Subject: User-Trusted Computing Definition
Posted by connie on Tue, 13 Feb 2018 19:28:36 GMT
View Forum Message <> Reply to Message

UTC Definition

For the purposes of this definition, an anti-feature is a programmed behavior which is detrimental or outright malicious to a computer system and/or its user, and which can be used by hostile parties to exploit the user and/or the computer system. Anti-features can be distinguished from exploitable side-effects because the latter are challenging or impossible to mitigate, regardless of circumstances, while anti-features can be easily removed by at least some parties other than the user.

A computing system or component is UTC when it is designed to act in full agreement with the user's will, to the extent that is technically possible. A UTC system

1. Should not contain any anti-features.

2. Should not endorse or suggest any computing components with known anti-features.

3. Should not make it difficult to detect and/or remove an anti-feature when one creeps in by accident or is introduced by a hostile party.

UTC Implies Free Software

A software component must be free in the sense used by the Free Software Foundation in order to be UTC.

Freedom 0 is required because any behavior which arbitrarily prevents a user from running a program is an anti-feature.

Freedom 1 is necessary to detect anti-features.

Freedom 2 is folded into Freedom 3.

Freedom 3 is required to remove anti-features. Software development is a collaborative process by its very nature, and the only way a typical user can afford to modify software is by joining a large community of users and developers. Within that community, there should be no barriers to distributing copies of the software, either modified or unmodified, either in source or in binary form. The reason for that is any form of censorship at the development stage can and will be used by censors to insert and/or protect an anti-feature.

Not Just For Software

When we talk of computer systems, we are not talking merely of the software component, or even the soft+hardware components. These and other components of the process we call computing can be evaluated with respect to UTC either individually or as a whole. In general, we can and often should take a step back and look at the entire computing ecosystem in question. If we evaluate a software package, for example, we can consider a wide variety of factors, included but not limited to licensing, documentation, intended/actual platform, intended/actual audience, and everything up to and including the political process within the community of users and developers. We are forced to take this very holistic approach because none of these factors can guarantee a relief from anti-features when used in isolation.

Specifics by Examples

Anti-features Versus Side-effects

If a hand-held device wants to use a wired or a short-range wireless network, then it will leak its physical location to the network infrastructure. This behavior can be exploited by third parties, starting with the network provider, but it is unclear how to mitigate it: the physical location does not have to be reported, since the network equipment knows its own location, and knows that the connected device is very close by. This is what we would call an unpleasant side-effect.

On the other hand, a programmed feature which reports the exact location periodically over the network to a third party can be seen as an anti-feature, as long as significant difficulties arise when a user attempts to get rid of it without losing any other functions provided by the computer system.

Suggesting and Endorsing Anti-features

When a user opens a Web browser and uses a third-party, general-purpose search engine, the user assumes full responsibility for avoiding anti-features. Nothing in these search results should be interpreted as a suggestion by a UTC system to obtain an anti-feature.

On the other hand, when a Web browser (like Firefox) suggests non-UTC plugins via its own configuration interface, it makes a definite endorsement, which makes it non-UTC. A trivial solution would be to supply the same plugins via a regular Web page, along with an explicit un-endorsement of this software compilation, or at least the part of it which is known to be non-UTC. Of course, an even better solution would be to purge the non-UTC plugins.

Holistic Approach

AdBlock Plus

Licensing alone cannot stem the anti-feature creep. As the famous example of AdBlock Plus have shown, a permissively licensed piece of software can in fact introduce an anti-feature and start exploiting users, even though in theory this kind of behavior is unlikely, due to the possibility of forking, which is guaranteed by the license. Yet as it stands, AdBlock Plus is free software, but not UTC, having included an advertising-related anti-feature. A major role in this evaluation is played by the facts which have nothing to do with licensing.

What matters in this case is the ongoing presense of an anti-feature, which enables the exploitation of users by advertisers, and therefore also by the lead developer of ABP, who is getting paid by advertisers. And even though this anti-feature has been removed in multiple forks of ABP, there is no practical way for ABP users to fix ABP itself. This is in part because the development is controlled by a single person, who benefits from the anti-feature directly, and the history of alleged corruption erodes the hope that frequent software updates will not re-introduce the same anti-feature, or perhaps a different one. Besides, ABP is also afoul of UTC(3), since this software is designed to make the detection and the removal of the anti-feature harder than it should be, by making the malicious setting the default.

Signal

Another example of a permissively licensed software which is not UTC is Signal app produced by Open Whisper Systems. Signal is free software but it comes with an easy-to-miss anti-feature: it fails to protect the privacy of conversations due to specific design choices made by its developers. According to their own documentation, in order for Signal to function, the encryption credentials must be generated and set up on a system known to be fully compromised by a user-hostile party (users get to pick between Google and Apple), so users are goaded towards non-UTC systems, which violates UTC(2). As a direct result, all encryption keys are made available by design to untrusted parties, and OWS is definitely aware of it, but is very careful not to mention it to its users. Instead, they claim falsely that no one can decrypt the private conversations, which is a definite effort to conceal the anti-feature, violating UTC(3).

Subject: Re: User-Trusted Computing Definition
Posted by rk4n3 on Wed, 14 Feb 2018 05:18:02 GMT

View Forum Message <> Reply to Message

I'm not quite satisfied with this treatment of the term "anti-feature", in that I think that it conflates two concepts that should be distinct for practical and significant reasons in the context of the clarity being sought in this definition (UTC).

The two concepts I'm referring to are commonly known as "bug" and "malice". The reason the distinction between these concepts is critical, imho, is that one should preclude a qualification of software as "UTC", while the other should not.

The difference between a "bug" and "malice" is primarily one of either intent or neglect. An analogous illustration might be the difference between an "error" and a "lie" ... if one makes a mistake in arithmetic and bases a conclusion on it, with no intent to be incorrect, it could not be called a "lie", as lying requires intent to deceive. Similarly, if software behaves in an unintended way because of an error in the author's logic, despite all intent by the author to implement the desired behavior, it can't be called "malice", because malice requires intent.

My assertion is that a construct can be qualified, and remain qualified, as "UTC" while being afflicted by "bugs", under certain conditions. By contrast, a construct can never be qualified, or remain qualified, as "UTC" while being encumbered with anything malicious at all.

Some "bugs" will be, by nature, particularly immersed in or causal of concerns that the term "anti-feature" is intended to cover, and so could be classified as "anti-features" ... but certainly not all (and I'd content not even a majority of) "bugs".

I'd also like to point out the difference between these concepts and those of "side-effect" and "omission". While the notion of "side-effect" is correctly treated in the definition so far, "omission" deserves similar exclusion from what "anti-feature" covers as well, imho. The changes to a construct required to address an "omission" can be described as simply doing work that hasn't been undertaken yet. I'd assert that "omissions" deserve treatment identical to "bugs", in that some may, in light of certain criteria, deserve classification as "anti-features", but certainly not all (or even many) would.

I suggest that the term "anti-feature" thus becomes a bit more complicated because it should refer an overlap between anything malicious and SOME other forms of deficiencies. If the qualification of "UTC" is to be based on what the term "anti-feature" means, then aligning the term with the intended/desired outcome should be of self-evident value - and I'd assert that alignment to be as I've described here, which would resemble something like:

The term "anti-feature" includes:

Anything malicious
Some kinds of bugs, as qualified by certain criteria
Some kinds of omissions, as qualified by certain criteria

With all this said, I do believe insightful and "correct" sets of criteria for what the term "anti-feature" covers can be arrived at, so that the term can be used as a cornerstone for assignment of a "UTC" endorsement/certification that aligns with intended outcomes - it will just take a bit of crafting to get there, imho.

Subject: In conversation with Julie Marchant
Posted by connie on Sun, 18 Feb 2018 21:49:34 GMT

View Forum Message <> Reply to Message

The contents of this post are the edited version of an email thread between me and Julie Marchant. Published by permission, this entire post is licensed under CC0.

connie:

Based on the kinds of topics you usually discuss, I thought you might be interested in our most recent mini-project, where we are brainstorming an alternative to the FSF's free software definition and things like that. To be more precise, while similar to the free software definition, it is intended as an alternative to FSDG; not as a replacement, either, but another take on the task of evaluating and certifying projects such as gnu+linux distributions, which are not mere software products, but come with a whole ecosystem attached to them.

JM:

I think your definition of "anti-feature" is too vague, and I disagree with the conclusion that Adblock Plus has an anti-feature. I suppose that is in reference to the way that extension approves certain ads by default. But that's the point, really: this is only the default behavior. Adblock Plus doesn't stop you from blocking approved ads.

I don't know about Signal, but the description given lends to me disagreeing with the conclusion that Signal has an anti-feature. To not perform encryption at all would have the same effect, and that would not be an anti-feature.

connie:

May be it would help to clarify that any kind of behavior, even emergent behavior, can qualify as an anti-feature; doesn't have to be something we can narrow down to a few lines of source code.

ABP has a traditional feature, which users can disable of course, but I want to see the definition triggered by Palant's ongoing effort to make sure as many users as possible have it enabled, as well as by the fact that he's got a massive conflict of interest, and will probably not hesitate to re-enable it sneakily during update under some bullshit pretense, so no, no one can really hope to disable it completely. I mean, when he announced that the "feature" is in, people were so spooked that several forks emerged within weeks, which shows a near-complete erosion of trust by users.

Signal's insistence on sharing all credentials with Google or Apple is not a feature in a traditional sense, but is an emergent behavior nonetheless, and the one which is pre-determined by the design choices OWS made.

JM:

I understand why it is classified that way, I just disagree with that imperative. I rather like Benajamin Mako Hill's definition of an anti-feature: a feature that is so bad that some users would pay to have it removed. That is to say, "features" that are to the detriment of their users. Faulty encryption is not to users' detriment; they can simply add on their own encryption system (like OTR). Optional whitelisting of ads is not to users' detriment; they can simply toggle the option to not whitelist the ads.

I'm not entirely sure that basing a definition on concepts like this are really useful, however. Not all proprietary programs have anti-features, and such an exclusive appeal to practicality lends itself to non-solutions (like locking up proprietary programs in sandboxes rather than excluding them).

connie:

Thanks for the reference, this is indeed where I got the idea, but I forgot the guy's name I will make sure to cite that video in the forum. And while I do not use his language, I do perceive my current definition as a more circumscribed version of his. While I understand the intent of his definition, it leaves the door wide open to call anything anti-feature as long as there is a user or two who would pay to get it removed. I tried to constrain that somewhat by further stipulating that it should be exploitable or at least detrimental in some sense, and that there should be at least one party capable of removing it without running into severe technical difficulties.

Without these stipulations, one can identify a feature they find unpleasant: say, systemd; declare their desire to pay to remove it, and then run their head against a wall of perfectly free+libre distributions with all kinds of developer communities, all of which adamantly refuse to remove it for what they perceive as technical reasons. And we can agree, I hope, that while systemd is not perfect, there is no compelling reason to consider it malicious or screwing its users somehow.

Sure, there are folks who will say NOOOOO, systemd is cancer, and the damage it does is long-term, EEE and all that, but at least with my definition they have to start building the case and collect the evidence, whereas the naive version allows them to declare it as anti-feature right away, and declare every distro which uses it an anti-feature garden.

JMFaulty encryption is not to users' detriment; they can simply add on their own encryption system (like OTR).
The deviousness of OWS is truly great, it seems, as you still don't seem to understand precisely what they did. Their encryption algorithm, to the best of my knowledge, is as flawless as pidgin's. But there's no point to OTR when an untrusted (really, hostile) party such as Google or Apple has a near-
permanent, direct, and surreptitious channel by which to access private keys, keystrokes, and the screen. Once again, OWS, the maker of Signal, designed it so that every user is forced to set up on a compromised device, compromising xerself at the moment of keygen, and remaining fully compromised ever after. If you read their documentation, you will see that even though they have a chrome plugin, you cannot use it to set up. You *have* to use a spy-phone.

So I would say that Signal will become OTC automagically the moment there is a functional free+libre phone with non-trivial market share. Alternatively, OWS can allow users to use the "desktop" version without ever using a spy-phone. This last option seems like something OWS could implement easily, which is exactly what makes it a full-fledged anti-feature: there is a non-user party with the ability to fix it, but not the desire.

JMOptional whitelisting of ads is not to users' detriment; they can simply toggle the option to not whitelist the ads.
But just the act of whitelisting the parties which pay off the dev team IS to users' detriment, and we can't call it optional in any practical sense when the EDFL in charge of the project is corrupt, and would not hesitate to (re-)enable the feature behind the users' back, or lie to them with the intention of convincing them to enable it. And there is ample, ample documentation to the fact that ABP is developed by scammers.

JMI'm not entirely sure that basing a definition on concepts like this are really useful, however. Not all proprietary programs have anti-features,
I agree. Open-source proprietary software may well be free of anti-features, but since it makes it harder to remove them in the future, it gets the axe. Blobs should be assumed to contain anti-features by any sensible 21st century person.

JMand such an exclusive appeal to practicality lends itself to non-solutions (like locking up proprietary programs in sandboxes rather than excluding them).
As it stands, I argue that all non-free software runs afoul of our definition by virtue of making it harder for the community to remove current and future anti-features. Are you thinking something

like a blob which is so well-contained that we would consider it harmless? We definitely can agree on making sure that our definition should continue to exclude all non-free software in most unequivocal terms, regardless of how well it is isolated within the computing system, broadly construed. I would go as far as to argue that a piece of free software which is highly dependent on a proprietary component elsewhere in the ecosystem is already suspect, because my vision of a "computing system" is very holistic. I'd like to think that "computing" really happens within large communities consisting of people, hardware, and software.

Here's a completely off-the-wall question prompted by this last topic:

Is electron (you know, the seemingly elementary so-called-particle) a computational blackbox? It seems to compute alright, but we don't have the source code. Does that mean we should be careful about using it?

JM:

connieSure, there are folks who will say NOOOOO, systemd is cancer, and the damage it does is long-term, EEE and all that ... and declare every distro which uses it an anti-feature garden.
True enough, but I look at it in a different way: this is just an indication to me that this sort of definition is impractical.

connieThe deviousness of OWS is truly great ... You have to use a spy-phone.
So, in other words, Signal is dependent on proprietary software, and that compromises the security of your system.

I think this is a good demonstration of why I think this sort of definition is impractical. See, from your description, I was under the impression that Signal was a libre program, but that the servers it worked with had access to its encryption keys, rendering the encryption useless. If this were a proper definition of the problem, it would be easy to solve: just add another encryption layer, one not accessible by Google or Apple, on top of the built-in encryption. This is why I used OTR as an example; as I'm sure you know, XMPP has no end-to-end encryption by default, so what OTR does is send an encrypted version of the text you told it to send. The XMPP network has no idea that it's encrypted, and it's then up to the opponent's XMPP client to recognized what the encrypted text means.

But since Signal is dependent on proprietary software, it doesn't work that way. It would be possible for the proprietary dependency to subvert any attempt to do this, and even if it weren't because of a sandbox, Signal is not truly libre if it depends on a proprietary program, and that's the important point.

connieBut just the act of whitelisting the parties which pay off the dev team IS to users' detriment ... And there is ample, ample documentation to the fact that ABP is developed by scammers.
Some of these appeals would function identically if you were trying to oppose use of systemd, which I see you agree is unreasonable. Some of it is in reference not to the software, but to its developers. This is a distraction; it doesn't matter who developed a program. A libre program developed by Hitler is still libre, and a proprietary program developed by Jesus is still proprietary.

You don't have to trust a libre software developer. If you don't trust the ABP developers, you're free to fork ABP (which has happened multiple times), or you can simply refuse to upgrade.

connieIs electron ...
No, an electron is a component of our universe. I don't even see how an electron is similar to a computer program in an abstract sense.

The issue at hand with proprietary software is not just that we don't fully understand it. It's that we don't, but someone else does. The issue is not really knowledge itself, but the power dynamics involved.

If, somehow, nature ended up causing a computer to exist on its own with software on it that we have no source code to, that wouldn't be proprietary software. It would be natural software. You could say that natural software might be dangerous (and indeed, nature often is dangerous). But you can't say that one person has unjust power over another person in that situation, so it's not an ethical issue.

References:

Antifeatures article by Benjamin Mako Hill
Antifeatures talk by Benjamin Mako Hill