Reverse-Engineering Someone Else's Software: Is It Legal?

Pamela Samuelson, University of Pittsburgh

Does reverse-engineering software infringe intellectual-property law? While opinion is divided, courts seem to be saying it's legal if you take no actual, protected expression.

Lawyers disagree about whether it is legal to use the sophisticated reengineering technology now in existence (and being developed) to reverse-engineer programs developed by other firms. The benefits of the reengineering technology all but ensure that the law will not ban the technology outright, just as videotape recorders could not be banned from the market merely because the machines could be used to make unauthorized copies of copyrighted movies (even though two movie studios once sought such a ban against them).

Because so much of the valuable software available in today's market is copyrighted, copyright law has become the main battleground of the legal debate over whether reengineering technology can be used to reverse-engineer other firms' programs.

The technicality on which this legal debate hangs is whether it infringes the other firm's copyright if you make unauthorized copies of a copyrighted program, either in the course of analyzing it or to analyze it. Some lawyers believe it does infringe the copyright. Other lawyers do not.

While there are conflicting judicial decisions on the legality of reverse-engineering under copyright law (see the box on p. 92), the trend in US case law is toward finding copyright infringement only where the reverse engineer has developed a competing program that is substantially similar in expression to the copyrighted program whose code was analyzed.

This trend is not surprising if you understand the traditional principles of copyright law and doctrines of other intellectual-property laws like patents about reverse-engineering activity.

Defining terms

Technologists may find a sharp contrast between their own perspectives on whether such conduct is or should be legal and the perspectives of lawyers. For
lawyers, such questions must be asked in the context of an existing legal framework — a framework in which technological issues may have little or no relevance.

**Definitions.** Even how technologists and lawyers define the term "reverse engineering" may differ. From a technologist’s standpoint, reverse engineering is generally understood as the act of creating a set of functional specifications for a system by someone other than the original designer based on an analysis of an existing system and its component parts. In the case of software, you can reverse-engineer it through a variety of means.

For intellectual-property lawyers, software reverse engineering has two meanings, both of which differ from the technological meaning, but only one of which is troublesome under the law.

There is general agreement among lawyers that studying a copyrighted program through extensive black-box testing of its operations under many conditions and inferring its logic by analyzing its output is legal conduct and does not infringe the copyright (unless there is a valid contractual agreement to the contrary).

The debate over the legality of reverse engineering of software concerns whether you can go inside the black box to study the code. More concretely, the legal question is whether you infringe a copyright when you copy a copyrighted program to study it (for example, making a core dump of the code) or when you run the code through a disassembler, decompiler, or other program that makes copies of the code so you can effectively recreate an equivalent of the source or assembly code.

(It would take another article to explore in detail yet another legal debate affecting the legality of going inside the black box to reverse-engineer software: whether shrink-wrap or other software licenses, in so far as they prohibit decompilation or other forms of reverse engineering, are valid and enforceable as a matter of contract law. However, the Fifth Circuit Court of Appeals recently struck down parts of a Louisiana statute that tried to validate software-licensing restrictions on backup copying, reverse engineering, and modifications. In the court’s view, the Louisiana law conflicted with federal copyright law, which gives purchasers the right to do such things with their software. In view of this ruling, it is doubtful whether shrink-wrap or other licenses for mass-marketed software that prohibit making copies of programs for reverse-engineering are enforceable contracts.)

Interpreting the law. The lawyers who argue that reverse-engineering copyrighted software is (and should be) illegal believe that such conduct violates not just one law, but two laws: It is both an infringement of copyright (because of the unauthorized copying of the program that must be done to analyze its contents) and a misappropriation of trade secrets (because copyright infringement is an improper way to obtain trade secrets). Because this view relies on a narrow reading of the copyright statute, I call it the strict-constructionist view.

Lawyers who believe that reverse engineering does not violate either law take a more pragmatic view, arguing that any copying of the code during reverse engineering is an incidental part of this process and not the sort of copying with which copyright law is really concerned. Because reverse engineering is generally considered a legal way to take trade secrets in other engineering disciplines, the pragmatists think it should be considered as an appropriate competitive conduct in software as well.

**Strict-constructionist theory**

The strict-constructionist theory, which holds that reverse-engineering copyrighted software — whether by core dump, disassembly, decompilation, or other such process — is always illegal, relies on a strict reading of the US copyright statute. This statute gives copyright owners the exclusive right to copy or authorize the copying of their works.

For software, there is a special provision giving owners of copies of copyrighted programs the right to make copies as backups and as an essential step in executing the program. The strict-constructionist theory holds that if you copy a program to reverse-engineer it, you have gone beyond the two copying privileges in this special statutory provision and thus have infringed the copyright.

**Fair use.** Some people argue that such copying is permitted by the copyright law’s fair-use doctrine. The fair-use doctrine lets people other than the copyright owner copy a protected work (or, more often, parts of it) for purposes and to an extent that won’t significantly interfere with the rights of the copyright owner. The copyright statute lists a set of factors to be considered in determining whether

January 1990
a use is fair or unfair, including
- the defendant's purpose in using the
  protected work,
- the nature of the copyrighted work,
- the amount and substantiality of what
  is taken, and
- the potential for harm to the market
  for the protected work.

For example, fair-use doctrine would let
you copy or quote from part of this article
(with credit to the source) but not copy
the entire article for distribution or sale.

But the strict constructionists argue
that copying software to reverse-engineer
it cannot be a fair use of the copyright
because people who do this generally
have a commercial motivation and are
seeking to misappropriate trade secrets.
Moreover, reverse engineers copy
the whole of the work, not just some
little piece of it. There is clear harm to the
software developer's market arising
from the reverse engineering. All these
factors weigh heavily against a fair-use
defense, in the eyes of strict construc-
tionists.

Trade secrets. If a court agreed with the
assumption that copying a program in the
course of reverse-engineering it is an im-
proper way to extract trade secrets, it
could find that the reverse engineer mis-
appropriated the original developer's
trade secrets. If the strict-constructionist
theory is correct, the only legal way to
learn what trade secrets copyrighted soft-

### Reverse-engineering case law

There have been several judicial decisions on whether making cop-
ies of programs for reverse engineering constitutes copyright infringe-
ment. The case law is conflicting, although the two cases in which
judges ruled that such copying was infringement involved other infringing
and otherwise inequitable conduct. The more recent cases have
found infringement only when the results of the reverse-engineering
process have been used to create a program with substantially similar
expression to the protected program that had been analyzed.

MAI v. Hubco. In one case, Hubco obtained copies of the different
versions of MAI's software (apparently in a legal manner) and made
printouts of the 1's and 0's representing its machine-readable form.
By comparing the printouts, Hubco was able to discern where the
software's governors were located in the cheaper versions and what
the pattern of 1's and 0's had to look like for the governors to be
removed. Hubco then began contacting MAI's customers and asked
those who had acquired the cheaper version whether they'd like to hire
Hubco to "fix" MAI's software so it would work as if they'd purchased
the more expensive version. Hubco eventually developed a program
to automate this enhancement of the MAI software.

MAI sued Hubco for infringement of its copyright of operating-syst-
em software and for misappropriation of the trade secrets in the soft-
ware. Although the judge ruled that Hubco had not stolen any trade
secrets from MAI because it had deciphered them through reverse
engineering, the judge issued a preliminary injunction against Hubco's
use of its program and its on-site visits to customers of MAI to remove
the governors.

The noncontroversial part of the judge's ruling in the Hubco case
was the finding of infringement based on Hubco's preparation of its
upgrade program, which included within it a copy of the MAI program.
The more controversial part of the Idaho federal court's 1983 ruling
was the injunction against on-site visits, which was based on a finding
of copyright infringement because of the printouts Hubco had made of
the upgraded versions of the MAI program. Hubco brought these print-
outs to MAI customers' places of business and used them as a basis
for making changes in the MAI software. (The judge did not rule that it
was infringement for Hubco to make changes to the software for MAI
customers, but only to copy the program to learn where the governors
were.)

SAS Institute v. S&H Computer Systems. Two years after Hubco,
a Tennessee federal court decided another case, SAS Institute v. S&H
Computer Systems, that involved the copying of a program for reverse-
engineering purposes. SAS successfully charged the developer of a
competitive statistical-analysis program with copyright infringement and
trade-secret misappropriation. S&H had acquired a copy of the source
and object code of the SAS software by a license agreement with the
intent of studying it so SAS could make a similar program to run on a
different kind of computer than that on which SAS ran.

Although the finding of copyright infringement rested heavily on
similarities between the two programs (some for source-code listings
and some for the code's structural aspects), the fact that S&H had
made unauthorized copies of both the source and object code so it
could study it was also ruled to be copyright infringement, as well as a
breach of a licensing agreement that S&H had signed.

Disassembly decisions. Later that same year, another judicial
decision was rendered in a software copyright case where competitive
software had been developed in part by disassembling the plaintiff's
programs. Notwithstanding the rulings in the Hubco and SAS cases,
the judges in the E.F. Johnson v. Uniden case (decided in Minnesota)
ruled that disassembly — said to be a common industry practice —
alone was not copyright infringement. Rather, copyright infringement
had to be based on similarities in the programs produced after disas-
sembly, the judge ruled.

The judge ruled that Uniden had infringed the copyright because the
two programs were more similar in details of their implementation than
was explainable by the need to be compatible with the radio-system
software produced by Johnson.

NEC v. Intel. The most recent copyright case involving reverse
engineering is NEC v. Intel, decided in a federal court in California
in February 1989. Although NEC personnel admittedly had disassem-
bled the Intel 8086/8088 microcode, Intel mainly relied on similarities
in the microcode that NEC developed for a competitive product (not
the disassembling) for its copyright-infringement claim. The clean-
room process NEC had set up to separate the analysis of the Intel
microcode from the development of NEC's functionally equivalent mi-
crocode was an important factor in persuading the judge that the simi-
larities in the two sets of microcode resulted from constraints imposed
by hardware, architecture, and specifications, and therefore did not
infringe the Intel copyright.
ware might contain would be by testing the software and treating it as if it were a black box. (The law generally lets people try to reverse-engineer a product sold in the market to discern its producer’s trade secrets (see the box on pp. 94-95), as long as no improper methods are used.)

The strict constructionists argue that there needs to be a strong rule against copying software for reverse engineering because the software industry is economically a very fragile industry whose costly and valuable innovations can too easily be pirated. Among the most valuable parts of software products are the algorithms, concepts, and techniques embedded in the software — parts not traditionally covered by copyright that thus must be protected as trade secrets.

The competitive edge these valuable secrets provide should, in the view of the strict constructionists, be maintained and respected by copyright law. Other firms are always free to write competitive software. There is then no need to reverse-engineer to develop a competitive product.

To the extent that a strict rule against reverse engineering might clamp a lid on competition in the modification and enhancement of software, the strict constructionists would say this is appropriate, since copyright gives software developers the exclusive right to control the making of derivative works. Unlike the semiconductor-chip industry, which actively sought a reverse-engineering privilege when a new law was created to protect chip designs, the software industry, they would argue, has not clamored for a reverse-engineering right, which is why there is no such provision in the copyright statute.

**Pragmatist theory**

It is a curious feature of programs, unlike other copyrighted works, that they do not directly reveal their contents to users. One reason the strict constructionist theory of illegality is so hard for traditional copyright lawyers to accept is that it would mean that the weight of the copyright law could be brought to bear against someone who purchased a copy of a copyrighted work and tried to figure out what it "says."

In another article, I have argued at length that — consistent with the historical purposes of copyright and patent law — disclosure of a program’s contents ought to be required to obtain copyright protection. Although copyright law does not now require disclosure of program contents, it would be an aberration from copyright tradition to say that someone can be penalized by copyright law for seeking to obtain disclosure on one’s own by reverse engineering.

**Fair use.** Lawyers who do not adhere to the strict-constructionist theory of copying for reverse engineering take a very different view of the fair-use privilege. For one thing, they say that it is in the nature of machine-readable programs to be copied to be read and studied. For another, they do not regard copying a program to reverse-engineer it as directly affecting the copyright owner’s economic interest any differently than does copying it for backup. Such copying does not, for example, displace a sale that might otherwise come to the copyright owner — the reverse engineer, like the person making a backup, has already bought the program.

The pragmatists regard the strict constructionists’ argument as similar to one unsuccessfully raised by the two movie studios who sued Sony for contributing to copyright infringement because Sony knew that its customers could use Betamax videotape machines to make unauthorized copies of the studios’ movies. The studios argued these copies were infringing copies, even if the copies were made only so viewers could watch the movies at more convenient times than they were broadcast (called “time-shifting”).

In the Sony case, the US Supreme Court ruled that it was a fair use for owners of videotape recorders to use their machines to copy the whole of a copyrighted movie shown on television for time-shifting. It also ruled that, because Sony’s machines were capable of noninfringing uses, Sony could not be held liable for contributory copyright infringement — even though Sony knew that some of its customers might make infringing copies of protected works. The Sony decision forecloses the theory that the mere copying of the program is enough to negate a fair-use defense.

Another factor affecting the fair-use theorists’ judgment about reverse-engineering programs is that it often requires exceedingly tedious, time-consuming, and concentration-intensive activity. Reverse-engineering machine-readable code is not unlike trying to break the “code” of Egyptian hieroglyphics or Sumerian cuneiform. Indeed, it can be more difficult than either of these, for there are only two elements — 0’s and 1’s — to be read, and the same sequence appearing at one place in the code may mean something entirely different than the same sequence later on.

Moreover, many software developers encrypt parts of their code to make it more difficult to read. Reading code — assuming it can be done at all — would seem to be conduct that traditional copyright law would not envision as an infringing activity. Even with the aid of reengineering technology that may considerably ease the task of deciphering the code, reverse engineering is a far cry from the straight piracy that virtually everyone in the industry would agree is illegal — and rightly so.

**Trade secrets.** It is not so much the copying of the code that is of concern to the software developer who wants to make reverse engineering illegal but more the reading of the code that, once deciphered, will let the reader discern the trade secrets that might be embedded in...
Different laws and reverse engineering

Before the legal debate over the legality of software reverse engineering, the legality of reverse engineering had come up in several other contexts.

**Trade-secret law.** Trade-secret law was the first intellectual-property law to develop a rule about the legality of reverse engineering. Owners of trade secrets have the right to be protected against misappropriation of their trade secrets. The two most commonly recognized ways to misappropriate such secrets are either by breach of a confidential relationship that arose when the owners disclosed the secret in confidence to another or by use of improper means (like industrial espionage or bribery) to obtain the trade secret.

Trade-secret law has traditionally not protected any property interest as such in a trade secret, just the right to be free from unwarranted interference with confidential relationships and use of improper means to get the secret. Because of this, someone who can learn another's trade secret by reverse engineering — working backward from the final product to analyze how or of what the product was made — acquires the trade secret legally.

Competitors, then, have the "right" to use reverse engineering to obtain trade secrets. It is in the public interest for competitors to have such a right — even if it means that the successful reverse engineer may be able to take a free ride on the research and development efforts of the innovator firm. Reverse engineering may well lead to further innovation by the firm that does the reverse engineering or to expanded production of and lower prices for the desired product.

Reverse engineering is considered to be a good thing under trade-secret law, even if it has the potential to harm some economic interests of the trade-secret owner.

**Patent law.** Reverse engineering has generally not arisen as a problem area in patent law for the simple reason that patent law has always required that an inventor reveal a considerable amount of detail about the invention to get a patent. Patent applications must contain a written description of the invention, and of the manner and process of making and using it, in such full, clear, concise, and exact terms as to enable any person skilled in the art to which it pertains, or with which it is most nearly connected, to make and use the same, "according to the patent law (emphasis added).

Someone obtaining a patent must also "set forth the best mode contemplated by the inventor of carrying out the invention," the law says (emphasis added). (Failure to abide by these disclosure requirements can be grounds for not issuing the patent, for invalidating any patent mistakenly issued, and for denying relief in a later trade-secret lawsuit about things that should have been disclosed in the patent application.)

When a patent is issued, the disclosed information is printed as part of the patent. During the lifetime of the patent, anyone is free to read and learn from the patent; what is forbidden is only making, using, or selling the invention during the 17 years the patent lasts.

If it does become necessary to try to make another person's invention to evaluate how it works or to understand the scientific principles underlying it, patent law leaves some room for a person to make the invention for "experimental purposes" without infringing the patent. It is only making or using the invention for its intended end use that infringes the patent.

The experimental-use doctrine is, then, a kind of doctrinal cousin to trade secret's reverse-engineering rule in that it too serves as a public-policy limitation on the scope of rights of the holder of an intellectual-property interest.

**American copyright law.** Until software came along, reverse engineering had never arisen as a problem area for copyright law. There are several reasons for this.

First, until the admission of machine-readable software to the copyright realm, all copyrighted works revealed their contents, either directly (like a book) or indirectly (like sound recordings when played). You never had to reverse-engineer a copyrighted work to find out what it "said."

Although copyright law never formally required disclosure as patent law has done, this may be because — until software — there was never a need to do so, since disclosure traditionally happened as a matter of course when the work was published. It is unquestionably a fundamental purpose of US copyright law to promote dissemination of knowledge.

Second, "reverse engineering" is a term that describes conduct about utilitarian things. (It makes sense to talk about reverse-engineering a carburetor, but not a poem.) Until software was made copyrightable, copyrights did not protect utilitarian works. Works that have a utility beyond merely conveying information or displaying an appearance have been considered utilitarian and, because of this, have traditionally been excluded from copyright. Yet it is only utilitarian works that you would try to reverse-engineer.
Third, until software, copyright law gave only the most limited protection to engineering designs. Reverse engineering, as the term itself implies, is an engineering enterprise, a search for the underlying technical ideas, techniques, or designs of a work, perhaps to make a competitive product.

In the US, engineering drawings, because they are drawings, are protectible under the copyright statute. But the copyright on an engineering drawing does not give rise to an exclusive right to make the engineered product depicted in the drawing nor to an exclusive right to make other drawings of the engineered product. No matter how much effort went into developing the engineering design itself, and no matter how detailed its content, that design was not protected by the copyright. Indeed, traditional copyright law considered the engineering design to be in the public domain. To protect the design, you must patent it.

**British copyright law.** Until 1988, the rule about copyright protection for engineering designs was different in the UK than in the US.

This difference is illustrated by a case decided in 1986 involving a dispute between British Leyland and Armstrong, a firm that manufactured an exhaust system that could be installed as a replacement part in British Leyland cars. British Leyland copied a set of engineering drawings of, among other things, the exhaust system of its automobiles. These drawings depicted the configuration of the exhaust system and its various intersection coordinates. Armstrong did not have or have access to the British Leyland drawings, but it did have an opportunity to inspect a British Leyland exhaust system.

From its examination of this system, Armstrong made its own drawings of the British Leyland exhaust system and subsequently made and sold exhaust systems of the same basic design as the British Leyland system — without, of course, paying British Leyland any royalty. British Leyland sued Armstrong for copyright infringement of the engineering drawings — and won, despite the fact that Armstrong had reverse-engineered the system, not copied any of British Leyland's drawings.

This result is the reverse of what a US court would reach if presented with the same or a similar situation. US copyright has always assumed that unless a utilitarian work, like an exhaust system, was patented, it was in the best interest of the public in a competitive economy for such works to be freely copied by competitors.

The UK has just recently adopted a new intellectual-property law that changed copyright law to overturn the result in the British Leyland case and to adopt a copyright rule similar to the US rule for engineering drawings.

**US semiconductor-chip law.** Before the passage in 1984 of the Semiconductor Chip Protection Act, some chip manufacturers had copyrighted technical drawings of the layouts of their chip products and then argued that the mask works and finished chip products were covered by the copyright in the technical drawings or were separately copyrightable derivative works.

The US Copyright Office, although it accepted registration of the technical drawings, denied registration to the mask works and finished products on the ground that these were utilitarian works not qualified for copyright protection.

Unsatisfied with this, some chip firms went to Congress to get protection for their chip designs. Although some thought was given to amending the copyright statute to protect chip designs, Congress eventually decided to create a new intellectual-property law to protect chip designs. (Some have argued that software too needs a special-purpose statute of this sort because its utilitarian character makes copyright such an awkward law to apply to programs.)

Industry proponents of chip-protection legislation insisted that it was only the exact copyists of their layouts against whom they sought protection, not those who might copy the chip design to reverse-engineer it. Reverse engineering was said to be a common practice in the chip industry and was believed by those who testified about it to be a positive form of competition. There was considerable discussion in the chip community about whether copying for reverse engineering would be considered a fair use if the chip-protection provisions were put in the copyright system or whether a special provision authorizing reverse engineering ought to be made a part of the statute. The industry strongly desired a reverse-engineering privilege.

Responding to this concern, Congress eventually adopted a special reverse-engineering privilege in the chip law. This provision lets competitors not only copy a protected mask work to study the concepts or techniques embodied in the mask work or in the chip's circuitry, logic flow, or organization but also to incorporate results of this study and analysis in a competitive chip product as long as the latter is itself an original work and not substantially identical to the protected mask work that had been studied.

Making compatible products. Another reason to reverse-engineer software is to discover how the program’s interfaces are constructed so you can develop programs that will be compatible with that program. Although whether copyrights can be infringed based on interface similarities needed to achieve compatibility is itself a hotly contested issue in copyright law, the better view seems to be that similarities of this sort are not infringing.

As an aid to proving that you took only ideas, functional specifications, or the like from the program under study, it may help to have a clean-room procedure, where one team works on reverse-engineering the code and a second team works from the functional specifications derived from the code and reimplements them.

But a clean-room procedure is not necessary to prove that you didn’t infringe the copyright. What is necessary is to show that you took no protected expression.

Certainly, taking the literal expression used by the original programmer, or making only minor variations (such as in the naming of variables) would be an infringement, even though much work might have gone into this effort. Sometimes even plagiarism is hard work, but that doesn’t mean it isn’t plagiarism!
such a situation, it is not the reverse engineer-  
ing per se that would support the copyright-infringement charge but the making of an infringing work.  

Describing the software's workings.  
Apart from modifying and developing a new program, yet another category of things you might do with the results of reverse engineering would be to disclose the results to other people, for example, by publishing an article about them. This would destroy the trade secret, unquestionably injuring the economic interest of the trade-secret owner. Yet it is doubtful that copyright law—which is aimed at promoting the dissemination of knowledge—should be used to vindicate this kind of trade-secret interest.

The policy reasons that support permitting reverse engineering of all other kinds of trade secrets ought to come into play when judging the reverse engineering of software trade secrets. Software developers have not yet made the case that they should be given special privileges not granted to other trade-secret owners.

Are strict rules enforceable? But perhaps the best reason not to have a strict rule against reverse engineering is because such a rule would likely be unenforceable. Software reverse engineering by competitors, would-be competitors, customers, and computer scientists is widespread and considered by many to be perfectly legal behavior.

Many would say that the reverse engineering of software has contributed to more rapid growth in the software industry and to the increased competition in product development and refinement than would have occurred in an environment in which software reverse engineering was illegal.

To persuade those who hold such views that a strict rule against any reverse engineering must now be respected would probably be impossible. To even try to enforce it might backfire, for those who stop respecting copyright principles based on what is widely perceived to be an overly strict rule may tend to stop respecting other rules of copyright as well.

One important lesson of the Sony Betamax case is that copyright owners are not entitled to control every economic benefit that the public might derive from their works. Another important lesson of the Sony case is that copyright law should take into account the interests of the public in the use and availability of new technologies. A reasonable balance between the public’s right and the innovator’s right must be maintained. Software owners should not be able to get special privileges under the copyright law simply because it would strengthen their market position.

The strict constructionist view of copyright law that holds that reverse engineering is always illegal confuses means with ends. Reverse engineering is a set of methods and tools that may enable a software designer to create new and better programs. But whether copyrights are infringed by reverse engineering must depend on the programs that ultimately result, not on the incidental copying that is required to use the reverse-engineering methods and tools.

Acknowledgments
I thank Elliot Chukowski, the anonymous reviewers of this article, and my husband, Robert J. Gushko, for their helpful suggestions about how the article might be improved. I also thank the faculty and students of the Computer Science Dept. at Carnegie Mellon University for their enlightening comments on this subject when I presented an earlier version of this article at a programming systems seminar series a few years ago.

References

Pamela Samuelson is a law professor at the University of Pittsburgh. During the 1989-90 academic year, she is a visiting professor of law at Emory University. During 1985 and 1986, she was the principal investigator of the Software Engineering Institute’s Software Licensing Project, which studied and recommended changes to the US Defense Dept.'s software-licensing policy. She worked at the law firm Willkie, Farr & Gallagher before becoming a law professor. Samuelson graduated from Yale Law School. She has written extensively on software-related intellectual-property issues.

Address questions to Samuelson at Law School, Gambrell Hall, Emory University, Atlanta, GA 30322.

96 IEEE Software