Terrorists Will Use Teslas to Kill Us

A cyber-security war is coming for driverless cars. Here's what it will take to win.

| By ZACH AYSAN

It's a calm Saturday morning in August of next year. Suddenly, across the nation, 12,000 Tesla Model S sedans start up at the same time. They engage Tesla's vaunted autopilot feature and head out onto the road. Some of them make their way to local gas stations. Some to electrical substations. And then, as they approach, they accelerate to top speed. The explosions are fantastic as the Model S batteries rupture and spark fires, which ignite anything flammable in the area. The power grid in the Los Angeles area is brought down almost immediately. Hundreds of fires rage. America is under attack. This might sound like science fiction. It's not.

* * *

With cyber security the first thing to understand is that the internet is ungovernable because locality is irrelevant and identity is shroudable. This is by design—it's *the Internet*—we're all supposed to be able to talk to everyone and it wasn't designed at the protocol level to require payment or identification.

Cyber criminals make mistakes and are arrested occasionally, but attacks can originate from states that do not cooperate with international institutions or foreign governments. And outside of the developed world there is often even less of a distinction between private individuals and state actors: A software security expert can work to enrich herself—or a criminal enterprise—one day and then work for their government's intelligence service the next. This makes international cooperation even more difficult.

The second thing to recognize about cyber security is that attack is much easier than defense. Attackers can probe from multiple points, such as previously-hacked computers or servers rented with stolen credit card information. They can patiently try different strategies until they succeed. Talented attackers may first invent a new method of attack, then write software to scan servers or internet traffic to create a prioritized list of potentially vulnerable organizations, and only then begin systematically breaching them.

A defender, on the other hand, is a sitting target. Her application is public facing, with URLs, domains, and data centers that anyone can investigate. She has a consistent, detectable set of operating systems, software languages, and libraries that are as well understood by potential adversaries as they are by the in-house team that is responsible for managing them.

And a defender only has to make one mistake: A single entry point incorrectly secured will allow access to attackers. Defending all entry points and perpetually keeping them defended, despite changing organizational requirements, personnel, and a never ending stream of vulnerability updates to software libraries, is nigh on impossible. And even organizations that have the technical competence and resources to defend against persistent attack—such as the NSA, for example—are one insider away from critical breach or exfiltration. The attack surface is huge and the threat is persistent. If you can't arrest the attackers and the attackers have an infinite amount of time to find vulnerabilities, then the question isn't *if* the system can be breached. It's *when* the system will be breached. And what the attackers will do when they breach it.

Once a system is breached, a sufficiently prepared attacker can use pre-written software to accomplish prioritized objectives. For example, malware attacking a retail platform might exfiltrate password lists and cryptographic keys first, and only later collect information such as purchasing history. Once an attacker begins to suspect that their activities have been noticed, they can encrypt the compromised server's hard drives before ransoming them back to their owners.

Welcome to the cyberpunk future: it's stranger and less dingy than promised, but hackers have panicked organizations scrambling to pay cryptocurrency ransoms. I know. I'm the one helping organizations during crisis.

In the ’90s, as an irresponsible youth, I hacked computers illegally for fun, before starting to write security-critical commercial software for the telephone industry. My early experimentation with cryptocurrencies and work as a whitehat led to companies coming to me after being hit by ransomware. I help them understand their options and the steps they need to take in order to get their systems up and running, with mitigation strategies for next time. (If you're reading this and need help, sorry, I only take introductions through personal contacts to prevent getting targeted by cybercriminals. I also no longer personally hold cryptocurrencies for the same reasons.)

Even the most dedicated defenders have *some* vulnerabilities. The most dangerous kind are called “zero-day” exploits because they find cracks in the software foundation that no one—not even the original designers—knew existed. The most famous zero-day might be Heartbleed, a security hole in a widely used library called OpenSSL, which, after it was discovered, caused a number of organizations to panic as system administrators rushed to patch in advance of a breach. Hackers exploiting Heartbleed used it to steal—just as one example—the security keys at a major U.S. hospital system, which comprised the privacy of 4.5 million patient records.

The third thing to understand about cyber security is that certain classes of cyber attack, including most zero-day exploits, can break all instances of the same system at the same time. For example, while it would take two separate missiles to destroy two separate predator drones, a single software vulnerability could be exploited by a virus to breach and disable both missiles simultaneously. This is how the WannaCry virus was able to infect hundreds of thousand computers, including life saving machines used by hospitals in the United Kingdom.

And once attackers have control of a system, it can be very hard to wrench it back. Data moves very, very quickly. A server in Austin takes about 140ms to send data to a server in Tokyo. What this means is that if you rely on human judgment during an attack to safeguard a system by taking it off-line, you might be too late because compromised devices can disengage themselves from remote control. A hacked phone, for example, may update its network code to pass all information through a VPN controlled by an assailant. And without countermeasures enacted ahead of time by the phone manufacturer, instructions to update the phone's vulnerable software can be automatically blocked, resulting in a permanently compromised device.

To reiterate, here are our three precepts:

1) The internet is anarchy. It is difficult to attribute attacks and even when it is possible, public disclosure reveals sources and methods.

2) Cyber defense is extremely difficult, especially over time as organizations change.

3) Some classes of cyber attack allow control of all instances of a device and, with the right pre-planning, can prevent access to the device owner once breached.

* * *

Which brings us to the intersection of computers and the real world.

In 2010 a team of researchers discovered Stuxnet, a virus written in a joint collaboration between Israel and the United States to disrupt the Iranian nuclear weapons program. Though the Iranians took steps to ensure that their nuclear material processing equipment was not connected to the internet, the virus was ferried into the facility on a USB stick. Once Stuxnet took hold, it subtly altered the operation of the facility’s centrifuges so that they would slowly, and

seemingly inexplicably, destroy themselves without revealing the presence of the malware.

Stuxnet taught the cyber security world two things: First, viruses aren't just intelligence tools—they're tools of war. Second, whether or not a computer is internet-connected is more of a continuum than a binary fact. The Iranian weapons program can be pretty dark but still let in USB keys; a Twitter profile can be white hot but still go offline occasionally. And data can be exfiltrated through a multitude of methods.

With advances in machine learning, data can be analyzed and filtered locally so that low-bandwidth or high-latency methods of communication aren't the barriers they once were. For example, an adversary can employ voice-to-text on a company's internal videos and only exfiltrate those that mention keywords critical to research.

And to show you how complicated this forest of mirrors really is, consider that Kaspersky Lab, the research group which discovered Stuxnet, has recently been classified by the United States Department of Homeland Security as likely having ties to the Russian FSB, the post-Soviet replacement of the KGB. So maybe Kaspersky's ferreting out of Stuxnet wasn't just an accidental gift to the Iranians after all.

* * *

Over the past 20 years a multitude of everyday devices have become computers. Refrigerators are now computers. Watches are now computers. Even things like one-time use sensors employed to ensure correct concrete hardening are now computers.

Cars are computers, too.

And they're just about as secure as computers everywhere else.

In 2015, Jeep got hacked and had to ship millions of USB sticks to patch the automotive software. But why *just* Jeep? If hacking a Jeep is as straightforward as hacking a server, and servers are routinely breached, then where are all the hacked cars? It's a bit like the Fermi Paradox.

The explanation is likely a mix of factors. It could be that security researchers just aren't looking closely enough at cars. It could be that the blackhats aren't as motivated to attack vehicles because the ROI is more elusory. It could be that hackers have trouble adapting techniques that work on servers and personal computers for cars because the attack surface is smaller. Or the answer could be more sinister. Uber hid a 57-million user data breach by paying off the hackers. Perhaps automotive companies have quietly been doing the same.

* * *

Structural engineers limit how much deflection a beam or causeway can undergo during expected load not because the deflections themselves are necessarily unsafe, but because they expect people to reliably report when things feel wrong—and if large, but otherwise safe, deflections are routine, then large-but-unsafe deflections will go unreported. Structural engineers also use more conservative designs when systems cannot exhibit potential weakness before failure.

You should think about computers the same way.

Computers hit with sophisticated malware show no sign of infection. Even if an attack requires multiple stages or intermediary computers, as Stuxnet did, carefully-written viruses are invisible. All software, including viruses, is just code and code is just data and data doesn't change how we perceive the computer it resides in unless the software on the computer intends for us to perceive the change.

And now for the scary stuff. Remember our third precept: that some classes of cyber attack compromise all instances of a device.

We think of Teslas as cars just like we think of an iPhone as a phone, but a more accurate account of reality is that they're both just computers. One drives you around while the other sits in your pocket, but that's basically the sum of the difference. No matter how strange it may sound to a layperson, to a software developer the similarity between the two so obvious that it isn't worth even mentioning: They're both just operating systems on a piece of hardware.

Which means that something like WannaCry is just as possible for Teslas as it was for hospitals. They are both hackable, at scale.

There is another difference, of course: Your iPhone can't move on its own. But an autonomous vehicle, like a Tesla, that crashes into a chemical plant, electrical subsystem, oil line, or gas station while traveling 125 miles per hour could do a great deal of damage.

Now let's combine those two thoughts together. What would happen if someone hacked thousands of autonomous cars all at once and turned them into weapons?

Nothing good, that's what.

* * *

One of the problems I've had over the past year and a half is how to communicate this idea without either sounding like a crank or giving bad actors ideas.

After the thought first occurred to me a year and a half ago I reported it to Public Safety Canada. A year later I met with my member of Parliament to find out what efforts we were putting forward to mitigate the potential threat. What I learned

was that there were not only no regulations on autonomous vehicles, but that there were no plans to create regulations, either.

After talking with most of the major autonomous car makers, including Tesla, BMW, GM, and Toyota, I realized that decision makers in large automotive companies don't have a magic solution any more than startup software developers do.

They know they need autonomous driving technology to compete in the marketplace. They also know they're exposed. At the moment, their primary defense is the obscurity of their platform, which means that the more successful they become, the more vulnerable they'll be. Not a great position to be in.

And it turns out that most software developers haven't really thought of self driving cars and cyber security at the same time, nor do they know the interfaces that the electrical systems on automotives expose.

What this means is that we have time. These exploits aren't difficult for organizations like the NSA, but they aren't something that ISIS or North Korea is capable of easily pulling off. We're exposed, yes, but the sky isn't falling. We have time to create the right regulations and international agreements—if we can foster the political will and act.

* * *

There are a number of ways to approach autonomous vehicle security, but let's start with a frank assessment of what *won't* work:

1) Relying on off-the-shelf anti-virus software. The only anti-virus that should be trusted is the one that comes with the operating system.

2) Trying to air-gap autonomous devices. Between aerial Bluetooth viruses traveling across smart light bulbs, debugging devices at your local automotive

repair shop, or just plain old mistakes (like vulnerable software defined radios) air gaps can't be ensured. The Iranian nuclear facilities were air-gapped and that didn’t stop Stuxnet and the CIA, so we shouldn't count on it stopping the DPRK.

3) Code review. When Western governments build their security systems, they often rely on pieces of hardware built in China. The “security” of these components is certified by inspecting the code on a handful of samples.

But while the British government may code review Chinese network gear in Banbury, Oxfordshire, if it ever came to total war with the Chinese they would likely have to replace all of it with American stuff. Because unless you inspect each individual component you get from a supplier, and it all has physically unmodifiable code, you have no idea what code is currently running your system.

The chief deterrent against a Chinese supplier such as Huawei slipping malware into its products is loss of trade and reputation—commercial considerations that are moot when it comes to state strategic priorities in war.

4) Trusting automotive companies. Equifax and Ashley Madison were secure. Until they weren't. National security isn't something to entrust to corporations and certainly not to corporations from countries with a poor history on cyber security, such as China. Capitalism rewards invention and risk, not long-tail risk mitigation.

5) Certifying individual components or vehicles. Detailed, prescriptive regulation and individual certification is too slow to keep pace with the fast-changing nature of modern software development. Our most secure corporations update their code multiple times per day. This isn't just a correlative artifact of well-run tech companies—it's causal. The first actor to find a vulnerability is usually the organization responsible for the service or device and they get the fix out as fast as possible.

Instead, regulations should be functional. For instance, a maxim such as "data should never be readable by an intermediary network device" or "no action taken by the media computer should change state in the control computer" so fines and security bounties aren't arbitrary, but automotive companies can still compete on the speed of their technology advancement.

6) Allowing the market to price wide-scale cyber attack as part of existing automotive insurance. With all due respect, insurance companies have neither the balance sheet, nor the expertise, to effectively estimate these risks. The Poisson Distribution is wonderful, but computer viruses invalidate all classes of the same system at the same time—so it shouldn't be used as a basis for pricing or predicting attacks. Without statistical independence, vulnerabilities of this scale cannot be accurately priced because it is impossible to get precise probabilities of the likelihood of a black swan event. Civil engineers design for 1-in-100-year storms. What does a 1-in-100-year cyber attack even look like? No one knows.

7) Waiting for autonomous vehicles to be used in small-scale attacks before crafting legislation. If we wait for such an occurrence then the resulting legislation is likely to be geared toward small-scale attacks and not focused on the bigger risk. Our first concern should be focusing on our national security (large-scale hacks of entire fleets), not securing soft targets (individual car hacks).

* * *

So what would work? Effective policies should all start with a recognition that governments aren't going to be able to intelligently regulate the issue. A well-funded, open-source effort with clear recommendations will be the most efficacious way of securing the driverless vehicle.

1) Software professionals should educate and encourage electrical and mechanical engineers to submit proposals that will help autonomous vehicle

companies and governments protect the public.

2) The OSINT and arms control community should help with drafts of international agreements to make the cyber targeting of civilian systems during wartime illegal under international law—and we need legal minds to craft sample regulations that less-technical countries can use as a baseline.

3) Our trade agreements should reflect the changing nature of our interdependence. China recently announced that foreign automotive companies such as Google's Waymo cannot photograph every square inch of their roads due to concerns over national security.

But autonomous vehicles require both cameras and an internet connection to operate, so this regulation will have the effect of keeping foreign-made autonomous vehicles off of Chinese roads.

The Chinese either understand the threat that autonomous vehicles pose and want to limit their exposure—or they're using national security concerns to mask an attempt to incubate their own autonomous vehicle industry.

Either way, the Chinese clearly understand something that is lost on many Westerners: Liberalized trade is great, but national security is more important. Without international cooperation on autonomous vehicle regulation and symmetric trade agreements with harsh violation provisions, we should not allow non-friendly states access to our autonomous vehicle markets. (Nor should we allow components from these countries.) The Chinese get this. America should, too.

4) Any permanent, non-military device that can fly, drive, walk, rocket, or swim autonomously should allow for a standardized safety module. The power of the propulsion mechanism, as well as the computers and sensors that command the autonomous device, should connect through this safety module. And if this isn't

possible due to the nature of the propulsion system (e.g., devices with chemical rockets), then an emergency disabling system should be present instead.

The device must not be powerable or operable unless the safety module is present and the device should not be able to access its own safety module in any way. (Military devices shouldn't be subject to these regulations.)

Countries should be able to specify what safety modules are acceptable within their domain, and you would expect that most major countries will develop their own module or only trust devices with safety modules from their closest allies. But devices could be designed to accept multiple modules, any of which can initiate the safety procedures. That way autonomous movement does not need to cease when crossing a border.

(Though care would need to be taken to ensure that the devices were truly independent. Any safety module should be able to shut down the device or take it offline, even if other modules have commands to act maliciously. Security should be additive, not multiplicative.)

The safety modules should be able to communicate through multiple channels; including satellite, radio, LTE, and even pulsating light via onboard camera. This way, by utilizing cryptographic keys and certificates, governments could order autonomous devices (through the safety module) emergency commands, like "Shutdown in 5 seconds" or "Cease software updates until further instruction." The safety module itself must be able to disengage both the power and the controlling computer in emergencies.

And finally, in order to ensure the integrity of the module itself, no matter what code is present on the autonomous device's computer, the device should not be able to interfere with its safety module or the safety module of any other autonomous devices.

(The most straightforward way of securing the safety module would be to employ traditional computer architecture with a one-way connection to the main computer and a fallback to an unchangeable ASIC with its own cryptographic key pair and a direct connection to the power.)

5) Standardized, redundant systems are another safeguard. If power is removed from the primary computer of an autonomous device, then the vehicle should still be able to land or park safely. Hard shutdowns should always be available, but it shouldn't be the first resort during a cyber attack.

6) In order for an autonomous vehicle to go faster than a preset speed limit it should request permission to do so from the module. That way, you could get to the hospital quickly during an emergency, but governments could limit how many speeding vehicles are allowed at one time. Why is this important? Because a car moving half the speed has one quarter of the kinetic energy—and is less likely to have a battery explosion on contact.

7) Isolate the control computer. Most automotive electronic control units communicate using the Controller Area Network (CAN bus)—an unauthenticated, unencrypted multi-master bus. Around the world software developers reading the previous sentence just spit out their coffee. *Do not allow control computers to read data directly from the CAN bus.*

And do not connect the vehicle’s media computer to the control computer.

If we must get state from the CAN bus then it should be through an intermediary module that converts signals to one of a finite series of states / enums. (And while we’re at it, we should create an international agreement to sunset the CAN bus and replace it with something more secure.)

8) Take a note from Apple and use a Secure Enclave dedicated to security-critical tasks, such as updating the control computer's software. As with the safety

module, the security enclave should have fallback communication methods to safely disable the car during a critical vulnerability. Ideally, the security enclave should be designed with multiple sets of chips from different manufacturers in order to mitigate industrial espionage, or vulnerabilities such as Intel's Meltdown.

9) Do not trust the network. Do not trust DNS or certificate chains. Employ IP pinning and certificate pinning with fallback strategies. Do not rely solely on HTTPS. Protocols and ciphers aren't perfect and protocol downgrade attacks are too easy. Use client-side encryption in addition to HTTPS and use *really big* keys.

Ship every car with its own, massive One Time Pad (OTP). And create the OTP with multiple secure random sources on computers never connected to the Internet in the secure, guarded facility used for code signing. It should not be physically possible to read the same bit twice from the OTP. Final code review should be on computers never connected to the internet. And never, ever, allow SSH access to any autonomous vehicles—even those under development.

10) Employ code and data signing and encryption on everything that is possible, including data in volatile memory. All updates to the control computer should be encrypted, signed, and double checksummed. (The checksum should be shared with governments and broadcast to the safety module.) If the control computer cannot verify the software update's signature or checksum with the safety module, the control computer should shut the vehicle down safely.

There are other ways for governments to nibble at the periphery of the problem: Governments ought to band together to create a system of bug-bounties, in order to incentivize security researchers. (Bounties for real remote control should range between $100 to $10,000 per device, depending on factors like max speed attainable.) Governments should also agree to impose harsh fines and prison sentences for the manufacture, sale, or provision of counterfeit automotive electronics

And in America, we ought to both dramatically increase funding for cyber warfare units and find a way to expand cyber reserves to engage computer experts in the private sector. For those who couldn't possibly get a security clearance, direct them to open source initiatives and think tanks.

Finally, America should increase funding for research into chips that are specialized for security, not performance (so that vulnerabilities like Meltdown and Spectre are less likely), and create regulations that encourage safer programming languages, such as Rust, over those that have unsafe or undefined behaviour.

There is a silver lining to all the work we have to do: The nature of the autonomous vehicle threat may finally bring about the political will, economic incentives, and ideas we need to secure our real-world computer systems. And with a little luck, we could wake up decades from now and talk with amazement about cyber events of the early 2000s like we do of the chemical fires on the rivers in the mid 1900s.

Zach Aysan is a software expert from Toronto working to protect the public. This piece is adapted from the essay "Self-Crashing Cars."

Article Tags CULTURE, HACKING, POLITICS, STUXNET, COMPUTER HACKING, TESLA, TODAY'S BLOGS, COMPUTER UPDATES, ZACH AYSAN