



# Autonomy, Loyalty and Trust in the World of Machines

Eudoxa Policy Study #5

**Eudoxa Policy Study #5:**

**Autonomy, Loyalty and Trust in the World of Machines**

Written by Dr. Anders Sandberg.

The Eudoxa Think Tank publishes this series of policy studies. They address issues that are current or will be in the near future. Eudoxa Studies are in depth studies on how emerging technologies impact our culture and our society.

Eudoxa AB  
Sveavägen 133  
113 46 Stockholm  
Sweden  
+46 (8) 338-168  
[www.eudoxa.se](http://www.eudoxa.se)

Copyright © 2003 by the Eudoxa Think Tank AB

ISSN 1652-0726

Nothing written here is to be construed as necessarily reflecting the views of the Eudoxa Think Tank or as an attempt to aid or hinder the passage of any legislation.

## Table of Contents

*Table of Contents*..... 1

*About Eudoxa*..... 2

*Executive Summary*..... 2

*Autonomy*..... 3

*Loyalty*..... 4

*Trust*..... 7

*Autonomy, Loyalty and Trust – all together now!* ..... 9

## About Eudoxa

The Eudoxa Think Tank is a think tank based in Stockholm, Sweden. The main focus of the group is explaining the cultural impact of emerging technologies integrating the analysis with classical free-market ideas and dynamist thoughts of experimentation, innovation and decentralization.

We work for a diverse society based on a strong moral foundation of individual rights, where individuals have the right to utilize modern technology and medicine according to their own moral judgment. We believe this foundation promotes tolerance and acceptance that will tie our society together, not break it apart. The inspirations behind our vision are a firm belief in individual liberty, free enterprise, a limited government and that ideas have impact on our society.

Eudoxa currently has staff working out of Stockholm, Sweden and Kansas City, Missouri, giving them the advantage of being able to approach both the European and American market with a comprehensive perspective drawn from experiences from both continents.

## Executive Summary

We are surrounded by machines that are increasingly autonomous, although the autonomy we have managed to create is far from robust or comprehensive. Simple devices such as automatic doors might be broken. In other instances the automation will behave wrong, because it does not interpret the input correctly. Or the automation will behave according to specification, but the interaction between the machine and the human will produce unexpected behavior. Who is to blame when autonomous technology fails?

We are used to our tools being loyal to us, but will that always be the case in autonomous machines? Our computers provide numerous examples of machines that communicate with its creator without the knowledge or consent of its user. Malicious code has even sent people to jail until it could be proved that the software acted on its own accord. The question of who an autonomous machine is loyal to is eventually a question about security.

It all boils down to the eternal issue of trust. The author looks at trouble with current Trusted Computing solutions, such as monopolization and less dynamic markets.

Increased autonomy leads to reduced trust and loyalty. Trust is something you earn, machine or human. Increased transparency in the ability of a machine is one of several keys to the solution. Another one is to redefine how we look at machines, the author suggest that we could gain from considering them as, and treating them as pets. They are not completely in our control, but we have a certain degree of authority over them based on a mutual relationship. He concludes the study this way: "The problems are stupid machines, stupid laws and stupid people. Let us try to fix all three of them!"

## Autonomy

Someone or something is autonomous if it decides what it does for itself with no direct outside control. We humans – and living beings in general - are mostly autonomous, but technology is becoming autonomous too.

As a computer scientist and neuroscientist it is very much my business to come up with ways of making machines more autonomous and independent; more like living beings. We are still far from the robustness and flexibility of even an insect, yet alone the shimmering goal of real intelligence. But it does not matter, since we are already surrounded by simple autonomous technology.

A classic example is the thermostat. It turns on the heat when it gets cooler than a certain temperature, and turns it off when it gets warmer. It makes a simple decision and does not need anybody to decide for it. Other autonomous devices we meet every day are automatic doors, automatic windshield wipers and of course much of the programs in our computers – everything from instant messengers to web browsers and anti-virus programs.

Autonomy poses a number of problems. The simplest is when it does not perform at all. The broken thermostat, the automatic doors with clouded lenses that do not open and hence makes us bump into them. Machines break down all the time, even autonomous ones, and relying on them is always a bit uncertain.

A trickier situation is when they do the wrong thing than they are supposed to. A very good example is the glass windows above the central atrium of the SAS corporate headquarters offices north of Stockholm. They were designed to open during warm days to let air in and close when it rained. In case of a fire they would open regardless, to let the hazardous smoke out quickly. After being installed everything was fine... until the first summer thunderstorm. First they closed when they detected rain. And then they opened because the lightening mistakenly made them believe there was a fire. They opened just in time for the biggest rain torrents, and incidentally the main archives were built under the atrium. The costs for SAS for this design mistake became very high indeed.

The problem was that the design was almost right, but the system made the wrong decision. One could probably redesign it to “know” about lightening, but there are a million other potential problems. Each is so unlikely that it is not worth bothering to implement a safeguard, but taken together they are so many that problems are bound to happen.

A third problem with autonomy is when the system makes an unexpected decision. The crashes of the Swedish JAS-39 Gripen assault aircraft are a good example. The plane moved in one direction, the pilot tried to compensate (unnecessarily), the plane compensated for that, thus the pilot tried to compensate even more, and disaster struck. Here the problem was a failure of communication. The plane and pilot did not work together as a team.

Many of these problems of autonomy would be just as common with a human running the system. A door opener sometimes misses that someone is

approaching; a janitor regulating the roof windows is out on a coffee break when the rain starts or someone happens to push incidentally the opening button during a rainstorm, and airplane pilot teams have failed to communicate properly, sometimes with fatal consequences, for a century of air transports.

But when a machine is involved there is nobody to blame. One cannot argue with these systems and they have no common sense. This is far more irritating and dangerous than incompetent staff. It also leads to some interesting practical, legal and social problems, as we are moving into a world filled with autonomy.

In the long run we are probably moving towards a "techno animistic" world where a device spirit, able to communicate and act in a limited and sometimes smart fashion, inhabits almost everything. But whom do the spirits answer to?

## Loyalty

When somebody or something is loyal to you, they act in your best interests and do not act in the interests of others, at least not if they relate to you.

We are used to the fact that everyday objects and tools are loyal to us. As long as we hold it properly the knife will work as we want it to. We may cut others with it, or ourselves, but that is due to our lack of skill or destructive intent. It is always the owner or user that is responsible for the action because they initiated it and the tool only passively continued it. Even a powered device such as a car or a chainsaw only amplifies a human decision. This makes assigning blame when something goes wrong simple in most cases. It was whoever used the device that was responsible for the act.

By limiting access to tools we limit their potential for shifting loyalty. Ownership often connotes access control, although mostly by placing owned stuff within the access controlled containers of locked buildings, cars and suitcases. By adding keys and codes to our tools we ensure that they will only be used by our designated representatives or us. One such application is making guns "smart" so that they will recognize a registered handprint and only fire when held by the legitimate owner. This would hopefully reduce the number of policemen hurt or killed with their own weapons every year and other accidents with handguns (although many gun users are highly skeptical of issues of reliability).

But when equipment becomes more advanced and autonomous, loyalty becomes more problematic to ensure.

Some printer producers add a chip to their printer cartridges, enabling the printer to detect if they are using the producer's own brand of cartridges or a competitor's (the same is relevant for some mobile phone batteries). This way, competition can be prevented at the expense of the printer's owner – the printer is not quite loyal to him.

Radio frequency identification tags (RFID tags) are being placed into more and more products. A microchip connected to an antenna allows them to send back an identification code when requested by a reader. They help identify goods in supply chains, enable stores to update inventories, prevent theft and run walk-

through-counters that automatically debit the buyer the goods bought. Used right they could enable automatic recycling and smart homes where everything can be located. But they also enable tracking people through their individually marked possessions and to find out what they own. Our possessions are getting too talkative.

Many computer programs regularly connect to servers at their manufacturer's system in order to check for updates. If they find any, they ask the user if they wish to install them (this happened at least twice – that I know of – while writing the text for this Eudoxa Policy Study, and most likely many more times that I am not aware of). In principle, the programs could just go ahead anyway and nobody would be the wiser; some software has this as an option, and sometimes it is actually the default option. This automatic updating is mostly a desirable thing. It allows new functionality, and fixes bugs and security holes automatically. But it can also make a program behave in unexpected ways, interfere with other programs and add security and privacy breaches. Even more risky is the installing new software. Many software packages come with default add-ons of unrelated software that change home pages, add advertisements, redirect the modem to a pay number, send customer information to their manufacturers (so called “spy ware”) or interfere with competitors' software that is installed on the computer – and this is still not deliberately malicious code (“malware”). Often such add-on software is quite hard to get rid of, and several programs actively resist removal software by disabling it! This is of course not just sleazy (or desperate) business practice, but also a case of disloyalty. These programs are no longer acting in the owner's best interest, but act as a fifth column inside the computer, trying to promote the interests of others.

Even worse risks come from deliberately malicious code. A case in point is the case of Julian Green. He was recently exonerated of possession of child pornography because his computer had been hacked. At some point a Trojan program was run that opened Mr. Green's computer to store pornography, possibly by a direct malicious act or by clicking on the wrong link on the wrong website. He was arrested for child pornography and risked jail, lost visitor's rights to his daughter and was generally shunned in his community for being a sexual predator. In his case it was possible to show that in all likelihood he was innocent and exonerate him, but it does not take much imagination to see how such software could destroy entire lives.

In the past, disloyal equipment such as devices with planned obsolescence or deliberately limited compatibility could be handled through easier means. The disloyalty of the product was easily observable and could be limited through competition, spread of information among customers, and to some extent through legislation. The present's problem with disloyal autonomous equipment is that it moves faster than these processes. The program you have been using for many years without problems or complications may suddenly reveal itself to be a Mr. Hyde and turn on you – or even more sneakily, start to report back information you would prefer to keep private to other parties. Many of the most serious cases of software disloyalty today involve silent breaches of privacy that

are unobservable except for those paranoids that constantly check their computer's firewalls.

There are also loyalty problems where there is no particular interest to complain to or prosecute: the software misbehaves for reasons of its own, and not even the manufacturer can tell why. Beyond a certain limit of complexity there is no longer a clear cause and effect in technological systems. Several of the problems mentioned in the autonomy section above have no clear answer that is easy to fix. But we should at least strive to avoid compounding them with deliberate loyalty problems.

Disloyal equipment complicates finding responsibility. In the case of the child pornography Trojan program it tried to force the responsibility for the pornography crime on the victim. Given that Mr. Greene was exonerated, the Trojan defense is found to become viable, and deliberately placing such Trojans on one's computer might give child pornographers at least some defense in the case of being caught. One could even imagine a version that actively – like a computer virus – tried to install itself on computers and find other computers to infect (such viruses already exist, attempting to set up server networks for a variety of shady purposes). The ultimate responsibility lies of course on the people that develop the program, but what about the liability of people with bad security (or just the bad common sense of opening email attachments)?

The lack of liability from software companies seriously compounds this problem. If you do not update your Windows installation you are responsible for the lack of security. But if you use the new update, install it while following the best practices and it still introduces a new security fault unknown to you, who gets the blame when that security fault causes damage?

The EULA from Microsoft that I clicked “I accept on” when installing a software patch forswears Microsoft from liability. But I have myself been acting according to best practice. Here it seems the chain of liability becomes broken. On the other hand, holding Microsoft liable for all damages caused by insecure software would likely drive it out of business (neatly solving the monopoly debate) – but it would probably bankrupt every other major and minor software company too, including the free software projects. On the third hand, letting vital parts of the infrastructure remain as insecure as they are now might be an even larger problem.

The renowned American security specialist Bruce Schneier mentioned in a discussion another example of how the insecurity and potential autonomy of software complicates things legally about digital signatures and authentication. Signatures are necessary for secure digital communications, contracts and e-commerce – we need to be able to prove that we are who we claim to be in order to negotiate binding contracts, and it must be impossible afterwards to renege on this authentication.

But as Schneier pointed out, if he was ever called to witness in court about such a case, he could not truthfully claim that a particular person signed a digital signature even if the cryptographic protocols involved were impeccable. The

reason is that in signing, what actually happens is that a series of programs performs the actual cryptographic operations unlocked by the user writing their password (or using their thumbprint, DNA or whatever). If disloyal software was present in the machine it could intercept this communication and sign contracts that the user has never seen (or switch the contract the user thought he was signing for another contract). The only way to prove that the signature was authentic would be to prove that no such software had been present in the machine at that time, a daunting task that could very likely be impossible to perform. Here the mere possibility of disloyalty disrupts the entire legal certainty of digital signatures.

As you might have noticed, this discussion regarding loyalty appears to have slipped over into a discussion of security. This is no coincidence. Loyalty and security are – technologically – largely identical issues: how do we guarantee that devices do what we as owners want and not what others want? The focus in security has been mainly on direct attacks from the outside (like hackers and pirates) while my loyalty concerns are more of an inside problem with the producers of the devices and software.

One lesson to be drawn from this must be that computer security is becoming more and more part of ordinary everyday life. Just as we tell our children to lock the door and not to follow strangers, we now must tell them not to open email attachments or tell others present in the chat rooms where they live.

Another important lesson is that our exo-selves – our entourage of autonomous machines and programs – pose us with serious security and responsibility problems. The more complex and autonomous they become, the harder it is to keep control over them. The irony is that we get most of our machines when they save us work and make our lives more comfortable. But the better they become at this task, the more potential for unpleasant surprises they offer.

## Trust

We now come to the thing that has been lacking today: trust. To trust someone means to believe that the other will behave in a certain (usually beneficial) way towards us.

One attempt to solve the problems mentioned above is the Trusted Computing initiative. Trusted computing as a concept comes in many different flavors, but the basic idea is to add some impregnable hardware to computers that enables control over exactly what the software is doing. It should ensure that software processes couldn't be altered and interfered with by other programs or the user. It should store data in an encrypted form so that no other program than its manager's can decrypt it. It should enable secure input and output to the user, so that no other software can alter or see it. And finally, it should be able to prove (in a strong cryptographic sense) to remote systems precisely what local program is running.

This would solve a number of problems. It would enable digital rights management that would make it impossible for the user to pirate contents she

had licensed, or send restricted files outside secure networks. It would prevent reverse engineering as well as sabotage of software. It would limit the power of viruses and Trojans. It would (ideally) make digital signatures stick. It would enable connecting client software to remote servers where both parties could be sure the other was running the right software (very useful both for P2P networks, online gambling, games and trade).

The problem with this is that it is all about making sure that the programs (and its maker) get what they want. They know they can trust the users, or at least their computers. But is there any reason for the users to trust them? In some cases trusted computing could achieve this, such as in verifying that an online gambling system runs software that has been inspected by officials and found to be fair. But in many situations it would only exacerbate already existing power relations. It is not Trustworthy Computing.

Much has been written and debated about Trusted Computing, ranging from corporate hopes for security and prosperity to paranoid rants about the imminent New World Order controlling all our computers. There are some very real and serious concerns about technological lock-in, or the ability of software/hardware monopolies to permanently squeeze out competitors from the market. Perhaps most serious is the risk of reducing software innovation. One of the key reasons of the entire computer revolution has been the low cost of innovation: anybody can start writing software with minimal investment, unlike manufacturers of physical goods. This has not only produced a low threshold for market entrance, but plenty of user innovation. Unfortunately, it seems likely that Trusted Computing as it is discussed today would seriously raise the cost of innovation by forcing innovators to buy relatively expensive certificates. There is also the important risk raised by among others Lawrence Lessig of locking up the cultural commons.

The present situation is that we do not own much of the contents of our computers. We license most of our key software and we mainly just own the files we have produced ourselves. Trusted computing might make this situation even clearer: we might own the physical hardware (with the possible exception of the tamper-proof cryptographic chips performing the trusted computing part?) but all software is licensed and in the end controlled by somebody else. While this might prevent some of the autonomous misbehaviors mentioned previously in this text, it will not prevent all of them. If suddenly your word processor program decides to report you for writing what it considers illicit political manifestos, the computer will let it.

There is clearly a great need to get Trusted Computing right, and this requires both technical and legal discussions. We need guarantees of the ownership of the information we produce, and that it will not be spread by inquisitive programs. This can be partially controlled through coding (making it hard for untrusted applications to get it) but also through law (making it illegal to gather it without permission). There is likely a need to also safeguard a large area of non-Trusted Computing to enable cultural commons, innovation and software competition. We want to establish that Trusted Computing is trustworthy, just as we want to

establish that software suppliers are trustworthy (and we trustworthy by them) – not to mention that there is mutual degree of trust with the government’s systems. An impressively tall order, but necessary if we want to avoid turning Trusted Computing into a nightmare we are cryptographically unable to wake up from.

As any security expert already knows, there is no real security anywhere. Most security we encounter is actually just trust. We trust the security guards (and they us); we trust that our landlords do not sell copies of our front door keys to thieves and that guests we invite to our home do not steal our possessions. The cryptographic ideas behind Trusted Computing are great on paper, but will fail in practice in the face of a determined adversary. And such adversaries certainly do exist, and can make many of their tools available to anyone that desires them. But even ignoring piracy rings, the complexity and autonomy of our systems reduce security. Software will find a way to interfere even in trusted computing. Applications will make the wrong decisions. Users will write passwords on post-it notes and put them on their screens.

Perhaps one way of both gaining the benefit of trusted computing, and retaining a high level of innovation and cultural commons, would be to promote a change in how software is sold. Rather than just license the software to the customer it should be sold, placing the responsibility of its upkeep on the user– just as we place the responsibility of upkeep and safety of cars and houses on their owners. But just as we insure ourselves against accidents and mishaps with such capital goods, we could also insure ourselves against software problems.

Some might suggest that litigation could act as a useful correcting factor on the loyalty of autonomous systems. But this is unlikely to work in the European legal systems, and it is also too slow – as we have seen, many problems develop and spread swifter than legal systems can keep abreast. Insurance companies are still relatively slow, but far faster than legal systems. They can also introduce soft constraints on behavior in the forms of premiums, and different kinds of insurances for different systems, unlike many legalistic approaches. They also have incentives for promoting education about safety, good design practices and other forms of information feedback.

As systems become more autonomous we must cease to view them as tools and more and more as pets – or even slaves. We acknowledge that animals may misbehave and place them in a somewhat different legal category from mere tools. It seems likely this autonomous possession category needs to be expanded in the future to encompass a variety of independent systems. In the real long run we might have to start considering their welfare too, just as for pets. But that is still some years off.

## **Autonomy, Loyalty and Trust – all together now!**

How do autonomy, loyalty and trust relate to each other? If somebody or something is loyal to us, we tend to trust it – even though there might still be some degree of bugs and unpredictability. On the other hand, if we trust

something we do not necessarily expect it to be loyal. We trust merchants selling us things, despite their self-interest, because we know being fair to us is in their interest. The more autonomous the other part is, the less guarantees we have of their loyalty and trustworthiness. It has to be earned, and maybe that is both the key problem and a possible solution.

We need greater transparency to check that our equipment is trustworthy. The open source approach is a partial solution: by allowing the source code to be reviewed the programmers show they have nothing up their sleeves. But we might also want to view what is going on in our machines' minds. Nothing convinces us so much as finding out why somebody or something did wrong, since it makes us able to predict under what conditions it will do it again.

The smarter our devices become, the more they will be able to change their own behavior. This makes them more and more unpredictable, but also enables them to learn. We can train devices to act in particular ways, but that usually requires some leeway for them to fail in the first place so we can tell them "Never delete all my business files again!" and make them learn this important lesson.

Fortunately, truly learning software could perhaps share these experiences: as one digital assistant learns not to delete business email even when the email seems to make its owner upset, it can send this knowledge to all other instances of the assistant within the company (or even the same manufacturer, if the owners agrees to it). This is a beneficial autonomy, as long as we can keep track of what lessons are sent where.

Learning to live with autonomous systems is going to require both changes in what we expect of machines, how we use them and what laws and institutions we surround us with. The key problem in this area is at the moment, the lack of communication between the engineers creating the technology, the users using it, and the legal and political sphere where official laws are created (the other groups also create their own rules of engineering practices and social norms, that are often just as binding as legal agreements).

The current debate about intellectual property shows how dangerous this lack of communication can be. They are held in two separate communities, one technical, and one economic-legal, with nearly no communication between them. Due to this separation laws are enacted that are unenforceable (and hold great risks for integrity, freedom of speech and innovation while not providing any corresponding benefit) and technologies are developed with no heed to their legal standing or consequences. What are needed here are not laws that are technology-independent, since such laws are usually just technology-ignoring and usually just circumvented by new technological advances. Neither are we helped by the fast growth of a shady gray market for software that thrives when business models, technology and rules do not fit the demands of the citizens'.

Autonomous machines are going to have far more radical consequences than easy digital copying. They will likely require even greater changes, than presented here, in how we make business and organize our lives. But we can shape the abilities and application of the autonomous machines. We can figure out technical, social and legal solutions to the problems the pose beforehand and

while they are being taken into use. It will not be a sudden transition unless we choose to ignore the issue until it is too late.

The problems are stupid machines, stupid laws and stupid people. Let us try to fix all three of them! But we should not expect that we would succeed with pulling off a perfect fix, so we need to design for robustness too. When things fail, loyalty is misplaced and trust is broken, so we must be able to rise again, dust of our clothes and continue in a better way.