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

3. Hacker-AI, Cyber Ghosts, and Cyber Devils

Hacking is usually an illegal activity. Hackers try to gain illegal and prohibited access to systems, steal sensitive information or cause harm to users or system owners. But hacking can also be seen as a method of exploring how systems work. Laws protect computer systems and their information; it makes unauthorized hacking punishable by fines or even imprisonment.

Still, hackers, i.e., individuals with exceptional technical skills, are developing technologies used in malware, spyware, ransomware, and trojans for fun, profit, or protest. And yes, there are steep differences in hacking.

There is ethical permission-based hacking to test the system's security and identify vulnerabilities, also called "white-hat" hacking. "Gray hat hacking" is done without permission but with the intent to identify and disclose vulnerabilities without exploiting them for personal gain. Finally, "black hat hacking", on the other hand, is malicious hacking to cause harm or steal sensitive information. They use all the same techniques; permission and intention are the only differences.

Hackers and organizations employing advanced automated hacking tools could develop software that uses AI methodologies (machine, reinforcement learning, etc.) to hack other software. I call this Hacker-AI.

I have chosen the term Hacker-AI to emphasize the role of AI in hacking. Hacker-AI is AI-assisted hacking; other terms are AI-power or AI-driven hacking. I don't want to argue that it will necessarily be something different or better. I have chosen the term Hacker-AI to emphasize the role of AI in hacking. It will assist humans, but AI could also have a more active or independent role than as an automated tool or resource used by the human hacker.

Hacker-AI should convey that AI can perform many of the same actions and achieve better results with greater speed, accuracy, and scale than humans. This terminology is useful when discussing the impact and implications of AI-assisted hacking when we discuss creating proactive measures to protect us against it. I hope that "Hacker-AI" is a term useful for building a new narrative.

More dangerous is Hacker-AI when it actively tries to have its malware stay hidden on a computer system, entirely invisible and undetectable. I call this software Cyber Ghosts. The most extreme software I discuss here is what I call a Cyber Devil. It is an undetectable Cyber Ghost making itself irremovable on every visited system, dominating every occupied device, and fighting off any late-coming malware or security software developed by others.

Hackers are Challenged

With cybersecurity around, hacking is difficult and is being made even more difficult. Ideally, hacking should be almost impossible; otherwise, we could have a security problem. Security is normally measured via the time it takes to get a difficult, unauthorized task done. So with tools, hacking could get easier and cheaper, but more importantly, it gets faster; that's when security gets lower or weaker.

Systems which could be automatically hacked are not safe or protected. Designers of security / protection systems know about attackers' capabilities. They have some labor- or time-intensive surprises that require humans to be actively involved in the hacking process (at least for a while). Alternatively, if attackers use expensive computational resources for an extended time, then this justifies that the computation-causing security measures are called secure.

Passwords are getting stronger, i.e., they cannot easily be guessed or taken from wordlists anymore. Two-factor authentication (2FA) requires users to provide two pieces of evidence to verify their identity. These factors require attackers to use more sophisticated hacking methods. However, regular security updates fix known vulnerabilities or threats - so attackers must improve their tools.

As technology evolves, Internet traffic is being eavesdropped on a massive scale; it is countered by the widespread and mandatory adoption of encryption in protecting sensitive information or Internet communication. The idea is that even if encrypted data are intercepted, anyone can read or modify them without the crypto keys, making it much harder to steal sensitive information - at least on the Internet. But breaking encryption is not impossible - actually, it's quite easy: crypto keys are being stolen.

This arms race between attackers and defenders was and hopefully is balanced. Defenders need time to find out about hacks, identify attack details and fix vulnerable systems on all affected devices. Attackers gather information about systems they want to attack, understand in detail how to bypass protective measures, and create, develop and test exploits so that attacks can easily be repeated. Attackers already have an important advantage: they are unnoticeable to defenders, i.e., attacks are initially not detectable. Over time, remaining undetected is the attackers' best and most important attribute.

Cyberdefenders are running detection programs that try to find anomalies or discover anything unusual. But in the absence of unusual events, defenders must conclude that there is no attack. Unfortunately, this absence of evidence doesn't mean that an assailant does not occupy the (observed or protected) system. Currently, all defenders can say is: nothing has been found (yet) - a true statement, but not good enough under today's conditions. Later in this book, I will make proposals that could give us a fighting chance to change this situation, but only if we are not too late - another condition I will discuss later.

Although hackers use many sophisticated hacker tools, software vulnerabilities are mainly found by chance, and only a few are found automatically. Hacking is extremely time and labor-intensive. This has and continues to motivate smart hackers to use automation. Moreover, it will invite using AI as a productivity accelerator sooner than later. Except for one commercial product, Mayhem. Mayhem is a fully automated test-case generator with advanced code analysis features. These and other hacking automation tools are designed to help humans with the different labor-intensive aspects of hacking.

I don't have concrete evidence about (advanced) AI utilization in hacking. But there is a world with a lot of closed-door projects. At this point, no one has told me that advanced AI techniques are used or are not used in hacking behind these closed doors, under national security secrecy, or paid for by criminal organizations. The only ones who could know wouldn't likely talk. I admit my warnings are speculative.

However, if a reader has evidence and is courageous enough to share it, please do so. The only story I can contribute is that I found suspicious software in the process- and network activity lists, made screenshots, and saved them. They were later deleted (not by me or anyone I know) on my main computer and other (selected) word files related to my work on Hacker-AI and Cyberwar as well. There was a backup for the word files but not for the screenshots. I was looking for additional traces or forensic evidence for this event, but there was absolutely nothing. So, except for this anecdote, me witnessing it, I have no evidence. I have beefed up my security and data protection. Other events around that time I want to keep to myself and write about it at another time. To clarify, I am not claiming an encounter with Hacker-AI.

Realistically, the chances of anyone extracting evidence from advanced (AI-assisted) hacking are probably slim. I assume it is getting smaller by the day as better tools for removing data traces are being developed. Data forensic experts may disagree. They see only malware that failed in removing data traces, not how attackers operate successfully.

Hackers have the same mindset as every typical developer. They have an attitude that can be summarized as: "don't make me busy - make it easy and fast" or one step further: "don't make me think", particularly about anything already being solved. Assuming AI-assisted hacking is not already available, we should better assume that it is just a matter of time until attackers create advanced "Hacker-AI" tools that accelerate the detection and understanding of hardware/software vulnerabilities in complex/unknown technical systems.

The game-changing moment is when hacking is so easy that we can simply state a concrete attack problem, and an effective solution for that problem is developed, deployed, and available almost immediately.

Currently, hacking requires an in-depth understanding of how technical solutions are implemented within multiple, often independent, layers of more generic solutions. Experienced hackers have an intuition of how a certain hacking

approach could deliver results. This experience correlates with tool knowledge and proficiency in its utilization. Hacker-AI will require (likely) different types of information humans need to provide similar decisions or actions.

Time from stating a problem to a finished deployment is decisive. If a hacking task was already solved, humans applying that known attack method could still take a few minutes. If hackers are familiar with the task and problem or platform context, it could take a few hours or potentially a few days of person-hours done by a hacker team to make necessary adaptations. If the platform or used applications are less studied, hacking could take days, weeks, or even months, not just by a single hacker but even by a full team.

The Edward Snowden revelations on NSA's hacker capabilities in 2013 and Glenn Greenwald's book "No Place to Hide", released in May 2014, revealed that the NSA has a "plumber team" of elite hackers who could do advanced hacks within days. This revelation is over 10 years old, and we must assume that they are already better than that.

Who will Develop and Use Hacker-AI

Tools (on complex tasks) are continuously simplified until they are usable by someone sufficiently trained to know what to do with the tool without learning or knowing any detail done by the tool. It can be similar to using the filesystem. Only a few developers know the complexity of what happens under the technical hood. Users see what they need to see or what they are curious to see; this is how it should be.

I can imagine a situation where attack descriptions would take a minute or two to be stated while developing and deploying the attack would take only a few seconds. This tool, Hacker-AI, would represent a new quality in cyberwarfare and warfare in general. To spell out the consequences: Hacker-AI would make all aspects of cybersecurity/-defense obsolete and useless when confronted with that tool.

We should stay realistic when we ask who could develop things like that. As a physicist, I know not every physicist could develop a nuclear bomb - even with the knowledge already published. It requires a bid more than that - but a small team of physicists could do it. So, in principle, computer scientists should be able to put the pieces together that could lead to Hacker-AI. Unfortunately, a lot more knowledge critical for this technology is already published, more source code is available as open source, and the problems that must be overcome seem to be less difficult than nuclear engineering.

Additionally, many AI problems could probably be studied in backrooms and tested covertly; much less material resources than in other comparable technical disciplines are required for progress in software. Getting to Hacker-AI is probably more of a political (or business) decision, a human resource problem, and not so much a technical or financial one.

The next chapter will show that we don't need to assume extraordinary abilities from an offensive Hacker-AI. We do not need human-equivalent artificial general intelligence (AGI) abilities. However, defending with cyberdefense tools against attacks is much more challenging because defenders start from nothing; they don't even know if we are under attack or what to expect from an attack.

Designers of (offensive) Hacker-AI should be extremely cautious. If they include too much autonomy in their software and detect severe side effects too late, this tool could create huge damage while trying to hide and evade deactivation or termination. If it self-improves, it could be the tool's decision to remove its off-switch features from its code. Software that can modify other software (via reverse code engineering) could also modify itself (potentially) in unpredictable ways.

Who could work on Hacker-AI except for academics who want to study and preserve a historical record of legacy technologies; or engineers who need to fix quickly 30- or 40-year-old software code while anyone with deep knowledge of these technologies is already retired? These groups may have the required features but not the malicious attitudes or understanding of how these features could be nefariously misused. Still, they could provide essential features as open source.

Other groups who could work on Hacker-AI may have criminal intent and not be ashamed of it. Additionally, governmental teams in intelligence services could push for results because of perceived threats to national security. For superpowers, having Hacker-AI tools quickly is potentially more important than getting them done safely. The same applies to 5-10 other nations ambitious enough to be first.

Unfortunately, many groups and individuals have the financial resources to hire top hackers, i.e., someone with strong computer science and system- or kernel-programming background. What's required is to have a technical driver (a computer scientist) who understands OS components, layered features, how compilers work, and how to use that to their advantage to get AI solutions without being a top expert in many of these topics. Reading 5-10 books on the above topics would probably suffice to be the project lead.

The technical skills of getting the artificial intelligence (AI) components are probably already widely accessible. Several ideas/concepts are available by smart inventors to be used within innovative solutions. Engineers could borrow Deepmind's "Reward is Enough" hypothesis, OpenAI's GPT (Generative Pretrained Transformer) approach, or Google's Pathway for creating more complex AI solutions on individually optimizable AI sub-components.

Responsible business people and IT professionals won't develop dystopian surveillance tools. But this book looks into scenarios where Hacker-AI is part of cyberweapons developed by or for nation-states. Nations have the resources and the legal framework to enforce secrecy with criminal prosecution. These

developments do not need broad support even within the government. A group within a nation's executive could understand the attractiveness of having a quiet, instantly usable, cyber-equivalent of a first-strike capable super-weapon. Some may (innocently) argue that their country's national security obligates them to prepare for cyberwar and that this would require intellectual resources to be deployed into that weapon program. Because there is no defense from cybersecurity yet, the only tool is deterrence, they would argue. I hope that the solution in a later chapter will invalidate this argument.

Software / Application Environments

The underlying concern is that current CPUs and OS (implementations) are too complex to be trusted and considered safe. I have discussed this in the first two chapters. Additionally, a lot of software receives sysadmin privileges. Can we prevent attackers from receiving sysadmin rights or manipulating the OS? No! Why? Complexity is the enemy of security. Also, it requires only one vulnerability for an attacker to achieve his goal. Finally, the weakest point (i.e., a vulnerability) determines the strength of the security for the entire system. Under these circumstances, we should conclude that software-based security is unlikely to provide security. Unfortunately, hardware-based security does not guarantee better performance by default: it depends on how hardware is instantiated, used, and checked for anomalies.

Cybersecurity protection depends on the OS and adversary's assumed abilities. Although I have no view behind closed doors, I dare here to make some educated guesses. I am open to acknowledging that I may be too optimistic or pessimistic in my assumptions. However, the lack of open / public information should not be used as an excuse for ignoring this problem.

Most agree hacking is labor and time intensive. Hackers have created (internal) libraries of proprietary automation tools that they share among members of a hacker team. As professionals, they invested in tools to analyze extracted data (including context-related (meta-) data) so that humans could infer more easily their next steps toward their goal (e.g., stealing an encryption key or methods to elevate the user role / privilege, i.e., become super-user or even system).

For software developers, labor-intensive steps are seen as something they could optimize to make it faster or better for decisions on how to go forward. In the end, hacking has a binary success criterion: A simple test will show if they succeeded or failed. Still, understanding the reasons for failures motivates more innovations - it is a productive feedback loop. There is also gradual improvement measured with some parameters and metrics. Are we better (closer to our goals) now than before: yes or no. There are multiple methods of autochecking the results of an automated process.

An important tool for hackers is Reverse Code Engineering (RCE), i.e., the usage of decompilers, disassemblers, profilers, and debuggers. Hackers' biggest

obstacle is understanding the meaning of variables and functions. In source code, meaningful names are essential for making sense of code snippets. But there are other ways to help humans or algorithms to get to a similar level of information or understanding. However, if hacking is (fully) automated, the meaning of variables or functions is secondary and irrelevant.

Software is often post-processed via code obfuscation or RASP (runtime application of self-protection) to preserve core features, making reversed-engineered code more difficult to understand and modify. In both methods, additional code is inserted to distract or slow down attackers' understanding when manually analyzing the decompiled code. Both techniques are based on code transformations that do not change what the software is supposed to do but include unnecessary code for misdirection.

However, unnecessary code is detectable in specially prepared (hacker) sandboxes. Using human tools, it is conceivable that Hacker-AI starts with minimal assumptions on features; AI could turn the problem of removing or simplifying code into a game with much simpler code as output. This approach would follow Deepmind's hypothesis: "Reward is Enough". AI analyzes technologies without knowing the underlying CPU/OS details. As a result, it could provide methods to bypass OS's low-level access-control security or detect ways to gain sysadmin rights using resources available within that environment.

All required hacking tools have an open-source version. But these tools are relatively difficult/ inconvenient to use in their basic form. They are improved with additional automation tools that are often commercially available or shared among teams - otherwise, RCE or hacking is very labor-intensive. Extracting certain low-level data or transforming output into an easier usable, enhanced form helps attackers to understand/decide what is relevant or irrelevant to their goal. I am writing this as a self-educated practitioner, not a mentored expert. Therefore, more experienced hackers will follow a more efficient approach, or they know tons of valuable shortcuts that I don't.

Without saying that this is already done, AI could help systematically test different hypotheses for variables and functions and rank them via probabilities if they are relevant/irrelevant to a binary goal. Low probabilities could reduce the search space significantly. High-probability variables/functions are tested continuously, directly, or in combination with other values/functions whether the goal is accomplished.

The automated removal of RASP-related code, i.e., code that detects code modifications and prevents proper execution, is not being discussed publicly yet, probably for legal reasons. But RASP is (easily) detectable, and via overwriting variable values, as done by debuggers (or via special sandboxes), we could suppress consequences from RASP as part of a game played by an AI. Removing RASP for humans is difficult, time- and labor-intensive; not so for Hacker-AI.

For attackers, the search space for finding vulnerabilities is huge. But for defenders, the challenge is worse; they must find all vulnerabilities. Vulnerabilities are detected all the time but compared to large amounts of existing source code, these problems are (so far) relatively rare exceptions. That should make us suspicious.

Even if we had relentless AI persistently seeking vulnerabilities, would we conclude that we have removed all vulnerabilities? Would the AI tell us (at some point) that all vulnerabilities are removed? I doubt that we will have the conceptual tools to answer these questions. Therefore, I would side with the assumption that we will always have vulnerabilities in our code despite the use of AI in cyberdefense. And I would assume that an attacker could always find an exploitable vulnerability for his attack.

Undetectable Cyber Ghosts and Irremovable Cyber Devils

Many cyberattack methods are done poorly, i.e., they leave traces due to ignorance or human mistakes. Attackers use only a relatively small number of similar methods to attack systems; that is why we use blacklisting in cybersecurity. Also, malware is not extensively trained or optimized in concealment, camouflage, or self-defense; it is not yet very sophisticated in evading detection.

Detectability of software and leaving data traces is a core feature of every multitasking operating system. Providing full process transparency in an OS is no small feed. These features are complex; unfortunately, they can also be misused for hiding Cyber Ghosts. From studying these systems at a low level, I concluded that the number of methods to have Cyber Ghosts hidden is scarily high. Hardening the kernel/OS against this threat is theoretically possible, but I concluded that software-only protection is nearly hopeless. Protecting the system in combination with hardware is feasible, but current solutions are flawed.

I am predicting undetectable software. I am aware of the problem with this argument. You cannot prove it exists because we can only find detectable malware. Proving that this kind of software does not exist is not doable. Still, people responsible for security should be concerned about that. I am not convinced that Cyber Ghosts are permanently invisible; on the contrary, I assume we could develop reliable tools to detect and stop them consistently - but not with the tools we currently have in our toolbox. The same applies to stopping unknown exploits. These problems are all solvable, and I will explain that in a later chapter.

Modifying the OS/kernel by a Cyber Devil, i.e., an irremovable Cyber Ghost, who wants to remain the only dominant instance on a device, is feasible by malware. These irremovable Cyber Devils must stop late-coming AI to gain its unassailable status against other malware that is trying to be undetectable and irremovable – but is a late-coming challenger. This Cyber Devil would need

to control the device's boot phase, which is quite complex. From studying it, I concluded that hardening the existing approach with software-only protection is futile.

Invisibility, undetectability, and irremovability are probably the most dangerous malware features that Hacker-AI could improve on. Developing undetectable (ghost-like) software and, in a second stage, irremovable (devil-type) software should ideally be challenging; it should also be expensive and require large and diverse teams of contributors. But that might be false hope.

However, this book assumes that the effort to gain undetectability or irremovability is beyond what we could expect from non-governmental (i.e., private/criminal) initiatives. We could assume that the level of effort or the risk is too high or not worth it; non-committed organizations would likely back off before getting it done. The negative consequences or high-stake risks for people developing Cyber Ghosts or Devils could be a significant show-stopper. If early discovered, the responsible people could likely be criminally prosecuted. They could be marked as (cyber-) terrorists and even targeted by military force. This risk may not hold back all. Controlling all features of millions or even billions of hardware devices represents a value that could quickly go into the hundreds of billion or even trillions of dollars. Getting something like that represents unassailable wealth. Making this gain irremovable with advanced cryptography and low-level system/kernel technologies is much cheaper than building vaults for even a fraction of that wealth.

However, if a group of people succeeds with some early, not-quite-perfect Ghost and Devil features, they could use Hacker-AI's Cyberwar 2.0 features to take over political power to protect their wealth/control, and their initiative would turn into a governmental enterprise after that step. So, due to the gravity of these features, undetectability and irremovability are considered features associated with nation-states and not corporations or criminal organizations. These features are similar to deterrence from ABC (nuclear, biological, or chemical) weapons. Only nations with their resources, protective laws, and institutions could give peoples or entities the required protection to own or control weapons with that high level of consequences.

Creating irremovable ghost-like malware is an attractive goal for nations competing for (persistent) global supremacy. Some political leaders could consider this goal irresistible. However, deploying these Cyber Devils is likely considered an act of war. It is a high-stakes gamble: it could lead to a war, but it could also make regular military actions rather unlikely or unmanageable. With malware, the nation with these capabilities could disrupt the supplies and logistics of other nations that try to resist.

Additionally, the status of nuclear deterrence is probably unknown under these conditions. Are all essential components for a counterstrike still operational? If Cyber Ghosts or Devils are undetectable, how can anyone commanding these weapons know if they are still usable? It is possible that the adversary,

i.e., the party who operates/masters the malware knows more about the operational readiness status of the weapons than its owner. If a controller of a Cyber Ghost/Devil has reasonable confidence that he could stop a retaliatory nuclear counterstrike, he could be motivated to initiate a preemptive deployment of Cyber Devils immediately because we could not know how long he would be in that position.

The problem with complexity is that large investments could turn against defenders when the technical solutions are too complicated. An old-fashion circuit breaker against sophisticated attackers was too cheap to be considered. Just saying the level of effort to disrupt nations to wage war is probably smaller than most military planners want to admit. Basic security must be simple, or attackers outsmart defenders.

Hacker-AI Types

In its simplest version, Hacker-AI (Type-I) will use AI on servers to create malware for hacking arbitrary IT systems. I.e., it would find vulnerabilities automatically, e.g., elevating privileges/user roles (sysadmin or system), stealing encryption keys, hiding attacking tools from being detected, or creating code that is irremovable on (once) occupied systems. Type-I would create malware apps with exploits automatically. Hacker-AI (Type-I) would generate and provide software features to be downloaded by the client-sided malware. Primarily, Type-I would create spy or malware tools that are being used and commanded by humans via action decisions from server-sided attack management.

In a more advanced version, Hacker-AI (Type-II) would be actively operating malware on computer systems with sysadmin rights, trying to stay hidden from detection and making itself irremovable from any later deployed software. Software extensions are not (primarily) generated on centralized servers. Type-II would take independent actions and use server resources to get data it cannot receive via its actions directly. The different Hacker-AI (Type-II) instances could communicate and help each other, so they would not depend on instructions or additional software developed by centralized computational resources. Type-II could adapt to all relevant events in its local environment via instructions generated by other Hacker-AI (Type-II) instances for its self-defense.

Where are we right now? It is conceivable that some intelligence services, cyber-commands, or companies supplying governments with hacker tools have already developed features we expect in Hacker-AI (Type-I). The next chapter will elaborate on features I would expect from Hacker-AI.

Because we have not yet seen malware fully generated by Hacker-AI on servers, I conclude that we are Pre-Type-I. I define this stage of AI in hacking as category Type-0. Hacker-AI might be used to solve selected issues for human operators. Some results are being put in partly-generated malware, but the pro-

cess from goal articulation to the deployed solution is still labor-intensive. Human experts are required in many detailed decisions. Hacker-AI (Type-0) is not battle ready for a fast-paced cyberwar.

I want to reflect on the prospect of creating Hacker-AI (Type-II). Once considering the pros and cons, I am reasonably certain that governments would only create Hacker-AI (Type-I), even if that implies that ghost features would not deliver perfect invisibility, undetectability, and irremovability. The expected results of Type-I Hacker-AI are likely sufficient in defending systems against late-coming solutions.

However, if any software-based hacker solution has a chance to remove malware from Hacker-AI (Type-I) from a system, then only a much more agile and faster-adapting solution could change its coding on the attacked system fast. Only speedy adjustments could deliver measures designed to stop late-coming AI. In that case, Hacker-AI (Type-I) may not even see what is coming before it is too late. Hacker-AI (Type-II) could then create measures to be distributed covertly against other unprepared Type-I instances. The speed of adaptation is the decisive feature if we allow every (non-blacklisted) software to be executed.

Early on, I thought Hacker-AI (Type-II) was required to gain an unassailable status against any late-coming challenger. But with other methods stopping late-comers (like malware-controlled whitelisting of apps, i.e., only software accepted by the malware is executable), I would hypothesize that Hacker-AI (Type-I) could create an irremovable Cyber Ghost / Devil.

Stopping the rise of Hacker-AI (Type-II) is a good thing because developing Type-II might even be easier than Type-I once we have AI that could modify code intentionally, particularly its own code.

So I assume in this book that we have to deal only with Hacker-AI Type-I.

Limits to Hacker-AI and Blindspots

Computer systems are, by design, open systems. Even high-security computer systems have a von-Neumann Architecture, i.e., a unified address space in which instructions and data are stored together. All multi-user, multithreading systems share the same underlying OS concepts, even with its less-safe cousins. Optimization and redundancy within some components make them different in many details, but all systems are designed in layers to be more easily extendable or modifiable. These layers are very similar; all multithreading OS systems are (pretty) similar and, thereby, vulnerable in similar ways.

I assume that Hacker-AI can read (or receive) compiled binaries from the filesystem; this feature is probably enough to reverse code engineer the software and discover its vulnerabilities. There is the expectation that OS's access control system could prevent read access to critical files, but even "simple" read features

are too complex to be trusted; it is safer to assume that an attacker will have access to all device features if no other measures are preventing that.

If we consider limitations to Hacker-AI, they could only be set by the computer system (CPU/OS) and not inherently from limitations in the AI used by the Hacker-AI. AI is software that can be updated in the next iteration. If the probed system is vulnerable, Hacker-AI will eventually find it.

Theoretically, Hacker-AI, even Type-II, could be deceived via honey-pots, tripwires, or secret code that it doesn't know or is prevented from detailed analysis. The problem with secrecy within the defense is that we don't know what an attacker knows; secrets may already be known and analyzed by the software.

Currently, the idea of an attack is that an attacker actively and repeatedly probes for holes in the defense. A sophisticated Hacker-AI would try to be successful with the least amount of interactions with the targeted systems because it knows the system's vulnerabilities; it has tested it under much safer conditions. After determining the situation, it would start an attack only once, expected to be successful. Afterward, it would delete all traces as if nothing had happened. Attacks with massive numbers of attack events might be used to masquerade a single successful event. Detecting successful Hacker-AI intrusions (even with hindsight) is a failure of Hacker-AI. This temporary limitation or failure of that Hacker-AI version is likely prevented in the next iteration.

Some operating systems are designed to detect malware or protect private data better and more easily/reliably than others. Without first-hand knowledge, I assume that some military-grade systems are that way; they are making malware detection easier. However, to analyze the problem with undetectability, we should ask a different question: Is it possible to have an OS kernel that can near-perfectly hide certain apps and hide that it is doing that? If yes, malware could modify the OS accordingly.

From studying OS architectures, I concluded that we could modify an OS so that it hides apps and hides that it has this feature. I am not aware that super experts have published ideas to prevent that. I am not claiming that a modification to the OS could be hidden on storage devices within independent audits, but if we would deal with an advanced, ubiquitous Hacker-AI that has additionally stolen keys related to modifications of the CPU's microcode, boot guards of BIOS/UEFI or TPM/TEE, then I am open to the hypothesis that this Hacker-AI could find ways to hide from a comprehensive independent audit as well. In that case, we could have Cyber Ghosts in our existing IT ecosystem.

The main problem with Hacker-AI is that it could operate in a complex ecosystem that allows software to be modified covertly while removing all traces that it even happened - this feature is the reason for our blindspots. It could have many methods trying to hide or camouflage its existence. Software and hardware have many layers to insert code and features (covertly). Software developers have no problem using these layers to simplify and accelerate their product development; the same applies to attackers misusing them.

Soft- and hardware are designed to provide redundancy against failures without escalating small failures or inconsistencies to users or manufacturers as big issues. E.g., some storage media and network technologies are dealing with (high) error rates in their normal operations. Some of this error handling is delegated to hardware components that get their operating software from a potentially compromised operating system. Additionally, storage cells could be marked damaged but still be used to hide information from security audits. Moreover, many files or data formats have comment fields or segments that could be misused to hide information. Unfortunately, software developers and manufacturers have introduced many features without being aware that they could also be misused to hide information. Therefore, the problem and our blindspots are based on our ignorance, lack of imagination, or vigilance about which software (features) could be used against us.

We allow every developer to provide us with software without establishing sufficient accountability. Computers and their software are essential for our civilization; we must protect ourselves against developers or adversaries who try to hide in anonymity. To put it into perspective: We don't take prescriptions or legal advice from anonymous sources. Also, we don't have our money managed by unnamed institutions or organizations that don't care who they hire. But our carelessness with software is remarkable. Technology doesn't prevent us from accepting unknown, malicious code from anonymous sources.