updates it has on installed software. Information about the security of systems could be highly dynamic. Even whitelisted software must be updated if security flaws demand it or are being flagged and blacklisted.

### (4) Safe Updates for L2S2 Implementations:

Even after extensive testing, all software solutions are never perfect or completely safe. Therefore, L2S2 must be updatable without exposing itself to being hijacked by advanced attackers. Currently, encryption and digital signatures based on public-private cryptography are used to facilitate security around the integrity of updated solutions. Unfortunately, that is the best we can do without having dedicated hardware with protected, hardware-based crypto-units and keys-safes.

As soon we can use hardware-based encryption/decryption using keys that never appear outside protected hardware in cleartext, we have much better security around L2S2. Currently, it is difficult to say how weak, i.e., vulnerable to attacks, a software-only L2S2 solution could be. The only way to find out is to have external tools to check if the L2S2 implementation is modified.

**(5) Independent L2S2 Integrity Checks:**

Software-based L2S2 solutions are much more vulnerable than solutions using separate hardware in which only standard algorithms and encrypted keys are being processed. For software-only security solutions, we need interfaces that reliably validate the integrity of the L2S2 solution without making too many assumptions about the device.

**Additional Remarks:**

L2S2 solution supports but does not require software registration and white-listing of hashcodes.

Additionally, L2S2 hardware will need independent, protected hardware-based encryption and key safes. However, the first version of this technology may not have all the features of auto-detecting misused keys, multiple equivalent secret keys, or multi-unit security.

Creating a sound global ecosystem with support systems for trustworthy encryption that protects keys reliably in all conceivable aspects will take some time.

Creating a protected crypto-system 2.0 using experiences from an initial 1.0 version would be prudent. After some time, we should use all available experiences from problems and deal with more advanced challenges from potentially autonomous artificial intelligence or artificial superintelligence. L2S2 and hardware-based trustworthy encryption must be extendable from the beginning.

## Expert Development Community

Cybersecurity is big business. It is generating a lot of recurring revenue and has created many jobs. Cybersecurity claims to provide solutions for all computer-related security problems, but its performance is disappointing, inadequate, even poor. Cybersecurity is failing, but their paradigms and ways of approaching security threats are still dominant. Let me give you three examples. Cybersecurity depends solely on the main CPU/OS, although we already know these foundational elements are the main reasons for our vulnerability. Cybersecurity doesn't trust developers because they are seen as the source of vulnerabilities. And third, it uses commercial encryption without sufficiently acknowledging that keys can be stolen and misused by malware.

Our economic system is agile and innovative enough to develop sound technologies. But it is also entrenched and slow when existing business interests must adapt to new realities. It seems (at least on the surface) that solving many cybersecurity problems is not in the business sector's interest - solving problems will take away future business opportunities and endanger recurring revenue models.

I believe that security and safety are human rights for which individuals or organizations should not pay extra. I am not paying more for flying in a safe airplane.

Some may say security guards are being paid to provide additional security because police can't be everywhere. Also, there are different methods of storing money, and depending on the investments in security, the money is safer than if less is being spent. That is all true; however, it is not the right comparison to justify business models in data security.

The possible danger from Hacker-AI and later from autonomous artificial intelligence or a hypothetical artificial superintelligence does not give us a choice. Computer and data security are much more important than economical or political interests. We will (soon) enter a world in which every vulnerability will be found and exploited. Security doesn't exist if it can be bypassed easily. It's not enough if we react to security breaches. What we need is what was called in a previous chapter, low-level "security overkill" that can't be bypassed or claimed without (actually) having it.

Currently, consumers are accepting the trade-off between security and money. If you want more data or computer security, more privacy, and less worry about spyware or ransomware, you need to buy more expensive security products or have paid subscription services. However, I acknowledge that many data security features are already included in products and provided for free. User identification and authentication are free, also controlling file access via file ownership and access control lists (ACLs) are free to use. Even basic anti-virus and firewall solutions are free. However, the general perception is that if you want more security, buy a commercial security product.

It is reasonable to assume that the solution for a technical problem comes from experts in the field. I assume that system programming, knowledge of details of the operating systems like Linux, or people deeply involved with hypervisor technologies are more than any other group of software developers qualified to lead the afford in developing countermeasures for Hacker-AI.

The entire software development effort should be done as open source. This should also include development done for hardware components. The advantage of open source is that other experts could check and contribute improvements or test tools.

In the development of technology, other technical tools are used. It is important not only to focus on new tools but also on simplifying or hardening support tools. We must be sure that the tools we use are not compromised. It is assumed that an open-source community will get significant support in its effort to create a development environment that is potentially less efficient and capable but much safer than right now.

Another important contribution of the expert community is the education on applying the developed tools, new cybersecurity paradigms, or concepts in new or existing products. It will e impossible to have an open-source community do the entire work of applying the concepts to all the different possible applications. Instead, it is much better to focus on the core features and prepare them to be standardized.

The author intends to start an open-source expert/developer community under the name: **NoGo-*** (pronounced: nogostar). More info can be found at: **www.nogostar.com**.

However, the development, production, distribution, and deployment of counterdefenses will take time. Unfortunately, we should assume that activities leading to countermeasures could be sabotaged or even made impossible. In the worst case, unprotected developers or manufacturers could be harassed or actively attacked by malware or other cyberweapons generated by Hacker-AI.

## Threat-Levels

The status of Hacker-AI and the abilities of adversaries trying to stop the development of countermeasures is unknown. The effort will likely fail if adversaries actively fight against the development, production, distribution, and deployment of countermeasures. In that case, we would be too late.

Predicting future events or capabilities is impossible. Still, we can proactively categorize Hacker-AI-related scenarios into threat levels (TL). Depending on the threat level, we must prepare different protective measures.

### Threat-Level 0 (TL-0):

In TL-0, we assume that there is no advanced threat from an already existing Hacker-AI that could sabotage the development, production, distribution, or deployment. We would use existing tools and create the technology as if Hacker-AI and adversaries opposing this development would not exist.

### Threat-Level 1 (TL-1):

TL-1 assumes that there is a slight chance of an advanced Hacker-AI. It might already be developed or possibly deployed to sabotage the development up to the deployment of countermeasures. Even if most experts agree that we are still in TL-0, it is probably a matter of professional prudence to assume that adversarial Hacker-AI will interfere with developing reliable countermeasures.

### Threat-Level 2 (TL-2):

TL2 is announced or declared internally (not publicly), i.e., among anyone involved with the development, production, distribution, or deployment of Hacker-AI countermeasures. As soon as there is sufficient evidence that an advanced malware has modified the OS to remain hidden, TL-2 is declared. Modifying OS to remain hidden is what Cyber Ghosts would do. Without hardware-based low-level separate security (L2S2), it is very difficult to determine the scope of this attack.

With TL-2, I propose that all steps to final deployment must be done repeatedly on (multiple) different, isolated, non-networkable systems with older software versions from immutable storage media (e.g., old write-once CDs). Checking the same results repeatedly and redundantly will cost additional time but must be done to avoid undetected infections from advanced malware.

More simplified hardware systems must be developed to ensure harddrives in isolated systems are completely overwritten and uncompromised. Other simplified tools must be developed and used to ensure that the BIOS/UEFI or other microcontrollers with persistent memory is not compromised. Additionally, it would be better if the used hardware components were older.

The entire data transfer should be done with immutable data-storage media (e.g., CD-RWs read by older CD-R drives only). These transfer media are then archived so we can later check if or when additional (attack) data have entered the development zone or started to become active within the development tools.

Unfortunately, we cannot prevent early Hacker-AI interference, but we should be enabled to detect and remove them later. In TL-2, every step toward deploying countermeasures must be distrusted and analyzed many years later with more advanced and secure tools.

Hacker-AI has self-improving capabilities supporting and supported by smart operators focused on defending their position of global supremacy at (almost) all costs. In TL-2, we are certain that Hacker-AI exists; this would elevate the urgency for establishing effective countermeasures (globally) to new heights.

### Threat-Level 2-X (TL-2-X) or Emergency Level:

If key contributors, producers, or facilitators are attacked directly by malware/Hacker-AI, then TL-2 should internally be elevated to an emergency level (TL-2X). Elevated protection measures from the next level (TL-3), which would be publicly announced, are used to protect people and deliverables with more (non-electronic) resources.

### Threat-Level 3 (TL-3):

TL-3 is when Hacker-AI was (presumingly) already used within a Cyberwar 2.0 to occupy another country, or a government was replaced via a malware-using cyberwar. Public discussions about possible advanced capabilities or vulnerabilities would likely create panic among leaders, media, and helpless citizens. Cyberwar 2.0 events are assumed to create global shockwaves. They would indicate that no computer and no defense system is good enough protected to prevent Hacker-AI from being used against the civilian population, business, government, military, or additional countries.

Although fear and uncertainty could potentially lead to nuclear war, it is assumed that engineers and scientists from non-occupied countries worldwide would work tirelessly to limit Hacker-AI's scope of capabilities.

The public announcement of TL-3 would trigger a significant change in the civil defense posture. Because every network-connected device could turn hostile, this could get very personal quickly, like fear of being automatically surveilled and taken advantage of. In the first (non-technical) step, citizens would need to be trained to be extra vigilant about their surroundings and deactivate

as many electronic devices as possible. We would probably be advised to depend more on older, less capable devices (mobile or burner phones instead of smartphones) until hardware-based countermeasures are in place.

**Threat-Level 4 (TL-4) - Defeat:**

TL-4 is when Hacker-AI has effectively defeated all proponents or forces providing countermeasures against Hacker-AI. Surveillance and collaborators instructed to destroy all possible countermeasures within the development, production, distribution, deployment, and usage would prevent any chance to circumvent comprehensive surveillance.

I am not discussing criteria for "too-late" or a global TL-4 (defeat) situation. However, there is probably a tipping point where "too late" or defeat is an appropriate description.

In TL-1, we only deal with an imaginary adversary malware-generating Hacker-AI or, at most, a passive Cyber Ghost. Developers would entertain scenarios from which we don't know how realistic they are. Therefore, our assumptions may overestimate Hacker-AI's current or future capabilities, but we would act as if this Hacker-AI waits for a chance to interfere against us adversarially. Starting with TL-2, we are dealing with a real cyber-threat that we should not underestimate - so we might overestimate its capabilities.

If the USA or some other country (with a liberal system) has or uses Hacker-AI for defensive, retaliating purposes, then we should better hope that they are also using their capabilities on the side of supporting the development of comprehensive countermeasures. Governments' capabilities might be provided for digital protection around all systems involved in developing, producing, distributing, and deploying countermeasures as part of their TL-2 or TL-3 support.

## About Security Measures

In TL-1, all relevant contributors are advised to get educated on improving their cybersecurity in a meaningful but not overly aggressive way. Still, at minimum, developers should do their core development within a virtual machine disconnected from the Internet. Their web activities or communication could be done in separate virtual machines. Users could restart these machines regularly to remove potential spyware or malware from the Internet. Additionally, recommended standard tools with provided configurations are used to detect anomalies immediately or via tools automatically checking the log files. Although it is not expected that Hacker-AI generated-malware could be caught with these tools, it should be done to leave no stone unturned if something suspicious happens.

In levels TL-2 and above, we would need comprehensive security for all people involved in the product development, manufacturing, distribution, and deployment of the countermeasures. They should feel safe, i.e., free from harm

or threat of harm. Police and other security organizations must help create conditions in which all involved people are safe and protected regardless of their level or importance of contribution.

Personal protection for the directly involved and their immediate family must be monitored and safeguarded from unfounded accusations. Dangerous or intimidating acts like swatting, in which called police use SWAT teams to raid the house of innocent people, must be analyzed comprehensively. It is conceivable that some key contributors must be more vigilant and isolated so that they are not physically attacked via drones, fire, poison, etc. In TL-2, it would be best if key people's physical location could be hidden from surveillance and any data traces while they still have the means to communicate safely.

The mentioned security education should advise people working on the countermeasures that even in TL-1, their casual and thoughtless use of networks or removable storage drives should be over. They must be made aware of possible threats to their lives/safety or privacy from electronic equipment in their surroundings.

Multiple different offline tools should be developed so that people receive sufficient protection against threats, unjustified accusations, fabricated evidence, falsification, or deep-fakes using simple/reliable evidence preservation methods.

In more existential Level TL-2X or TL-3 situations, key contributors must be trained in best practice methods of going dark for weeks or months; trained but inconspicuous security teams might protect some. These measures seem overkill, but we should not take chances if humanity's future of living in freedom depends on it. We should prepare ahead for these situations while digital, identifiable footprints are avoided in the run-up.

The US Air-force has created secret test sides, like Area-51; the US cybercommand should work on similar sides to develop soft- and hardware with the help of pre-identified or recruited experts in protected and well-equipped environments.

All soft- and hardware used in countermeasures are being developed as open source. Dedicated experts can scrutinize and improve existing results and deliverables continuously. This method is more efficient than backroom code reviews or security audits for receiving certifications.

All code should be compiled independently by different developers and systems and checked/compared continuously for additional features slipped in by Hacker-AI. However, if many experts watch over simplified features and code, the probability that suspicious (hidden/malicious) features are being detected increases significantly. With the detection that Hacker-AI features have penetrated the security for code or compilation, experts would hunt down the source of these manipulations and remove them.

# Protection of Development

The purpose of the proposed methods is to continue the development of countermeasures when sabotaging these activities is a real possibility. I assume that the most significant method of attack by Hacker-AI is code modification, i.e., leaving backdoors or sleeper code in the to-be-developed/-deployed security code. The goal is, therefore,

(A) to prepare for late sabotage detection,

(B) to repair damages or consequences after detection quickly, and

(C) trying to reduce Hacker-AI's possible impact via clean-room-type soft-/hardware environments for developers.

The scope of the development effort is broad. It contains all activities that could be changed before fixed code or products are produced, distributed, or deployed via fixed, scalable, and immutable processes. Protecting these processes against covert changes is part of the development.

## (A) Preparation for Late Sabotage Detection

We better assume that sabotage cannot be detected when it happens. Malware from Hacker-AI could be on a machine when developers code or test their solutions. The problem is not the loss of privacy or secrecy when developers write their code; the code is already open source. The problem is that compiled code gets some additional (hidden) features just before, during, or after compilation.

Due to the system's complexity, developers are almost helpless in detecting possible backdoor code inserted by Hacker-AI via tools during the development.

When we also assume that Hacker-AI creates several layers of protection around its compromising features, it gets difficult to confirm the integrity of compiled code. Actually, validating compiled software is nearly impossible if someone or something smart prevents us from detecting a backdoor or sleeper code. This situation is realistic if Hacker-AI is already ubiquitous.

I usually assume that top-developers know their source code very well and would detect suspicious code changes. Inadvertently, Hacker-AI would reveal its existence if new code or features appeared within files that should not be contained within that code. Collaboration among several experts is done transparently; other developers would see every change. New/changed code is always assigned to someone responsible for it. However, this does not apply to (late-stage) feature insertions within compiled software.

Recently, supply chain security in open source was improved by several services, incl. GitHub. Unfortunately, the security is based on Public/Private Key (PPK) systems. Using PPK against Hacker-AI, from which we must assume it

can steal crypto-keys covertly, is not enough - we must demand that no crypto-key is exported or seen in cleartext or processed in a regular/shared CPU.

If we cannot check compiled code for hidden code modifications immediately and comprehensively, we can still create physically immutable data, e.g., on a CD, to be checked later by new hardware tools. These hardware tools don't need to exist when these immutable files are being created. Later, these files can serve as irrefutable evidence for detecting Hacker-AI activities; or we could confirm that there was no Hacker-AI interference. It will be challenging to develop tools we can trust and give us independent validation or confirmation.

These validation or confirmation tools must consist of simplified hardware with only required software features, i.e., code that is always/regularly being used. These tools should have no multi-tasking or -threading capabilities. Additionally, their RAM should be strictly segmented into a range with executable data only and another with data to be processed. This approach is also known from the Harvard CPU architecture. The executable code in these tools is simple, well structured (i.e., on a machine language level), and transparent to the outside so that qualified experts can do in-depth inspections anytime. Furthermore, users must be sure that no covert code changes happen in-between inspections, which is guaranteed if there is an air gap (physical separation from the network) between the device and how it receives data.

For validation, these tools could, e.g., prove the congruence of features (as defined in the source code) with features provided or defined in the compiled code. These follow-up validation and confirmation steps are potentially annoying or labor-intensive, but security must be more important than efficiency in using these tools.

## (B) Instant Repair of Damages

With the detection of covert modifications in security code, we use additionally (persistently) stored information to detect problems with the compromised tool or tools used in that attack. Once the tool is identified and fixed, we need to be able to fix the security code, i.e., recompile and distribute it to all compromised instances automatically. Also, we need to be sure that automation or distributed updates are not creating new security breaches.

This process of detecting problems, fixing, and redeploying solutions is essential for mitigating damages from attacks immediately. We need methods to flag devices that are not fixed as potentially unreliable. Security code is stored immutable (for attackers) but mutable by defender features in a physically separate security domain with multiple independent/redundant checks. We must put extra effort into developing or deploying tool features for detecting or revealing attacker code/features that the attackers could not know when they designed their attacks. Attackers must be prevented from reading and modifying low-level security code, i.e., they cannot adapt to new detection methods. With

these late, advanced changes, Hacker-AI's security around the protection of its attack method would eventually fail.

## (C) Hacker-AI Impact Reduction via "Digital-Clean-Rooms."

All security or countermeasure tools, their code, and all information related to these tools are open-source. We do not need secrecy around any component. All algorithms are isolated from the main OS and each other. The source code is simplified concerning internal complexity and features; it is not (prematurely) optimized for marginal performance gains. Every change is scrutinized for malicious intent or unnecessary features.

Still, source code is being written with tools, compiled, and distributed with other tools. Each tool the code came in touch with, including software present in RAM simultaneously, is logged via name, metadata, and its binary hashcode value. However, security-critical incidents could happen when, e.g., new security software and the generated hashcode, uniquely representing the security software, are generated simultaneously or in coordination by an attacker. Initially, we must accept that attackers could fool us. Methods of archiving/storing data about new security software, i.e., compiled security software and its hashcodes, are vulnerable to attacks despite all measures we could use to protect us.

Changes to the development, compilation, or distribution environment must be made more difficult using specially compiled Linux kernels that automatically track hashcodes of all loaded executable files. Continuous tracking of hashcodes and logging every change by storing it reliably on physically immutable storage media will preserve attack traces. These data are later analyzed via tools on simplified devices, e.g., a RISC V (an open-source CPU design) and simplified software for that system. Over time, we get increasingly cleaner digital clean rooms.

Additionally, some developers may intentionally use simplified devices for their regular work. They would separate their coding and code compilation on different devices. Transferring data between these systems could take additional time and go against the developer's propensity for efficiency, but security and code integrity have priority. These systems would have no hardware for wireless network support. Cable-based Ethernet should be physically disabled - the same applies to internal mics or cameras. Also, every unused USB connector is disabled as well.

Like hardware manufacturers, software developers (working on security) should also move their source code into digital- clean-rooms where suspicious, compromised code is easier detectable.

The expectation is that partial security/countermeasure solutions would throttle down the impact of Hacker-AI. Suppose this approach works; it could increase our confidence in the integrity and reliability of less compromised solutions on next-generation devices step-by-step. However, it is unknown if this

partial reduction of undetected impact by Hacker-AI is feasible, but it seems it is the best we can do for now.

I assume that increasing security is done by simplifying devices with no unnecessary interface. Less complex processors, smaller RAM utilization by a non-multitasking OS, and fewer features are helping us toward this goal. We may also take a closer look at some performance optimizations and remove them for simplicity within independent reviews.

## Is This Enough?

Starting the development with a TL-1 assumption is prudent. It won't have significant implications for people outside the development of countermeasures. It will give professionals a new perspective on vulnerabilities within their development, production, distribution, and deployment processes. The proposed protection measures, (A) Late Sabotage Detection, (B) Instant Damage Repair, and (C) Digital-Clean-Rooms, are then part of the development within TL-1. These measures are, by default, used at higher threat levels. However, beyond TL-1, we will have a more severe focus on device isolation and deactivation or control of unnecessary device interfaces.

The development is probably slowed down due to TL-1 security measures. But still, others develop in parallel with low or no security (i.e., TL-0); we would likely have deployable results quickly - TL-1 is just a backup, a precautionary measure. Other teams of developers are working on hardening the entire development/deployment process with soft- and hardware tools. The developed countermeasure solutions are independently validated as soon as more secure developer environments are available.

Detecting malware within the development process or later within the deployment is not a reason to assume we already have a TL-2 situation. It should require evidence or a credible whistleblower to call out this level. We need to detect malware with ghost-like features, which seems unlikely.

Currently, zero-day vulnerabilities (0-Days) are very expensive as they are found by hackers manually. Using 0-Days or having (expert-level) defenders know about them makes 0-Days quickly useless or worthless. Suppose we would see many more attacks with different 0-Days or reverse code engineering in combination with code-modifying attacks on the development of any countermeasure component. In that case, we should start worrying about TL-2. However, only experts seeing the numbers and their evidence in detail should be allowed to call for an internal elevation to TL-2.

I assume that we could find less-sophisticated technical measures within TL-1 and TL-2 that are sufficient to protect the first countermeasure deliverables; however, this might be a longer, iterative, and potentially competitive process in which we need to compare over a longer period the recorded results. Addi-

tionally, because of the heightened security warnings, developers will take security measures and processes more seriously, i.e., they will do many more code/system checks than they would otherwise.

Over time, protective solutions within developers' environments will detect attacks (eventually). They will not contribute to additional vulnerabilities in solutions if we prepare to fix the underlying issues immediately and safely. Different experts' intense scrutiny at every step will likely remove most problems at some point; this may not necessarily happen within version 1 of the countermeasures. I hypothesize that version 1 has enough redundancy to facilitate protection against covert change and limit damages. With operational experiences, we can make version 2 much safer.

However, operators behind the adversarial Hacker-AI could start directly threatening or harming key people within the development. Offline tools protecting developers should then be capable of gathering this evidence reliably. With evidence, we would then announce TL-2X internally; all people involved must be informed that malicious and personal attacks have happened and that a determined adversary is trying to prevent the development and completion of countermeasures. How people are protected is beyond this book's scope, but professional advice and support are likely warranted. Operational plans to protect people and product development at TL-3 (i.e., confirmed Cyberwar 2.0) should be developed as soon as possible, even if this is not being published.

When developers are forced to protect themselves, their families, and the physical integrity of used equipment or buildings, we must expect that the development, production, and deployment could be slowed down significantly. If this sounds like an overstatement, we should remind ourselves that anything bad could be expected in TL-2X or TL-3/4 - because these threat levels indicate war or preparation for war.

Additionally, we are dealing with many unknowns, and many iterations are required for defenders to get tools that handle Hacker-AI during development. The sooner we develop hardware-based security for our IT devices, the easier we can produce, distribute and deploy improved security.

Security is an arms race. We may solve some problems if we are too late. But if we are (really) too late, we may never catch up. We may fight against advanced malware of an adversary determined to take advantage of our vulnerabilities. In that situation, it is obvious: nothing will change the fact that we were too late.

## Protection of Manufacturing, Distribution, and Deployment

Software deployment via automated updates is not a distribution problem because delivery happens via the ubiquitous Internet. However, software-based

updates might come too late and would not eliminate irremovable malware/Hacker-AI from the system. In TL-2 or TL-3, this problem must be accepted because we were too late. However, these software-based countermeasures must still be distributed because they set the foundation for independent hardware security solutions that use the same hashcodes for their white-/gray- and blacklisting.

Hardware-based security solutions will not require high-end technology or manufacturing equipment. If we are beyond TL-1, it is assumed that they could be produced quickly within a war-effort-level utilization of different manufacturing facilities.

The biggest problem is to prevent or suppress malware-based sabotage. Unfortunately, time-consuming interruptions from malware won't happen before the equipment or systems are used in production. If critical computerized systems are isolated, potentially even from each other, we could test them and have malware activate itself prematurely.

Trained professionals prepare organizations with advice on workplace security and safety measures. Similarly, cybersecurity professionals should reveal threats from Hacker-AI and Cyberwar 2.0 in every organization involved with the countermeasures. Initially, we could have a lot of ineffective improvising due to a lack of guidance and misunderstanding of how Hacker-AI is spreading its malware. But the full mobilization of people trying to fix problems from different sides could show some (surprise) breakthroughs over time.

An (open-source) expert/development community, as suggested by NoGo-* (nogostar.com), could educate people dealing with software and network dependency that contributes to vulnerabilities critical within the development, production, distribution, and deployment of countermeasures. A dynamic exchange between people at the forefront and experts knowing about possible system vulnerabilities could provide improved solutions that isolate or fix processes within production, distribution, and deployment of the security hardware from targeted attacks.

In TL-1, many professionals will not take the threat of Hacker-AI interference seriously enough. Even if there are signs of Hacker-AI interference, most people within the production, distribution, and deployment chain would likely wait for TL-2X or TL-3 events until they actively participate in advanced security measures. Then they might be ready to accept the inconvenience and pain of isolating equipment from the network. Unfortunately, that might be too late because their software might be compromised with difficult-to-detect malware that interferes with reliable tools/hardware delivery.

The struggle to deliver sufficiently good countermeasure tools could go on over many years, in which countries, businesses, and peoples are potentially at risk of being attacked or damaged by Cyberwar 2.0 or Cybercrime 2.0 tools or events.

The reason for many problems with the development, production, distribution, and deployment of countermeasures (and likely events with Cybercrime 2.0) was that measures to protect systems/devices started too late.